TAG

HOW TO OVERCOME THE CHALLENGES OF THE MODERN SECURITY LEADER

DR. EDWARD AMOROSO,
CHIEF EXECUTIVE OFFICER, TAG

SecurityCompass
THE SECURITY BY DESIGN COMPANY

HOW TO OVERCOME THE CHALLENGES OF THE MODERN SECURITY LEADER

EDWARD AMOROSO, CEO, TAG

Modern development teams must deal with a set of significant cybersecurity challenges to their applications and software. Malicious threats, for example, are more present than ever in the typical software development lifecycle (SDLC) environment. This includes vulnerabilities that arise through open-source code, outsourced software development, supply chain complexity, compromised insiders, and other potential sources of malicious activity.

INTRODUCTION

Modern development teams must deal with a set of significant cybersecurity challenges to their applications and software. Malicious threats, for example, are more present than ever in the typical software development lifecycle (SDLC) environment. This includes vulnerabilities that arise through open-source code, outsourced software development, supply chain complexity, compromised insiders, and other potential sources of malicious activity.

The team from Toronto-based vendor *Security Compass* has worked with software leaders since 2004 to help them meet security and compliance requirements. This is done via their work in enabling teams to develop software that is secure by design, which is achieved with role-based, relevant Application Security Training and the use of a secure coding and developer-centric threat modeling platform called SD Elements.

A vital aspect of the work done at Security Compass has been to focus on the more preventive aspects of SDLC-based security in line with a secure software development philosophy called *Security by Design*, which ensures systems are built securely from the beginning of the development process, minimizing the attack surface at the planning and design phase, even before coding begins.

This framework involves attention to culture, principles, practice, and infrastructure – and our analyst team at TAG Infosphere sees this as an effective means for risk avoidance during DevOps and a path to DevSecOps. It focuses on earlier detection, vulnerability management, and countermeasures as opposed to relying on Find and Fix practices that can involve costly rework.

GETTING TO SECURE CODE

To achieve the goal of secure code, the development team must first accept the goal of building code that is *secure-by-default*. This objective involves products being secure to use out-of-the-box with little or no configuration changes necessary and security features available without additional costs. Releasing products with vulnerabilities puts customer data at risk and drives security activities toward responding to issues versus preventing them in the first place.

To achieve this security-by-default target state, software organizations are advised to adopt the Security by Design approach. This will drive developers toward embedding product security tasks throughout the software development lifecycle (SDLC). Implementing Security by Design is not a one-size-fits-all solution, but the tenets of the approach apply generally across virtually all software projects.

HOW THIS REPORT WAS DEVELOPED

This report emerged based on an analysis being done at TAG regarding best practices in shifting the software development process *to the left in the SDLC*. Such a shift is consistent with the objective of preventing vulnerabilities and is also highly consistent with shifts to more rapid coding using tools such as artificial intelligence (AI). Especially if code is to be created quickly, then it most certainly needs to be correct and secure.

We thus reviewed technical materials from Security Compass, which we believe to be the most prominent advocate for designing code securely. When we became aware of their emphasis on Security by Design (in fact, organizing their organization and solution set around the concept), we were excited to learn more. This report summarizes what we have learned, guided heavily by presentation materials offered to us by Rohit Sethi, Security Compass CEO.

As shown below, the Security Compass team makes the case that shifting left during coding toward more secure software is entirely feasible. This approach to software is important for any stakeholder developing or depending on software to support a critical mission. In such cases, the all-too-common approach of just waiting to be hacked so that detection and response tasks can be performed is simply mismatched with the consequences of an attack.

GETTING STARTED: A PEN TESTING CASE STUDY

We learned during our solution analysis of Security Compass that their team has had considerable experience supporting ethical hacking, penetration testing, and source code reviews. To illustrate, they shared with us the story of a specific engagement with a financial services customer where the security team's goal was to run tests and reviews to secure backend software.

Before the engagement had even started, the Security Compass test expert was able to explain and demonstrate confidently that he could break the customer's software easily. The rest of the test and review team was somewhat skeptical because their expert *literally hadn't looked at any code*. The individual was instead drawing the conclusion from experience with similar situations.

But sure enough, it was soon discovered that the testing team could perform arbitrary code execution on the target software. As any cybersecurity expert will attest, the ability to perform such execution at the discretion of the malicious attacker is a nightmare from a security perspective. It is how attackers achieve the goal of remotely hijacking or compromising software.

The details of the vulnerability exploited involved the software system using its own proprietary programming language. The Security Compass test team could thus inject code to run on the server. Since this software was responsible for the back-office accounting, it was realized that financial statements could be changed, checks could be printed, and other unauthorized transactions could be performed.

IMPLICATIONS OF THE EXPLOIT

While this type of story is obviously common to anyone working in application security or software development, it underscores that the root cause here was the way the software was engineered. There was no misconfiguration or sloppy administration. Instead, the software had been designed wrong, which implies that the design process needed to be improved. Things needed to "shift left" so to speak.

The problem, however – and this was true in the penetration testing example mentioned above as well as many other use-cases we've examined at TAG over the years, is that software teams become comfortable with and *used-to* their design process, and as a result, they end up using compensating controls, usually involving monitoring, for a software security problem that could have been avoided in the first place.

ANOTHER VULNERABILITY EXAMPLE

In 2021, U.S. Cyber Command issued an alarming public notification of potential exploitation of commonly used software called Atlassian Confluence that required immediate patching. This is not a typical occurrence for the US Cyber Command to issue such a warning. Luckily, most security administrators around the world recognized the threat and dropped everything to begin patching their systems.

Many of the readers here will recognize the Common Vulnerability Exposures (CVE) process and database as central to this notification and response approach. In the case of Atlassian Confluence, the reported issue concerned the web application and the different libraries and tools used in such an environment. It is promising that the security community has improved sharing methods, but this is certainly not nearly enough.

In the case cited above, the defect involved something known as an OGNL (Object Graph Navigation Language) injection vulnerability. What was interesting is that this report followed a common pattern that we've seen repeat over years and years. The OGNL injection problem was discovered in 2007 and was a registered CVE. Readers should keep this in mind as they develop programs to utilize notifications.

And again, with the case sited above, nearly fourteen years after notification, a known CVE was being exploited by attackers in the wild. This begs the question of why this or any other class of vulnerabilities known for so many years could continue to exist in the wild. And this is not just true for OGNL injection. This is also true for SQL injection. And for parameter manipulation. And for many other known problems.

WHAT IS THE ROOT CAUSE?

So why does this situation happen? In an ideal world, we would expect to identify the potential software weaknesses for a given product, and then these would be avoided through the actions of security teams, developers, and administrators. If you're building a web application, for example, you would implement proper controls from the start, ensuring that no weaknesses exist, or they're mitigated. You would have validated the existence of such controls.

In the example above, there was no good organizational process in place to mitigate OGNL injection because there had been no systematic means in the software process to address when there would be potential security weaknesses. Had such a process been in place, then this would have offered a means to implement a suitable control to mitigate the known vulnerability being exploited.

The good news is that the current state of software best practice is not completely devoid of cybersecurity. One often finds DevSecOps diagrams that show an overlay of static analysis, dynamic analysis, and penetration testing onto the standard processes included in modern software development. This might include commercial or open-source scanning tools, for example. This is a promising start.

The problem with this overlay method, however, is that very little happens systematically in the planning, design, and development phases to address security until the code is written, compiled, and then used to search for vulnerabilities through static or dynamic analysis. That is largely where the state of the industry is – namely, dealing with software security problems after they are introduced into code.

IMPROVING SOFTWARE PROCESS SECURITY

So, as we've suggested, the current development lifecycle process for software security involves running a series of tests to find vulnerabilities in code. It does not, however, typically involve taking preventive steps to avoid these problems from being introduced to the software in the first place. As one might expect, there are quite a few issues and challenges that result from this more reactive approach.

First, there's an unnecessary cyber risk the results from allowing vulnerabilities to enter the code and then trying to find them after the fact. On average, such detections can take hundreds of days (or more). Recent empirical studies from organizations such as NTT Security and White Hat Security have suggested that 149 days is a typical average to find and fix a critical software risk issue.

Second, fixing discovered software vulnerabilities often isn't trivial or even straightforward. Development teams have many priorities, so critical vulnerabilities deployed to production environments can also take roughly 149 days to fix, during which time they are exploitable. Perhaps there are compensating controls that mitigate the impact. But this is not the most desirable situation.

A third issue is that there are not enough application security experts to support all the developers in most organizations. The ratio, in the experience shared with us by the team at Security Compass, is one to a hundred. And the case could be made in many companies that the scale is more like one to five hundred or even a thousand. So, there is not enough expertise, and it's tough (and expensive) to hire this kind of expertise.

Finally, it's worth pointing out the rising cost of rework after discovering security flaws. That is, in general, the later one finds an issue in the software process, the harder it is to fix. This is because there are just more remedial steps that must be taken, and there's more impact to the change, which explains that there are 149 days on average to fix a vulnerability. Again, the goal should be to prevent problems from occurring in the first place.

GETTING STARTED WITH SECURITY BY DESIGN

We hear many security experts beginning to talk today about the desirable process of *shifting left*. What they mean is the use of test methods such as static analysis earlier in the process. For example, a programmer writes code, scanners are run early in the process, vulnerabilities are found early, and they get fixed. This is certainly better than finding these problems later in the process.

But readers should recognize that this security process could shift even further left. It could start in the design process, then the vulnerability will never have occurred in the first place, and that is by far the cheapest and most proactive way to address it. And this has significant implications because the cost of breaches is increasing. Regulatory changes are one driver of this increase.

In Canada, for example, Security Compass shared with us the details of a new bill being put in place to levy fines for breaches. In Europe, as you probably know, these fines are reaching record levels for security and privacy breaches as defined by the EU Cyber Resiliency Act. There are many other new laws around the world that are being set up to address risk by increasing fines to companies who do not avoid vulnerabilities.

This is a tough situation, however, because most chief information security officers (CISOs) are now largely saying that security breaches are inevitable and that they cannot be prevented. This is hard to argue, given all the breaches that occur, but the truth is that breaches are viewed as inevitable because there is so much vulnerable software. We try to reinforce this message with our TAG customers every day.

Let's suppose you have built software, and there's a breach because of a vulnerability in that software. The correct question is to ask what you did – *or what you might have done* – to secure that software. This is really where industry best practices become important. Today, this best practice is to generally ignore security during the design phase and simply show that you've tested for it.

This approach might help from the perspective of legislation and even avoidance of fines from a legal risk perspective. But the much better argument is that you should be integrating **security by design**. You should be able to prove that you've been doing it so that you're not just relying on validation to test for security issues. Our observation is that the team from Security Compass is leading the way in this regard.

ABOUT TAG

TAG is a trusted research and advisory company that provides insights and recommendations in cybersecurity, artificial intelligence, and climate science to thousands of commercial solution providers and Fortune 500 enterprises. Founded in 2016 and headquartered in New York City, TAG bucks the trend of pay-for-play research by offering unbiased and in-depth guidance, market analysis, project consulting, and personalized content—all from a practitioner perspective.

IMPORTANT INFORMATION ABOUT THIS DOCUMENT

Contributors: Edward Amoroso

Publisher: TAG Infosphere Inc., 45 Broadway, Suite 1250, New York, NY 10006.

Inquiries: Please contact Lester Goodman at lgoodman@tag-cyber.com to discuss this report. You will receive a prompt response.

Citations: Accredited press and analysts may cite this book in context, including the author's name, author's title, and "TAG Infosphere, Inc.." Non-press and non-analysts require TAG's prior written permission for citations.

Disclaimer: This book is for informational purposes only and may contain technical inaccuracies, omissions, and/or typographical errors. The opinions of TAG's analysts are subject to change without notice and should not be construed as statements of fact. TAG Infosphere, Inc. disclaims all warranties regarding accuracy, completeness, or adequacy and shall not be liable for errors, omissions, or inadequacies.

Disclosures: Security Compass commissioned this book. TAG Infosphere, Inc. provides research, analysis, and advisory services to several cybersecurity firms that may be noted in this paper. No employees at the firm hold any equity positions with the cited companies.

TAG's forecasts and forward-looking statements serve as directional indicators, not precise predictions of future events. Please exercise caution when considering these statements, as they are subject to risks and uncertainties that can affect actual results. Opinions in this book represent our current judgment on the document's publication date only. We have no obligation to revise or publicly update the document in response to new information or future events.

Copyright © 2024 TAG Infosphere, Inc. This report may not be reproduced, distributed, or shared without TAG Infosphere, Inc.'s written permission.