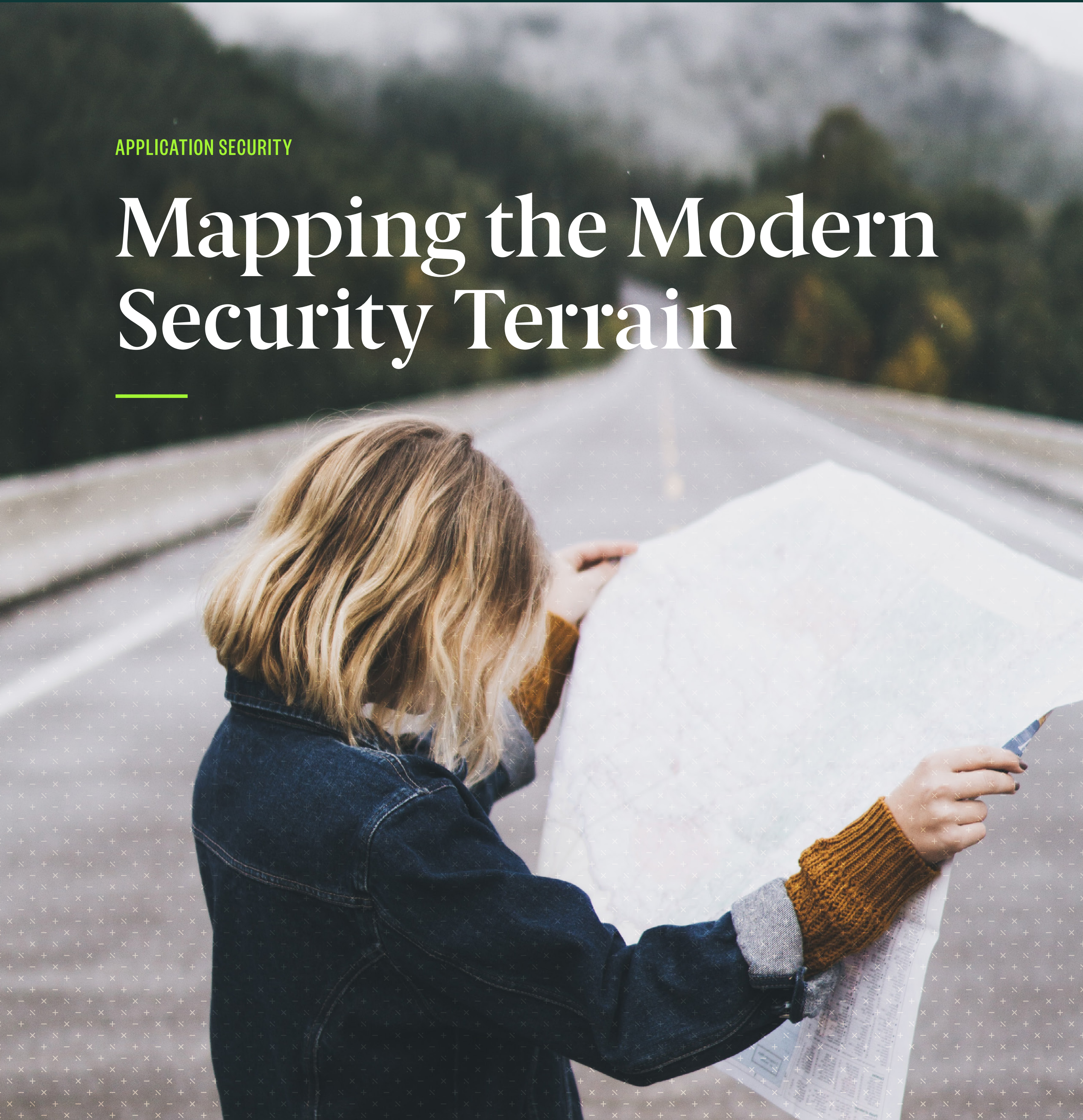
MOSSADAMS



APPLICATION SECURITY

Mapping the Modern Security Terrain

CONTENTS

INTRODUCTION

IDENTIFYING AND ADDRESSING VULNERABILITIES

Programming languages, operating systems, and network protocols were never designed to be inherently secure. After their initial specification by the Department of Defense, the internet protocols we've come to take for granted were developed in the relatively benign and permissive environment of research institutions and university collaboration during the 1970s. Widespread realization of security issues lagged significantly behind public and commercial adoption of the network in the 1990s.

Once the network-level security problem was recognized, perimeter security appliances such as firewalls and intrusion detection devices were brought into service. By the turn of the new century, network level security had become a well understood problem.

With the advent of web and other internet-enabled applications, threat actors realized the application layer had very little security, and could be used to effectively bypass network layer protections. Internet applications became the most common target for criminals attempting to obtain proprietary information, restricted customer data, and access to protected assets and accounts.

Now in 2019, it's necessary to secure not only traditional web applications, but also web application programming interfaces, mobile devices, microservice architectures, and internet-enabled appliances.

Selecting the right combination of application security risk management solutions requires taking business requirements and unique factors into account. There's no single approach that will fit every organization's needs.

Application Security Programs

Historically, software development was a slow and methodical process. All functional requirements were specified up front and frozen by the time implementation began. While this waterfall approach had many problems, it had the distinct advantage of allowing ample time for careful security evaluation prior to placing an application in production.

Modern development life cycles are agile; their development happens in short sprints. In recent years, pace has accelerated even further with the widespread adoption of DevOps where software development and operations are integrated. This coupling relies heavily on automation and having operations personnel on the same team as the developers to achieve a rapid deployment life cycle. In some cases, there can be multiple live application deployments in a single day.

In this fast-paced application development and deployment environment, it can be challenging to implement an effective program that's compatible with the software development life cycle (SDLC). There's little room for traditional security evaluations, relying instead on trained developers who avoid introducing security flaws in the first place, along with lightweight tools to give them instant feedback if they inadvertently create one.

Reasonable Safety

While absolute security is the goal, it's usually necessary to make tradeoffs to harmonize with target deployment schedules and business goals. If there isn't an adequate security program already in place, making these tradeoffs requires a careful analysis and assessment of the current baseline.

An application that handles banking transactions or deals with a customer's personally identifiable information requires a higher level of scrutiny and lower level of risk tolerance than a blog. However, even a blogging site can interact with an organization's other applications in unforeseen and damaging ways.

EXAMPLE

For example, if the database hosting the blog is compromised and an attacker is able to execute certain procedures, they may be able to obtain a command shell on the internal network, facilitating an assault on a sensitive financial database or application hosted elsewhere on the infrastructure. Alternately, a cross-site scripting vulnerability on that same blog could be used to exploit a user's trust in the organization's domain, leading them to give up credentials that could compromise their accounts on the financial site. A holistic view of the organization is required to establish a true baseline.

A reasonable level of safety varies depending on the context of a specific application and organization. The definition may be based upon government, such as defense classification levels, industry-group or business-domain requirements, regulatory requirements, and sometimes just plain old common sense.

Identifying Risks

A useful resource for understanding both web application and mobile application security risks is the **Open Web Application Security Project (OWASP)**. With hundreds of articles defining common application security flaws and offering useful remediation advice, OWASP also publishes an annual list of the top 10 most critical web and mobile application vulnerabilities.

While these lists are very useful for increased security awareness, they aren't intended to be comprehensive. The OWASP lists could be considered a prototype threat model for many applications and make a great starting point, but customization will usually pay dividends.

Custom Threat Modeling

Each application has specific, unique security concerns, and a generic list won't always work. While the OWASP list is a useful starting point, threat modeling—a technique used to identify specific threats an application must be ready to face—can help identify the specific threats that pertain to your application, and eliminate those that don't apply.

There are a variety of tools on the market that can assist with the development of a custom threat model. Some are paid commercial products offering a wide variety of useful features and reporting capabilities, but there are also free, proprietary, and open-source threat modeling tools available.

Establishing a Baseline

Developing a threat model could be an adequate first step for a start-up, because the model can illuminate which security threats require immediate attention and which can be safely ignored. However, an established enterprise will probably require a deeper assessment of their up and running applications to establish a starting point, even if they intend to move to more automated and lightweight methods going forward.

This deep assessment should include the use of a commercial-grade source code static analysis tool and an element of manual analysis by a trained engineer, a process often referred to as a business logic assessment (BLA).

YOUR OPTIONS

Types of Assessments

There are a number of approaches to testing application security that involve a combination of automated and manual analysis.

Some are strictly external tests, sometimes referred to as black box, because the evaluator has no insight into the application's internal architecture, configuration, or source code. Other types of tests are internal, or white box, and important information is made available to the tester. Often, these techniques are combined, sometimes called grey box testing.

Frequently Missed Vulnerabilities

Strictly automated testing is faster and less expensive, but there are some important vulnerabilities that an automated evaluation struggles to identify and could miss.

- Sensitive data that isn't being encrypted, such as hardcoded passwords
- Third-party services operating without proper protection
- Flaws in the entitlement check mechanism that may allow access to a user's data by another unauthorized user
- Authentication logic flaws
- Authorization logic flaws
- Disclosure of confidential data
- Inadequate audit logging
- Susceptibility to cross-site request forgery
- Presence of application back doors

This is why augmenting the automation tools with manual inspection, such as a BLA, is an important step.

On the other hand, automated tools are adept at detecting some flaws, at least for most common application architectures.

These could include the following:

- Missing entries in an xml configuration file
- Dangerous functions, including unvalidated user input data in webpage output, also known as a cross-site scripting vulnerability
- Unvalidated input data in the construction of a database query, also known as a structured query language (SQL) injection

Dynamic Application Security Testing

Dynamic application security testing (DAST) is a black box scanning method that interacts with a running application and essentially treats it as a black box to identify points of vulnerability.

VALUE

The application is attacked under realistic conditions, so the identified vulnerabilities are concrete and compelling. If the scanning tool can present evidence of an exploitable cross-site scripting vulnerability, it's difficult to claim that a real attacker couldn't do the same.

DOWNSIDES

While the value of this approach lies in its realism, it probably won't identify every instance of each vulnerability. It also requires a knowledgeable engineer to filter out the many false positives these tools produce.

Static Analysis Security Testing

Static analysis security testing (SAST) scans application code for vulnerabilities, usually high-level application source code. It's considered a white box assessment, because nothing is hidden from the tool. Application code is a larger and richer analysis target than the user interface addressed by DAST scanning, which means a broader range of vulnerabilities can be identified.

The best static analysis tools utilize sophisticated compiler technologies, such as data flow analysis, control flow analysis, and pattern recognition to identify security vulnerabilities. The results of automated analysis still include a high number of false positives, requiring a highly skilled security engineer to analyze the results to distinguish between the true and the falsely reported vulnerabilities.

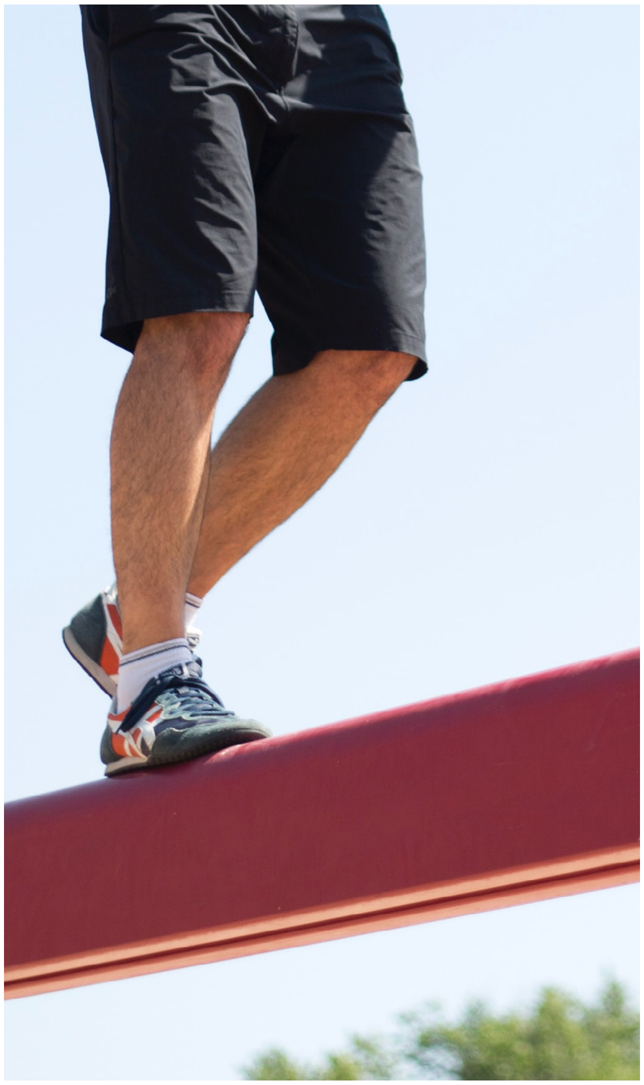
Most static analysis tools can also identify a range of poor programming practices, such as the use of uninitialized variables or the lack of error handling. While some of these examples may be found by external black box scanning, SAST scanning has a higher probability of detecting them and avoiding false negative findings.

VALUE

The main strength of SAST is analyzers identify potential issues in the face of highly complex application structures and data flows that would daunt most humans.

DOWNSIDES

With the current state of the technology, SAST isn't generally capable of testing algorithms, security policy adherence, and specialized issues that may be derived from the application's domain and business requirements. The main limitation of either type of automated tool is they can only find approximately 30% of security vulnerabilities that should be evaluated in a comprehensive security assessment.



Penetration Testing

Penetration testing goes a step beyond the external web application black box scanning described above.

A penetration tester, or pentester, is a talented security engineer with a deep knowledge of internet and web protocols. In many cases, they also have significant coding experience. This knowledge allows them to test both traditional web and mobile applications, which almost always use web protocols to communicate back to a server.

To find weaknesses in the application that the automated tools didn't reveal, a pentester has multiple options to utilize:

- Specialized commercial and open-source tools
- Imagination and expertise

Penetration testing can happen in one of two ways. One involves going deep, attempting to get as far into the application and supporting infrastructure as possible in the time available, which is what most people think of when they hear the term. The other version is more like an enhanced version of the vulnerability assessments. Using DAST along with manual tools and techniques, the pentester attempts to demonstrate as many vulnerable entry points as possible in the time allotted, rather than deeply exploiting just one or two of them.

A deep penetration test can be very revealing for the application's owners, even if it doesn't cover the full breadth of possible entry points.

For example, the results of a successful SQL injection attack might include data or metadata accessed and exfiltrated without authorization. If the pentester can successfully extract a table of users and account information from the database and present it to the security team or systems administrators—despite the fact that the pentester was never provided database access—it will be painfully clear that the organization needs to take steps to improve their application security program.

Security Code Review

Security code review involves an assessment of application architecture and source code by highly skilled software security engineers.

This typically involves the use of a static analysis security testing (SAST) automated scanning tool to supplement a manual analysis, but as some languages aren't supported by these tools, this could be a 100% manual effort. As previously discussed, the portion of this type of review that goes above and beyond the capabilities of the SAST tool may be considered a BLA.

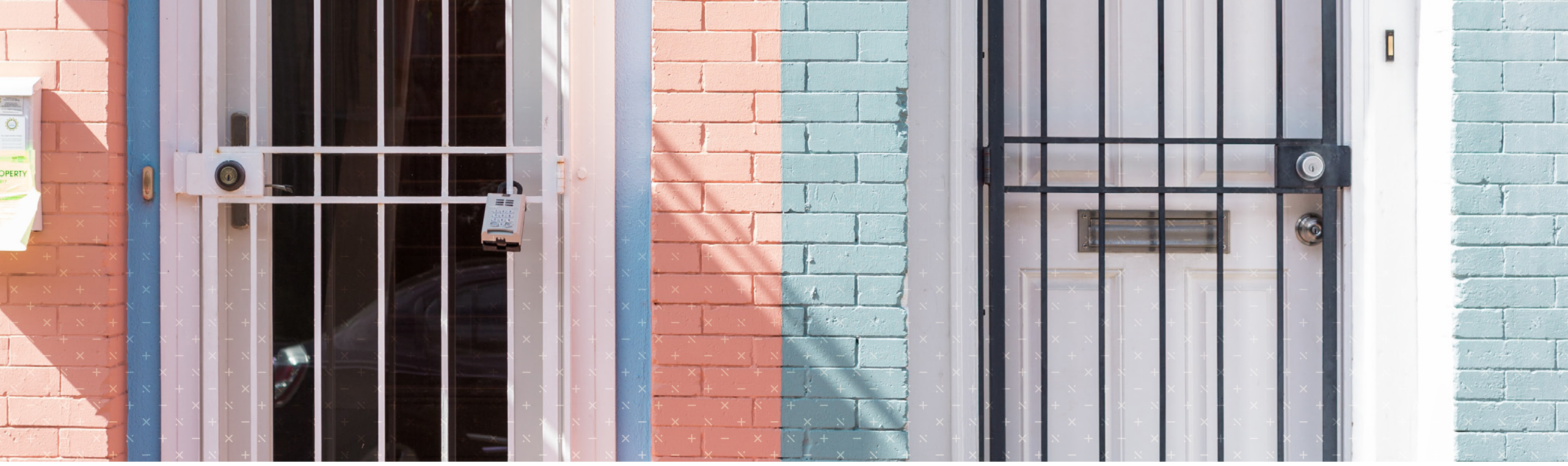
A security code review could be considered a combination of SAST and BLA. The resulting analysis is the most reliable and comprehensive of the approaches, making it the gold standard in industries where application security is a crucial concern, such as financial services.

VALUE

The strength of a security code review is in the depth and thoroughness of the assessment. The full range of security vulnerabilities can be identified, and most if not all of the instances of these vulnerabilities can be enumerated.

DOWNSIDES

The main drawback of this type of analysis is engineers with the necessary skills and experience—both extensive enterprise application development experience and deep security knowledge—are scarce and in high demand. The time and level of effort involved makes this approach costlier than other options.



Hybrid Approaches

Sometimes it's helpful to combine techniques and use a hybrid, or grey box, approach. These combine the best of both worlds by using realistic external attacks combined with the visibility of a white box audit.

EXAMPLE

One hybrid approach that has proven successful is performing a security code review alongside a running, testable version of the application. The engineer conducting the review deploys both DAST and SAST analysis tools, along with manual testing, source inspection, and a BLA.

This creates a natural synergy, because suspicious patterns spotted in the code can be rapidly assessed for vulnerabilities through actual testing. Conversely, any problematic responses observed in the DAST scanning or manual testing can be quickly confirmed or disconfirmed as a security issue through examination of the relevant code.

EXAMPLE

Another hybrid approach uses interactive application security testing (IAST) tools. These tools require creating an instrumented version of an application through the inclusion of various runtime libraries provided by the IAST tool vendor. The application is then scanned externally. When vulnerabilities are found, the included libraries allow the tester to pin them down in the source code.

The IAST approach offers a distinct advantage by combining automation with a reduced number of false positive findings. This makes it a strong contender for inclusion in the rapidly evolving, agile environment central to DevOps.

NEXT STEPS

Software Composition Analysis

Modern applications are seldom built completely from scratch. As applications become more interactive and complex, developers increasingly rely on open-source or third-party libraries to implement common functionality.

Well over half of a typical application is composed of external code. Unfortunately, including external code means also including any vulnerabilities within it. For example, in 2017 Equifax utilized the very common Apache Struts open-source library, which provides a framework for Java web applications. Because the version Equifax used included vulnerabilities, the company was hit by a very large and significant application breech that included data exposure.

Attackers are well aware that open-source code, and even third-party proprietary code represents a potentially successful attack vector. They spend considerable time and effort searching these components for flaws, often using the same tools and techniques described in this paper.

Third-party libraries can also be problematic because they can put an organization into legal jeopardy if used incorrectly. Even open-source software has license agreements which must be followed, and sometimes these can be quite stringent in their terms.

Software composition analysis (SCA) determines what external libraries are in use by an application, and if there are any publically known vulnerabilities that affect them.

A secure code review process will usually identify and report on these issues, but doing it manually can be tedious and time consuming. Several commercial tools exist for this purpose, and they will let the developers or security team know if there's a vulnerability warning for the library in question. They'll also identify if it's exploitable in the specific usage context of an application.

SCA isn't something that can be done once and forgotten, because new library issues are constantly discovered and reported. An organization must make a continuous effort to keep up with the status of their third-party libraries.

Runtime Protection

If security issues are missed during the development and testing process, all isn't necessarily lost. Modern tools make it possible to interactively detect and stop attacks against a running application in production.

One way to do this is through a web application firewall (WAF). A sophisticated WAF can be configured to not only match and block likely attack patterns, but also respond to new attack signatures through the use of advanced heuristics and, increasingly, artificial intelligence.

Despite their namesake, these devices aren't as straightforward as a traditional network level perimeter firewall, because securing the application layer isn't as simple as blocking or allowing access to certain ports and services. Web applications and application program interfaces (APIs) are varied and complex, and WAFs can't stop certain kinds of attacks, particularly those resistant to automated discovery due to logical flaws.

Attackers are aware of various WAF products, and are skilled at identifying which product is in use so they can attempt to evade them. The use of a WAF significantly raises the bar for the intruder, but it shouldn't be used as a crutch to avoid securing the applications themselves. Eventually, an attacker will get past the WAF and the approach will fail.

Another possible drawback to a WAF is it often incorrectly senses legitimate user interaction patterns as attacks. Unlike false positives produced by SAST and DAST scanning tools, the WAF can act on this erroneous information in the real world and cause an organization to effectively launch a denial of service attack against itself.

WAF products have passive learning and reporting modes in which they learn normal usage patterns. These modes allow their configurations to be tweaked, but they're not really doing their jobs when deployed in this mode, which is another reason the application itself must be secure.

A recent innovation on the WAF concept, known as runtime application self-protection (RASP), includes instrumentation libraries with the deployed application, a concept discussed earlier in the context of IAST.

Unlike IAST but similar to a WAF, RASP will respond to any attacks detected by the runtime libraries and attempt to actively block the incoming attack. The use of this onboard instrumentation makes attack detection more accurate, reducing false positives and hopefully overcoming the inadvertent denial of service that may occur with a traditional WAF. They're also more difficult for an attacker to evade.

Developer Training and Environment

Adequate developer training can do more for an organization's application security posture than all of the other tools and testing combined. Ideally, developers should receive secure coding training, customized with an accurate threat model, at the start of a project. This training must be periodically updated and repeated to keep it fresh in their minds. The best way to secure an application is to not introduce security flaws in the first place.

Even with good training, programmers aren't infallible. That's why it's important that they get rapid feedback to fix any flaws almost immediately upon introduction, when costs are lowest. A full static analysis scan combined with a BLA—a security code review—may be the gold standard for application security and the best way to establish a baseline, but something lighter and quicker is desirable for real time flaw detecting during the coding process.

Fortunately, such products are available both from security vendors and open-source, and they integrate well with popular integrated development environments (IDEs) to allow nearly seamless integration into the toolchain.

CONCLUSION

We're Here to Help

Each organization has unique application security needs. For more information on charting your organization's application security needs, and how we can help protect you and your stakeholders, contact your Moss Adams professional.

mossadams.com/applicationsecurity

About Moss Adams

With more than 3,200 professionals across 25-plus locations in the West and beyond, Moss Adams provides the world’s most innovative companies with specialized accounting, consulting, and wealth management services to help them embrace emerging opportunity. Discover how Moss Adams is bringing more West to business.