WHITEPAPER

The Business Value of **Secure Software**

Building secure software makes business sense. Improved software security can improve revenue growth (as a competitive advantage), raise margins (through lower maintenance expense), improve customer satisfaction (fewer security patches and updates), and simplify regulatory compliance. As hackers increasingly target the application layer, organizations need to respond appropriately.

HACKERS TARGET THE APPLICATION LAYER

Software security has become a board level issue. One need only look at the fallout from the 2017 Equifax breach to see why. The company's CEO, CIO, and CSO were all forced to resign, and the company faces up to $700 million in settlement costs.

In years past, adversaries focused on hacking networks. Now, applications are the target. The reason is simple. Software organizations continue to focus on features and functionality. Errors in the design and execution of software can result in vulnerabilities that are easy to access and simple to exploit using attacks such as SQL injection and cross-site scripting.

Unfortunately, many organizations continue to view security as it was many years ago; a challenge of perimeter defense. As shown in Figure 1, security spending by organizations continues to focus on the network layer, while risk is highest in the application layer. Focusing on perimeter defenses ignores the fact that attacks on web applications are the most common cause of data breaches[1].

Today, the application is the perimeter. Web applications manage critical information and IP. There is no need to attack network firewalls when the data is available through a web application.

[1] 2018 Verizon Data Breach Investigations Report

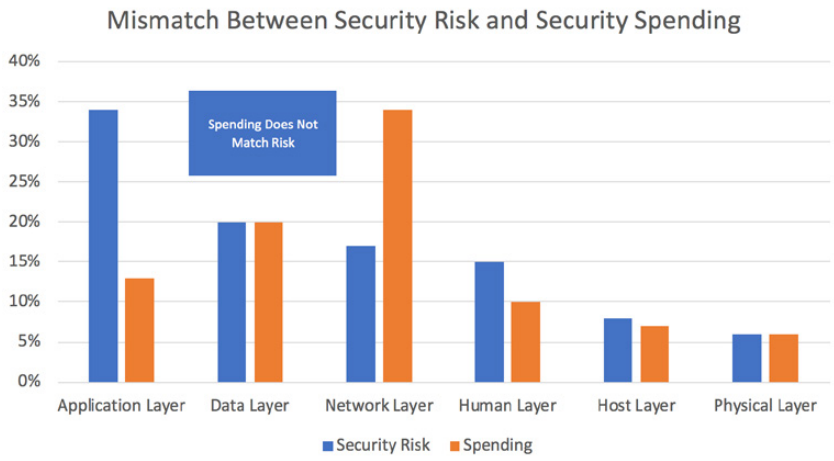

Figure 1:
Security Risk v. Spending
Source: The State of Risk-based Security Management, Ponemon Institute, 2013

HACKING APPLICATIONS IS BIG BUSINESS

Security professionals are not simply defending against script kiddies and amateurs. Adversaries are more skilled than ever, criminal organizations are well-funded and the market demand for financial data, health information, and consumers' personal information is high.

While consumer data is often sought for identity theft, industrial espionage is an ongoing concern. Most concerning for organizations with valuable intellectual property (IP) are attacks from nation states seeing to steal design information and trade secrets. State-sponsored attacks and organized crime groups are real and make for interesting headlines. The WannaCry ransomware attack in 2017 that infected over 300,000 devices was attributed to North Korea, and according to Europol's Internet Organised Crime Threat Assessment, Ransomware remains the top cybercrime threat.

"Five years ago, we were aware of nation-state attacks but we would've seen them as something that only a nation-state needs to worry about. Today they're a problem for everybody..."

Robert Hannigan
Former director general of GCHQ.

SOFTWARE SECURITY AND REGULATORY STANDARDS

The Equifax penalties are representative of a growing trend. Privacy regulations include substantial financial penalties for non-compliance. The EU General Data Protection (GDPR) lower level fines are up to €10 million or 2% of the worldwide annual revenue of the prior financial year. Upper level fines double those amounts! In 2019, British Airways was fined $230 million for violations of GDPR after data of around 500,000 British Airways customers was compromised. Meanwhile, Marriott paid over $120 million after it exposed personal data from 339 million customers, including credit card details, passport numbers and dates of birth.

Complying with the myriad of regulatory standards can be challenging for software development teams. Some are very prescriptive. For example, the PCI-DSS for software processing credit card information and the UL-2900 standard recently adopted by the Federal Drug Administration (FDA) for network-connected medical devices, require that organizations test for specific types of vulnerabilities such as those enumerated in the CWE Top 25 and CWE On the Cusp weaknesses, and the OWASP Top 10.

HIPAA is less prescriptive, requiring instead that covered entities "Conduct an accurate and thorough assessment of the potential risks and vulnerabilities to the confidentiality, integrity, and availability of electronically protected health information held by the covered entity" and "Implement security measures sufficient to reduce risks and vulnerabilities to a reasonable and appropriate level to comply with § 164.306(a)." Others provide no guidance at all. Section 5 of the FTC Act simply requires "reasonable security", California's SB-327 targeting Internet of things (IoT) devices requires manufacturers of any connected device sold in California to have "reasonable security features", and GDPR requires both "Privacy by Design" and "Privacy by Default".

While differences exist between the various standards, the underlying requirements are the same; organizations need to have visibility to the risks they face and a plan for addressing those risks.

RISK APPETITE AND RESIDUAL RISK

The goal of security testing is two-fold. First, to provide visibility into the risk resulting from coding errors that could be exploited by an adversary, and prevent those errors from entering the codebase by enforcing best practices.

Secondly, it's important to remember that, except in the most critical applications, eliminating risk entirely is unlikely to be the goal. Unremediated issues will result in residual risk and that may be acceptable. Different applications present different levels of risk, and a mature organization will determine their appetite for risk and make informed decisions about the amount of residual risk with which they are comfortable.

THE SECURITY TESTING TOOLBOX

There are several techniques for identifying vulnerabilities in systems, and smart organizations will use a combination of each of them, including static analysis, dynamic analysis, source composition analysis, vulnerability scanners, and penetration testing. As shown in Figure 2, because code refactoring becomes more complicated as the application nears release, the cost of remediating vulnerabilities increases dramatically as the software development lifecycle (SDLC) progresses. The goal of security testing, therefore, should be to "shift left" in the SDLC the identification and remediation of vulnerabilities as early as possible.

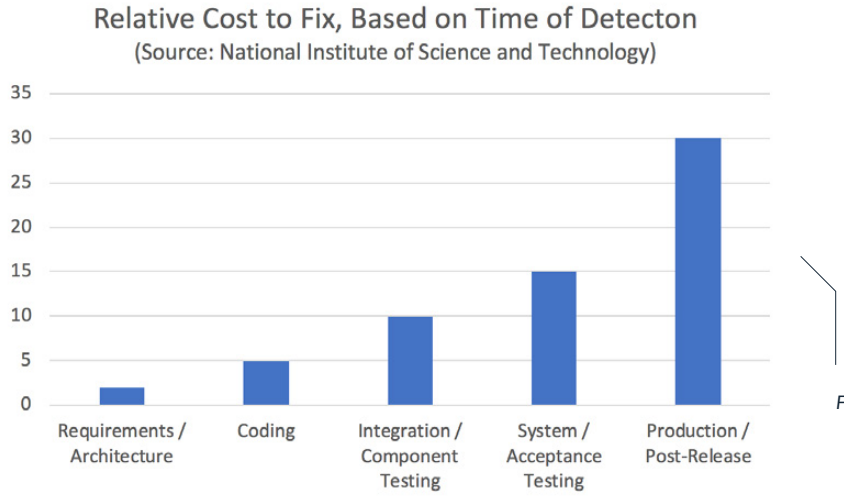

Figure 2

STATIC ANALYSIS

Static Application Security Testing (SAST) tools do not require a running application and therefore can be used early in the development lifecycle where remediation costs are low. At its most basic level, SAST works by analyzing source code and checking sets of rules against it. While most often associated with identifying vulnerabilities, SAST tools also provide early alerts to developers regarding poor coding patterns, violations of secure coding policies, or a lack of conformance with engineering standards that will lead to unstable or unreliable functionality.

There are two primary types of analysis used for identifying security issues.

Flow Analysis

In flow analysis, the tools analyze source code to understand the underlying control flow and data flow of the code. The result is an intermediate representation, or model, of the application. The tools run rules—or checkers—against that model to identify coding errors that result in security vulnerabilities. For example, in a C or C++ application, a rule may identify string copies, then traverse the model to determine if it is ever possible for the source buffer to be larger than the destination buffer. If so, a buffer overflow vulnerability could result.

Pattern Analysis

Modern software engineering standards like AUTOSAR C++14, MISRA C 2012 and Joint Strike Fighter (JSF) are based on the idea that certain constructs should be avoided in code that is safety critical, because of the possibility for that code to be misinterpreted, misunderstood or incorrectly implemented and therefore be unreliable. Pattern analysis helps developers

use a safer subset of the development language given the context of safety or security, prohibiting the use of code constructs that allow vulnerabilities to occur. Some rules can identify errors by checking syntax, similar to a spell-checker in a word processor. Others can detect more subtle patterns associated with poor coding patterns.

DYNAMIC ANALYSIS

Dynamic application security testing (DAST) tools analyze running applications to identify vulnerabilities. Commercial DAST tools are typically automated. They work by identifying inputs to an application (e.g., a login or order form) and applying various preconfigured data to attempt to cause the application to misperform or crash (i.e. "fuzzing data"). A common example would be to enter a SQL command into a form (SQL injection attack) to bypass authentication or access sensitive information.

Results from DAST tools do not link to a line of code. Instead, vulnerabilities are reported as a URL/action/result (e.g., "on https://app.mycompany.com/order, for the customer number I entered "99 ' OR 1=1" and received output of all customer names"). The software engineering team must determine where the error in the source code occurs. This is complicated when the error is from unvalidated user input, as the error will manifest itself wherever that untrusted data is used.

PENETRATION TESTING

For most organizations, hiring external penetration testers ("pen testers") will be their initial foray into security testing. Instead of running automated DAST scans, penetration testers are trained to understand and identify common errors in software development that can lead to security vulnerabilities. Pen testers use a combination of commercial, open source and custom tools to conduct reconnaissance on the target system, identify potential entry points, and gain/maintain access to a system. Like DAST tools, penetration tests report vulnerabilities in a URL/Action/Results format.

While helpful, penetration testing can be expensive. In addition, because it requires a running system and data in a staging environment, it occurs very late in the development process.

SOURCE COMPOSITION ANALYSIS

Applications are generally comprised of custom code and third-party components; being most often open source components. Like any software, open source can include vulnerabilities, and thousands are disclosed each year in NIST's National Vulnerability Database (NVD). Source composition analysis tools parse the application's package manager or inspect component "fingerprints" to generate a software bill of materials (SBoM) then map known vulnerabilities from NVD or other sources to those components. Vulnerable components are flagged (or those with restrictive licenses), but it is important to note that the component may or may not be exploitable depending on how it was used and in what portion of the component the vulnerable code is located.

VULNERABILITY SCANNERS

Vulnerability scanners analyze running IT systems to identify unpatched or misconfigured applications or systems. Typically, they will have hundreds of "plug-ins" or "rule packs", each designed to identify a specific issue on a specific platform. Vulnerability scanners can flag out-of-date operating systems, the use of default passwords, and previously disclosed vulnerabilities in applications and components. These solutions focus on commercial software and operating systems but, other than for a small portion of vulnerabilities in open source components, are blind to any in-house applications.

ADVANTAGES OF SAST VS. DAST

While each testing methodology has strengths, many organizations overly focus on DAST and penetration testing. However, there are several advantages to using SAST over other testing techniques.

» **Code Coverage –** The amount of code that is tested is a critical metric for software security; vulnerabilities can be present in any section of the codebase, and untested portions can leave an application exposed to attacks. SAST tools, particularly those using pattern analysis rules, can provide much higher code coverage than DAST or manual processes, as they have access to the application source code and application inputs, including hidden ones that are not exposed in the user interface.

Figure 3

» **Root Cause Analysis –** SAST tools also promote efficient remediation of vulnerabilities. Unlike DAST, SAST easily identifies the precise line of code in which the error is introduced. Integrations with developers' IDE can also accelerate remediating errors found by SAST tools.

» **Skills Improvement –** The recent SANS Institute report, "Secure DevOps: Fact or Fiction?" shows *'Shortage of application security personnel/skills'* is the #1 barrier in implementing secure DevOps and, as shown in Figure 3, NIST estimates "the ratio of existing cybersecurity workers to the number of cybersecurity job openings is 2-to-3". When using SAST from the IDE, developers receive immediate feedback on their code, reinforcing and educating them on secure coding practices.

» **Operational Efficiency –** Unlike DAST, static analysis can be used very early in the development lifecycle, including on a single file directly from a developer's IDE. Finding errors early in the SDLC greatly reduces the cost of remediation because you are essentially preventing the bug, not finding then fixing it.

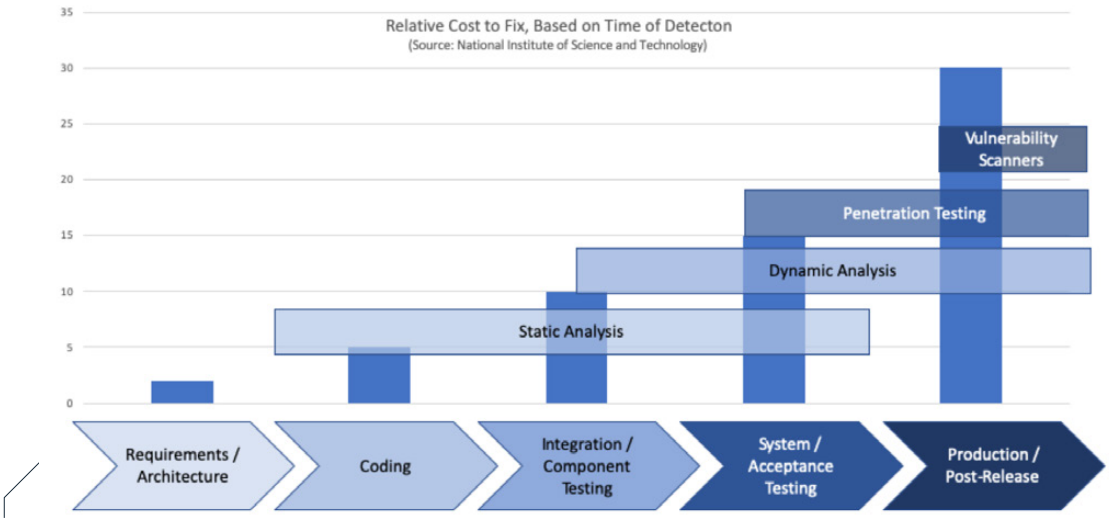

Figure 4

MISTAKES AND MISCONCEPTIONS WHEN DEPLOYING SAST

While SAST is the most comprehensive of the testing methodologies it can also present challenges to security teams.

- **Delaying Deployment of SAST –** While SAST tools can be used very early in the SDLC, some organizations elect to delay analysis until the testing phase of the lifecycle. While analyzing a more complete application allows for interprocedural data flow analysis, "shifting left" with SAST and analyzing code directly from the IDE can identify vulnerabilities such as input validation errors, and allow developers to make simple corrections before submitting code for builds.
- **Deferring Use in Agile Environments –** SAST analysis has a reputation for taking longer than DAST because of its comprehensive approach to code coverage and the need to build a model of the application. This can lead organizations to believe SAST is incompatible with rapid development methodologies. Instead, smart teams use SAST from within the IDE, providing immediate feedback to developers and ensuring that vulnerabilities are avoided, and perform incremental analysis to view results only from the code that has changed between two different builds.
- **Noisy Results –** Older SAST tools often included many "informational" results; low severity issues around proper coding standards. Modern tools, like those provide by Parasoft, allow users to select which rules are used and filter results by the severity of the error, hiding those that do not warrant investigation. Findings can be further filtered based on other contextual information such as metadata on the project, the age of the code, and the developer or team responsible for the code.

SELECTING AND USING SAST EFFECTIVELY

The first step in evaluating any tool is to understand your internal environment. This includes existing tools, skill sets, and workflows.

ON PREMISE OR IN THE CLOUD

Most software includes intellectual property valuable to the organization. If that source code were leaked or if vulnerabilities in proprietary software were publicly known, serious harm could come to the organization. For that reason, most organizations look for a solution that can be deployed within their own environment.

AGILE, CI/CD, OR WATERFALL

An organization's development methodology can influence which solutions they deploy. In a rapid development and deployment model, it is critical that analysis and feedback cycles are quick. In these cases, look for solutions that offer incremental analysis to provide quick feedback identifying vulnerabilities in newly modified code without having to rescan the entire codebase.

FOCUS ON DEVELOPERS

Successful deployments most often are developer-focused; providing them with the tools and guidance needed to build security into the software. This is particularly important in Agile and DevOps environments, where rapid feedback is critical to maintaining velocity.

This typically means a SAST solution that integrates with the developers' IDE(s). IDE integrations allow security testing directly from the developer's work environment—at the file level, project level, or simply to evaluate the code that has changed.

RULE CONFIGURATION

When analyzing software for security issues, one size does not fit all organizations. It is critical that the rules/checkers are addressing the specific issues critical to that specific application. For example, if an application is subject to UL-2900, you must ensure that the rules used cover the CWE Top 25 plus On the Cusp weaknesses. While many SAST tools claim Top 25 support, that support may be limited to isolated use cases.

Organizations just starting testing for security may wish to limit rules to the most common security issues like cross-site scripting and SQL injection. In addition to standard rules, organizations using custom frameworks or with custom coding standards will want to look for solutions that allow custom rules.

Most solutions classify vulnerabilities by severity in a fixed way that may not be appropriate for every application. If an application is not reachable on the Internet, its attack surface is greatly reduced and vulnerabilities that could result in denial of service attacks are going to be less critical than in an Internet-facing application. Look for solutions that allow controlled rule configuration.

REDUCE THE NOISE

In addition to minimizing informational issues, organizations may elect to downgrade or accept the risk from some vulnerabilities. In these cases, users will want to suppress an issue. Make sure the SAST solution offers this capability, that the suppressions persist across subsequent scans, and that all results are documented for compliance environments like UL-2900.

HOW PARASOFT CAN HELP

Parasoft static analysis helps organizations reduce the time and effort required to build and maintain secure software. Unlike other tools and "SaaS" solutions, Parasoft's highly configurable architecture allows organizations to test for the issues, best practices, and regulatory requirements most critical to their use cases and stakeholders. For organizations just starting with static testing, a smaller set of rules that address the most common coding errors accelerates adoption and time-to-value.

As shown, security needs to be addressed early in the development lifecycle to minimize costs and rework. Parasoft's IDE integrations allow organizations to "shift left", providing developers with the tools they need to build security from the beginning of the development lifecycle. Unlike competitive solutions, Parasoft understands that high quality software is more than just security. Checkers and rules that test for safe coding constructs don't allow vulnerabilities to occur.

Other solutions that claim coverage for industry standards will include incidental rules without evidence of what each rule provides for each risk. Parasoft provides standards-centric reporting that makes it easy to both scale and audit security compliance. Parasoft provides 100% coverage of the guidelines that are statically analyzable for CWE Top 25, CWE On the Cusp, CERT C, CERT C++, and OWASP Top 10, providing complete and mapped analysis and near-zero false positive rates.

Finally, Parasoft's on-premise solution provides organizations with complete control over their intellectual property. No sending sensitive source code into "the cloud" for processing. No off-premise storage of your vulnerability data.

PREVENTION IS BETTER THAN DETECTION

Building security into an application is much more effective and efficient than trying to secure an application by "bolting" security on top of a finished application at the end of the SDLC. Just as you cannot test quality into an application, the same is true for security.

Parasoft enables organizations to embrace software security from the requirements stage of the development lifecycle onward and provide their software engineers with the tools and guidance needed to build secure software.