O'REILLY®

97 Things Every Application Security Professional Should Know

Collective Wisdom from the Experts

Edited by Reet Kaur & Yabing Wang

97 Things Every Application Security Professional Should Know

Collective Wisdom from the Experts

Edited by Reet Kaur and Yabing Wang

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

97 Things Every Application Security Professional Should Know
Edited by Reet Kaur and Yabing Wang

Copyright © 2024 O'Reilly Media. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: Simina Calin
Development Editor: Rita Fernando
Production Editor: Christopher Faucher
Copyeditor: nSight, Inc.
Proofreader: Helena Stirling
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Kate Dullea

June 2024: First Edition

Revision History for the First Edition
2024-06-18: First Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781098169459* for release details.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *97 Things Every Application Security Professional Should Know*, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

The views expressed in this work are those of the authors and do not represent the publisher's views. While the publisher and the authors have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the authors disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

978-1-098-16945-9

Table of Contents

Part IV. Code Scanning & Testing

Part VI. Software Supply Chain

Part VII. Threat Modeling

Part XI. AI Security & Automation

Part XII. IoT & Embedded System Security

Preface

Cybersecurity, or information security, has always been a very broad and comprehensive field and has been a fast-evolving area for the past 10–20 years. Within, there are many domains, such as risk management, security operations, network and infrastructure security, identity access management, and others. This book focuses on one particular domain called application security (AppSec). That's because, in today's modern world, software development has become the core of any product or service. As such, ensuring the security of any product or application development is critical to the success of your business.

This book is a collection of wisdom from 77 security experts in application security across various industries. Organized into 12 topics, the book covers web applications, mobile applications, APIs, and the Internet of Things (IoT) (embedded systems). It also expands the safeguards to both on-prem and in-cloud development. More importantly, it explains all angles of AppSec such as secure software development life cycle (SDLC) practice, threat modeling, code scanning and testing, vulnerability management, and how to run a successful application security program. The book also provides insight into two emerging topics: software supply chain security and AI security. It is a treasure trove of those security practitioners' practical advice, distilled into bite-sized essays for both beginners and seasoned professionals in application security and cybersecurity.

You should read this book if you are:

- New to security and want to learn more about application security
- A developer and want to learn how to secure your application
- Interested in running a successful application security program

We hope you find this book valuable to meet your needs, and that you can take the lessons learned from other practitioners and apply them in your world to make your applications resilient against evolving threats. Get ready to absorb expertise from some of the best in the field—your go-to guide for application security success!

O'Reilly Online Learning

For more than 40 years, *O'Reilly Media* has provided technology and business training, knowledge, and insight to help companies succeed.

Our unique network of experts and innovators share their knowledge and expertise through books, articles, and our online learning platform. O'Reilly's online learning platform gives you on-demand access to live training courses, in-depth learning paths, interactive coding environments, and a vast collection of text and video from O'Reilly and 200+ other publishers. For more information, visit *https://oreilly.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-889-8969 (in the United States or Canada)
707-827-7019 (international or local)
707-829-0104 (fax)
support@oreilly.com
https://www.oreilly.com/about/contact.html

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *https://oreil.ly/97-things-app-security*.

For news and information about our books and courses, visit *https://oreilly.com*.

Find us on LinkedIn and watch us on YouTube.

Acknowledgments

We would like to thank all contributors for volunteering their time and sharing their wisdom and insights with the community. The security community will never be stronger if we don't have people devoting their time to helping each other!

We also would like to thank O'Reilly editors Rita Fernando and Simina Calin. They collaborated with us and with all contributors through the whole process, and finally helped make the book available to all O'Reilly learning platform users. Your dedication is highly appreciated by both of us, and we're so happy to have you along with us!

Finally, we thank our families for providing strong support to us while we worked on this book. Without them, we would not have been able to complete this book on time!

PART I

Program & Practice

Secure Code for Tomorrow's Technology

Alyssa Columbus

The software we build today has the potential to power critical systems for years to come. To ensure the longevity and security of tomorrow's technology, developers must adopt a security-focused mindset and consider security as part of the quality of the code. Writing secure code requires meticulous attention to detail. To assist you in this endeavor, I have compiled the following checklist of fundamental principles and important items to keep in mind:

Start by learning secure coding standards.
: Many common vulnerabilities result from a lack of awareness. Familiarize yourself with leading authorities such as the Open Web Application Security Project (OWASP) Top 10 and CWE/SANS Top 25. Study guidelines for your specific languages and frameworks. Understand basic security principles like least privilege, defense in depth, and secure by default. Reference these standards early when designing applications to build in security from the start.

Adopt a secure development life cycle.
: Integrate security practices into all phases of development, from conception to deployment. Conduct threat modeling to identify risks. Define abuse cases. Perform static analysis security testing to catch issues in code. Run dynamic analysis to test for vulnerabilities in running applications. Automate processes such as policy compliance checks, dependency upgrades, and credential rotation to reduce mistakes.

Use frameworks carefully.
: While frameworks (e.g., Django for Python, Express for Node.js, Laravel for PHP) boost productivity, they also introduce risks. Keep frameworks updated to avoid known vulnerabilities. Properly configure frameworks and disable unnecessary features. Extend frameworks securely rather than overwriting insecure defaults. Continuously monitor for

vulnerabilities in all dependencies. Lock down functionality to what your app actually requires.

Never trust inputs.
: Validate and sanitize all data entering your application, including from users, files, databases, third-party APIs, and internal services. Practice *zero trust* by assuming all inputs are malicious until validated otherwise. Limit exposure through compartmentalization and minimization. Escape outputs properly to prevent injection attacks. Sign and encrypt sensitive data end to end and at rest.

Promote a culture of security.
: Advocate for secure coding as a team effort, not just the developer's job. Instill a sense of shared accountability through training, mentoring, incentives, and leading by example. Make security reviews a regular part of the development process. Empower all team members to call out potential issues. Automate policy enforcement where possible.

Foster developer empathy.
: Understand that developers are often undertrained in security and overburdened with competing priorities under tight deadlines. Help them succeed by providing useful security libraries, user-friendly tools, and clear guidance baked into the software development life cycle. Reward secure coding efforts.

Prioritize appropriately.
: Focus first on security issues that pose the most significant risks based on your threat model and business context. Eliminate entire classes of vulnerabilities where possible. Build basic security capabilities before adding advanced features.

Take a long view.
: Consider how software architecture and design decisions today could impact security far into the future. Seek designs that are adaptable, resilient, and sustainable as both technology and threats evolve.

Stay current.
: Monitor emerging threats, revisit past assumptions, and keep your skills sharp through ongoing education. Contribute to open source security projects. Attend events and training. Learn from peers.

Teach others what you know.

Share your security knowledge with teammates through mentoring, code reviews, and organizational training. Write for publications and speak at events. Learning and progress accelerate when we all help each other.

Writing secure code requires knowledge, skill, and vigilance. By embracing these disciplines now, you can safeguard against tomorrow's breaches. While challenges may persist, you have the power to create resilient and trustworthy applications by keeping security at the forefront of your mind throughout development. The effort is not only worthwhile but essential in protecting our future.

Pragmatic Advice for Building an Application Security Program

Andres Andreu

Application security (AppSec) is a never-ending journey. The goal is to positively influence the relevant software engineering culture and stakeholders to willingly participate. A *shift left* approach, for instance, should become a mutually desired business enabler.

The first step of a successful AppSec program is to set up common goals and gain alignment from the engineering side. This requires tactful education, communication, and storytelling from cybersecurity leadership to engineering leaders. Therefore, getting their buy-in on how the security team is trying to achieve the common goals while enabling them to produce software becomes the key.

Ideally, application security is just silently there. This is ultimately the goal of security as a business enabler. The challenge is that security hurts and it costs (time, money, effort, etc.). As such, many organizations see security as a necessary evil. Weaving a security-first mindset into an organization's culture is foundational. It will take time, but it is the secret sauce of success.

A key area for positive impact will be the *software development life cycle* (SDLC). This needs to be transformed into a *secure* SDLC (SSDLC). Depending on resources, a great approach is to embed AppSec talent or champions into the engineering teams/squads. This will create institutional knowledge and domain expertise. Adjusting your program to be aligned with your engineering team's SDLC becomes another key. Don't apply a waterfall approach when your development team has an agile culture.

Maturity matters, and there should be formal tracking of progress. One framework (in regard to establishing a baseline and measuring this over time) is *OpenSAMM* (open framework Software Assurance Maturity Model), an open framework to help build security into the software development

process. Maturity scores may drop every now and then, as this space is sensitive to certain events. Take a Mergers and Acquisitions (M&A) event for example—you have little control over what you inherit.

Focus on solving problems and building solutions, not implementing products. The goal is to positively impact an entire ecosystem from the left and right. On the left, there is the SSDLC. On the right, there are architectural components and other initiatives that range from systems thinking to active protection.

The scope of your AppSec program will be critical. Scope is very subjective per organization. In order to set your team up for success, set the boundaries early. For example, is your program going to cover database security?

Make sure your AppSec team gains intimacy with all relevant software engineering processes. This will facilitate the implementation of an SSDLC. Intimacy also has a direct impact on relationships. Building relationships is critical. Bidirectional communication will prove invaluable. Getting into the weeds with software engineering teams, quality assurance teams may quickly identify who will be a champion for your AppSec program.

Take every opportunity to show how some security initiatives can be seamless. For example, if you have an engineering team that creates compiled code, why not build a library (shared or static) that performs security-related functions (i.e., input validation, header setting, encoding/decoding, etc.)?

You will need to relay the effectiveness of your AppSec program to corporate executives. Focus on metrics that matter, as not all will be relevant.

Security leaders set strategies and create programs, but more importantly, advise the organizations on risks and risk mitigation. A solid AppSec program is one of those advisory areas. Factor in the people, processes, technology, and culture of the organization. Make this a continuous process as things change; your program must adapt and overcome accordingly.

AppSec Must Lead

Brook S.E. Schoenfield

Someone must take responsibility and be accountable for security, especially for AppSec. Foremost, most people who have some role related to the production and operation of software often have limited or even no AppSec knowledge. Couple that reality with the complexity of our state-of-the-art AppSec requirements and practices, and what we have is a vacuum that cannot be filled simply by telling engineering, product management, project managers, and developers to "make code secure."

Someone has to lead.

Leadership is earned, never merely given. While many roles will specify a "leadership" component or requirement, true leadership is recognized by what each of us does. Leaders are those people who take responsibility and put themselves forward as accountable for the impacts and consequences, not just of their own actions but the results of collective effort. I don't mean just taking credit for successes; in fact, great leaders are happy to assign credit to everyone who contributes.

Leadership may not be about holding decision-making power. In fact, a task typically enacted by a security team is identifying and rating risks. But the risk decisions often must not be taken by those same people; it is their very independence from accountability for risk decisions that allows security folk the freedom to accurately build a risk picture.

How do we manifest leadership? By never sweeping security problems under the rug. And by fostering a drive to find security problems and then seek solutions, even in the face of resistance. We show leadership by helping others struggle with challenges and by sharing what we know so that security skills radiate throughout a development organization. We lead by building a "culture of security" so that security becomes part of how software is made and run.

How does a leader facilitate a culture of security? Executive mandates do serve a purpose. But mandates don't change the culture. A leader creates the conditions that foster the culture that they envision.

The most important change agent will be what the leader does. This is called *pro-social modeling*. We enact the behavior we wish to encourage. We model how we want things to be. A security leader can't expect others to take security seriously unless the leader unceasingly drives toward improving security practices, knowledge, and standards. Leaders demonstrate their care through what they do and what they prioritize.

If we want threat modeling to be performed from idea through design (at least), a leader proactively identifies new ideas and changes and then asks to be included. Once participating, the leader demonstrates the value of early threat modeling by identifying security requirements and then helping those to be designed into the software. Demonstrate value. Others will notice and want some of that value, too.

Leaders can also create forms where others are included, where diverse input is obviously valued, and where both successes and failures can be openly discussed. In other words, a culture (security and others) lives and breathes through human interaction that the participants find useful and worth their time. For a security culture, this may mean creating a "community of practice" where any and all interested parties are welcome to talk security "shop."

Something is obviously happening; the advancements taking place are very attractive. Many workers love to "hitch their wagon"—in this case, their career growth—to efforts that demonstrate success and improvement. As leaders, it's our job not only to make things happen, but also to welcome whatever help each person can offer. Eventually, there will be so many involved that the tipping point will be reached, whereby security is "just how we make software." In other words, security will be part of organizational culture.

Without leadership, AppSec will fail. With leadership, we can achieve our common objectives together.

Solving Problems for Application Security

Caroline Wong

Fundamentally, application security is about designing, building, and maintaining secure software. Good software helps organizations, and bad software hurts organizations.

There are four main categories of application security activities: governance, finding security problems, fixing security problems, and preventing security problems. This essay will provide a high-level description of each of these four categories, with an emphasis on fixing security problems:

Governance
: There are several high-level factors to consider when developing an application security program. These include compliance and regulatory requirements, contractual relationships with other organizations, and a solid understanding of what you're supposed to be securing in the first place. It's also important to define metrics up front so that the success of the program can be measured and demonstrated over time.

Finding security problems
: There are many ways to find security problems at different points in any software development life cycle, whether an organization follows a waterfall, Agile, or DevOps methodology. Security testing types include threat modeling, code review, and penetration testing. A combination of manual and automated security testing is likely to result in the most efficient and effective identification of true positive security vulnerabilities in software applications.

Fixing security problems
: Fixing security issues is not just a technical problem; people and processes are also required to get it done. Once security testing has been performed in order to find as many true positive issues as possible, the next step is to engage with the teams that can actually fix the issues. The

quality of software does not improve until the problems are addressed or eliminated. Fixing security issues requires effective communication, coordination, and integration with development teams and processes.

Security teams must recognize that developers are focused on building new features and meeting deadlines and have limited bandwidth to remediate security issues. It is certainly not possible to fix all the security issues at once. They must be prioritized in the context of business values and goals and addressed over time.

I recommend that security teams get curious about development team priorities and look for areas of common interest. They should ask questions about how development teams work and how much time they have to realistically spend on fixing security issues.

One of the best ways to get security bugs fixed is to integrate with developer tools and processes. Security teams should ask about the tools developer teams use to do their work and the processes they follow to manage it. For example, how frequently do they release code? This should influence the frequency of security testing. What bug tracking system(s) are they using to manage bug fixes? Make sure security bugs are included and don't get lost in separate systems or PDF reports.

Preventing security problems

The people who build software must understand why vulnerable code is insecure. Developers must be empowered with tech stack-specific knowledge and tools to help them avoid creating security bugs and flaws in the first place. Ideally, good programming practices and well-designed frameworks make it easier for developers to write secure software by default and harder for them to make mistakes. Cloud environments must be configured correctly to prevent security vulnerabilities from being exploited, and attacks must be discovered and stopped as early as possible to minimize damage.

The ways in which development and operations teams interact are changing, and security must keep pace. Security teams working effectively with DevOps teams, processes, and tools are absolutely critical to getting application security done right. Security teams sometimes place heavy emphasis on finding issues, without enough focus on engaging with the development teams and building the cross-functional relationships that are actually required to get security issues fixed.

Securing Your Enterprise Applications

Chadi Saliby

Enterprise applications are used to streamline business processes, manage data, and enhance collaboration. However, with the growing reliance on digital systems, the security of enterprise applications has become a paramount concern for businesses and governments.

Security for enterprise applications implies that there are measures and practices in place to safeguard these applications from various threats and vulnerabilities. The primary goal is to protect sensitive data, maintain business continuity, and ensure that the applications are available, reliable, and resilient against the inevitable attacks by cybercriminals.

Security is important for enterprise applications for the following reasons:

Data protection
: Enterprise applications will handle at some stage a vast amount of sensitive and confidential information, such as personally identifiable information (PII), protected health information (PHI), and/or intellectual property. A security breach can result in severe financial and reputational damage to any organization.

Compliance and regulations
: Many industries are subject to strict data protection regulations and laws (e.g., GDPR, HIPAA, PCI DSS). Adhering to these regulations is not only a legal requirement, but also demonstrates the organization's commitment to data privacy and security.

Business continuity and recovery
: Cybersecurity incidents, such as data breaches or ransomware, can disrupt business operations, leading to downtime and significant financial losses. Strong security measures help ensure business continuity and minimize the impact of such incidents.

The following are practical steps to protecting your enterprise application:

- Incorporate security at the early stages of your software development life cycle (SDLC) by conducting regular code reviews, vulnerability assessments, and penetration testing. Applications that do not properly validate user input are vulnerable to injection attacks, such as SQL injection and *cross-site scripting* (XSS). These attacks can lead to unauthorized data access or manipulation. Therefore, adopting static or dynamic code analysis capability in the SDLC is a fundamental step to protect your applications.
- Threat modeling is another control to add to the secure SDLC to verify that you haven't missed any gaps in any stage of your SDLC.
- Develop a thorough incident response plan to quickly detect, respond, and recover from security incidents. It's a fact that we cannot prevent attacks all the time, so preparing a detailed incident plan is a requirement.
- Applications are the gateways to access our data. Ensure that only authorized users have access to the specific parts of the application required for their role. This requires robust authentication, authorization, and access control mechanisms to be in place.
- Implement *multifactor authentication* (MFA) with *role-based access control* (RBAC) to ensure only authorized users can access specific functionalities and data; here you can utilize data masking.
- Encrypt sensitive data in transit and at rest to prevent unauthorized access. Secure storage and data masking techniques are often employed to mitigate these risks. As this will be a second layer of defense in the event the data is successfully exfiltrated, it will be useless to whoever stole it.
- Implement firewalls, intrusion detection/prevention systems (IDS/IPS), and secure network configurations to protect applications from network-based attacks.
- Implement a *zero trust architecture* (ZTA) principle, where you verify explicitly, use least privilege access, and always assume breach; conducting regular security training is essential to mitigate those risks.
- Regularly conduct security awareness training and tabletop exercises for employees to educate them about social engineering methods, so they won't fall victim to such attacks.

Developers as Partners in Application Security Strategy

Christian Ghigliotty

With the emergence of public cloud providers, developers now have more responsibilities than just writing software to power the business. In many cases, they now have direct responsibilities over security and infrastructure. The additional responsibilities create new challenges for developers, as they are likely operating in spaces they don't possess deep domain knowledge. These challenges create an opportunity for security teams to foster relationships and feedback loops with development teams to inform a successful long-term application security strategy.

A guiding principle in customer-focused cultures is *meeting your customers where they are*. If developers are our primary customers, understanding how they build and ship products is crucial to creating experiences that incentivize collaboration and produce positive security outcomes. The *continuous integration and continuous delivery/deployment* (CI/CD) pipeline—the automated workflow that encourages repeatability, iteration, and code quality—is one of our primary areas of opportunity. Adding select tools to scan code for insecure code patterns, secrets, and vulnerable dependencies creates a set of signals for your program. A well-informed developer's experience with those findings leads to nuanced discussions that form the basis for a more mature risk-based remediation strategy. That experience *should not* include longer build times or false positives that could erode trust.

Automated tooling also generates data, which can be queried for analysis to further drive investment areas. If secrets are frequently committed to code, should we add precommit hooks for quicker detection time or build wrappers around secrets management APIs to make management easier? If cross-site scripting is a common static code analysis finding, can the security team identify frameworks to abstract that responsibility away? Further

opportunities to create touchpoints also exist in communication platforms such as Slack and in local development toolkits, but the CI/CD pipeline is the most critical.

In the same way that the cloud expands the bounds of velocity and responsibility, considering threats and encouraging systems thinking broadens developer mental models to understand risks beyond code. This is typically described as *threat modeling*, the structured exercise of assessing systems to understand threats and mitigations. Through discussion, patterns (and potential antipatterns) emerge. If systems are using varying *Transport Layer Security* (TLS) cipher suites, can an agreed-upon set of algorithms become "secure defaults" embedded within infrastructure? Are teams using well-worn standards like Security Assertion Markup Language (SAML) and OpenID Connect (OIDC) for authentication? Threat modeling also cultivates a culture of discourse and curiosity that can scale with an organization.

In these discussions, you'll likely begin to identify people who are passionate about security and are interested in expanding personal knowledge. These candidates could be a great fit for yet another strategic initiative that deputizes security advisory to close collaborators—also known as a *Security Champions program*. With support from the security team, a strong Security Champions program takes the observed behaviors and lessons learned and infuses them into interactions across the company.

An effective application security strategy must be driven by meaningful partnerships where two-way communication is productive and continuous. This approach ensures that decisions aren't made in silos but in context, with customers always being part of the program. A key success factor is for security teams to work closely with the development team, not only to demonstrate "what's in it for them," but also to find all opportunities to enable them to do their work more effectively.

Be an Awesome Sidekick

Daniel Ting

Imagine you have a horrible flu and visit the doctor for immediate relief. But instead of providing medicine, the doctor starts discussing weight loss, which might help prevent future health issues. However, it doesn't address your current flu symptoms.

Or, what if the doctor tells you to wear a hazmat suit so you won't get the flu again? It would improve your safety and security from the flu. Yet would you do it? Probably not. Why? It's impractical.

How would you feel about that experience? I'd feel upset that my priorities were ignored and that the prevention advice was impractical. I'd never return to that doctor. I'd look for a doctor who is empathetic and understands my priorities.

Similarly, dev and business teams need to be supported with empathy and an understanding of priorities. Our ability to do this as AppSec professionals makes us indispensable sidekicks. The dev and business teams as a whole are the heroes here; we play the essential role of supporting them in making informed decisions and minimizing harm. To be an awesome sidekick, there are three things we need to remember for success. Let's unpack them.

It's About Them, Not You.

AppSec is a support role. It does not exist without an application to secure. Although risky and concerning, developers can still build applications without security. Understanding this puts into focus that, as AppSec professionals, we're here to support the developers and the business in building a trustworthy application that behaves as intended.

That means we must first seek to understand their goals and priorities. Focus on the help they want, and then opportunistically find ways to deliver on what they need. Our role is often to advise, influence, and support. Focus on the wrong priorities, and you would likely be excluded from future

conversations—like what happened with the doctor. Like a good sidekick, it's about the hero, not you.

Balanced Priorities (and Constraints)

As supporters, it is important that we understand the priorities of the application team and the business to help them accomplish their goals. This often means putting yourself into your teammates' shoes, untangling competing priorities and constraints, and helping the whole team make informed decisions. Understanding these priorities helps us be sensible in the support we provide as AppSec professionals. It is useful to understand how teams prioritize work and how constraints are considered. Understanding methods like the *reach, impact, confidence, and effort* (RICE) technique; the *must have, should have, could have, won't have* (MoSCoW) technique; and backlog grooming activities, we can consider AppSec within the team's workflow and priorities.

Like in the doctor example (although theoretically, avoiding the risk of death from severe flu is important to any rational person), the affordability, availability, and convenience of not being in a hazmat suit took greater priority. However, a different outcome may be likely if we're in a biohazard site.

It's almost always a "good enough" decision, as there's always more to do. Being safer from the flu isn't great if you die from starvation; likewise with AppSec. We need to continually improve AppSec smartly—through evolution, not revolutions.

Easier Is Easier

When presented with two options, we have a bias for choosing the easier of the two. Despite the risks, we still jaywalk if it means walking a shorter distance. Leverage this behavior and make the secure workflow way easier to adopt than the alternative. Use a modern development framework, such as ReactJS or Ruby on Rails, over coding boilerplate from scratch, as it "comes with batteries" and security is built-in. It's easier, faster, and reduces bugs, including security ones.

Conversely, it imposes meaningful costs on unwanted behaviors, like making the person who introduced the bug or security weakness the person to fix it. This creates an accountability feedback loop that helps encourage teams to do the right thing easier. Like most good sidekicks, they make the best option the easiest choice.

These lessons helped me empathize and collaborate more effectively with my development teams. I hope these will help you build amazing things, safely, with security built-in.

Understanding the True Boundaries of Modern Applications

Erkang Zheng

In the dynamic landscape of software development, the concept of an application has undergone a significant transformation. Gone are the days of monolithic architectures that encompassed all functionalities within a single codebase. Instead, modern applications have embraced a microservices architecture, offering increased flexibility, scalability, and resilience. However, this shift has made defining the application's boundaries more complex than ever before.

To understand the true boundary of a modern application, you need to consider three factors: components, infrastructure, and ownership.

Components

In the modern software landscape, applications are composed of a multitude of interconnected components. An application is no longer made up of a single codebase. Additionally, it is assembled by leveraging many layers of nested libraries, components, and dependencies, both private and public open source code modules.

The components of a software application are defined as the *software bill of materials* (SBOM), which is a nested inventory for the software and a list of ingredients. That's an in-depth topic by itself.

Infrastructure

The shift toward cloud- and software-defined infrastructure introduces a high degree of complexity and dynamism. We must consider the following factors:

Code to infrastructure
: To define the boundary, we need to encompass not only the application's codebase, but also the infrastructure required to run it. Modern applications are closely tied to their underlying infrastructure, often employing infrastructure as code practices.

Distributed nature
: Modern applications span multiple microservices, each operating independently yet as part of an interconnected web. The application's boundary must encompass all the interconnected microservices involved in delivering the desired functionality.

API endpoints and external integrations
: Applications expose various API endpoints, which are an integral part of the application boundary. Additionally, external integrations with third-party services, both upstream and downstream, contribute to the complexity of defining the application.

Middleware and communication channels
: The middleware components that facilitate communication between microservices and handle data transformation must be considered as part of the application boundary. Understanding how these components fit into the overall architecture is crucial.

Data stores and workloads
: Data stores and distributed workloads form an essential part of modern applications. Including these elements in the application boundary is necessary to ensure comprehensive visibility and understanding.

Ownership

With multiple microservices, code repos, modules, and infrastructure resources, each potentially owned by different teams or individuals, establishing clear ownership becomes nontrivial. We must consider:

Distributed responsibility
: Microservices are often developed and maintained by different teams, each responsible for their specific functionality. Determining who owns the application as a whole can become challenging.

Interdependencies
: Services and resources rely on each other to deliver the desired functionality. Ownership of the application should encompass these interdependencies to avoid potential gaps in responsibility.

Cross-functional collaboration
: Defining application ownership requires collaboration among various stakeholders. Building shared understanding and communication channels is crucial to ensure effective ownership.

The Foundation of Modern Cybersecurity

Understanding the modern application boundary is not merely an academic exercise; it forms the foundation of modern cybersecurity. In today's software-defined world, where software as a service (SaaS) is prevalent, applications' security posture relies on a comprehensive understanding of their boundaries:

Threat and attack surface
: By delineating the application boundary, cybersecurity professionals can identify the potential attack vectors and areas of vulnerability.

Access control
: Properly defining application boundaries aids in establishing effective access control mechanisms. Access can be limited to specific microservices, APIs, or data stores, reducing the attack surface and minimizing unauthorized access.

Incident response
: When security incidents occur, understanding the application boundary allows for targeted incident response and efficient mitigation. Incident response teams can quickly identify affected components, evaluate impact, and take appropriate measures to contain and resolve the incident.

Compliance and auditing
: By understanding which components fall within the scope of the application, organizations can ensure adherence to relevant security and privacy regulations.

Being able to connect the dots with context across all of the dependencies and infrastructure of a software application allows us to understand the true boundaries of a software application and therefore its threat model. Being able to identify the right owners is critical to the efficiency and efficacy of the product incident response process and software vulnerability management program.

Common Best Practices in Application Security

Laxmidhar V. Gaopande

Cyber threats are increasing every day. The damages are phenomenal in terms of loss of brand reputation and loss of important data to the hackers. Banking, finance, insurance, healthcare, and ecommerce sites are highly vulnerable to hackers. Globally, digitization has increased the risk of cyberattacks, more due to an increase in cloud-based development, use of open source technologies, and new insecure development tools other than poor coding practices.

Code Scanning and Reviews

It is important that, during coding, developers ensure that they write code that is secured and not vulnerable to cyberattacks. MITRE and the OWASP have published a list of critical coding errors that cause security risks.

Developers must ensure that various vulnerabilities are not open, such as unencrypted data, dangerous file upload, no validations on harmful data while uploading, poor strength of passwords, unchanged passwords, use of open source code without checking its integrity, broken algorithms, redirection to untrustworthy websites from URLs, avoiding the use of non-TLS for the website access, network misconfigurations, unpatched systems, and so on.

Application scanning tools must be regularly used to detect vulnerabilities, including improper configurations, poor quality of programming, remote code execution (RCE), SQL injection, command injection, cross-site scripting (XSS), cross-site request forgery (CSRF), broken authentication and session management, weak key generation, inadequate password hashing, hard-coded passwords, improper system resource allocation, flaws in business logic, improper restriction of XML external entity (XXE) references, path traversal, null pointer references, insecure direct object references, buffer overflows, insecure cryptographic storage, and unauthorized access.

Leverage AI for Better Detection and Automation

The use of AI in cybersecurity has increased for detection, prediction, response, vulnerability management, improvements in authentications, behavioral analysis, controlling phishing, and threat hunting. AI can increase the detection rate by almost 95% by replacing traditional methods.

AI-driven *security operations centers* (SOCs) are used by enterprises to tackle threats efficiently by detecting unusual behavior in the traffic. A sudden increase in traffic or abnormal traffic can indicate the possibility of cyberattacks. Developers must use tools and/or build the inherent functionalities in their software to detect such abnormalities and identify mechanisms to protect their applications.

AI-based cybersecurity tools are available in the market, such as AI-based firewalls, AI and ML to analyze the network traffic, AI software to detect email cybersecurity threats, neural networks to find code that bypasses security measures, etc. Developers and AppSec professionals must constantly watch various products, platforms, and tools to protect their applications and websites from cyber threats.

Build a Bug Bounty Program

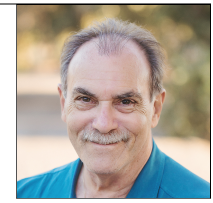
Enterprises are using either in-house teams to test the applications or outsourcing to external vendors for testing against cyber threats; in both cases, there is upfront cost, limited resources, and limited testing. Hence, the adoption of bug bounty platforms and their usage are increasing globally.

Bug bounty is a process where the companies list their digital assets on bug bounty platforms to detect the vulnerabilities and then partner with skilled hackers enrolled on the platforms. Companies then pay the bounty (the reward), for valid vulnerabilities reported. This makes the testing almost 24-7 and there is no upfront fixed cost because you pay as per bug reported. It's a win-win situation for both the companies and hackers (who get paid for their work). Bug bounty programs can be private or public. As the bug bounty industry matures, we will see specialized bug bounty programs in application testing, the Internet of Things (IoT), cloud security, and blockchain.

No doubt, AppSec is important to protect against cyber threats and vulnerabilities. Apply the AppSec best practices based on your organization's risks, culture, and resources.

AppSec Is a People Problem—Not a Technical One

Mark S. Merkow



Bootstrapping a new or low-maturity AppSec program using security technology alone is a guaranteed recipe for failure or suboptimal outcomes. Throwing technology into an environment that expects developers to address findings but fails to prepare them technically and psychologically with the knowledge and skills needed is a recipe for bad results.

It's people who are responsible for all aspects of software—from inception to design to development, testing, and implementation. They're also responsible for software security—whether they realize it or not; no one other than the person developing the software can effectively secure it. Ignoring this responsibility or ignoring the human aspects of software development when introducing new security-focused tools in developer workflow quickly leads to chaos, anger, missed deadlines, and bewildered management.

There are few more effective ways to demoralize an entire development organization than by running security scanners on their applications, throwing the results over the wall, and mandating those developers to "deal with it" somehow. Making matters worse, traditional education that prepares programmers and IT roles for new technologies, new languages, and new platforms doesn't arm learners with the skills they need to meet the demands of organizations that require resilient, high-quality applications that can be constructed quickly at acceptable costs. Many development team members may enter the workforce never hearing the term *nonfunctional requirement*, also known as quality or supplementary requirements, which are aspects of a software system describing its characteristics, attributes, or properties without specifying specific behaviors. Unlike *functional requirements*, which define what the system should do, nonfunctional requirements focus on how well the system performs its functions.

Since the foundation of any successful AppSec program is the people who compose it, a better bet is to start with the development community itself to understand its mandates, mechanics, structures, and "hooks" where AppSec can be tailored.

People are the lubricant in all software development projects—without required training, support structures, and commitment to its people, organizations have little chance at building (and operating) a useful and practical secure development life cycle that naturally produces secure software.

Software development management, not just the application security team, needs to own the responsibility to break old bad habits, instill good new habits, and educate the workforce adequately to fill these gaps. To start the process, awareness of software security as an institutional issue is needed to set the stage for everything that follows. Awareness drives interest and curiosity and places people on the path to wanting to learn more. This awareness greases the skids that enable smooth engagement in software security education and ongoing involvement in AppSec-related activities that "keep the drumbeat alive" throughout the year. The security team should assist in bringing up security awareness and training developers with the right knowledge and skill sets to take ownership of developing secure code.

Focusing and including the people whose lives you're planning on changing will carry through all the aspects of software development. Once converted, these people will go to great lengths to make sure your program is successful—they have "skin in the game" and want to make sure everything is (eventually) achieved. These people will serve as your evangelists and flip on its head the notion of "us versus them" for the chief information security officers (CISOs) charged with secure software programs. Some of these people will become part of your Security Champion network of security-minded engineers who will help to make sure the software factory is secure from end to end and can be relied upon to produce secure applications—every time!

Empowering Application Security Professionals Through Cybersecurity Education

Michael Bray

In today's technology-driven world, where data breaches and cyberattacks are becoming increasingly prevalent, the role of AppSec professionals has never been more critical. These experts are entrusted with safeguarding digital assets, sensitive user information, and the overall integrity of software applications. To fulfill this crucial responsibility effectively, AppSec professionals must prioritize continuous education and deep understanding of cybersecurity principles. This essay explores the reasons why investing time in cybersecurity education is essential for these professionals, how it can elevate their capabilities, and practical ways to embrace this knowledge into their AppSec principles.

The cyber threat landscape is in a constant state of flux, with adversaries developing sophisticated techniques to exploit vulnerabilities in software applications. As technology advances, so do the methods used by malicious actors to breach security defenses. By staying informed about the latest cybersecurity threats, trends, and countermeasures, AppSec professionals can better anticipate and respond to potential attacks. Education equips them with the knowledge necessary to adopt proactive measures and strengthen their organization's security posture.

Strive to enhance your problem-solving skills! A well-rounded education in cybersecurity may help professionals understand the intricacies of different attack vectors and vulnerabilities, allowing them to think critically and strategically when developing security solutions. It enables them to analyze complex security challenges, identify potential weaknesses, and implement

effective security controls. The ability to approach problems from various angles is crucial in staying one step ahead of cyber adversaries.

Building a strong foundation in cybersecurity education provides AppSec professionals with a solid foundation in security principles, standards, and frameworks. From the fundamental concepts of secure coding to in-depth knowledge of industry-recognized frameworks such as OWASP, this knowledge forms the backbone of their security practices.

Understand your role in effective incident response and recovery. No organization is immune to cyber incidents, making incident response and recovery capabilities critical for AppSec professionals. Cybersecurity education equips them with the skills to detect, analyze, and respond promptly to security incidents. Understanding incident response best practices enables them to mitigate the impact of an attack and restore operations swiftly.

Embrace and seek out continuous learning. Cybersecurity is an ever-evolving field and staying up-to-date with the latest developments is essential for AppSec professionals, allowing them to remain relevant and adapt to emerging threats effectively. Engaging in webinars, workshops, and conferences and reading industry publications are some ways to expand their knowledge base.

Integrate cybersecurity education into your application security principles with the following:

- Training and certifications such as Certified Ethical Hacker (CEH), Certified Information Systems Security Professional (CISSP), and Certified Secure Software Lifecycle Professional (CSSLP) may validate the expertise and commitment of the security professionals to the field.
- Collaborative learning can establish cross-functional collaboration between application development, operations, and security teams, foster knowledge sharing, and encourage the adoption of cybersecurity principles throughout the development life cycle.
- Capturing lessons learned encourages professionals to share experiences from past security incidents, facilitates collective learning, and helps prevent similar incidents in the future.
- Security awareness education on a regular cadence helps cultivate a security-conscious mindset and instills a sense of responsibility for AppSec among professionals.

The importance of cybersecurity education for AppSec professionals cannot be overstated. Armed with comprehensive knowledge and continuous learning, these experts can confidently face the challenges posed by an ever-changing cyber landscape. By embracing cybersecurity education and incorporating it into their AppSec principles, professionals are well-equipped to protect their organizations, users, and data from cyber threats, ensuring a safer and more resilient digital future.

Why You Need a Practical Security Champions Program

Michael Xin and
Sandeep Kumar Singh

As a security professional, you may have been challenged by application teams that couldn't make your priority their priority, and security controls in the software development process that were not as great as you wanted. You may want to test out *Security Champions* practices! A Security Champions program is an effective way to scale and distribute security across development teams. A Security Champions program is a collaboration between a security team, development teams, and an executive sponsor who supports and promotes the program. High-performing individuals within development teams are nominated as Security Champions, undergoing additional security training to act as ambassadors for security.

The Security Champion program offers several benefits. It improves security awareness and the adoption of secure coding practices during software development. It also enhances collaboration and knowledge-sharing between security champions and the security team, resulting in more effective security reviews and testing. Additionally, it improves the working relationship between the security and engineering teams, fostering a culture of continuous improvement and integration of security measures.

Effective deployment of a Security Champions program requires careful planning and preparation. Before launching the program, it is crucial to identify potential challenges related to the organization's culture, readiness, bandwidth limitations, training requirements, and resource allocation. Creating a centralized hub to store program information, such as an overview, roles and responsibilities, timeline, roster of security champions, success metrics, and training details can be highly beneficial. Encouraging top leadership support by discussing the program in staff meetings and motivating

departments and leaders to participate and nominate participants helps in expanding the program's reach.

Determining the required bandwidth commitment is also important. Developing a year-long plan that outlines the activities security champions will undertake can help calculate the average weekly hours per person. For example, security champions may dedicate about 25% of their time, allowing flexibility for teams with a minimum availability of 10%. Initially, the focus in the first year should be primarily on training, application review, and collaboration on *static application security testing* (SAST) to reduce the backlog of security defects. Subsequent years can involve additional collaboration on *software composition analysis* (SCA) and *dynamic application security testing* (DAST), threat modeling, and penetration testing (pen testing). It is advisable to limit the responsibilities in the first year to establish a strong foundation for the program and enable security champions to adapt comfortably to their roles.

For a Security Champions program to succeed, it is important to promote it effectively. Target teams need to be aware of the program's existence and its benefits. Promotion strategies can include creating posters, discussing the program in team meetings, and presenting an overview (led by executive leaders) to the entire organization.

To ensure ongoing participation and engagement, various techniques can be employed. This includes conducting quizzes and knowledge-sharing sessions on security and Secure Development Lifecycle (SDL), establishing dedicated communication channels, scheduling regular learning and collaboration meetings, providing guidance materials, and encouraging participation in security-focused events. Before launching the program, security champions should be familiar with various training topics. This can include an overview of the program, the organization's security policy, SDL activities, and security assessment tools.

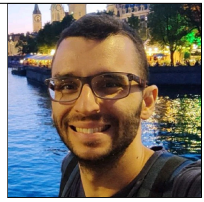
Motivation can be maintained by rewarding and recognizing the achievements of security champions. This can be done through custom swags, allocating budget for external security training for top performers, highlighting accomplishments through newsletters and company-wide content, and utilizing internal recognition tools to increase awareness of their impact.

While implementing a Security Champions program, valuable insights can be derived from past experiences. The OWASP Security Champions project serves as a helpful guide to provide consistent direction. At the same time, it's important to understand that different organizations have unique cultural

needs and challenges, so a customized approach is necessary. Regularly engaging in conversations with security champions is essential for ongoing evaluation and overcoming any obstacles. Ultimately, a Security Champions program can greatly contribute to the success of the application security program and to the overall success of your security team, making it an indispensable asset.

The Human Firewall: Combat Enemies by Improving Your Security-Oriented Culture

Periklis Gkolias



In cybersecurity, tech dams alone are not enough to defend against the ever-evolving landscape of threats. We all agree that network firewalls, web application firewalls (WAFs), security information and event monitoring (SIEM) solutions, and other related software are essential layers of protection.

But there is another crucial line of defense that often goes overlooked: the human firewall.

What do I mean by the term *human firewall*? I am referring to the collective awareness and security knowledge of an organization. A defense is formed when every individual actively participates in the defense strategy. It's similar to a firewall: one open port can expose the internal network to the attackers, and one individual can become the weakest link and allow the bad actors to come through.

We create defense when everyone understands that the weakest link can wreak havoc in an otherwise perfect creation. From the CEO to the newest intern, all are equally essential parts of the firewall.

As you try to add many technical controls in the application development, fix vulnerabilities in different places, or apply security policies in the environment, be aware that those tasks were done mainly by human beings. Understanding that human beings are a crucial part of security controls will help you shift the paradigm on how to get the security done right!

Recognizing External Threats

External threats are risks that most of us are familiar with. These come from entities outside the company that aim to attack it, with varying motives. They can use numerous tools, tactics, and techniques to achieve their goals, such as exploiting vulnerabilities in applications, leveraging open source intelligence, or launching phishing attacks. Such actions can lead to data breaches, intellectual property theft, financial losses, or reputational damage. Building a human firewall against these external threats will make employees more vigilant and better prepared.

Recognizing Insider Threats

Insider threats come in various forms, for example: disgruntled employees, unintentional errors, or manipulated staff. As with external threats, they can lead to similar classes of damage. Building a human firewall requires acknowledging that insiders have the potential to be attack vectors causing security breaches to help understand that proactive measures are necessary.

The problem with insider threats is that there is a thin line between being cautious and not trusting your people.

Empowering Employees Through Education

A security-aware culture begins with education. Employees should receive comprehensive security training. That should include best practices for handling sensitive data, recognizing phishing attempts, and adhering to security policies. Of course, find fun ways to engage people instead of providing them boring training materials.

Empower them with the knowledge to identify suspicious activities and the confidence to report potential threats promptly. Remember, only when everyone takes security as part of their own responsibility will the human firewall work properly.

Promoting Open Communication

A flourishing human firewall relies on a culture of open communication. Employees should feel comfortable reporting security concerns or incidents without fear of disciplinary action. "Say something when you see something" should be a common practice in a good security culture.

Engaging Leadership

Leadership sets the tone for the organization's culture, including its approach to cybersecurity. Leaders should participate in security training and lead by example by adopting the security measures as expected by others in the organization. For instance, because you are a CEO, this doesn't mean you are allowed to escape multifactor authentication (MFA) or skip security training.

Conducting Regular Security Drills

Reinforce security awareness through regular security drills and simulations. Conducting mock phishing exercises, incident response scenarios, and social engineering tests can help keep employees alert and prepared to tackle real-world threats.

Rewarding and Recognizing Secure Behavior

Positive reinforcement goes a long way in strengthening the human firewall. Ensure that you reward employees who consistently demonstrate security-aware behavior and contribute to maintaining a safe digital environment. Public recognition for security-conscious employees can motivate others to follow them.

Shifting Everywhere in Application Security

Sounil Yu

The Changing Landscape of Application Security

Mark Andreessen's observation from a decade ago that "software is eating the world" perfectly captures the pivotal role software now plays in almost every organization's success. In this age of rapid digital transformation, nearly every company is a software company, driving a rapid expansion of the software attack surface and redefining the landscape for AppSec.

With software taking on such a prominent role, there's a race to produce more and release it faster. From 2011 to 2014, Amazon went from deploying software every 11 seconds to less than one per second.[1] Today, it's safe to infer that the deployment cycle is nearly continuous. This breakneck pace not only produces a vast volume of software code but also presents security challenges, as the volume of code to inspect for vulnerabilities explodes.

The Traditional Shift Left Paradigm

To deal with these software vulnerabilities, the security community has rallied around the need to *shift left*, a strategy which, in health terms, is akin to preventing disease instead of treating disease. By enforcing security-by-design principles before software is deployed, security flaws could be addressed early in the development phase, where they are relatively cheaper and easier to fix.

However, the shift left concept originally emerged when software deployments had much longer timeframes and would occur every few months (or years). As with disease, the relative cost of fixing defects grows dramatically

1 *All Things Distributed* (blog); "The Story of Apollo—Amazon's Deployment Engine," Werner Vogels. Posted November 12, 2014.

with time, reaching 60 times to 100 times[2,3] what it would have cost had it been addressed earlier in the life cycle, so businesses were incentivized to focus on prevention.

Today, the modern development process iterates so quickly that the relative cost to fix flaws *after* deployment is minimal, thus removing much of the incentive to shift left. This means that *the cost imperative for shifting left is no longer there*, but we still have a security imperative to avoid building software without vulnerabilities. Although building security in may seem important to security practitioners, this security-first mindset often loses out to the more compelling business need to release software faster, even if the software contains vulnerabilities. In this new world, the goals for security now need to be about "shifting everywhere," injecting security measures throughout the entire development life cycle, not just in the beginning.

The Role of Infrastructure and Automation

A major change that has enabled developers to incorporate security checks throughout the development life cycle has been the adoption of an "everything-as-code" approach. This includes both how software is built and where it is run. The infrastructure layer has become an integral part of software delivery and the software's attack surface. This requires additional checks on containers, Kubernetes, cloud configurations, and infrastructure as code. This everything-as-code approach stores all information in code repositories, promoting versioning, drift detection, change control, and more. Crucially, automation is at its heart. In a fully automated software development pipeline, any manual steps stand out and slow the process. Application security teams need to adopt security automation wherever development teams have also incorporated some form of automation.

2 See the National Institute of Standards and Technology (NIST) report: Gregory Tassey, *The Economic Impacts of Inadequate Infrastructure for Software Testing*, (National Institute of Standards and Technology, 2002).

3 See the IBM article: Maurice Dawson et al., "Integrating Software Assurance into the Software Development Life Cycle (SDLC)," IBM System Science Institute, 2010.

Re-envisioning Application Security

This paradigm shift calls for a new approach to AppSec that shifts everywhere. In summary, these changes include:

Accelerated delivery
: Seamlessly integrated security checks throughout the entire development life cycle that perform the necessary checks without derailing the development process

Everything as code
: Security automation that aligns with the increasing prevalence of everything defined by code, from infrastructure to operations to security governance

Rapid feedback loops
: Immediate feedback systems that allow for swift diagnosis and recovery when something goes amiss

The future of AppSec is exciting because these changes, which improve security, are often embraced by the business because they usually lead to better code quality and speedier deliveries. Our goal with AppSec should not be just to keep pace but to proactively anticipate and adapt, ensuring that as the software world evolves, security isn't just an afterthought but an integral, seamless part of the entire process that helps propel the business forward.

Beyond Barriers: Navigating the Path to a Successful AppSec Program

Yabing Wang

I started my career as a Java developer and switched to security after five years. In my 20+ years of cybersecurity, I've run several successful application security assurance programs (ASAPs). Here are some lessons learned that may give you some insights.

What Are the Core Components of the AppSec Program?

As with other successful programs, the AppSec program should consist of people, processes, and technology:

Introduce an SSDLC process.
: Develop, publish, and communicate a policy or standard on this process. This is a proactive step toward bolstering the overall security posture of your software development practices. This structured approach integrates security considerations seamlessly throughout the software development life cycle, ensuring that security is not an afterthought but an inherent part of the entire process.

Evaluate threat modeling.
: Threat modeling is a step that can guide the architecture and the security controls implemented. However, it could become cumbersome, especially in big companies with traditional waterfall processes. You may want to test it out, automate it, and determine if this may work in your company or find another way, such as security reviews, to accommodate this.

Include automated testing capabilities.

Based on your company's situation, you may not need all types, whether they are static, dynamic, integrated, runtime application security testing, or pen testing. Consider at least one testing tool in the software development life cycle (SDLC) process, and pen testing outside of the SDLC process. It is highly recommended to adopt runtime protection capabilities, for example, a WAF is a great defense capability.

Define the SSDLC governance process.

This includes a security architecture review in the design phase, a shift left approach, and a sign-off before it's deployed to production. However, automating those reviews and sign-offs is more effective! This also includes creating metrics to measure the vulnerabilities discovered and remediated, especially for each team. Report out to the management above the chief information security officer (CISO) and the development leaders and track the progress as the risk mitigates.

Include API security in the program.

APIs are vital connectors between different software systems, enabling seamless data exchange and interaction. However, they also present potential vulnerabilities that malicious actors may exploit. By incorporating robust API security measures into your program, you fortify your system against various cyber threats and ensure the integrity, confidentiality, and availability of the data being exchanged. This involves implementing authentication protocols, authorization mechanisms, and encryption standards to safeguard sensitive information from unauthorized access.

Continuously train developers.

Figure out a fun, interesting, and attractive way to educate developers to learn about vulnerabilities and how to avoid them. To support the idea that security is part of software quality, we should equip them with the right tools as well as the right mindset.

What Are the Success Factors of the AppSec Program?

I've seen how AppSec programs ran in waterfall SDLC, adapted to Agile SDLC, and extended to DevSecOps. The most successful programs are the ones where security can be automated in every phase of SDLC, from initial design through integration, testing, deployment, and software delivery:

Make the development leadership your advocate and partner.

The AppSec program is not just a security initiative; it has to be a development initiative. When it's a key performance indicator (KPI) or objectives and key results (OKR) for both departments, the priority is set and the adoption is easier.

Take the opportunity to raise the security culture; that is, security is everyone's responsibility.

The key is to deliver a clear message that the success of the program relies on everyone in the SDLC to play an important role. Figure out a way that works in your culture, to hold everyone in the SDLC accountable for including security in their day-to-day job, especially as part of the definition of code or product quality.

Start the program small, and don't introduce too many changes at one time.

Mature and optimize the program by introducing incremental changes over time.

Make the process as simple as possible and as automated as possible.

Developers have their priorities to focus on features and other parts of quality. So apply automation wherever you can and enforce the policy-as-code, such as "no code with critical vulnerability should be deployed to production." Go toward DevSecOps.

Leverage security champions.

It can be challenging to spread security practices across large development teams, and it can be difficult for the security and development teams to fully understand each other. Bridge the gap by developing a security champion role. This is typically a developer who has an interest in security and can help communicate between the two teams.

Be agile, be simple, and be there with them! Remember, security is there to enable business, not to block business!

PART II

Secure SDLC

Building an Application Security Preparation Mindset

Andrew King

Applications are building blocks of functionality within an organization's infrastructure. These are software components that perform critical functions to meet an organization's needs, and as such, require special consideration. Applications are one of the weakest links in infosec systems due to the potential exploitation of known and unknown vulnerabilities. Roughly 70% of external attacks come from exploiting software or web applications. When you buy a product and insert it into your organization, you have a proverbial scapegoat when things go wrong. However, when you build the product, the responsibility lies solely on the creating team. As such, you should have security in mind before even beginning to build your application.

There are four concepts you need to consider before you build security in applications: mindset, logging and monitoring, scope, and best practices.

Mindset: How Can You Prepare?

Application security has a major impact on business operations—both good and bad. From project conception, it is key to build a security mindset. Security as an afterthought always takes exponential effort to retrofit.

It is essential to create a playbook for when things go wrong, because they will. Come armed with preventative reactability. Bad things are going to happen, and you don't want to chase issues in retrospect. Anticipate and plan for them. How will you and the team respond to a security event? What data will you bring to that conversation? Who are the system users? What systems and third parties do you interact with? Can you immediately shut off access and continue to operate the business? Can you roll back versions quickly? Can you audit all behavior, user access, and traffic or usage? This preparation can come from threat modeling if you think of answers for everything that arises.

This assessment includes identifying all information assets, categorizing them, identifying all possible threats, and outlining all vulnerabilities and risks based on impact and probability of occurrence. Now you have a framework of what you can expect when issues come up.

Logging and Monitoring: Do You See What Happened?

Assume compromise. During the threat modeling, think through how you would measure all the risks and ensure those use cases are covered with logging. These logs are your figurative eyes, offering analysis tools that security analysts access to trace actions and determine what occurred in retrospect. Monitoring is next, but you need to understand what to look for to detect anomalous behavior. Define, measure, baseline, and know deviations from expected behavior to flag for analysis.

Scope: Can You Do It All?

Keenly understand the purpose of your product and only build to that. What does your application do that others don't? What data does it have? How does that need to be protected? Amazon Web Services (AWS) has a notion of only adding features that deliver value to the customer and nothing else. This is an effective abstraction since anything extra is also an additional threat. By deliberately not adding superfluous code, you remove adding something potentially harmful. Every organization has finite resources, so by only taking on what adds value to the customer, thereby reducing risk, you allow your limited resources to work exclusively on the most critical features. It's a win-win.

Best Practices: Can You Borrow from Others' Experience?

You do not need to rebuild what others have built; instead, build upon it. Best practices already exist, such as using an SSDLC, strong authentication, least privilege, data sanitization, end-to-end encryption, API security, secure architectures, patching, and pen testing to validate controls and give artifacts for auditors to ensure due diligence. These can be the difference in fines and penalties. And if at all possible, you should switch to a continual patching cadence with the ability to patch at any time, to address zero-days.

Have fun. The more you enjoy your work, the better you perform, which will be reflected in your products and innovation.

By planning the security of your application from the very beginning—before you even build the application—you can establish a robust foundation for safeguarding sensitive data, mitigating potential vulnerabilities, and ensuring a resilient defense against security threats throughout the entire application development cycle.

How to Assess Security Mindset in Application Design

Anuj Parekh

We all want our applications and organizations to be secure. We come up with different processes and implement different tools to secure our applications and organizations, but sometimes we overlook the most important aspect of application security: to build a security-focus mindset of individuals and teams in an organization.

We often hear that security is everyone's responsibility, and that is true. When we are able to raise the security culture and build the security mindset within the product and engineering team, it will be more efficient than adding more tools or gates in the process. When they become the security champions in protecting applications and data, that's what a successful application security program would be!

You can start assessing your organization's security mindset from multiple aspects. The following are some sample questions that may guide you on the assessment and even help you build the security mindset in your product, engineering, and data groups.

Data exposure and minimization

What type of data exposure and severity of data exposure is there in case the application, data, or accounts get compromised?

Are we doing everything we can to minimize data exposure (especially sensitive data such as personally identifiable information) and is product design taking privacy by design into consideration?

Secure coding practices

Are developers following secure coding practices?

Are developers evaluating their implementation for logical vulnerabilities that can allow someone to circumvent authentication, authorization, and access controls within the applications?

Are developers writing test cases to validate that the applications are not vulnerable to attacks that could compromise applications?

Least privilege and infrastructure security

Is the engineering team following the least-privilege methodology when writing applications and building infrastructure?

Security tools and deployment

Does the organization have security tools such as SAST/DAST deployed as part of CI/CD to prevent certain vulnerabilities from making it into applications?

Is the CI/CD pipeline properly secure from data leakage and supply chain attacks?

Are there security tools such as WAF/network firewall deployed and configured to protect the application from attacks postdeployment?

Patch management

Is there an established patch management process to patch vulnerabilities in applications and infrastructure?

Application monitoring and alerting

What type of application monitoring and alerting controls are implemented to detect attacks against the application?

User training and awareness

Are the users properly trained to handle sensitive data that could potentially be accessible from the application?

Are the internal users properly trained so they don't fall victim to phishing attacks or download malicious software on their machines that could compromise sensitive data?

Endpoint security

Are there antimalware tools deployed and properly configured to prevent malicious software from being installed on endpoints?

Incident response

What is the incident response process to contain and investigate incidents?

These questions should give you some idea of where your organization stands when it comes to the security mindset, and these are some of the aspects of application security that should be embedded in the thought process of individuals and teams when working with applications and data. We want to get to a place where application and data security become second nature for individuals and teams in an organization; the way to achieve this is by continuously training them on various standards, processes, attack vectors, and any security issues we might have come across in our applications in the past.

Getting Your Application Ready for the Enterprise

Ayman Elsawah

Selling upmarket to enterprises has become ever so important for many startups. Selling to enterprises, however, is starkly different from selling to the consumer market or even other startups.

For one thing, enterprises have security teams, and oftentimes a long list of security requirements. Getting past the gauntlet of security questionnaires and scrutiny is a hurdle in itself. However, if your application does not have key security-focused features, it may be a nonstarter for the enterprise.

The following are some features you may want to consider when building your product.

Enterprise Single Sign-On

Enterprise single sign-on (SSO) in this case does not mean supporting Google or Twitter login. This means that you are allowing the enterprise to integrate your product into its internal enterprise user directory, which can often be Okta, Ping Identity, OneLogin, or similar.

Your application will need to support SAML and OIDC. A majority of enterprises use SAML, so if you had only one to pick, then start there. OIDC is a more modern approach, and it will just look good on your organization if you support both.

The benefits of SSO integration include a better user experience for customers, less product friction, and more importantly, relief from storing any credentials (passwords or security questions) outside the enterprise, which reduces your application attack surface. The customer is now managing authentication, so you absolve yourself of that responsibility. SSO also improves the deprovisioning of user access. When a user or customer is leaving, there is only one place to disable the user, not everywhere!

Roles and Access Controls

Enterprises will need to manage authorization, which dictates what privileges a particular user may have in a system. Enterprises will expect at least four roles, but more often than not, they would like a way to create custom roles with specific privileges.

If you had to pick just four roles, the following table is what I would suggest.

Role	Privileges
Administrator	Full privileges
Power user	All privileges except for user management
Standard user	Everyday user role for using your application
Read-only	Read-only access to the entire application

You can do it yourself or use standard libraries for these four roles, but make sure you always keep them updated and keep a lookout for vulnerabilities. There are plenty of third-party applications that do the heavy lifting for you as well, but, of course, come at a cost.

Audit Logging

Another feature enterprises are often looking for is audit logging. Enterprises want to know who did what and when in all their applications. There are a few parts to having a good audit logging feature in your application.

First, you need audit logging functionality. Not sure what to log? Here are some examples of artifacts to capture:

- Username
- Action performed
  - i.e., Login, Logout, Read/write actions
- Context
  - IP address (required)
  - User-agent

Second, audit logs will need to be available for *at least* 90 days. For a business-critical system such as a human resources information system (HRIS) or similar, it may be at least one year or an indefinite amount. Of course, it's hard to have all these logs available online, so you can have a function that requests archived logs on demand, for example.

Third, many enterprises will want to have the ability to integrate audit logs programmatically into their own log management or security information and event monitoring (SIEM) system. A manual export of logs to comma-separated values (CSV) will not be sustainable for most enterprises, so it's a recommended option.

Finally, there are many ways to allow companies access to your audit logging system. I won't go into detail here, but here are the options many providers can provide:

- HTTP(S) endpoint
- Direct API
- Webhook
- S3 bucket using AWS cross-account roles

Aside from ensuring your entire company is securely aligned with a cybersecurity framework, having your application be enterprise ready will reduce friction on your sales team, is good security, and will impress security teams on enterprises.

Reductio Ad Applicationem Securitatis

Darryle Merlette

Early in my career, a wise man once told me: "There are only three things you can do with a computer: read, write, and change (RWC) data." At first, I didn't appreciate this reduction to the field of computer science—*reductio ad computatrum scientia*—where I had just devoted more than six years studying. But as time went by, I realized he was actually correct. Although databases teach us the notion of create/read/update/delete (CRUD), the correspondence to his trifecta classification is clear.

The history of computer security shows that databases are a prime target for exploitation through attacking applications. What are the implications of the RWC classification? Whenever a new exploit or attack is disclosed, it can be instructive to examine where and how the three processes come into play—a reduction to application security or *reductio ad applicationem securitatis* so to speak—can provide insight to developers and help them apply proper controls to the applications and databases.

Read

When data is read by an application, it is used as input to a process that allows the application to do that for which it was intended. For example, an unsorted list of numbers will be used to create a sorted list in a sorting application. However, some applications can be given data that allows it to do that for which it was *not* intended. A good example is the critical Shellshock Vulnerability of 2014 (CVE-2014-6271), which affected the Bash shell that comes standard with most Unix/Linux distributions. This allowed attackers to execute arbitrary code via specially crafted environment variables containing function definitions. There are other types of attacks, such as SQL injection (SQLi) and cross-site scripting (XSS), that are similarly made possible by reading crafted input. The lesson to be learned when building applications is to be sure to validate and/or sanitize data before passing it on for further

processing. The lesson to be learned when testing applications is to check that malformed or unexpected input doesn't lead to undesired behavior.

Write

When data is written internally or externally by an application, it should be done in such a way as to maintain the integrity of the application itself. A few decades back, the most common form of application exploits came from buffer overflows—when specially crafted data is written to internal addresses in such a way that it overwrites the expected commands so that malicious commands can be run instead. Thankfully, modern architectures contain protections such as address space layout randomization (ASLR), which makes it hard for attackers to predict the execution address layout. Going back to the sorting application example above, if the output is still unsorted, then it would not be considered a very good sorting application! But suppose the sorting application actually gave the expected output, and in addition, also overwrote critical files with garbage data and then printed a message to the screen instructing you to pay in cryptocurrency to get your files back? Such would be the case of ransomware disguised as a sorting application. The lesson learned when building applications, even with ASLR, is to guard against using unbounded writes. The lesson learned when testing applications is to make sure the application is signed by a trusted entity and has not been modified.

Change

The bulk of complexity in computer science has to do with data being manipulated. The act of sorting or putting a list in order, analyzing patterns in data using tools like Fast Fourier Transform, making predictions with a nonlinear regression, training a large language model (LLM), etc., all have to do with changing data. The difficulty here is that malware and exploits also perform data changes that are often indistinguishable from valid manipulations. This is where more advanced techniques, such as running debuggers, disassemblers, and other static or dynamic code analysis tools, come into play. One must inspect the innards of the code and figure out where it is doing things that will lead to malicious results.

It is clear that principles of AppSec must be taken into account throughout the life cycle of data—whether the data is being read, written, or manipulated. Diligent attention to such details will ensure your applications are as secure as possible.

Automating the Risk Calculation of Modern Applications

Erkang Zheng

When it comes to assessing the security risk of a software application, relying solely on automated testing tools can provide an incomplete view. Security teams often employ tools such as static application security testing (SAST), dynamic application security testing (DAST), and software composition analysis (SCA) scanners, which generate a number of findings and assign severity ratings. These findings are then used to determine the risk associated with the application.

However, this approach fails to consider crucial factors. Merely comparing the number of findings or the severity of those findings between applications does not necessarily indicate their relative security or risk. For instance, an internal application used by 10 users with more numerous or severe findings may be less risky than a public-facing production app with a million users and with much fewer findings. The context in which the application operates greatly influences its risk profile.

Similarly, the frequency of testing and the application's history play crucial roles in assessing its security.

In order to achieve a comprehensive and accurate measurement of an application's security risk, a holistic approach is necessary. I'd like to introduce an automated application risk modeling process, *continuous application risk evaluation* (CARE). The concept of CARE is to continuously measure a software application's maturity and security on an ongoing basis and calculate a risk score that provides a holistic measurement of multiple aspects that contributes to the overall risk of a software product:

- What is the nature of the application? (design and business context)
- How is it built/maintained? (implementation and operations)

- Who is building and maintaining it? (team and maturity)

Design and Business Context

The design and business context of an application encompasses various factors that influence its security risk. This domain includes considerations such as the application's intended purpose, the sensitivity of the data it handles, and its overall architecture. For example, an application that processes financial transactions or stores personally identifiable information (PII) is likely to have a higher security risk than a simple informational website. Additionally, the business context, such as compliance requirements or industry regulations, must be taken into account to evaluate the application's risk accurately. The more business value an application provides, the higher its inherent risks are. Threat modeling is essential to assess the risk. Given the nature of the application, many of the risks are inherent and may not be avoidable.

Technology Implementation and Operations

This domain examines the technical aspects of the application's architecture, development, deployment, and maintenance. It involves assessing the robustness of the underlying technology stack, including programming languages, frameworks, libraries, and third-party components. Vulnerabilities in these technologies can introduce security weaknesses into the application. This is where vulnerability scanning comes in, for both applications and infrastructure. Additionally, operational protection and monitoring play an important factor here as well; for example, the deployment and operations of WAFs, API protection, or other forms of runtime protection.

Maturity of Team and Process

This maturity refers to the proficiency and experience of the development team and the effectiveness of the SDLC processes in place. An experienced and well-trained team that follows secure coding practices, such as threat modeling and secure code reviews, can significantly reduce the application's security risk. Additionally, a mature DevOps process that incorporates security checkpoints at various stages of the SDLC helps ensure that security is considered throughout the application's life cycle. Regular security assessments, pen testing, and incident response procedures are indicators of a mature security process.

The maturity of the team and process also extends to the organization's security culture and awareness. The presence of security training programs, clear security policies and procedures, and a proactive approach toward addressing security issues contribute to reducing the overall security risk.

By considering these three domains, security teams can evaluate an application's security risk more comprehensively. This holistic approach acknowledges that security is not solely dependent on the code but encompasses various interconnected elements that must be evaluated in conjunction to form an accurate assessment of an application's security posture. Further, a score can be calculated based on attributes from these domains for benchmarking and risk profiling, similar to the FICO credit score used for consumers.

A version of this approach is described in a Google patent.

A Coordinated Approach to a Successful DevSecOps Program

Han Lievens

Introduced over 10 years ago, DevSecOps has been accepted as one of the most important security practices today. Yet very few companies have been able to truly infuse security into every phase of their CI/CD pipeline.

Most companies I've worked with do understand the importance of the six pillars of DevSecOps, as defined by the National Institute of Standards and Technology (NIST):

- Collective responsibility
- Collaboration and integration
- Pragmatic implementation
- Bridging compliance and development
- Automation
- Measure, monitor, report, and action

And yet most organizations are unable to fully instantiate these principles into practice. Is achieving these seemingly reasonable principles really hard or are these concepts too idealistic and impractical? There's a lot of talk about changing mindsets and instilling the right culture. Does good DevSecOps truly require a company culture overhaul? And what or who would it take to get this done?

When we talk about infusing security into every step of the pipeline, the keys are visibility, correlated detections, and automated actions. The first prerequisite, of course, is collaboration between the development, security, and operations teams. Only when all teams are talking to each other will we achieve full visibility into processes, logs, and traces generated from tools

and a clear understanding of how and when automated actions can take place.

What's been missing for this complete cohesion to solidify is unified coordination in the form of a program manager. Not unlike the role of an insider threat program manager working with all the various teams involved, a DevSecOps governing person or team would be intimately familiar with the inner workings of each phase of the CI/CD pipeline and their corresponding teams. Led by the program manager, a working group would be responsible for identifying all appropriate data sources and working with each team to develop correlated detections and automated actions while being mindful of compliance and governance requirements.

Gaining visibility at each phase of the pipeline could be less challenging than you'd think. Most tools are already generating logs and are probably stored in disparate locations. It's just a matter of talking to the right person(s) to find out where everything lives. Once a data platform or some automated means to gather the data is in place, correlated detections can be put in place and automated actions set up to remediate and improve. A good DevSecOps practice requires security to be fully baked into the CI/CD pipeline. This means focused coordination between teams and a centralized data platform to detect, correlate, and act.

Let's not forget why we're here in the first place:

- Early identification and mitigation of security risks
- Faster response to security incidents
- Compliance with regulatory requirements
- Holistic risk management
- Cost-effective security measures
- Continuous improvement

It's time we take DevSecOps a little more seriously and hopefully start seeing organizations develop and build dedicated DevSecOps roles. The six pillars (or principles) are not unrealistic and impractical to achieve. A company culture overhaul should not be required. A successful DevSecOps program can be achieved with focused coordination from a program manager leading a working group, with support from leadership and access to the right teams.

What Makes Someone a Developer?

Helen Umberger

The definition of a software developer has changed in recent years. Traditionally, applications are developed by people who have specialized expertise in writing code. They have the training and the know-how to secure their apps. Today, apps can be developed by people without technical expertise. By using powerful no-code/low-code platforms, ChatGPT, and other generative AI tools, nontechnical people can create applications without actually writing code. Although it enables and empowers more people to create apps, it is also a cause for concern regarding security. Citizen developers are not trained to develop secure code. As such, application security professionals must pick up the responsibility of citizen programmers.

Now AppSec must pick up responsibility for all the new developers regardless of background, training, and—well, if they are even human. How can AppSec create security champions beyond their normal playground? How do we deliver security training beyond our normal annual "don't fall for phishing emails" training that we roll out to our companies? How do we make every employee and AI security aware? Because soon all our employees will be potential programmers. The genie is out of the bottle and cannot be put back.

AppSec needs to ensure that code from nontraditional developers still goes through pipelines and gets scanned, versioned, and controlled just like that of traditional developers. There is a need to create guidelines for code development and deployment for apps from nontraditional sources.

Code that accesses sensitive data and sources should be codified into libraries so that non-IT developers can use preexisting code for security-sensitive operations. Guidelines enforced by tools are needed to prevent code development for certain logic.

Guidelines that lock down generative AI code dictate which libraries can be scanned for code samples, put in rules based on OWASP, and require the use of the same type of pipeline your IT developers use for code promotion.

In addition, your new developers are not familiar with all the nonfunctional requirements that need to be followed, such as standards on data handling, standards on logging, and so on. Now the AppSec teams need to share this information with a wider audience, providing training on the how and why of the business.

Low-code/no-code still has code that can be hacked or poorly coded. The issue is that the developer is usually a business super user who can create functions by drag-and-drop logic flows, but behind the scenes, it is still code. Before it's released into your environment and manipulates your sensitive corporate data, it still needs to be scanned, reviewed, and versioned. AppSec needs to be involved in product selection, and the criteria is how easy it is to plug it into existing monitoring, libraries, repositories, and scanning. Low-code/no-code is not an excuse for poor practices.

Regulators don't care who created the code that causes a breach, exposes PII data, or creates erroneous outputs. The corporation is responsible. Make sure all requirements imposed on IT are also imposed on nontraditional developers. Make sure they have the training and awareness to be successful.

Without proper controls and guardrails, widespread citizen development could result in chaos. Because citizen-developed systems typically link to, change, or extract and analyze data from existing transactional systems, their developers also typically need an understanding of corporate IT architecture and guardrails for safe data access and usage. At the very least, someone in the organization should keep track of what applications have been developed and who developed them.

Why risk it? Because without low-code/no-code citizen development, you end up with shadow IT. Software as a service (SaaS) usage without corporate approval is at an all-time high. A citizen development platform is centralized, fills the need for shadow IT, monitors, trains, provides libraries, enforces data rules…and allows it to be brought under AppSecOps controls.

Segregation of duties still applies, and governance still applies! Having one source for AI code development, or low-code/no-code, selected by AppSec with control of the environment will help mitigate the negatives of this democratization of IT and data usage. It will allow monitoring, continuous deployment, the ability to triage outages, data compliance, logging for auditing, and so on.

Total AppSec

Hussain Syed

You may have heard the phrases "software is eating the world" or "every business is a software business."

In other words, software plays a huge role in how we live today. You may also have seen many companies claim they are tech companies because their products or services are software! Today, we have technology at our disposal that we have never had before. Sadly, in the world of software development, we struggle to keep up with this relentless change, leading to confusion and failure when we adhere to old ways of thinking and doing things.

We still have "buckets" of specializations where development, ops, and security teams do not work with things beyond their subject matter. Concepts and methodologies such as Lean, Agile, and DevOps/DevSecOps are all well-intentioned movements, but they are often used superficially, digressing to become a veneer of commercial productization or tooling.

In Vikram Mansharamani's famous HBR article, "All Hail the Generalist,"[1] and paraphrased in his book, *Think for Yourself*, he said, "The future might belong to those who are skilled not only at generating the proverbial dots of specialized information but also at connecting them. Those who see the big picture and tap into appropriate expertise when needed, would likely rule the future."[2]

So, it is time to realize that it is up to us to remove the barriers and expand our domain of expertise to have a better and deeper holistic understanding of the complete system. When you add security to the mix, it will start building a culture of awareness, and diligence will be reflected during the early phase of product management itself. AppSec or cybersecurity will truly become everyone's responsibility, and the stronger the collaboration, the more secure and reliable the ecosystem is.

1 Vikram Mansharamani, "All Hail the Generalist," *Harvard Business Review*, June 4, 2012.

2 Vikram Mansharamani, *Think for Yourself* (Harvard Business Review Press, 2020).

To restate the significance of this way of thinking, let's talk about soccer. Traditionally, each team member had a clearly defined role; that is, a defender was a defender, a midfielder was a midfielder, and an attacker was an attacker. Every position acted as a silo that had to interact with other silos. I hope by now you realize where I am going.

Then, at the 1974 World Cup, the Netherlands team stunned everyone and won hearts all over the world with their beautiful style of playing, called *Total Football*. With this style, no player—save for the goalkeeper—has a fixed position. Any player could take over any position as needed, and their knowledge of other positions aided their ability to play their primary role. Attackers would help with defending, and defenders would help with attacking. It resulted in many wins for the Dutch team. Think of what could happen if development, ops, and security teams learned more about each other.

I am not proposing such a radical approach toward modern product development, but a certain amount of curiosity and keenness to learn, share, and collaborate beyond each other's area of expertise will go a long way toward a stable and efficient AppSec community. Could we have imagined a day where, with the progress of the LLM, we have the option of collaborating—pair programming of sorts—with technologies like Copilot where the future has surely turned into today?

You're More Than Your Job

Izar Tarandach

One day at the end of a particularly busy quarter, I woke up at the usual dark-o'clock and realized it was a national holiday. I did not need to start getting ready for a busy day ahead. I did not need to caffeinate and get myself in the right mindset to tackle the latest challenge.

Without the usual agenda, I didn't know what I would do with myself that day, nor did I know what I wanted to do. Or what I could do. Or what was important to get done. Or that the option to *not* do the important thing also existed.

And so it dawned on me (pun not intended!) that my profession had finally turned into my identity. Without something to break, to protect, to analyze, or a crisis to respond to, I was lost in the woods without a compass. I had spent so much time training my head to think, "What could possibly go wrong and what do we do about it?" that now my fight-or-flight mechanism was deeply connected to conscious, constant threat modeling. I became my function, and I lacked a framework without doing it.

In psychology, the term for what I experienced is *enmeshment*, which describes a "situation where the boundaries between people become blurred, and individual identities lose importance."[1] But, according to Janna Koretz's article, enmeshment doesn't need to be only between people—it also happens between people and their jobs.

AppSec is a high-pressure, fast-paced domain. We measure ourselves by the amount of influence we yield, the criticality of the findings we identify and solve, and the fast time responses we strive for when people need us. It is

1 Janna Koretz, "What Happens When Your Career Becomes Your Whole Identity," *Harvard Business Review*, December 26, 2019.

easy to lose ourselves in the urgency of our processes and the responsibility of our defense roles. We guard the crown jewels of the realm.

This, together with the already high stress of the technology world at large, the need to constantly keep pace with new developments, and the inherent context-switching we are under, can certainly lead to burnout and, definitely, to enmeshment. An often cited stat is that there is 1 AppSec engineer per 100 developers, but "your mileage may vary."

If you describe yourself in terms of your work ("I am an application security person!"), find yourself thinking all the time about work (to the point of waking up with a solution to the problem you went to sleep thinking about), become anxious about your capabilities and talents when you know you have them, find it hard to spend time and energy on anything not related to your job or to AppSec, or feel lost at the thought of not being able to do your job anymore—you're dealing with professional enmeshment.

I am not going to embark on a list of tools and practices that can help here. After all, "I am an application security person," not a mental health person. But I will suggest you take proactive steps away from the path to enmeshment. We cannot be the best at what we do if we are not also at our personal, non-work-linked, best selves. If you have them, use mental health tools provided to you by your employer. Have a nonsecurity hobby (Brazilian jiu jitsu is ridiculously helpful there—nothing makes you more mindful and aware of the moment than someone else trying to separate you from your limbs). Find joy outside of the dopamine hit of finding that next vulnerability. Read something other than a Request for Comments (RFC) or a Top Ten list. Dive into the pool and not into source code.

Put yourself first, so you can be of better service to others.

TAP Into the Potential of a Great SSDLC Program with Automation

Jyothi Charyulu

A successful *secure software development life cycle* (SSDLC) depends on having intentional thought, taking action, and persevering. Remember TAP: think, act, and persevere for success.

Think

First, create a vision and strategic multiyear road maps. Your vision defines *why* you do something.

Own the mission, strategy, product life cycle, and value creation for SSDLC and enterprise teams. Create a strong *why* based on industry trends, business drivers, threat landscape, and enterprise architecture and systems. Here are the top three points to consider when defining your vision:

Gather customer feedback.
: Ask what works, what behaviors are encouraged, and what the biggest bottlenecks are. Digest the data, analyze it, and create a customized gap analysis based on your company's threat landscape and inventory.

What are the drivers?
: Business cases help show why your project is worth the company's or client's time, money, and resources.

Think about platform engineering, internal pipelines, and automation.
: How easy are these to use, configure, scale, and deploy? How can the vision/strategic road map drive down the overall risk of the firm by proactively scanning, identifying, mitigating, and remediating security vulnerabilities across all the applications?

Act

Second, act on your vision and turn it into a workable plan. This could mean implementing automation, threat detection, vulnerability management, a platform engineering model, a road map for scaling across the enterprise, or integrating security within CI/CD pipelines.

Here are the top three points to act on your plan:

- Consider using a payoff matrix. Conduct a cost-benefit analysis that will help determine the necessity of a robust AppSec program. What percentage of your budget is allocated back to AppSec? What is the charge-back model? Who pays for the AppSec program? Does each business unit get a percentage of costs based on usage or scope of high-risk applications?
- Involve automation at all stages of your pipelines. For example, this includes automation of internal platforms, infrastructure, testing coverage, cloud or on-prem deployment, and use of patterns or provisioning. Consider how your SSDLC action plan will help developers prevent security vulnerabilities in their applications.
- For example, you can integrate training, security guidance (requirements), scanning automation, continuous integration (CI) automation, and vulnerability management that include open source software flaws.

In addition, think about your source code repository system. Is it possible to integrate security at this level—really, to the left of the pipeline? Can you automate your build tools? Assess your testing coverage, including test cycle time, velocity, and customer satisfaction. Integrate packaging code artifacts within pipelines. All CI steps should be automated and integrated with the development pipeline.

Persevere

Third, persevere against challenges such as having a lack of time, standardization, and technical skills within your teams. Remember, you are stronger than these challenges, and you have a secret weapon: automation!

Here are the top three points to consider when persevering against challenges:

- A product mindset that correlates with the usage of platform engineering will help address the challenges faced in SSDLC.

- Automation will help conquer challenges dealing with organizational mindset, culture, manual integration, lack of standardization, and lack of time.
- Standardization, consistent security training, automation integrated within pipelines, and vulnerability management all assist in dealing with challenges. Having consistent communication and easy-to-use road maps that clearly show the prioritized backlogs and deliverables, including incremental implementation of product releases without tight coupling of application architecture, help persevere against challenges as well.

This is how you use TAP for a successful SSDLC program: *think*, *act*, *persevere*.

Vulnerability Researcher to Software Developer: The Other Side of the Coin

Larry W. Cashdollar

I've been finding vulnerabilities in software since 1999. I've reported over 300 vulnerabilities in that time, ranging from format string vulnerabilities to cross-site scripting. I also started developing a web server in C around 1994. The web server has been tweaked, patched, and modified over the years since I started using it to serve traffic to my website in 2002.

My site only served static content, as the HTTP server I developed was much too simple to provide dynamic content. In 2010, one of the top vulnerability and bug curators in the infosec community mentioned that my site should be searchable, and that he'd had trouble finding vulnerabilities I had published while he was building his database. So I added the ability to execute PHP code and put all of my advisories in a MySQL database.

In April 2013, I had the opportunity to put my website behind Akamai's massive content delivery network (CDN) and enlist some of our security products to help my site. As time passed, I ran into bugs when fixing the code or adding features.

I eventually ran my code through Valgrind and fixed various null pointer dereferences, memory corruption issues, and leaks. I mentioned my Valgrind adventures to my friend, who is a seasoned C developer. He works as a penetration tester and participates routinely in Capture the Flag (CTF) challenges. I decided to email him a copy of my HTTP server code named Bunyip. This was the first time anyone other than myself had seen the ball of C code I had been tempering over the years.

After a few days, my friend returned saying he had found some buffer overflows and a path traversal attack. I'm now shocked and embarrassed as to where I went wrong. Then I remembered disabling the attack checking code when testing my server with Akamai's CDN and Kona WAF. I knew I made a major mistake with that code being disabled.

That wasn't the worst of it; in my code, I had been checking that the executable bit was set on files for execution. If that bit was set, then the `execve()` function was called on the file requested. This, coupled with the path traversal, meant that my friend was able to get remote command execution on my server via my HTTPd process. He could execute commands as www-data, the user that the HTTPd server dropped its root privileges to.

I've made many changes to the code, running it through Valgrind and auditing all of the functions of the system and my own to ensure I'm checking return values correctly. I no longer check the executable bit, but instead check the file extension. The code now executes cleanly and I can build it with compiler optimizations set (-O2), which would have caused a crash before due to the memory corruption issues I had. I've been running the code on my production web server for years now with no issues whatsoever.

It's easy to lose sight of all the important things in a programming project when you're focused on getting a specific task completed. In my situation, I was working on stability and functionality and lost the hacker mindset along the way. How can I break this? How can this be abused? If I were looking to get this to behave differently how would I do it? If I had taken a step back and looked at my code from a hacker's point of view like I had when I started writing it, I think I could have prevented most of my friend's attacks.

So, the moral of the story is that if you're developing software, try to take a step back and examine your code from an adversary's perspective.

Strategies for Adding Security Rituals to an Existing SDLC

Laura Bell Main

From time to time, we stumble upon a new approach, ritual, process, or tool that can potentially improve our software's cybersecurity and can be embedded into our software development lifecycle (SDLC). Of course, we want to try it.

So how can you successfully implement new processes, tools, or rituals in an existing SDLC? Here are some lessons I have learned over the years that I would like to share.

You Can't Change What You Don't Understand

This probably seems obvious, but if you are hoping to influence or change something, it's a great idea to understand it first. If you aren't actively part of the software team or using these processes day to day, now is the time to learn.

You can start understanding the following:

- What tools and processes are currently in this SDLC?
- Who runs them? How much time does it take?
- What is an acceptable completion time for each phase? This is especially important for tools embedded into deployment pipelines.
- What is currently working well?
- What is hurting the team, and what would they change if they could?

This process is valuable, not only for you as you prepare to weave cybersecurity through it, but also for your relationships with your engineering

teams. Security is about collaboration, which starts with understanding and empathy.

Start with Experiments, Not Solutions

The rollout of a new process, tool, or ritual must start as an experiment, as you run the minimum viable product (MVP). By doing this, we encourage *fail fast* and create room for the engineering team to provide feedback and assess it from their perspective. It also gives you time as a leader to understand if this is the right approach.

Like any experiment, you need to start with a hypothesis and some criteria that you can measure. Additionally, you will need to understand what success and failure would look like.

For example, an experiment for rolling out a source code review tool into the CI/CD pipeline would need the following:

- Hypothesis. Using a source code review tool in the CI/CD pipeline would:
  - Allow the team to review all source code on commit.
  - Identify cybersecurity vulnerabilities in our specific language set and technology stack.
  - Reduce the time taken to check code for cybersecurity issues.
- Criteria:
  - *Run time*. The tool must run in less than 10 minutes so that the deployment pipeline is not compromised.
  - *Run frequency.* The tool can run on every commit and be triggered from our existing tools.
  - *Output*. The tool output can be automatically raised with engineers and also recorded in our ticketing or issue-tracking system.
  - *Exceptions*. The tool can be configured to allow exceptions that are specific to our environment.

Experiments will typically run for between one and three months to allow for testing over a sustained period and a wide range of development milestones. In the best case, experiments include brand-new code, projects, and legacy systems to ensure the range of the capability is understood.

Create a Rollout Plan with the Engineering Team

If the experiments succeed and you (and the team) are satisfied with the outcomes, it's time to roll out.

Rather than just turning things on and calling it a job well done, consider the following:

Training
: Ensure this links to clear explanations of the tool's purpose, outcomes, and impacts.

Support
: Have a plan for what happens if things change, if there are issues, or if the team needs help. If people find it is no longer working and you don't have a clear support system, the process or tool will be removed or avoided.

Review
: Have an annual process for reviewing the effectiveness of the tools and processes you have in place.

Collaboration Is the Key

No matter how big or mature your team is, introducing new elements to an existing SDLC can take time and effort. Remember, however, you can make these changes together by collaborating with your engineering team and focusing on the purpose, outcome, and impacts of the proposed changes.

Challenges and Considerations for Securing Serverless Applications

Manasés Jesús

With the recent cloud computing advancement, serverless applications came to light and became more and more attractive. At its core, serverless computing is a model of cloud computing where the cloud provider manages the infrastructure and automatically provisions and scales the resources as required. This means that developers can focus on writing code without having to worry about managing servers or infrastructure.

Security in serverless applications is a topic of great importance in the modern digital landscape. The rise of serverless computing has brought about a new set of challenges when it comes to security. In this essay, we will explore the key considerations for securing serverless applications.

One of the biggest challenges is the fact that serverless applications are event driven, which means that they are triggered by events such as user requests, database updates, or file uploads. This makes it difficult to predict when and where the application will be executed, which can make it harder to implement security measures.

Another challenge is the fact that serverless applications are composed of multiple functions that are executed independently and asynchronously. This means that each function needs to be secured individually, which can be time-consuming and complex.

Because it is a multitenant cloud service model, serverless computing is susceptible to security risks that can be categorized into five groups. External attacks on applications from malicious users are part of the first group, such as injection and cross-site scripting attacks. The second group consists of applications attacked internally by insiders, such as sniffer attacks. For

instance, in an ehealth application, a distinct series of fired functions may represent a particular patient's health status. The last three groups are horizontal attacks between tenants, vertical attacks on serverless infrastructures from malicious tenants, and vertical attacks on applications from malicious platforms.

So, what are the key considerations for securing serverless applications? Well, it is important to implement several strong mechanisms. In a nutshell:

Authentication and authorization mechanisms
: Users should be required to authenticate themselves before they can access the application, and access should be restricted to only those users who have the necessary permissions.

Encryption mechanisms
: All data should be encrypted both at rest and in transit. This can be achieved through the use of SSL/TLS certificates, encryption algorithms, and secure key management practices.

Access controls
: Access to the application should be restricted to only those users who have the necessary permissions, and those permissions should be reviewed regularly to ensure that they are still appropriate.

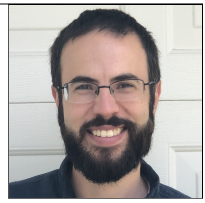
Monitoring and logging mechanisms
: All events should be logged and monitored for suspicious activity, and alerts should be triggered when suspicious activity is detected.

Testing and validation mechanisms
: All code should be thoroughly tested and validated to ensure that it is free from vulnerabilities and meets the necessary security requirements.

Securing serverless applications is a complex and challenging task, but it is essential in order to protect against the growing threat of cyberattacks. Whether you are a developer, a security professional, or a business owner, it is paramount to take the necessary steps to secure your serverless applications and protect your business from cyberattacks.

Using Offensive Security to Defend Your Application

Nathaniel Shere



Developers are often at a disadvantage when it comes to defending their applications from attackers. But the reason is simple: their focus is different.

Developers focus on functionality, performance, and ease of use for customers, while hackers focus on vulnerabilities, data exfiltration, and how users can be manipulated. This disconnect in focus can lead, through no direct fault of the developers, to security issues that attackers regularly target and exploit.

The key to giving the advantage back to developers is to *think like attackers* when designing, implementing, and testing application features. To highlight just a few examples of this type of thinking, let's look at various features that are common in most applications.

Helpful Response Messages

Because developers attempt to assist users as much as possible, they will often add helpful response messages based on the user's input. One example of this is when a user forgets their password and the application tells the user whether or not the submitted email address is valid.

Unfortunately, these helpful responses also assist an attacker in identifying valid accounts within the application, a prerequisite to performing password and other authentication attacks. So, in the end, the feature that helps users troubleshoot their own issues also helps attackers learn more about the application and how to attack it.

API Endpoints

Developers often fall into the trap of believing that API endpoints are more secure than standard web pages because they are nearly invisible to standard users. These endpoints have gained popularity with the advent of jQuery and advanced JavaScript frameworks that handle all of the application interface rendering. However, attackers know to use proxy tools such as Burp Suite or Postman to see those backend requests that go unnoticed by others.

Developers who understand this attack vector, though, can plan accordingly and ensure that appropriate security controls and features are implemented at those endpoints.

Administrative Features

A lot of trust is placed in administrator users. They are usually given the most features, the most access, and the most ability to destroy everything. As a result, developers often assume that only trusted individuals will ever be administrators and, thus, administrator features aren't as vulnerable as lower privileged ones.

Hackers are always interested in advanced features, though, and many types of attacks have been developed to target them. From cross-site request forgery (CSRF) and clickjacking attacks where an active administrator is tricked into taking an action they didn't intend, to outright phishing and social engineering. If a cross-site scripting vulnerability is discovered elsewhere in the application, it can also often be used to target administrators who visit the infected page. Finally, hackers are experts at finding and analyzing administrative functionality, even from lower-privileged user accounts, by enumerating endpoints, studying JavaScript code, and reviewing help desk and developer documentation.

As with the other examples, developers who understand the types of attacks that can target administrators are better prepared to defend against those attacks, without necessarily limiting the ability of administrators to manage the application. Using appropriate request headers and application configurations mitigates the risk of CSRF and clickjacking attacks, for example, and ensuring that administrative functionality is appropriately protected against privilege escalation attacks can limit potential exposure to lower privileged users.

The examples can go on and on, but they all demonstrate the value of thinking like an attacker so that application features can be designed with these attacks in mind, appropriate defenses can then be implemented during

development, and relevant attacks can actually be tested against the application before deployment.

Beyond "No": The Modern Paradigm of Developer-Centric Application Security

Nielet D'mello

In the rapidly evolving landscape of modern software development, AppSec engineers like me, find ourselves operating in an environment of high agility and velocity. Securing things at scale and pace where security debt can potentially loom large as vulnerabilities and defects find their way into the product pipeline is a reality to account for. Naturally, AppSec professionals need to redefine their approach to working with engineering teams. This essay delves into the concept of *maximum yesness*.[1] *Yesness* refers to having a willingness or desire to succeed; to remove the boundaries that would ultimately cause you to fail. Applying this concept, and maximizing it to its full potential, is an approach that bridges the gap between security and development, fostering an environment where security becomes an enabler rather than an impediment. Maximum yesness signifies not just a willingness to say yes but doing so strategically and optimally by creating an environment where measures are not merely restrictive but contribute positively to the development process, ultimately fostering a culture of innovation and success.

Traditional security operates with the opposite paradigm. It defaults to saying no to ensure protection. As a result, security becomes a harsh, negative thing. It becomes an impediment and a gatekeeping practice.

1 *Maximum yesness* embodies the role of an AppSec engineer as an ally to developers, fostering collaboration and transformation through a holistic approach that integrates security into every facet of modern software engineering.

Now, what if you adopt the opposite approach to thinking about when to say *yes*? This looks at what is acceptable, rather than what is prohibited. Although it ultimately achieves the same thing, it has a positive connotation that puts a new spin on security culture. It depends on having a streamlined intersection of technology, people, and processes.

These days, with the pace of change, security teams have to lead their work with a focus on an acceptable risk posture, leveraging a holistic approach through best practices guidance, developer-centric technology, people, and processes in the development life cycle.

Crucial to this transformation is the selection of tools and security programs, which play a pivotal role in shaping a culture where application security acts as an enabler (the "yes" factor). Aligning with engineering teams' aspirations for seamless operations, AppSec teams must construct reliable solutions.

For instance, transitioning from conventional vulnerability management, characterized by scans leading to an abundance of remediation tickets, to developing or acquiring products that empower developers through preset safety mechanisms, forms a critical strategy. By creating *security paved roads*, organizations can substantially mitigate application risks and enhance overall security.

Consider the scenario of developers crafting infrastructure as code (IaC) to generate resources. Instead of grappling with securing resources, the approach shifts to providing easily customizable and secure modules, simplifying the process. Furthermore, the integration of SAST/SCA scanning into CI/CD pipelines, operating in both nonblocking and blocking modes to guide developers during pull requests, empowers developers while maintaining security standards.

Additionally, a successful approach is the introduction of an engineering-focused Security Champions program. This program, thoughtfully structured with defined roles and responsibilities, offers tailored threat modeling training for internal technology stacks, guidance on vulnerability management, and more. Empowered engineers can conduct security design reviews, perform secure code reviews, and address vulnerabilities' remediation, fostering trust and organic risk management.

The key to the success of the application security program is a transformational shift in communication and engagement. In essence, maximum yesness is not synonymous with granting approval to every request from the AppSec team. Rather, it involves reshaping communications and engagements through careful inclusion of caveats and considerations, with a

resolute emphasis on constructing security paved roads. You can start by aligning with the engineering team on common goals, understanding their priorities, and supporting their objectives. This helps AppSec professionals provide a win-win solution, instead of throwing a no as an answer to their solutions. The shift in mindset will naturally shift your communication and engagement model.

By incorporating AppSec at the design phase, there's a better understanding of the problem statement, a shared language of the business goals, and a lower cost of securing the solution much earlier in the SSDLC. After all, outcomes are directly related to the processes that support it.

Security Paved Roads

Nielet D'mello

If you don't know where you are going, any road will get you there.
—Lewis Carroll

In modern software companies, there are usually centralized platform teams and various product teams. The challenge that many security teams face is to empower developers to ship or develop things as quickly and efficiently as possible while maintaining an appropriate level of security that minimizes the business risk.

As AppSec engineers, we embrace the philosophy of *paved roads* to introduce well-supported and smooth security controls that are automated and integrated across the SDLC.

What Are Security Paved Roads?

A concept first popularized by Netflix, security paved roads involve building software, libraries, tools, and processes (very close to the developer's workflow) to ensure that developers can build secure things by default. The goal is to make security as transparent as possible and as easy as possible for developers—not to make security a roadblock for them to adopt. An ideal paved road would allow engineers to be fully autonomous in designing, building, and deploying with little to no bottlenecks from security teams because the security baseline requirements are already baked in.

When it comes to AppSec reviews and vulnerability remediations, the paved roads support developer autonomy and accelerate velocity because mitigations can be accounted for centrally via self-serve guidance. This helps engineers focus on delivering their core business value and reduces friction with security teams. In a nutshell, it means aiming to make the quickest path to production the most secure.

Some common examples I've seen or contributed to in terms of paved roads are:

Continuous integration and delivery/deployment (CI/CD)
: Coupled with a continuous integration pipeline to provide mechanisms for automated releases to environments with capabilities like rollouts, rollbacks, and configurations as code that avoid drifts

Software life cycle management
: Providing secure, hardened and up-to-date container base images; a common internal store for retrieving third-party dependencies, a software bill of materials (SBOM)

Authentication
: Centralized mechanisms incorporating authentication standards, protocols, rate limiting, monitoring, and auditing for web applications and APIs

Secret storage
: Secure storage and granular access control mechanisms for secrets and credentials across the infrastructure and applications

Similar security paved roads can be built for access control, automated threat modeling, vulnerability scanning tools, continuous monitoring and response, compliance and regulatory considerations, and so on.

How to Decide What Security Paved Roads Are Needed?

AppSec engineers should begin by identifying common insecure usage patterns and vulnerabilities across teams in the applications and services they are reviewing. They should also consider the business risk profile for the applications and other factors, such as exposure to the internet, business criticality, types of users, types of data handled, and so on.

Additionally, focusing on large strategic initiatives (e.g., partnering with a central platform team responsible for authentication, access, and identity in building these components) yields a huge return on investment.

Adoption and Effectiveness

The effectiveness of paved roads is hugely dependent on adoption. Some ideas that I've found work well are sharing/advertising in technical circles or

brown-bag talks, evaluating outreach efforts for different teams and organizations, and improving guidance around self-service use of these controls.

Product-Centric Approach and Feedback Loops

A product-centric approach is crucial to adopting paved roads as it helps to build the solutions by scoping the problem, outlining use cases, focusing on the developer experience, gathering feedback, and measuring success.

With feedback loops, these can be either balancing (by providing easy-to-use paved roads and minimizing the probability of insecure software and reducing security incidents) or reinforcing loops (by building easy-to-use, smooth paved roads that accelerate velocity, making the safe path the default path).

Conclusion

Overall, security paved roads provide a framework for integrating security into the software development process, enabling developers to build secure applications efficiently. By leveraging security paved roads properly, we as AppSec engineers can further enhance our approach to application security and effectively mitigate risks.

AppSec in the Cloud Era

Sandeep Kumar Singh

Over the past few years, there has been significant growth and adoption of cloud-based applications. The shift to cloud computing has been driven by its many benefits, including scalability, agility, cost savings, and global accessibility. This adoption has also brought about significant changes in application security. Traditionally, AppSec focuses on protecting applications within an organization's premises. However, the move to cloud computing has pushed applications and their associated data outside the organization's perimeter, leading to new security challenges and considerations. As an AppSec professional, learning the cloud challenges and adopting new controls is a must for protecting applications in the cloud.

Learn Shared Responsibility Model

One of the fundamental changes that cloud computing brings to application security is the *shared responsibility model*. In traditional on premises environments, organizations had full control over the security of their infrastructure and applications. With cloud computing, the responsibility for securing the underlying infrastructure shifts to the cloud provider, while the organization is responsible for securing applications and data through access controls, authentication, secure coding, and managing third-party risks.

It's important to note that the exact division of responsibilities can vary depending on the cloud service model being used (IaaS, PaaS, SaaS) and the specific offerings of the cloud provider. Each model presents its unique security challenges. For example, in the infrastructure as a service (IaaS) model, where organizations have more control over the infrastructure, they need to ensure proper configuration and hardening of virtual machines, network security groups, and storage accounts. On the other hand, in the platform as a service (PaaS) and software as a service (SaaS) models, organizations rely on the cloud provider for many underlying security controls, such as securing the platform or application stack.

Secure Configurations

Cloud service providers typically provide a secure baseline configuration for their services, but it is up to the organization to customize and optimize those configurations to align with their specific security requirements. This includes configuring access controls, network settings, storage configurations, encryption, logging and monitoring, and other security-related settings. Cloud providers often offer documentation, best practice guides, and security recommendations to assist organizations in securely configuring their services.

Continuous Logging and Monitoring

Another significant impact of cloud computing on application security is the need for continuous logging and monitoring. Implementing comprehensive logging mechanisms enables organizations to track user access, data transfers, configuration changes, and security incidents within their cloud infrastructure. Real-time monitoring of logs facilitates prompt detection and response to threats and suspicious activities, while the analysis of logged data provides insights into system performance and vulnerabilities. Additionally, logging and monitoring assist in compliance by providing an audit trail of activities.

Data Protection in Multitenant Environments

Risk mitigation in multitenant environments is also crucial for organizations to protect their data and maintain privacy. To mitigate risks, organizations should focus on strategies such as data isolation, encryption, access control and authentication, vulnerability management, security monitoring and logging, SLAs, contractual agreements, and regular security assessments. Ensuring the logical separation of data, implementing robust encryption, and adhering to strict access control practices safeguard against unauthorized access to sensitive data.

Adopt Cloud Security Services

Additionally, cloud security services and tools provide essential measures to enhance application security. Secure WAFs protect against common web vulnerabilities, while cloud access security brokers (CASBs) offer control and visibility over cloud data and applications. CASBs enhance security with data encryption, access control, and threat prevention. They tackle shadow IT, enforce data loss prevention (DLP) policies, and ensure compliance with industry regulations.

Container security tools ensure the integrity of containerized applications, and identity and access management (IAM) solutions manage user access to cloud resources. Encryption and tokenization tools protect sensitive data, while SIEM solutions collect and analyze security event data. By leveraging these services and tools, organizations can implement robust security controls and detect and respond to threats in real time.

Conclusion

Application security is crucial in the cloud era to protect sensitive data and defend against cyber threats. Cloud computing offers numerous benefits but also introduces unique security challenges. By implementing these security measures, organizations can mitigate risks and enhance AppSec in the cloud. With a strong focus on AppSec and the adoption of robust security practices, organizations can confidently embrace the cloud and leverage its benefits while maintaining the integrity, confidentiality, and availability of their applications and data.

Code Provenance for DevSecOps

Yashvier Kosaraju

The term *provenance* alludes to the original source or historical lineage of ownership. When approached from the realm of application security, the concept of *code provenance* or *code ownership* emerges as the foremost challenge that every AppSec team must confront before entering the captivating realm of DevSecOps automation and advanced tooling. This prelude is crucial, as it entails discerning the appropriate individuals to designate or alert when AppSec tools unearth vulnerabilities.

Picture this: you find a deadly SQLi (or pick your favorite bug), in a piece of code through the various practices in your AppSec program (internal code review, external security review, etc.). How do you go from here to figuring out if this code is deployed into production, which team owns it, who needs to fix it, and who it needs to be escalated to if the need arises? Now repeat this for *every* code bug you find in your code. You are going to spend the majority of your time finding owners rather than finding issues to fix or reviewing designs.

The solution to this is to figure out a way to have ownership information of code (and other systems) either in a separate system with APIs or by throwing a simple file into every code repo that has the information you need. Once you have this information, you can quickly go from a bug to alerting the relevant team to fix things, reducing the time from "finding to fixing," which also helps reduce the time for which your assets are at risk. You can also build automation on top of your tools that provides high-confidence results (i.e., file tickets for the issues and assign them to the owner based on the system that maintains ownership).

There are multiple solutions to help you accomplish this. Consider Spotify's centralized software catalog, the innovative approach by Twilio through the introduction of an *about.yaml* file adorning each repository, replete with pertinent insights, or enterprise AppSecOps tools which also help you maintain

ownership information. Regardless of the chosen modus operandi, one must possess a firm grasp on which team commands dominion over a specific code fragment, the product to which it contributes, and its stage of development, be it within the crucible of development, staging, or the grand theater of production.

Wait, why not use CODEOWNERS you ask?

The problem I have had with CODEOWNERS is that I cannot tell you which team owns the code, what feature it deploys to, what stage of development it is in, which product group or business unit in your company it belongs to, and any other information you might want.

Once you start down the path of figuring out code ownership, and maintaining this for future code in your organization, there are a few roadblocks you will face, two of the most common ones being:

Shared ownership
: There is a lot of code in companies that have shared ownership, aka, no clear owner. This is a problem because when stuff hits the fan, you need to know who is responsible. (This is similar to asking if five people own a car. Who pays when the car needs to be taken to the shop?)

Legacy code
: This is another common problem where the code has been written and deployed by folks who are no longer at the organization. (This is similar to you driving a car but not knowing whose it is, except that it belongs to someone in your family.)

In conclusion, effectively managing code ownership intricacies is crucial for successful software development and maintenance. Solutions such as Spotify's Backstage, Twilio's *about.yaml* approach, and enterprise AppSecOps tools provide valuable means to establish and handle ownership information. While CODEOWNERS is a widely recognized tool, its limitations in offering comprehensive insights into code ownership, deployment features, and development stages highlight the significance of exploring alternative strategies.

As you embark on the journey of unraveling code ownership complexities, you may encounter challenges such as shared ownership and legacy code. The former poses difficulties in pinpointing responsibility during critical situations, while the latter involves managing code left behind by individuals no longer with the organization. Addressing these challenges is crucial for maintaining transparency, accountability, and efficiency in the software

development life cycle. By overcoming these roadblocks, organizations can cultivate a culture of clarity and responsibility—ensuring smoother collaboration and more effective code management.

PART III

Data Security & Privacy

Will Passwordless Authentication Save Your Application?

Aldo Salas

Web applications can use a number of different options when it comes to authentication schemes. Without a doubt, and historically, usernames and passwords have been the preferred method of authentication for the last few decades.

The shortcomings of password-based authentication have been discussed at length, and almost every week we keep hearing of a new breach that involved or resulted in a password compromise.

Passwordless and WebAuthn

Thankfully, we now have stronger authentication mechanisms, such as passwordless systems. Specifically for web applications, we now have the Web Authentication (WebAuthn) specification that uses public key cryptography in order to authenticate any given user to a web application. The Fast Identity Online (FIDO) Alliance is the association behind this specification.

WebAuthn allows consumers to use strong authenticators such as security keys, biometrics, and mobile devices to prove their identity. Essentially, WebAuthn is a set of browser-based APIs that allow web applications to authenticate users by using the aforementioned authenticators.

Most modern devices have at least one of these authenticators already built in. For instance, you can use biometrics for mobile devices (fingerprint, face recognition, etc.), Touch ID for MacOS, and Windows Hello on Microsoft devices, to name a few.

These authenticators are phishing-resistant, which means that even if a bad actor somehow manages to trick users into providing their login information and, for instance, the user tries to use a security key to authenticate to a fake

website, this will not work since the FIDO authenticators are tied to a specific domain or website. These websites are referred to as relying parties (RPs) under the FIDO specification, and a web application can have multiple RPs.

Passwordless Pros and Cons

This sounds amazing, and it would appear that this is the solution to all of our authentication problems. Just think about it: when using WebAuthn, it's impossible for an attacker to conduct any of the following attacks:

- Brute force
- Phishing
- Password spray
- Dictionary attacks

Furthermore, it wouldn't make sense if an attacker wanted to target one specific application, since the web application only stores the public keys to authenticate users—which is meaningless by itself. An attacker would be forced to target users one by one and try to compromise their devices one at a time in order to try and extract the private key needed to authenticate that specific user to a specific website. As you can imagine, this is not cost-effective for bad actors.

However, WebAuthn is not something that can be leveraged and deployed instantly. For instance, think of an application that has 10 million users, and the company decides they want to start requiring physical security keys for everyone. This represents a challenge since not everybody would be willing or able to procure a security key. And let's not forget the cost of acquiring such devices.

Implementation challenges aside, it's really important for any organization that is considering using a passwordless (WebAuthn) solution to be aware of the risks that this paradigm involves.

Passwordless Vulnerabilities

The main concern when implementing WebAuthn is making sure the web application is performing all the authorization and authentication checks at every single endpoint and preventing tampering in every single parameter.

If a web application using passwordless authentication does not perform the proper server-side checks, this can lead to serious vulnerabilities such as:

- Account takeover
- Privilege escalation
- Username impersonation
- Unauthenticated access
- Insecure direct object references (IDORs)
- Etc.

At the end of the day, we have to remember that web applications using WebAuthn, or any other passwordless solution, are still susceptible to all the web vulnerabilities we are already familiar with.

Other Recommendations

Passwordless authentication is powerful, but it won't save your application by itself. It is important to keep in mind other crucial AppSec components, such as SSDLC, automated and manual testing, application firewalls, and others.

Securing Your Databases: The Importance of Proper Access Controls and Audits

Dave Stokes

Databases are the core of every project or product and are a critical piece that every AppSec professional should understand and pay attention to. However, very rarely do you hear of security steps taken to secure that data. Usually, the databases are firewalled away from the general world, which limits vulnerabilities from outside exploitation. This means much of the ability to adversely affect the data will come from insiders.

Like securing applications, securing databases starts with the basic security requirements, especially on access management. Databases, like their underlying operating systems, have usernames and, hopefully, passwords. Most databases no longer support "anonymous" accounts where there is neither a username nor a password. So what is needed are good passwords and restrictions on account access to schemas. It is unsurprising to find an organization with several projects using the same server for multiple separate schemas with the same account name and password. And many times, that will sadly be the root account.

It should be obvious that using the root account for all database access is a potential ticking time bomb waiting for the wrong delete or a data breach. However, this is unfortunately a common occurrence. Restricting root account access is always countered with claims of stifled creativity by those who do not have it. Root access should be reserved for administrative functions only. The lower the number of persons with the ability to perform potentially catastrophic actions the better.

Separating access to individual schemas by unique usernames and passwords contains that data to a known group. This greatly reduces the possibility of

confusion where Project A does function X while Project B does function Y and mistakenly uses the wrong function. It is even advisable to have separate accounts for reading only and for writing only—especially in cases where it is imperative to have short-duration transactions. It is too easy to lock many rows and block others from access while doing the SQL equivalent of a scavenger hunt.

Auditing account permissions is a thankless task until after a painful incident. It is a common practice to copy permissions from a known senior developer to a junior when that junior probably does not need higher-level privileges. Most databases now have *roles* where you set the privileges needed for a job function and then users are assigned a role. So a lower-privileged role may have read, write, and delete permissions on a set of tables, but they may not have created schema or drop abilities. Being stingy with permissions may seem harsh unless you have had to restore large schemas from backup repeatedly.

Another part of auditing is removing accounts that are no longer needed. In most organizations, the closest thing to immortality is a database account that will survive well past the account user's tenure. This is a vulnerability waiting to happen. Most relational databases have a way to lock accounts in cases where a developer changes projects but may eventually return.

It helps to think of the login credentials as the last bastion of the defense of your database. If the credentials are used maliciously, then there is nothing between that bad actor and your data or what was your data. Do all the things you do with operating system credentials, such as password rotation schedules, complexity and length requirements, and decommissioning upon employee exit. These credentials are often the last line of defense of your data.

Of course, logging and monitoring activities in the database is a great detective control. Whether it's insider threats or external threats, the recommendation is to always implement a solid alerting and auditing capability for the databases. Protecting your application is ultimately protecting your data, which goes hand in hand with securing your database.

DataSecOps: Security in Data Products

Diogo Miyake

As the world is moving toward data-centric security, *data security operations* (DataSecOps) is an evolving approach that emphasizes integrating security throughout the data life cycle. DataSecOps involves collaboration between security teams, data scientists, and engineers to ensure that appropriate security is considered at all stages of the data life cycle, including data creation, storage, processing, sharing, and disposal. Similar to DevSecOps, this paradigm recognizes that all teams involved in data management and use must take responsibility for maintaining security.

Several components contribute to the implementation of DataSecOps. These components include:

Security operations center
: A security operations center (SOC) is a dedicated team or facility responsible for monitoring, detecting, and responding to security incidents in real time. The SOC uses various tools, technologies, and processes to identify and mitigate security threats, aiming to protect an organization's information assets and infrastructure.

DevSecOps
: DevSecOps is a software development approach that integrates security practices into the DevOps process. It emphasizes the collaboration between developers, operations teams, and security professionals throughout the SDLC. By incorporating security early on and automating security checks, DevSecOps aims to build secure and reliable systems. Application security through DevSecOps becomes the core of the DataSecOps.

Data privacy
: Data privacy concerns the protection of personal information and how it is collected, stored, processed, and shared. It focuses on ensuring that

individuals' personal data is handled securely and in compliance with applicable privacy laws and regulations. Organizations implement data privacy measures, such as data encryption, access controls, consent management, and data breach notification, to safeguard sensitive information and respect individual privacy rights.

Chaos engineering
: This practice involves deliberately introducing failures into a system to test its resilience and recovery capabilities, helping identify weaknesses and improve system design.

Data governance
: Data governance encompasses policies, procedures, roles, and responsibilities for managing data assets throughout their life cycle. It includes components such as data architecture, data stewardship, data quality, metadata management, and data security.

Data quality
: Data quality refers to the health of data at any stage in its life cycle. Various metrics, such as incident counts, response and resolution times, table uptime, and query performance, help measure and improve data quality.

Data classification
: Data must be categorized and labeled based on its sensitivity, importance, and regulatory compliance requirements for implementing robust security measures.

In summary, DataSecOps is a paradigm that integrates security into DataOps, ensuring that security is considered at every stage of the data life cycle. It involves collaboration between various teams and incorporates practices and tools from cybersecurity, big data, and product management to protect data and minimize security risks.

Data Security Code and Tests

Diogo Miyake

Securing data code and tests is essential to protect data, maintain data integrity, prevent cyberattacks, ensure business continuity, uphold trust, and comply with legal and regulatory standards. One way to do this is to secure data pipelines. The importance of securing data pipelines lies in some steps and processes that must be taken before simply trying to solve the problem. For example, making a security assessment in data pipelines helps to understand risks and vulnerabilities to mitigate them before deploying data pipelines.

A *data pipeline* is an automated method in which data processes are used to fetch, transform, or make data available, either via API, frameworks, or in-house created systems; the sources can be diverse, such as SQL, NoSQL, files, and videos.

To secure data pipelines, consider asking some questions before simply delivering a certain code that performs a certain operation. For example, if it performs data ingestion or data transformation, in order to deliver it is needed to check what the business needs, to deliver more quality and security. The following are examples of questions:

- What is the demand of the business area?
- Do we have a sensible default describing the best practices and processes for the technology area?
- Do we have an SOC area? If yes, how can we create a product that is in conformity with SOC processes?
- What regulatory rules do we need to follow?
- What is the deliverable? a data product? a dashboard? reports?

Based not only on these questions but also on what is to be delivered, let's orient the security of the data pipelines either to the ingest, transformation, aggregation, IaC codes, or even the CI/CD pipeline codes for creating a data

platform (these encompass codes for the infrastructure, identity and access management [IAM], and so on).

Besides being data driven, the data pipeline needs to be security driven.

There are several tools that help secure much of what needs to be protected, such as SCA, SAST, DAST, and interactive application security testing (IAST). Techniques like hardening (for the clusters, container images, and used machines), IAM, CI/CD, security as a code, policy as code, data quality tests, data contracts, unit tests, and integration tests also help to mitigate risks and improve the quality of the code.

Along with these techniques, it is important to pay attention to the General Data Protection Regulation (GDPR), a privacy and security law required in European countries, and local data protection laws to maintain PII. Something that is necessary to understand is that it is almost impossible to have a secure data project without governance because both technology and data governance will guide the team to use the best means to work and make decisions.

Remember that security is not a job only for the company's security team but for each developer, because protecting the client's data is the responsibility of everyone who has contact with it, whether in the creation of a system, application, or analytical or reporting dashboard.

To ensure the security of a data pipeline, it's crucial to confirm that the appropriate logs are being transmitted to the monitoring tool, eliminate any exposed credentials, adhere to legal requirements for handling PII, and employ all necessary security measures. This comprehensive approach may help prevent vulnerabilities that could be exploited and lead to incidents, potentially causing business disruptions and financial losses due to data breaches.

Data Security Starts with Good Governance

Lauren Maffeo

In some ways, walking through Vancouver Convention Centre in May 2023 felt like déjà vu: I had attended the same summit (The Linux Foundation Open Source Summit North America in the same location (beautiful British Columbia) five years earlier. Many of the same sights, projects, and faces graced my presence. Still, I saw a distinct difference from 2018, when there was little to no talk of open source's role in application security.

By 2023, security had its own track at the OS Summit, and a brand new project, the Open Source Security Foundation (OpenSSF), had a full day of programming along with a new home under the Linux Foundation's umbrella. The surge in recent breaches caused by insecure coding practices, lack of encryption, and inadequate access controls means that AppSec can no longer be ignored. Likewise, AppSec teams can't ignore the role of data governance in their efforts. I suspect that five years from now, data governance won't be swept under the AppSec rug like it is today.

Data governance—your strategy for the people, processes, and tools to manage big data at scale —can sound like a buzzkill. It's often conflated with legalese, or it is made the scapegoat for why teams can't innovate. The truth is much more optimistic: done well, data governance engages colleagues across silos to cocreate the standards for keeping your data safe. It gives subject matter experts ownership of data in their respective domains, and the autonomy to help define the standards for each domain.

This doesn't mean that your marketing director will start leading your two-factor authentication (2FA) strategy. It does mean that as your organization's most senior marketing lead, this colleague has the expertise to define marketing data, metadata, definitions, and hierarchies within databases. When it comes to security, your marketing director also helps you define access to the marketing data domain within your architecture.

Let's say that your chief technology officer adopts the data-as-a-product approach and asks your data team to start building data mesh architecture. This involves designing the architecture to support each predefined data domain as its own mini data lake with its own respective catalog. All domain data lakes integrate with a mesh catalog, which consuming apps connect with to receive the data they need.

During this design process, you'll work with your data stewards—the lead subject matter experts per data domain—to define levels of access for each data domain. In this case, colleagues are the core users of your data mesh—but not all colleagues need (or should have) the same levels of access. This could cause data privacy breaches by exposing protected data to those who shouldn't have it.

Part of data governance involves defining levels of access based on user needs. In this case, your marketing director might propose these levels of access based on which users will need to pull marketing data from the data mesh:

Level 1: Sub-Domain Marketing Leads
: Search engine optimization (SEO), content, brand, etc., can pull data within their respective subdomains and view data in other marketing subdomains.

Level 2: Superuser
: Heads of marketing at the director level and above can pull data from all subdomains and approve access requests from colleagues.

Level 3: Admin
: Data architects, colleagues in the CISO, and your own data team maintain the system and uphold your governance standards.

Although simplistic, this example shows how data governance involves everyone in application security. Too much data exists today for one team or colleague to manage it all.

Under data governance, AppSec can take a role-based approach to user access based on distinct user needs for specific data. It also improves data literacy by engaging colleagues across the organization, encouraging them to consider who should access the data in their domains. As an app development expert, your job is to keep data safe in the right hands. Data governance defines whose hands those should be.

Protect Sensitive Data in Modern Applications

Louisa Wang

In the application security world, other than security design, code scanning, and WAFs, there are other security controls that developers should know and apply as needed, such as encrypting data and managing crypto keys properly.

Learn Key Management

Key management involves the procedures and systems for the generation, exchange, storage, use, crypto-shredding (destruction), and replacement of encryption keys. Meticulous key management is vital, as it ensures the security of even the strongest encryption solutions. However, designing and implementing a *key management system* (KMS) can be complex at times. For example, you can pick a *hardware security module* (HSM), which is a physical device specifically built to securely store and manage encryption keys. Or you can choose Trusted Platform Module (TPM), which is the computer chip built into endpoints (e.g., laptops) that securely stores encryption keys and certificates. Knowing what they are and where to apply them is the key.

Additionally, learn the crypto ecosystem. Systems like TLS use fundamental elements such as ciphers, hashes, blockchaining modes, and key agreement protocols for security. Avoid creating your own cryptographic primitives; it's time-consuming, demanding of expertise, and crucial for correctness and effectiveness. Pay attention to key storage, application integration, and security integration, which are the core components of a crypto ecosystem.

Security Needs During the Data Life Cycle Vary

Data in transit is often mandated, using TLS, while encrypting data in use is complex and less common. Data at rest can be encrypted at storage or database layers, with storage layer encryption being the simplest and usually available through major cloud providers. Application layer encryption is a

fallback for cases where neither storage nor database layer encryption is possible.

Design and Implement a Combination of Technical and Administrative Controls

To enhance data security, combine technical and administrative controls. Use data encryption alongside measures such as network segmentation, access control, auditing, and secure SDLC. Application-level encryption adds extra security but comes at a cost, with trade-offs to consider. Avoid building your own key management solution; use established services or libraries. Effective communication between application owners and security architects is key for finding solutions that meet typical use cases and design patterns, fostering collaboration and strong security practices.

Insights and Security Recommendations

Follow these suggestions for the best results:

Prioritize data classification and threat modeling.
: Start by identifying your application's data classification and conducting thorough threat modeling before embarking on data encryption. This proactive approach helps tailor your security measures to effectively mitigate risks.

Leverage cloud native services.
: For applications hosted in the cloud, consider utilizing cloud native services. They offer built-in encryption support, seamless interoperability, straightforward implementation, and easy maintenance. This not only reduces costs but also enhances security.

Consult database vendors.
: Before implementing transparent database encryption (TDE) or database encryption, always reach out to your database vendors to confirm native supportability. This ensures you're making the most of existing solutions.

Balance secure design.
: Secure design is a delicate balance, where security is just one aspect among several. Application security professionals often navigate trade-offs between cost, performance, scalability, ease of use, and security, especially in data encryption design.

Keep data protection as your goal.

Keep in mind that the ultimate objective is not merely encryption, but robust data protection. Encryption is a means to this end, and it should align with broader security goals within a well-architected framework.

Leverage Data-Flow Analysis in Your Security Practices

Manuel Walder

In pen testing, it is a standard practice for testers to record and analyze the HTTP-based data flow of an application to gain an understanding of the application and its function. This is usually done using tools such as the open source Zed Attack Proxy (ZAP) or similar tools. The browser's traffic is intercepted and stored in the tool and is then available visually. However, the same methodology is rarely used in other security practices, although it can provide significant added value with little effort.

However, recording and visualizing the HTTP data flow across all functions of a web application has proven to be a value-add for us in many practices. In this context, a record of the data flow between the browser and the backend creates transparency and visualizes the real interaction with a function in a more understandable way. Let me share some examples.

In threat modeling sessions of already existing functions, the implementation of a function may significantly diverge from the developer's description or the function's documentation. Initiating threat modeling sessions with a recording of the data flow provides participants with an overview. This recording can serve as a foundational basis to create a data flow diagram showing the real implementation of a functionality and also showing the parameters that are transferred during the execution, as well as the business logic of a functionality. Quite often, I have seen that the developers themselves were surprised about the implementation of the function, and also that obvious business logic errors were already revealed at the beginning of the session.

Another example involves security code reviews. A human brain is simply not built to read complex programming code over a longer period of time without significantly decreasing its performance. Furthermore, if the code is

written by someone else or in a language that you don't usually develop in, you are more likely to experience rapid fatigue and miss vulnerabilities. A visual representation of the data flow between the browser and the backend can serve as a welcome addition in many code reviews, providing a complementary understanding of the logic and creating an additional perspective.

In addition, the data flow can help the brain focus longer on a function by visually assisting. For various types of vulnerabilities, an HTTP data flow dump from a tool like ZAP can be the missing piece of the puzzle that leads to the discovery of a vulnerability that would otherwise be overlooked.

Also, when working in the field of WAF security policy engineering, a data flow can be of enormous help. The use of a WAF has become common in many organizations. When writing a WAF security policy, data flow analysis has proven invaluable. After the entire application has been used via the browser and the data flow has been recorded in the tool, it is transparently visible how the application interacts with the backend. All required HTTP methods are clear, as well as which content types are used and which uniform resource identifiers (URIs) or endpoints are required. Also, the used parameter data types can be determined. This helps write a WAF security policy, which protects more efficiently against attacks and leads to fewer false positives on the WAF. As an additional benefit, it is not unusual, with some routine in the analysis of the data flow, that vulnerabilities and attack points are discovered, which can then be protected by an appropriate policy.

Recording and storing the HTTP data flow can be of great help in many security practices. It visualizes, simplifies otherwise complex and obscure flows, and uncovers weaknesses in systems that would otherwise be overlooked.

To get started with the practice, I recommend simply letting all web traffic flow through a ZAD to develop a feeling for the topic and to understand the protocol. Many application purposes then become apparent step by step.

Embracing a Practical Privacy Paradigm Shift in App Development

Maria Nichole Schwenger

In the era of digital transformation, data has emerged as the fundamental asset propelling contemporary societies and economies. The pivotal role of application development in data collection, storage, and utilization is undeniable. However, conventional perspectives on data privacy and security often tend to prioritize protection over innovation, impeding progress and advancement. DevSecOps and Agile methodologies advocate for a pioneering approach that harmonizes user data safeguarding with the increasing demands of an ever-evolving technological landscape.

The Paradox of Privacy and Innovation in Data Security

Historically, data privacy has been associated with the imposition of stringent measures and limitations to forestall data breaches and mitigate misuse. While well-intentioned, this mindset inadvertently creates a paradox: excessive privacy constraints stifle innovation by obstructing access to valuable data insights that could propel technological breakthroughs and augment user experiences. Today, reconciling data privacy with innovation presents a multifaceted challenge, necessitating a radical reimagining of traditional frameworks and ideologies. Federated learning, privacy-preserving frameworks like PySyft, and early-stage privacy impact assessments are key tools for building a balanced approach to data privacy.

Reconceptualizing Data Ownership

An innovative approach to data privacy and security in application development revolves around reconceptualizing data ownership. Departing from the conventional dichotomy of exclusive ownership vested in users or service

providers, a pioneering paradigm must advocate for shared data ownership between users and providers. This grants users control over their personal information, permitting them to specify access permissions to different applications. Simultaneously, service providers gain access to anonymized and aggregated data, enabling innovation and service improvements without compromising individual privacy. Emerging technologies like generative AI (Gen AI) lend support to this transformative initiative. Integrating attribute-based access control (ABAC) and self-sovereign identity (SSI) solutions, or using tools like DataWallet and Solid, may empower users to manage and share data attributes selectively. Data trusts may offer further control through independent governance.

Leveraging Privacy-Enhancing Technologies

Contrary to popular belief, innovation and privacy are not mutually exclusive; they can coexist in synergy. Privacy-enhancing technologies (PETs) offer a cutting-edge approach to application development, preserving user data privacy while still providing valuable insights to developers. Techniques like differential privacy, homomorphic encryption libraries (e.g., HElib or SEAL), and secure multiparty computation frameworks (e.g., SecureML) can safeguard sensitive information and enable data analysis for trends and patterns throughout data processing.

Transparency and Informed Consent

Data privacy concerns in application development stem from a dearth of transparency concerning data collection and usage. A progressive resolution to this issue entails the adoption of a policy of full transparency and informed consent by application developers. Forging trust requires replacing complex terms of service engagements with clear, concise explanations of data usage and explicitly seeking user consent. Such transparency fosters trust between users and organizations, cultivating a mutually beneficial relationship. Verifiable consent records on blockchains like Ethereum or user-centric consent management platforms like OneTrust alongside interactive dashboards can empower users in their data journey.

Data Minimization and Purpose Limitation

While historically gathering vast amounts of data for potential future uses was seen as the gold standard, the data minimization and purpose limitation principles offer a more secure and sustainable path. By collecting only essential data, exclusively for its intended purpose, and avoiding making copies

everywhere, developers mitigate the risk of data breaches and misuse, ensuring user privacy while effectively extracting valuable data insights. By using Gen AI to create synthetic test data, employing tools like ARx or Mondrian AI for data anonymization, storing data in dedicated solutions like SingleStore, and utilizing zero-knowledge proofs for user verification, we can ensure privacy-centric data practices.

Exploring Decentralized Data Storage

While centralized data storage has long been the conventional standard for data security, recent advancements in blockchain technology introduce a disruptive alternative—*decentralized data storage*. By distributing data across multiple nodes, blockchain-based systems prevent data breaches through a single point of attack. This empowers users, giving them greater control over their data and fostering a more secure and private application environment. Some innovative approaches include IPFS's peer-to-peer data sharing, Storj's distributed cloud infrastructure, and blockchain-enabled data ownership models from Ocean Protocol and Filecoin.

Data Privacy as a Competitive Advantage

Challenging traditional notions of data privacy and security demands an acknowledgment of privacy as a competitive advantage. Privacy-conscious consumers increasingly favor organizations that prioritize data privacy. Embracing "privacy by design" as a core value sets companies apart from competitors, attracting a more loyal user base and boosting brand reputation. For a competitive edge in the data-driven market, integrate privacy by design, leverage differential privacy frameworks (e.g., TensorFlow), adopt consent-driven personalization tools (e.g., Twilio's Segment), and conduct regular risk audits to identify vulnerabilities and demonstrate commitment to user data protection.

In a Nutshell

Shifting the landscape of data privacy and security in application development necessitates embracing an innovative, balanced approach that respects user privacy while nurturing innovation. Striking this equilibrium empowers developers to create applications that simultaneously uphold data privacy and drive progress in the digital era.

Quantum-Safe Encryption Algorithms

Rakesh Kulkarni

In the realm of application security, asymmetric encryption plays a pivotal role in securing applications through SSL/TLS, digital signatures, and secure key exchange—ensuring the integrity and confidentiality of data. The foundation of this security lies in the presumption that cryptographic keys are both random and resistant to guessing. However, the advent of quantum computing poses a booming threat, particularly to widely adopted asymmetric algorithms like RSA and elliptic curve cryptography (ECC). As quantum capabilities advance, security experts and AppSec professionals should proactively look for innovative solutions for quantum-resistant cryptography such as quantum random number generators (QRNGs).

QRNGs generate truly random numbers that are unpredictable and nonreproducible, leveraging the inherent uncertainty in quantum states. Incorporating QRNG in asymmetric key generation significantly bolsters the security against computational attacks, including threats from powerful quantum computers.

As many of the *Fortune* 500 companies are investing in building quantum computing capabilities. It is evident that we will be seeing exponential growth in quantum computers as well as the adoption of the technology. The concept of *decrypt later* quantum technology suggests that, once powerful enough, quantum computers could potentially break current cryptographic algorithms like RSA and ECC, compromising the security of data transmission and storage in applications.

Quantum computers, with their ability to perform complex calculations exponentially faster than classical computers, have the potential to solve these mathematical problems much more efficiently. As a result, they could render conventional cryptographic methods ineffective, leading to the decryption of sensitive information in applications that were previously considered secure.

The threat of quantum attacks on conventional cryptographic systems has led to the field of *post-quantum cryptography* (PQC), which focuses on developing quantum-resistant algorithms. Organizations and governments are investing in research and transitioning to quantum-safe cryptography standards to ensure data security in the era of quantum computing. Proactively addressing this threat is crucial to safeguarding sensitive data and ensuring the privacy of digital communication.

From an application security standpoint, integrating a QRNG with an existing public key infrastructure (PKI) can enhance key generation. This ensures that the cryptographic keys are truly random, a critical attribute for securing them against potential quantum computing attacks. Most importantly, this integration doesn't alter the fundamental PKI but revolutionizes the way keys are generated to make them secure and quantum-resistant.

The United States government, through NIST, has been conducting research and standardization processes to endorse quantum-safe cryptographic algorithms. The move is a proactive step to secure digital communications in the era of quantum computing.

As of May 31, 2023, NIST has chosen CRYSTALS-Kyber for general encryption due to its speed and small encryption keys, and CRYSTALS-Dilithium, Falcon, and SPHINCS+ for digital signatures in applications.

For example, in the case of CRYSTALS-Kyber, randomness plays a crucial role:

- During the key generation phase, the secret key components are generated randomly.
- In the encryption phase, a random matrix is used in the process to ensure that the same plaintext will not result in the same ciphertext when encrypted multiple times.
- The decryption phase, while deterministic, relies on the secret key that was generated with randomness.

Application Integration Security

Sausan Yazji

With the rising need for data democratization and data availability, sharing data between systems internally and externally becomes a definite requirement for the success of any business. Understanding and applying the controls needed to protect data is one of the key areas for application security and product security. Those controls should also comply with all data privacy laws and data security regulations, such as GDPR, PiplL, Brazilian General Data Protection Law (LGPD), FedRAMP, SOX, and more.

The following is a list of best practices that should be followed by application developers to reduce data security risks of application integration:

Data classification
: All data assets and data attributes should be classified based on their sensitivity and criticality following the classification guidelines provided by your organization. Data classification helps in determining the appropriate security measures to be applied to different types of data.

User persona
: All systems should have a user persona built to identify the right level of access to these systems. These personas should have a clear representation of all users, internal and external, including people and automatic system interfaces.

Encryption
: Strong encryption techniques should be implemented to protect data at rest, in use, and in transit.

Access control
: Define and implement access control processes to restrict unauthorized access to sensitive data. Map user personas to the appropriate level of access for each data set. Regularly review and update access privileges.

Authentication and authorization
: Per defined user persona, develop the correct level of access for each integrated system.

The following are application integration-specific rules:

Secure APIs
: All APIs should be designed with security in mind. Implement authentication and authorization mechanisms for each API access. Apply rate limiting and throttling to protect against API abuse and potential denial-of-service (DoS) attacks. Regularly update and patch API frameworks to address security vulnerabilities.

Secure communication
: Ensure secure communication between integrated applications by using encrypted channels such as HTTPS/TLS. Implement SSL/TLS certificates from trusted authorities to establish secure connections and protect against man-in-the-middle (MITM) attacks.

Data minimization
: Minimize the amount of data exchanged between applications to reduce the risk of data exposure. Only transmit or share data that is necessary for the integration process and avoid transferring sensitive data when it is not needed.

Auditing and monitoring
: Monitoring and auditing data logs is essential for data security. Review logs regularly to identify any anomalous behavior. Document integration points between systems on a quarterly basis. Remove integration points when they are not needed anymore. Monitor access logs, API usage, and system events to detect any suspicious activities or unauthorized access attempts. Identify and address any weaknesses or vulnerabilities.

Data retention and disposal
: Establish clear data retention policies. Once data is no longer needed, ensure secure data disposal by permanently deleting or anonymizing it to prevent unauthorized access.

Employee training and awareness
: Develop a training program to educate employees involved in application integration about data privacy and security best practices. Conduct regular training sessions to raise awareness about potential risks, social engineering attacks, and the importance of adhering to security protocols.

Risk assessment
: Identify potential risks and vulnerabilities associated with application integration.

Application patch
: Keep integrated applications up to date with the latest security patches and updates.

Outdated protocols
: Before using any protocol in your application integration, review the validity of this protocol. Always opt for secure protocols and encryption standards recommended by industry best practices.

Multilayer security
: Perimeter security measures are not enough. Implement layered security controls at multiple levels, including within integrated applications, to provide defense in depth.

Third-party security
: Always conduct due diligence for all third-party tools to ensure adherence to strong security standards and include data protection measures in place.

Compliance and regulations
: Ensure compliance with relevant data privacy regulations and industry-specific standards. Stay updated with evolving regulations and ensure your integration practices align with compliance requirements, such as GDPR, California Consumer Privacy Act (CCPA), or industry-specific standards like HIPAA or payment card industry Data Security Standards (PCI DSS).

Remember, data privacy and security should be an ongoing process. Regularly assess risks, perform vulnerability assessments, and stay informed about emerging threats and best practices in this field to ensure the highest level of protection for your integrated applications.

PART IV

Code Scanning & Testing

Modern Approach to Software Composition Analysis: Call Graph and Runtime SCA

Aruneesh Salhotra

Since the early 2000s, the industry has seen a rapid expansion in open source adoption. Embracing open source is a strategic decision for cost savings and innovation toward a more collaborative, flexible, and high-quality software development paradigm.

Handling dependencies in software development is crucial and intricate. Developers commonly use external libraries and packages to improve and speed up their work, which can unintentionally lead to security weaknesses and operational dangers in their code. Secure and verify these external elements to keep a strong and safe software environment. A quick online search on "open source risks exploited at Yahoo, Equifax, Linksys, Uber" highlights the significance of meticulously managing the risks tied to open source libraries in your organizations.

SCA tools are designed to integrate seamlessly into development workflows, providing real-time analysis, automated alerts, and remediation guidance.

Traditional Approach to SCA

SCA tools scan applications statically by analyzing dependency manifest files to identify vulnerabilities associated with the included packages. They enable organizations to respond swiftly to emerging vulnerabilities in open source projects and ensure adherence to legal requirements in software licensing.

Until very recently, organizations have relied on the traditional SCA to mitigate these risks effectively, playing a crucial role in dependency management, license compliance, and security assessment.

The primary challenge that organizations face with traditional SCA tools is the security scans report every potential vulnerability linked to the packages in your software, regardless of whether or not these vulnerabilities are relevant or utilized by your applications. This approach can lead to a flood of alerts, many of which might not be relevant, overwhelming developers and leading to alert fatigue. SCA tools might communicate risks and vulnerabilities without sufficient context or guidance on remediation, leading to misunderstandings or misalignment with the development team's priorities and workflows.

Modern Approach to Manage Open Source Risks

SCA is undergoing two paradigm shifts in the industry: the call-graph approach and runtime SCA.

The call-graph approach statically analyzes the source code and all its used packages to examine the call graphs and data flows.

Runtime SCA instruments the code to derive a dynamic call graph and then combines it with the static call graph and vulnerable method call chains to find more potentially exploitable vulnerabilities.

Call Graphs

Call graph–based SCA represents a significant advancement in software security. This approach involves creating a call graph, a visual representation of all function calls within a program. By mapping out these interactions, call graph–based SCA provides a comprehensive view of the software's structure and behavior. This granularity is particularly effective in pinpointing the exact locations within the code where vulnerabilities may exist based on the actual usage of open source components.

The transformative aspect of call graph–based SCA lies in its precise and context-sensitive analysis. Traditional SCA tools often provide a broad overview of potential vulnerabilities based on the presence of open source components without considering how these components are used within the application. Call graph–based SCA, by contrast, offers a detailed understanding of the software's behavior, highlighting the areas where vulnerabilities could be exploited.

Runtime SCA

Runtime SCA analyzes software dependencies in real time during the application's actual running. This method dynamically detects and evaluates the actively used open source components and dependencies, offering a live view of the software's behavior in its operational environment. Unlike traditional SCA, which assesses dependencies at the development or deployment stages, runtime SCA provides insights based on how the application utilizes these components in real-world scenarios, leading to more precise and context-specific vulnerability identification.

The key advantage of runtime SCA over traditional SCA lies in its enhanced accuracy and relevance in identifying vulnerabilities. Focusing on the components used during the application's runtime significantly reduces false positives and irrelevant vulnerability reports, enabling developers and security teams to concentrate on genuine threats.

Summary

Both of these approaches align well with DevOps practices, adapt to changes in application use, and support a more proactive security posture. It streamlines resource allocation, focusing efforts on fixing critical and directly impactful security issues, thus making the management of software dependencies more efficient and targeted.

Application Security Testing

David Lindner

In the ever-evolving landscape of software development, ensuring the security of applications has become paramount to combat the ever-shifting threat environment. *Application security testing* (AST) stands as a shield against the growing array of threats that seek to exploit vulnerabilities within software systems. Within the realm of AST there are three pivotal methodologies—SAST, DAST, and IAST. These methodologies all bolster an organization's defense by assessing applications for vulnerabilities. Each offers advantages and limitations, contributing unique dimensions to security strategy. By understanding the nuances, organizations can ensure they are using the correct AST for their environments.

Static Application Security Testing

SAST involves analyzing source code or an application's binary. SAST examines the codebase line by line, looking for security vulnerabilities. SAST can identify issues early in the development life cycle and provide developers with feedback to fix vulnerabilities before they make their way into the final product. SAST is especially effective at detecting issues related to code logic and design flaws.

Advantages of SAST:

- Early detection. SAST identifies vulnerabilities during the development phase, allowing developers to address issues before code is deployed.
- Automation. SAST tools can scan large codebases, making it suitable for ongoing integration and continuous deployment pipelines.

Limitations of SAST:

- False positives. SAST tools typically generate many false positives due to their pattern-based analysis approach.

- Limited runtime context. SAST doesn't consider the runtime environment, so certain vulnerabilities may only manifest during runtime.

Where to apply SAST:

- SAST is typically performed preproduction either on developers' machines or within the CI/CD or testing pipelines.

Dynamic Application Security Testing

DAST involves testing an application while it's running to simulate real-world attack scenarios. DAST tools interact with the application, sending various inputs and observing the responses to uncover vulnerabilities. DAST is effective at identifying issues that are tied to using a running application.

Advantages of DAST:

- Realistic testing. DAST mimics attack scenarios, providing insights into how vulnerabilities might be exploited by malicious actors.
- Runtime context. It captures issues that only manifest during runtime, such as configuration errors, authentication bypasses, and session management flaws.
- Less false positives. DAST typically generates fewer false positives.

Limitations of DAST:

- Late detection. Vulnerabilities are identified after the code has been written, potentially delaying fixes and increasing the cost of remediation.
- Incomprehensive coverage. DAST might miss security issues that can only be detected through code analysis, such as logic flaws and design vulnerabilities.

Where to apply DAST:

- Generally, DAST is used against compiled code, especially on applications. This can be used in a testing environment.

Interactive Application Security Testing

IAST is a hybrid approach that combines the best elements of SAST and DAST. It instruments applications during runtime to capture data about its behavior and interactions in real time. This data is then correlated with the application's source code to pinpoint vulnerabilities and their context within

the code more accurately. IAST provides insights into the runtime context of vulnerabilities, helping developers understand how security issues arise during execution.

Advantages of IAST:

- Unmatched accuracy. IAST's runtime analysis and contextual awareness results in more accurate identification of vulnerabilities with fewer false positives.
- Deeper insight. IAST provides developers with a comprehensive view of how vulnerabilities manifest during runtime.
- Optimized developer workflow. IAST integrates into the development workflow seamlessly, providing insights directly to the developers as they write and test code.

Limitations of IAST:

- Performance overhead. Instrumenting an application might introduce some performance overhead.
- Deployment complexity. Setting up IAST tools and integrating them into the development process can be more complex.

Where to apply IAST:

- IAST is a real-time security monitor that instruments running code, typically used in the testing or staging environment.

In conclusion, SAST, DAST, and IAST are three application security testing strategies. They each have their strengths and weaknesses, and organizations often use a combination of these methodologies to achieve comprehensive security coverage throughout the software development life cycle. SAST helps catch vulnerabilities at the code level, DAST focuses on runtime behavior, and IAST offers a hybrid approach that combines the benefits of both techniques for better accuracy and less total cost of ownership.

WAF and RASP

David Lindner

Web application firewalls (WAFs) and runtime application self-protection (RASP) are both essential tools for protecting web applications from various security threats, but they have different approaches and benefits. These are the key features between a WAF and RASP.

Web Application Firewalls

A WAF is a security solution designed to protect web applications by analyzing incoming traffic and filtering out malicious requests and attacks. It acts as a barrier between the web application and the external world, examining HTTP requests and responses for suspicious patterns or known attack signatures. WAFs are deployed as hardware appliances, virtual appliances, or cloud-based services.

WAF advantages:

Layered defense
: WAFs provide an additional layer of security by sitting in front of the web application and intercepting potentially harmful traffic before it reaches the application itself.

Ease of deployment
: WAFs are relatively easier to deploy and manage, and they can be set up to provide protection without requiring changes to the application's source code.

Signature-based protection
: WAFs identify and block attacks based on known attack signatures, which makes them effective against many well-known attacks.

Performance
: A WAF is typically very efficient with very little noticeable speed implications.

WAF downsides:

Tuning and patching
: Due to the WAFs signature-based technology, they do require constant upkeep and patching. Configuring and maintaining WAFs can be complex, especially for larger and more complex applications.

Alert fatigue
: WAFs may generate many false positives, block legitimate traffic, and fail to detect new and evolving attack patterns, leading to false negatives. This tends to increase alert fatigue for security teams.

Limited context
: WAFs do not understand the context of an application and its logic, potentially leading to inadequate protection against certain attacks.

Runtime Application Self-Protection

RASP is a security approach that focuses on protecting web applications from within. It is integrated directly into the application code or runtime environment and monitors the application's behavior for suspicious activities or anomalies.

RASP advantages:

Deeper contextual insights
: RASP understands the application's internal behavior, allowing it to detect and respond to attacks that do not trigger with traditional WAFs.

Real-time protection
: RASP can respond to threats in real time, dynamically adapting to the application's behavior and the evolving threat landscape.

Accurate results
: RASP's context awareness reduces false positives, as it can better differentiate between legitimate application behavior and malicious activity.

Zero-day protections
: Since RASP operates within the application itself and is detecting attacks against classes of vulnerabilities, it can identify new attack vectors and attacks that are designed to bypass perimeter defenses like WAFs.

Little to no tuning required
: RASP does not require the same level of tuning as a WAF. RASP is not a signature-based technology and instead focuses on tracking tainted data entering a vulnerable class or function.

RASP downsides:

Performance
: Depending on the implementation, RASP can introduce performance overhead, as it requires monitoring and analyzing application behavior in real time.

Integration challenges
: Implementing RASP might require changes to the application's source code or runtime environment, which could be more complex than deploying a WAF.

Lack of environment awareness
: RASP primarily focuses on protecting the application layer. It might not provide comprehensive coverage against attacks targeting other layers of the technology stack.

Necessity of framework knowledge
: True RASP requires knowledge of the underlying framework (e.g., Java's Spring). Protection from RASP may be limited without framework knowledge.

In conclusion, both WAF and RASP are valuable tools for protecting web applications, but they have distinct strengths and weaknesses. WAFs offer an additional layer of protection at the network perimeter, while RASP provides deeper, context-aware security, from within the application. Using both WAF and RASP in tandem provides the most comprehensive and effective security strategy, covering both signature detectable external threats and those that require more contextual understanding within the application.

Zero Trust Software Architecture

Jacqueline Pitter

No doubt you've heard of *zero trust*, information security's favorite buzzword! Technology security vendors label *everything* as a *zero trust solution* these days. And yet, since their implementations are so vastly different from one another, it's difficult to grasp what zero trust stands for.

The confusion is because, in short, zero trust is simply an idea.

NIST SP 800-207 laid out the components of a zero trust architecture in 2020, which is modeled after John Kindervag's original think tank–generated idea in 2008 of "never trust; always verify," achieved by focusing on reducing and protecting your attack surfaces:

- Shrink implicit trust zones with security boundaries as much as possible.
- Deploy reasonable security controls on all protect surfaces.
- Controls should include constant scrutiny of anything crossing a security boundary for verified authenticity and nonanomalous intent.

The zero trust concept applies to all technology architecture, including applications and software. Software design can achieve improved security by developing application environments with a zero trust mindset. In software architecture, this would be achieved with the intentional creation of functional security boundaries and revalidation of any person or process that attempts to cross it, as well as scrutinizing application data for indications of compromise or incompatibility. Containers (e.g., Docker) make this easier by bundling together an application and all its dependencies (i.e., reducing the attack surface of the application), and yet, there are still additional security boundaries that should be established to achieve best-practice container security. Anything traveling between security boundaries should be abstracted and/or tokenized to maintain integrity, nonrepudiation, and where required, confidentiality. This includes having a mechanism that refreshes

that trust token regularly so it isn't just "good" ad infinitum for conveying trustability.

How can you identify where security boundaries and controls need to be added?

Sound application security programs include threat modeling to analyze an application's architecture, design, and functionalities to identify potential vulnerabilities that could impact the security of an application. Start with known application threats.

The Open Web Application Security Project (OWASP) "Top 10 Web Application Security Risks" list establishes a decent foundation for a zero trust web app architecture:

- Identity access should be verified through encryption protocols like identity certificates. Enforce role-based access control (RBAC) for any access beyond a security boundary with meaningful user accounts (i.e., don't run things as root, limit capabilities beyond authorized administrative accounts, and implement two-factor authentication [2FA] wherever feasible).
- Never trust data. Data validation mitigates injection flaws, XSS, and XML vulnerabilities.
- Proactively test for security misconfigurations and verify that security logging and monitoring functionality is in place to prevent broken authentications, compromised credentials, and unintentional sensitive data exposure.

Finally, if your application handles sensitive data, reducing the attack surfaces and focusing security controls on the protect surfaces must be a priority.

For example, the STRIDE AppSec threat modeling method (where you consider each of the categories of spoofing, tampering, repudiation, information disclosure, denial of service, and elevation of privileges) could also be adopted with a zero trust mindset. Consider these threat categories to scrutinize your design and systematically identify any vulnerabilities related to confidentiality, data integrity, availability, nonrepudiation, and IAM, and then further utilize this information to determine the additional security boundaries and controls that need to exist within your application to address these vulnerabilities. And of course, test everything to verify.

AI is becoming increasingly useful in the identification of anomalous protocols and access behaviors crucial for implementing zero trust principles in application security. However, it's essential to remain mindful of a potential issue with AI: the integrity of its training data. If the data used to train AI systems is flawed (such as in cases of AI data set poisoning), it can compromise the accuracy of any AI-driven analysis. When integrating AI into software applications, it's imperative to incorporate robust security metrics and a stringent validation process. Defining what constitutes a successful metric is vital. Adhering to the adage of "never trust; always verify" remains key in this context.

Rethinking Ethics in Application Security: Toward a Sustainable Digital Future

Pragat Patel

Unprecedented levels of connectivity have been made possible by the digital era, but it has also created new ethical concerns for application security. To ensure that our digital systems are not only secure but also ethical, we must adopt a more inclusive approach to application security ethics as more and more of our lives become digital.

In the past, ethical standards for AppSec have emphasized issues with personal security and privacy. Unfortunately, professional associations frequently create these recommendations alone without receiving enough feedback from a wider spectrum of stakeholders. We must include a wider variety of voices in the creation process if we are to produce ethical standards that accurately represent the interests of every member of society.

To ensure that their viewpoints are taken into consideration, this entails interacting with representatives from marginalized populations, advocacy groups, and civil society organizations. For instance, civil rights organizations and populations disproportionately affected by facial recognition technology, such as Black and Indigenous people, should be consulted in the establishment of ethical guidelines for this technology. We may design ethical standards that better represent the concerns of every member of society by incorporating a broader range of stakeholders in the process. This will contribute to ensuring that our digital systems are both just and equitable in addition to being secure.

It is imperative to widen our attention beyond individual privacy and security issues in addition to involving a larger variety of stakeholders in the establishment of ethical norms. The broader social effects of AppSec must be

considered. This covers concerns related to human rights, environmental sustainability, and data sovereignty. For instance, the energy needed to run cloud computing services may have a substantial negative impact on the environment. AppSec experts need to be aware of these effects and take precautions to leave as little of an environmental imprint as possible. The usage of data in AppSec also creates significant issues with data sovereignty. Provide training and guidance to application developers to use energy-efficient code, reduce unnecessary computing resources, and adopt green practices to help reduce the carbon footprint.

By broadening our focus to include these wider social impacts, we can ensure that our digital systems are secure with contribution to a more sustainable society. We can create digital systems that respect the rights and dignity of all individuals and contribute to a more sustainable future. In order to stay aware of new ethical issues, we must also commit to constant discussion and introspection. This entails regular stakeholder engagements, ongoing development, and a readiness to face and resolve ethical issues as they emerge. Because ethics in AppSec is not a one-time fix, this constant dedication to discussion is crucial. New ethical problems will inevitably emerge since the digital ecosystem is continually changing. We can make sure that our ethical standards remain applicable to the interests and concerns of all societal members by committing to constant debate.

To ensure that our digital systems are secure, we need to adopt a more cooperative and inclusive approach to ethics in application security. This entails engaging a broader spectrum of stakeholders in the creation of ethical principles, broadening our attention to encompass the broader social effects of application security, and committing to continual reflection. By implementing these actions, we can create a more equitable and long-term digital future that upholds the dignity of every person. The future is in our hands; what matters now is what we do with it.

Modern WAF Deployment and Management Paradigms

Raj Badhwar

Given the business need and customer experience–driven digital transformation of our applications and the rapid migration toward the (public) cloud, the WAF is an important tool for CISOs and other cybersecurity professionals to protect these digitally transformed, internet-facing (high-risk) web applications and services. As part of the application security controls, real-time monitoring and blocking of threats becomes crucial. Understanding WAF capability, and deploying and managing it, is a must-have skill set for application security professionals.

This essay talks about the modern way of hosting WAFs and provides the separation of roles and responsibilities between the security team and the cloud providers on how best to operationally manage the WAF infrastructure.

Some of the commonly used WAF deployment architectures are as follows.

On Premises WAF Infrastructure for Hybrid Cloud

In the case of a hybrid (public/private) cloud deployment, one legacy architecture that has been used in the recent past is to use an on premises hosted WAF to protect both on premises and cloud-hosted applications. This is generally done by using Domain Name System (DNS) techniques to redirect any application traffic destined for the cloud-hosted application through the on premises hosted WAF, which is primarily configured inline to protect the on premises hosted applications. This approach allows a common set of WAF (access and protection) rules and on premises, full-time security operations staff to protect both public and private cloud-hosted apps. In general, this can cause serious application performance issues for cloud-hosted apps depending on the network connectivity—virtual private network (VPN),

Oracle's FastConnect, or AWS Direct Connect—between the on premises application and the cloud provider. In this paradigm, the on premises operational staff performs all the WAF maintenance activities, including but not limited to monitoring, incident response, and application and vulnerability patching.

Cloud Native WAF Infrastructure for the Public Cloud

In the public cloud, the most common paradigm now is to use the WAF provided and operationally maintained by the cloud provider (e.g., Oracle Cloud Infrastructure (OCI) or AWS). This separates the roles and responsibilities—the WAF implementation, management, and maintenance would be performed by the cloud provider, and the WAF (access and protection) rules would be written and maintained by the customer cloud security team.

Cloud provider responsibilities for cloud native WAF are:

- Configure vulnerability and threat detection and mitigation rules.
- Perform periodic infrastructure and software updates.
- Monitor logs for any suspicious or anomalous behavior in the logs.
- Monitor for distributed denial of service (DDoS) attacks.
- Enable local high availability and remote disaster recovery of the WAF infrastructure.

Other advantages are:

- Use the power of the cloud provider's access to real-time threat intel from various open source and private sources.
- Detect and mitigate the malicious traffic and any attacks away from the application network.
- WAF fits well with the managed *security operations center* (SOC), which can also be provided by the cloud provider or a partner.
- There is faster ramp-up and onboarding of security talent and personnel when needed.
- WAF allows the security team to focus on the detection and mitigation of threats, and not on the patching or daily/weekly maintenance of the (physical/virtual) WAF infrastructure.

- It reduces the security teams' need for capital expenditure and investments.
- The need for compliance with certain regulatory requirements and compliance paradigms (e.g., Service Organization Controls Type 2 [SOC2] testing) stemming from timely vulnerability management and penetration testing of the WAF infrastructure are provided by the cloud provider.

Managed WAF Services

Managed WAF services are generally provided by the WAF *original equipment manufacturers* (OEMs), such as like Imperva, AWS, or Cloudflare, that host WAF farms either in shared colocation data centers or leased cages from public cloud providers within their data centers. They can use DNS-based capabilities to protect both on premises and cloud-hosted applications. They can provide fully managed services that can perform all the (configuration, incident response, and operational management) services mentioned in the two previous paradigms. While this scheme can work well (for less complex applications), the services can become very expensive and are generally not native to the public cloud. These can also bring customer experience issues (from WAF blocks caused by false positives) if the security or application team is not fully engaged on a continuous basis with the WAF managed service provider.

Do You Need Manual Penetration Testing?

Shawn Evans

The goal of web application penetration assessments is to enumerate and exploit risk across all manner of web platforms via automated and manual methods. The responsibility of the assessor is to cause unintended application behavior. This sounds simple, but it is the most basic thread that connects all vulnerabilities. Unintended behavior can manifest itself through information disclosure, subtle variances in HTTP response size, command injection on the client or server side, logical errors, and out-of-band interactions with other servers.

Reliably injecting faults and positively detecting risk is greatly enhanced if the assessor has a fundamental understanding of the code that is being executed behind the scenes. AppSec professionals, even those without a software engineering background, should be able to read and understand code. The ability to read and understand code then lends itself to pseudocode. If you can look at a few lines of code or an entire function and distill that functionality into a concise logical description, then you can effectively describe an entire application. The same is true of application users. By using an application and observing the submitted parameters and expected responses, it's possible to describe in plain language what discrete functions are likely being called on the server side to satisfy a request.

Consider a component of a web application that facilitates account profile updates. The profile update code should first verify that a request contains a valid session identifier that is bound to an account with sufficient privileges. The application then needs to verify that the request parameters or `POST` data is valid. This could include the following highly granular validation checks:

- Is an email address in a valid format?
- Is the email address domain valid?
- Has the password satisfied complexity requirements?

- Do any of the parameters contain potentially malicious characters?
- Is the avatar URL pointing to a trusted domain?
- Did the username supplied with the update request match that of the active session?

Security-conscious developers have a broad range of considerations to make even when implementing basic features, and increasing complexity increases the likelihood of security gaps. Thoughtfully analyzing the application from the perspective of a developer establishes a direct connection to likely attack paths and the potential to trigger unintended application behavior.

The wonderful aspect of this exercise is that it does not require familiarity with the syntax of any language. Instead, it requires that the application security professional be able to visualize and explain the logical steps required to execute a given function, such as updating a profile. All of this lends itself well to efficient manual application pen testing.

Manual testing is slow, but greatly reduces instances of false positives. During a time-boxed pen testing engagement, it's impossible to manually evaluate an application's response to every input permutation. By considering parameter context and postulating server-side workflows, attack paths can be significantly reduced on a per parameter basis. This not only improves the efficiency of the assessment, but also highlights more subtle attack vectors that aren't caused by the blunt injection of malicious characters such as those associated with SQLi, XSS, or command injection. It's the subtle attack vectors that automated scanners are incapable of detecting.

Automated solutions are excellent for coverage but should never be relied upon exclusively. Context is always a component of a successful attack chain. Consider exploits that rely on the injection of negative numbers to change a deduction into a deposit or the ability of an attacker to change the password of other user accounts. These attack scenarios do not rely on the injection of malicious code or metacharacters, but the injection of valid input that results in an unintended application state. Vulnerabilities that fall into these categories require logic to detect. It's the ability to identify purpose and context as it relates to request parameters that distinguish manual from automated testing. This can only be achieved by penetration testers who take the time to understand the application and the parameters being passed to it before they ever start attacking it.

Bash Your Head

Shawn Evans

I often compare security professionals to magicians. These are individuals who retain such deep knowledge of protocols, networks, applications, and security that the act of exploiting flaws in well-designed products is magic to the untrained observer. It's all sleight of hand to a certain degree. This is mostly true for security professionals, but you're probably in the wrong profession if seeing a reverse shell doesn't feel a bit like magic. Hollywood has aided in perpetuating this persona. Traditional security professionals (aka "hackers") are portrayed as being forever engaged in a black and green terminal screen, furiously typing esoteric commands that result in page after page of scrolling binary data.

While these analogies and stereotypes are a stretch, it's worth acknowledging that some of it is rooted in fact. I entered the field of cybersecurity more than fifteen years ago having never familiarized myself with Linux or Unix systems. I observed colleagues formatting unstructured data with a few Bash commands and then sending that data to an HTTP proxy tool with a few more. It might as well have been magic. Today, hardly a day passes without me extensively utilizing a familiar black and green Bash terminal to carry out penetration assessments with increased effectiveness.

Having a fundamental understanding of Bash basics provides significant efficiency gains for penetration testers and security professionals in general.

Consider a scenario where you generated a list of usernames and need to convert that data into properly formatted email addresses. In Bash, this can be accomplished in a single line of commands:

```
$ cat user_name_list.txt | awk '{print $0"@nopsec.com"}'
```

Let's get more complex. Say we have a file that contains a list of email addresses obtained through open source intelligence gathering and we identified an application endpoint vulnerable to username enumeration. How can we use Bash to identify the email addresses that correlate to a valid account? We could use the following command:

```
$ for username in $(cut -d '@' -f 1 usernames.txt); do code=$(curl -I
'http://localhost:8083/authors/'$username 2>/dev/null | head -n 1 |
cut -d$'' -f2); if [ "$code" == "200" ]; then echo $username'
is a valid user!'; fi;
done
```

Output:

```
rfranklin is a valid user!
```

That is magic! Evaluating this command, we see that it combines a Bash for loop, if...then, pipes (|), and variables to chain together the output of the Bash commands `curl`, `cut`, and `head` and `return` valid user accounts. It executed a fairly complex sequence of events in a self-contained command line that facilitated the enumeration of hundreds of potential valid accounts. Sure, this kind of operation could be accomplished in an HTTP proxy tool such as Burp, but how much time would have been wasted formatting data and setting up the intruder attack? Oddball, one-off challenges like this present themselves in nearly every AppSec assessment. The ability to live off the land to quickly manipulate, isolate, and create data based on a few simple commands increases engagement efficiency and helps identify use cases not possible with any one tool. The most useful piece of code I ever created was a Bash function to pull valid IP addresses from arbitrary input:

```
function ipgrab() {
read line; echo $line | grep -E -o
'(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9]
[0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|
[01]?[0-9][0-9]?)';
while read line ; do echo $line | grep -E -o
'(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9]
[0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|
[01]?[0-9][0-9]?)'; done
echo $line | grep -E -o
'(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9]
[0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|
[01]?[0-9][0-9]?)';
}
```

I use this function at least 10 times a day. It is included in the Bash configuration of every Linux machine I create, and I regularly pipe the output to other security tools.

At a high level, there are few limits to what can be accomplished in Bash. If you want to send some data to a generic TCP connection, use Netcat. If you want to send an HTTP request, use cURL or Wget. Want to carve up a **.csv* file? Use `cut`, `grep`, and `awk`. There exist tens of thousands of additional command-line utilities available via "apt." The potential for quick solutions is unmatched. The command syntax can get ugly, especially when you're dealing with conditional statements, piped commands, regular expressions, for loops, and other languages such as Python. But the time saved is significant once you hit a certain level of comfort with the syntax. Manually manipulating disparate sets of data via text files or spreadsheets is inefficient. Not to mention that when it works, a well-crafted Bash command is akin to magic.

Exploring Application Security Through Static Analysis

Tanya Janca

Start securing software by performing *static application security testing* (SAST). Unlike *dynamic analysis*, which is performed when an application is running and its state is changing constantly, *static analysis* is performed on a version of code that is static and unchanging. You don't need to run the application, nor do you even need to build it or compile it. You can just point a static analysis tool directly at your codebase and find out if there are potential vulnerabilities. You can also do a manual code review, which is also a form of static analysis.

Static analysis has changed a lot in the past few years. The first generation of static analysis tools were slow, tedious, and full of false positives. These tools used *symbolic execution* to parse your application in small chunks, just like a compiler would, then go down every single possible avenue that could happen to your application. Every potential outcome of your code is examined for potential vulnerabilities. These types of systems are extremely thorough, but also slow and prone to producing false positives. Unfortunately, with DevOps and the need to develop software faster than ever before, it is difficult to accept a high rate of false positives or wait several hours for a scan to run. I generally only recommend these tools if you need an extremely high level of precision and/or security assurance. These systems generally require a security expert to validate the results and then explain what needs to be done to the software developers.

Advances have been made over the past few years, and next-generation or second-generation static analysis tools have come on the market. These newer tools use flow analysis and antipattern matching to find vulnerabilities.

Flow analysis, in its most basic form, tracks input to the application and sees where that input is used. The static analysis tool wants to ensure the program verifies the data is safe *before* it is used within your system. For instance, a user could put a script into a search field, then press the search button, and if your application used that data as part of a query to send to the database, it could have dire consequences. Flow analysis shows software developers where they have made an error in trusting user input before validating that it is what they are expecting and safe for them to use. Note: first-generation SAST tools sometimes use flow analysis as well as symbolic execution.

Antipatterns refer to something that is known for certain to be problematic. With *antipattern matching*, the tool uses regular expressions to find functions or other patterns that are certain to introduce a vulnerability. Antipattern matching is extremely fast.

These newer versions of static analysis provide significantly faster and more accurate results than ever before. They provide significantly fewer false positives, almost none. That said, they do miss some vulnerabilities, occasionally. No system is perfect.

When it comes to securing applications and APIs, be sure to test or verify them from all three angles discussed in this essay (static, dynamic, and third-party code). Starting with static analysis may be easier, as the static code tooling is mature enough, and developers are able to adopt them more readily as part of the processes they already follow.

Introduction to CI/CD Pipelines and Associated Risks

Tyler Young

Continuous integration and continuous delivery (CI/CD) pipelines are an essential part of modern software development processes. They provide a streamlined way for development teams to manage code changes, automate builds and testing, and deploy changes quickly and efficiently. However, CI/CD pipelines also present unique risks and challenges to the security of your software and infrastructure.

The following are notable risks of CI/CD pipelines:

Code injection
: One of the most significant risks associated with CI/CD pipelines is the possibility of code injection. Attackers can exploit vulnerabilities in your pipeline to inject malicious code into your software, potentially compromising the security of your systems and data. This risk becomes more significant when companies leverage open source software libraries, as vulnerabilities can be identified and exploited at scale.

Secret/credential hygiene
: CI/CD pipelines often require access to sensitive credentials, such as API keys, database passwords, or other critical authentication tokens. If these credentials are not adequately secured, they can be easily exposed to threat actors, who can then use them to access systems and data. While this isn't unique to CI/CD security and is a persistent risk to AppSec in general, any time you are deploying automated workflows that access secrets there is an associated risk that needs to be scrutinized.

Pipeline poisoning
: Similar to what we witnessed with the SolarWinds attack (malicious code deployed as part of legitimate software), CI/CD pipelines can be

exploited to deploy malicious code at scale and automatically. Attackers can insert malicious code into your pipeline, which can then be executed during the build process, potentially compromising the security of your entire system. Oftentimes, this goes undetected as code-scanning software looks for vulnerable code.

Misconfigurations
: Another risk associated with CI/CD pipelines is the possibility of misconfigurations. If your pipeline is misconfigured, it can lead to unintended access to sensitive data, insecure code deployment, or accidental exposure of data. These types of risks can arise due to improper access control or an accidental keystroke.

As with any risk, there is always a countering course of action to put your organization in a better position to reduce the likelihood of the threat. The following are a few security hygiene steps you can take to better secure your CI/CD pipelines:

Implement access control
: One of the most important steps to secure your CI/CD pipeline is to implement strong access control measures. Only authorized personnel should be able to access your pipeline, and you should use tools like IAM to manage and control access.

Secure your credentials
: Ensure that your credentials are stored securely and are not accessible to unauthorized personnel. It's imperative when working with automated systems to avoid sharing. Leverage tools such as KMS or Key Vault to protect your sensitive authentication data.

Secure build environments
: As with any system, it's important to ensure that your CI/CD pipelines are maintained in your asset inventory. Knowing what systems are being leveraged, where, and by whom is fundamental to securing any system. In addition, ensure your build environments are isolated from the rest of your network. Ensure that these environments are regularly updated with security patches, access is monitored, and that they are properly configured to minimize the risk of attacks.

Implement code reviews
: Implement a code review process to ensure that code changes are thoroughly reviewed for potential vulnerabilities and errors before they are deployed. Creating "gold" images of your images and running hash

checks against them, ensures that any deviation in the code base is able to be detected. This can help to catch potential security issues before they become critical problems.

Use security best practices
: Implement security best practices such as password management, least privilege access, and regular software updates. These practices can go a long way in securing your CI/CD pipeline and protecting your systems and data from potential threats.

CI/CD pipelines are critical to modern software development processes, but they also present unique risks and challenges to the security of your infrastructure. By implementing strong access control measures, securing your credentials, using secure build environments, implementing code reviews, and following security best practices, you can help secure your pipeline and minimize the risk of attacks and vulnerabilities.

PART V

Vulnerability Management

Demystifying Bug Bounty Programs

Aldo Salas

I've been involved in a couple of companies where the use of a bug bounty program was generally perceived as negative. There are a number of reasons I've heard many reasons why implementing a bug bounty program was not sought after at that time, including:

- The program can be expensive.
- It takes several months before seeing results.
- It's hard to manage.
- Researchers can be hostile.

The financial issue is a misplaced concern, since it's significantly cheaper to find a critical vulnerability by doing a routine pen test, or through a controlled bug bounty program than having one of your customers discover that vulnerability and demand a fix as soon as possible.

The consequences of customers and prospects finding security issues in an application can range from a customer reducing their confidence in your product, to actually losing an opportunity because the prospect did not consider the application to be secure enough. This is vastly more expensive than an average bug bounty program.

The results of a bug bounty program can be obtained almost instantly because the researchers involved are usually highly motivated and work across multiple time zones, allowing for continuous progress. Lastly, the challenges of managing the program and coordinating with researchers can be easily addressed by using a bug bounty partner or platform, which streamlines the process and facilitates effective collaboration.

Preparing the Test Environment

For startups, it is usually easier to run a bug bounty program when the company only has one main core product or a limited number of offers. When you think of a larger company that has hundreds of applications and thousands of endpoints, providing full access for testers can become a challenge.

Ideally, it should be fairly easy to provide a new test environment that researchers can freely use to run all the tests they want, but if this is not possible, most applications already have a testing environment where there's no real data and can be used for providing access to researchers.

Testing in Production

I realize this sounds like a meme and that you probably have heard this as a running joke about how tough developers run their testing in a production environment.

The reality is that it is recommended to include all your production assets in scope for a bug bounty program; otherwise, you most likely will be missing several vulnerabilities that are only applicable to production applications and configurations.

I actually was told by a coworker that we would look like *amateurs* if we were to run our bug bounty program in production in case the researchers brought down our system and our customers couldn't use it.

This couldn't be further from the truth.

Just think about it for a minute: wouldn't you want to know if some random person with their limited resources and an open source web scanner is able to bring down your entire production environment? Wouldn't it be really concerning if this were the case? That's something I'd like to fix if it happened.

Recommendations

Here are some tips for a successful bug bounty program:

Use an existing platform.
: Implementing a bug bounty program can be challenging, and that is why I recommend using one of the many platforms that already exist. They can help your organization get started in no time, and they will handle all the communications, accounts, payouts, etc.

Implement guardrails.

If there are some assets/domains/vulnerabilities that you don't want to include for any reason, these can be marked as out of scope.

Define payouts.

Bounties are a great way to motivate researchers to participate in your program; consider increasing the bounties whenever possible.

Test in production.

It may be risky, but this gives your company every reason to make sure high availability is guaranteed for your application. You may mitigate the risk of bug bounty testing in production by isolating environments, adopting gradual rollouts, implementing feature toggles, and maintaining vigilant monitoring to ensure high availability while minimizing potential disruptions.

EPSS: A Modern Approach to Vulnerability Management

Aruneesh Salhotra

Making good predictions and intelligently foreseeing what might happen next before acting is crucial for decision making. These goals are also essential in managing security risks.

Cybersecurity is a high-stakes race, and the winner is the one who finds the weakness first. Most security professionals work against the clock to find and fix system holes before opportunistic malicious actors exploit them.

Traditional Approaches Are Dated

Traditional vulnerability management approaches, such as the Common Vulnerability Scoring System (CVSS), are increasingly facing scalability challenges in today's rapidly evolving cybersecurity landscape.

The core issue with CVSS lies in its static nature: it assigns a severity score to vulnerabilities based on a fixed set of criteria without considering the dynamic context of each organization's unique network environment. These traditional approaches don't account for the ever-changing tactics of threat actors, making it challenging to prioritize vulnerabilities based on real-world threat intelligence.

According to research by the Forum of Incident Response and Security Teams (FIRST), businesses and technology vendors fix only 5%–20% of vulnerabilities every month. Yet only 2%–7% of vulnerabilities are ever exploited. But which ones should we focus on remediating exactly? Since we cannot be sure which vulnerabilities must be remediated first, we must prioritize.

The World of EPSS

Fast forward to 2019. Exploit Prediction Scoring System (EPSS) was started as an open community-driven effort to model and manage vulnerability risk from a probabilistic perspective. EPSS is increasingly recognized by research communities and a growing number of enterprises as a contemporary method for managing vulnerabilities. This approach is seen as a way to overcome the limitations inherent in traditional vulnerability management strategies.

EPSS scores range from 0% (lowest) to 100 % (highest) probability of exploitation. EPSS also provides percentile rankings; percentile rankings measure EPSS probability relative to all other EPSS scores. The combination of probability and percentile enables advanced prioritization inputs.

EPSS leverages machine learning algorithms to predict the likelihood of a vulnerability being exploited in the wild in the next 30 days. By integrating real-time data from various sources, including threat intelligence feeds and active exploitation trends, EPSS provides a more dynamic and context-aware assessment. This approach enables organizations to prioritize their remediation efforts more effectively, focusing on vulnerabilities that are not just theoretically severe but are also likely to be targeted by attackers. By doing so, EPSS helps organizations optimize their "scarce" and usually "expensive" security resources, ensuring that they address the most pressing threats, thus enabling a more proactive and efficient vulnerability management process.

Key Aspects of EPSS

Here is what's new in EPSS that makes it more attractive than CVSS:

- Gathers data from vulnerability databases, threat intelligence, and exploit occurrences.
- Analyzes data using ML to identify exploitation patterns.
- Continuously refreshes data for current threat landscape relevance.
- Considers factors like exploitability, exploit code availability, and software popularity.
- Assigns probability scores to vulnerabilities, indicating exploitation likelihood.
- Adapts scores based on specific network environments and exposure levels.

- Guides efficient response with emphasis on high-probability vulnerabilities.
- Displays exploitation activity as evidence that exploitation of a vulnerability was attempted, not that it was successful against a vulnerable target. The model collects data from honeypots, intrusion detection system (IDS) sensors, intrusion prevention system (IPS) sensors, and host-based detection methods.

The EPSS model is invaluable for prioritizing patching efforts based on vulnerability exploitation likelihood. One of the primary benefits of the EPSS model is its open source nature, allowing for widespread access, transparency, and community contributions. Using vulnerable data to make predictions provides more accurate results than scoring systems relying solely on severity ratings. However, given that EPSS relies on community feedback and knowledge, it's crucial to emphasize that companies should not solely rely on it. It's essential to consider CVSS scores and make decisions based on the specific setup of their infrastructure, especially regarding asset exposure to the internet. Integrating various sources of information ensures a more comprehensive and informed approach to cybersecurity decisions. By implementing the EPSS model, organizations can enhance their cybersecurity measures and protect their digital assets more effectively. It is not mainstream yet, but it's worth your organization considering and trying it!

Navigating the Waters of Vulnerability Management

Luis Arzu

Vulnerability management is a cornerstone in the defense of modern applications, ensuring the identification, prioritization, and mitigation of potential weaknesses. While academia and industry frameworks provide valuable guidance, the real-world challenges and experiences of a practitioner bring a unique perspective to this critical discipline. In this essay, I will share insights about vulnerability management, drawing from my own journey in the field.

Understanding the Dynamic Landscape

The world of vulnerabilities is ever evolving, with threat actors constantly seeking new ways to exploit weaknesses. I have witnessed firsthand the shifting landscape of vulnerabilities, from the traditional network perimeter to the realm of web applications, APIs, and cloud-based infrastructures. I have learned to adapt and stay ahead of emerging threats through continuous monitoring and vulnerability scanning.

Prioritization: The Art of Decision Making

One of the greatest challenges in vulnerability management is prioritizing the multitude of identified vulnerabilities. Balancing the severity of a vulnerability with its exploitability and potential impact on critical assets requires a deep understanding of the organization's risk appetite and business priorities. Through iterative cycles of assessment, validation, and remediation, I have honed the ability to make informed decisions that align with an organization's unique requirements. For instance, when faced with limited resources, I recall a time when we had to prioritize patching vulnerabilities on servers hosting critical customer data over those on non-customer-facing systems. Such an example highlights the importance of aligning vulnerability

management efforts with the organization's goals, priorities, risks, and other compensating controls in place.

Building Collaborative Relationships

Effective vulnerability management extends beyond the realm of technical expertise. It requires building collaborative relationships with various stakeholders, including developers, system administrators, and business units. Over the years, I have discovered that fostering open communication channels and establishing trust is crucial for driving a security culture. That communication is also based on security professionals like us to understand developers' priorities and provide them guidance on the remediation. By bridging the gap between security and development teams, I have been able to advocate for secure coding practices and create a shared responsibility for vulnerability management.

Leveraging Robust Vulnerability Management Solutions

In today's complex threat landscape, organizations can derive immense benefits from robust vulnerability management solutions. These tools offer automation, scalability, and actionable insights, empowering practitioners to efficiently handle vulnerabilities. By leveraging such solutions, organizations can streamline vulnerability scanning, prioritize risks, and facilitate efficient remediation efforts. These tools also facilitate integration with various systems and provide comprehensive reporting capabilities, empowering practitioners with visibility and data-driven insights. Through their usage, practitioners can drive vulnerability management initiatives with confidence, effectively communicating the urgency and impact of vulnerabilities to stakeholders and securing necessary investments from organizational leadership.

Moreover, these solutions bring broader advantages beyond their technical functionalities. They enable practitioners to demonstrate the value of vulnerability management efforts to organizational leadership, fostering a culture of security and securing buy-in for necessary investments. Additionally, by effectively managing vulnerabilities, organizations can enhance their reputation and build trust with customers, partners, and stakeholders. For instance, by diligently addressing vulnerabilities and showcasing proactive security measures, my organization was able to win the trust of a major client and secure a long-term partnership.

Conclusion

Vulnerability management is a multifaceted discipline that requires a comprehensive understanding of the dynamic threat landscape, the art of decision making in prioritization, building collaborative relationships, and leveraging robust tools. Through years of experience as a practitioner, I have learned the importance of aligning vulnerability management efforts with organizational goals and priorities, fostering open communication channels, and advocating for a culture of security. At the same time, by leveraging robust vulnerability management solutions, organizations can streamline their processes, prioritize risks, and effectively communicate the urgency of vulnerabilities to stakeholders. Furthermore, continuous improvement through knowledge sharing and remaining adaptable ensures that practitioners can navigate the evolving security landscape with proficiency and resilience.

Safeguarding the Digital Nexus: "Top 25 Parameters to Vulnerability Frequency"

Lütfü Mert Ceylan

In an era where the digital realm intertwines with our everyday lives, ensuring the security of our virtual ecosystems has become more vital than ever. Enter the realm of "Top 25 Parameters to Vulnerability Frequency," a comprehensive initiative that dissects the intricate anatomy of web applications to expose the most critical parameters vulnerable to exploits. Let's embark on a journey to unveil these parameters, understand their significance, and arm ourselves with the knowledge to fortify our digital ramparts.

Exploring Vulnerability Categories: A Profound Expedition to Parameter Frequencies

Within the complex architecture of web vulnerabilities, the "Top 25 Parameters" patterns reveal the most common parameters across six major vulnerability categories, each representing a vulnerability in the firewall that must be addressed:

Cross-site scripting (XSS)
: Our expedition commences with parameters susceptible to XSS attacks. These vulnerabilities enable attackers to inject malicious scripts, bypassing security measures and potentially compromising user data:

```
?q=, ?s=, ?search=, ?id=, ?lang=, ?keyword=, ?query=, ?page=,
?keywords=, ?year=, ?view=, ?email=, ?type=, ?name=, ?p=, ?month=...
```

Server-side request forgery (SSRF)
: Our journey then leads us to parameters enabling manipulation of server-side requests. Attackers exploit these weaknesses to gain unau-

thorized access to internal resources, undermining data integrity and system confidentiality:

```
?dest=, ?redirect=, ?uri=, ?path=, ?continue=, ?url=, ?window=,
?next=, ?data=, ?reference=, ?site=, ?html=, ?val=, ?validate= ...
```

Local file inclusion (LFI)
: Our exploration shifts to unauthorized file access vulnerabilities. Attackers infiltrate these parameters to gain unauthorized access to critical files, potentially leading to data breaches and unauthorized access:

```
?cat=, ?dir=, ?action=, ?board=, ?date=, ?detail=, ?file=,
?download=, ?path=, ?folder=, ?prefix=, ?include=, ?page=, ?inc= ...
```

SQL injection (SQLi)
: Our quest continues with parameters vulnerable to SQLi attacks. Inadequately sanitized inputs create gateways for attackers to manipulate databases, extract sensitive data, and compromise system integrity:

```
?id=, ?page=, ?dir=, ?search=, ?category=, ?file=, ?class=, ?url=,
?news=, ?item=, ?menu=, ?lang=, ?name=, ?ref=, ?title=, ?view= ...
```

Remote code execution (RCE)
: Our expedition delves into vulnerabilities facilitating RCE. Exploiting these vulnerabilities grants unauthorized access to an application's backend, allowing attackers to execute arbitrary code and compromise system security:

```
?cmd=, ?exec=, ?command=, ?execute=, ?ping=, ?query=, ?jump=,
?code=, ?reg=, ?do=, ?func=, ?arg=, ?option=, ?load=, ?process= …
```

Open redirect
: Our journey concludes by examining parameters enabling manipulation of URL redirection. Malicious actors orchestrate these manipulations to deceive users and direct them to potentially harmful websites:

```
?next=, ?url=, ?target=, ?rurl=, ?dest=, ?destination=, ?redir=,
?redirect_uri=, ?redirect_url=, ?redirect=, /redirect/,
/cgi-bin/redirect.cgi?, ...
```

The full version of the patterns can be found in the official list.

Empowering with Knowledge: The Path Forward

In the codex of vulnerabilities, the patterns emerge as a guide, orchestrating the harmonious blend of knowledge and action. By revealing the vulnerabilities that undermine digital integrity, this project empowers us to secure our applications, bolster our defenses, and champion a safer digital landscape.

This knowledge equips us to identify vulnerabilities and implement robust countermeasures. It's an invitation to adopt secure coding practices, embrace stringent input validation, and conduct comprehensive security assessments.

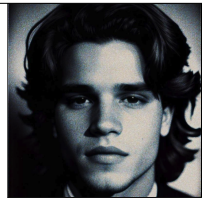
As the lead behind the OWASP Top 25 Parameters project, I want to guide us toward empowered digital spaces. With my unwavering commitment to digital security, I strive to empower developers and security practitioners to act with community awareness in our quest to protect digital creations.

These parameter patterns transcends being a mere compilation—it is an embodiment of collective effort and shared purpose. It symbolizes our commitment to secure applications, elevate industry standards, and forge a future where digital innovation thrives within the fortress of security.

To view the latest OWASP Top 25 Parameters list, visit the OWASP website.

Unveiling Paths to Account Takeover: Web Cache to XSS Exploitation

Lütfü Mert Ceylan



I am pleased to share with you a real-life experience on a security vulnerability that showcases the intricate interplay between seemingly benign technical nuances and profound security implications. In 2023, I embarked on a journey that led me to uncover a reflected XSS vulnerability within a video game company's infrastructure. My investigation traversed the realms of caching mechanisms and vulnerability escalation, culminating in the discovery of a pathway to account takeover. In this discourse, I illuminate the technical underpinnings that drove this narrative.

Discovery of Vulnerability

At the heart of this narrative is a reflected XSS vulnerability lurking within the language input field. At its inception, it might have appeared as a conventional XSS vulnerability; however, a closer examination unveiled nuances that held the potential for drastic consequences. Upon accessing a specific URL, the vulnerability could be exploited to compromise user sessions through a meticulously crafted payload. The vulnerability stemmed from the absence of critical security measures such as the HttpOnly and secure flags.

But What Is Reflected XSS Vulnerability?

Reflected cross-site scripting (XSS) is a subtype of XSS where malicious scripts injected into a web application are reflected off the server, such as in error messages or search results. This occurs when a user clicks a malicious link or submits a form, causing the script to execute in their browser. It differs from other XSS types, like stored XSS, where the script is permanently on the server, or DOM-based XSS, which involves the web page's document

object model without server interaction. Reflected XSS is unique, as the malicious script is not stored but reflected in response to user input:

HttpOnly
: An HttpOnly cookie is a security mechanism that curtails scripts' access to sensitive cookie data, mitigating potential attacks.

Secure
: A secure flag, when employed, confines cookie transmission exclusively to secure HTTPS connections, bolstering data integrity and confidentiality during transit.

Amplification Through Web Cache Exploitation

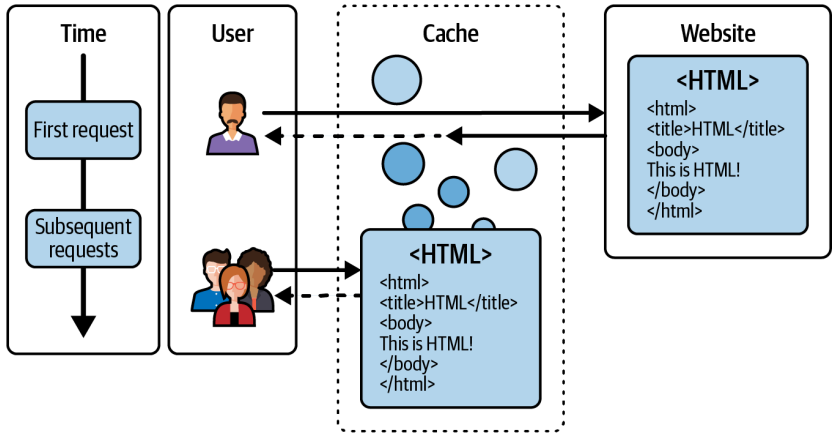
Deeper exploration revealed a noteworthy facet—the caching behavior of the server in relation to the language parameter's value. This revelation held a pivotal implication: the potential for a vulnerability's payload to be ubiquitously embedded across diverse pages within the site. Distinct from conventional web cache poisoning, this approach selectively cached sections housing the payload, avoiding an all-encompassing page cache. To further comprehend this mechanism, reference the illustrative framework I formulated.

The Genesis of Account Takeover

The vulnerability's magnitude was compounded by the glaring absence of pivotal security measures—namely, the HttpOnly and secure flags—alongside the omission of a Content-Security-Policy (CSP) framework on the website. This compounded vulnerability permeated the site's framework, enabling potential breaches that could compromise sensitive user data.

Exploiting the Dynamics of Web Cache Poisoning

Web cache poisoning provided an avenue for exploitation that was as discreet as it was impactful. By incorporating the vulnerable URL within external requests, carefully camouflaged in seemingly innocuous elements like images, attackers could systematically exploit the cached payload. On each occasion a user accessed the target site, the payload was discreetly executed, silently granting unauthorized access.



Mitigation and Beyond

Prompt action was initiated upon reporting the vulnerability, prompting an acknowledgment of its gravity. Swift measures were enacted to address this vulnerability and reinforce the security posture.

In summation, this journey weaves a tapestry that underscores the intricacies of web security and the criticality of maintaining vigilance across technical landscapes. As we navigate the digital frontier, it is essential to recognize that even the minutest of vulnerabilities can reverberate with far-reaching implications.

Embarking on this exploration has illuminated the dynamic intersection of security, technology, and vigilance, paving the way for fortified digital domains.

Sometimes the Smallest Risks Can Cause the Greatest Destruction

Lütfü Mert Ceylan

In the intricate realm of cybersecurity, where digital fortresses are constructed to ward off looming threats, it's easy to focus solely on the grand and complex vulnerabilities. Yet, amid the labyrinth of sophisticated exploits, it's the unassuming vulnerabilities that can spark catastrophic consequences. This phenomenon underlines the profound truth that sometimes the smallest risks can cause the greatest destruction.

In the dynamic landscape of cyber threats, security practitioners often prioritize their efforts by focusing on high-profile vulnerabilities with elaborate attack vectors. While this approach is undoubtedly crucial, it's equally essential to recognize the latent dangers concealed within seemingly minor security gaps. Hackers have a remarkable knack for exploiting the unexpected, and even the most innocuous vulnerabilities can become a launching pad for devastating attacks.

The interplay between seemingly insignificant vulnerabilities and larger security weaknesses can create a domino effect of compromise. Cybercriminals, armed with an intimate understanding of system intricacies, can weave together multiple low-severity vulnerabilities to orchestrate a high-impact breach. These smaller vulnerabilities might individually appear harmless, yet their synergy can lead to an intricate chain reaction that eventually breaches the digital bastions. Moreover, the art of blending distinct vulnerabilities—like harmonizing a deceptively innocent melody into a symphony of chaos—can enable attackers to traverse the security landscape unnoticed.

This concept also extends to the insidious practice of "risk layering" or "vulnerability chaining." Hackers recognize that a single vulnerability might not suffice to breach a fortified system, so they ingeniously employ multiple vulnerabilities in unison. By stacking these vulnerabilities like puzzle pieces,

they construct a composite threat that can dismantle even the most fortified defenses.

In the intricate landscape of application security, it remains a truth that the most inconspicuous vulnerabilities possess the potential to birth the most devastating breaches. These vulnerabilities, often dismissed as mere afterthoughts, can sow the seeds of peril when interconnected or exploited in unanticipated ways. This underscores the imperative for a comprehensive cybersecurity strategy. Adopting a proactive stance toward risk identification and mitigation becomes indispensable, ensuring that even the smallest risks are addressed before they snowball into catastrophic events. Security professionals must view the threat landscape through a nuanced lens, identifying not only the glaring weaknesses but also the subtle interconnections that can magnify their impact.

As the cybersecurity environment continues to evolve, I'd like to define this situation with the following proverb: "Sometimes the smallest risks can cause the greatest destruction." This resonates more than ever. Armed with this awareness, organizations can take a proactive stance by supporting the smallest cracks in their digital armor and eliminating the potential for successive breaches.

In conclusion, the cybersecurity realm is a battleground where the most inconspicuous vulnerabilities can serve as critical points of entry for malicious actors. Recognizing the potential harm that these seemingly minor risks can inflict is paramount. With an unyielding commitment to understanding the interconnectedness of vulnerabilities, security practitioners can navigate the complex landscape and protect their digital domains from the unexpected and potentially catastrophic threats that lurk within the shadows.

Effective Vulnerability Remediation Using EPSS

Reet Kaur

Every day, new software vulnerabilities are disclosed. However, due to a lack of resources and conflicting business requirements, it is impossible for organizations to patch all these vulnerabilities and perform effective vulnerability remediation within defined service-level agreements (SLAs). Most companies can only fix between 5% and 20% of known vulnerabilities per month.

We also know that only a small subset of these many vulnerabilities are ever seen to be exploited in the wild. With a multitude of vulnerabilities to address and limited resources, it's essential to prioritize remediation efforts while allocating resources more efficiently and effectively.

To tackle this issue, the FIRST (Forum of Incident Response and Security Teams) organization developed the Exploit Prediction Scoring System (EPSS). It is a community-driven effort to combine descriptive Common Vulnerabilities and Exposures (CVE) information with evidence of actual exploitation in the wild. By collecting and analyzing this information to include in our vulnerability management platforms, we may improve vulnerability prioritization by estimating the likelihood that a vulnerability may get exploited. The EPSS model produces a probability score between 0 and 1. The higher the score, the greater the probability that a vulnerability will be exploited within the next 30 days.

EPSS gathers data from many sources, including CVE lists, text-based tags from CVE descriptions, exploit codes from repositories like Metasploit and GitHub, security scanners, CVSS v3 vectors, vendor information through Common Platform Enumeration (CPE), and real-world exploitation data submitted by researchers and the community. This rich data set, updated regularly, is the basis for training the EPSS model and improving its accuracy.

This data-driven approach is a significant step forward in focusing remediation efforts where they are most needed.

Combining CVSS and EPSS helps with resource optimization, as it allows focus on patching fewer, but critical, vulnerabilities that have a high likelihood of being exploited while maintaining or even improving efficiency and coverage. By embracing EPSS, enterprises can enhance their resource allocation and respond more effectively to potential threats.

To maximize the effectiveness of EPSS, it's essential to evaluate vulnerability reachability within your enterprise environment. This assessment entails gauging how easily a hacker can exploit a security vulnerability in a system or environment. Factors such as the accessibility of the system/application with the vulnerability, its exposure to external threats, and the ease with which attackers can exploit the vulnerability to compromise the software are considered.

Key elements influencing vulnerability reachability include the effectiveness of security controls, the capabilities and resources of potential attackers, system configurations, and the potential impact of an exploit. In essence, the goal is to determine the likelihood of a successful breach by evaluating these factors.

As cyber threats keep evolving, it's important to rely on expert advice and real-world context to make informed decisions. By considering both vulnerability reachability—how likely a vulnerability is to be exploited—and EPSS, companies can prioritize which issues to address first and allocate resources effectively to tackle the most serious threats.

Bug Bounty—Shift Everywhere

Sean Poris

A *bug bounty program* is an initiative that organizations adopt to invite ethical hackers to identify vulnerabilities in web applications, software, or even hardware.

When I first started looking at bug bounty programs as a part of a holistic AppSec program, business and legal teams at that time pushed back incredibly hard. Their response was somewhere between incredulity and blatant aggressive defiance. A typical conversation with the legal department could have easily ended with, "If anyone intentionally engages in hacking our websites, we will issue them a cease and desist letter immediately." Clearly, the hurdles were too great.

Fast forward to about 2019, and bug bounty programs were becoming a mainstay in many programs. But with the *shift left* mantra of getting security focused earlier in the development life cycle, where does a right-shifted bug bounty program truly sit? The reality is that AppSec efforts aren't perfect, and internal corporate security programs simply don't have the scale of resources to bring the level of diversity necessary to execute testing to the degree and duration a healthy bug bounty program does. So, in the classic sense of security, bug bounty programs are part of a "belt and suspenders" model of complementing shift left security activities, following today's more appropriate philosophy of "shift everywhere."

With an active bug bounty program, you can have dozens, if not hundreds, of eyes with unique techniques, tools, and perspectives to test your application's resilience in ways you hadn't had the time yourself. In fact, once you build some loyalty with hackers in the community, they'll come to know your site as well as some of your developers do. At that point, they are an invaluable resource to identify increasingly complex issues that might slip through the pipelines you've built, the scans you execute, and the manual reviews you conduct.

A bug bounty program is an excellent complement to a strong vulnerability management program, and in fact, bug bounty cannot exist without a vibrant vulnerability management capability. Why? First, if you can't articulate your vulnerability posture, or at least be sure that you've established basic hygiene with a foundational program, you cannot expose yourself to the hacker community. They will immediately deplete your budget by finding the low-hanging fruit your program should have identified. Rather, it makes sense to invest in your product security and vulnerability management functions. Once you have the basic hygiene accounted for, you'll be much better able to adopt a bug bounty program and leverage hackers to find the things you cannot.

Further, the concept of a bug bounty life cycle drives increased value to your security program as a whole. As hackers find more interesting techniques, internal security practitioners, corporate infrastructure, and application development engineers can leverage those insights to fine tune scanners, build better tests to include in subsequent test suite updates, and scour other products for the same types of vulnerabilities. This uplevels and hardens the security program as a whole. This drives hackers to identify more and different vulnerabilities, whose remediations also find their way into left-shifted security practices. By combining a healthy vulnerability management and product security function with a full round trip bug bounty life cycle, security programs can harness the power of hackers and the value of a healthy bug bounty program.

PART VI

Software Supply Chain

Integrating Security into Open Source Dependencies

Alyssa Columbus

Open source software has become ubiquitous in application development, providing ready-made components that accelerate time-to-market, reduce costs, and foster shared innovation. However, unchecked open source usage creates significant security risks that must be proactively managed throughout the software life cycle. Here are some essential practices to mitigate these risks and ensure the overall security of the software development process.

Selecting Secure Open Source Libraries

When selecting open source libraries, frameworks, and components, thoroughly vet both code and community:

- Review the open source project's public vulnerability history to assess susceptibility to vulnerabilities and how quickly issues are addressed. An engaged, responsive maintainer community adept at rapidly patching flaws is essential.
- Evaluate the community. Look for a large and active community with a diverse set of contributors. A healthy community is a good indicator of a well-maintained project.
- Analyze the frequency of releases. Frequent, incremental updates indicate active maintenance. Go for libraries with recent releases over stale, unsupported ones.
- Check for the use of sound engineering practices like input validation, error handling, proper use of encryption, and the principle of least privilege. Avoid projects with insecure designs.
- Check for licensing. Make sure the open source library is licensed under a permissive license that allows for commercial use. Avoid libraries with

restrictive licenses that could limit your ability to use or distribute your software.

Auditing and Hardening Open Source Dependencies

Before integrating any new open source dependency:

- Perform thorough audits of the source code, watching for SQLi, XSS, insecure deserialization, broken authentication, insufficient authorization, and other issues.
- Use SAST, SCA, DAST, and pen testing tools to identify risks, address discoveries in the open source software project itself, and wrap usage to limit the attack surface.
- Evaluate how the dependency fits within the application's secure architecture and modify configurations to optimize for security.

Staying Current with Vulnerability Management

During development and in production:

- Closely monitor mailing lists and security advisories related to all incorporated open source dependencies, regardless of how obscure or popular they are.
- Maintain a comprehensive inventory of dependencies and versions in an SBOM, and continuously scan for newly-disclosed issues.
- Conduct regular security assessments of your open source dependencies to identify and address any potential vulnerabilities. These evaluations can include pen testing, vulnerability scanning, and code reviews.
- Have a remediation plan to rapidly update open source components with security fixes as they are released. Do not let applications lag behind the latest secure versions.

Making Open Source Security a Priority

Integrating open source components securely requires upfront effort and constant vigilance:

- Train your development team on secure coding practices and the importance of open source security. This can help ensure that your team is

aware of the risks associated with unchecked open source usage and is able to proactively manage those risks.

- Make security a primary criterion during open source evaluation and selection, along with functionality, licensing, and support.
- Build a governance process such that all open source libraries can be reviewed before they're added to your application or environment.
- Build security practices deeply into processes for dependency integration, monitoring, alerting, patching, and version update management.
- Know your open source dependencies inside and out. The health of an open source project profoundly impacts its application security posture.

With rigorous processes in place, open source risks can be tamed. The widespread use of open source technology is a double-edged sword, but its benefits outweigh the dangers if open source security is made an ongoing priority.

Supplier Relationship Management to Reduce Software Supply Chain Security Risk

Cassie Crossley

Software suppliers introduce risk to application security, even if proper due diligence is performed before selecting the commercial or open source supplier. Security professionals should know that AppSec is not just about writing security code; it is also about understanding your software dependencies and managing the risks introduced by your software supply chain.

A supplier relationship can either be one-sided, as is the case for open source or a licensed product, or mutual, where you have a contract with the supplier. For the one-sided supplier relationship, you can monitor for patches, updates, and vulnerabilities, but for mutual relationships, there is so much more that can be done to reduce supply chain security risk.

Signing a contract with a supplier is like a marriage, which means the dating process is just as important as the marriage. Unfortunately, suppliers are sometimes selected as quickly as "swiping right"; in other words, the supplier selection may occur without any investigation. Every supplier, however, needs to be examined for all types of risk, because supply chain security risk is much more than cybersecurity or technical risk. The MITRE Corporation's System of Trust is a free supply chain security framework that can help any size organization identify risks in many different categories, including financial stability, ethics, quality, and service resilience.

Risks to AppSec should be carefully reviewed. Evidence for reviews may come from questionnaires, assessments, requests for information (RFIs), certifications, or other information collected from the supplier. For example, a cloud service provider should provide a SOC2 audit report to demonstrate

that an external certified examiner has reviewed the security, availability, processing integrity, confidentiality, and privacy of the platform or service.

By thoroughly examining a supplier—also known as due diligence—the decision to select the supplier is well-informed and intentional. Once you select the supplier, the contract process can be quite intensive and last for months. During this process is when previously identified risks can be discussed and addressed with the supplier—before something happens such as a data breach, security incident, or critical vulnerability. The contract should have clauses specific to the capability (i.e., product, application, or service) that is being provided, the expected outcomes and SLAs, and how the contract's termination should be managed, including procedures for breach notification in case of security incidents or noncompliance with agreed-upon terms. For example, a contract for developer services should clearly specify the secure development life cycle requirements before they start the services engagement and that any source code must be removed from supplier systems at the termination of the contract.

Once the contract is signed and the capability is in use, the supplier relationship is usually forgotten. Regrettably, this is where the risk to AppSec truly begins. If the capability is critical to your organization, then an ongoing supplier management process must be established. The following critical supplier management activities can reduce supply chain security risk to your organization:

- Create a direct relationship between the supplier and your organization's cybersecurity leadership. This can be in the form of a CISO-to-CISO meeting at a regular cadence or between leadership accountable for cybersecurity in the two organizations. An established relationship will be crucial in the event of a data breach, security incident, or critical vulnerability.
- Schedule a review, at least annually, to discuss the supplier's service level, quality, and cybersecurity. The cybersecurity leadership from both organizations should attend in order to maintain the previously established relationship.
- Monitor the supplier for any changes in business health and cybersecurity posture. You can use a separate third-party service, configure monitoring tools, or create Google alerts to receive notifications for analysis and review.

As with any relationship, there might be challenges to overcome, but with ongoing supplier management, you can continuously evaluate the supply chain security risks to your organization. Even in the situation where you might terminate the relationship, the termination decision can focus on the facts and reasons and hopefully will occur without any need for legal recourse.

Since managing third-party risks is very different from identifying and remediating application vulnerabilities, it's recommended to have someone who can focus on this and work with legal and procurement to continuously monitor suppliers.

Fortifying Open Source AI/ML Libraries: Garden of Security in Software Supply Chain

Chloé Messdaghi

In the domain of artificial intelligence and machine learning, open source libraries play a pivotal role—comparable to the essential morning ritual of sipping your favorite coffee or tea. However, navigating the extensive landscape of open source AI/ML libraries isn't a straightforward path; it's like coaxing a squirrel into performing the "Macarena"—full of unexpected turns and fascinating complexities.

Exploring the world of open source AI/ML libraries unveils a multitude of vulnerabilities. These open source projects, much like individual garden plots in a shared space, might overlook potential intrusions. Vulnerabilities range from unpatched bugs to overlooked security risks. Compounded by outdated dependencies, these challenges demand vigilant attention and proactive measures.

To successfully embark on your AI/ML journey, let's dive deeper into the key facets of this expansive landscape, equipping you with essential indispensable knowledge and effective strategies.

Dependency Scanning

Automated dependency scanning and analysis serve as essential tools to mitigate these risks. They function as vigilant sentinels, uncovering vulnerabilities and risks that might otherwise remain hidden within the system. Reactivity won't suffice; proactive measures are critical to preemptively counter threats. Much like diligent gardeners inspecting every plant and shrub for signs of pests or disease, these scanning tools scrutinize every nook and cranny of the code, identifying areas of weakness and potential threats.

CI/CD for AI and ML

The incorporation of CI/CD practices in the AI/ML domain presents unique challenges. Integrating open source AI/ML libraries requires meticulous care. Every phase of the CI/CD pipeline potentially exposes the system to vulnerabilities, emphasizing the need for robust security checks at each stage to avoid potential risks. It's like ensuring that each stage of the garden's growth process, from planting to harvesting, is safeguarded against bugs and environmental factors to ensure the best possible yield.

Software Bill of Materials

The SBOM functions as a comprehensive inventory of software components, including open source libraries. During security incidents, it serves as a guide, expediting identification and resolution efforts. Much like a detailed map leading to a crucial discovery, the SBOM accurately directs attention to areas requiring immediate focus and resolution. Think of it as a detailed record of the entire garden's species and plant health, helping gardeners swiftly identify areas requiring immediate attention or care.

Auditing and Verification

Transparent audit processes and verification mechanisms differentiate secure components from potential threats. Audits function as evaluators, identifying vulnerabilities, while verification mechanisms confirm the authenticity of secure components. The objective is to ensure that open source AI/ML libraries claim their deserved spotlight, devoid of vulnerabilities. It's akin to having expert horticulturists and botanists examine each plant, ensuring they are healthy and free from any intruders or diseases before showcasing them in the garden exhibition.

Community Collaboration

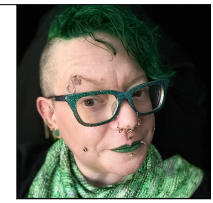
The open source AI/ML community thrives on collective teamwork. Through unified endeavors, the community ensures seamless operations, whether through coordinated vulnerability disclosure programs or fostering a culture of security awareness. Much like a team of experienced gardeners collaborating to maintain a beautifully landscaped garden, the community works together to ensure that the shared environment is secure and thriving.

Securing open source AI/ML libraries in the software supply chain necessitates proactive measures, much like setting up defenses to safeguard a prized garden against potential risks. Leveraging tools such as automated scanning,

CI/CD integration, SBOMs, audits, and community collaboration is akin to establishing digital barriers within this virtual garden. It's an endeavor to protect against vulnerabilities and threats, where these protections act as a shield to ensure secure data, similar to safeguarding valuable persimmon trees from potential squirrels.

SBOM: Transparent, Sustainable Compliance

Karen Walsh



Modern, interconnected application ecosystems are much like coral reefs. A single vulnerability can disrupt the delicate, symbiotic balance between open source code and application security, poisoning entire systems. Just as regulatory compliance sought to solve physical pollution, governmental initiatives seek to limit digital pollution by establishing rules for supervising digital contamination. As compliance initiatives start focusing on supply chain security at the code and component levels, AppSec professionals will increasingly be held accountable for maintaining a software bill of materials (SBOM) to achieve the organization's compliance objectives.

Building Transparency

Essentially, the SBOM is the software equivalent of the Nutrition Facts Label on packaged food. With visibility into ingredients, organizations can make informed, healthy decisions so they can monitor applications for vulnerabilities impacting the components.

In 2021, the National Telecommunications and Information Administration (NTIA) updated their publication, *Framing Software Component Transparency: Establishing a Common Software Bill of Materials (SBOM)*. NTIA defined SBOM as:

> a nested inventory, a list of ingredients that make up software components… [that] identifies and lists software components, information about those components, and supply chain relationships between them.[1]

1 NTIA Multistakeholder Process on Software Component Transparency Framing Working Group. *Framing Software Component Transparency: Establishing a Common Software Bill of Materials (SBOM)*, 2nd ed. (NTIA, 2021).

The NTIA noted that software supply chain transparency could increase trust and trustworthiness while reducing cybersecurity risks and overall costs.

Designing Sustainably

Through transparency, the SBOM enables AppSec professionals to design cyber-sustainable systems.

Vulnerabilities are the digital equivalent of a chemical spill. A look at the ripple effect that Log4J vulnerabilities caused across organizations and their ecosystems is a prime example. Although identified in December 2021, the Cybersecurity & Infrastructure Security Agency (CISA) updated its publication, "Apache Log4j Vulnerability Guidance" in April 2022, noting organizations should plan for "long-term vulnerability management," including "newly vulnerable third-party software" because they may "lack insight into certain applications."[2]

By incorporating SBOMs into their workflows, developers build cyber resilience and sustainability into their processes with insights that help them identify potential system pollution.

Developing Compliantly

Despite having ten letters, *compliance* is often considered a four-letter word. As executive, legislative, and agency bodies seek to mitigate the digital pollution arising from open source components, compliance mandates, and national strategies increasingly turn to the SBOM as their solution.

Over the last few years, laws, draft legislations, and strategies incorporate SBOMs as supply chain security risk mitigation tools. Some examples of these include:

German Security IT Act 2.0 Section 9b(3)

> Requires manufacturers of critical components to provide a declaration of trustworthiness showing that the critical component does not have a technical property that can undermine critical infrastructure security, confidentiality, integrity, availability, or functionality

2 CISA, "Apache Log4j Vulnerability Guidance," CISA (April 8, 2022).

Proposed European Cyber Resilience Act (CRA) (2022), Annex 1, 2: Vulnerability Handling Requirements

Requires manufacturers of products with digital elements to identify and document vulnerabilities and components in a product by drawing up an SBOM

US National Cybersecurity Strategy Strategic Objective 3.4

Incentivizes secure software development practices by promoting further development of SBOMs[3]

The Future of Secure, Compliant Application Ecosystem

As security shifts left, compliance mandates will follow. While compliance will never equate to security, regulatory and legislative bodies will continue to use the stick of compliance violation penalties rather than the carrot of rewards as they attempt to establish a set of security best practices. AppSec professionals will likely find that SBOMs become the required audit documentation essential to their jobs.

3 The White House, *National Cybersecurity Strategy* (Washington, DC, 2023).

Secure the Software Supply Chain Through Transparency

Niels Tanis

At the beginning of my development career, I developed applications that were solely built and used inside of an organization and run on premises. That's a big difference compared to today where software might be built and released several times a day and deployed to a cloud-based infrastructure. The development process has become a lot more complex. Starting with the source and then going through different stages like building, testing, and deploying artifacts (such as binaries or containers) can be referred to as the software supply chain.

Because the whole process has become more complex, and even turned into the software itself, it has become a main target for security attacks. This is because its overall attack surface has increased. For example, it can vary from stolen Git credentials from Canonical (the people behind the widely used Linux Ubuntu distribution) to the compromised build servers in case of the SolarWinds supply chain attack. In the latter case, it even resulted in the release of a product that was an authentic cryptographically signed installation to SolarWinds customers. It took some time and investigation to realize the hack happened, and people with malicious intent added additional functionality to it.

To secure our supply chain of applications in a better way, there is a need to better understand what is inside and how it was built from source to deployed binaries. An industry that solved this problem well is the automotive industry. On average, a car that gets built from scratch will have 10,000 parts manufactured by 3,500 different suppliers. To keep track of what's been put in a car, a bill of materials (BOM) will describe which different identifiable parts are used in which car. That's why recalls of certain cars happen to

replace parts that turned out to be of bad quality or have a design flaw that will make the car less safe.

In the case of software, we can achieve the same with the SBOM. On average, our software consists of 80% of third-party libraries. For example, with a project called CycloneDX, it's possible to create an SBOM that describes the full transitive tree of dependencies used. It supports a large list of different technology stacks. When you rely on containers, Docker even has built-in support for generating an SBOM of a Docker image. Having this kind of information available gives the ability to act later if a vulnerability gets disclosed in one of the used components. That problem could even originate from a compromise of the supply chain of the component itself, because by using it, you extend your own supply chain with the one used to produce the component.

When you're aware of what's inside the build software, another good addition is the build provenance. Build provenance of software describes details of the build process, such as what materials were consumed, what build parameters were set, and which source was used to build the software. For example, if the software is built on Tekton, it can be achieved with the help of Tekton Chains. On GitHub Actions, Google's SLSA GitHub Action also creates non-forgeable provenance with the help of a project called Sigstore. This cryptographically signs the output and allows you to check its origin, who produced the provenance file, and whether it was altered.

With this information at hand, you can create transparency by knowing what's inside software and how it's being built. This will certainly not prevent any supply chain incident from happening, but it will give you the right information for handling the incident properly!

Unlock the Secrets to Open Source Software Security

Travis Felder

Open source software (OSS) refers to software whose source code is available to the public, allowing anyone to view, use, modify, and distribute the software. While OSS offers numerous benefits, such as cost savings, flexibility, and collaboration, it also raises security concerns. Historically, proprietary software dominated the market, with developers closely guarding their source code. However, the open source movement emerged in the 1990s, advocating for the free sharing of software code. OSS has since grown exponentially, with major projects like Linux, Apache, and PostgreSQL becoming industry standards.

Invisible Open Source Software

Neglecting OSS security in enterprise environments can expose organizations to a range of risks, including increased vulnerabilities, compliance failures, and reputational damage. The exploitation of these vulnerabilities by malicious actors can lead to unauthorized access to sensitive data, operational disruptions, and intellectual property theft.

As a result, organizations may face significant financial, legal, and reputational consequences. To mitigate these risks and fully harness the benefits of OSS, enterprises must invest in a comprehensive security program that actively manages and addresses security risks while adhering to industry best practices.

Establishing an OSS Program

Establishing an OSS program involves the following:

- Establish a security policy for OSS use and development.

- Assess the security of the OSS project, including its development process and history.
- Conduct regular security audits of OSS code.
- Implement a patch management system to address known vulnerabilities.
- Train developers in secure coding practices and the secure development life cycle.
- Monitor and analyze OSS usage to identify potential security threats.
- Participate in OSS community forums to stay up to date on security best practices and emerging threats.

Open Source Software Security Pro Tips

Choose mature, widely used OSS projects with active communities. Selecting mature and popular open source projects is crucial for ensuring a strong security foundation. These projects benefit from extensive community involvement, which includes regular code reviews, vulnerability identification, and patch submissions. An active community indicates a higher likelihood of rapid security updates and improvements, resulting in more reliable and secure software for your organization.

Consider using automated tools for vulnerability scanning and patch management. Vulnerability scanners can help identify potential security risks in your open source components, while patch management tools can streamline the process of applying security updates. By utilizing automation, your organization can stay up-to-date with the latest security patches and quickly address vulnerabilities, reducing the risk of exploitation.

Common Open Source Software Security Mistakes to Avoid

One of the most common mistakes in open source software security is failing to regularly update components and apply security patches. This oversight leaves systems exposed to vulnerabilities that malicious actors can exploit.

Relying solely on community-driven security measures without conducting internal security audits is an issue to avoid. While the OSS community plays a crucial role in identifying and addressing vulnerabilities, organizations should not depend solely on external security efforts. Conducting internal security audits is essential for uncovering potential risks specific to your organization's implementation of the software.

Failing to establish a clear security policy for OSS usage within an organization is a common mistake. Without a clear security policy, organizations may struggle to consistently manage and secure their open source software components.

Investing in a comprehensive OSS security program not only enhances the security posture of an organization but also fosters a culture of collaboration and innovation. By embracing a proactive approach to OSS security and staying up to date with best practices, organizations can confidently leverage the benefits of open source software while ensuring the safety and integrity of their systems and data.

Leverage SBOMs to Enhance Your SSDLC

Viraj Gandhi

In today's increasingly digitized and interconnected world, data breaches have become a pervasive and costly concern, affecting businesses, governments, and individuals alike. These breaches can lead to devastating consequences, including financial losses, reputational damage, and compromised personal information. The rapid adoption of technology, coupled with the interconnected nature of software applications, has created a vast attack surface for cybercriminals to exploit. As a result, organizations are increasingly focusing on strengthening their security postures from the very beginning of the software development life cycle; therefore, the importance of secure software development life cycle (SSDLC) practices cannot be overstated.

The SDLC is a systematic approach to designing, developing, testing, and maintaining software applications. Integrating security measures into each phase of the SDLC helps identify vulnerabilities early in the process, reducing the likelihood of introducing security flaws that could be exploited by malicious actors. By implementing SSDLC practices, organizations can proactively address security concerns, thereby minimizing the risk of data breaches and other cyber threats.

One of the emerging challenges in software security is supply chain risk. Organizations often rely on third-party vendors and components to build their software products. However, this reliance introduces a potential weak link in the security chain, as vulnerabilities in third-party components can be exploited to compromise the entire software ecosystem. Recent high-profile supply chain attacks have highlighted the need for robust mechanisms to assess and manage the security of these components. This is where the SBOM comes in, which plays a pivotal role in enhancing transparency, traceability, and overall cybersecurity. An SBOM is a comprehensive inventory of all the components and dependencies used in a software application. It provides a detailed list of the software elements, including libraries, frameworks,

and modules, along with their versions and sources. This transparency enables organizations to gain better visibility into their software supply chain, identify potential vulnerabilities, and take appropriate remediation actions.

The SBOM serves as a critical tool for risk management and decision making. It empowers organizations to assess the security posture of the software components they rely on, ensuring that any known vulnerabilities are promptly addressed. Furthermore, an SBOM facilitates effective communication between developers, security teams, and third-party vendors, fostering collaboration and accountability in maintaining software security. The significance of the SBOM has been underscored by regulatory initiatives and industry standards. In the US, for instance, Executive Order 14028: Improving the Nation's Cybersecurity, emphasizes the importance of SBOMs in enhancing software supply chain security. Similarly, NIST has published guidelines that advocate for the integration of SBOMs into software development and procurement processes.

In conclusion, the escalating threat landscape marked by data breaches and supply chain vulnerabilities necessitates a proactive and comprehensive approach to software security. Secure SDLC practices, integrated with the principles of transparency and traceability championed by the SBOM, offer a potent strategy for safeguarding software applications against cyber threats. By embedding security measures throughout the software development life cycle and leveraging the insights provided by SBOMs, organizations can significantly reduce the risk of data breaches, mitigate supply chain vulnerabilities, and ultimately fortify their overall cybersecurity posture. As the digital landscape continues to evolve, embracing these practices will be essential for building resilient and secure software ecosystems.

PART VII

Threat Modeling

Learn to Threat Model

Adam Shostack, Matthew Coles, and Izar Tarandach



Every application security professional should know how to threat model. It doesn't have to be a big or complex process. The *Threat Modeling Manifesto* says threat modeling focuses on four simple questions:

1. What are we working on?
2. What can go wrong?
3. What are we going to do about it?
4. Did we do a good enough job?

We ask, "What are we working on?" to focus our attention and scope our analysis. Some people ask, "What are we building?" and accidentally make a waterfall threat modeling process. Some people choose to "threat model every story," which is another approach to scoping and determining which stories have security value and which do not.

The second question, "What can go wrong?" is the heart of threat modeling. It is hard but essential, and you need to involve people with different perspectives. You can also use structured approaches, like STRIDE or kill chains, to be more systematic in which threats you discover. Free flow or unbounded approaches like "think like a hacker" prove less effective and sometimes daunting; if you don't know how a hacker thinks, how are you supposed to emulate the process? A structured process offers scaffolding to get you through the first iterations.

Identifying "what we're going to do about it" makes threat modeling, as a whole, valuable by addressing the problems identified; to not do so is like driving to a foreseeable car wreck. Fixing things often translates to "build new features" or "change the design" (which sometimes may include your deployment model and choices). Sometimes that's reasonably obvious and we don't need to quantify or prioritize in a security-specific way: the bucket holding data should be encrypted by default. Other times, the answer is more

complex, and contextualizing the threat in a way that risk management processes can more easily ingest (probability, impact) becomes more important.

Asking, "Did we do a good enough job?" is crucial to ensure you reflect as you do the work. There's more than one way to bake a cake, after all, so it's important to know if the way you're doing it is the best way for you. Another way to look at this is to ask, "Did the things we did about the threat improve the system?" and "Are our teams getting as much value from threat modeling as they can?"

Many organizations want to do threat modeling, but it's best when we focus on the question: "Is this adding value to our development practices?" A retrospective may provide the answer. If the people involved are not extracting actionable items or at least learning more about security and about their system, then there is no value to the process, and it should be reevaluated, changed or dropped. But more likely, threat modeling will help elevate the security posture, and the overall quality, of a system.

Threat modeling is a way of considering the security effects of your choices, and tools help you do that and help a team show that it's been done.

Understanding OWASP Insecure Design and Unmasking Toxic Combinations

Idan Plotnik

In the constantly evolving world of application security, we are seeing unprecedented challenges that traditional testing tools like SAST, SCA, and DAST alone cannot solve. They're unable to keep up with the pace of Agile development and detect the new breed of interconnected, nuanced, and varied risks that modern applications face.

Two often overlooked mechanisms for improving the efficacy and efficiency of application security are the ability to programmatically identify potential design flaws before code has even been written and the ability to connect the dots between disparate types of application risks. The former helps AppSec teams be more proactive and prevent ad hoc work down the line, while the latter helps close the gaps left by siloed tools.

Understand the Implications of Insecure Design

A testament to our dynamic threat landscape is the addition of OWASP A04:2021—Insecure Design as a new category to the OWASP Top 10 in 2021. This signals the need for more proactive and scalable defense strategies to identify application risks stemming from inherent design oversights.

Incorporating threat modeling during the feature design phase and integrating Agile pen testing during feature development is now essential to align with OWASP A04:2021 but faces scalability challenges due to manual processes and AppSec professional-to-developer ratios.

The only way to proactively handle risky feature requests and code changes is to shift security even further left to the design phase, ensuring ongoing visibility throughout the development cycle. Programmatically identifying

insecure design issues triggers threat modeling and pen tests, optimizing limited security resources. Prioritizing design risks based on application context and business needs prevents developer overload from manual forms or questionnaires.

The top five examples of risky changes that you should focus on while running threat models or Agile pen testing:

New endpoints
: Introduction of new APIs, Protobuf services, serverless functions or other services, especially without appropriate authentication or validation.

Changes in data flow
: New data models, data access objects and modifications in how data is received, processed, stored, or transmitted.

Altered trust boundaries
: Integrating new third-party components or changing data sharing with external entities.

Misconfigurations
: Leaving debug modes on, unprotected sensitive files, or opening unnecessary ports.

Deprecated libraries and/or downgraded OSS dependencies
: Using outdated software libraries or reverting to an older OSS dependency version due to compatibility issues can expose applications to known vulnerabilities, enlarging the attack surface.

Unmask the "Toxic Combinations" in Application Security

The industry is awash with guidelines and frameworks to detect vulnerabilities, weaknesses, misconfigurations, or risky changes in complex modern applications expanding across application code, APIs, OSS, IaC, containers, and even source control managers and CI/CD pipelines.

When vulnerabilities or other risky changes—that may or may not have been a serious risk on their own—when connected, they can manifest into what is known as a *toxic combination*, creating a whole new attack path or exponentially amplifying an existing risk.

Moreover, siloed alerts hinder the identification of toxic combinations, demanding interconnected evaluation for better insights to identify,

prioritize, and remediate them before delivering the application to production. Some examples of toxic combinations are:

- Valid secrets that are being used to access AWS resources in a public repository that contains a new API that writes PII to a storage bucket
- An internet-facing API that exposes PII data, missing input validation that uses OSS dependency with a new known exploit
- CI/CD pipeline misconfiguration that builds code in a repository that contains an API that exposes PII data and abnormal developer commit behavior was identified

To help identify toxic combinations early within the SDLC, it is crucial to have a continuous code inventory with all code components and their relationships, map the application architecture as a graph, and augment these siloed alerts on top of the graph. This is the only way we can identify these toxic combinations.

The Right Way to Threat Model

Josh Brown

Threat modeling is a critical part of application security, and it is a proactive and structured process used to identify and measure risks associated with a system to determine any needed design changes or risk mitigations. There are many different risk assessment (RA) frameworks and threat modeling techniques. Most are incomplete and contradict the wider concept of security risk programs for an organization. There is not one correct framework to be used for all RAs and threat models. If you understand which one to use for different situations, you can build a healthy RA and threat modeling program.

The following are indicators of a healthy RA and threat modeling program:

- It will *not* be a blocker.
- It will keep pace with high-velocity Agile development/project teams by contributing vetted architecture blueprints to the organization's library for future use. This increases developer velocity by pulling from these vetted designs. This also speeds up design reviews. This lowers costs by reducing development effort and waste.
- It will reduce effort related to security incidents, mitigations, audit responses, and certifications.
- It will, most importantly, build trust between security and other departments.

Properly scoping questions are the most important part of threat modeling engagements. The following are tips on how to properly scope questions:

- Don't ask questions about things that could change tomorrow. Those items are best handled by other *continuous* operations processes or capabilities. For example, organization-wide infrastructure, operating system hardening, patching, vulnerabilities, etc.

- Don't ask this project team about systems that are not theirs. Instead, ask the team that "should" know how it works, such as authentication directories, etc.
- Don't ask detailed questions about things that haven't been built yet. You can ask about a major library if you need to, but logical questions and minute details are not fruitful in the design phase of a project, such as workflows, logic protections, application throttling, etc.

Data flow diagrams are everywhere and…they're terrible. These are not useful for security as they don't reflect the system's actual attack surface. As an example, you'll see "trust boundary" as an abstract defensive layer. The risks are completely different if that trust boundary represents a Windows server on the internet versus an AWS load balancer and WAF. So, make sure that you reflect the shared security responsibility model of all systems.

These tenets offer a robust framework for integrating security into system design and maintenance, guiding developers and infrastructure teams, including those managing their infrastructure. By adhering to these principles at each development phase, teams can effectively avoid common security pitfalls and enhance overall system resilience.

Developers:

Tenet	Phase
Use a mature authentication system	Design
Use a mature authorization system	
Use a mature session management system	
Use good secrets management	
Add application throttling for all inputs	Development
Use mature input sanitization libraries for all inputs	
Use mature crypto libraries	
Use automated documentation practices	Testing
Use the latest version of dependencies	Maintenance
Continuous training on secure coding practices	

Infra team:

Tenet	Phase
Maintain optimal blast radius	Design
Enforce TLS 1.2+ on all data flows	Deployment
Maintain security agent installs	
Maintain mature accounting operations	
Encrypt data at rest	
Use least privilege design	
Tune boundary devices	Maintenance
Harden all computation systems	
Maintain mature authentication systems	
Maintain attack surface reduction operations	
Maintain proper secrets hygiene	

In summary, threat modeling is not a one-size-fits-all approach. A healthy RA and threat modeling program aligns with Agile development, offering vetted architectural blueprints that speed up design reviews and reduce costs. Key to threat modeling is asking the right scope of questions, focusing on the project at hand rather than external or future variables. I caution against the misuse of data flow diagrams but rather urge teams to reflect the actual attack surface. Using role-based tenets is a simple, succinct guide to use through different phases of a project, aiming to build shared understanding and trust between security and other departments.

Attack Models in SSDLC

Vinay Venkatesh

For a long time—at least since the start of my product security career—Secure Systems Development Lifecycle (SSDLC) has followed a four-step process:

1. Define product security requirements at the beginning of the project.
2. Threat model the product and identify security controls.
3. Configure scanners to identify vulnerabilities in code as well as in open source packages it uses.
4. Perform pen testing to confirm that the product is free of vulnerabilities or to identify and address security weaknesses before deployment.

While this process provides a lot of feedback, it is missing one key ingredient—there is no mechanism between threat modeling and code scanning to perform a security analysis of the detailed design. Some have addressed this by expanding the threat model to include component-level details. The usefulness of this approach is limited because threat modeling, by nature, is an architectural exercise where we model independent, interacting processes and data flows between them to identify security gaps. Detailed analysis involves a closer look at how the system should behave in every possible situation and for all possible inputs. I've found attack modeling to be a better fit here.

Attack modeling is not a new concept. It was first introduced by Bruce Schneier in his 1999 paper, "Attack Trees."[1] Since then it has been used to model different aspects of attacker behavior and associated impact. A few years back, Kelly Shortridge wrote a blog on building security decision trees,[2] which was later incorporated into the book *Security Chaos Engineering*

1 Bruce Schneier, "Attack Trees," *Schneier on Security* (website), December 1999.

2 Kelly Shortridge, "Creating Security Decision Trees with Graphivz," *Kelly Shortridge* (blog), January 25, 2021.

(O'Reilly, 2020). For the remainder of this essay, I am going to call her approach *attack-defense modeling* (ADM).

ADM encourages us to think from an attacker's perspective and capture all possible attacks and defenses. It can be built using a diagramming tool or using some specialized as-code tools. The as-code tools can generate the diagrams and enable semantic analysis of attacks and defenses. Immaterial of the tool we choose, the first step is to create ADMs for every entity and flow from your project.

Attacks typically involve a set of steps that form one or more cyber kill chains. After all possible attack chains are defined, defenses that would mitigate each attack step from each chain should be defined. Sometimes, there may not be any defensive design that would mitigate the attack. In such cases, a monitoring or incident-response procedure can be defined as a reactive defense.

When analyzing a system, you should create individual ADMs for each entity and technology used in that system. When possible, it is a good idea to create ADMs for each external package used in the product. These can be built using information from security bulletins and CVEs associated with each package. In this case, ADMs should capture how attackers can exploit these vulnerabilities within the context of the product and what defenses the product needs to mitigate them. At the end, you will have a bunch of ADMs that can be copied to a central knowledge base for other security engineers in your company to use.

Over time, the knowledge base will contain enough detailed information to help everyone from a junior security engineer to the most senior analyst. The knowledge base itself can be a folder full of specification documents along with some code examples for clarity. Or, if you are using an as-code tool, code files can be stored in a central location like a shared drive or a git repository.

As of 2023, there are a few as-code tools available to build an ADM. Kelly's Deciduous.app uses YAML to capture attacks and defenses and generates the graph in Graphviz, PNG, and SVG formats. I wrote another tool called adm where attack and defense information is captured using a variant of Gherkin language. The tool generates security decision graphs in Graphviz format. This can then be loaded into graph processing tools for further analysis. A companion tool, adsm, lets you associate ADMs to each entity and flow from the threat model and generate a comprehensive attack-defense report from it. Adsm also includes support for ADDB, the attack-defense database, which

is an external knowledge-base folder containing ADMs and entity specifications.

PART VIII

Threat Intelligence & Incident Response

In Denial of Your Services

Allen West

The attack of disclosure to discourage further snooping is somewhat of a tradition now within the cybersecurity world. Old-timers in the vulnerability research world can tell you that the ecosystem of getting paid for responsible disclosure of findings used to be much riskier than it is today. Bug bounty was designed for that. Without formal bug bounty programs, let alone bug bounty hosting platforms like we have today, it was often a toss-up of how an organization would react to the disclosure of vulnerabilities in their systems. Many times, companies that were unsure how to handle situations like this would often resort to legal action against the researcher, which inevitably led to fear of disclosure and community backlash to the responding company. A lose-lose scenario.

Over time, most companies have come to appreciate the contributions that freelance vulnerability hunters provide and have tried to formalize the process, laying out clear guidelines of what they do and do not want researchers to try when testing their applications. Some common off-limit items include brute forcing, social engineering, purchase of compromised credentials on the dark web, and commercial vulnerability scanners, just to name a few. Basically, anything that would incentivize misbehavior or interrupt actual business.

Security decisions are often made based on risks. In many situations where the perceived risk of a specific vulnerability may be lower than the actual risks revealed through vulnerability testing. Denial-of-service (DoS) vulnerabilities are the perfect example of this. Almost every single bug bounty program on platforms like HackerOne or Bugcrowd puts DoS vulnerabilities out of scope. This is because vulnerability disclosure requires a repeatable proof of concept, and in order to do this for a DoS vulnerability, you must deny service, which often costs the company money. The risk of this potential vulnerability existing is therefore necessarily lower than a positive disclosure, which would guarantee an interruption in service.

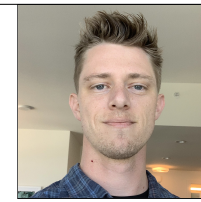
There are, however, still plenty of nondestructive ways to test for DoS vulnerabilities, such as Nmap Scripting Engine's test for slow loris–style attacks. The scan tests whether conditions required for the attack to be successful are present in the nonintrusive interactions with a server, and then indicates a positive if they are found. In this case, however, a report of potential vulnerability is still not appreciated by companies, due to the fact that they don't want to spend time testing "maybe vulnerabilities" without a proof of concept (PoC) attached to them.

This is further exacerbated by the fact that disclosure reports are often triaged by the platform's analysts, not the actual security engineers at the company you are testing. These analysts have the luxury of following a given organization's scope to the letter when looking at reports, often simply throwing disclosures out of scope without forwarding them to the impacted company at all. It is also possible that since the researcher strayed from scope, they could potentially lose platform reputation points as well, which can impact future opportunities, and sort of bring us full circle to the old days of vulnerability disclosure.

The takeaway here is that although the ecosystem of responsible vulnerability disclosure has dramatically improved for the average researcher, there are still gaps in our strategy to handle the touchier ones. If compromised credentials go unreported, they still pose a significant risk. If a DoS vulnerability exists in a platform, it is possible that analysts won't be able to figure it out quickly when a threat actor takes advantage of this gap in defenses. Therefore, while security researchers must respect the laws and wishes of the organizations they are testing, companies must thoroughly vet every risk they are presented with, come up with internal or contracted ways to test for the riskier security gaps, and reward responsible disclosure with the respect and appreciation it deserves.

Sifting for Botnets

Allen West



Squeezing original intelligence from the unknown can be accomplished through the process of methodical enrichment of application logs. Common Vulnerabilities and Exposures (CVE) are a great way to track cybersecurity vulnerabilities that most security professionals should already be familiar with. This standardization allows for conversations to be had around important weaknesses within specific applications, but not all vulnerabilities are known, and not all known vulnerabilities get CVE. The question then becomes, How do you position yourself to defend threats that have not been officially recognized by the security community?

As soon as you publish virtually any application on the internet, you quickly start seeing an overwhelming amount of unsolicited traffic. The internet is full of malicious actors, bots (often programs that perform automated actions for their owners, but sometimes in the context of botnets, they can be compromised endpoints), researchers, and organizations that are constantly scanning the entire internet for various purposes. It is up to defenders to stay current with threat trends, especially for applications, which are designed to be accessible, often at the expense of security principles. This can be accomplished by paying close attention to application traffic logs, using good automation or even AI to find unique crumbs, giving you a better understanding of the current threat landscape as it pertains to your specific applications' technology stacks.

Filtering down the background noise of the internet is unfortunately not as simple as classifying the interesting and ignoring the uninteresting. The sheer volume is astounding, and what adds to this is widespread brute forcing, numerous exploit variants, credential spraying, application fingerprinting, and, of course, a lot of garbage. When combing through logs, however, we can use some reliable tactics to make this mess a lot more manageable.

For starters, deduplicating and saving the date you first observed a given request will be a massive time-saver. It also may make sense to have a minimum length of content to trigger analysis, as 2–3 character exploits are not

feasible. You can also require that at least a portion of the request appears within a dictionary of sorts to get rid of spam. From here, you can set up regex filtering to eliminate exploit variants, and you may even consider implementing a system to generate these regex patterns automatically. With these filters and others, you can start attempting to attribute requests, isolating the interesting yet unknown artifacts.

With traffic attribution, you will want to at least focus on targeted technologies, and known CVE if possible. You can incorporate third-party threat intelligence tools, automated web searches, regex patterns, open source vulnerability scanners scraping, vulnerability databases, and API calls to AI-based systems such as ChatGPT. If we can deduce the CVE, we will know the targeted technology, and if not, we can still try to identify if the traffic is relevant to our tech stack. What you will be left with is brand-new, non-CVE, uncommon traffic that is likely to be targeted and relevant. Vetted unknowns in this context have the potential to be exploits flying under the radar, requests targeting little-known misconfigurations, or even zero-days. This sort of intelligence is crucial to an organization, as it allows it to make educated fixes, stay in tune with relevant threat actors, and prioritize follow-on efforts.

As a proof of concept, I developed a tool utilizing this methodology to filter down honeypot cluster traffic, and it had some impressive results. The intelligence gathered allowed us to prioritize the creation of targeted honeypots, which shifted our focus to non-CVE exploits of misconfigurations in common applications. By focusing on these weaknesses, we were able to unearth multiple high-performing botnets that directly targeted customers and industries we protect. We launched full investigations and subsequently published articles and gave conference talks on them. Most importantly, however, we were able to develop controls and recommendations that improved our defensive posture. This cemented in my mind the importance of generating original intelligence through available resources when possible through meticulous enrichment and smart filtering.

Incident Response for Credential Stuffing Attacks

Fayyaz Rajpari

In today's remote world with defined perimeters, the cloud, and microservices, understanding how to respond to unique application attacks can be impactful to any organization. As an application security professional, it is important to understand and apply an incident response (IR) framework, such as the one from the SANS Institute.

First, when a software vulnerability is exploited to gain access to systems for malicious intent, it is known as an application security attack. The problem gets worse when an attacker gains access to credentials via exploiting a vulnerability and then compounds it with another attack. A credential stuffing attack is a good example of this because it involves stealing compromised credentials gained from an exposed database. Although it's bad security hygiene, it is common for people to use the same passwords for numerous applications. The attacker will always use this to their advantage. Once an exploit is found and used against the vulnerability, numerous credentials are exposed on the database. The attacker then applies the username and passwords to hundreds of websites that the victim frequently visits.

Let's say that an adversary has successfully exposed a database and has stolen credentials from that database to access other websites that share those credentials. Furthermore, they discover that those credentials can be used on other systems with that environment. Examples can be transactional systems, S3, or Azure blobs holding confidential data.

Now let's apply the six steps of the SANS IR life cycle to a credential stuffing application attack:

1. Preparation

   The preparation step involves having a plan regarding how and when to classify an incident before it has occurred. A great example of this can be seen in the Florida Department of Transportation's IR plan.

2. Identification

   Identification involves detecting deviations from normal operations. This gets easier with proper logging in place. It is just as important, if not more, to look at successful logins than analyzing log failures for brute force attacks. In the credential-stuffing application attack scenario, successful logins highlight compromised activity from an illegitimate user!

   It's important to do a sweep of the compromised accounts and check all system logs that show signs of attempted, successful, and failed logins after the date of the breach. You may find more logins and other systems for illegitimate access to systems during the response efforts of identification.

3. Containment

   Containment is normally thought of as disabling network access to systems so that communications are severed in the case of a bad actor communicating outbound or exfiltrating data. It can also mean disabling user accounts. In the credential-stuffing scenario, containment is done via disabling access to the compromised users and/or resetting passwords. Disabling network access is not always the best route. If internal systems' network access is cut off and the attacker is operating from a system in Russia, the damage continues since access to public web applications is still open!

4. Eradication

   Eradication involves the removal of any malicious software from systems so no further damage is done. In the credential-stuffing scenario, actual systems may not be impacted, but the database and backend systems are the culprit. What software was running that allowed the database to get breached in the first place? Was it an older version, misconfiguration, network? These are all important questions to ask. And also, make sure that any software versions are not only up to date but also that all versions of security software used for analysis are also on the latest version in case any vulnerabilities exist.

5. Recovery

   It should go without saying—patch, patch, and patch! In the credential-stuffing situation, an application vulnerability existed. Compromise also is possible via misconfigurations, simple passwords, or loosely defined policies that let the intruder in.

6. Lessons learned

   Any good incident response is never complete without studying the lessons learned from the incidents. Always make sure to have a postmortem! This includes what, how, where, and when the compromise occurred. What steps were taken, and how can the organization improve its process in the response? This will call out various gaps in your current processes and capabilities and will help you prioritize the work to improve your secure posture!

Advanced Threat Intelligence Capabilities for Enhanced Application Security Defense

Michael Freeman

In the digitally transformed world, the security of applications is constantly threatened by many sophisticated cyberattacks in the ever-evolving cyber landscape. Among the most valuable defenses an organization can implement is a comprehensive threat intelligence program capable of providing actionable intelligence on new Tactics, Techniques, and Procedures (TTPs) and vulnerabilities being exploited in the wild. Developing these advanced capabilities is critical for an organization to proactively detect, prevent, and respond to these threats while protective and offensive security controls are also developed for those applications.

Advanced threat intelligence capabilities for AppSec start with the intelligence life cycle, which starts with the question, What application security flaws are being exploited right now? That will drive your collection and analysis of information from a multitude of sources. These include open source intelligence (OSINT), cyber threat intelligence feeds, industry-specific threat-sharing groups, and even the dark web.

Leveraging AI and ML technologies can help target the correct sources, automate the analysis process, and give you actionable and timely intelligence of what to prioritize.

The first facet of actionable intelligence is understanding new TTPs. Cyber adversaries constantly innovate their attack strategies to evade detection and maximize damage. By staying abreast of new TTPs, an organization can adjust its security posture and defenses to match these evolving threats.

A robust threat intelligence program should include a detailed analysis of the TTPs used by threat actors and provide actionable insights on how to

counter them. For instance, if a new exploit for a software library was discovered being exploited in the wild, with actionable intelligence, the organization could quickly identify it in many applications in their environment and address it.

The second part of actionable intelligence is identifying vulnerabilities exploited in the wild. Vulnerabilities represent one of the primary attack vectors for cyber adversaries. Remediating an organization's own vulnerabilities is absolutely needed. However, recognizing which vulnerabilities are being actively exploited allows an organization to prioritize its patch management and application vulnerability mitigation efforts effectively. For example, suppose the vulnerability management team has a list of thousands of vulnerabilities to address, knowing which ones are being exploited will help prioritize those particular vulnerabilities first.

Effective asset management is imperative for organizational security. Beyond simply gathering and evaluating data, entities must execute promptly on insights gained. Seamless integration with robust asset management solutions facilitates swift action. These systems empower organizations to leverage actionable intelligence efficiently, automatically implementing necessary adjustments across their security infrastructure. For instance, upon detecting a malicious IP address through threat intelligence, the asset management solution can swiftly block it across all firewalls, enhancing response time and mitigating the likelihood of successful cyberattacks.

Ultimately, advanced and targeted threat intelligence capabilities aim to understand the threat landscape from both external and internal and to leverage that intelligence to be proactive to prevent future attacks. By using actionable intelligence on new TTPs and vulnerabilities being exploited in the wild, organizations can quickly move from a reactive security posture to a proactive one, staying one step ahead of the threat actors.

In conclusion, developing advanced threat intelligence capabilities requires an integrated approach that combines technology, processes, and people. It is about collecting the correct data, analyzing it for actionable insights, and acting upon those insights swiftly and effectively. Together with other application security controls, this process will ensure organizations can keep pace with the ever-evolving threat landscape, safeguarding their assets and ensuring business continuity.

PART IX

Mobile Security

Mobile Security: Domain and Best Practices

Aruneesh Salhotra

The inaugural iPhone marked a pivotal moment in mobile application evolution, introducing groundbreaking hardware innovations, a user-centric interface, and the centralized App Store. Today, mobile devices and applications are integral to businesses' digital presence, fostering global connections.

As application usage surges, addressing security becomes imperative for ensuring business, revenue, and PII protection. This involves comprehensive safeguarding of mobile applications to mitigate potential disasters. Enterprises rely heavily on these applications, making risk management a top priority. In a constantly evolving landscape, proactive security measures are crucial. In summary, the iPhone's debut reshaped mobile technology, and now, application security is paramount for businesses to thrive and secure their digital foothold.

Mobile applications commonly grapple with recurring challenges, which include the following:

Inadvertent exposure of sensitive data
: This pertains to the inadvertent storage or disclosure of confidential information in a manner that permits access by other applications residing on the user's mobile device.

Laxation on authentication and authorization measures
: The implementation of inadequate authentication and authorization protocols poses vulnerabilities that can be exploited by malicious applications or malevolent users, potentially leading to unauthorized access.

Susceptible data encryption techniques
: The use of weak encryption algorithms, methods with known vulnerabilities, or those susceptible to facile decryption introduces security risks, compromising the confidentiality of data.

Unencrypted transmission of sensitive data
: The transmission of sensitive data without encryption over the internet represents a significant peril, as it exposes the information to interception by unauthorized parties.

OWASP publishes a Mobile Top 10 list, now in its third iteration, which includes security vulnerabilities and provides best practices to help remediate them. Organizations building mobile applications should minimally refer to the list to build appropriate defenses that can handle static attacks based on source code and dynamic attacks that exploit application functionality.

The following are best practices for mobile application development.

Fundamentals

There are several checks an application developer can implement to detect any backdoors or extraneous functionality before releasing an application or publishing an update to it. These include:

- Having a trusted third party manually review the code
- Examining and documenting all API endpoints
- Revising information in logs
- Removing all test code from final releases
- Checking configuration settings to ensure no easy access to extra functionality is inadvertently granted

Supercharging Your CI/CD Pipeline with Security

Incorporating security testing tools into your CI/CD pipeline is essential for identifying and addressing security vulnerabilities early in the software development process. This will allow organizations to proactively identify and remediate security vulnerabilities and improve the overall security posture of their applications.

These are some of the recommended security scans (although the list is not exhaustive):

- Static application security testing (SAST)
- Software composition analysis (SCA)
- API security testing
- Secrets scanning
- Pen testing

Navigating Privacy Concerns in Mobile Application Development

Consider application compliance with regulations like the California Consumer Privacy Act (CCPA), GDPR, the Digital Operational Resilience Act (DORA), etc., to protect user privacy and avoid penalties. Key points include:

- Develop a data breach response plan for timely notifications to users and authorities.
- Evaluate third-party service privacy compliance.
- Secure data transfers with safeguards like standard contractual clauses (SCCs) or Privacy Shield.
- Enable user rights (access, rectify, delete).
- Use robust encryption for data protection.
- Implement secure authentication and authorization.
- Collect only the data necessary for the intended purpose and avoid collecting excessive data.

Mobile Application Security Using Containerization

Reet Kaur

The remote working culture has led to more and more people using their laptops, tablets, and smartphones for both work and personal use. Many companies have developed Bring Your Own Device (BYOD) policies to allow remote employees to use their personal devices to connect to corporate networks. While these provisions in policy surely enhance workforce mobility, it also brings several security and privacy challenges.

There is a direct risk of commingling personal and corporate data, including potential exposure to sensitive corporate risk, which may lead to unintentional data leaks or security breaches caused by interaction between personal and corporate applications containing sensitive, work-related information.

Containerization can help partition applications on mobile devices to separate corporate data and personal data, while preventing unauthorized access by other applications or malicious actors on mobile devices. This ensures better privacy, data protection, and control over sensitive information within secure containers on the same mobile device, helping to be compliant with regulations.

Separation between applications is done using virtualization or application sandboxing techniques, by isolating each application within its own container, creating a boundary that separates it from other applications and the underlying operating system. This isolation ensures that the application and its data are independent of other apps running on the device, making it possible to assess and manage your emails, documents, or business applications securely, without the risk of interference or exposure to personal apps or data.

Companies can enforce specific policies and restrictions on corporate applications. For example, they can control data sharing between applications,

restrict copying or sharing of data outside the container, or enforce stronger authentication measures like MFA for accessing corporate resources.

If a device is lost or stolen, or if an employee leaves the organization, containerization makes it easy to do selective data wiping/erasing of company data without affecting the personal data on the device.

The containerization of mobile applications is commonly used in enterprise mobility management (EMM) or mobile device management (MDM) solutions. These solutions provide centralized administration and control over corporate devices, including the ability to deploy and manage containerized applications. Application containerization can be configured using the mobile application management (MAM) functionality offered by MDM, through granular control over application policies, data separation, and application-specific security features. One important thing to note is that your organization should have a clear policy on deploying EMM/MDM/MAM to your employees' personal devices and give them the option to opt-out if they have any privacy concerns about putting work-related applications on these devices.

While containerization may satisfy the operational needs of security, compliance, and mobile device trust, it's important to also consider potential risks.

Containerization may not protect against all threats, as malicious applications outside the container can still pose risks. Advanced malware can still bypass or break out of containers, and insider threats can compromise the security of applications within the container. Compatibility issues, complexity, and a false sense of security are additional concerns.

To address these risks, a layered security approach must be adopted to complement containerization security. Performing comprehensive application vetting and risk assessments, network security, monitoring and threat detection, secure coding practices, user training, and other security measures. This is crucial for effective mobile application security in today's evolving threat landscape.

PART X

API Security

API Security: JWE Encryption for Native Data Protection

Andres Andreu

In the spirit of zero trust, implementing encryption and decryption capabilities into an API ecosystem should be considered standard practice. APIs are generally accessed remotely via a web protocol, such as HTTPS. Using any version of HTTPS, including TLS, is not enough for protection at depth. It is only effective for transport security or protecting the streams that carry your data. This technology does not protect the data itself.

A better protection is to use native data-level, or payload, protection via an orthogonal layer of encryption and decryption. This aims at protecting the data itself no matter what happens in transit, even a successful MITM attack. Moreover, it starts to push the API ecosystem toward a model where trust is based on the fact that cryptographic key exchanges have taken place between the two parties (e.g., zero trust) of a given request and response relationship (e.g., an API call gets made). Obviously, the cryptographic keys need to be protected on both sides of the relationship, but the use of this technology raises the bar such that communication is only accepted if specific cryptographic keys have been used on it. This mode of operation facilitates native data protection but also opens up the possibility for machine authentication via mutual TLS (mTLS), obviously using different sets of private/public key pairs (another example of zero trust).

Given the popularity of JSON Web Tokens (JWT) for both representational state transfer (REST) and GraphQL models, let's focus on natively protecting it. JWTs are Base64 URL-encoded objects that can optionally be signed via JSON Web Signatures (JWS). JWS does not provide confidentiality since anyone can decode and read a payload. Enter JSON Web Encryption (JWE), RFC 7516. JWE allows you to encrypt a JWT payload, making it so that only

the appropriate recipient with the appropriate cryptographic materials can decrypt and read the protected data.

With the use of JWE, the protected data gets serialized. Two types of serializations can be utilized when leveraging JWE for data protection: compact serialization and JSON serialization. Typical implementations of JWE utilize compact serialization. This data is structured as a Base64 encoded dot-separated string with four dots and five parts.

Part 1 carries a header. This header consists of metadata describing:

- How the plain-text payload was encrypted to create the ciphertext (i.e., which algorithm was used to protect both the content-encryption key [CEK] and the plaintext)
- The type of content that has been protected

This underlying data is not encrypted. The encoded header looks like the following:

```
JSU0EtT0FFUC0yNTYiLCJjdHkiOiJKV1QiLCJlbmMiOiJBMjU2R0NNIiwiaWF0IjoxNTg1N
zAyMjg1LCJ0eXAiOiJKV1QifQ
```

Decoding the header yields this:

```
{
    "alg":"RSA-OAEP-256",
    "cty":"JWT",
    "enc":"A256GCM",
    "iat":1585702285,
    "typ":"JWT"
}
```

Part 2 carries the CEK that was used to protect/encrypt the plain-text payload via the symmetric encryption process. The CEK is unique per token and generated for one-time use only. This key must never be reused.

Part 3 carries the initialization vector (IV) that gets used if the symmetric encryption algorithm in use requires it. This is a randomly generated element of data.

Part 4 carries the actual ciphertext or the protected data. This is the end result of the encryption algorithm using the CEK and the IV (if applicable) on the unprotected plain text.

Part 5 carries the authentication tag. This is a value that gets generated during the encryption process and can be used for validation during the decryption process.

A complete example of a JWE encrypted string that you would see as a protected payload in an API call is provided below, with each part noted with a numbered icon:

```
eyJhbGciOi❶JSU0EtT0FFUC0yNTYiLCJjdHkiOiJKV1QiLCJlbmMiOiJBMjU2R0NNIiwia
WF0IjoxNTg1NzAyMjg1LCJ0eXAiOiJKV1QifQ.❷HY8160chJ4VeWWQzPZzehFApkhUzhoL
Hg9pUvnnWRiW33bvsYUBNV3dKCLrV_KRosXnbnYgLgOp5nRV3Wj9GG2ZcEniJzRzZpDlgvN
0lmNwVRS1kiNw3s7-5eK0kmZz2hsVzGnvPS5Ng9-tzABVyerDi7USmp59lsfqtFCQHRk6ha
AW7PZ-YpynRWXl4mDseLKgwsGJBxiW-eymoIMJeOUENWbtiJXfEz5vVB1fhCPQo7INPiQTC
N9DPhzV2i4fES7cJZkDBO5SiSBZGgt_xTaIej0ZRdRkZjkqYUJtCJvCEeEGISSo9aM7UBEq
3wk_WIAGKU3xCuashZOhHQrw7gL9AeY4AXZQzph4Ny134fdZakJjWwJmyZD3EkfzVlM0_lW
jGUsje1POkZKoyej2OMF3bGK2_OPaMySZrvwYMRdEtZiQPGHFRJS42EYctcImuLHXt9KGa4
wBtsur5V-_ioaePaSAKv3es6xQTmdYRMDCVPZjckD7Gsy2ZQHAI9epN8zQttlhh2JATuLe1
o2FxtIc5xF_hYQ9ujxRQ0g-bHjeUwCN-huPuimazWZ5LycKYrd-rNqcoDDz8SzGv2no8KoO
UUIH7b7V4NjvMQ1oUM5Hahfi0Btuqg372s1qkUHRTGNuJRcmflxmBwRGRx1ds48kCbeTGjn
4YCh9FjmogSH4.❸BPVBQEv7z5hAhHf6.❹LV9A0irLOenlA_bSYWr2x5Ro8O7SS-lyyZqm
mmRomoLSWziBloWWYV8gGPGIFcqpiQJjf1evnfD41ZKkM1VQMwQTsLEJDU91ljkrfnuH504
arHQIKcgYrwp1bfEM-TJYdsSHmd5YdLNwkmczsEu8ZJ3DsLKxPGUzLcucxMVMDBtr5wF_5F
dAk0yEyAIP8mEnVa2XecReC-FJer0X-B8IPq9Hv3Zf7cmFBNqrc74vDu_0oC876HEnhLb7-
e1C6d4tRitWu8fq.❺cE1dqElIb8SEejRcbll9Xw
```

APIs Are Windows to the Soul

Brook S.E. Schoenfield

Too many times to count, developers, founders, and creators misunderstand the importance of their APIs in application security. "It's just our API." An API is the attacker's gateway to all information processing and storage behind or downstream from the API. If available through the internet, each API routable address will be found, and then the API probed for weaknesses, whether targeted or not. This has been the sad and unfortunate reality on the internet for several decades: all routable addresses receive at least automated attacker attention.

Risks

What can an attacker do with a poorly defended—or worse, undefended—unmonitored API?

Each data mechanism has its own set of issues. Any protocol based on XML will be subject to XML External Entity (XXE) attacks, which can expose information and downstream processing and allow an attacker to redirect processing to their URL. GraphQL might also allow XXE, but GraphQL introspection reveals data, program logic, and data organization (schema). Any API that fails to rigorously disallow unintended data may unwittingly pass attacks to whatever services lie behind the API. Authentication and authorization may not work as intended or include vulnerabilities.

These are just a few examples of the sorts of issues regularly found in APIs.

Orphaned APIs are routinely left receiving and passing data, but no longer monitored and no longer tested. Old APIs are an attacker's playground.

In threat modeling, we call the exposed, reachable attacker contact point to our network or interfaces as an *attack surface,* exactly what APIs always present. Defenses like firewalls can only do so much; these won't usually understand your data forms and expectations, or any proprietary message

formats and encapsulations. The API thus becomes your internal logic's "firewall." Otherwise, the attacker will use your API as their vector of attack and exploitation.

Defenses

Some absolutely required defenses include:

- Firewalls, network restrictions, and interprocess authentication
- Data validation against the schema, expected type, and expected range and size

Access Management

It's critically important that we restrict communications to only those required to function. For public APIs that must endure internet exposure, the best we may be able to do is authentication before allowing communication. Most public APIs today require the user to apply for a credential. The most common credential is an *API key*—that is, a nonpredictable value that becomes the API user's credential. The dangers inherent in API keys are beyond the scope of this essay, but API keys are rarely sufficient, since attackers may apply for keys—or even steal them.

However, if the use of the API can be restricted, it should be done by all available means, including:

- Limiting to only the networks that must have access
- Providing user and interprocess authentication
- Authorizing only required actions
- Encrypting tunnels for point-to-point use cases

Input Validation

The utmost necessity to validate and sanitize API inputs remains—unfortunately—not well understood. Because attackers frequently obtain API keys, it must be assumed that authenticated and authorized users include some adversaries. These attackers will then "fuzz" the API to identify any vulnerabilities or weaknesses by trying every known exploit and experimenting with various combinations of unexpected input.

Hence, one of the most important protections that every API (and most especially, publicly available APIs) must implement are input validations. Limit every input solely to:

- Allowed size or length of input data
- Expected data types
- Proper formats and form of data (schema)
- Expected ranges

Let nothing else pass on to downstream processing so that any exploitable conditions behind the API cannot be reached. Log every abnormal input. These might be the result of programming errors by users or one's own functionality. But these abnormalities are one's first warning of an attack. Bear in mind that for message-encapsulated exploits, the API must function as one's proprietary "firewall," since no firewall will have the details of how your API and its messages work.

It's been said that a person's eyes are the window to the soul. In the same way, each API provides a "window to the soul of your applications." Don't give your soul to an attacker "devil."

API Security: The Bedrock of Modern Applications

Charan Akiri

The evolution of software from monolithic architectures to microservices has elevated the importance of APIs. APIs facilitate interactivity among software components, systems, and third-party services, and they have become the gateways to a plethora of services and data. Thus, APIs have become prime targets for cyber adversaries. Breached APIs can result in unauthorized data access, system infiltrations, and even complete system takeovers. Their often subtle presence can lead to oversights during security assessments. These oversights can create many potential vulnerabilities. In this world progressively defined by integration and connectivity, ensuring the security of APIs is paramount.

API breaches can wreak havoc, revealing confidential data and core functionalities of an organization's software infrastructure. Some significant breaches within the past five years include:

Facebook (2019)
: A third-party Facebook app's API was exploited, compromising over 530 million users' data.

LinkedIn (2021)
: An open API led to a data breach affecting nearly 700 million users.

T-Mobile (2022)
: A breach through one of T-Mobile's APIs exposed 37 million customers' personal details.

OWASP underscores API security's significance by enumerating potential API risks in its renowned OWASP Top 10 API list, including Broken Object Level Authorization, Broken Authentication, and Excessive Data Exposure.

Here are the top 10 key principles of API security that must be used by developers as a definitive guide for safeguarding APIs in an increasingly interconnected digital landscape:

1. Authentication and authorization. Knowing who is accessing your API and what they're allowed to do is foundational. Implement robust authentication mechanisms to ensure only authorized entities access your API. Test authentication and authorization rigorously, verifying that each identity gets appropriate access rights.
2. Zero trust approach. Zero trust is a security policy centered on the principle that companies should not trust anyone by default. Zero-trust ideology should be applied to even authorized API endpoints, authenticated clients, and unauthenticated and unauthorized entities.
3. Rate limiting. This isn't just for ensuring your API can handle traffic—it's a security measure. By limiting request frequency, you can mitigate denial-of-service (DoS) attacks and slow malicious actors.
4. Input validation. Treat API input as potentially malicious. Data should be strictly validated and sanitized, especially for dynamic APIs like GraphQL, where queries can be complex and multifaceted.
5. API discovery. Maintain your API inventory and regularly update it using automated tools, ensuring you're aware of all active endpoints.
6. Error handling. Avoid exposing stack traces or detailed error messages. Ensure your API returns generic error messages and logs details securely.
7. Expose only limited data. Ensure APIs only expose as much data as needed to fulfill their operation. Enforce data access controls and the principle of least privilege at the API level.
8. Monitoring and logging. Establish a "normal" operational baseline through continuous monitoring, facilitating the identification of anomalies.
9. Security audits and pen testing. Conduct reviews and penetration tests regularly to identify vulnerabilities, especially in authentication and authorization mechanisms.
10. WAF integration. Embrace the WAF solution which has distributed denial-of-service (DDoS) protection, bot management, and direct API protection.

GraphQL is a query language developed by Facebook and released as an open source project in 2015. It is used for developing APIs and a runtime for executing those queries with existing data. While it offers flexible querying, it can also inadvertently introduce vulnerabilities. Organizations must ensure rigorous type checking for GraphQL endpoints and employ query complexity analysis to avert potential attacks.

Innovations like quantum computing will necessitate evolving security paradigms. The proliferation of internet-connected devices will lead to an exponential rise in APIs, broadening the potential threat landscape. Automation, coupled with AI and ML, will be instrumental in safeguarding this expansive digital terrain.

In conclusion, API security is a continuous endeavor. Given APIs' pivotal role in today's software ecosystem, their fortification will consistently remain at the forefront for enterprises and developers.

API Security Primer: Visibility

Chenxi Wang

APIs are a crucial component of modern application architecture. APIs play a central role in enabling interoperability, communication, and data exchange between diverse systems and applications.

Because of the pervasive nature of APIs, ensuring secure operations via APIs is critically important for safeguarding data, maintaining the integrity of systems, and protecting the proper functioning of applications in an interconnected digital ecosystem.

In practice, security professionals forming an API security strategy should consider these major aspects. I'll deep dive into each aspect in three essays:

Visibility and inventory
: Knowing which APIs you have, what they are, and how long they have been around, and then creating an inventory of APIs with context metadata are some of the first steps of getting your hands around API security.

Chapter 83, "API Security Primer: Risk Assessment, Monitoring, and Detection"
: After inventory, you need to assess whether the API endpoints conform to security architecture/design requirements and policies. Further, you need to monitor and detect any policy violations that may arise.

Chapter 84, "API Security Primer: Control and Management"
: Once you know what APIs you have and what risks they carry, you need to exercise control over the design, deployment, and management of APIs to mitigate risks and secure your applications.

Visibility and Inventory

Many organizations do not have an up-to-date and accurate inventory of all of their APIs. As such, the first task of an API security initiative is gaining

that visibility—what APIs you have, how many you have, whether they are internal or external facing, and how long have they been around, etc. The discovery and inventory aspects of this process is a must-have for any API security strategy.

More specifically, the visibility tasks include:

Discovery
: Scan and report which applications use APIs. The factors of consideration here include whether or not you need real-time and continuous discovery. If not, how often should you scan and update the inventory?

Classification
: Categorize the list of APIs. Are they internal or external facing? Do they process PII? Do they process your customer data? This classification step provides a structured look into the list of APIs and serves as the foundation for risk exposure assessment and API management.

Metadata enrichment
: This step provides further contextual data, such as the date the API went live, team ownership, relevant usage metrics, date of the last update, pointer to the repo, etc. It is always useful to know where an API endpoint is. In the event of a breach or investigation, it is often the deeper contextual information that provides the critical clues.

The requirements for visibility may vary from organization to organization. For example, you may or may not need real-time discovery of APIs. Similarly, dynamic updates of inventory versus batch updates is another decision that may or may not be critical for your use cases.

What is clear, though, is that everyone would need automation-based discovery, classification, and context enrichment. Manual discovery or management of an API inventory is a nonstarter.

Architecture consideration for API discovery is also relevant here—you may care how intrusive the discovery function is. Some tools require deployments deep into your network to process traffic in order to discover API usage. In environments where span ports are readily available and siphoning network traffic is generally acceptable, it is not a big deal. In other environments, however, it can create conflict between security and the operations team.

DNS monitoring and firewall plug-ins are two alternative ways of discovering API endpoints. If a new DNS record is published, it could be that a new API is created. Similarly, if a new firewall rule is created, this could mean the network is opening up access to a new API.

API Security Primer: Risk Assessment, Monitoring, and Detection

Chenxi Wang

The second aspect of forming an API security strategy[1]—after the discovery of APIs in your environment—involves assessing whether the API endpoints conform to your security architecture/design requirements and policies.

Of course, this assumes that you have an API secure design policy. Such policies should cover these considerations:

Authentication
: Does the API access have authentication? What kind of authentication?

API secrets management
: Are the API tokens/keys managed? Do they have an expiration date? Should secrets be rotated and refreshed periodically?

Decommissioning of APIs
: Should obsolete APIs be decommissioned? What is the decommissioning process?

Vulnerability management policy
: What happens when a critical vulnerability is found associated with an API? Do we report? Do we block its use? Do we mitigate threats by enacting an API filter? When do you have to remediate this vulnerability to be compliant with policies and any relevant SLAs?

Once you have a policy, you can detect violations such as unauthenticated API endpoints, obsolete APIs, or those with high or critical vulnerabilities.

1 The other two aspects are discussed in "API Security Primer: Visibility" and "API Security Primer: Control and Management."

This step also allows you to understand your risk exposure, which is critical to being able to manage or mitigate API-related risks.

To assess risk and detect errors and violations, *continuous monitoring* of the API runtime environment is necessary. For instance, API usage and associated metrics such as response times, throughput, and volume provide insight into the normal functioning of APIs. On the other hand, tracking error rates and the distribution of error codes aid in identifying potential vulnerabilities or issues that may compromise system integrity.

Monitoring is also critical for defenses against malicious activities as well as the prevention of data leaks. For example, monitoring data movements and detecting anomalous patterns can help identify unauthorized data transfers, ensuring compliance with data protection standards. By continuously monitoring and assessing changes in API usage and system interactions, anomalies can be flagged and investigated promptly.

In addition, the monitoring system must generate and retain logs for a specific retention period as per policy, compatible with major SIEM platforms. This ensures that your security team can effectively gain insights into the security posture of the system, launch investigations, and respond to incidents promptly.

Once your monitoring infrastructure is set up, you need to *establish a detection logic and engine*. Detection can be done via rules or anomaly identification. Leveraging statistical analysis, ML, or other anomaly detection techniques, your system can automatically detect deviations from normal behavior. You may also create custom rules and alerts within your SIEM system to identify specific events or patterns indicative of malicious activity.

The final step in the analysis and detection process is to *implement automation* to streamline the detection and response process. Automating routine tasks and responses to known threats can expedite the remediation and response process. You may also want to deploy an orchestration system to ensure a coordinated and efficient response in the event of a security incident.

API Security Primer: Control and Management

Chenxi Wang

The third aspect of effective API security, besides Chapter 82, "API Security Primer: Visibility" and Chapter 83, "API Security Primer: Risk Assessment, Monitoring, and Detection", is implementing robust security controls to protect APIs against potential risks and threats. Here are some key considerations for API security controls:

Manage the API life cycle
: Properly managing the life cycle of APIs is an effective way of ensuring your security posture. For example, timely decommissioning of unused APIs is crucial to mitigate potential risks introduced by vulnerable and obsolete APIs. Similarly, regularly updating and patching APIs is a must-have in your API management strategy.

Enforce Data Encryption
: Enforcing data encryption is crucial to protect data in transit. This includes forcing the use of Secure Sockets Layer (SSL) to encrypt communication between clients and the API server. Additionally, you need to ensure that sensitive data fields are never transmitted in clear text, especially in API requests where parameters may be visible.

Implement authentication and access control
: Authentication and access control are essential components of API security. Authentication ensures that the entity interacting with an API is who it claims to be. By verifying the identity of users, systems can prevent unauthorized access and protect sensitive information. Access control ensures that sensitive data is only accessible to authorized entities, thereby preventing data exposure.

Adhere to deployment standards
: Establishing strict standards for API deployment is essential. APIs that do not meet defined standards or requirements should not be deployed.

This helps maintain consistency, reduces the risk of vulnerabilities, and ensures that all APIs within the ecosystem adhere to a common security posture.

Properly handle secrets

Secrets such as API keys or authentication tokens need to be properly managed and secured. For instance, using a vault system to store and rotate secrets can provide much-needed protection. Similarly, using dynamic tokens instead of static ones can better prevent unauthorized access to APIs and the applications behind them.

Validate API input

Input validation is a fundamental security measure to prevent API injection attack, where malicious entities attempt to inject code into API requests. Input validation includes defining and enforcing API schemas that specify allowed methods, expected JSON/XML syntax, and individual parameters. By validating inputs at the entry point, we can prevent a range of common security vulnerabilities, such as injection attacks.

Use rate limiting

Rate limiting is often utilized to defend against DDoS, volumetric attacks, content scraping, and credential stuffing. Rate limiting API calls is a table-stake control today to prevent abuse and ensure fair usage.

Include hardening

Hardening the various components in an API infrastructure allows a more robust security posture. Examples include ensuring the use of secure TLS, implementing robust ciphers, and stripping unnecessary headers to help reduce the attack surface. Careful management of error responses is another hardening measure—the API should be configured to hide detailed error messages that might reveal sensitive information. Data masking could be applied to ensure that error messages do not disclose sensitive or critical information, thereby minimizing the risk of potential exploitation.

Protect existing APIs

Existing APIs that may not have been developed with security in mind represent distinct risks to the organization. One approach to mitigate such risks is to place a protective layer in front of the APIs. Often referred to as an API proxy, this layer enforces security-specific controls, such as authentication, authorization, and input filtering. For example, ensuring that incoming requests adhere to predefined standards and

requirements before reaching the actual API is critical to prevent potential exploits.

API is the glue for today's digital economy. Proactively addressing security considerations at every stage of the API life cycle helps to safeguard data, prevent unauthorized access, and ensure regulatory compliance. At the same time, it also helps to foster trust among users and partners who rely on API-driven applications in today's interconnected digital landscape.

PART XI

AI Security & Automation

LLMs Revolutionizing Application Security: Unleashing the Power of AI

Alexander James Wold

The realm of application security has witnessed a pivotal advancement with the emergence of large language model (LLM) security solutions driven by AI. In this essay, we explore the transformative impact of LLMs on SAST and threat hunting, showcasing their potential to revolutionize cybersecurity practices.

LLMs and Static Application Security Testing

LLMs bring a paradigm shift to SAST methodologies by employing AI-powered static code analysis. This enables a comprehensive scrutiny of source code across diverse programming languages and frameworks, granting a profound understanding of application structure and logic.

Unlike conventional SAST tools with rigid rulesets, LLMs leverage advanced natural language processing capabilities to discern contextual semantics and programming idioms. Consequently, they excel in identifying intricate security vulnerabilities that often evade rule-based systems. Furthermore, LLMs continuously improve through learning from extensive data sets and the collective expertise of security professionals, enhancing their accuracy and effectiveness over time.

LLMs and Predictive Threat Hunting

The predictive abilities of LLMs are a game-changer in threat hunting. Their deep learning algorithms analyze historical threat data, patterns, and indicators of compromise (IoCs) to forecast potential zero-day vulnerabilities. This

proactive approach equips security teams to stay ahead of evolving attack vectors.

By integrating LLMs into the threat-hunting process, organizations gain the ability to detect and analyze sophisticated attack techniques. LLMs assimilate knowledge from various sources, such as cyber threat intelligence feeds and industry reports, refining their threat detection mechanisms to become valuable allies in combating the ever-changing threat landscape.

Unique Advancement: LLMs and Intelligent Security Patching

LLMs offer a unique feature—intelligent security patching. Leveraging their in-depth understanding of code structure and vulnerabilities, LLMs provide specific patch recommendations and code modifications to proactively address security weaknesses. This goes beyond mere identification, enabling LLMs to actively contribute to the remediation process.

In evaluating suggested patches, LLMs consider contextual factors, such as application functionality and dependencies, to gauge potential impacts. Automating patch management through LLMs streamlines the remediation process, reducing the burden on development teams and minimizing the exposure window for vulnerabilities.

Challenges and Considerations

While LLMs present numerous advantages, they also pose challenges that warrant careful consideration. Ethical concerns arise from AI-based decision making, demanding efforts to ensure fairness and impartiality in security assessments. Bias in training data can propagate into LLMs' recommendations, leading to skewed results.

Moreover, LLMs' predictive capabilities are not infallible. False positives may occur and true positives may get missed in threat-hunting scenarios. Calibration and validation with real-world testing are necessary to avoid overreliance on LLMs and potential misinterpretation of results.

Conclusion

In conclusion, large language model security solutions empowered by AI have ushered in a new era of application security. Their proficiency in SAST, predictive threat hunting, and intelligent security patching highlight their unparalleled value in safeguarding applications against evolving cyber threats.

As AI technologies advance and human expertise complements LLMs, we stand on the cusp of a safer digital future. However, vigilance is essential in monitoring, addressing ethical considerations, and employing LLMs judiciously to unlock their full potential in application security. With this approach, we pave the way for a more secure digital landscape, where applications are fortified against even the most sophisticated attacks.

LLMs exemplify the symbiosis between AI and cybersecurity, empowering organizations to build robust defense mechanisms. As AI continues to evolve, LLMs are poised to become even more adept at identifying emerging threats, enabling proactive security measures. However, to fully realize this potential, continuous research and development are necessary to address LLMs' limitations, refine their accuracy, and tackle ethical implications. By fostering interdisciplinary collaborations, the cybersecurity community can unleash the full potential of LLMs, driving us toward a future where applications and digital assets remain shielded from the ever-growing threat landscape.

Mitigating Bias and Unfairness in AI-Based Applications

Angelica Lo Duca

AI continues to revolutionize the world, and there is a growing concern for bias and unfairness in AI-based applications. In practice, bias in AI refers to the presence of systemic and unjustified preferences or prejudices in AI systems. Bias can be introduced to data sets in the traditional way, by being present in the raw data (because of environmental circumstances, accidental omission/addition, etc.). But it can also be introduced by malicious actors who want to skew the results of an AI-based app. In contrast, unfairness refers to the actual outcomes resulting from such biases, leading to unequal and discriminatory treatment of individuals or groups. Bias is the underlying issue, while unfairness is the consequence of that bias. As such, it's important to protect your AI-based apps against that added bias and unfairness in AI training models.

Among the most popular threats in this field, there are poisoning attacks, which are a significant concern for AppSec because they could compromise the AI model. If an adversary understands the weakness of the biased model, they can produce carefully crafted inputs to cause the model to make incorrect decisions. You can also have malicious actors who inject unfair inputs into training data to manipulate AI models to produce targeted responses, such as manipulated text sentiment.

To mitigate bias and unfairness in AI development, the following can be a good start, even though the industry is still learning.

Collaborate with Domain Experts

One possible solution to mitigate bias and unfairness in AI development is collaborating with domain experts. Domain experts are individuals who deeply understand the industry or field targeted by the AI application. Their

knowledge and expertise can help identify potential biases and discrimination before they occur.

Collaborating with domain experts at every stage of the development process ensures that all perspectives are considered, including those from underrepresented groups. This collaboration also enables developers to understand better how their technology will impact different communities.

Improve Data Quality

Improving data quality and understanding the data used is another possible solution to avoid bias and guarantee fairness in AI-based applications. The performance of an AI model heavily relies on the quality of the data it is trained on. Therefore, developers must ensure their data sets are diverse, inclusive, and balanced. This means collecting data from different sources while considering all demographic groups involved.

Monitoring and updating the data set is also crucial to ensure that the AI model remains fair and unbiased as societal dynamics evolve. This iterative process enables developers to adapt their algorithms and address potential biases promptly, promoting an environment of continuous improvement and striving for greater fairness in AI-based applications.

Perform User Testing

Performing user testing is another approach to reducing bias and unfairness in AI-based applications. User testing enables developers to get direct feedback from diverse users, helping to identify any potential biases or discriminatory outcomes that may have been overlooked during development. In addition, user testing provides an opportunity to assess the application's effectiveness across different demographic groups, ensuring fair treatment for all users.

As AI-based applications become more prevalent in our lives, it is crucial that we address the potential issues of bias and unfairness before they become more widespread in application development.

Secure Development with Generative AI

Heather Hinton

With all of the attention given to generative AI (GenAI) and its potential use within development, we cannot neglect the practice of secure development in GenAI-assisted development environments. This essay proposes considerations for the adaptation of secure GenAI-assisted development and testing practices, starting right at the initial "light bulb" moment and throughout the development life cycle:

User stories
: User stories are a key input to the design process, used to help define a product or feature and how it is expected to be used. If using GenAI solutions to help identify user stories, these stories must be independently reviewed by a human and modified and enhanced if necessary. Do not assume that all required stories will be created by the AI or that these stories will be accurate, complete, or correct.

Specifications
: User stories (should) provide input to technical specifications. The specifications must be clear, accurate, and comprehensive. This doesn't change with the use of GenAI-created user stories. If using GenAI to create specifications from user stories, the specifications must be reviewed: no one (and nothing) should write code if you do not have a clear definition of what is expected of that code.

Development considerations
: There is much evidence that (common) code can be "developed" much faster with the assistance from GenAI tools. This savings in time should be put to good use, enhancing the too-often-neglected testing phase. In particular, focus on a human-in-the-loop explicit code review of any code (regardless of authorship) to make sure that it can be supported by a human and it meets the logical functionality expected by the design

specifications. This is likely easier and less time-consuming for common functions but may require additional time for new or complex functions.

Testing considerations

Now is the time to reinvigorate the testing disciplines, including testing for fail-safe, fail-secure, and (a new one) fail-private (no unintended exposure of personal, private, or sensitive information). While test cases may be created through GenAI, a human in the loop to lead and guide the destructive and out-of-the-box testing phases is part of an overall defense-in-depth strategy.

Creating data sets for testing

One area where many teams struggle is in the desire to use live data, including customer data, for testing. Suppose a GenAI tool is going to help redact personal or sensitive data from a live data set to create an artificial data set for testing purposes. In that case, you may end up breaching confidentiality requirements, doing exactly what you were trying to avoid by creating the artificial data set in the first place. Instead, consider using a limited, human-redacted set of data as a starter set, to allow an internally hosted- and managed GenAI to build an enhanced artificial data set.

Training considerations

When building a solution that embeds GenAI and requires an initial round of training of an underlying large language model (LLM), an artificial data set must be used instead of live customer data. Given that training is an ongoing activity (as opposed to prerelease testing), multitenant products have additional considerations, as they may need to enforce limited-tenancy, limiting or excluding individual customer data sets, while allowing for multitenant training for those customers who opt in to broader training. This will, of course, add its own complications to the prerelease testing process (justifying our recommendations that the time saved in development be put to good use in effective testing).

At the end of the day, whether or how much of a product is generated by and with GenAI, reused from another internal project, or written by an employee, it must not skip the overall secure development disciplines ensuring security oversight at all stages including design, coding, and testing. Perhaps the biggest opportunity we see with the adoption of GenAI-assisted development and testing is the ability to use the shortened development

times to focus on robust testing, including testing for fail-safe, fail-secure, and fail-private.

Managing the Risks of ChatGPT Integration

Josh Brown

The integration of AI type systems and applications is bringing significant changes. AI, particularly in the form of ChatGPT, offers unparalleled capabilities in natural language processing and reasoning skills. While these advancements hold great promise for streamlining development, they also introduce novel security risks. Let's explore potential integration risks and offer insights into managing these challenges.

ChatGPT represents a new frontier in human-computer interaction. Its ability to understand and generate humanlike text, makes it a valuable tool for developers and organizations looking to automate development efforts and support business workflows. As ChatGPT finds its place in applications, it introduces both opportunities and risks for AppSec.

The integration of ChatGPT into applications may expose them to several security risks:

- To be useful, ChatGPT requires access to data, potentially including sensitive user information. Mishandling this data can lead to privacy breaches and regulatory compliance issues.
- Just like any other software, ChatGPT models can have vulnerabilities that malicious actors may exploit. There have been tests to produce bias and to force hallucinations.
- Adversaries can misuse ChatGPT to generate attacks such as convincing phishing content, automated spam, or even manipulate it for malicious purposes.
- ChatGPT may inadvertently generate biased or offensive content if not properly trained and monitored.

- Organizations using ChatGPT must navigate the legal landscape, considering aspects like content generation rights, input training rights, and intellectual property.

To mitigate these risks, developers and security professionals should consider the following strategies:

- Build your own proxy layer to detect communication that you deem not safe or sensitive.
- Make sure you sanitize both input and output with this proxy layer; all communication to and from the GPT system must be evaluated.
- Set up continuous monitoring for unusual chat behavior, enabling early detection of security issues.

The rapid pace of development presents a significant challenge for AppSec professionals. As development teams adopt GPT workflows, the velocity of code deployment will accelerate. This acceleration will likely outpace security measures, leaving systems vulnerable to emerging threats.

To address the disconnect between development speed and security, organizations should consider a new set of operations and capabilities to run alongside the normal core set of CI/CD, secure code training, etc.:

- Implement automated security testing workflows that communicate with a GPT system to build security capabilities such as detections, new code review models, and fuzzing.
- Implement auto GPT clients to build more intelligence into the conversation. You need to pull the human out of the loop. You must automate all the steps of a process and write quality checks.
- Build memory into your conversations. At the time of this writing, the conversations have limited memory, forgetting early parts of the conversation. This is a major limitation of the service. Redundant prompting can help; another possibility is to build a memory source for your conversation client side to allow for better retention.
- Make sure to measure the responses against metrics. When the metric shows ineffective, make sure to send the prompt back to the GPT system several more times. This is proving to be a highly effective process to get better answers when the response misses on the first try.

To stay ahead in this evolving landscape, organizations should:

- Invest in AI-specific security training.
- Stay informed about emerging AI-related security threats and best practices.
- Press hard for good security hygiene. The GPT system could expose mistakes without your knowledge. There's research out there showing forced hallucinations leaking another entity's data.
- Accept that you're not going to get quality tools to help you for a while.

By understanding the risks associated with external GPT systems and adopting synchronized development and security practices, organizations can harness the benefits of AI-powered technologies while safeguarding their applications and user data. In conclusion, the future of AppSec with AI and ChatGPT integration is bright, provided organizations proactively address the associated security challenges and adapt their practices to the evolving development landscape.

Automation, Automation, and Automation for AppSec

Michael Xin

In the technology world, automation is the key to improving speed and efficiency, and the same is true in the security world. Automation in AppSec refers to the use of special tools and techniques that can automatically perform important security tasks for applications. These tasks include, but are not limited to, scanning for vulnerabilities, finding weak points in security, and testing how well an application can withstand potential attacks. Instead of relying on manual and time-consuming processes, automation allows developers and security professionals to quickly and efficiently identify and fix security issues. Automating these processes saves time, improves accuracy, and helps organizations proactively address security concerns in their applications.

Automation is significant in application security for several crucial reasons. First, the widespread adoption of CI/CD practices has revolutionized the software development process, emphasizing the need for automation. With CI/CD, organizations aim to release new features and updates quickly and frequently to stay competitive. For instance, industry giants such as Amazon and Airbnb deploy code changes over 125,000 times daily. It is nearly unimaginable to manually ensure the security of each deployment in such a high-volume environment. Automation allows you to perform efficient and reliable security checks, scanning, and analysis of code, thereby mitigating the risk of introducing vulnerabilities into production systems. Moreover, the growth of cloud technology has further amplified the necessity for automation in AppSec. With cloud-based infrastructures, applications are highly dynamic, requiring rapid and seamless security updates. Automation enables continuous monitoring, vulnerability assessment, and threat detection in the cloud, ensuring the security of applications despite their constantly changing nature. Additionally, the introduction of DevSecOps promotes better

collaboration between engineering and security teams. Automation facilitates improved communication and coordination, enabling security measures to be integrated early in the development process, ultimately enhancing the overall security posture of applications. In summary, automation plays a pivotal role in AppSec, enabling organizations to effectively protect their applications while keeping pace with the demands of CI/CD, cloud technology, and collaborative DevOps practices.

Automating things in AppSec involves using technology and tools to streamline and improve security processes. Here are some steps to get started:

Identify repetitive tasks
: Begin by identifying security tasks that are time-consuming or require manual effort. This could include tasks like vulnerability scanning, code analysis, or access control management.

Research automation tools
: Look for tools specifically designed for automating application security tasks. For example, tools like OWASP ZAP, Arachni, or Brakeman can help automate vulnerability scanning and pen testing.

Plan and design
: Determine how you want to automate security tasks. This may involve developing scripts or configuring tools to perform specific actions automatically.

Implement automation
: Once you have a plan, start implementing the automation. This could involve writing scripts in languages such as Python or using specific features of security tools to automate tasks.

Test and monitor
: Validate the automated processes to ensure they are functioning correctly. Regularly monitor the automation to check for any errors or issues.

Iterate and improve
: Continuously refine and improve your automation processes. Address any issues that arise and seek feedback from security professionals or colleagues to make improvements.

While automated application security offers many benefits, it also comes with challenges and pitfalls. Initial setup and configuration of automation tools require time and effort to integrate into existing processes.

Maintenance and updates are necessary as security threats constantly evolve. Overreliance on automation can create a false sense of security, as it cannot catch all vulnerabilities. Human intervention and critical thinking are still crucial. Poorly implemented automation may result in false positives/negatives, wasting resources. To overcome these challenges, careful planning, regular testing, and continuous improvement are vital for effective and reliable automation in AppSec.

Remember that automation is an ongoing process that requires regular maintenance and updates as new security threats emerge. With time and practice, you will become more proficient in automating AppSec tasks, optimizing your workflow, and enhancing the overall security of your applications.

Embrace the power of automation and take control of your application security journey, starting today!

Will Generative and LLM Solve a 20-Year-Old Problem in Application Security?

Neatsun Ziv

In application security, we find ourselves at a crossroads where the integration of GenAI and LLMs is redefining traditional approaches, offering solutions to the issues of fragmented workflows and excessive tool clutter.

Traditional AppSec models were great at classifying or clustering data based on trained learning of synthetic samples. However, they struggle to keep pace with the hyperactive landscape of techniques, tactics, and procedures used by attackers to exploit any vulnerabilities. Let's explore the significant impact that GenAI and LLMs can have in transforming the field of application security.

GenAI, with its advanced algorithms and ML capabilities, is proving to be a powerful ally in the battle against vulnerabilities. By analyzing vast volumes of data, including security reports and code samples, GenAI can detect suspicious activities, identify potential malware, and even generate automated fix recommendations with the precision of a seasoned security professional. It is like having an assistant who tirelessly scans and safeguards your applications —and never takes a vacation.

Today's LLMs are a huge advancement over older models used in ML algorithms that were great at classifying or clustering data based on trained learning of synthetic samples. Modern, sophisticated models, like GPT-3 and GPT-4, can analyze code snippets, comprehend complex programming languages, provide developers with guidance, and even suggest fix snippets, all rooted in contextual awareness. Equipped with the insights offered by modern LLM, developers can confidently navigate the intricate terrain of secure coding like seasoned experts in the field without years of training.

Here are just a few ways that GenAI may revolutionize application security:

Automated vulnerability detection
: Traditional vulnerability scanning tools often rely on manual rule definition or limited pattern matching. GenAI can automate the process by learning from extensive code repositories and generating synthetic samples to identify vulnerabilities, eliminating this manual analysis that can often take weeks or even months.

Adversarial attack simulation
: GenAI can generate realistic attack scenarios, including sophisticated, multistep attacks, allowing organizations to strengthen their defenses against real-world threats. A great example is BurpGPT, a combination of GPT and Burp that helps detect dynamic security issues.

Intelligent patch generation
: GenAI can analyze existing codebases and generate patches that address specific vulnerabilities, saving time and minimizing human error in the patch development process.

Enhanced threat intelligence
: GenAI can analyze large volumes of security-related data, including vulnerability reports, attack patterns, and malware samples. It uses this data to improve threat intelligence capabilities by generating insights and identifying emerging trends from an initial indication to a real actionable playbook, enabling proactive defense strategies.

However, as we delve into the potential of GenAI and LLM models, we should proceed cautiously. LLM models, like any nascent technology, have areas in which they can further mature—particularly in aspects such as contextual understanding and real-time response. And while GenAI demonstrates remarkable proficiency, it is not a panacea for all application security woes. Moving forward, organizations will take a comprehensive approach to application security and combine GenAI and LLM models with dedicated security tools, external enrichment sources, and scanners to help bridge these gaps and ensure a robust security posture for their applications.

We are at the forefront of an exciting era where the synergy between GenAI and LLM models offers a path forward to conquer the persistent challenges in application security. These include trivial challenges, like running AppSec tools to very difficult challenges, such as protecting against new attack vectors. By cautiously embracing the transformative potential of GenAI and LLM models we can protect our digital landscape with confidence.

Understand the Risks of Using AI in Application Development

Yasir Ali

Generative AI in software development is going mainstream, evident by the 40K+ organizations using GitHub Copilot. These tools promise enhanced productivity by automatically generating code snippets based on given prompts but raises significant concerns about security vulnerabilities. A Stanford study highlighted that developers using codex tend to produce more insecure code, escalating risks in the SDLC.

Most industry studies show an average of 20–30 bugs per 1,000 lines of code written in a given project. Multiply this by the probable billions of lines of code that modern LLMs have been trained on and the odds of having real security issues inadvertently introduced becomes absolutely massive.

Traditionally, vulnerability concerns were centered around software components. But, with the rise in popularity of transfer learning, the reuse of pre-trained models, and the use of crowdsourced data, these issues have extended to AI systems. We are seeing a 49% increase in malware package creation quarter on quarter (QoQ) in 2023, which points to this problem becoming massive for 2024.

Main Risk Categories and Recent Incidents

Three main risk categories are identified with LLM use in software development:

- Security issues such as malicious code insertion and sensitive data leakage
- Legal liabilities, including concerns over code ownership and licensing

- Supply chain vulnerabilities extending to AI systems beyond traditional software components

Examples of malware insertions through LLMs have been documented, and upcoming tools such as Polymer's data loss prevention (DLP) with software supply chain scanners aim to mitigate these risks.

Major Threat Vectors from LLM

The OWASP Top 10 list for LLMs pinpoints significant threat vectors, including application vulnerabilities from LLM inputs, prompt injections, denial-of-service (DoS) attacks, and data leakage.

The supply chain for LLM-based methods can be susceptible to vulnerabilities. The vulnerabilities compromise the integrity of various components including training data, ML models, and deployment platforms. This can result in biased outcomes, security breaches, or even complete system failures.

Key Risks in the SDLC

The integration of LLMs into SDLC presents new security and legal challenges. Key issues include:

- Security problems such as "package hallucinations," where LLMs suggest nonexistent or malicious packages
- Propagation of unintentional software defects, magnified by the vast amount of code LLMs are trained on
- Legal complexities around AI-generated source code, especially regarding derivative outputs and licensing

Legal Concerns

There are emerging legal challenges regarding AI-generated code's ownership and its sourcing from potentially restricted open source packages. Samsung and others have blocked all AI-assisted code writing once due to inadvertent usage of noncommercial open source software packages.

LLM Concerns and Software Supply Chain Impact

While LLMs affect internal code security, their impact extends to third-party packages and libraries, often less rigorously vetted. The use of generative AI also enables more efficient attacks on software ecosystems.

Increased Supply Chain Risks

Here are several real-world examples of vulnerabilities and attacks:

- A breach in OpenAI was traced to a supply chain vulnerability in the Redis open source library, leading to unauthorized data access.
- Real-world supply chain attacks, akin to the SolarWinds breach, have escalated, targeting software developers and CI/CD infrastructures.
- A study showed over 100K public GitHub repositories exposed secrets, a concern amplified by the growing reliance on autogenerated codebases.

Remediative Controls

The SBOM is gaining traction with CISA and industry circles to gain some visibility on this risk. This is a welcome step, and although we feel that this is important, SBOMs are (at best) backward looking. If the application package has been shipped with vulnerability, then it's too late!

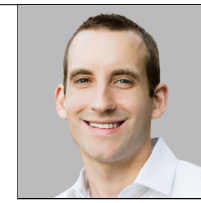
Inline supply chain controls in SDLC are the only method to secure your supply chain risk.

PART XII

IoT & Embedded System Security

Secure Code for Embedded Systems

Jason Sinchak



Unlike web, mobile, or local applications that are designed to run on a variety of platforms, applications for embedded systems are purpose-built for a particular system, including a custom platform (operating system) and associated hardware. The application becomes part of the embedded device, functioning as a key interface into the device and the orchestrator of many backend processes working in concert to provide a critical function. As a result of the marriage between an individual application and the entirety of the embedded system, the manner in which an application is developed and the associated cybersecurity concerns can have many downstream effects.

Coding

Speed and reliability are core tenets of an embedded system. These imperatives generally require coding applications in native unmanaged languages, such as C and C++ or interacting with associated libraries where this level of complexity is present. Speed requires close integration with hardware and the unescapable utilization of privileged operations. Cybersecurity considerations for coding on an embedded system focus on reducing vulnerabilities common to unmanaged languages and ensuring the application sustains a trust boundary between its primary interface with the outside world and the underlying platform.

Injection

The application serves as a key trust boundary and interface to the operating system, privileged functionality, or direct hardware. Protecting the application from becoming a vector through which a threat actor accesses these privileged functions is paramount.

Developers must ensure all untrusted data is validated and sanitized to prevent unintended system execution. Various injection-related attacks exist in

AppSec, but due to the prevalence of an embedded application interfacing with the platform, the most important category of vulnerability for an embedded system is operating system (OS) command injection. The worst scenario is when an application combines user input to a privileged call into a supporting application, operating system, or hardware. This situation typically arises when an application accepts untrusted/insecure input and passes it as the application name or arguments without validation or proper escaping.

Memory Corruption

Due to many embedded applications coded in C/C++, or any language that doesn't provide managed memory, developers should avoid the use of known dangerous functions and APIs. In the event a threat actor fuzzes the application inputs, it's possible for a memory corruption to occur, regardless of whether the application intended to or not; it would execute code directly on the operating system. Example functions to avoid include unsafe C functions such `strcat`, `strcpy`, `sprintf`, and `scanf`. Memory-corruption vulnerabilities, such as buffer overflows, can consist of overflowing the stack (stack overflow) or overflowing the heap (heap overflow). In the event a buffer overflow is possible, the application will likely crash, but a threat actor will write an exploit to take advantage of the situation. In the most classic example, a threat actor will overwrite the instruction pointer register via the overflow to execute the arbitrary malicious code on the underlying operating system. Additional considerations include knowing where your memory buffers are, when it is freed, data that could leak, initializing to known values, and where the buffer is stored (stack, static, or allocated structures).

Third-Party Code

Due to the nature of an embedded system, there are many moving parts and associated third-party components that ship with the system. When not properly evaluated, third-party components can put your own application and system at risk. It is important to ensure that the kernel, software packages, and third-party libraries are updated to protect against publicly known vulnerabilities. Built chains should be checked against vulnerability databases to determine if and when an update is needed. The last thing a developer or product owner wants is to ship an embedded device, which is hard or impossible to update in the field, with a known vulnerability. Embedded projects should maintain a bill of materials that includes third-party and open source software included in the entire application and firmware. The

bill of materials should be reviewed during development to confirm that none of the third-party software included has any unpatched vulnerabilities.

Platform Security for Embedded Systems

Jason Sinchak

Most embedded systems are devices that contain hardware from the era in which it was designed; regardless of how cutting edge it may be at design, it will eventually become legacy. Unlike applications designed to run on non-embedded systems, these applications cannot be moved; they live forever on this hardware. As a result, applications developed for embedded systems often have responsibility for the entire platform yet are constrained by the hardware. Therefore, the developers need to understand and apply other security controls to pair with AppSec practices.

Maintaining Data Security

As embedded systems can be physically accessed by a threat actor, such as through purchasing on the aftermarket, reverse engineering is often performed to gain access to the firmware. Due to the physical risks associated with storing data on embedded systems, applications must avoid critical mistakes such as hardcoding passwords, usernames, tokens, or keys in firmware. Globally default and hardcoded sensitive data will eventually be identified and published. Once published, that information will impact the entire fleet —all customers, or any device running the same firmware. Developers should avoid authenticating users or components through hardcoded information that is globally the same across products. System designers should enable the device owner to configure new sensitive material that is unique and chosen. For example, allow a user to change the default credentials or force a change of key material used to encrypt data at rest.

Secure Firmware Updates

The ability to perform updates of your own code and the underlying platform via a user application is critical. A firmware update procedure is a process for which a user or service technician can update the embedded system; this functionality is often built into the core user application. Application

developers should ensure robust update mechanisms by building an update procedure that requires and validates an update that has been cryptographically signed. Cryptographic signatures allow for verification that files have not been modified or otherwise tampered with since the developer created and signed them. In the event a firmware update process does not require authentication of the update, a threat actor can procure a firmware update, backdoor it with malicious binaries, and update the embedded system.

Attack Surface Reduction

Where possible, the platform itself and associated applications should limit components that are not necessary for operation. Removing insecure libraries and protocols minimizes the attack surface and provides a secure-by-design approach to building software by thwarting potential future security threats. For example, if you remove unnecessary functionality, when a zero-day exploit is released in the future, you may not need to issue a firmware update because your platform doesn't have the impacted function.

Secure Communications

Embedded devices typically must communicate the output of their function to components both internal to their own system and external. Defense in depth is achieved through securing internal communications and ensuring external communications employ mutual trust.

Embedded applications often communicate via a variety of potential application-level buses or API. In many cases, a publisher/subscriber method may be employed. While it is tempting and easier to create an information highway accessible to all internal applications on the "trusted" embedded device, developers should ensure that authorization and authentication exist on all messages passed internally.

At the same time, external communications can only be trusted when an embedded system's incoming and outgoing communications are of high integrity, authenticated, and authorized.

Application Identity for Embedded Systems

Jason Sinchak

When embedded system software is shipped, it is shipped in an out-of-the-box form that is the same for all users or customers. Identities used to authenticate to user interfaces or the platform are typically managed locally and uniquely for the system. As a result, the user or customer must be provided a mechanism for managing these identities through the main application interface or an external component.

As embedded systems can be physically accessed by a threat actor, such as through purchasing on the aftermarket, reverse engineering is often performed to gain access to the firmware. Once in the hands of a threat actor, it can discover critical mistakes such as hardcoded passwords, usernames, tokens, or keys in firmware. Globally default and hardcoded sensitive data will eventually be identified and published. Once published, that information will impact the entire fleet, all customers, or any device running the same firmware.

Embedded systems should avoid storing secrets in a hardcoded or globally default fashion. In addition, they should provide features that enable the separation of user accounts for user interface, internal management, console access, remote management, etc.

Where possible, service accounts should be avoided, and instead service-like features should be afforded as privileges for various users. Providing various levels of authorization and privilege at the identity level further mitigates attacks on known user accounts, such as automated attacks over the network.

While enabling identity management for customers on accounts they should have access to is ideal for cybersecurity, every embedded system manufacturer typically needs some form of access that a customer may not necessarily be able to obtain. A level of access at this level is typically meant to enable service technicians to recover a device from an error, debug an issue, or perform a complicated update that a customer may not be trusted to perform. In

these situations, we recommend that systems designers create functionality in the system where support/service accounts are derived uniquely on a per-device basis. For example, a system based on device-specific criteria that cannot be guessed, such as the combination of hardware serial numbers that are unique. Customers and users should still be afforded a mechanism to change these credentials after knowledge of the risk is accepted, for example, that the system could become unstable or unrecoverable.

In standing with best practices, the following are a few key identity management capabilities to implement on an embedded system:

- Static passwords should be avoided or removed as part of the release process.
- Ensure users are forced to change their passwords upon device setup or activation.
- Ensure that users have the option to change all secrets.
- Provide no built-in or service accounts.
- Enable specific service or privileged capabilities via identity-managed privileges.
- Required service credentials should be unique to each device and not guessable without internal information about the device.
- Implement account lockout threshold to prevent automated brute force attacks.
- Provide a password complexity policy configurable by the user.
- Ensure that passwords are not displayed on the user interface and are always masked, regardless of the user privilege required to view.
- Provide users with the option for password and certificate rotation policies.

Top Five Hacking Methods for IoT Devices

Manasés Jesús

In the realm of the Internet of Things (IoT), where a myriad of devices are interconnected and woven into the fabric of our daily lives, the vulnerabilities that exist can be daunting. As technology advances, so do the methods employed by hackers to exploit these devices. Let us delve into the top five hacking methods for IoT devices, revealing the ingenuity and cunning of those who seek to breach the interconnected world and look at potential solutions to protect against them.

The Trojan Horse

This method takes its name from the ancient tale of the Greeks infiltrating the city of Troy. In the realm of IoT, it involves hackers embedding malicious code within seemingly innocuous software updates or applications. Once these updates are installed, the Trojan Horse (malicious code or software) silently unleashes its payload, granting unauthorized access to the device. Like a wolf in sheep's clothing, this method preys on trust, exploiting the vulnerability of users who unknowingly invite the enemy into their midst.

The Man-in-the-Middle

This method capitalizes on the weakness in communication protocols between IoT devices and their corresponding servers. By intercepting and altering the data transmitted between the two, hackers can gain control over the device or even eavesdrop on sensitive information. Like a shadowy figure lurking in the background, the man-in-the-middle (MITM) silently inserts himself into the conversation, manipulating the flow of information for his own nefarious purposes.

The Zero-Day Exploit

In the world of hacking, a zero-day exploit refers to a vulnerability in software that is unknown to the developer. Hackers who discover such vulnerabilities can exploit them before the developers have a chance to patch them. In the realm of IoT, this method is particularly insidious, as it allows hackers to compromise devices without leaving a trace. Like a phantom slipping through the cracks, the zero-day exploit takes advantage of the unknown, leaving the victim vulnerable and unaware.

The Brute Force Attack

This method is as straightforward as it sounds. It involves hackers using automated tools to systematically guess passwords until the correct one is found. With weak or easily guessable passwords still prevalent in the IoT landscape, this method remains a favorite among hackers. Like a battering ram relentlessly pounding against a fortress, the brute force attack exploits human fallibility, relying on the laziness of users who fail to protect their devices with strong, unique passwords.

The Denial-of-Service (DoS) Attack

This method seeks to overwhelm a device or network with an influx of traffic, rendering it inaccessible to legitimate users. By flooding the target with an overwhelming volume of requests, the hacker effectively cripples the device, preventing it from carrying out its intended function. Like a wave crashing against a fragile sandcastle, the DoS attack capitalizes on the vulnerability of the device, exploiting its inability to handle the onslaught.

As we explore these top five hacking methods for IoT devices, we are reminded of the ever-present threat that looms in the digital world. These hacking methods are not only for IoT devices; they have been used also in other systems. Like a masterful writer crafting a suspenseful tale, hackers employ these methods with skill and finesse, exploiting the weaknesses of technology and human nature alike. However, it is crucial to remember that knowledge is power, and by understanding the methods employed by hackers, we can better protect ourselves in this interconnected landscape.

Securing IoT Applications

Manasés Jesús

The IoT has the potential to revolutionize the world as we know it, and it is rapidly becoming an essential part of our daily lives. The concept of IoT is built around the idea of interconnected devices and sensors that can communicate with each other and perform various tasks automatically. From smart homes and cities to industrial automation and healthcare, IoT offers countless opportunities for innovation and growth. However, with great opportunities come great responsibilities and significant challenges, particularly in the area of cybersecurity.

Securing IoT applications, including securing devices and networks, requires multiple tiers of security controls. There is an increasing number of devices and sensors that are connected to the internet as a result of the IoT's explosive growth, and they are now more open to attacks and threats from cybercriminals. As a result, IoT security is currently one of the most important problems that governments, organizations, and end users must deal with. The risk of cyberattacks and data breaches has greatly risen with the increasing number of connected devices. Therefore, it is paramount to understand and implement the following best practices for IoT security:

Secure communication
: One of the most critical aspects of IoT security is secure communication between devices and networks. All communication channels should be encrypted using strong cryptographic algorithms to prevent unauthorized access and data interception.

Authentication and authorization
: IoT applications must use authentication services to reduce or minimize network risks and breaches. Attackers can enter through the front door due to improper device authentication, posing a security concern. The privileges of IoT device components and applications are limited by access controls—by specifying who is allowed access to the data and IoT devices as well as how much access should be granted.

Access control

A key component of IoT security is access control. Making sure that only authorized users have access to IoT devices and applications is crucial. To prevent unwanted access, access control measures like passwords, biometric authentication, and MFA should be used.

Regular updates

IoT devices are susceptible to security risks, and fresh flaws are regularly found. For this reason, it is crucial to regularly install security patches and upgrades to keep hardware and software up to date. This practice aids in preventing vulnerabilities from being exploited by attackers.

Data protection

IoT devices gather and send private data, including location and personal data. To maintain the confidentiality and privacy of data, it is crucial to use data protection measures including encryption, access control, and anonymization during the phases of transition or storage of the data.

Physical security

IoT security frequently overlooks physical security, despite the fact that it is equally crucial. To avoid manipulation and theft, physical access to IoT devices should be restricted. Devices should also be safeguarded from external elements like temperature, humidity, and power surges.

Monitoring and logging

For identifying and responding to security issues, monitoring and logging are fundamental. IoT devices need to be monitored for unusual activity, and logs need to be examined frequently to spot any security risks.

In a nutshell, applying best practices for securing IoT applications and devices can greatly lower the risk of cyberattacks and data breaches. Nevertheless, IoT security is a complex and difficult endeavor. The aforementioned best practices can help to maintain the safety and privacy of users and their data. Knowing and implementing these best practices can help IoT applications become more secure and reliable.

Application Security in Cyber–Physical Systems

Yaniv Vardi

You'd be hard-pressed to find a physical process in our connected world that isn't controlled and managed over the internet. These so-called cyber–physical systems (CPS) are prevalent everywhere—and behind every good CPS is an application responsible for its intended purpose.

Factory floors, for example, are automated marvels where devices are programmed to build the things central to our lives. Engineers and asset operators use applications known as Engineering WorkStations (EWS) to upload data from devices, download new instructions, and respond to failures in order to ensure that critical services remain available and safe.

Treatment within hospitals and physicians' offices is also increasingly reliant on connected devices that share patient information that rapidly informs diagnosis and treatment. Remote patient monitoring applications, medical diagnosis software, imaging applications, and digital electronic health record systems are just some examples of CPS and their impact on patient care and safety.

Every modern building is essentially a connected device managed by software applications. Smart homes are rife with connected devices that can be managed by an app that allows users to control climate conditions, oversee home surveillance systems, or raise and lower shades. HVAC and elevators in pharmaceutical settings, office buildings, apartments, hospitals, factories, and elsewhere are also reliant on building management systems that are online.

Securely designed applications are the hub of our connected world. The key pivot around CPS and cybersecurity is that the CIA triad of confidentiality, integrity, and availability isn't completely applicable here. Instead, strategically, CPS aims for safety, reliability, and productivity.

Operational technology (OT) at the core of CPS is often intolerant of the downtime that a cyberattack would bring. Ransomware attacks against energy companies, hospitals, and other critical infrastructure sectors have already demonstrated devastating consequences to fuel production and delivery in the case of the Colonial Pipeline, and negative impacts on patient care in dozens of successful attacks against hospitals worldwide.

Threat actors, meanwhile, will always take the path of least resistance. Attacking a device directly requires expertise that raises the cost of an attack significantly for a criminal or state-sponsored group. However, a successful exploit of a commodity vulnerability in a modern application or cloud-based management system can allow an attacker to reach edge devices at scale.

On the OT side, engineering workstations and historian databases are a crossover point sitting between an organization's DMZ—which hosts applications servers and other critical networking technology—and control systems including connected human-machine interfaces (HMIs) and SCADA systems.

A compromised Windows-based engineering workstation can allow an attacker to pivot in either direction of the Purdue Model for ICS. By disrupting an application server, for example, an attacker would impede an organization's ability to accurately understand and process data from field devices, or properly update them with new instructions. A compromise of HMIs and SCADA systems, meanwhile, could be devastating. An attacker who successfully manipulates an HMI can present an asset owner with a false representation of what's really happening around pumps at a water treatment facility, for example. Alterations of chemicals in such a process as a result of these false readings could put human safety at risk.

In a healthcare scenario, an attacker who is able to exploit a known vulnerability to manipulate patient information in an application storing and transmitting that data could negatively impact patient care. Imagine a doctor acting on a patient's vital signs that have been manipulated by a hacker.

Security leaders must understand these gaps and potential consequences. They must work with software and firmware vendors, as well as device manufacturers, to ensure the quality and safety of application code. Stress the importance of dynamic and static code testing, pen testing, and urge vendors and manufacturers to embrace the need for transparency around application code. In healthcare, SBOMs will soon be a mandated facet of application security, one that can bring visibility into programming components that

other industries can adopt to improve vulnerability management, and ultimately the safety and availability of cyber–physical systems.

About the Editors

Reet Kaur

Reet Kaur is a seasoned information security professional experienced in both public and private sectors. She is currently an Executive Director of IT Risk Management & Security at Merck Co. & Inc. Formerly, she held the role of Cabinet Executive VP and Chief Information Security Officer (CISO) at a leading higher-ed institute, Portland Community College in Portland, Oregon. She is also the board member of the audit committee for the Higher Education Coordinating Commission (HECC) for the State of Oregon.

With over 20 years of experience in IT, information security, and risk management leadership roles at *Fortune* 100 and 500 companies like Merck, Nike, Fidelity, AECOM, and CIBC, Reet has established herself as a respected industry expert. Renowned for her expertise in organizational transformation and a change agent, she adopts a unique, globally informed, risk-based approach to information security, data privacy, IT, and digital transformation.

Certified in CISSP, CRISC, CISM, and PMP, Reet is a trusted advisor to technical, executive, and board-level teams. Reet is a champion of diversity and inclusion initiatives and is deeply committed to addressing talent, gender, and underrepresented minority gaps in cybersecurity. A compassionate leader, she mentors aspiring professionals and actively contributes to the industry through speaking engagements and content development on LinkedIn.

In her strategic role, Reet specializes in developing and implementing successful programs in information security, risk management, and data privacy across various industry verticals and geographies.

Chapter 60, “Effective Vulnerability Remediation Using EPSS”

Chapter 78, “Mobile Application Security Using Containerization”

Yabing Wang

Yabing Wang is the VP and Chief Information Security Officer at Justworks, a tech-forward payroll company supporting small businesses. She has been in the technology world for over 25 years and has over 20 years of extensive leadership experience in cybersecurity across different industries. Yabing thrives in transforming security into a business enabler through executive leadership, program delivery, and partnership with all stakeholders. She has built global security practices and strengthened cyber resilience at multiple *Fortune* 100 companies such as Allstate Insurance Company, Alight Solutions, Carrier Corporation, and H-E-B. Before her cyber journey, Yabing studied philosophy and computer science, and during the early days of her career, she worked in application development at Netscape.

Chapter 15, "Beyond Barriers: Navigating the Path to a Successful AppSec Program"

Contributors

Adam Shostack

Adam Shostack is the author of *Threat Modeling: Designing for Security* and *Threats: What Every Engineer Should Learn from Star Wars.* As founder of Shostack + Associates, he focuses on teaching threat modeling. He has decades of experience delivering application security. Early in his career, he helped create the CVE. He serves as a member of the Blackhat Review Board, an advisor to a variety of companies and academic institutions, and an Affiliate Professor at the Paul G. Allen School of Computer Science and Engineering at the University of Washington. You can learn more at *adam.shostack.org*.

Chapter 69, "Learn to Threat Model"

Aldo Salas

With more than 15 years of experience, Aldo Salas has had the opportunity to work on all stages of Application Security, from penetration testing to program management and everything in between. He is currently on a quest to get rid of passwords by leading the application security program at HYPR. Aldo has participated as an OWASP local chapter leader for many years, and he has been active in the bug bounty community as well. Aldo has worked with several technologies and businesses, including financial, healthcare, media and entertainment, education, and information technology.

Chapter 34, "Will Passwordless Authentication Save Your Application?"

Chapter 54, "Demystifying Bug Bounty Programs"

Alexander James Wold

Alexander James Wold is a global cybersecurity manager with over 14 years of experience in security. He has built multiple security programs in complex environments across multiple industries. He is constantly learning and staying on top of the latest developments in the field.

Chapter 85, "LLMs Revolutionizing Application Security: Unleashing the Power of AI"

Allen West

Allen West is a Security Researcher on Akamai's Security Intelligence Response Team who loves investigating threats and building tools. He is currently pursuing his master's in information security and assurance from Carnegie Mellon University. He received his undergraduate degree in cybersecurity from Northeastern University and is a Marine Corps Veteran. During his free time, Allen loves traveling, flying drones, hiking, swimming, or really anything outdoors and adventurous.

Chapter 73, "In Denial of Your Services"

Chapter 74, "Sifting for Botnets"

Alyssa Columbus

Alyssa Columbus is a Vivien Thomas Scholar at Johns Hopkins University and a member of the Spring 2018 Class of NASA Datanauts. Her technical guides, tutorials, and articles have been featured by leading organizations, including *Forbes*, Google, Microsoft, and the Association for Computing Machinery (ACM). Previously, Alyssa has worked as an information security analyst, data scientist, and machine learning researcher, and she is a contributor to open source software with over 49 million downloads. Alyssa holds a Bachelor of Science degree in Mathematics from the University of California, Irvine, and a Master of Science degree in Applied and Computational Mathematics from Johns Hopkins University.

Chapter 1, "Secure Code for Tomorrow's Technology"

Chapter 62, "Integrating Security into Open Source Dependencies"

Andres Andreu

Andres Andreu, CISSP-ISSAP, is the Deputy Chief Information Security Officer (CISO) at Hearst Corporation, ex-CISO at 2U/edX and Bayshore Networks, and a Boardroom Certified Qualified Technology Expert (QTE). Andres is an industry veteran and recognized industry leader whose career has spanned federal government service, corporate America, global consulting, the entrepreneurial journey in the cybersecurity product space, and executive advising. He has won numerous industry awards and is the sole author of *Professional Pen Testing for Web Applications* as well as numerous magazine articles. He holds an internationally granted patent.

Chapter 2, "Pragmatic Advice for Building an Application Security Program"

Chapter 79, "API Security: JWE Encryption for Native Data Protection"

Andrew King

Andrew King is a seasoned technical leader with over two decades of expertise, possessing a 360-degree comprehensive view of the computing landscape. His proficiency spans various domains, including software engineering, computing forensics, UI design, analytics, policy development, federal standards creation, validation, compliance, security operations, corporate infrastructure, and vendor management. With a wealth of experience in diverse industries—commercial product development, higher education as a professor, ecommerce, finance, federal, military operations in direct combat zones—Andrew brings a unique perspective to technology and security. He has served on non-profit and customer customer advisory boards, focusing on the strategic direction of these organizations.

Having navigated roles as engineer and analyst, professor, corporate leader, and cloud expert overseeing one of the world's largest infrastructures in the world, Andrew excels in both offensive and defensive strategies for network and computing systems. His journey includes focusing on building high performing, diverse teams, building cohesion and operational direction among conflicting teams, and leading large-scale technical transformation.

Andrew holds a master's degree in information assurance from Norwich University and a Bachelor of Science degree in computing and software systems from the University of Washington. Beyond his professional achievements, he is passionate about giving back to his community with the volunteer fire and rescue services.

Chapter 16, "Building an Application Security Preparation Mindset"

Angelica Lo Duca

Angelica Lo Duca is a researcher at the Institute of Informatics and Telematics of the National Research Council in Italy. She is also an adjunct professor of Data Journalism at the University of Pisa. She is the author of *Comet for Data Science*, published by Packt Ltd., coauthor of *Learning and Operating Presto* (O'Reilly), and author of *Data Storytelling with Generative AI Using Python and Altair*, published by Manning Publications.

Chapter 86, "Mitigating Bias and Unfairness in AI-Based Applications"

Anuj Parekh

Anuj Parekh has been in the cybersecurity realm working as a security engineer for over six years and is a cybersecurity enthusiast. Anuj specializes in product, infrastructure, and enterprise security. Anuj enjoys hiking and is always up for exploring new parks and hiking trails.

Chapter 17, "How to Assess Security Mindset in Application Design"

Aruneesh Salhotra

Aruneesh Salhotra is an experienced technologist/generalist and servant leader with expertise in development, DevSecOps, cybersecurity, PMO, infrastructure, Kubernetes, AI, the ethical side of AI, GRC, and technical sales. Aruneesh has a deep proficiency in cybersecurity, cloud security, and project management, with professional certifications including C-CISO, CISSP, CKA/CKAD, AWS Security Specialist, and PMP. He serves on multiple advisory boards, including VirSec, Strategio, ICPS, and Dazz, providing valuable counsel and contributing to critical decisions on cybersecurity and diversity to business strategy and growth. Aruneesh is a highly effective communicator and industry thought leader, with numerous awards and recognition for presenting, serving as a panelist, and authoring content for prominent industry events such as ADDO, Elevate, Ignite, QA Forum, RFG, SINET, PBC, Tech Trek, and beyond. Furthermore, Aruneesh is known to foster productive relationships with key industry security bodies, including CISA, EPSS, CFF, and PBC, to help shape security policies and promote better cybersecurity practices. He is also an active investor in the cybersecurity and GenAI domain (angel investor, limited partner in VC firms).

Chapter 44, "Modern Approach to Software Composition Analysis: Call Graph and Runtime SCA"

Chapter 55, "EPSS: A Modern Approach to Vulnerability Management"

Chapter 77, "Mobile Security: Domain and Best Practices"

Ayman Elsawah

Ayman Elsawah has managed and led security for startups of all sizes and shapes. His technical focus areas are Identity and access management, product security, and infrastructure security. He is an active member of the cybersecurity community, regularly speaking at industry events and sharing his insights and expertise on social media.

Ayman is the author of *Breaking In: A Practical Guide to Starting a Career in Information Security*, the host of the podcast *Getting Into Infosec*, runs a newsletter called *Last Week as a vCISO*, and is a coffee enthusiast. Check out his personal site at *https://coffeewithayman.com* or connect on social media @coffeewithayman.

Chapter 18, "Getting Your Application Ready for the Enterprise"

Brook S.E. Schoenfield

Books by Brook S.E. Schoenfield include *Building In Security at Agile Speed* (Auerbach, 2021, coauthored with James Ransome), *Secrets of a Cyber Security Architect* (Auerbach, 2019) and *Securing Systems: Applied Security Architecture and Threat Models* (CRC Press, 2015). He coauthored *The Threat Modeling Manifesto* (2020), *Avoiding the Top 10 Security Design Flaws* (IEEE, 2014) and *Tactical Threat Modeling* (SAFECode, 2017). He has technically led five AppSec/software security programs and four consulting practices. Currently, Brook is CTO and Chief Security Architect at Resilient Software Security. He also teaches at the University of Montana and regularly speaks at conferences and on podcasts/webinars.

Chapter 3, "AppSec Must Lead"

Chapter 80, "APIs Are Windows to the Soul"

Caroline Wong

Caroline Wong is the Chief Strategy Officer at Cobalt. She has over 15 years of cybersecurity leadership, including practitioner, product, and consulting roles. Caroline authored the popular textbook, *Security Metrics: A Beginner's Guide*. She teaches cybersecurity courses on LinkedIn Learning and hosts the *Humans of InfoSec* podcast.

Chapter 4, "Solving Problems for Application Security"

Cassie Crossley

Cassie Crossley, Vice President of Supply Chain Security in the global Cybersecurity & Product Security Office at Schneider Electric, is an experienced cybersecurity technology executive in Information Technology and Product Development. She is the author of *Software Supply Chain Security: Securing the End-to-End Supply Chain for Software, Firmware, and Hardware* (O'Reilly). She has many years of business and technical leadership experience in supply chain security, cybersecurity, product/application security, software/firmware development, program management, and data privacy.

Cassie has designed frameworks and operating models for end-to-end security in software development lifecycles, third-party risk management, cybersecurity governance, and cybersecurity initiatives. She is a member of the CISA SBOM working groups and presents frequently on the topic of SBOMs and supply chain security.

Cassie has an MBA from California State University, Fresno, and a Bachelor of Science degree in Technical and Professional Communication with a specialization in computer science.

Chapter 63, "Supplier Relationship Management to Reduce Software Supply Chain Security Risk"

Chadi Saliby

Chadi Saliby is a cybersecurity subject matter expert with extensive knowledge and experience in the field. He has worked with diverse organizations ensuring the protection of sensitive data and uplifting their cybersecurity posture. He has designed, architected, and integrated complex security solutions for private enterprises and government departments. He identifies and implements cybersecurity controls, such as threat and vulnerability

management, incident response, digital forensics and incident response (DIFR) capabilities, cyber threat hunting, and intelligence.

Chapter 5, "Securing Your Enterprise Applications"

Charan Akiri

Charan Akiri is a Senior Application Security Engineer at Reddit, with over 13 years of experience in the software industry. He began his career as a software developer and later transitioned into security due to his strong passion for the field. His main focus is on solving security, compliance, and privacy issues within software architecture. He has a keen interest in application security, cloud security, infra security, pen testing, cryptography, and ML. In addition, Charan holds several security certifications and developer certifications, including AWS Cloud Developer and AWS Security certifications. Furthermore, he has obtained certifications such as GPEN, GCSA, GWAPT, GSSP-Java, and he is a member of the GIAC Advisory Board.

Chapter 81, "API Security: The Bedrock of Modern Applications"

Chenxi Wang

Dr. Chenxi Wang is the founder and general partner of Rain Capital, a Silicon Valley–based venture fund. A well-known investor, technologist, and thought leader in the cybersecurity industry, Dr. Wang also serves on the Board of Directors for MDU Resources, a critical infrastructure energy company. Chenxi held executive positions at Twistlock, Intel, and Forrester Research and as vice-chair for the OWASP Foundation. Chenxi was named by *Fortune* as one of the top cyber investors in the world, and by CyberRisk Alliance as Women Investor of the year. She was recognized as a Woman of Influence by *SC Media* magazine.

Chapter 82, "API Security Primer: Visibility"

Chapter 83, "API Security Primer: Risk Assessment, Monitoring, and Detection"

Chapter 84, "API Security Primer: Control and Management"

Chloé Messdaghi

Chloé Messdaghi stands out as a distinguished security executive, acclaimed for her exceptional expertise in advising and shaping solutions that elevate security teams and set industry standards ablaze. A highly sought-after speaker and a trusted luminary in major publications such as *Forbes* and *Business Insider*, Chloé has rightfully earned the title of a power player in cybersecurity, acknowledged by both *Business Insider* and *SC Media*.

Exemplifying an unwavering commitment to positive impact, she not only excels but leads as an expert solutionist and strategist in the realms of cybersecurity, sustainability, and human rights. Chloé's exceptional career and unyielding commitment to driving positive change not only position her as a leader but as an unstoppable force in these dynamic fields, forging the future with unparalleled expertise and purpose.

Chapter 64, "Fortifying Open Source AI/ML Libraries: Garden of Security in Software Supply Chain"

Christian Ghigliotty

Christian Ghigliotty is a security technologist with over nine years of experience across multiple disciplines within information security, serving as both practitioner and leader. He has also served as a technical editor for publications and educator in acadmedia and certificate programs. Christian was part of the influential security program at Etsy, and helped build the security organization at Compass, a tech-enabled real estate brokerage. He is currently building the Security Architecture & Engineering function at the New York-based tech company Justworks.

Chapter 6, "Developers as Partners in Application Security Strategy"

Daniel Ting

Daniel Ting is a cybersecurity sherpa by day who lives at the intersection of human-centered design and cybersecurity, helping teams safely navigate building amazing things in cyberspace. By night, they have spoken multiple times at DefCon Villages, BSides Melbourne, and many other industry conferences. They are active contributors of the OWASP Education Committee and leads the OWASP How to Get into AppSec project, the OWASP Melbourne chapter, and various cybersecurity community groups in Australia.

Chapter 7, "Be an Awesome Sidekick"

Darryle Merlette

Darryle Merlette, CISSP, is a computer scientist with over 25 years of experience in the software and security industry. He is currently the Executive Director of Product Security at NIKSUN, Inc., a private tech company specializing in network security and performance monitoring. He holds a BSE from Princeton University and an MSE from the University of Michigan, both in computer science with an emphasis in AI.

Chapter 19, "Reductio Ad Applicationem Securitatis"

David Lindner

David Lindner is an experienced application security professional with over 20 years in cybersecurity. In addition to serving as a chief information security officer, he has worked within multiple disciplines in the security field—from application development to network architecture design and support, to IT security and consulting, to security training, to application security. He has worked with many clients across numerous industry sectors, including financial, government, automobile, healthcare, and retail.

Chapter 45, "Application Security Testing"

Chapter 46, "WAF and RASP"

David Stokes

Dave Stokes is a Technology Evangelist for Percona. He is the author of *MySQL & JSON: A Practical Programming Guide*. Dave resides in Texas with the mandated pickup truck, hound dogs, and guitars.

Chapter 35, "Securing Your Databases: The Importance of Proper Access Controls and Audits"

Diogo Miyake

Diogo Miyake is a Senior Big Data Architect at Thoughtworks. He has worked since 2019 with big data helping companies to deliver data products in financial, marketing, retail,

and airlines. He started to improve data products with security in 2021 when he learned about using DevSecOps and SOC to create secure data pipelines and improve deliveries with security standards to guarantee privacy and anonymization of customer data.

Chapter 36, "DataSecOps: Security in Data Products"

Chapter 37, "Data Security Code and Tests"

Erkang Zheng

Erkang Zheng is the founder and CEO of JupiterOne, which helps customers proactively manage the risk and compliance of their cyber assets. Additionally, he is a strategic advisory board member to the Computer Science department at the North Carolina State University and a CxO Trust Council Advisor at the Cloud Security Alliance.

Previously, he was CISO of LifeOmic and Head of Software Security at Fidelity Investments Personal Investing. He has held several security leader and practitioner roles at IBM, including Global Practices Leader, and GTM strategy and product management for Security Services.

Erkang is a strong supporter of making security a basic right that all companies should have, at a cost they can afford, and in a way that allows them to build a security program the right way over time. He is an engineer by training, an entrepreneur at heart, and passionate about combining innovation and execution to deliver practical solutions that address challenges at their root cause.

Erkang earned a BS and an MS degree in computer science from North Carolina State University. He holds five patents, multiple industry certifications, and is a regular speaker at major security conferences including RSA, BSides, and many community events.

Chapter 8, "Understanding the True Boundaries of Modern Applications"

Chapter 20, "Automating the Risk Calculation of Modern Applications"

Fayyaz Rajpari

Fayyaz Rajpari is the CEO and Managing Partner for Intelliguards Corporation. The firm is focused on security operations and advanced cybersecurity solutions. Fayyaz also sits on numerous advisory boards for cybersecurity tech startups.

His expertise includes serving as an Incident Responder and then later driving products for Symantec, Mandiant, FireEye, Recorded Future, and Gigamon. He also led Optiv's Global security consulting practice.

Chapter 75, "Incident Response for Credential Stuffing Attacks"

Han Lievens

Han Lievens has been contributing to the field of Information Security for over 20 years, with expert focus on engineering and threat response. As the field continues to rapidly evolve, Han helps customers stay on top of the latest cyber threats and new technologies. He currently spends most of his time on all things SIEM, SOAR, and DevSecOps.

Chapter 21, "A Coordinated Approach to a Successful DevSecOps Program"

Heather Hinton

Heather Hinton is currently the CISO at PagerDuty, having previously been the CISO at RingCentral and CISO of IBM's Cloud and Cognitive Systems divisions. Heather has been involved in computer security for over 30 years. She has a PhD in Electrical & Computer Engineering from the University of Toronto. She has taught computer security at the University of Toronto, the Nortel Institute, and Harvard (through the Harvard Extension School). Heather has over 75 published patents in various areas of computer security, including identity management, federated identity management, mobile security, governance, and security assessments.

Chapter 87, "Secure Development with Generative AI"

Helen Umberger

Helen Umberger is a software architect with over 20 years experience on multiple platforms, business specialties, and products. She has on-the-job experience with complete life cycles of large projects.

Chapter 22, "What Makes Someone a Developer?"

Hussain Syed

Hussain Syed is a pioneer in the successful designing, implementation, and secure operation of large-scale enterprise IT infrastructure (data networking, information security, hybrid multicloud operations management. He is adept in managing hybrid multicloud-based network security operations as part of enterprise digital transformation strategy. Hussain is an Agile coach and has worked very closely with application cloud native application modernization.

Chapter 23, "Total AppSec"

Idan Plotnik

Idan Plotnik is cofounder and CEO of Apiiro, the leader in application security posture management (ASPM). He is a serial entrepreneur and product strategist, bringing more than 20 years of experience in cybersecurity. Idan served at the IDF in a cybersecurity elite unit and was Director of Engineering at Microsoft following the acquisition of Aorato—a pioneer in the UEBA space—where he served as the founder and CEO.

Chapter 70, "Understanding OWASP Insecure Design and Unmasking Toxic Combinations"

Izar Tarandach

With more than 25 years of security experience, Izar Tarandach is the coauthor of *Threat Modeling: A Practical Guide for Development Teams* and a member of the Threat Modeling Manifesto group of authors. He has extensive experience exploring both the hard and soft skills of application security. He is currently a Senior Principal Security Architect at SiriusXM.

Chapter 24, "You're More Than Your Job"

Chapter 69, "Learn to Threat Model"

Jacqueline Pitter

Jacqueline Pitter, CISSP, is a Senior Strategic Consultant with Vantage TCG. Prior to joining Vantage, Jacqueline was the Chief Information Security Officer (CISO) and Senior Infrastructure Administrator at Reed College in Portland, OR, and before that a Software Design Engineer with Hewlett-

Packard. She has more than 22 years of experience in higher education and corporate IT arenas with a strong focus on information security, infrastructure administration, and software design. Jacqueline holds an MS in Computer Sciences from University of Wisconsin–Madison, a BA in Mathematics from Reed College, and certificates in Information Assurance and Information System Security from University of Washington. Jacqueline currently lives in Portland, Oregon, where she spends too much time thinking about sailboat racing.

Chapter 47, "Zero Trust Software Architecture"

Jason Sinchak

Jason Sinchak leads Level Nine's Product Security and is the Founder/CEO of Elton, a Medical Device Cybersecurity company. Jason began his career as a penetration tester and has brought that attacker mindset along with him for over 15 years of advising clients. Jason is regularly sought out as an advisor, keynote speaker, and industry commentator who is able to connect the deep technical aspects of security with insight at an executive decision-making level.

Chapter 92, "Secure Code for Embedded Systems"

Chapter 93, "Platform Security for Embedded Systems"

Chapter 94, "Application Identity for Embedded Systems"

Josh Brown

Josh Brown brings a thoughtful and dedicated approach to his role as an AppSec engineer at Datavant. With over two decades in cybersecurity and system engineering, Josh has contributed to a range of projects and teams, including Patreon, Amazon Web Services, and Booz Allen Hamilton. His focus has been on enhancing cloud security, refining threat modeling processes, and creating secure system designs. Committed to lifelong learning, he is currently furthering his education working on his Master of Science in Software Development degree at Grand Canyon University. An active participant in DEF CON and various security competitions for over 20 years, Josh values community and knowledge sharing in the ever-evolving landscape of application security. He is known for his collaborative spirit and a down-to-earth approach that has earned him respect in the field.

Chapter 71, "The Right Way to Threat Model"

Chapter 88, "Managing the Risks of ChatGPT Integration"

Jyothi Charyulu

Jyothi Charyulu has led an enterprise SSDLC program and revamped it with the use of automation and integration within platform engineering processes and tools at two large *Fortune* 500 companies. She is a published author at the Purple Book of Software Security.

Chapter 25, "TAP Into the Potential of a Great SSDLC Program with Automation"

Karen Walsh

Karen Walsh is a lawyer and former internal auditor-turned-subject-matter expert in cybersecurity and privacy regulatory compliance who provides consulting and content services for cybersecurity startups, translating technology features into business-oriented and compliance solutions. In 2023, she published her newest book, *Security-First Compliance for Small Businesses* (Taylor & Francis Group).

Chapter 65, "SBOM: Transparent, Sustainable Compliance"

Larry W. Cashdollar

Larry W. Cashdollar has been working in the security field as a vulnerability researcher for more than 20 years and is currently a Principal Security Researcher on the Security Intelligence Response Team at Akamai. He studied computer science at the University of Southern Maine. Larry has documented more than 300 CVEs and has presented his research at BotConf, BSides Boston, Carnegie Mellon, and DEF CON. His research has been covered by ZDNet, Slashdot, The Register, Ars Technica, Bleeping Computer, Dark Reading, Yahoo, and MSN.

Chapter 26, "Vulnerability Researcher to Software Developer: The Other Side of the Coin"

Laura Bell Main

With over 20 years of experience in software development and application security, Laura Bell Main specializes in bringing application security and secure development practices into organizations worldwide.

She is the cofounder and CEO of SafeStack, an online education platform offering flexible, high-quality secure development training for fast-moving companies.

Laura is an experienced conference speaker, trainer, and regular panel member and has spoken at various events such as BlackHat USA, NDC, RenderATL, and OSCON on application security, DevSecOps, secure development, and security mindset. She is also the coauthor of *Agile Application Security and Security for Everyone*.

Chapter 27, "Strategies for Adding Security Rituals to an Existing SDLC"

Lauren Maffeo

Lauren Maffeo is an award-winning author and designer who currently works as a senior service designer at Steampunk, a human-centered design firm serving the US federal government. She is also a founding editor of Springer's *AI and Ethics* journal and an adjunct lecturer in the Interaction Design program at The George Washington University. Lauren has written for *Harvard Data Science Review*, *Financial Times*, and *The Guardian*, among other publications. Lauren is a fellow of the Royal Society of Arts, a former member of the Association for Computing Machinery's Distinguished Speakers Program, and a member of the International Academy of Digital Arts and Sciences, where she helps judge the Webby Awards. Her first book, *Designing Data Governance from the Ground Up*, is available from The Pragmatic Programmers and was adapted into a LinkedIn Learning course.

Chapter 38, "Data Security Starts with Good Governance"

Laxmidhar V. Gaopande

Laxmidhar V. Gaopande has a bachelor's degree in mechanical engineering from VNIT Nagpur India, a master's degree in technology from IIT Madras, and a management diploma from Symbiosis Pune. Laxmidhar has been in the IT industry for over 35 years and has worked in the US and UK. He has operated at various CXO levels. Laxmidhar has three patents in the US and

has published more than 10 papers in national and international conferences. He was invited as visiting faculty to teach IT for Global MBA students in Dubai, and he has mentored EMBA students at the Asian Institute of Management in Manila, Philippines.

Chapter 9, "Common Best Practices in Application Security"

Louisa Wang

Louisa Wang is a Security Professional with expertise in cutting-edge cloud security architecture, application security, DevSecOps, IAM, and infrastructure security. Her strong cloud and security skills have been utilized at *Fortune* 500 companies across a broad range of industries that include software, aviation, financial services, pharmaceutical, ecommerce, IT services, and retail. She is a proven leader who drives successful business outcomes in cloud adoption, software security, risk governance, M&A security, training awareness, vulnerability management, security operation, and so on. Louisa is a lifelong learner, with an MS in Computer Science and BA in Business. She is a member of OWASP, CSA, ISACA, and ISC2. She has expertise as an AWS, GCP, and Azure Solution Architect and trained/certified in CISM, CCSK, CISSP, and CCSP. With her unique combination of technical cloud expertise and business acumen, she builds relationships across silos in highly matrixed organizations.

Chapter 39, "Protect Sensitive Data in Modern Applications"

Luis Arzu

Luis Arzu is currently the CISO at Urban One, Inc., the largest distributor of urban media content in the US. Key successes include improving security governance, reducing security incidents, accelerating incident response, and advancing critical systems security, scalability, availability, and costs. In addition, he fostered a culture of security awareness by shaping and delivering an enterprise cybersecurity awareness training program.

Luis merges a proactive approach with continual scanning of the industry landscape to stay ahead of the curve in enterprise cybersecurity. His leadership and achievements at Urban One have been featured in industry media, such as The Global Center for Cyber, NOPSEC, and Security Current.

Chapter 56, "Navigating the Waters of Vulnerability Management"

Lütfü Mert Ceylan

Lütfü Mert Ceylan is a security researcher who specializes in the web application field of cybersecurity. He is an OWASP Project Leader and owner of the OWASP Top 25 Parameters project. He is also the OWASP Poland Chapter Board Member and the founder of TR Bug Hunters, Turkey's active security researcher and bug hunter community. He is specifically focused on server-side and client-side attacks. He is actively involved in the bug bounty field and he has helped detect and exploit over 500 security vulnerabilities over 75 web applications for companies such as Apple, Oracle, Adobe, Mozilla, and more than 30 other prestigious companies and organizations. He is currently pursuing a bachelor's degree from the Warsaw University of Technology. He can be reached via Twitter @lutfumertceylan, on his website *lutfumertceylan.com.tr* and on LinkedIn *https://www.linkedin.com/in/lutfumertceylan*.

Chapter 57, "Safeguarding the Digital Nexus: "Top 25 Parameters to Vulnerability Frequency""

Chapter 58, "Unveiling Paths to Account Takeover: Web Cache to XSS Exploitation"

Chapter 59, "Sometimes the Smallest Risks Can Cause the Greatest Destruction"

Manasés Jesús

Manasés Jesús has been happily hacking since high school, employing different programming languages, tech stacks, and software development methodologies. He has architected and implemented distributed systems, mobile and cloud native applications, as well as led projects and delivered working software to a range of international corporations in the IT, banking, and IoT sectors. He enjoys learning, teaching, giving workshops, connecting with communities, and helping developers discover their opportunities for innovation with APIs and ecosystems. He has trained fellow software engineers, published scientific articles, spoken at international conferences, and has also been featured in *97 Things Every Cloud Engineer Should Know*.

Chapter 28, "Challenges and Considerations for Securing Serverless Applications"

Chapter 95, "Top Five Hacking Methods for IoT Devices"

Chapter 96, "Securing IoT Applications"

Manuel Walder

As a security engineer with a huge enthusiasm for the various topics of application security, Manuel Walder supports development teams in developing secure software. He follows a holistic approach, which includes perspectives from the offensive, defensive, and technical as well as from the security management complex, to achieve a sustainable defense against today's cyber threats. In addition, he helps operations teams run their applications securely, implement attack mitigation measures, and respond quickly when vulnerabilities arise.

Chapter 40, "Leverage Data-Flow Analysis in Your Security Practices"

Maria Nichole Schwenger

Maria Nichole Schwenger is a seasoned information security executive. She leverages her deep expertise across cybersecurity, privacy and compliance, cloud modernization, and software development to spearhead transformative digital journeys. Renowned for her leadership in integrating emerging technologies like AI/GenAI, DevSecOps/SRE, Blockchain, IoT/Edge, and cloud native optimization, she seamlessly delivers innovative business capabilities. The transformative results of her work demonstrate a substantial increase in ROI, productivity gains, and enhanced business agility.

Chapter 41, "Embracing a Practical Privacy Paradigm Shift in App Development"

Mark S. Merkow

Mark S. Merkow, CISSP, CISM, CSSLP, works at Freeport-McMoRan in Phoenix, Arizona, leading application security architecture and engineering efforts in the office of the CISO. Mark is a faculty member at the University of Denver where he works instructing online courses in topics across the information security spectrum, with a focus on secure software development. He also works as an advisor to the University of Denver's Information and Computing Technology Curriculum Team for new course development and changes to the curriculum. Mark has authored or coauthored 18 books on IT and AppSec and is a contributing editor to 4 others.

Chapter 10, "AppSec Is a People Problem—Not a Technical One"

Matthew Coles

Matthew Coles is a product security architect and secure systems engineering leader for connected devices and the ecosystems and processes that create, enable, and support them. He is currently a Distinguished Member of Technical Staff in the Product and Application Security team at Dell Technologies and holds a CSSLP certification from ISC2. He coauthored *Threat Modeling: A Practical Guide for Development Teams*, is a member of the Threat Modeling Manifesto group of authors and is active in security community initiatives.

Chapter 69, "Learn to Threat Model"

Michael Bray

Michael Bray is a global award-winning cybersecurity professional. He is the Chief Information Security Officer and HIPAA Security Official for the Vancouver Clinic. Michael actively participates in public/private sector GRC associations, committees, and boards of directors.

Chapter 11, "Empowering Application Security Professionals Through Cybersecurity Education"

Michael Freeman

Michael Freeman has over two decades of developing offensive and defensive capabilities for various government and *Fortune* 100 companies. He is currently the Head of Threat Intelligence at Armis.

Chapter 76, "Advanced Threat Intelligence Capabilities for Enhanced Application Security Defense"

Michael Xin

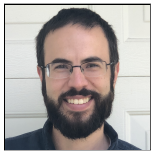
Michael Xin is the Head of Product and Application Security at FactSet, where he oversees a global team dedicated to ensuring the security of FactSet products and applications. Prior to joining FactSet, he held the position of Director of Security Assessments and Risk Mitigation at McAfee. In that role, he led a global team in identifying and addressing security vulnerabilities in McAfee products, as well as on-premises and public cloud infrastructure, within specified timeframes to enhance their overall security landscape.

Outside of work, Michael actively contributes as a leader in the OWASP San Antonio chapter, where he helps educate others about security.

Chapter 12, "Why You Need a Practical Security Champions Program"

Chapter 89, "Automation, Automation, and Automation for AppSec"

Nathaniel Shere



Nathaniel Shere works as a security consultant specializing in penetration testing and secure coding. He loves to teach others and dreams of a day when we will have a truly secure internet. In his free time, he enjoys spending time with his family, reading, and playing board games.

Chapter 29, "Using Offensive Security to Defend Your Application"

Neatsun Ziv

Neatsun Ziv is the CEO and cofounder of OX Security, the first Active ASPM platform that integrates application security practices throughout the SDLC. Before OX Security, he served as the Vice President of Cyber Security at Check Point, responsible for leading all cyber initiatives. His team was among the first to respond to SolarWinds and NotPetya, collaborating closely with law enforcement agencies, including Interpol and Local CERT. As a seasoned entrepreneur with extensive experience in cybersecurity, Neatsun is a frequent speaker at global forums, including CPX. He served in the Israeli Defense Forces Cyber Intelligence Unit, contributing his expertise to national security efforts. Neatsun holds a Bachelor of Science degree with honors from Israel's Open University and an MBA, graduating *magna cum laude* from the Technion.

Chapter 90, "Will Generative and LLM Solve a 20-Year-Old Problem in Application Security?"

Nielet D'mello

Nielet D'mello is a security engineer at a leading observability SaaS company where she focuses on secure design, implementation, and deployment of products and infrastructures. Formerly a software engineer, she likes to approach security from a developer-centric lens.

She enjoys writing and public speaking and loves to share her learnings publicly. Outside of that, she mentors grad students on various career design and growth topics.

Chapter 30, "Beyond "No": The Modern Paradigm of Developer-Centric Application Security"

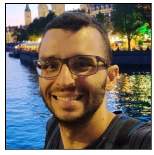
Chapter 31, "Security Paved Roads"

Niels Tanis

Niels Tanis has a background in .NET development, pen testing, and security consultancy. He is Microsoft MVP and has been involved in breaking, defending, and building secure applications. He joined Veracode in 2015; currently he works as a security researcher on a variety of languages and technologies related to Veracode's Binary Static Analysis service. He also is an international speaker and loves to talk about software development, new technologies, and, of course, application and software security. He is married, a father of two, and lives in a small village just outside Amersfoort, The Netherlands.

Chapter 66, "Secure the Software Supply Chain Through Transparency"

Periklis Gkolias



Periklis Gkolias is a security engineer with more than 13 years of experience in the industry and in many aspects of informatics, such as programming, DevOps, and technical leadership. He is a technical blogger and has coauthored the book *Your First Year in Code*. He is quite interested in exploit development and in combining security with other informatics domains like AI and Big Data.

Chapter 13, "The Human Firewall: Combat Enemies by Improving Your Security-Oriented Culture"

Pragat Patel

Pragat Patel is a student at the University of Pennsylvania studying neuroscience and philosophy, with a passion for sparking change in health policy. He aspires to work in healthcare, solving long-term issues in medicine. Pragat is currently working on projects in epidemiology and biostatistics, alongside a health tech startup he has founded.

Chapter 48, "Rethinking Ethics in Application Security: Toward a Sustainable Digital Future"

Raj Badhwar

Raj Badhwar has 28 years of experience working for large firms where he held senior cybersecurity and IT leadership roles, including CISO and CTO. Raj has the following certifications: CISSP, CEH, OCP, and Finra Series 99. He has authored four security books, coauthored 14 security patents, has presented at various security conferences, and advises several boards on cybersecurity strategy.

Chapter 49, "Modern WAF Deployment and Management Paradigms"

Rakesh Kulkarni

As a technology leader with 15 years of experience, Rakesh Kulkarni has navigated the complexities of the tech industry through leadership, strategy, and architecture roles in the semiconductor, startups, retail, and healthcare sectors. He has an engineering background in computer science and he advocates the importance of quantum computing technology. Rakesh embraces the mantra that "change is the only constant; uncertainties lead to probabilities, which then lead to realities." He is currently expanding his strategic and entrepreneurial skills as an MBA student at CMU Tepper Business School.

Chapter 42, "Quantum-Safe Encryption Algorithms"

Sandeep Kumar Singh

Sandeep Kumar Singh has over 18 years of extensive experience involving the SDL, security automation, and development. Currently serving as a director, Sandeep oversees the security assessment effort for FactSet's applications and products, ensuring robust security measures are in place to protect the organization's applications, products, and infrastructure. Before this role, Sandeep spent 14 years at McAfee, where he successfully managed the security team with a primary focus on implementing and enforcing the SDL throughout the organization. Sandeep's areas of expertise encompass secure design, threat modeling, security automation, cloud security, the Product Security Incident Response Team (PSIRT), SAST, DAST, SCA, and penetration testing.

Chapter 12, "Why You Need a Practical Security Champions Program"

Chapter 32, "AppSec in the Cloud Era"

Sausan Yazji

Sausan Yazji is an executive advisor who aims to assist organizations and their C-level leaders realize their business value through the adoption of emerging technologies, innovation in the cloud, and utilization of AI/ML. Sausan has more than 25 years of experience in multiple industries. She led cross-functional teams to build industry-specific solutions and to ensure that these solutions meet the high bar for quality, performance, and security. Sausan holds a PhD in Computer Engineering and is an Advisory Board member for the MSIT program at Northwestern University.

Chapter 43, "Application Integration Security"

Sean Poris

Sean Poris has led information security functions including product and application security, vulnerability management operations, security engineering, cloud and container security, bug bounty, and a variety of other technical engineering and IT roles. He's been in security and IT leadership for years at companies such as Yahoo, College Board, and IBM.

He's passionate about providing the bedrock of security solutions that ensure the protection of critical customer and company data at the enterprise scale. The aspect of security he enjoys most is building solutions that guide the delivery of resilient products and infrastructure.

At Yahoo and College Board, his teams equipped software development groups with the critical tools, training, and processes to address information security holistically throughout the product development lifecycle.

Sean currently serves on the Board of the Northern Virginia Chapter of OWASP. In that capacity, he held the role of Global OWASP AppSec DC Conference Co-Chair.

In his spare time, Sean loves riding his road bike, playing tennis, and listening to great music, from classical to heavy metal and everything in between.

Chapter 61, "Bug Bounty—Shift Everywhere"

Shawn Evans

Shawn Evans is an experienced cybersecurity professional with more than 15 years of offensive security experience across multiple industries with a focus on information security and a background in software development. Shawn has authored a number of open tools, some of which are included with various Linux distributions such as Ubuntu, Kali, Arch, and ParrotOS. During the week, Shawn serves as the Head of Research at NopSec, a Cyber Threat Exposure Management platform provider. There he leads NopSec's research and pen testing efforts, serving large enterprise companies.

Chapter 50, "Do You Need Manual Penetration Testing?"

Chapter 51, "Bash Your Head"

Sounil Yu

Sounil Yu is the creator of the Cyber Defense Matrix and the DIE Triad, which are reshaping approaches to cybersecurity. He's a Board Member of the FAIR Institute, a visiting fellow at GMU Scalia Law School's National Security Institute, guest lecturer at Carnegie Mellon, and advises many startups. Sounil is the cofounder and Chief AI Safety Officer at Knostic and previously served as the CISO at JupiterOne, CISO-in-Residence at YL Ventures, and Chief Security Scientist at Bank of America. Before Bank of America, he helped improve information security at several *Fortune* 100 companies and federal government agencies. Sounil has over 20 granted patents and was recognized by multiple publications as a top CISO and one of the most influential people in security. He is a recipient of the SANS Lifetime Achievement Award and was inducted into the Cybersecurity Hall of Fame. He has an MS in Electrical Engineering from Virginia Tech and a BS in Electrical Engineering and a BA in Economics from Duke University.

Chapter 14, "Shifting Everywhere in Application Security"

Tanya Janca

Tanya Janca, also known as SheHacksPurple, is the best-selling author of *Alice and Bob Learn Application Security* (Wiley). She is also the Head of Education and Community at Semgrep, sharing content and training that revolves around teaching everyone to create secure software. Tanya has been coding and working in IT for over 25 years, has won countless awards, and

has been everywhere from public service arenas to tech giants, writing software, leading communities, founding companies and "securing all the things." She is an award-winning public speaker and active blogger and has delivered hundreds of talks on six continents. She values diversity, inclusion, and kindness, which shines through in her countless initiatives.

Tanya is an advisor to Nord VPN and Aiya Corp; on the faculty at IANs Research; and founder of We Hack Purple, OWASP DevSlop, #CyberMentoringMonday, WoSEC.

Chapter 52, "Exploring Application Security Through Static Analysis"

Travis Felder

Travis Felder is a forward-thinking cybersecurity consultant skilled in information security and risk management, with a particular focus on leveraging AI and ML to address security challenges. His expertise spans cloud services, containers, and DevOps, enabling him to drive the integration of advanced security measures from the outset of development processes.

Chapter 67, "Unlock the Secrets to Open Source Software Security"

Tyler Young

As Chief Information Security Officer at BigID, Tyler Young is responsible for the development and implementation of BigID's security strategy (Cyber and Product Security), as well as developing security use cases for BigID's products. He also serves as a board member on *CDO* Magazine, Glilot Capital, GTM Capital, and Merlin Capital's Security Advisory Boards, as well as an advisor for several early-stage security startups. Before joining BigID, Tyler served as the Head of Security at Relativity, building out Relativity's security program, Calder7. He also held positions as Global Forensics Manager, Digital Forensics Incident Response consultant, as well as leading internal incident response investigations. He gained a variety of cybersecurity and strategy experience from BigID, Relativity, Zurich Insurance, RSM, Arete IR, and a government agency.

Chapter 53, "Introduction to CI/CD Pipelines and Associated Risks"

Vinay Venkatesh

Vinay Venkatesh has spent about eight years working on product security/AppSec for products ranging from building automation, and home automation to AI and web-based offerings. Before that, he spent 12 years in software development covering everything from programming, testing, software design, and architecture.

Chapter 72, "Attack Models in SSDLC"

Viraj Gandhi

Viraj Gandhi is a results-oriented, data-driven, hands-on security professional. She successfully led "shift-to-left" transformations of security programs to solve challenges in open source software security, application security, cloud security, and privacy domains. Her strengths include strategic planning, building high-performance teams, analyzing security trends, reviewing secure architecture design, performing code reviews, and penetration testing.

Chapter 68, "Leverage SBOMs to Enhance Your SSDLC"

Yaniv Vardi

Yaniv Vardi is Claroty's Chief Executive Officer. Yaniv is a highly accomplished entrepreneur with more than two decades of executive leadership experience in the cybersecurity and enterprise solution industry. He has established a long-standing and impressive track record of developing and executing global business strategies and directing worldwide growth.

Chapter 97, "Application Security in Cyber–Physical Systems"

Yashvier Kosaraju

Yashvier Kosaraju is the CISO at Sendbird where he oversees Security, Compliance & IT. He has worked with Twilio, Box, and iSEC Partners in the past. He has been working in security for more than a decade. Yashvier has worked in a variety of roles, ranging from consulting to enterprise security teams. He is a big proponent of security through automation and defense in-depth solutions.

Chapter 33, "Code Provenance for DevSecOps"

Yasir Ali

Yasir Ali is the founder and CEO of Polymer Data Loss Prevention for SaaS apps and AI. Before founding Polymer, Yasir consulted with large financial institutions to solve problems related to data, technology, and regulations. Prior to that, he worked as a bond trader at Bear Stearns, Barclays, and various hedge funds. Yasir is passionate about risk management, data security, and compliance. He regularly writes and speaks about data governance, ROI of security investments, and how organizations can reduce the risk of inadvertent or malicious leakage of PII, PHI, and other sensitive data.

Chapter 91, "Understand the Risks of Using AI in Application Development"

O'REILLY®

Learn from experts.
Become one yourself.

Books | Live online courses
Instant answers | Virtual events
Videos | Interactive learning

Get started at oreilly.com.

©2023 O'Reilly Media, Inc. O'Reilly is a registered trademark of O'Reilly Media, Inc. 175_6x9