

A decorative horizontal line consisting of a series of white, interconnected geometric shapes, resembling a stylized zigzag or a series of overlapping triangles.

# The DevSecOps Approach

Using AppSec Statistics to Drive Better Outcomes

# Table of Contents

<b>Foreword</b>	<b>2</b>
<b>Why Read this report</b>	<b>3</b>
<b>Executive Summary</b>	<b>5</b>
Key Takeaways	6
WhiteHat's Methodology	7
<b>Three-Phased DevSecOps Approach for Secure Applications</b>	<b>8</b>
<b>PHASE 1: Risk Discovery and Management</b>	<b>9</b>
Goals, Actions, & Metrics	9
Analysis & Insight	10
Typical DevSecOps Outcomes	12
Additional Insights	13
Mobile Application Security Testing	16
<b>PHASE 2: Release Assurance</b>	<b>18</b>
Goals, Actions, & Metrics	18
Analysis & Insight	19
Typical DevSecOps Outcomes	21
<b>PHASE 3: Developer Enablement</b>	<b>23</b>
Goals, Actions, & Metrics	23
Analysis & Insight	24
Typical DevSecOps Outcomes	24
<b>Lessons Learned</b>	<b>26</b>
<b>Recommendations</b>	<b>27</b>
<b>Innovation in Application Security</b>	<b>28</b>
<b>Appendix - Glossary of Terms</b>	<b>29</b>

# Foreword

WhiteHat Security is now a wholly-owned, independent subsidiary of NTT Security. With this strategic development, we are able to combine the global reach of NTT with WhiteHat's deep expertise in application security. As a result, our research now offers the most comprehensive perspective on the current state of application security, as well as recommendations on how to implement DevSecOps effectively.

Web applications are under constant attack. In fact, according to the 2019 NTT Global Threat Intelligence Report, application-specific and web-application attacks comprise over 32% of all threats, earning the top category of attack activity<sup>1</sup>. Unfortunately, businesses continue to struggle against this rising tide, especially as 43% of organizations globally say they do not have adequate resources or skills in-house to cope with the number of security threats.<sup>2</sup>

There is hope on the horizon, however, in the form of a robust DevSecOps approach. By embedding application security testing at each stage of the software lifecycle, organizations are now able to make demonstrable improvement in app security and significantly reduce time to delivery of secure applications. WhiteHat Security's application security platform provides the foundational DevSecOps capabilities (DAST, SAST & SCA) that organizations require at each stage of their software lifecycle – enabling innovation and security to thrive simultaneously.

The 2019 WhiteHat Application Security Statistics report looks at our underlying application security data to derive conclusions, identify trends and highlight what's working and what's not when it comes to DevSecOps and secure application delivery. This 2019 report is the product of data analysis derived from evaluating data from approximately 17 million application security scans performed by organizations in 2018.

This report shows that organizations that succeeded in improving their security posture last year are those that have embraced a robust, three-phased DevSecOps approach, where application security testing is embedded into each stage of the development lifecycle. With this level of insight and control, business leaders can orchestrate better risk outcomes for their applications and their businesses.

This year, we are also including a short section on innovation in application security. With artificial intelligence and machine learning becoming industrially viable, there is hope that accuracy, speed and guidance no longer need to be mutually exclusive.

**Craig Hinkley**  
CEO, WhiteHat Security

**Matthew Gyde**  
CEO, NTT Security

---

<sup>1</sup> 83% of businesses are sinking or only treading water against the rising tide of cyber security threats. Source: 2019 NTT Security Global Threat Intelligence Report

<sup>2</sup> Source: NTT Risk:Value report 2019

# Why Read this Report



## Application Security has become critical to business success

The cliché ‘there’s an app for that’ underlies today’s business maxim: applications are at the foundation of today’s enterprise. Whether they’re managed in-house or delivered via your service provider, their availability, reliability, and scalability are table stakes for business success – across industries and geographies. As the digital transformation continues, organizations are beginning to realize that security is not merely another table stake, it’s the timber the rest of the organization is built upon.



## Pace and rate of change in today's application development are blindingly fast

Apps are now the way to out-innovate competition across industries which is why teams are increasingly focused on time-to-market and time-to-value when it comes to application development. Each Line of Business (LOB) is driving its own app development, putting pressure on DevOps to scale operations – “more apps, and more apps now.” In response, application development techniques are rapidly evolving: cloud, microservices and APIs are good examples, with new architectures coming online faster than ever before. In the midst of this chaotic change, most agree that understanding and addressing security is essential. Yet, remediating all vulnerabilities for apps that are already online remains an elusive task.



## Security & DevOps are converging – and an approach for success has emerged

By analyzing trends in application security results and their impact over the years, this report will help global organizations improve their app security year-over-year. Part of this effort involves facilitation and active cooperation between IT security and DevOps teams. The phased approach to DevSecOps we outline in this year’s report is a macro-trend that supports the Security and DevOps convergence, and empowers teams to deliver better performing and more secure apps – and meet the goal of rapid innovation and reliable service delivery.

**This report is essential reading for executives, security practitioners and development teams who want to better understand the present state of software security risk, and who seek to benchmark and improve their own organization's performance.**



### **For Business Decision Makers...**

How to measure the effectiveness of your application security investments in helping to mitigate overall business risk.



### **For Security Professionals...**

How to best defend your applications by evaluating how your vulnerability levels and remediation times compare with industry benchmarks.



### **For Application Development and Operation Teams...**

How to develop software more securely by partnering with the security team in adopting technologies and methodologies compliant with your software lifecycle (SLC).

# Executive Summary

The Application Security Statistics report is an annual study. In addition to examining how organizations are faring in their efforts to secure traditional applications, we also examine how effective organizations are in securing modern applications built using Agile development frameworks, microservices, APIs, and cloud architectures. In fact, last year we took our first look at how these evolutionary changes in the SLC have impacted application security.

This year, we're able to share the specific metrics, goals, and tactics for implementing a DevSecOps approach that results in positive business outcomes for your application security program. We'll share year-over-year data analysis that shows how implementing a DevSecOps framework results in improving application security and reducing risks, costs, and complexity – all while accelerating application service delivery and resilience.

Sadly, enterprises are too often the exception to the rule when it comes to application security. Most enterprises still suffer from unmet needs created from team inertia, lack of coordination, and legacy procedures. While technological advances offer the promise of increased efficiency and performance, inadequate application security approaches expose organizations to risks (e.g. vulnerable applications, data breaches, etc.)

This report offers a three-phased approach to application security that has been proven to succeed. In addition to providing the latest statistics on the biggest application security threats, this report offers a DevSecOps framework that builds cross-team consensus on how secure applications are developed, monitored, and measured. Additionally, we'll demonstrate how our suite of application security products and services delivers the specific functionality to support this robust DevSecOps framework for enhanced operations and security.



## The effort required to secure the rapidly growing volume of existing and new applications is overwhelming already short-staffed teams.

Expanded app testing is an encouraging sign, yet remediation/mitigation rates continue to fall. An increased awareness of application security risks has naturally expanded the scope of applications being tested. In fact, in a single year, we saw a 20% increase in the number of apps organizations are testing. At the same time, however, remediation rates have fallen, which is a huge concern. The limited pool of global application security professionals exacerbates the situation due to a constant shortage of skills and resources required to keep up with remediation/mitigation needs.



## Customers are scanning 20% more apps yet remediation rates are still falling.

### AppSec investment is unbalanced across Development, Security and Operations.

While security teams are taking on more accountability, responsibility, and producing more results, they don't have the adequate resourcing or subject matter expertise in engineering and operations to remediate or mitigate found vulnerabilities<sup>3</sup>. End-to-end AppSec requires adequate funding for all functions involved in the production of applications from Development to Security to Operations. The lack of balanced funding results in poor application security outcomes as organizations are unable to fix software vulnerabilities that are found during application security testing.



### Organizations that scan applications in production have a reduced risk of being breached.

Most growing organizations have an ever-expanding attack surface due to the large number of new applications as well as the large number of legacy applications they have in production. Organizations that adopt continuous DAST testing in production, integrated with the software lifecycle, have demonstrably lower risk of being breached in production.



### Organizations that embed security in DevOps are able to reduce risk, reduce cost and improve time-to-market.

In addition to reaping much higher fix rates for vulnerabilities, they are also able to shrink overall exposure in terms of time-to-fix (TTF) for the same vulnerability types in production apps. Faster risk reduction means more uptime, better performance, and happier end users. Plus, developers learn that better security can also mean fewer headaches now and in the future.



## In a single year, we saw over a 50% increase in unpatched library vulnerabilities.

### Embeddable components within the software supply chain account for 1/3 of all AppSec vulnerabilities.

Unpatched libraries continue to be the most prevalent vulnerability discovered by Software Composition Analysis (SCA) testing. In a single year, we have witnessed over a 50% increase in unpatched library vulnerabilities. This points to a dangerous trend that we see continuing to snowball. As more open source and third-party software is embedded, it's creating an inherently insecure environment for production apps - more than 1/3 of all AppSec risks are inherited rather than written. If software vendors producing embeddable components improved their security standards, it would have positive cascading security outcomes throughout the rest of the global application landscape.

# WhiteHat's Methodology

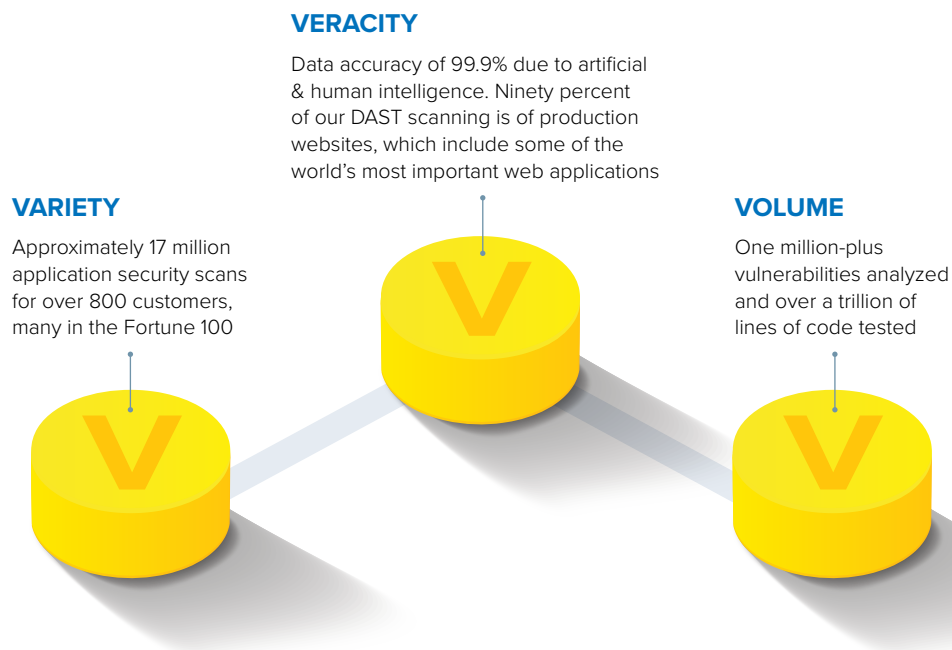
WhiteHat Security has been publishing this report since 2006. The study comprises statistical data and analysis gathered from continuously updated security testing information in WhiteHat Sentinel™, a cloud-based application security platform.

The conclusions in this report are analyzed from aggregated results of approximately 17 Million application security scans performed by WhiteHat's Application Security Testing platform, Sentinel. Sentinel inspects the full spectrum of applications, including components and shared libraries.

The report's statistical analysis focuses exclusively on assessment and remediation data obtained as a result of application security tests performed by WhiteHat Security. Data is segmented along multiple dimensions including vulnerability risk levels, vulnerability classes, and industries. Data analysis uses key indicators that include the likelihood of a given vulnerability class, remediation rates, time-to-fix, and age of open vulnerabilities.

Risk levels are based on the rating methodology of Open Web Application Security Project (OWASP). Vulnerabilities are rated on five levels of risk – Critical, High, Medium, Low and Note. Critical and high-risk vulnerabilities taken together are referred to as "serious" vulnerabilities. Vulnerability classes are based on the threat classification of Web Application Security Consortium (WASC).

## The methodology used to produce the 14th edition of the report ensures the following:



In this section of the report, we compare the overall data set (including all application security data irrespective of assessment technique for the application) with a "DevSecOps" data set (including application security data where at least two assessment techniques are applied and a phased DevSecOps program was implemented).



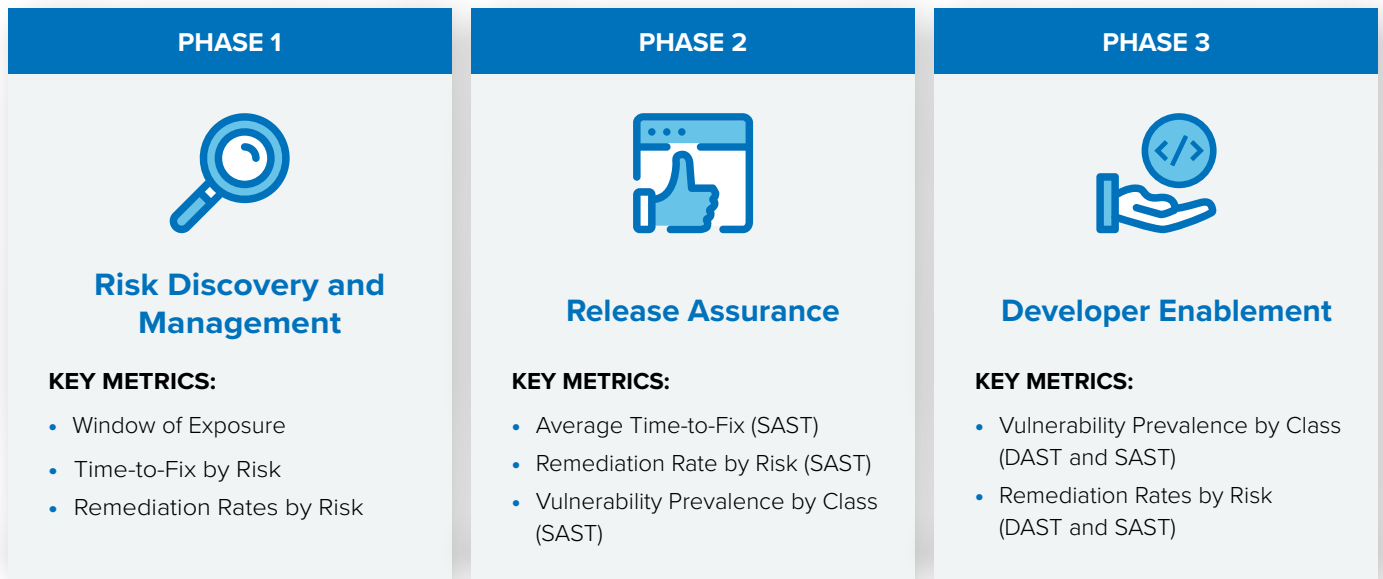
# Three-Phased DevSecOps Approach for Secure Applications

**“If you can’t measure it, you can’t improve it.”**

– Peter Drucker, *The Practice of Management*

The DevSecOps approach outlined in this report is based on the core principle of effective business practices for software development. It prioritizes risk discovery and effective risk mitigation.

Unless and until you can measure something, you can’t make it better. Or, in our case, more secure. At each stage of the software lifecycle, capturing core metrics like those we outline in the DevSecOps approach will offer advanced insight into how applications are already at risk or can be exposed to risks, even when they are in production. These metrics offer the chance to make adjustments when they are far less costly to the organization.



**This phased approach to DevSecOps that has quantifiable success within our customer base offers a clear roadmap for:**

1. Comprehensively addressing the diverging needs of existing in-production applications and an increasing number of new applications coming online
2. Implementing application security in a programmatic manner with verifiable metrics for tracking success
3. Embedding security into the fabric of the application development process

# Risk Discovery and Management



### GOALS

Incorporate DAST to discover risk and use the following application security metrics as key performance indicators to measure your success over time.



### ACTIONS

During this phase, the organization is focused on discovering exploitable risks in the application in its current released state. The findings will contribute to the organization's top line risk metric for the application. They will also form the basis of comparison and analysis for the purpose of planning and prioritizing remediation initiatives, as well as providing the canonical data model for release assurance and developer enablement activities.



### METRICS



#### Window of Exposure

The "Window of Exposure" metric represents the amount of time that an application has a serious vulnerability that can be exploited to data breaches. Develop an SLA for "Window of Exposure" for your organization and aggressively try to reduce it for all your applications.



#### Time-to-Fix by Risk

The "Time-to-Fix by Risk" metric provides an industry baseline for how long it takes to fix a vulnerability and associates that with risk (to the organization). Develop an SLA for "Time-to-Fix by Risk" for your organization and work aggressively to reduce it for vulnerabilities in your applications.



#### Remediation Rate by Risk

The "Remediation Rate by Risk" metric represents the percentage of vulnerabilities that are fixed, organized by level of risk. Develop SLAs and procedures/training to support the SLAs that promote remediation efforts by taking a risk-based approach. Track the "DAST and SAST Remediation by Risk" metrics to see that the most serious vulnerabilities are being prioritized for remediation.

**Organizations that have implemented the three-phased DevSecOps approach have reduced their Window of Exposure for apps that are always vulnerable to an average of 22%<sup>4</sup>. An average of more than 50% of apps are always vulnerable for organizations that have not adopted DevSecOps.**

<sup>4</sup> Considering that this population of apps represents more than 35 million lines of code, reducing the WoE by this amount has significant downstream benefits for reducing the overall risk surface area for the enterprise.

# Analysis & Insight

## Window of Exposure (average number of days)

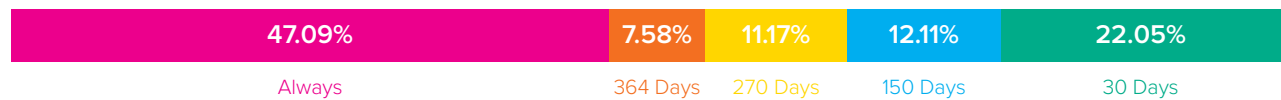
### Transportation and Warehousing



**KEY TAKEAWAY**

Overall, several industries have reduced their Window of Exposure (WoE) by focusing on outcomes. In particular, Transportation & Warehousing showed the most improvement, perhaps due to some high profile data breaches in 2018.

### Finance and Insurance



**KEY TAKEAWAY**

Finance remained the same between 2017 and 2018 (47.09 days).

### Healthcare and Social Assistance



### Retail Trade



**KEY TAKEAWAY**

Healthcare (43.05 days) and Retail (55.42 days) both improved year-over-year, consistently reducing their Window of Exposure for their applications from 2017 to 2018.

### Information Technology



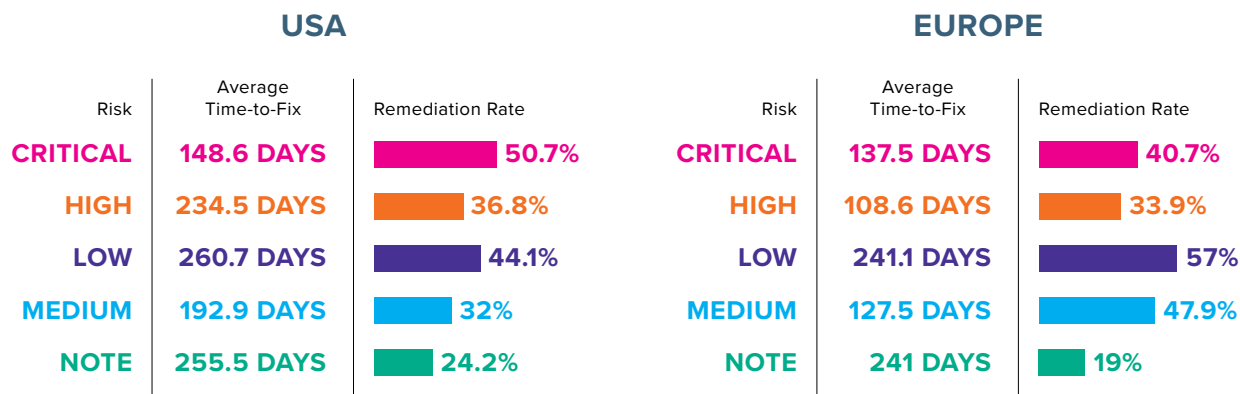
**KEY TAKEAWAY**

At 56.88, Information Technology has not reduced its WoE year-over-year. In fact, WoE has increased from 2017 to 2018. Despite the fact that IT teams are well-versed in good AppSec best practices, they do not seem to be putting them into practice.

## Average Time-to-Fix by Risk & Remediation Rate by Risk

A combined analysis of Time-to-Fix and Remediation Risk is essential to gain a complete picture of the current state of your application security risk posture.

### *Average Time-to-Fix by Risk & Remediation Rate by Risk - DAST Data*



#### KEY TAKEAWAYS

- ◆ In general, remediation rates have fallen, which is a huge concern. We can attribute this to an increased awareness and focus on application security, which naturally expands the scope of applications to be tested. In fact, in a single year, we saw a 20% increase in the number of apps organizations are testing. Common sense tells us that when you are testing more apps, but not increasing your investment in fixing vulnerabilities, remediation rates will naturally rise.
- ◆ EU remediation rates are far worse than the rest of the world, which is quite a surprise considering that GDPR is now in full force. Our hypothesis is that organizations are still in the early phases of implementing GDPR compliance, and next year we expect to see marked improvement in our EU customers' remediation rates.
- ◆ Traditional approaches for application development are failing; as one might guess, when you run a single DAST scan just to satisfy a checkmark on an auditor checklist, you may not move the needle on reducing AppSec risks. Instead, structured DevSecOps phase-wise implementation is essential. Specifically, feed DAST results into a bug tracking system to get prioritized and fixed, then run DAST again to verify those fixes take hold.

# Typical DevSecOps Outcomes

In this section, we will present the outcomes experienced by organizations who have implemented at least two assessment techniques and have taken a phased-approach to DevSecOps. Specifically, this section highlights the metrics for Phase 1 of the three-phased DevSecOps approach.

## Window of Exposure (WoE)

When we examine the WoE rates for apps developed via a robust DevSecOps framework, we can see the benefits of these outcomes.



### KEY TAKEAWAYS

- ◆ Overall Window of Exposure is significantly lower than the industry average. We attribute this outcome to teams prioritizing the testing of in-production applications using DAST to discover vulnerabilities that are currently exploitable and then taking a risk-based approach to mitigating or remediating these vulnerabilities. The testing cycle comes full circle once the app is re-tested using DAST to verify fixes are implemented correctly.
- ◆ The apps that have a permanent WoE (“e.g. WoE always”) is still higher than woe\_364 and woe\_270, but significantly less than the overall WoE\_Always (~50%) for apps developed by organizations not yet implementing DevSecOps.

## Average Time-to-Fix by Risk and Remediation Rate by Risk (DAST data)

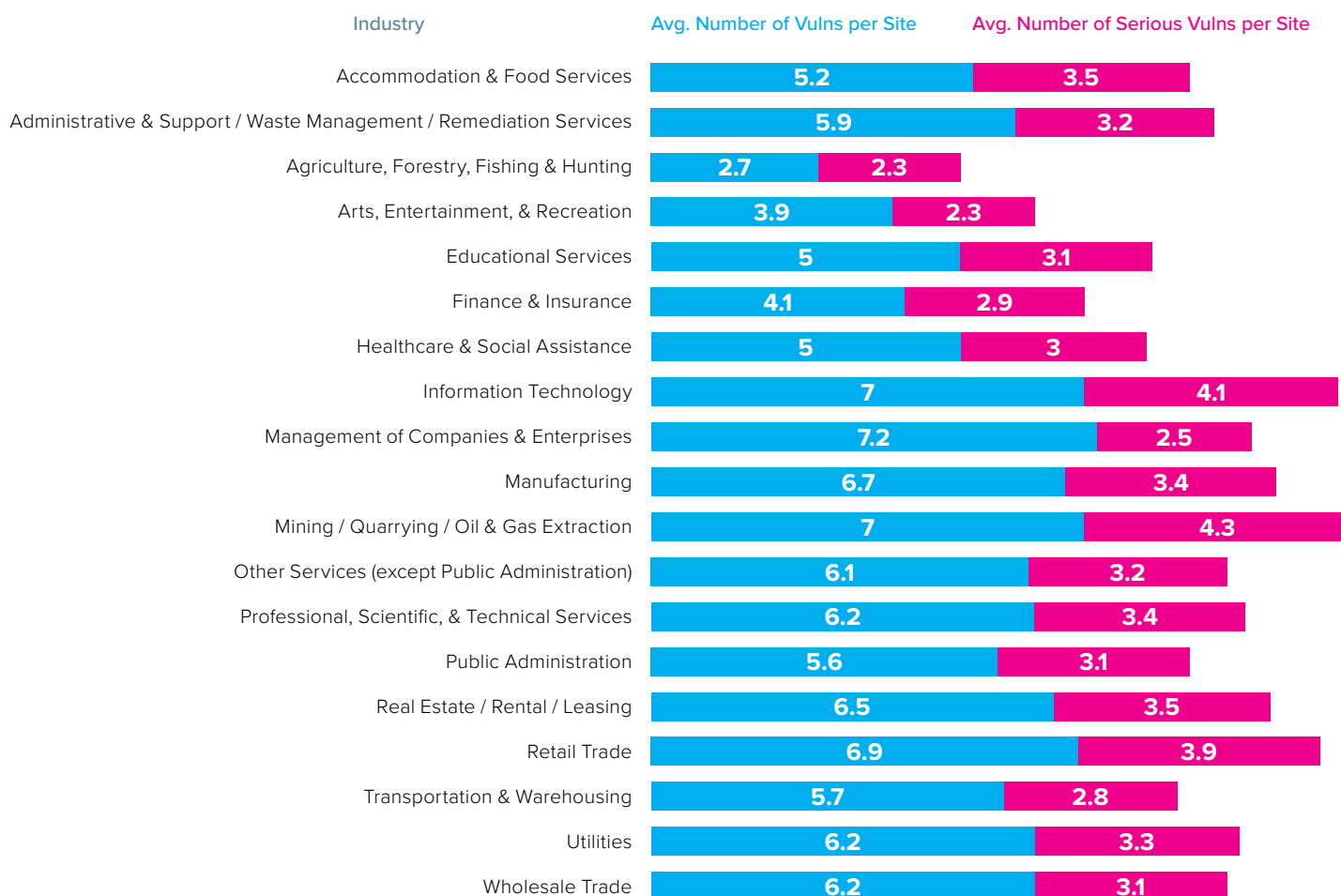
Risk	Average Time-to-Fix	Remediation Rate
CRITICAL	22.1 DAYS	89.36%
HIGH	49.7 DAYS	87.25%
LOW	284 DAYS	67%
MEDIUM	71.6 DAYS	91.36%
NOTE	289.4 DAYS	8.35%

### KEY TAKEAWAYS

- ◆ These Time-to-Fix metrics reflect a rational and well-implemented AppSec strategy. Just as one would expect, Time-to-Fix is lowest for critical vulnerabilities and highest for notes, since these vulnerabilities are considered optional rather than ‘required to be fixed.’
- ◆ While organizations are doing relatively well with remediating critical, high, and medium ranked vulnerabilities, there is room for improvement to more quickly remediate vulnerabilities in the low risk category.

# Additional Insights

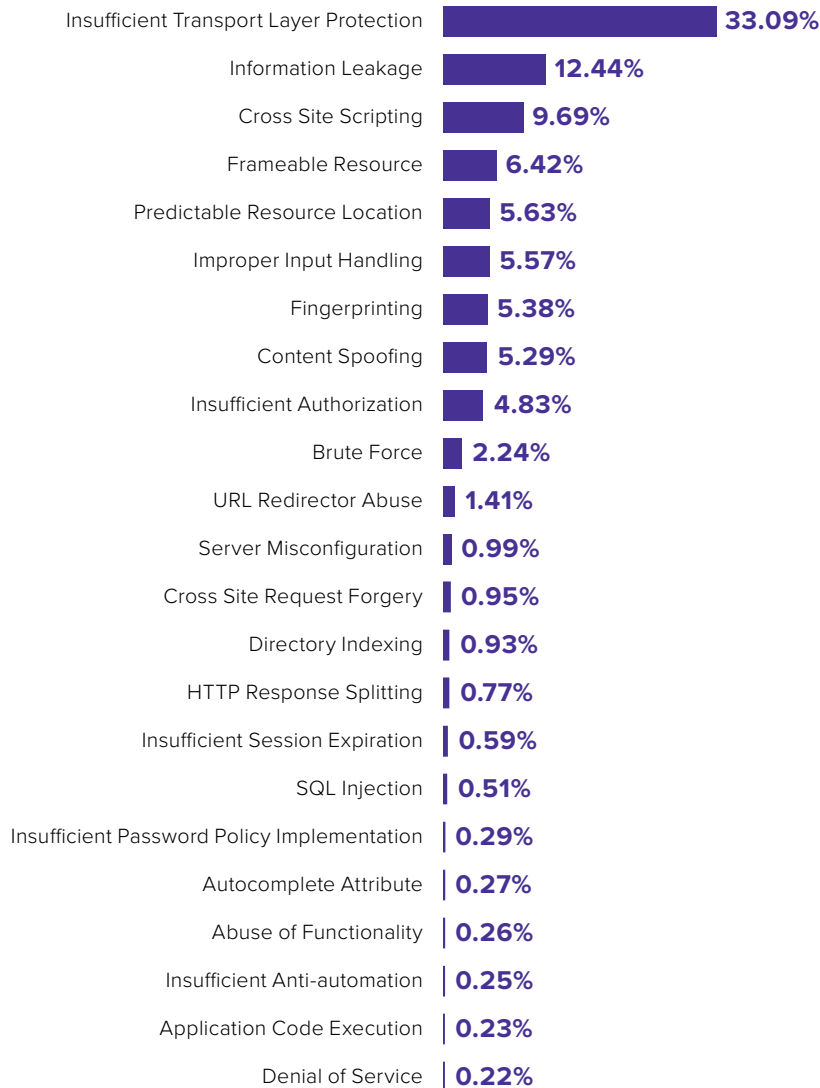
## Vulnerabilities by Industry per Site



### KEY TAKEAWAYS

- ◆ The overall average of 3.2 critical vulnerabilities per website across all industries has remained constant over the last three years.
- ◆ IT is one of the worst offenders when it comes to the sheer volume of vulnerabilities. One possibility could be based on its lack of regulation as compared to well-regulated industries like finance or healthcare. Additionally, the IT industry has a curious relationship to risk. Rather than taking on the security risks associated with their own products, IT passes this risk on to their customers. As the source of all of these vulnerabilities that show up in other industries, IT is also in a unique position to be a hero. If this industry got its act together, there would be a positive cascading effect and outcome throughout the rest of the global economy.
- ◆ The Arts & Entertainment industry is one of the best industries in terms of number of vulnerabilities. Our hypothesis is that these companies recognize the essential need to protect their highly monetizable content, which is only possible through the implementation of a strong DevSecOps framework. In other words, what's continually measured can be continually secured.
- ◆ Honorable mentions go to the manufacturing industry, which found fewer vulnerabilities than last year. For these companies, securing their digital supply chains has become an existential priority. As more and more of their delivery infrastructure becomes connected, the software lifecycle is now as essential as the factory line.
- ◆ Despite their well-regulated status, the finance, healthcare, retail, and utilities industries had more vulnerabilities than they did last year. We find that while these regulated industries have application security programs, a majority of these programs are focused on check-the-box compliance needs.

## Vulnerability Prevalence by Class (DAST)

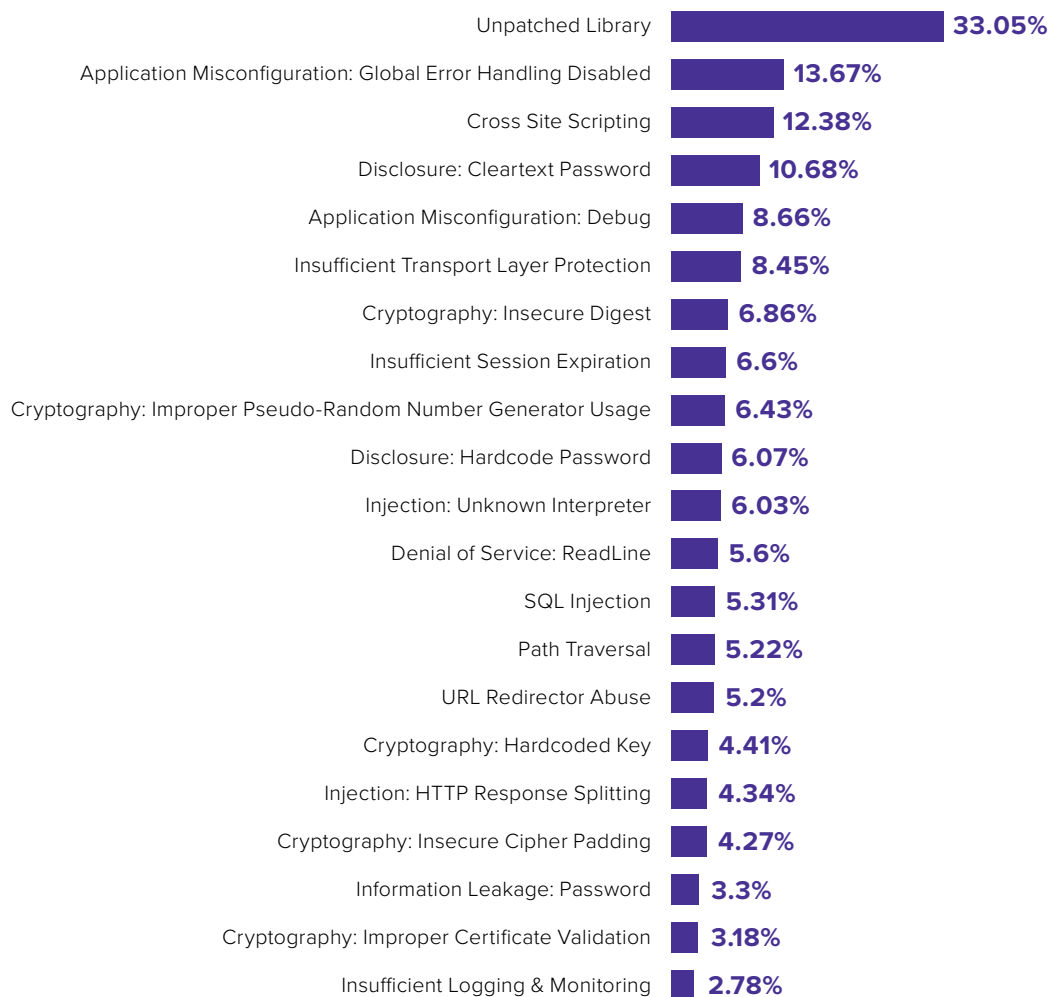


### KEY TAKEAWAYS

- ◆ Insufficient Transport Layer Protection (TLS) continues to be the most prevalent vulnerability type. In the past, without any TLS exploits available, this vulnerability was purely theoretical. Of course, that is no longer the case. Many TLS exploits are currently being written and released, and with more organizations moving their production apps to the cloud, we anticipate this trend will continue. Specifically, we expect that TLS testing will evolve in two ways: a) theoretical exploits will become manifest, and b) new tests will continually be written.
- ◆ Cross-site request forgery (XSRF) has decreased significantly since last year. As more API-first and API-based apps are being developed, XSRF protections are automatically implemented at the framework-level. Additionally, browser-level protections are often enabled by default preventing malicious intent requests like "get" or "delete" requests without sufficient authorization.

It is important, at this point, to briefly examine SAST vulnerabilities by prevalence to get a complete picture of DAST vulnerability prevalence.

## Vulnerability Prevalence by Class (SAST)



### KEY TAKEAWAYS

- ◆ Information Leakage vulnerabilities point to application misconfiguration issues that can be easily resolved by doing more SAST testing to prevent these from showing up in DAST results and endangering production apps.
- ◆ In both DAST and SAST data sets, Cross Site Scripting (XSS) is the third most common vulnerability type. In theory, if organizations are discovering XSS vulnerabilities in SAST (pre-production) and remediating them at this stage, we would not expect to see these vulnerabilities in DAST results.
- ◆ Only one third of Insufficient Transport Layer Protection vulnerabilities are seen in SAST results (8%) compared to how prevalent these are in DAST results (33%). This suggests that TLS errors may likely be introduced outside of the application, and perhaps inherited as part of the network infrastructure and operations. We recommend that DevOps and security teams work closely with operations teams to get to the root cause of this common vulnerability and fix it. This is yet another reason why it is essential to conduct SAST and DAST testing in concert to capture a comprehensive view of application security health.
- ◆ Injection vulnerabilities are on the rise. In isolation, these vulnerabilities would never be reportable. However, in a highly containerized architecture, these vulnerabilities have much more potential to do harm. With APIs and microservices playing critical roles in this new architecture, injection vulnerabilities will continue to rise in importance, prevalence, and impact. And, of course, this trend provides even more reason to use both SAST and DAST in your risk discovery process.



## Mobile Application Security Testing

Consumers rely on dozens of mobile apps to shop, bank, travel, and play. But what most don't know is that an abundance of Android apps have privacy shortcomings that put their personal data at risk. Cybercrime rates are soaring, and more than 60 million Americans<sup>5</sup> have fallen victim to fraud or identity theft stemming from a breach of their personal information.

A review of 250 popular Android mobile apps from leading brands reveals that 70% leak sensitive personal data. These online retail, brick-and-mortar retail, finance, insurance and travel apps have privacy risks that expose personally identifiable information (PII). Consumers should carefully consider halting use of apps that don't safeguard their privacy while lobbying app makers to fix them. Likewise, organizations should respect consumer privacy demands by ensuring their developers follow best practices for building secure mobile apps and close any privacy gaps that are found.

Conducted in February to April 2019, the evaluation focused on popular publicly available Android mobile apps downloaded from the Google Play™ store using the NowSecure automated mobile AppSec security testing platform. The analysis zeroed in on leakage of unencrypted personal information stored on the mobile device and transmitted over the network, as well as potential exposure to phishing attacks.

# More than 60 Million Americans have fallen victim to fraud or identity theft.



Out of 250 Android apps,  
nearly **3 in 4** leak  
sensitive personal data



The risk of privacy compromise is greatest in mobile apps from online digital marketplaces and leading brick-and-mortar retailers. Travel apps which are among the most heavily used apps in Google Play™ fell somewhere in the middle. And as expected due to the level of regulation imposed on the finance and insurance industries, those mobile apps fared the best in the analysis.

**82%** of tested retail  
apps leaked sensitive data



**67%** of tested travel  
apps leaked sensitive data



**50%** of tested finance  
and insurance apps leaked  
sensitive data



# Can Bad PR Lead to Good AppSec Outcomes?

Perhaps the Transportation and Warehousing Industry has had better application outcomes thanks to the tough lessons learned from ugly headlines. Lest we forget these tough, yet valuable, lessons, here are a few of the big ones.



DUBAI, JANUARY 2018

## Transportation Network Company

**What happened?** 14 million records were stolen. Data included personal information such as names, email addresses, phone numbers, and trip data.

**How did it happen?** According to the company, "access was gained to a computer system that stored customer and driver account information."<sup>6</sup>



USA, JANUARY 2017

## Travel Fare Aggregator Website

**What happened?** Over a two-year period starting in January 2017, attackers stole 880,000 credit card holder records that, in addition to payment card information, also included personal data such as billing addresses, phone numbers and emails.

**How did it happen?** Cyberattackers accessed travel bookings via exploiting the company's website application.<sup>7</sup>



UNITED KINGDOM, AUGUST 2018

## International Airline

**What happened?** In August and September 2018, more than 380,000 card payment transactions were 'compromised', leaking financial and personal data of the company's customers.

**How did it happen?** Cyberattackers took advantage of vulnerabilities in the web application to compromise the booking process.<sup>8</sup>



HONG KONG, MARCH 2018

## International Airline

**What happened?** The company discovered an enormous data breach. 9.4 million records were stolen. This huge volume of personal data included 860,000 passport numbers, 245,000 Hong Kong identity card numbers, 403 expired credit card numbers, and 27 credit card numbers without the card verification value (CVV).

**How did it happen?** Not many details have been publicly released, other than that passenger data was accessed 'without authorization.'<sup>9</sup>

KF BR 392

# Release Assurance



### GOALS

Organizations conduct the necessary risk assessment due diligence to verify software quality prior to every release (whether classified as minor or major releases). Integrate SAST, SCA and DAST into your software lifecycle and leverage the following application security metrics to baseline your organizations “Release Assurance” strategy.



### ACTIONS

The organization aims to ensure that a new release candidate does not add additional risk compared to the application’s current release. Additionally, the organization aims to ensure that remediation activities have been successful in reducing the application’s risk profile. The release assurance concern is an ideal place to integrate portions of the assessment methodology into the release pipeline and is most successful when combined with a pre-existing and robust risk discovery program.



### METRICS



#### Average Time-to-Fix (SAST)

The “Average Time-to-Fix” metric provides an industry baseline for how long it takes to fix a vulnerability in general. Develop an SLA for “Average Time-to-Fix” for your organization and aggressively try to reduce it for all vulnerabilities.



#### Remediation Rate by Risk (SAST)

The “Remediation Rate by Risk” metric represents the percentage of vulnerabilities that are fixed, organized by level of risk. Develop SLAs along with procedures and training to support the SLAs that promote remediation efforts by taking a risk-based approach. Track the “DAST and SAST Remediation by Risk” metrics to ensure the most serious vulnerabilities are being prioritized for remediation.













#### Vulnerability Prevalence by Class (SAST)

Organizations are increasingly adopting open source and commercial off-the-shelf components to rapidly innovate. As such, the likelihood of inheriting vulnerabilities is higher than ever before. Baseline your development teams’ security goals by creating an SLA around reducing the most likely vulnerabilities by class.

## Analysis & Insight

In this section, we will discuss the key findings as they relate to Phase 2 measures, namely Average Time-to-Fix and Vulnerability Prevalence by Class.

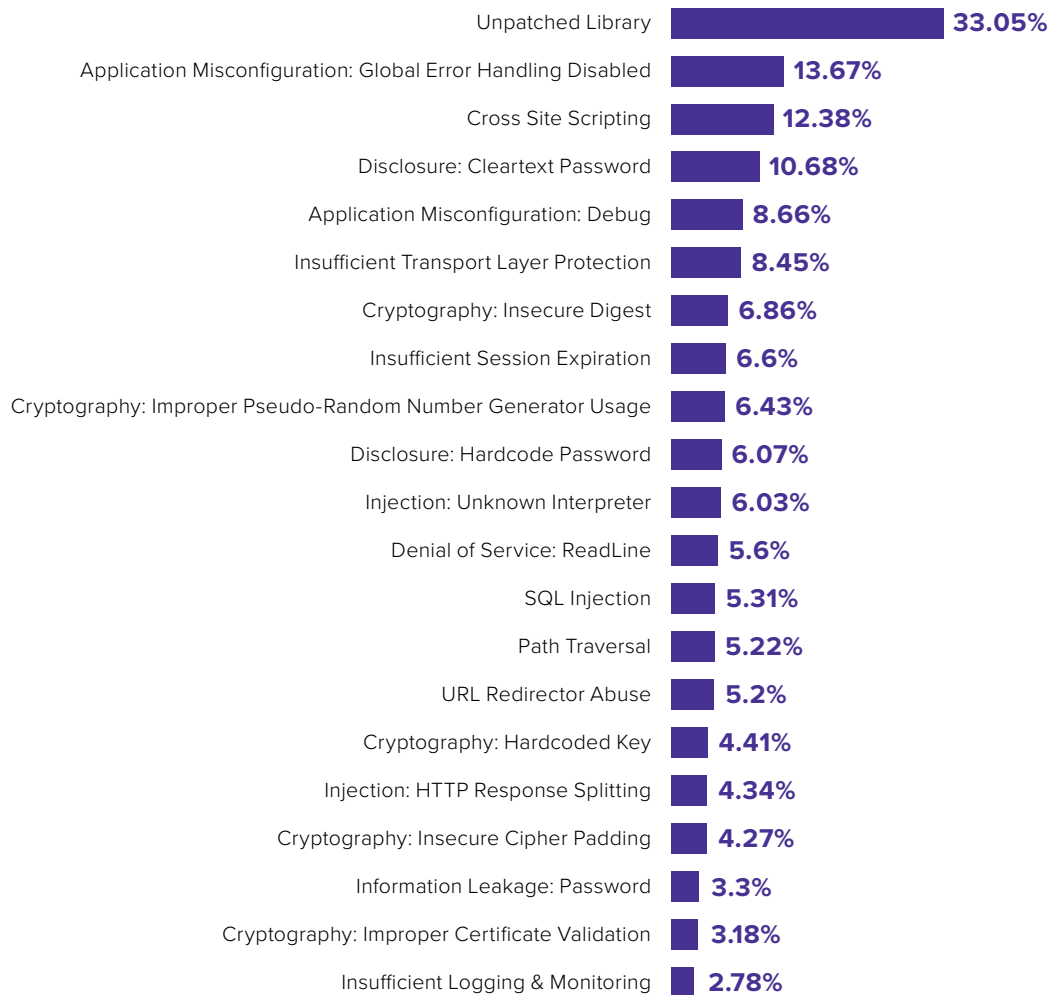
### Average Time-to-Fix and Remediation Rate by Risk

SAST			DAST		
Risk	Average Time-to-Fix	Remediation Rate	Risk	Average Time-to-Fix	Remediation Rate
<b>CRITICAL</b>	<b>51.7 DAYS</b>	 <b>55.2%</b>	<b>CRITICAL</b>	<b>148.6 DAYS</b>	 <b>50.7%</b>
<b>HIGH</b>	<b>102.3 DAYS</b>	 <b>36.2%</b>	<b>HIGH</b>	<b>234.5 DAYS</b>	 <b>36.8%</b>
<b>LOW</b>	<b>97.8 DAYS</b>	 <b>41.9%</b>	<b>LOW</b>	<b>260.7 DAYS</b>	 <b>44.1%</b>
<b>MEDIUM</b>	<b>117.9 DAYS</b>	 <b>28.1%</b>	<b>MEDIUM</b>	<b>192.9 DAYS</b>	 <b>32%</b>
<b>NOTE</b>	<b>126.7 DAYS</b>	 <b>19.7%</b>	<b>NOTE</b>	<b>255.5 DAYS</b>	 <b>24.2%</b>

#### KEY TAKEAWAYS

- ◆ While we're seeing similar outcomes in terms of vulnerability prevalence between SAST and DAST testing, we're seeing much lower time-to-fix metrics in SAST results.
- ◆ Clearly, a focus on Critical vulnerabilities is driving better outcomes earlier in the SLC from a remediation rate perspective. Organizations take 1/3 of the time to fix a Critical vulnerability when it's discovered during SAST testing (early in the SLC) compared to discovering it via DAST. Yet, more attention on high risk vulnerabilities is needed.
- ◆ When you incorporate SAST into your AppSec program, you can expect much better outcomes as your fix rate increases and time-to-fix (TTF) shrinks for the same vulnerability types in production apps.
- ◆ A focus on high-risk vulnerabilities is still sorely lacking. This suggests that the volume of vulnerabilities and applications organizations are managing has steadily increased while investment in remediation and security teams has stagnated.

## Vulnerability Prevalence by Class (SAST)



### KEY TAKEAWAYS






- ◆ Unpatched libraries continue to be the most prevalent vulnerability discovered by SAST testing. Clearly, more Software Composition Analysis (SCA) is needed to identify these inherited vulnerabilities before apps are moved into production. The prevalence has increased considerably from last year (over 50% increase). It points to a dangerous trend that more open source and third-party software is being embedded. This creates a situation where more than 1/3 of all risks are inherited.
- ◆ 10 out of the top 20 CVEs detected by SCA and over 50% of SCA vulnerabilities are directly or indirectly related to API-based applications. This trend points to the importance of running SCA scans with modern microservices architecture based API-first applications.
- ◆ Despite increased awareness of the dangers of Cross Site Scripting (XSS), it remains at the top. Over the past three years, XSS vulnerabilities have remained in the number two position. Despite widespread knowledge of this vulnerability and its broad and crippling impacts, the industry continues to fail to address XSS issues.
- ◆ Application Misconfiguration Errors are on the rise. While development frameworks are taking hold, organizations are failing to change default configurations which leaves apps and their data exposed to risk (e.g. leaving exposed default administration endpoints, enabled debugging, or failing to enable TLS controls). Some of the world's largest data breaches were due to mistakenly relying on default configurations that are unnecessarily insecure and easy to exploit. According to security firm Skyhigh Networks, 7% of all S3 buckets have unrestricted public access, and 35% are unencrypted, leaving data unnecessarily exposed to the threat of being stolen or misused<sup>10</sup>. Since the default configuration for MongoDBs is to allow external connections to the internet, ransomware attackers have targeted these data stores to steal and replace data in return for the victim paying a ransom.

## Typical DevSecOps Outcomes

In this section, we will present the outcomes experienced by organizations who have implemented at least two assessment techniques and take a phased-approach to DevSecOps (typically DAST & SAST). Specifically, this section highlights the metrics for Phase 2 of the three-phased DevSecOps approach.

### Average Time-to-Fix (TTF) by Risk and Remediation Rate by Risk

When we examine the TTF rates for apps developed via a robust DevSecOps framework, we can clearly see the benefits of these outcomes (outlined below).

Risk	Average Time-to-Fix	Remediation Rate
<b>CRITICAL</b>	<b>51.9 DAYS</b>	 <b>66.17%</b>
<b>HIGH</b>	<b>72.8 DAYS</b>	 <b>53.47%</b>
<b>LOW</b>	<b>58.7 DAYS</b>	 <b>49.07%</b>
<b>MEDIUM</b>	<b>64.6 DAYS</b>	 <b>32.55%</b>
<b>NOTE</b>	<b>169.9 DAYS</b>	 <b>17.77%</b>

#### KEY TAKEAWAYS

- ◆ Organizations that implemented a three-phased DevSecOps framework achieved a lower average TTF as well as better remediation rates across all risk categories. This is a measurable improvement from the global average and eventually contributes to better AppSec posture in production.
- ◆ These organizations have found traction by integrating SAST, SCA and DAST results into their bug tracking and software lifecycle management tools. These results continue to demonstrate the adage “that which is measured is managed,” and we, of course, would add “secured”. In simple terms, security defects are treated and addressed in the same way as software defects (e.g. security is a subset of core functionality rather than as an adjunct). These teams discuss security defects in stand-ups and while planning software releases, as well as how to remediate them, just as they would any other aspect of functionality.



## The AppSec Funding Model: Woefully Out of Date

The funding model for application security is flawed and incomplete. While security teams are taking on more accountability, responsibility, and producing more results, they don't have the adequate resourcing from engineering and operations to fix AppSec vulnerabilities<sup>11</sup>.

We find that a lack of organizational security awareness prevents organizations from taking a holistic approach to planning for AppSec. Ensuring that every organization has access to the right level of training and enablement is the first step to creating a well-rounded AppSec strategy.

There is a dearth of AppSec subject matter expertise in AppSec, and that prevents a well-thought out plan from succeeding. Enable your AppSec SMEs to focus on driving change by adopting a platform solution that provides the capabilities, the accuracy, the speed and the guidance required to scale your SMEs to the entire organization's DevSecOps needs.

Finally a culture of collaboration between Development, Security and Operations that is based on mutual Objectives and Key Results (OKRs) will help balance resources between AppSec testing and AppSec remediation, and establish company-wide consensus on the operational game plan – from scoping to testing to fixing. As part of the AppSec scoping effort, this will help identify remediation costs (and anticipate obstacles), and add these to the funding request.

<sup>11</sup> According to a 2018 DevSecOps Community Survey, nearly one-half of developers say they don't have enough time to spend on security, even though they are aware of its importance. Source: <https://www.whitehatsec.com/blog/partnership-with-rs-eases-burden-of-remediation-on-devsecops/>

# Developer Enablement



### GOALS

Initiate programs to educate and empower developers throughout the SLC, including adding AppSec tools to the developer workspace and training developers on their use. Like most strategically successful initiatives, developer enablement is best done as a data-driven and customer-driven enterprise. All activities are informed by application assessment data, and all remediation decisions and activities are viewed in the context of the customer perspective: the protection of their data, and the quality of their experience. Specifically, the data gathered from the first two phases (Risk Discovery and Release Assurance) will guide the focus areas for Developer Enablement.



### ACTIONS

The organization aims to reduce the number of vulnerabilities developers introduce into the release pipeline by bringing assessment and education into the developer workspace. Lightweight application security tools can be run in the developer's own sandbox to eliminate security issues before they are committed to version control or the release pipeline. Education is conducted based on the common findings of the risk discovery and release assurance, secure enterprise libraries are developed to replace commonly used components that are prone to misuse, and a question and answer feedback loop is established via integrated security knowledge bases or direct interaction with application security experts.



### METRICS

Focus on staff training based on the following application security metrics to drive your own organization's Developer Enablement strategy.



#### Vulnerability Prevalence by Class (DAST and SAST)

Use the "Vulnerability Prevalence by Class" (for both DAST and SAST) from this report to set up focused and recurring training for your development, operations and security teams. Track your teams' progress by tracking Vulnerability Prevalence by Class for the applications they develop and evolve your training efforts to meet the evolving needs of your teams.



#### DAST and SAST Remediation by Risk

Use the "DAST and SAST Remediation by Risk" metric to baseline your teams' goals. Develop SLAs and procedures/training to support the SLAs that promote remediation efforts by taking a risk-based approach. Track the "DAST and SAST Remediation by Risk" metrics to see that the most serious vulnerabilities are being prioritized for remediation.



## Analysis & Insight

### KEY TAKEAWAYS

- ◆ While Insufficient Transport Layer Protection related vulnerabilities found during DAST scanning are not entirely related to developer errors it is an issue that merits education within Development and DevOps teams. As more product teams become full-stack teams, having knowledge and training around this topic will go a long way in timely mitigation and remediation of these vulnerabilities.
- ◆ By analyzing DAST & SAST Vulnerability Prevalence by Class, it is clear that developers need training on how to avoid Information Leakage and Cross Site Scripting errors. As basic as these vulnerability types seem, they still constitute over 20% of the vulnerabilities we find.
- ◆ The Remediation Rate by Risk data for DAST and SAST suggests that development teams need additional information to prioritize a larger volume of critical vulnerabilities during sprint cycles. Currently less than 60% of critical vulnerabilities are being remediated.

## Typical DevSecOps Outcomes

In this section, we will present the outcomes experienced by organizations who have implemented at least two assessment techniques and take a phased-approach to DevSecOps and have integrated AppSec training within their software development teams.

By tracking Vulnerability Prevalence by Class, security managers and development managers in these organizations are able to devise better strategies to remediate frequently occurring security flaws. For example, many SQLi errors can be fixed at an architectural level by using variable binding abstracted using a simplified facade such as using one or both of the Data Access Object or Object Relational Mapping patterns.

This report analyzes the time spent by course per individual to draw insights around the current focus areas for security enablement. Organizations that prioritize integrating security training and enablement find that development teams do actually spend considerable time learning about security when secure development becomes a requirement.

## Time Spent on Secure Development Courses per Individual

WhiteHat Security: OWASP Top Ten for Developers	<b>600 minutes</b>
WhiteHat Security: Building Secure JavaEE Applications	<b>257 minutes</b>
WhiteHat Security: OWASP Top Ten for Managers	<b>109 minutes</b>
WhiteHat Security: General Security Awareness	<b>103 minutes</b>
WhiteHat Security: Building Secure JavaScript Applications	<b>92 minutes</b>
WhiteHat Security: Building Secure Mobile Applications	<b>76 minutes</b>
WhiteHat Security: Foundational Exam	<b>62 minutes</b>
WhiteHat Security: Integrating Security Throughout the SDLC	<b>56 minutes</b>
WhiteHat Security: Defensive Enterprise Remediation Series	<b>49 minutes</b>
WhiteHat Security: Threat Modeling	<b>42 minutes</b>
WhiteHat Security: Building Secure ASP.NET Applications	<b>24 minutes</b>
WhiteHat Security: OWASP Mobile Top Ten for Developers	<b>14 minutes</b>

This report finds that there is strong correlation between the time individual developers spend in training and long-term DevSecOps outcomes. Organizations that consistently demonstrate higher remediation rates and lower time-to-fix for higher risk vulnerabilities typically incorporate topical, timely and repeated developer enablement.

### DAST (Time-to-Fix and Remediation Rates)

For organizations that combine developer enablement in their overall AppSec program, the average time-to-fix a critical vulnerability found in production is approximately 28 days when compared to approximately 148 days for organizations that either do not combine developer enablement within their AppSec program or do not offer developer enablement at all.

Similarly, remediation rates of critical vulnerabilities found in production also sharply increases to almost 90% compared to approximately 50%.

Both of these trends, in part, are attributable to increased awareness about the impact of these vulnerabilities in a production system as well as an improved knowledge of fixing security vulnerabilities in code.

### SAST (Time to Fix and Remediation Rates)

For organizations that combine developer enablement in their overall AppSec program, the average time-to-fix a critical vulnerability found during a development sprint is approximately the same when compared to the average time-to-fix organizations that either do not combine developer enablement within their AppSec program or do not offer developer enablement at all.

However, there is a sharp increase in SAST remediation rates for organizations that combine developer enablement into their overall AppSec program. This, in part, is attributable to increased awareness about identifying and fixing security vulnerabilities in code.

Average time to fix a critical vulnerability found in production for an organization that combines developer enablement in their AppSec Program

**28 DAYS**

Average time to fix a critical vulnerability found in production for an organization that does not combine developer enablement with their AppSec Program or does not offer developer enablement at all

**148 DAYS**

# Lessons Learned

The DevSecOps framework is designed to deliver positive business outcomes to those organizations that embrace it. Specifically, by implementing these three phases into their SLC, organizations that succeed with DevSecOps have achieved the following:



## Lower overall business risk from applications in production

- ◆ Reduced their Window of Exposure for apps that are always vulnerable to an average of 22%<sup>12</sup> compared to an average of 50% in traditional organizations.
- ◆ Prioritized the testing of in-production applications using DAST to find vulnerabilities that are currently exploitable and then mitigating these vulnerabilities first.



## Integrated hence impactful DevSecOps

- ◆ Achieved a lower average Time-to-Fix across all risk categories. These teams have found great traction by integrating DAST results into their bug tracking and software lifecycle management tools.
- ◆ Aligned their DevSecOps goals with their SLC, so that each is measured in tandem rather than in silos. In simple terms, security defects are treated and addressed in the same way as software defects (e.g. security is a subset of core functionality rather than as an adjunct).



## Better security enablement of Dev, Sec and Ops teams

- ◆ By tracking Vulnerability Prevalence by Class and Remediation Rates by Risk, security managers and development managers in these organizations are able to devise better enablement strategies to remediate frequently occurring security flaws (e.g. Cross Site Scripting errors).

# Recommendations

- ◆ **Implement a three-phased DevSecOps approach.** In addition to increasing remediation rates and shrinking vulnerability windows, it can also facilitate communication, collaboration and consensus among IT security, DevOps, and IT Operations.
  1. As this report illustrates, the biggest challenge facing organizations is dealing with an increasingly huge volume of AppSec findings and remediation tasks. By discovering, categorizing, and prioritizing the biggest risks first, through DAST risk discovery, teams have a strategic, targeted plan to address the most vulnerable apps in production.
  2. Following up with SAST and SCA testing and remediation as well as a second DAST cycle helps close the loop.
  3. Prioritize developer enablement to get maximum remediation benefits for vulnerabilities discovered by DAST, SAST, and SCA.
- ◆ **Use DAST and SAST testing results for risk assurance, developer enablement and to drive consensus for an updated application security funding model.** Bringing distinct departments together for a common purpose supports the kind of consensus-building necessary to update the budget process for application security investments. Until and unless remediation costs are included in the risk assessment budget calculus, remediation rates will continue to fall. The bottom line is: finding defects earlier in the SLC when it's cheaper to fix them helps everyone.
- ◆ **Incorporate SCA into your DevSecOps program.** Since more than one third of all AppSec risks lie within 'reused' code (aka unpatched libraries), it's essential to conduct SCA testing as part of the risk assurance process. It's the only way to find and fix inherited flaws (developer enablement and SAST don't make much sense when it's 'not my code'). Additionally, with the rise of microservices apps, APIs, and the popularity of code sharing platforms like GitHub, we expect this trend to continue... which makes SCA a fundamental part of any AppSec program.
- ◆ **Ensure balanced investment across Development, Security and Operations for Application Security.** End-to-end AppSec requires adequate funding for all functions involved in the production of applications from development to security to operations. Use some or all of the the metrics outlined in the report to create goals and SLAs for development, security and operations teams. With those pan-organizational goals for application security, develop an investment plan that allows security to test applications and DevOps/TechOps to get trained, mitigate/ remediate vulnerabilities and incorporate security testing throughout the Software Lifecycle.



# Innovation in Application Security

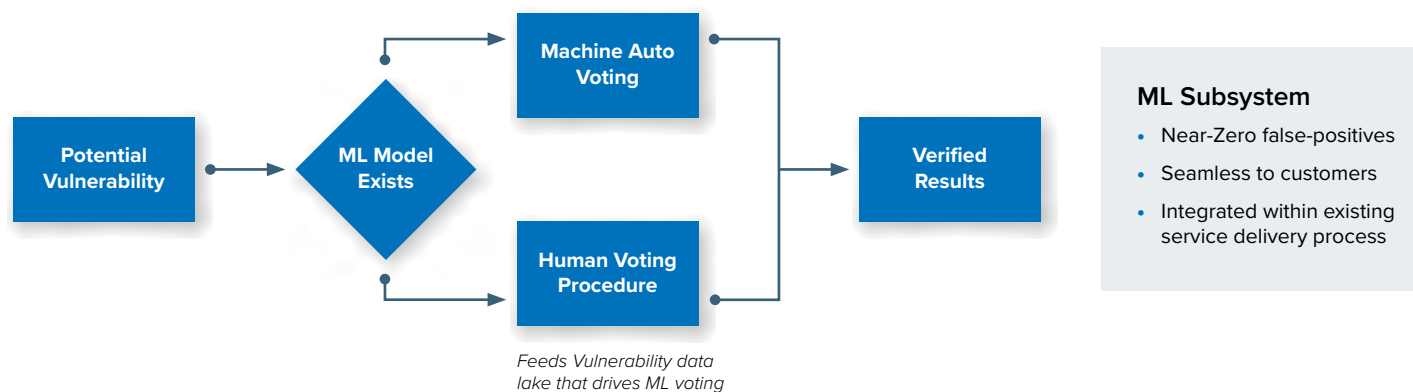
With Artificial Intelligence & Machine Learning becoming industrially viable, there is hope that **accuracy**, **speed** and **guidance** no longer need to be mutually exclusive.

At WhiteHat, we leverage over 150 TB of security data that corresponds to over 100 million attack vectors to drive a very important facet of application security – accuracy of results through vulnerability verification. In doing so, WhiteHat Security engineers and researchers are now actively developing, training and testing machine learning models that enable increasingly automated vulnerability verification. This automated vulnerability verification allows engineers and researchers the time to develop additional security tests and spend an increasing amount of time in security research that benefits our customers.

The machine learning subsystem is tightly integrated into the WhiteHat Application Security Platform as well as the Threat Research & Operations Center's service delivery process. This allows the Threat Research & Operations Center to govern and evolve the system without disrupting the customer experience. In addition, a subset of vulnerabilities always go through human verification for the following reasons:

1. To ensure that vulnerabilities that can't be automatically verified by the machine learning subsystem are verified by humans
2. To add new human curated vulnerabilities to the 150+ TB attack vector data lake for future machine learning endeavors
3. For performing quality control on a sample of the automatically verified vulnerabilities and provide a feedback loop to fine-tune machine learning models as needed

**At a high level, here is how the machine learning subsystem works in production:**



By analyzing massive amounts of application security data, and by keeping human curation at its core, WhiteHat has created an approach to AppSec that brings together accuracy, speed and guidance for our customers.

# Appendix - Glossary of Terms

## Web Application Vulnerability Classes

### ABUSE OF FUNCTIONALITY

Abuse of Functionality is an attack technique that uses a website's own features and functionality to attack. It misuses an application's intended functionality to perform an undesirable outcome. These attacks can consume resources, circumvent access controls, or leak information. The potential and level of abuse will vary from site to site and application to application. This category of attacks is broad and includes situations where an application's features can be functioning properly but still be exploited.

### APPLICATION MISCONFIGURATION

Application Misconfiguration exploits configuration weaknesses found in applications. Many applications come with unsafe features enabled by default, such as debug and QA features. These features may provide a means for a hacker to bypass authentication methods and gain access to sensitive information, perhaps with elevated privileges.

### BRUTE FORCE ATTACK

Brute Force Attacks are used to determine an unknown value such as a password by using an automated process to try many possible values. The attack takes advantage of the fact that the entropy of the values is smaller than perceived. For example: while an 8-character alphanumeric password can have 2.8 trillion possible values, many people will select passwords from a much smaller subset consisting of common words and terms.

### BUFFER OVERFLOW

Buffer Overflow is a flaw that occurs when more data is written to a block of memory, or buffer, than the buffer is allocated to hold. Exploiting Buffer Overflow allows an attacker to modify portions of the target process address space.

### CONTENT SPOOFING

Content Spoofing is an attack technique that allows an attacker to inject a malicious payload that is later misrepresented as legitimate content of an application. This attack compromises the trust relationship between the user and the application.

### CREDENTIAL / SESSION PREDICTION

Credential or Session Prediction is a method of hijacking or impersonating an authorized application user by deducing or guessing the unique value that identifies a particular session or user, which can allow attackers to issue site requests with the compromised user's privileges.

### CROSS SITE REQUEST FORGERY

Cross-Site Request Forgery is an attack that involves tricking a victim into sending an HTTP request to a target destination without the victim's awareness, so that the attacker can perform an action as the victim. CSRF exploits the trust that an application has for a user.

### CROSS-SITE SCRIPTING

In Cross-Site Scripting attacks, a malicious site includes a particular URL – one that will cause the target site to include a script chosen by the malicious site – in a target site's page, and makes the user agent request it. Since the page is loaded with the user agent's credentials, the script is able to perform actions at the target site in the user's name.

### CRYPTOGRAPHY: INSECURE DIGEST

Insecure Digest refers to an application that utilizes an insecure cryptographic hashing algorithm. The potential consequences of using an insecure cryptographic are similar to using an insecure cryptographic algorithm: data theft or modification, account or system compromise, and loss of accountability – i.e., non-repudiation.

### DENIAL OF SERVICE

Denial of Service and Distributed Denial of Service (D/DoS) attacks attempt to prevent an application from serving normal user activity. DDoS attacks, which are normally applied to the network layer, are also possible at the application layer. These malicious attacks can succeed by starving a system of critical resources.

### DIRECTORY INDEXING

Directory Indexing exploits insecure indexing, threatening a site's data confidentiality. Site contents are indexed via a process that accesses files that are not supposed to be publicly accessible. Information is collected and stored by the indexing process and can later be retrieved by a determined attacker, typically, through a series of search engine queries. Directory Indexing has the potential to leak information about the existence of such files and their content.

### DIRECTORY TRAVERSAL

The Directory Traversal attack technique (aka Path Traversal) allows an attacker access to files, directories, and commands that potentially reside outside the root directory. An attacker may manipulate a URL in such a way that the application will execute or reveal the contents of arbitrary files anywhere on the server. Any device that exposes an HTTP-based interface is potentially vulnerable to Directory Traversal.

### FINGERPRINTING / FOOTPRINTING

Fingerprinting or Footprinting is often an attacker's first goal. They will accumulate as much information as possible including the target's platform, application software technology, backend database version, configuration, and possibly even their network architecture/topology. Based on this information, the attacker can develop an accurate attack scenario to exploit any vulnerability in the software type/version being utilized by the target.

### FORMAT STRING ATTACK

Format String Attacks alter the flow of an application by using string formatting library features to access other memory space. Vulnerabilities occur when user-supplied data is used directly as formatting string input for certain C/C++ functions (e.g. fprintf, printf, sprintf, setproctitle, syslog).

### HTTP REQUEST SMUGGLING

HTTP Request Smuggling abuses the discrepancy in parsing non-RFC compliant HTTP requests between two HTTP devices – typically a front-end proxy or HTTP-enabled firewall and a back-end server – to smuggle a request to the second device through the first device. This technique enables an attacker to send one set of requests to the second device while the first device interacts on a different set of requests. This facilitates several possible exploits, such as partial cache poisoning, bypassing firewall protection and XSS.

### HTTP REQUEST SPLITTING

HTTP Request Splitting forces the browser to send arbitrary HTTP requests. Once the victim's browser is forced to load the attacker's malicious HTML page, the attacker manipulates one of the browser's functions to send two HTTP requests instead of one.

### HTTP RESPONSE SMUGGLING

HTTP Response Smuggling uses an intermediary HTTP device that expects or allows a single response from the server to send two HTTP responses from a server to a client that expects or allows a single response from the server.

## HTTP RESPONSE SPLITTING

HTTP Response Splitting allows an attacker to manipulate the response received by a web browser. The attacker can send a single HTTP request that forces the web server to form an output stream which is then interpreted by the target as two HTTP responses instead of one, as is the normal case.

## IMPROPER FILESYSTEM PERMISSIONS

Improper Filesystem Permissions are a threat to the confidentiality, integrity and availability of an application. The problem arises when incorrect filesystem permissions are set on files, folders, and symbolic links. When improper permissions are set, an attacker may be able to access restricted files or directories and modify or delete their contents.

## IMPROPER INPUT HANDLING

Generally, the term input handling is used to describe functions like validation, sanitization, filtering, or encoding and/or decoding of input data. Improper Input Handling is a leading cause behind critical vulnerabilities that exist in systems and applications.

## IMPROPER OUTPUT HANDLING

Improper Output Handling is a weakness in data generation that allows the attacker to modify the data sent to the client.

## IMPROPER PSEUDO-RANDOM NUMBER GENERATOR

Insufficient randomness results when software generates predictable values when unpredictability is required. When a security mechanism relies on random, unpredictable values to restrict access to a sensitive resource, such as an initialization vector (IV), a seed for generating a cryptographic key, or a session ID, then use of insufficiently random numbers may allow an attacker to access the resource by guessing the value. The potential consequences of using insufficiently random numbers are data theft or modification, account or system compromise, and loss of accountability (i.e., non-repudiation).

## INFORMATION LEAKAGE

Information Leakage allows an application to reveal sensitive data, such as technical details of the application, environment, or user-specific data. Sensitive data may be used by an attacker to exploit the target application, its hosting network, or its users.

## INSECURE INDEXING

Information is collected and stored by the indexing process. Insecure Indexing allows this information to be retrieved by a determined attacker, typically through a series of queries to the search engine. The attacker does not thwart the security model of the search engine; therefore, this attack is subtle and very hard to detect. It's not easy to distinguish the attacker's queries from a legitimate user's queries.

## INSUFFICIENT ANTI-AUTOMATION

Insufficient Anti-Automation occurs when an application permits an attacker to automate a process that was originally designed to be performed only in a manual fashion, e.g. registration for a site.

## INSUFFICIENT AUTHENTICATION

Insufficient Authentication occurs when an application permits an attacker to access sensitive content or functionality without having to properly authenticate. For example, accessing admin controls by going to the /admin directory without having to log in.

## INSUFFICIENT AUTHORIZATION

Insufficient Authorization occurs when an application fails to prevent unauthorized disclosure of data or a user is allowed to perform functions in a manner inconsistent with the permission policy.

## INSUFFICIENT COOKIE ACCESS CONTROL

Insufficient Cookie Access Control occurs when cookie attributes such as "domain," "path" and "secure" are not correctly utilized to limit access to cookies containing sensitive information. These attributes can be used by the user-agent when determining cookie access rights.

## INSUFFICIENT CROSS-DOMAIN CONFIGURATION

The crossdomain.xml file is used to determine from which resources a Flash application is allowed to access data. Insufficient Cross-domain Configuration reflects a poorly configured Flash application that can be compromised to allow an attacker access to all the resources allowed in the cross-domain file. This error often occurs because a cross-domain file makes use of wild-card notation.

## INSUFFICIENT PASSWORD AGING

Insufficient Password Aging allows a user to maintain the same password for an extended length of time, increasing the risk of password-based attacks.

## INSUFFICIENT PASSWORD RECOVERY

Insufficient Password Recovery occurs when an application permits an attacker to obtain, change or recover another user's password without permission. This happens when the information required to validate a user's identity for recovery is either easily guessed or circumvented. Password recovery systems may be compromised through the use of brute force attacks, inherent system weaknesses, or easily guessed secret questions.

## INSUFFICIENT PASSWORD STRENGTH

Insufficient Password Strength exists when a password policy does not aid the user in selecting a password that is less vulnerable to brute force attacks.

## INSUFFICIENT PROCESS VALIDATION

Insufficient Process Validation occurs when an application fails to prevent an attacker from circumventing the intended flow or business logic of the application.

## INSUFFICIENT SESSION EXPIRATION

Insufficient Session Expiration occurs when an application permits an attacker to reuse old session credentials or session IDs for authorization. Insufficient Session Expiration exposes an application to attacks that steal or reuse a user's session identifiers.

## INSUFFICIENT SESSION INVALIDATION

A user should be able to invalidate a session simply by logging out. This error occurs when the application removes the session cookie but doesn't invalidate the session.

## INTEGER OVERFLOWS

Integer Overflow occurs when the result of an arithmetic operation such as multiplication or addition exceeds the maximum size of the integer type used to store it. Attackers can use these conditions to influence the value of variables in ways that the programmer did not intend.

## INVALID HTTP METHOD USAGE

HTTP Methods can be used inappropriately and compromise the integrity of the application. For example, the GET method is not intended to contain sensitive information or change the site state. Doing so increases vulnerability to Cross Site Request Forgery, Information Leakage, and accidental damage by crawlers.

## LDAP INJECTION

LDAP Injection uses the open standard Lightweight Directory Access Protocol (LDAP) to perform exploits similar to those used in SQL Injection.

## MAIL COMMAND INJECTION

Mail Command Injection is an attack technique used to exploit mail servers and webmail applications that construct IMAP/SMTP statements from user-supplied input that is not properly sanitized.

## NON-HTTP ONLY SESSION COOKIE

A session cookie value can be accessed and manipulated by malicious client-side Javascript. Setting the "HttpOnly" attribute instructs the User-Agent to restrict access to the cookie only for use with HTTP messages.

## NULL BYTE INJECTION

Null Byte Injection is an active exploitation technique used to bypass sanity checking filters in infrastructure by adding URL-encoded null byte characters (i.e. %00, or 0x00 in hex) to the user-supplied data.

## OS COMMAND INJECTION

OS Command Injection, aka OS Commanding, is an attack technique used for unauthorized execution of operating system commands.

## PATH TRAVERSAL

(see Directory Traversal)

## PERSISTENT SESSION COOKIE

This error occurs when cookies whose values contain sensitive data have a future expiration date and do not expire with the session.

## PERSONALLY IDENTIFIABLE INFORMATION

Personally Identifiable Information (PII) is information that identifies a single person or can be used with other information sources to identify a single person. Examples of PII include: name, age, birth date, birth place, credit card number, criminal record, driver's license number, education history, genotype, social security number, race, place of residence, vehicle identification number, and work history.

## PREDICTABLE RESOURCE LOCATION

Predictable Resource Location allows an attacker, by making educated guesses via brute forcing, to guess file and directory names not intended for public viewing. Brute forcing filenames is easy because files/paths often have common naming conventions and reside in standard locations. Predictable Resource Location is also known as Forced Browsing, Forceful Browsing, File Enumeration, or Directory Enumeration.

## REMOTE FILE INCLUSION

Remote File Inclusion (RFI) exploits dynamic file inclusion mechanisms in applications. When a user input specifies a file inclusion; the application can be tricked into including remote files with malicious code.

## ROUTING DETOUR

Routing Detour is a type of "man-in-the-middle" attack in which intermediaries can be injected or hijacked in order to route sensitive messages to an outside location in such a way that the receiving application is unaware that it has occurred.

## SERVER MISCONFIGURATION

Configuration weaknesses found in servers and application servers can trivially allow abuse of default functionality.

## SESSION FIXATION

Session Fixation is an attack that forces a user's session ID to a known value. After a user's session ID has been fixed, the attacker will wait for that user to login and use the predefined session ID value to assume the same online identity. Session Fixation provides a much wider window of opportunity than would be provided by stealing a user's session ID after they have logged into an application.

## SESSION / CREDENTIAL PREDICTION

Session or Credential Prediction (aka Session Hijacking) is a method of hijacking or impersonating an authorized application user by deducing or guessing the unique value that identifies a particular session or user. This can allow attackers to issue site requests with the compromised user's privileges.

## SOAP ARRAY ABUSE

In XML SOAP Array Abuse, a service that expects an array can become the target of an XML DoS attack by forcing the SOAP server to build a huge array in the machine's memory, thus inflicting a DoS condition on the machine due to the memory pre-allocation.

## SQL INJECTION

SQL Injection exploits applications that construct SQL statements from user-supplied input. When successful, the attacker is able to execute arbitrary SQL statements against the database.

## SSI INJECTION

Server-Side Include Injection is a server-side exploit that allows an attacker to send code to an application to be executed later, locally by the server. SSI Injection exploits an application's failure to sanitize user-supplied data before inserting the data into a server-side interpreted HTML file.

## UNPATCHED LIBRARY

All software components, runtime environments, platforms, and libraries need to be kept up to the very latest version of security fixes, to avoid exploits written specifically against known out-of-date library vulnerabilities.

## UNSECURED SESSION COOKIE

If a session cookie does not have the secure attribute enabled, it is not encrypted between the client and the server. This means the cookie is exposed to theft.

## URL REDIRECTOR ABUSE

URL redirectors can be abused to cause an attacker's URL to appear to be endorsed by the legitimate site, tricking victims into believing that they are navigating to a site other than the true destination. (See Content Spoofing)



## **WEAK CIPHER STRENGTH**

In the Weak Cipher Strength vulnerability, the application's server allows the use of weak SSL/TLS ciphers which are typically weaker than 128 bits and do not use signed certificates (e.g. SHA-1 hash).

## **WEAK PASSWORD RECOVERY VALIDATION**

An application permits an attacker to illegally obtain, change or recover another user's password because the information required to validate a user's identity for password recovery is either easily guessed or can be circumvented. Password recovery systems may be compromised through the use of brute force attacks, inherent system weaknesses, easily guessed or easily phished secret questions.

## **XML ATTRIBUTE BLOWUP**

XML Attribute Blowup takes advantage of some XML parsers' parsing process. The attacker provides a malicious XML document, which vulnerable XML parsers process inefficiently, resulting in severe CPU load. Many attributes are included in the same XML node, resulting in a denial of service condition.

## **XML ENTITY EXPANSION**

XML Entity Expansion exploits a capability of XML Document Type Definitions that allows the creation of custom macros called "entities." By recursively defining a set of custom entities at the top of a document, an attacker can overwhelm parsers that attempt to completely resolve these entities, resulting in a denial of service condition.

## **XML EXTERNAL ENTITIES**

An XML External Entities attack takes advantage of a feature of XML that allows you to build documents dynamically at the time of processing. It uses an XML message that can provide data explicitly or points to a URL where the data exists. In the attack external entities may replace the entity value with malicious data. Alternately, referrals may compromise the security of the data to which the server/XML application has access.

## **XML INJECTION**

XML Injection manipulates or compromises the logic of an XML application or service. The injection of unintended XML content and/or structures into an XML message can alter the intended logic of the application. Furthermore, XML Injection can cause the insertion of malicious content into the resulting message/document.

## **XPATH INJECTION**

XPath Injection exploits applications that construct XPath (XML Path Language) queries from user-supplied input to query or navigate XML documents.

## **XQUERY INJECTION**

XQuery Injection is a variant of the classic SQL injection attack against the XML XQuery Language. XQuery Injection uses improperly validated data that is passed to XQuery commands.



### **About WhiteHat Security**

WhiteHat Security has honed its 18 years of experience in the application security space to provide developers and businesses with the tools and services they need to write and deliver the most secure software at the speed of business. The award-winning WhiteHat Application Security Platform, which has been featured on the Gartner Magic Quadrant for Application Security Testing for the last five years, is empowering DevSecOps by continuously assessing the risk for organizations' software assets and helping them to embed security throughout the software life cycle (SLC). The company is an independent, wholly-owned subsidiary of NTT Security and is based in San Jose, California, with regional offices across the U.S. and Europe.