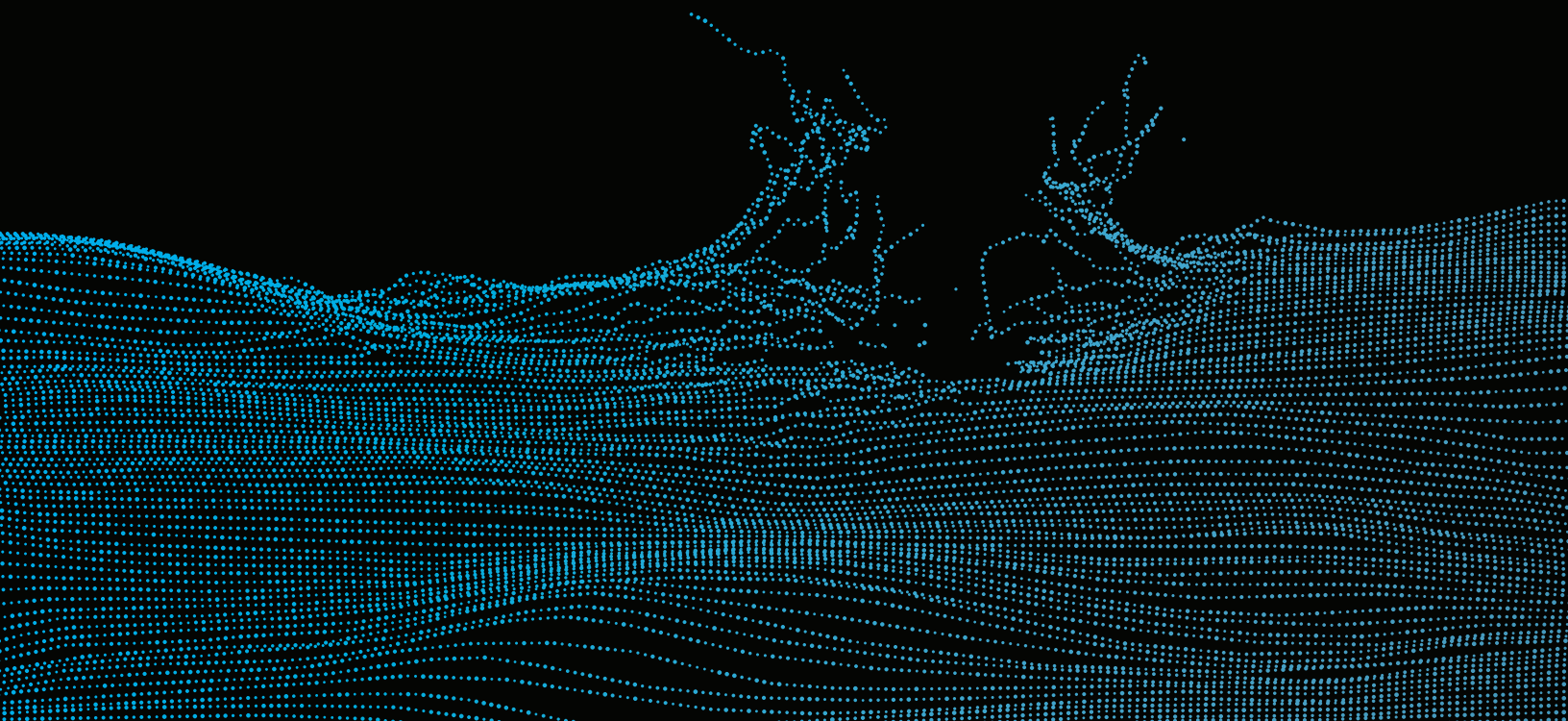
WhiteHat SECURITY

2017

APPLICATION SECURITY STATISTICS REPORT

The Case for DevSecOps

VOL.12



FOREWORD

This is year 12 of the WhiteHat Security Application Security Statistics Report, and for the first time in its history (and maybe all history) we are providing some real metrics around DevSecOps. ***Does taking this approach really make a difference when it comes to improving application security?*** We've also added a new SAST section, and – thanks to our partner NowSecure – a mobile application security section.

The idea of DevSecOps has captured the imaginations of many of us working in security. Making security part of a developer's processes and workflow makes sense to a security practitioner, but it represents a greater challenge to a developer who doesn't have a background in security. Compounding the problem is the fact that developers are being asked to release apps faster and faster all the time. Like it or not, it's becoming imperative that security "move left in the SDLC" to become as much a part of the developer's coding process as any other aspect of software development. This is because applications have become the driving force of the digital business yet they remain the most vulnerable component. The Verizon 2017 Data Breach Investigations Report found that "almost 60% of breaches involved web applications either as the asset affected, and/or a vector to the affected asset."

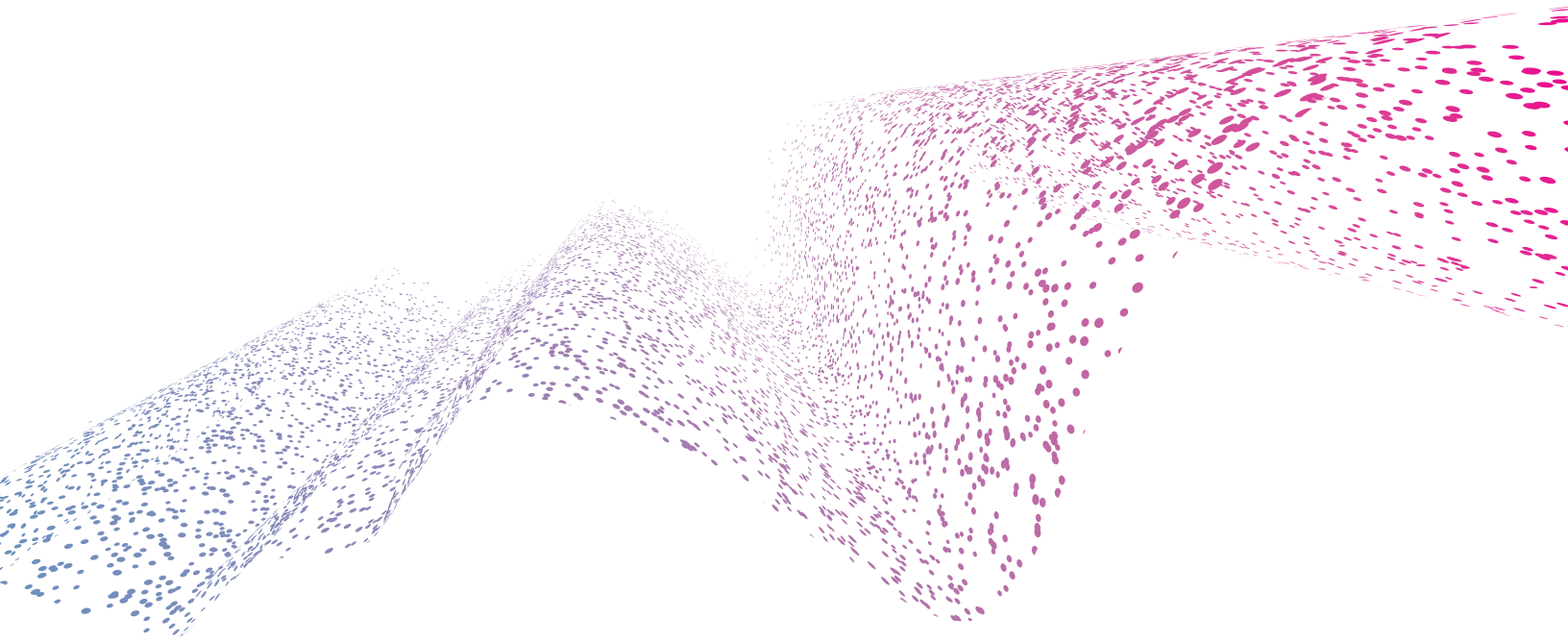
The great news for developers is that we've uncovered some compelling evidence that the security effort is worth it. In the section of this report titled "Case Study: Making the Case for DevSecOps", we've profiled a WhiteHat customer that implemented a program for creating "Security Heroes" in the development organization, putting the training and infrastructure in place necessary to support secure coding in an agile DevOps environment. The results of this effort were impressive: critical vulnerabilities in applications in development and in production were resolved in a fraction of the time that it takes organizations that haven't engaged DevOps teams in the security effort.

What you'll find as you read this report is that there are still too many vulnerabilities in applications and it's still taking too long to fix them. But the DevSecOps phenomenon offers light at the end of the tunnel, and we're starting to see real evidence of the value of security and development working together to protect the applications that we rely on every day in our personal and professional lives.

Ryan O'Leary

Vice President,
WhiteHat Security Threat Research Center
and Technical Support

TABLE OF CONTENTS



WHY READ THIS REPORT

Software sits in the center of digital business

For many organizations today, applications ARE the business. New code is developed and released faster than ever before using agile, rapidly iterative methods. New applications launch every day.

Yet the practice of application security is a constantly moving target.

Every day it seems, new hacks and attacks are discovered that exploit weaknesses in the software that runs your business. According to the 2017 Verizon Data Breach Investigations Report, 30 percent of data breaches target the application layer. Even as digital transformation requires that software be built faster, application security is required to reduce your organization's overall business risk.

This is the largest and most accurate report of its kind available focused on application security.

It imparts the latest real-world security intelligence, providing one-of-a-kind breadth and depth across dynamic, static and mobile security testing – including a case study proving the benefits of a DevSecOps approach.

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, this report is essential reading.

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 compared with industry benchmarks.

FOR APPLICATION DEVELOPMENT TEAMS...

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

INTRODUCTION

The Application Security Statistics Report is an annual study.

For the past 11 years, this report has provided perspective on the present application security posture of hundreds of organizations. It has simply become the largest and most accurate report of its kind available focused on application security. Unlike a survey, the data highlighted in this report comes directly from actual code-level analysis of billions of lines of code and tens of thousands of websites and applications.

The objective is to provide timely insights into how any organization can measurably improve its application security program.

Insecure software applications have become a critical part of assessing total business risk. All organizations must actively manage the security posture of their business software portfolio and greater software supply chain. Information security teams must work to mitigate the risks from vulnerable applications – for the benefit of business users, consumers, or both. Everyone involved in securing the software that runs the organization should keep this true equation in mind:

2017 findings include analysis of multiple software testing methods – dynamic (DAST), static (SAST), and mobile app security.

This volume captures data collected in the twelve months of 2016 analyzing 15,000 web applications, billions of lines of code, and more than 65,600 mobile apps. Its findings comprise analysis of DAST results, and new this year: SAST results, DAST/SAST applied in combination, and mobile app security data provided by WhiteHat partner NowSecure. Software vulnerabilities are examined by criteria such as type, class, time to remediate, industry, severity and other criteria. This analysis therefore reports the latest real-world security intelligence, providing one-of-a-kind breadth and depth.

It is hoped that the findings from this important study can help any organization to identify and scale its application security approach with insight into issues that all organizations must address. As the only industry leading, pure-play application security platform provider, WhiteHat Security continues to work tirelessly to improve the security posture of our customer organizations, and by extension the entire software industry. We welcome any questions or comments from readers as we continually strive to improve and enrich the coverage, quality and detail of this report.

EXECUTIVE SUMMARY

The size of the application security problem continues to grow.

Software sits at the center of digital business. For many organizations, their applications simply ARE the business. Websites and applications drive great digital experiences, but only secure applications can deliver them safely. In industry after industry, organizations rely on hundreds if not thousands of applications – scaling the job of application security exponentially.

The latest edition of the Verizon Data Breach Investigations Report highlights the statistics in the side bar that are important to understand in light of the state of application security.

The pace of digital transformation has been breathtaking, creating non-stop demand for rapid business innovation. This means software applications are being developed and released faster than ever before using agile, rapidly iterative methods. No longer written from scratch, today's mobile app or embedded device software is instead "assembled" from interconnected APIs, open source components, and cloud delivery methods such as containers and micro-services.

Meanwhile, new hacks and attacks are launched every day that exploit hidden weaknesses in new software. This means new vulnerabilities are being exposed and exploited faster, at a pace that many organizations simply cannot match. The practice of application security is therefore required to reduce overall business risk. Yet many organizations lack the tools, talent and expertise required to succeed.

According to Gartner, enterprises spent $719M on software security in 2016, up from $630M in 2015.[1] That's good news; however relative to other cybersecurity methods, application security still isn't enjoying nearly enough investment.

of total breaches reported...

30%
involved attacks on web applications

62%
featured hacking to exploit vulnerabilities

81%
hacking related breaches that leveraged weak or stolen passwords

of attacks on web applications...

93%
financially motivated, perpetrated by organized criminal groups

77%
carried out by botnets, not individuals

32%
exploited SQL Injection errors

SOURCE: VERIZON 2017

[1] *Gartner, Inc., "Forecast: Information Security, Worldwide, 2015-2021, 1Q17 Update", Ruggero Contu, Christian Canales, Sid Deshpande and Lawrence Pingree, 18 May 2017. Constant dollar-Tables.*

The 5 Major Findings on Application Security Statistics in 2017

FINDING.1

THE APPLICATION SECURITY POSTURE OF THE AVERAGE ORGANIZATION HAS IMPROVED, BUT ONLY MARGINALLY.

In 2015, the average web application had four vulnerabilities; in 2016 that number dropped to three. That's an improvement of 25 percent, and overall a sign that many organizations are starting to mature their application security programs. One factor that could explain this marginal improvement is greater integration of testing tools with the software development lifecycle (SDLC). Yet progress remains too slow. The multiple of total vulnerabilities in any single organization's software portfolio remains staggering. Average time-to-fix is still too high, and remediation rates too low. Overall, further process improvement is needed.

FINDING.2

USE OF BOTH SAST AND DAST TESTING IN TANDEM IS ESSENTIAL FOR APPLICATION SECURITY PROGRAM EFFECTIVENESS.

Certain code vulnerabilities take a shorter amount of time to fix and are easier to remediate during development, when static testing (SAST) is best employed. Other errors show up only in dynamic testing (DAST) of applications once in production. However certain vulnerabilities such as SQL injection and Cross-Site Scripting are likely to be found by both testing regimens – these are the errors most critical to address. Every vulnerability fixed during development will improve the security posture of the production application. That said, SAST errors are still taking an average of 113 days to remediate, which is not nearly fast enough.

FINDING.3

ORGANIZATIONS MUST TAKE A RISK-BASED APPROACH TO REMEDIATING APPLICATION SECURITY FLAWS.

Remediation priorities need to be set based on the criticality of the software errors found. Vulnerability ratings depend on many factors, and an error that is considered critical by one organization may be considered medium-risk by another. Software developers need more education by security teams to understand the risk levels of different vulnerability types. Remediation is too often being prioritized by path of least resistance (i.e. the easiest ones are the first to be fixed). Most organizations still need to adopt risk-based remediation processes.

FINDING.4

APPLICATION SECURITY TESTING METHODS ARE MORE EFFECTIVE WHEN INTEGRATED WITH THE SDLC.

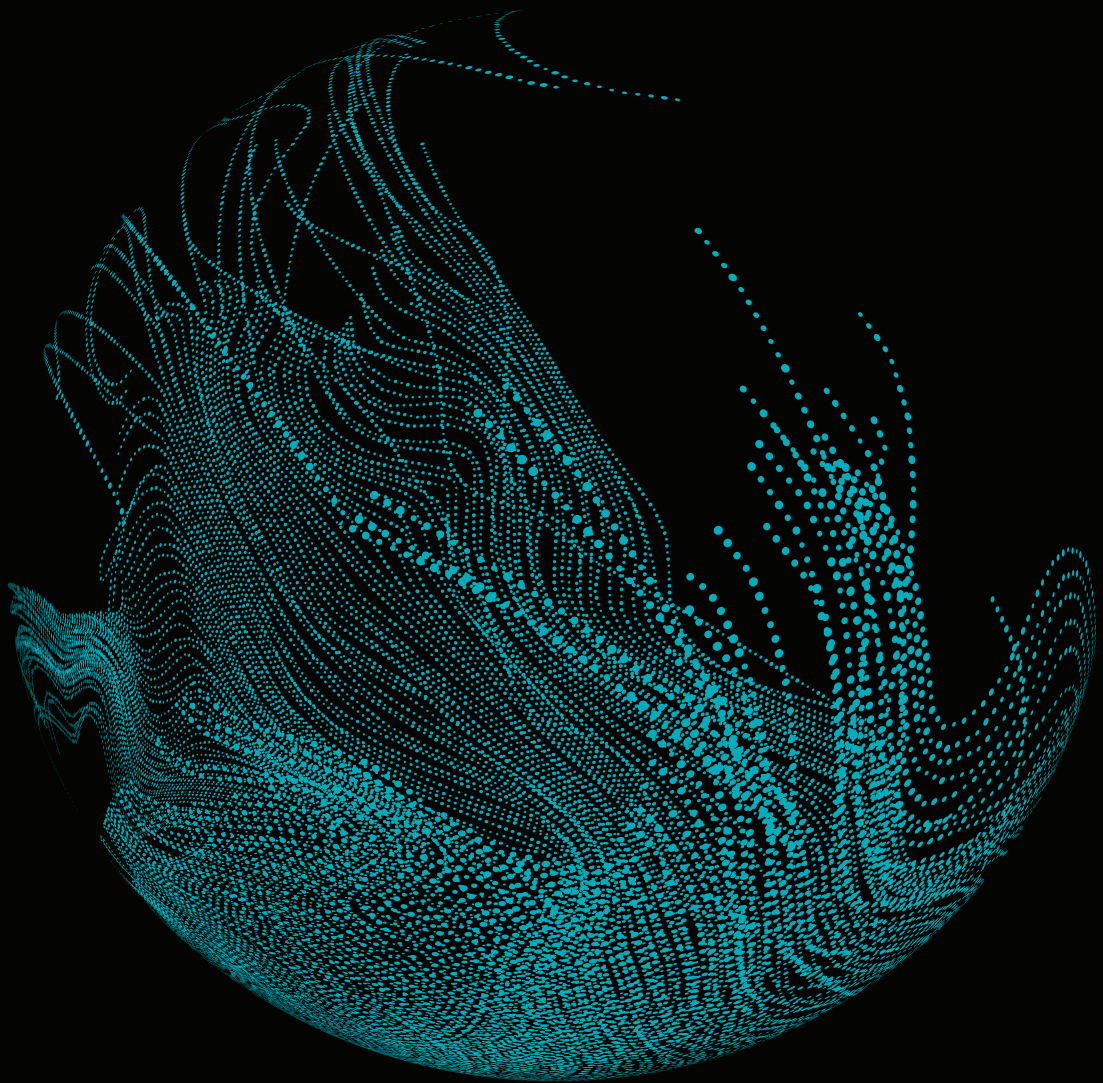
All organizations must implement application security procedures earlier in their SDLC. It has been proven faster and less costly to catch and fix flaws earlier rather than later. Routine security testing in development makes resulting production applications stronger. Organizations that integrate multiple kinds of testing regimens (e.g. DAST, SAST, mobile, etc.) directly with their SDLC see the best results. Today's application security platforms extend visibility and control even further with Software Composition Analysis, API testing, training and other services.

FINDING.5

ADOPTION OF DEVSECOPS IS IMPERATIVE FOR APPLICATION SECURITY TO DELIVER COMPETITIVE ADVANTAGE.

Organizations adopting DevOps practices must extend them to application security practices as well. The time is now. Many enterprises are starting to make DevSecOps real. Real-world examples exist of working DevSecOps programs that prove its advantages (see Case Study, pg. 42). Positive collaboration, not punitive, between the security team and application developers is the best approach to achieve an application security Center of Excellence.

DYNAMIC APPLICATION SECURITY TESTING | DAST



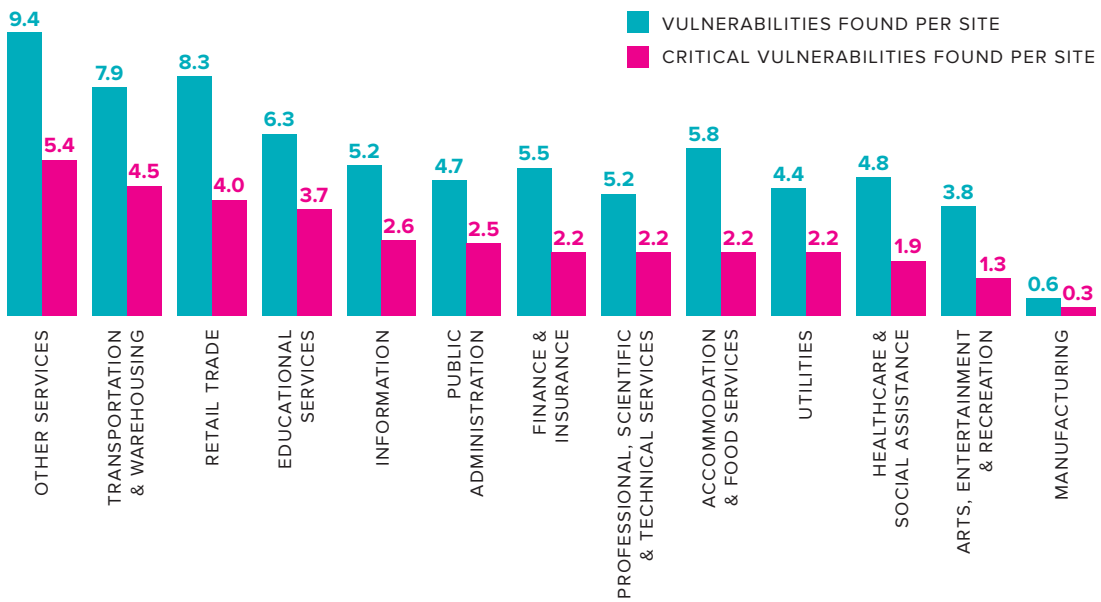
VULNERABILITIES BY INDUSTRY

Despite growing security awareness, applications continue to remain vulnerable across all industries.

Application vulnerabilities continue to be a significant problem; however there has been marginal improvement across the board. In 2015, web applications analyzed had an average of four vulnerabilities. That number dropped to three vulnerabilities in 2016. While this represents a 25 percent improvement year-over-year, most applications have three or more vulnerabilities, with almost half of them being "critical". These errors could result in data loss, theft or denial of service attacks if not properly remediated.

As *Figure 1* indicates, the service industry suffers the highest number of vulnerabilities, both critical and non-critical, followed by the transportation sector. Among regulated industries, Retail has one of the highest serious vulnerability ratios at 33 percent. By comparison, Finance and Healthcare each have less than 28 percent "serious" vulnerabilities (the combination of "critical" and "high risk" classifications). These better numbers may reflect an increased level of investment in cybersecurity by those two industries. On the other hand, even with PCI compliance imposing a regulatory mandate for better application security, the Retail industry continues to be plagued with insecure software.

FIGURE 1. VULNERABILITY PROFILE BY INDUSTRY - DAST



WINDOW OF EXPOSURE

Close to 50 percent of applications remain vulnerable on every single day of the year.

Window of exposure is defined as the number of days an application has one or more serious vulnerabilities open during a given time period. Window of exposure is categorized as:

ALWAYS VULNERABLE
An application falls in this category if it is vulnerable on every single day of the year.

FREQUENTLY VULNERABLE
An application is called frequently vulnerable if it is vulnerable for 271-364 days a year.

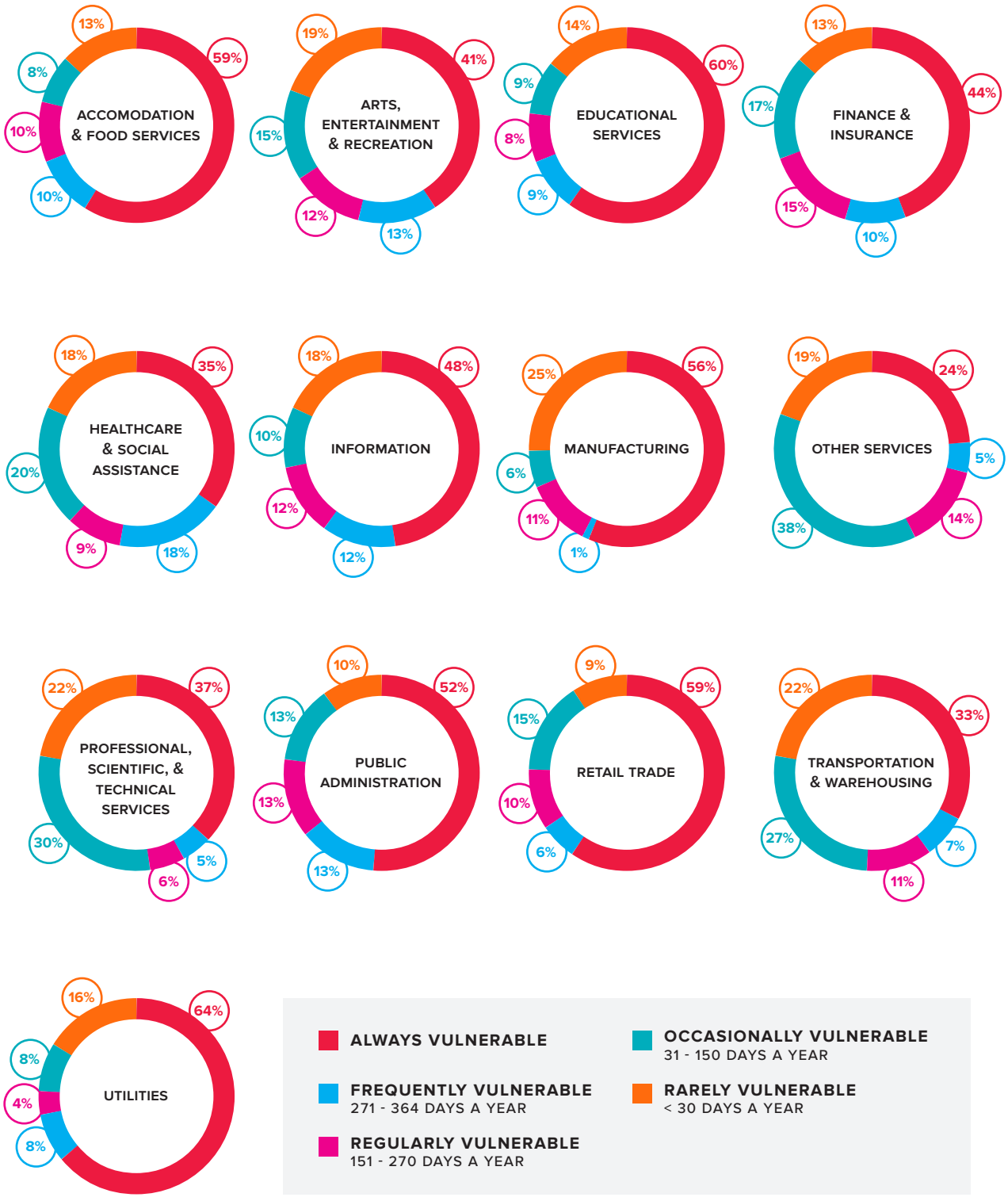
REGULARLY VULNERABLE
A regularly vulnerable application is vulnerable for 151-270 days a year.

OCCASIONALLY VULNERABLE
An occasionally vulnerable application is vulnerable for 31-150 days a year.

RARELY VULNERABLE
A rarely vulnerable application is vulnerable for less than 30 days a year.

Figure 2 shows that in five key industries – Utilities, Education, Accommodations, Retail and Manufacturing – around 60 percent of their web applications remain "always vulnerable". In all but one industry, at least one-third of web applications are always vulnerable. This implies that organizations are not able to resolve all of the "serious" vulnerabilities found in their applications, and it takes them a long time to remediate serious vulnerabilities.

FIGURE 2. WINDOW OF EXPOSURE - DAST



VULNERABILITY LIKELIHOOD BY CLASS

The top 4 most likely DAST vulnerabilities and how to fix them

Vulnerabilities fall into different categories, or "classes", that have unique attributes. In *Figure 3*, the percent likelihood shown reflects how likely it is that a production application will have a specific class of vulnerability. This is calculated based on the number of applications that have at least one open vulnerability in a given class, compared to the total number of production applications tested. There is wide variation in likelihood of various types of vulnerability classes.

The top four most likely DAST vulnerabilities are:

Information Leakage is the most prevalent vulnerability with 37 percent likelihood. This problem is very common, with one out of every two applications having some type of information leakage vulnerability. It is a broad classification that can be as serious as leaking usernames and passwords or as benign as "leaking" software version numbers. The good news is that it's usually an easy reconfiguration fix, but remediation often depends on the kind of data that is leaking – sensitive leaks are addressed, others are not.

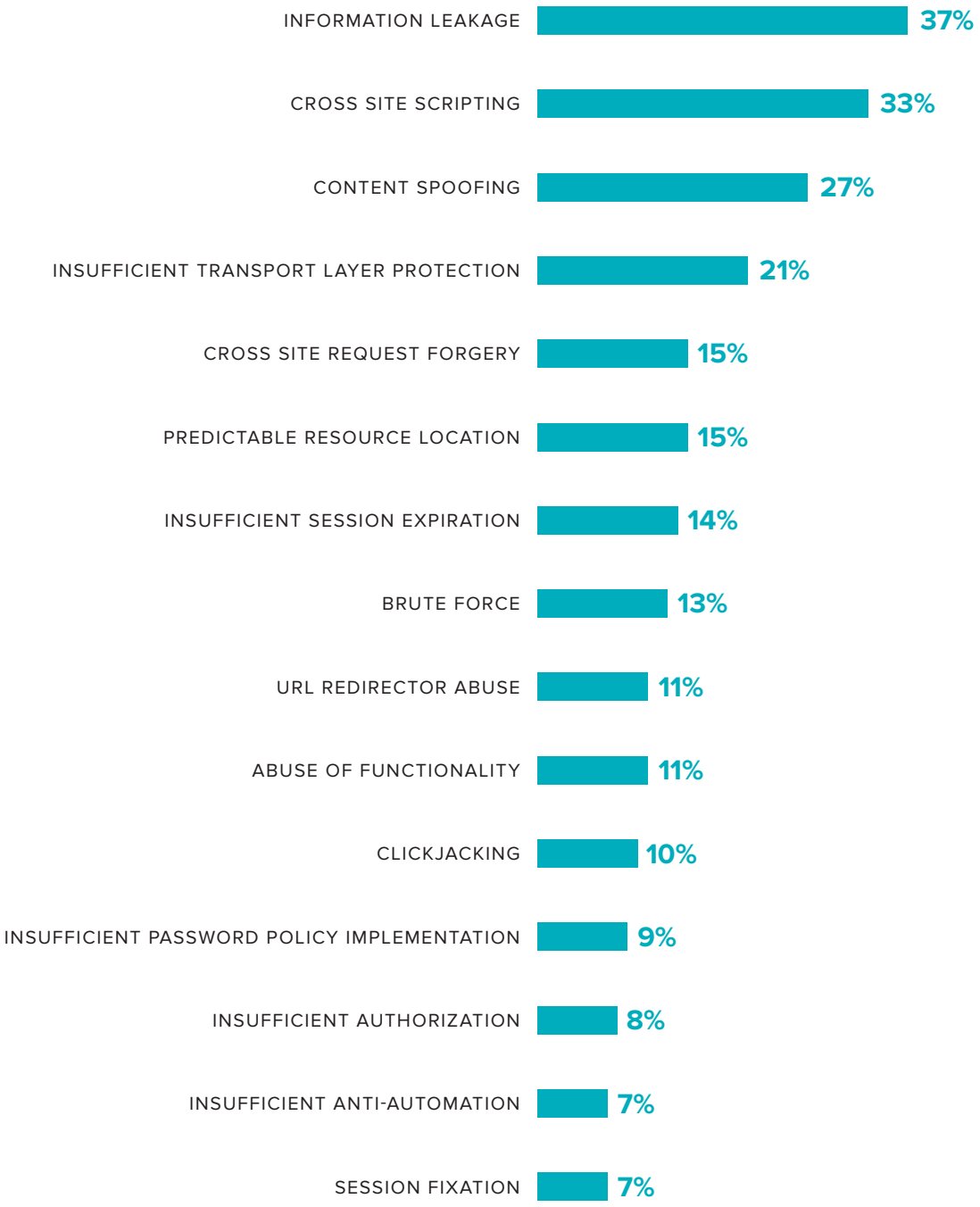
Cross-Site Scripting (33%) is an injection attack that targets the client-side of an application to execute some malicious script via the end user's browser. Because production applications are not static, they take input and data from users all the time. However, this user input/data must be properly encoded and sanitized, not merely executed, depending on the specific case. As browsers will interpret and execute script code differently, developers must know how and when to apply the proper sanitization solution.

Content Spoofing (27%) is an injection attack that allows an attacker to control the content displayed to the end user. It could deface the web page, mimic the authentication screen to steal user passwords, or display an error message to compel some other action. Developers must always have their guard up when taking text from an input request to ensure data is properly displayed as data, not content.

Insufficient Transport Layer Protection (21%) is a class used to describe errors such as weak ciphers, certificate misconfiguration or known vulnerable protocols. In recent years it has been exploited in zero day attacks such as Poodle, Shellshock and HeartBleed – making it a critical but tricky error to fix. Developers must properly maintain protocols, ciphers and certificates in live applications to keep them safe and up to date – while balancing security and usability.

To learn more about any of these vulnerabilities, see Appendix B.

FIGURE 3. VULNERABILITY LIKELIHOOD BY CLASS - DAST



MOST SERIOUS VULNERABILITIES BY CLASS

SQL Injection and Cross-Site Scripting remain the most critical vulnerabilities.

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 ratings depend on many factors, and an error that is considered critical by one organization may be considered medium-risk by another.

However, there are four classes of vulnerabilities found to be critical in the most cases across the thousands of production applications analyzed.

SQL Injection (SQLi) is the most critical by far, at fully 94 percent of vulnerabilities. Both static and dynamic security testing detect it equally well. SQLi exploits the database layer through rogue code injection techniques. Unauthorized access to backend databases can occur because user-supplied input is entered as part of a SQL command or SQL query without proper validation or encoding controls. These commands are then sent to the database server where they are executed. SQLi allows an unauthorized third-party to create, read, update, alter or delete sensitive data accessed via the unprotected application. SQLi attacks are easily preventable, highlighting the importance of remediating these quickly.

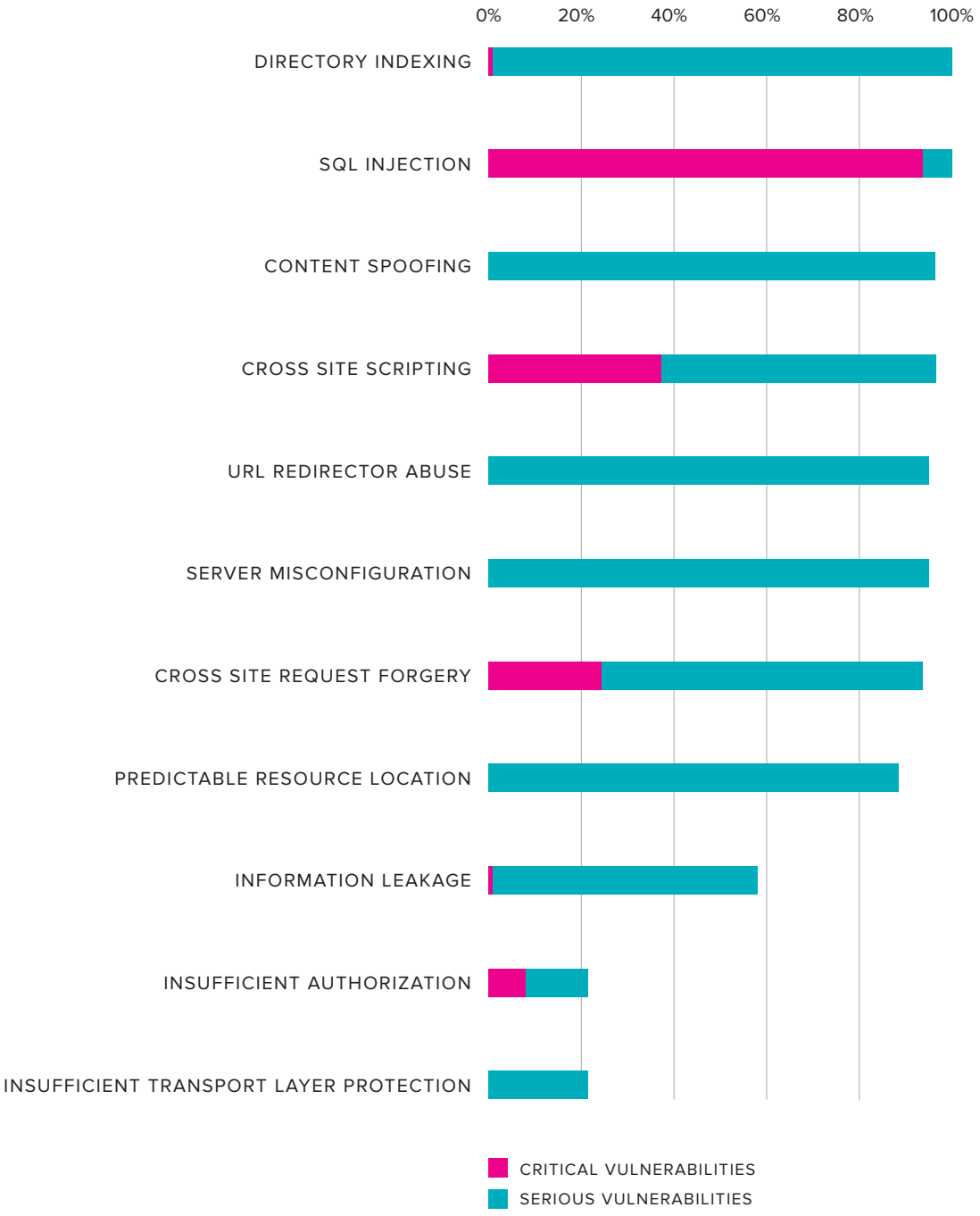
Cross-Site Scripting (XSS) is an equally likely and critical vulnerability, with 38 percent criticality in DAST. XSS is important to remediate because the user is the victim in these attacks, not the application. This kind of attack can execute every time the page is loaded into a browser, or whenever an associated action is performed by the user. Potential outcomes include hijacking the session, stealing account credentials, displaying unwanted advertising, or virus/malware infections. A successful XSS attack results in an attacker controlling the victim's browser or online account in the vulnerable application – with or without their knowledge.

Cross-Site Request Forgery (XSRF) is a critical error in 25 percent of production applications. An attacker manipulates a victim's browser into sending a HTTP request to take some action on the behalf of the user – usually a known, repeatable request. The user's session cookie helps with both tracking and authorization, so it is hijacked to make the attacker's request appear legitimate.

Insufficient Authorization is not as prevalent as other serious vulnerabilities, but critical 40 percent of the time it is present. This error occurs when an application fails to prevent unauthorized disclosure of data, or when a user is allowed to perform functions in a manner inconsistent with the permission policy.

Information Leakage is never critical, but is the most likely. Compare *Figure 4* to *Figure 3*. This vulnerability is why data breaches happen! Organizations should understand which data is leaking out of their applications, and plug the holes that matter.

FIGURE 4. MOST SERIOUS VULNERABILITIES BY CLASS - DAST



REMEDIATION RATES BY RISK LEVEL

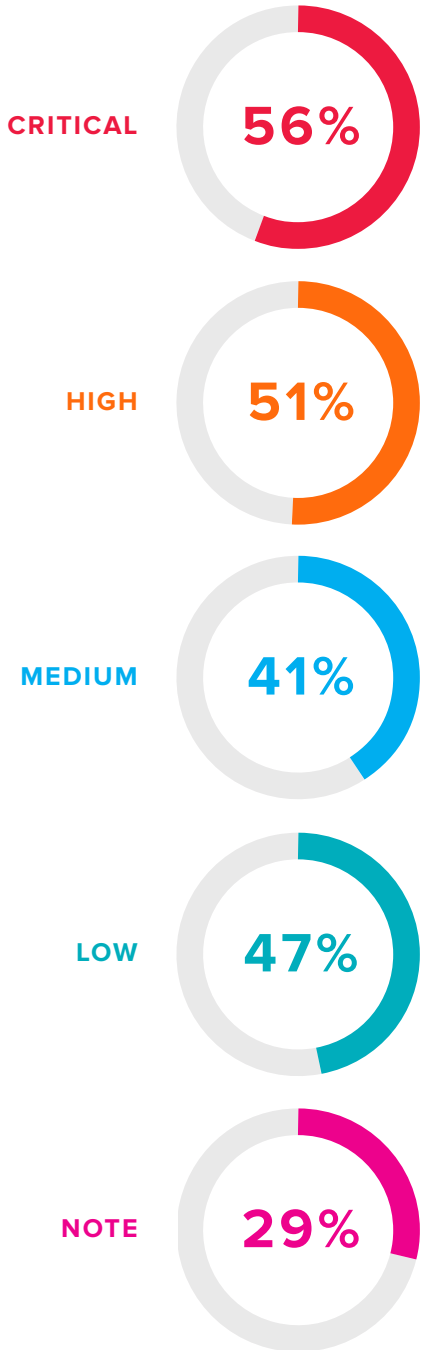
Easy fixes are being prioritized over more serious flaws.

Remediation rates are along the lines expected – where critical vulnerabilities enjoy the highest remediation rate, followed by high-risk ones.

What is interesting is that remediation rates for low-risk and medium-risk vulnerabilities are transposed. Why do less serious errors get fixed first? The answer isn't that surprising...

Development teams are prioritizing critical software problems first, but then move on to easier fixes. This is human nature. After patching tricky vulnerabilities, why not knock out a few simple ones?

FIGURE 5. REMEDIATION RATES BY RISK LEVEL - DAST



REMEDIATION RATES BY VULNERABILITY CLASS

Cross-Site Scripting is not being addressed, despite being an equally likely and critical vulnerability.

Major classes of application vulnerabilities enjoy varying rates of remediation. This becomes clear when the data is sorted by number, or prevalence of each class as in *Figure 6*.

SQL Injection enjoys the highest remediation rate at 60 percent. SQLi is a critical error, so this finding is good to see. Fix rates have been rising in recent years, up 8 percent from 2015.

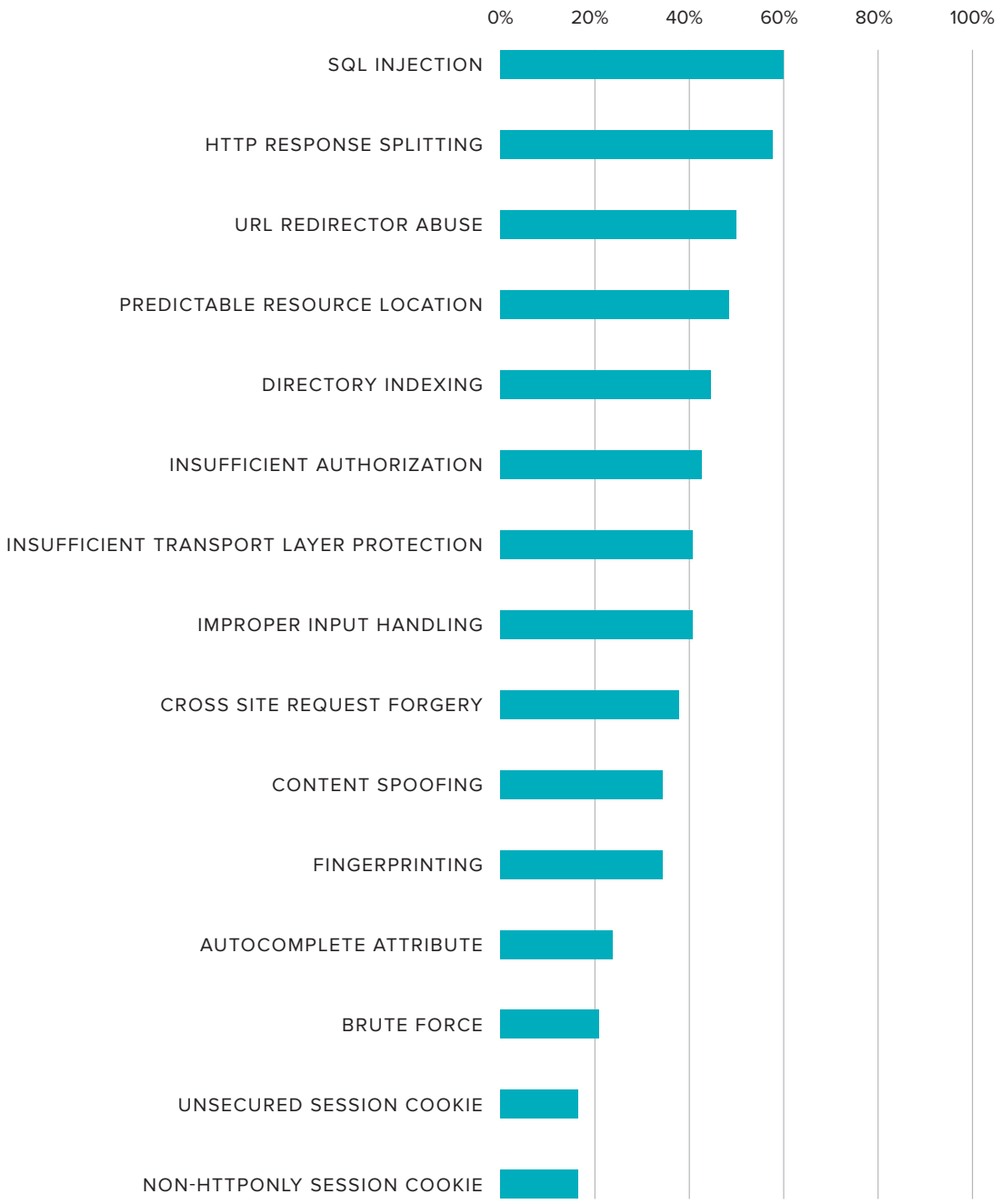
HTTP Response Splitting is a close second. This error allows an attacker to manipulate the response received by a web browser. The attacker can send a single HTTP request that forces the web server to form an output stream which is then interpreted by the target as two HTTP responses instead of one,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

URL Redirector Abuse is very similar to Content Spoofing, and both are serious vulnerabilities greater than 95 percent of the time. Victims are tricked into navigating to an attacker's site other than their true destination.

Sadly, all other vulnerabilities are remediated less than half of the time.

Perhaps the most distressing finding is the fact that Cross-Site Scripting suffers a very low remediation rate. This error is always serious, critical 40 percent of the time, yet it is completely off this chart. In past years of this study, XSS enjoyed a remediation rate between 50-55 percent, so developers simply chose not to deal with it in 2016 – a dangerous oversight. Fixing XSS involves locking down the process of passing necessary data between the application and the user's browser. Three remediation options include validating/sanitizing data input from the user's browser, encoding all output to the user's browser, or giving users the option to disable client-side scripts entirely.

FIGURE 6. REMEDIATION RATES BY VULNERABILITY CLASS - DAST



TIME TO FIX BY RISK LEVEL

Organizations are not using risk levels to prioritize vulnerability remediation.

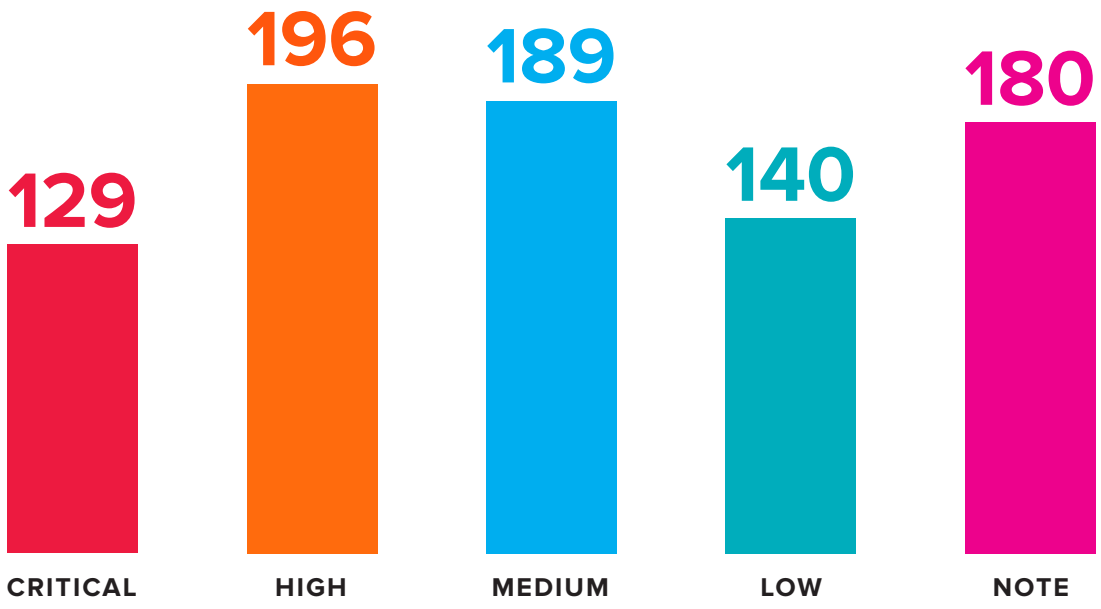
Critical vulnerabilities were fixed quickest in 2016, within an average of 129 days. This reflects an improvement, down from 146 days in 2015. SQLi errors have been the beneficiary of this trend.

However, high risk vulnerabilities suffer the highest time-to-fix (TTF) – a shocking 196 days on average. Ideally these vulnerabilities should be fixed the fastest after the critical errors, and yet high risk TTF is instead getting worse. The number of days rose from 171 on average in 2015. The fact that these serious software errors are fixed very late reflects the need for better awareness and sharper prioritization.

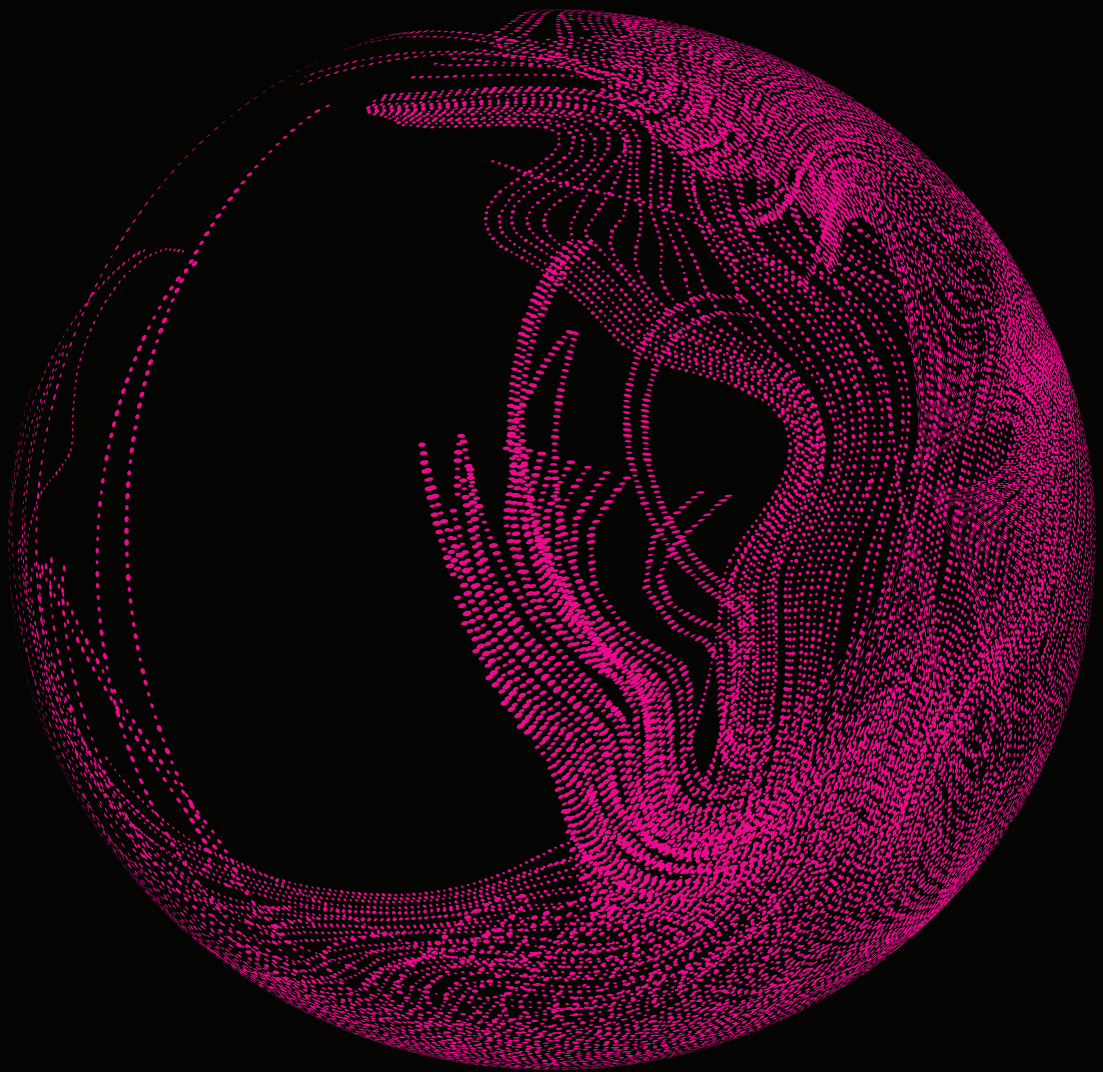
In the two-year period 2015 to 2016, the average TTF for high and medium-risk vulnerabilities increased, while TTF for low risk vulnerabilities remained steady.

Compare *Figure 7* to *Figure 5*, where easy fixes are being remediated quickly and often. This leads to the conclusion that security organizations are not using risk levels to set remediation priorities, with the exception of “must fix” problems.

FIGURE 7. AVERAGE TIME TO FIX BY RISK LEVEL IN DAYS - DAST



STATIC APPLICATION SECURITY TESTING | SAST



VULNERABILITY LIKELIHOOD BY CLASS

Use of open source and re-usable components in software development introduces and propagates new vulnerabilities.

The most common code vulnerability evident in static security testing during the software development process is **Unpatched Libraries**. Why? Because modern software is largely assembled of component parts, and everybody uses open source libraries today. These are readily available options, but often not very secure.

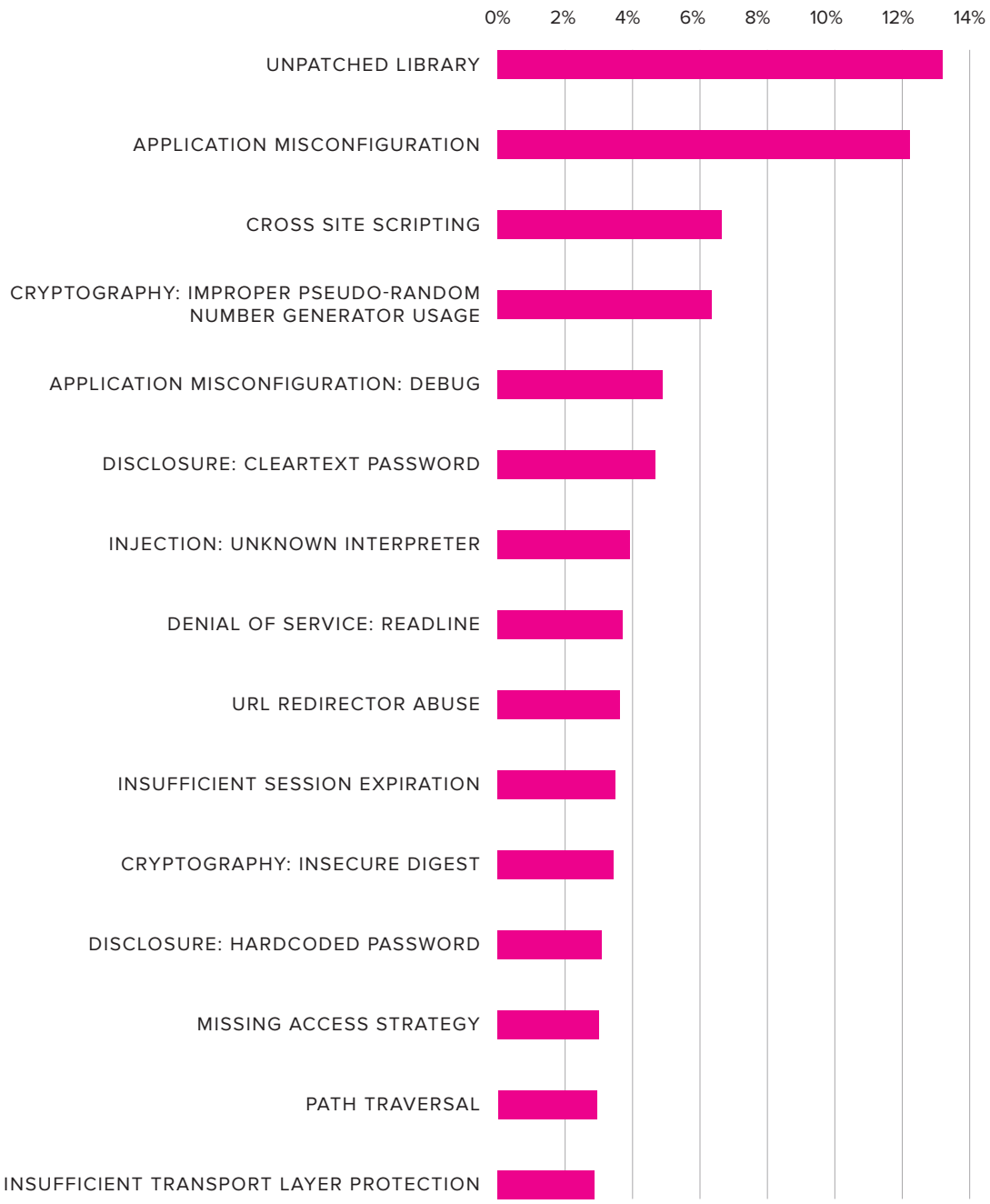
The same applies to the second most prevalent error: **Application Misconfiguration**. Many software components such as embeddable debug and QA features worry little about security. Developers will enable them by default, creating configuration weaknesses that attackers can exploit. These features may provide a means to bypass authentication methods and gain access to sensitive information, perhaps with elevated privileges.

It is clear that Software Composition Analysis (SCA) is also required. SCA looks deeply at the source code of proprietary, open source and commercial code to identify and inventory all vulnerabilities present.

Comparing *Figure 8* to *Figure 3*, note that most vulnerability classes that rank prominently in SAST scans of applications in development do not appear in DAST scans of production applications.

The notable exception is **Cross-Site Scripting**, the third most likely error to be uncovered in development. This illuminates the fact that development teams are not being vigilant enough, allowing XSS to make its way into live applications far too often. Perhaps even more distressing: they are not remediating XSS in production either *(Figure 6)*. XSS is making its case as the most dangerous vulnerability out there.

FIGURE 8. VULNERABILITY LIKELIHOOD BY CLASS - SAST



MOST SERIOUS VULNERABILITIES BY CLASS

Critical software errors such as SQL Injection must be fixed during development to reduce exposure in production.

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.

As shown in *Figure 9*, there are three classes of vulnerabilities found to be critical across thousands of applications analyzed in development.

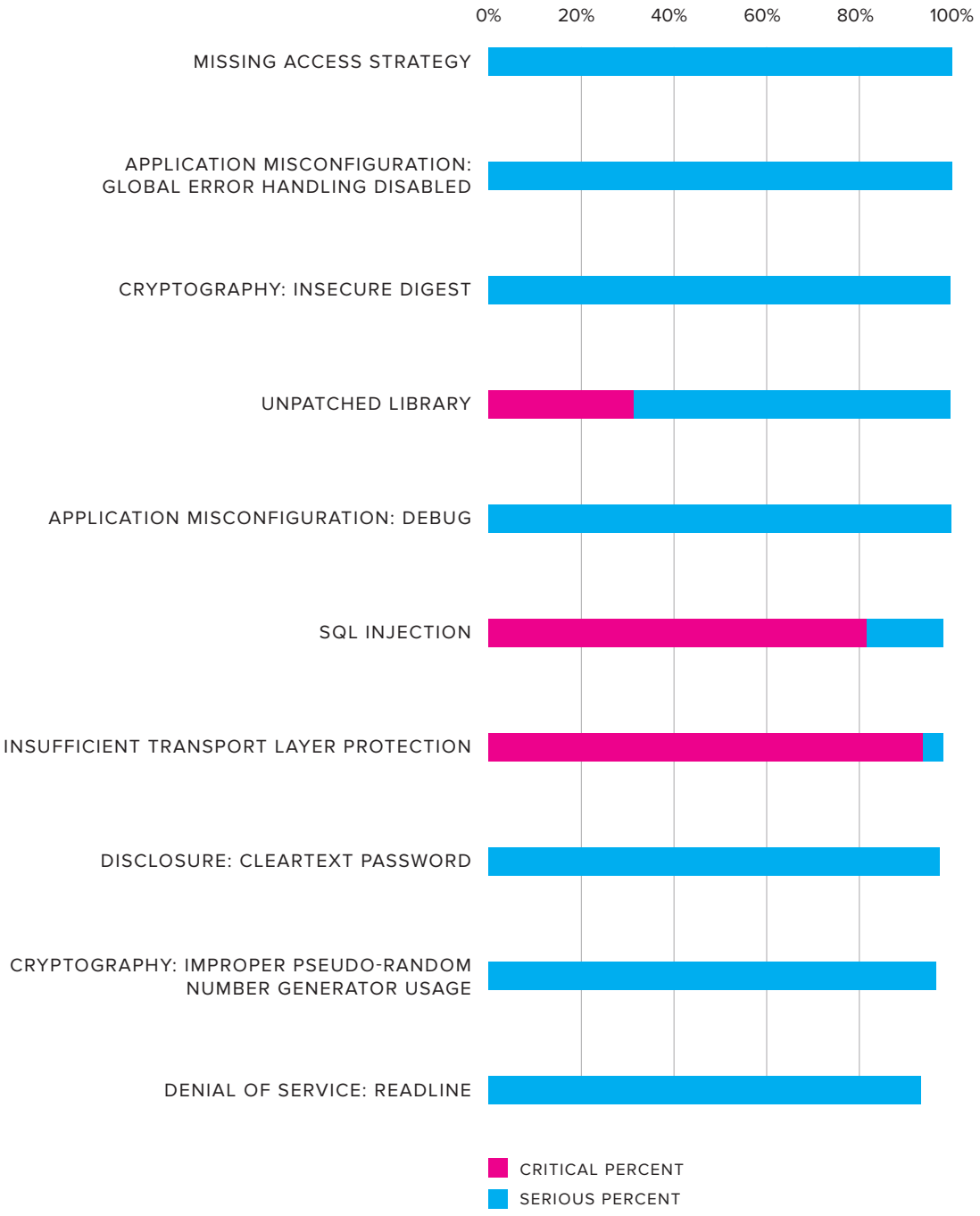
Insufficient Transport Layer Protection (ITLP) is the most critical, at 94 percent of vulnerabilities. This is a class used to describe errors such as weak ciphers, certificate misconfiguration or known vulnerable protocols. While ITLP is highly likely to be found by DAST scans *(Figure 3)*, it is rarely a critical error *(Figure 4)* because many organizations are mitigating it via a Web Application Firewall. A better fix is for developers to properly configure protocols, ciphers and certificates to make them safe. This should be a "must do" during the development process so this error does not make it into live applications.

SQL Injection (SQLi) also suffers a very high serious-to-critical ratio, at 81 percent. Both SAST and DAST detect SQLi criticality equally well, but unlike ITLP it *cannot* be fought with a firewall. While SQLi attacks are not easily mitigated, they are easily preventable, highlighting the importance of remediating these errors in development.

Unpatched Libraries are not only the most likely vulnerability found by SAST, but also critical one-third of the time. Open source components such as libraries must be fixed in development. The best method is to include Software Composition Analysis (SCA) testing which examines the security of all source code, including components.

Certain vulnerabilities can be mitigated in production, while others like SQLi must always be remediated in development. A variety of software security testing regimens routinely performed across the SDLC is the best Application Security approach. Platform solutions provide this level of visibility and control, leaving organizations with enough intelligence to understand how best to fix any software error... for the least cost.

FIGURE 9. MOST SERIOUS VULNERABILITIES BY CLASS - SAST

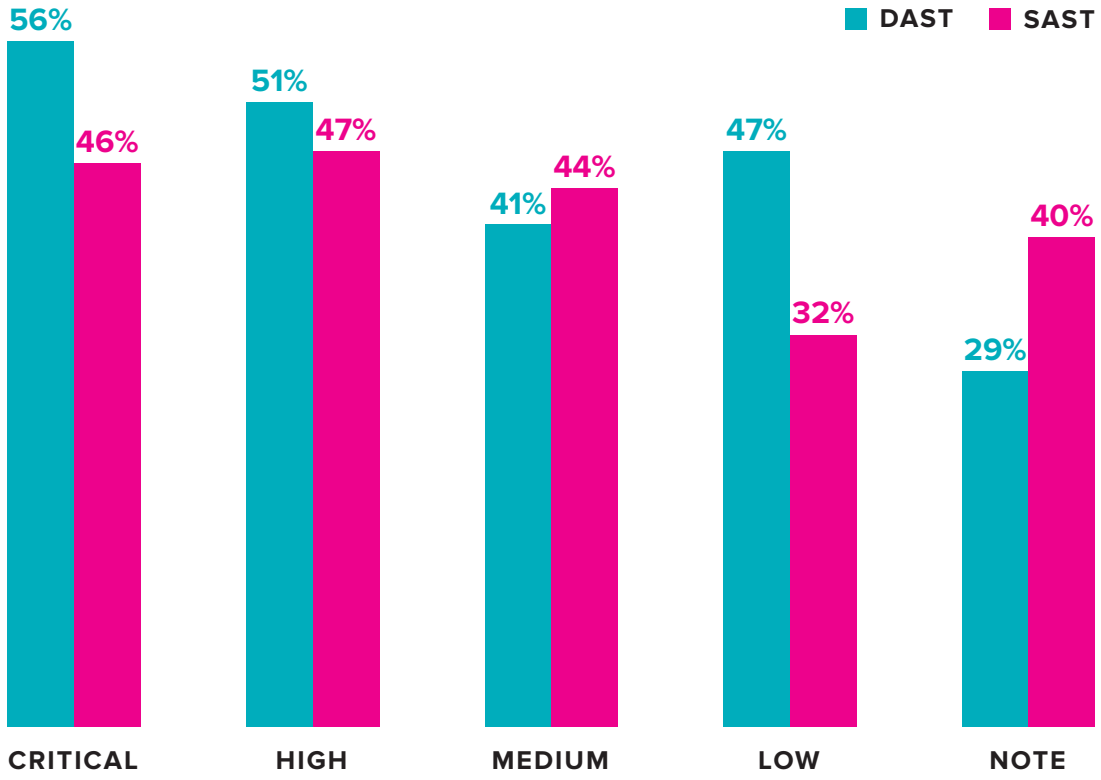


REMEDIATION RATES BY RISK LEVEL

Less than half of critical vulnerabilities are fixed in development, yielding high risk in production.

The easiest and cheapest place to fix any software errors, including security vulnerabilities, has long been understood as during development. Yet an examination of remediation rates observed in static versus dynamic testing reveals some dramatic differences. Less than half of any kind of vulnerability – critical or not – are ever fixed before release. The majority of serious errors found in live software are eventually remediated, but why are organizations taking this unnecessary risk? More education is needed by security professionals to help development teams understand risk levels and how to use them to prioritize remediation in development and production applications alike.

FIGURE 10. REMEDIATION RATES BY RISK LEVEL - DAST VS. SAST



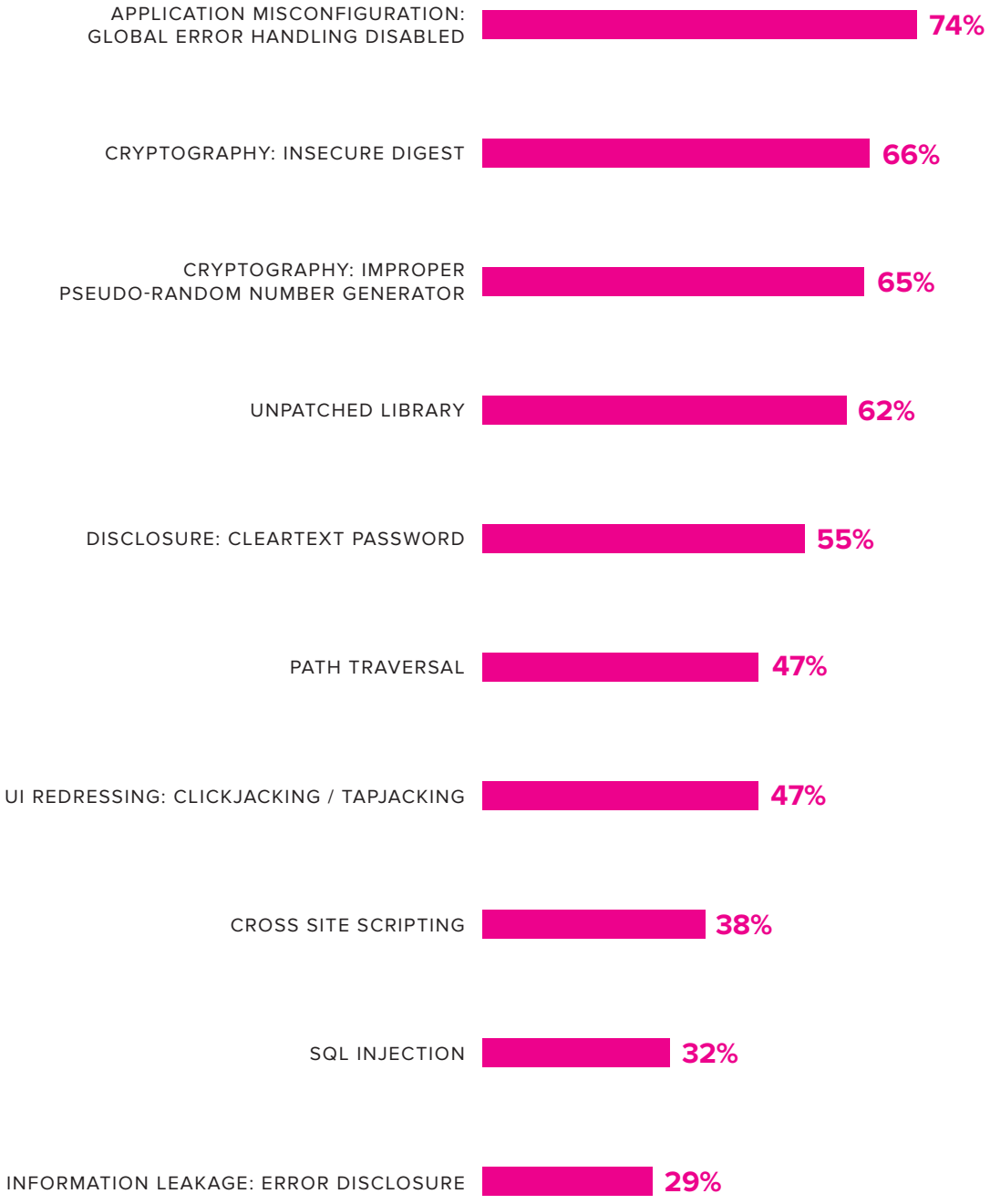
REMEDIATION RATES BY VULNERABILITY CLASS

Developers are prioritizing easy fixes over the tougher, more serious vulnerabilities.

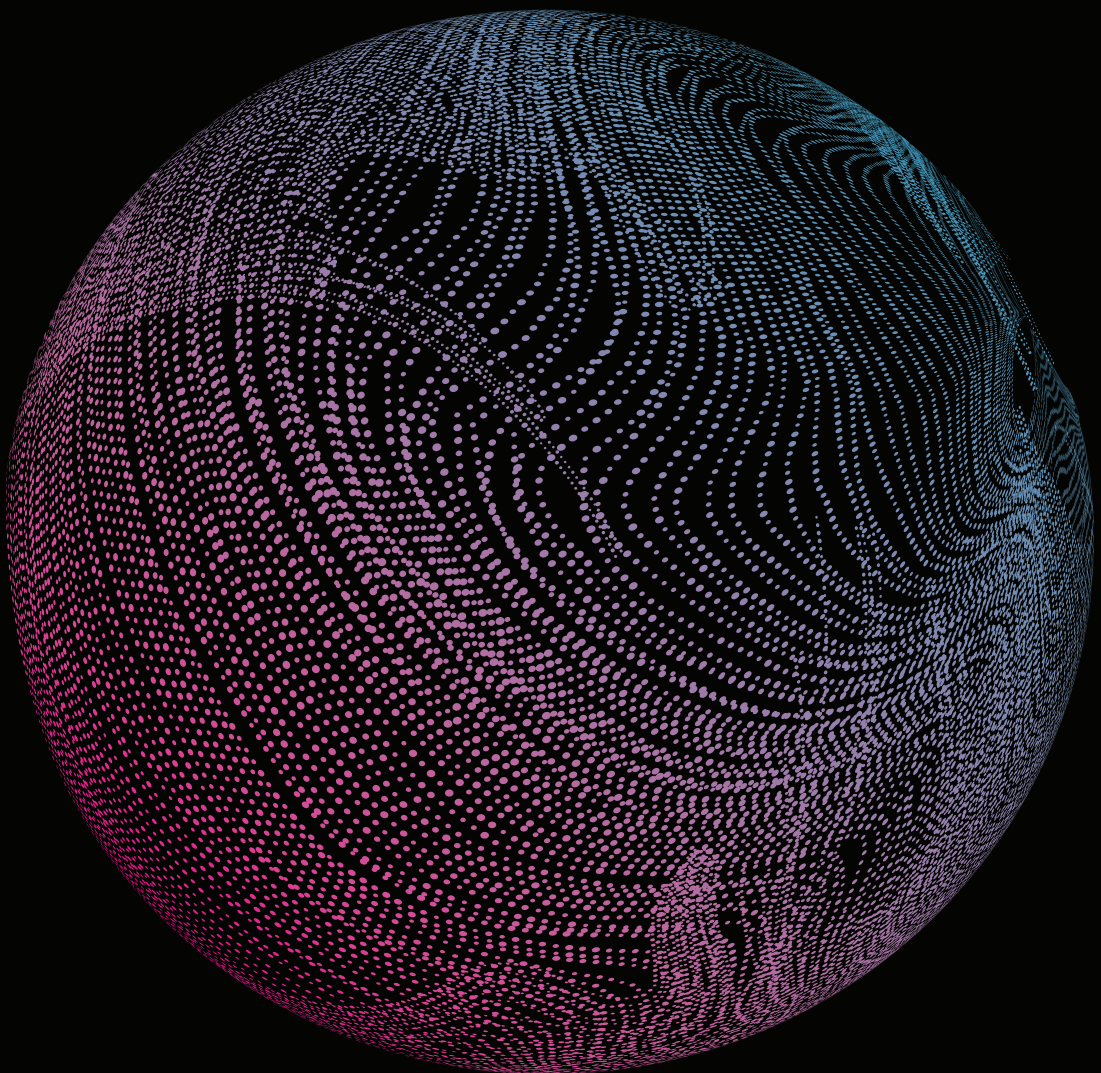
It's human nature to take the path of least resistance. It's natural to pick the lowest hanging fruit. Among the top application vulnerabilities revealed by SAST scans, it seems the easiest fixes are the ones getting all the attention *(Figure 11)*. Those enjoying the highest remediation rates by developers – Application Misconfiguration (a rate of 74%), Insecure Digest (66%), and Unpatched Library (62%) – are all easy fixes. While the toughest, most complex to resolve – XSS (38%) and SQLi (32%) – are not being addressed adequately.

As shown in *Figure 9*, unpatched open source components like libraries are a critical error a third of the time, so it's a good thing that they are being addressed. The severity levels of SQLi and XSS merit more attention whenever and wherever they are found. However, Insufficient Transport Layer Protection is critical 94 percent of the time it is found, yet appears nowhere on this chart.

FIGURE 11. REMEDIATION RATES BY VULNERABILITY CLASS - SAST



USING DAST & SAST IN COMBINATION



DISTRIBUTION OF VULNERABILITIES BETWEEN DAST & SAST

DAST & SAST are complementary application security testing tools that should be used in combination.

It is important to compare the results of DAST & SAST scans in combination.

Here are a few reasons why:

- The number of vulnerabilities per asset is very high in SAST compared to DAST. See *Figure 12*.
- Vulnerabilities found by DAST aren't always found by SAST. (Compare *Figure 3* to *Figure 8*)
- DAST has more uniform distribution of errors compared to SAST.
- Not everything found in development may be exploitable when the production application is running.

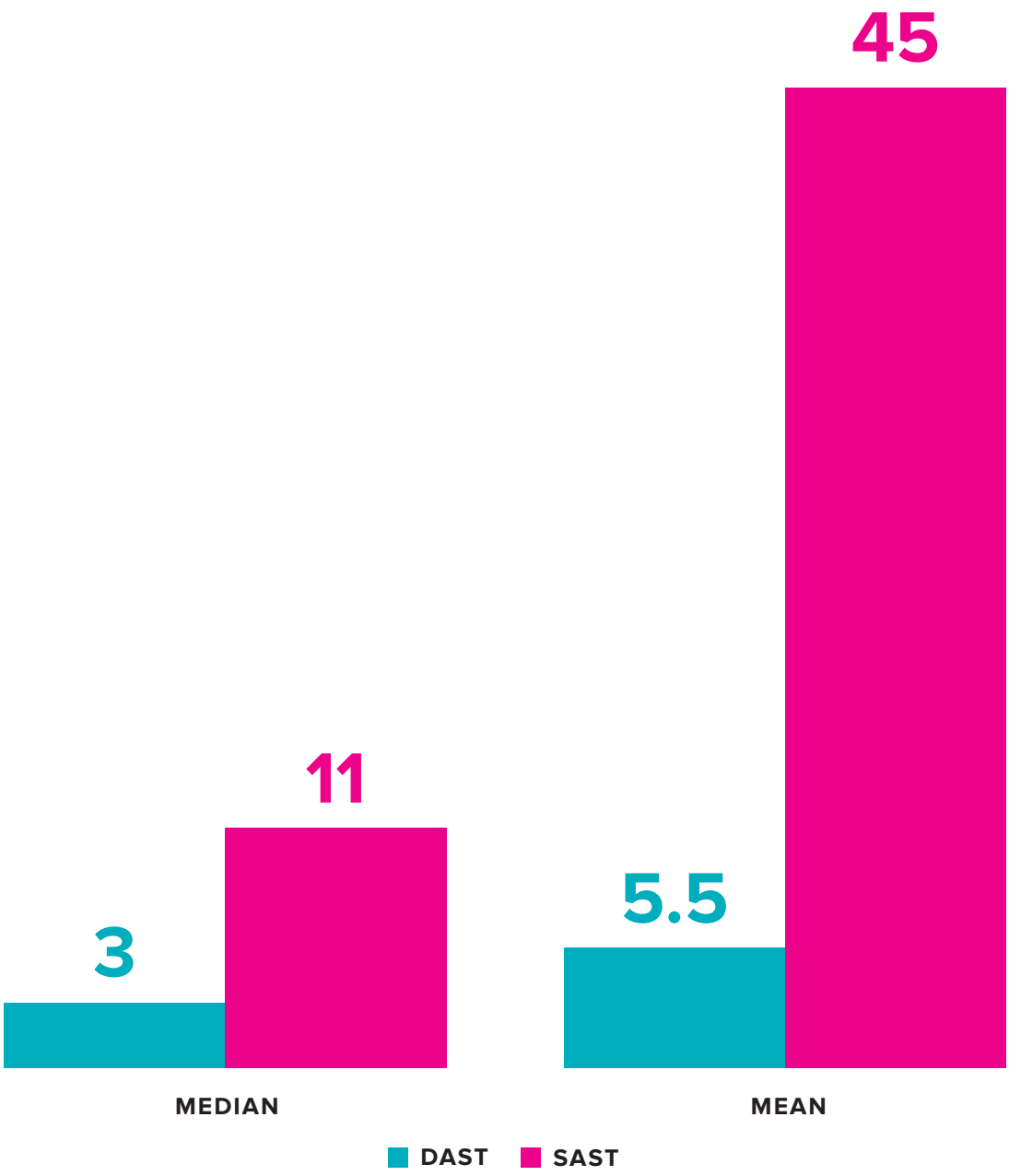
Organizations should pay attention to finding and fixing serious flaws in development. Some vulnerabilities may not show up at all in production. Developers can fix things early and have them never show up in DAST scans.

Compare DAST & SAST results and take action on the most critical issues. For example, focusing on SQL Injection and Cross-Site Scripting as exploitable errors in both DAST & SAST is always advisable.

Application Security solutions – the combination of people, process and technology – must address the entire product lifecycle. They must provide visibility and control across the entire SDLC. Some mistakes made in development won't be found by testing only in production. Employ these two unique testing tools in tandem to discover and remediate the most serious vulnerabilities.

FIGURE 12. DISTRIBUTION OF VULNERABILITES BETWEEN DAST & SAST

Number of vulnerabilities per asset between SAST & DAST



TIME TO FIX DAST VS. SAST

It's faster and easier to fix software errors in development instead of production.

Figure 13 shows the average time-to-fix (TTF) in days for vulnerabilities found in DAST, at 174 days, versus SAST, 113 days. It should be noted that TTF measures are not reflective of level of complexity to fix any given error.

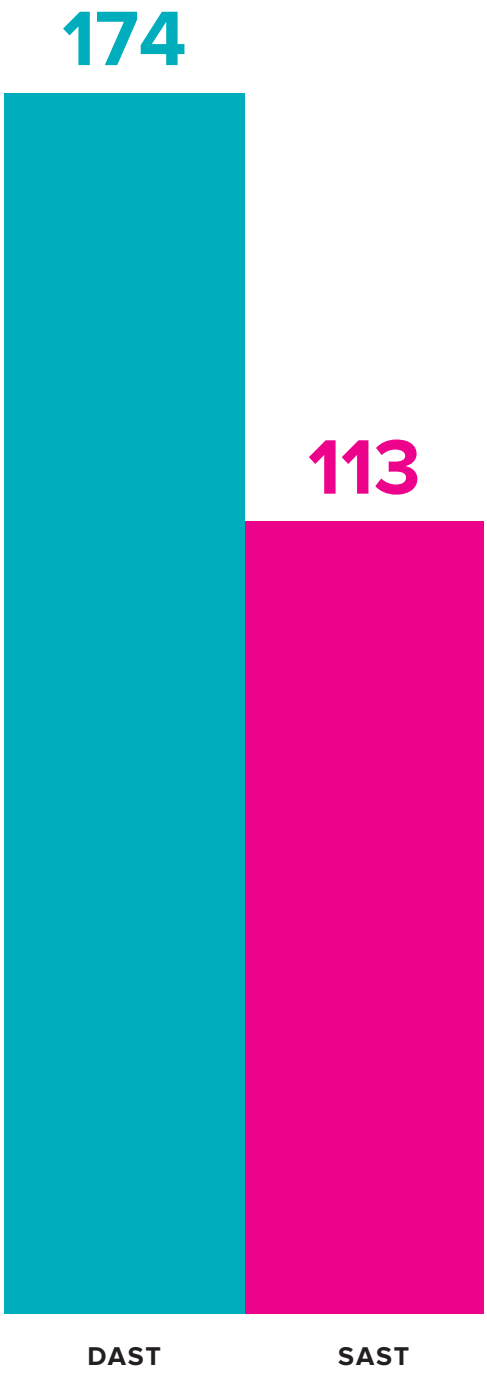
DAST vulnerabilities take longer to fix on average because applications are already in production. This is why it is so important that SAST scans occur when software is still in development. That being said, remediation periods of almost four months to nearly six months are still too long, too slow.

This finding indicates that vulnerability remediation processes could be locked into waterfall development cycles, which are slower and more deliberative than iterative, more agile methods. In any case, it points to the need for organizations to streamline remediation processes across the board, and simplify the handoff between security and development teams.

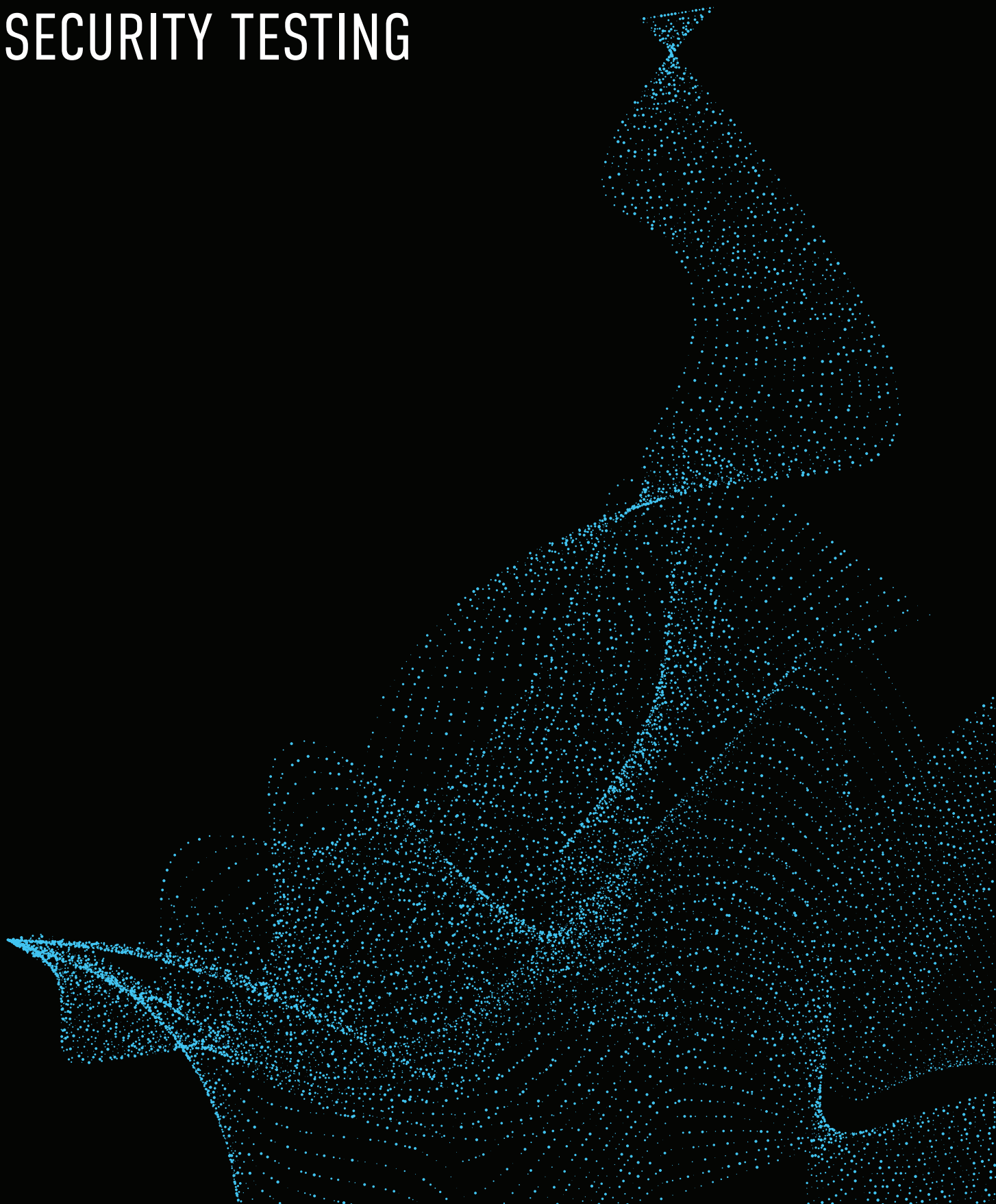
Not everyone has adopted DevOps, and those that have may not be extending or expanding its methods to include DevSecOps. An agile security approach employs both DAST & SAST testing integrated with the SDLC, followed by risk-based remediation strategies. DevSecOps still remains a theory, not yet widely in practice (or not being implemented correctly).

Any formalized Application Security program may want to consider investment in a platform solution for greater visibility and control. Most combine a shared set of testing services integrated with common, popular SDLC tools. By making it easier for enterprises to leverage software security at the right points in their development and release cycles, platform solutions may be able to reduce these TTF times.

FIGURE 13. AVERAGE TIME TO FIX FOR VULNERABILITIES (IN DAYS) - DAST VS. SAST



MOBILE APPLICATION SECURITY TESTING



The last few years have seen an explosion in the usage of smart phones and tablets, which has led to a tremendous increase in the number of mobile apps. One of the most appealing parts of the mobile app is that it can be built by anyone with the right platform or mobile development environment. However, mobile app development is often done faster and with less attention to security compared to traditional web applications.

Even though Android has the largest market share among mobile customers[2], iOS has made slight gains, which leads most companies to want to create mobile applications for both platforms so that their users can use their favored phones to do business on the go. This doubles the work for the development teams who need to test their most business-critical mobile applications to make sure these are not an avenue of information exfiltration for users or your organization.

An average phone connects to over 160 different IP addresses during the day, with about a third of the information flowing in and out of a phone unencrypted (SMS, some emails, etc.). Within Android, for example, there are "normal" permissions that are granted by the system when requested by an application, and other, more "dangerous" permissions that are granted by the user. In terms of what an app will share with other apps, the developer needs to carefully implement the appropriate limitations, keeping in mind the idea of least privilege.

There are visual cues for security and encryption for web applications (https, browser warnings) that are not present for mobile apps. In analyzing the data from the NowSecure mobile app security intelligence engine for this report, the most common findings on both platforms had to do with session-level events, from cookies and session handling to lack of encryption of sensitive data like logins and passwords, and weak key size.

The following data points come from the NowSecure mobile app security intelligence engine. The engine consumes data from hundreds-of-thousands of mobile app security assessments consisting of dynamic or static analyses of Android and iOS apps.

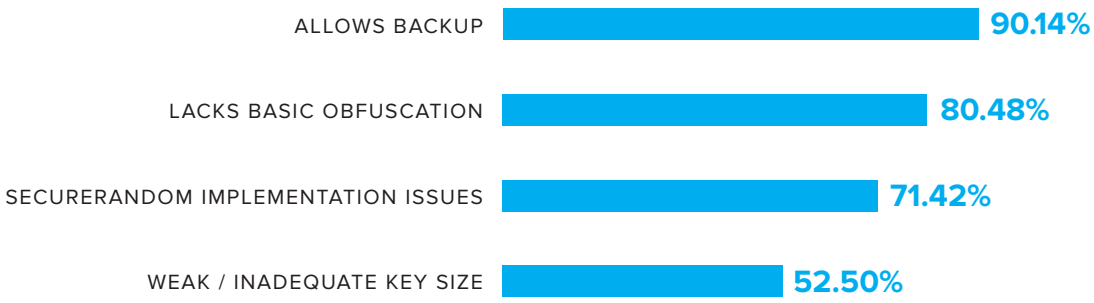
[2] *IDC, "Smartphone OS Market Share, 2016 Q3", http://www.idc.com/promo/smartphone-market-share/os*

TOP ANDROID APP SECURITY ISSUES

WORTH NOTING:

- 90% of the Android apps we tested set the allowBackup flag to true which can allow an attacker to backup the app folder and recover private data from it.
- 80% of the Android apps we tested did not use basic obfuscation which can make it easier for an attacker to reverse-engineer an app, which can put intellectual property at risk.

FIGURE 14. TOP ANDROID APP SECURITY ISSUES



Allows Backup: If the AllowBackup flag is enabled, it could allow for easier access to the app files stored on the mobile device. The severity of this issue is "Medium" and it is caught 90.14% of the time during a static scan of Android apps.

Lacks Basic Obfuscation: If an app's source code is not obfuscated, it's easier for an attacker to reverse engineer the app and puts intellectual property at risk. The severity of this issue is Medium and it is caught 80.48% of the time during a static scan of Android apps.

SecureRandom implementation issues: An app that uses the Java Cryptography Architecture (JCA) for key generation and does not properly initialize the pseudo-random number generator (PRNG) may not receive cryptographically strong values on Android devices. The severity of this issue is Medium and it is caught 71.42% of the time during a static scan of Android apps.

Weak/inadequate Key Size: The key used to sign the app is less-than-or-equal to 1024 bits, which leaves the app vulnerable to forged digital signatures. The severity of this issue is Medium and it is caught 52.50% of the time during a static scan of Android apps.

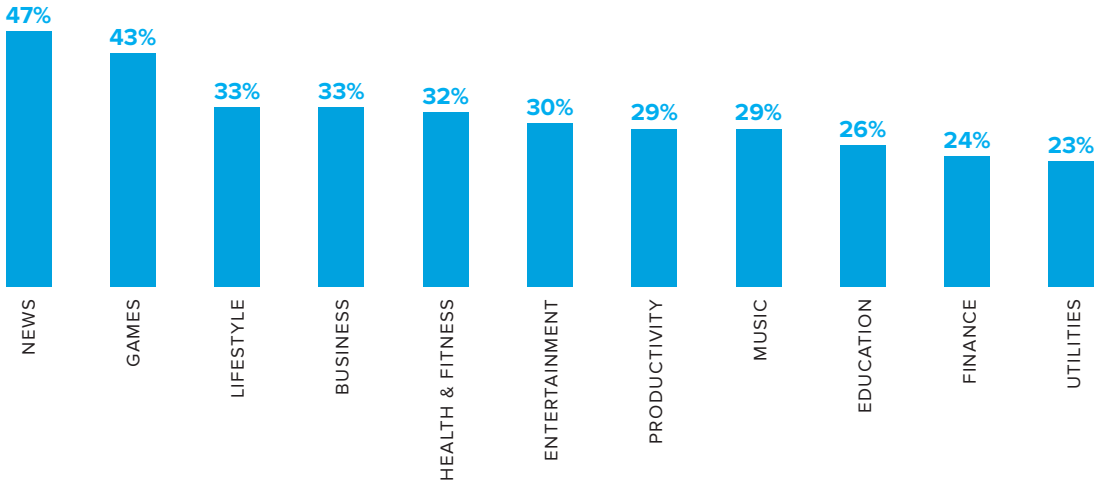
ANDROID VULNERABILITIES BY MOBILE APP CATEGORY

WORTH NOTING:

- We identified a security issue in 26.17 percent of Android apps tested.
- Of those apps that included a finding, there was an average of 2.28 issues per app.
- We identified a high-risk security issue in 21.22 percent of Android apps tested.
- Of those apps that included a high-risk finding, there was an average of 2.36 high-risk issues per app.

Apps within the Android sample that we were unable to categorize are not included in this particular statistic. An app can be listed under more than one category in Google Play. If an app was listed under more than one category, we counted that app in the data set for each category. We did this because it tells a more complete story about apps within any one genre. Example: if an Android app is listed under "Business" and "Communication", one user might consider the app a Business app while another might consider the app a Communication app.

FIGURE 15. ANDROID VULNERABILITIES BY MOBILE APP CATEGORY

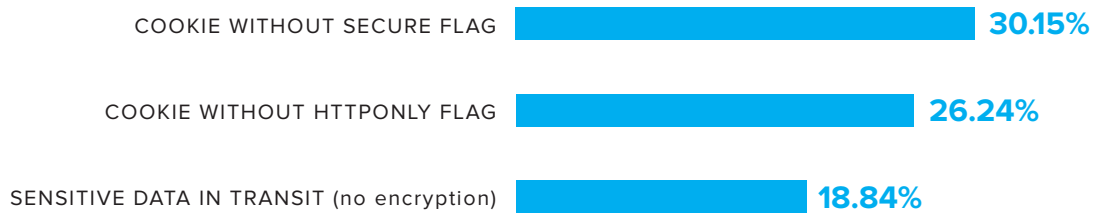


TOP iOS APP SECURITY ISSUES

WORTH NOTING:

- Cookie without secure flag is a common configuration issue in iOS which leads to security vulnerabilities.

FIGURE 16. TOP IOS APP SECURITY ISSUES



Cookie without secure flag: A cookie within the app was not marked "Secure" and therefore may be transmitted over HTTP even if the session with the host is secure. The severity of this issue is Medium and it is caught 30.15% of the time during a dynamic scan of iOS apps.

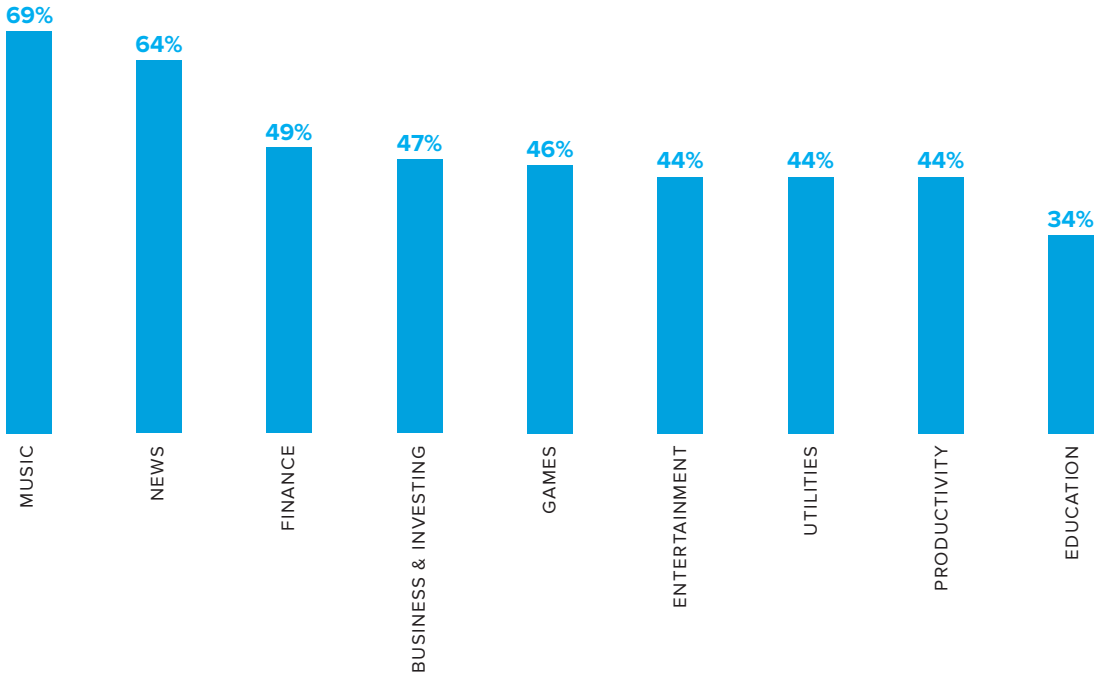
Cookie without httponly flag: A cookie within the app was not marked "HTTPOnly" and therefore may make the cookie accessible via the client-side and leave the app vulnerable to cross-site scripting. The severity of this issue is Medium and it is caught 26.24% of the time during a dynamic scan of iOS apps.

Sensitive data in transit (no encryption): Traffic included sensitive data (which might include username, password, GPS coordinates, WiFi MAC address, IMEI, Device Serial Number, Phone Number, etc.) transmitted over the network without encryption (i.e., via HTTP, not HTTPS). The severity of this issue is High and it is caught 18.84% of the time during a dynamic scan of iOS apps.

iOS VULNERABILITIES BY MOBILE APP CATEGORY

We were able to categorize all iOS apps within the sample. An app can be listed under more than one category in the Apple App Store. If an app was listed under more than one category, we counted that app in the data set for each category. We did this because it tells a more complete story about apps within any one genre. Example: if an iOS app is listed under “Business” and “Communication,” one user might consider the app a Business app while another user might consider the app a Communication app.

FIGURE 17. IOS VULNERABILITIES BY MOBILE APP CATEGORY



CASE STUDY

MAKING THE CASE FOR DEVSECOPS

INDUSTRY

Healthcare

PROFILE

This company is one of the largest U.S. healthcare providers. It is a multi-billion dollar enterprise serving millions of customers across many states. The provider offers insurance coverage through its own network of many major hospitals and medical groups.

SITUATION

Healthcare is a highly regulated industry in the U.S. Healthcare delivery models are constantly evolving, and this large enterprise was not immune to the tectonic changes wrought in its industry by digital business transformation. Service provision was becoming more web-oriented, so its application portfolio became a critical part of delivering its high standard of patient care. Over time, the provider's transactional systems were all becoming web-facing, with patients able to largely self-service.

To keep up, the company's IT group adopted DevOps to realize more agile, rapidly iterative application development. This methodology relies on continuous release cycles and non-stop innovation. At the same time, HIPAA compliance mandates require healthcare applications to protect patient privacy. As the attack threat surface was only growing larger, its cybersecurity team understood that a greater focus on application security would now be required. Under this constant pressure, the team also realized that its software development organization lacked the tools, talent and expertise required to succeed... and they were never going to be able to train everybody.

SOLUTION

To secure patient-facing web applications quickly, tight feedback loops would be necessary between the cybersecurity team and development teams, otherwise valuable cycles would be wasted. Yet the idea of a centralized application security office that would act as gatekeeper was debated and rejected. Application security needed to be front and center, but not a road block to DevOps. Instead, it was decided to have individual developers become the champions of more secure software, and the cybersecurity team would empower them to succeed.

A formal application security program is essential to reduce overall business risk. With the CISO's support, the cybersecurity team set up a sub-group for designing, implementing and running a robust application security program within a greater mission. It would become cybersecurity's job to raise awareness among software development teams, enroll individual developers in the effort, and build the bridges necessary for success. Starting in 2015, they began reaching out to each development team's leader asking them to nominate one person to act as liaison to the cybersecurity office. After training them on secure coding techniques, the cybersecurity team proclaimed these new subject matter experts "Security Heroes"! The goal was to foster *positive* collaboration, not a *punitive* approach correcting developer mistakes.

With the infrastructure of its new application security program now in place, this healthcare organization decided to switch to **WhiteHat** and its scalable cloud-based application security platform that provides highly accurate, highly actionable results. The platform provides a comprehensive set of holistic testing services, including both DAST and SAST.

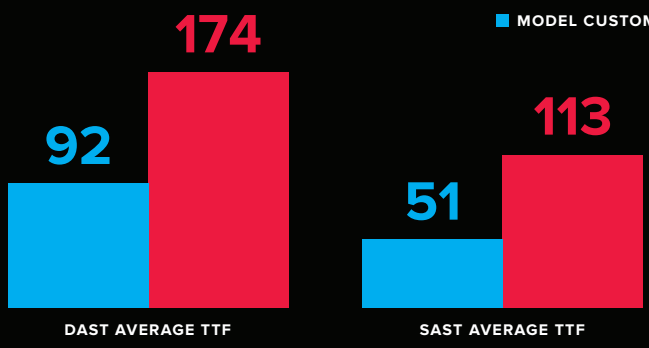
The ultimate goal was automation. WhiteHat needed to be seamlessly integrated into existing software development processes and tools of choice, so there would be no negative impact on developer productivity. This meant integrating WhiteHat Sentinel via APIs with a Jenkins build server and JIRA workflow system. Code is developed daily and checked in for scanning on a nightly basis. A worklist for each individual developer then shows up directly in JIRA the following morning. There are no extra steps or added security barriers blocking in the way. In fact, a code assessment can be ordered at any point in the software development lifecycle, making it easy for development teams to catch and fix critical vulnerabilities earlier in the SDLC. WhiteHat doesn't meter its services, so software security improvements are frictionless and continuous.

RESULTS

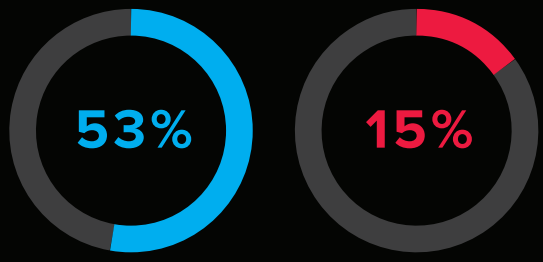
This large healthcare organization has demonstrated a successful implementation of a real DevSecOps approach. It has created a sustainable infrastructure for software development teams to be not only successful, but self-sufficient. The cybersecurity team understands its role is to provide value, advice and expertise acting as change agents and thought leaders in application security. In the process, it has proven to be a true center of excellence for application security.

The proof is in the data. Critical DAST vulnerabilities have been cut in half. Errors in production applications are fixed in an average of 92 days compared to the industry average of 174 days. Even better, SAST vulnerabilities are remediated in less than half the time of the overall industry. For example, teams discovered 75 SQL injection errors in 2016. Of these, 78 percent were able to be remediated, many of the most serious flaws in under ten days apiece.

TIME TO FIX (IN DAYS) FOR MODEL CUSTOMER VS. OTHERS



VULNERABILITY REMEDIATION RATE (WITHIN 10 DAYS) FOR MODEL CUSTOMER VS. OTHERS



Metrics are returned continuously to improve the program. These are measured and monitored over time, like strengthening a muscle for quality software code. Starting with baseline training and certification, more advanced developer courses have been created and offered on a scheduled basis. In the future the cybersecurity team will provide additional fun incentives for development teams, such as gamifying vulnerability mitigation with a leaderboard. The goal is always to work toward data breach prevention and attain even more success in the future.

THE PATH FORWARD

RECOMMENDATION.1

ADOPT DEVSECOPS TO DELIVER THE COMPETITIVE ADVANTAGES OF APPLICATION SECURITY.

Organizations already adopting DevOps practices must extend their approach to include application security practices as well. DevSecOps is the path to application security in agile development environments. The time is now. There are real-world examples that prove its advantages. Results of DevSecOps focus on speedier process improvements include a lower average time to fix and higher remediation rates.

RECOMMENDATION.2

TAKE A RISK-BASED APPROACH TO REMEDIATING APPLICATION SECURITY VULNERABILITIES.

Avoid taking the path of least resistance in prioritizing remediation of code fixes (i.e. don't just fix the easiest ones first). Risk-based remediation processes weigh criticality against an application's primary use cases. Educate all development teams to understand the relative risk levels of different application vulnerability types. The path to a true Center of Excellence is positive collaboration between the security team and developers, not punitive.

RECOMMENDATION.3

FIX CODE VULNERABILITIES COMMONLY FOUND IN PRODUCTION SOFTWARE EARLIER, DURING DEVELOPMENT.

Application security testing in development makes resulting production applications stronger. Different classes of vulnerabilities are more likely to be found, more often, in SAST as opposed to DAST. Certain vulnerabilities take a shorter amount of time and are easier to fix during development. For example, both SQL injection and Cross-Site Scripting are critical to address wherever and whenever they are detected. Coordinate static and dynamic scanning together as routine processes for the optimal application security posture.

RECOMMENDATION.4

INTEGRATE ALL APPLICATION SECURITY TESTING METHODS WITH THE SOFTWARE DEVELOPMENT LIFECYCLE (SDLC).

Organizations with the most mature application security programs integrate multiple kinds of testing (DAST, SAST, Mobile, etc.) throughout the application lifecycle. Think about application security earlier in the SDLC to see the best results. Why? Because it is faster and costs less to catch and fix flaws during the development process. At a minimum, integrate DAST and SAST in combination. Application security platform solutions are a viable option for better visibility and control, as they extend to SCA, API testing, training, and other shared services.

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 aggregated data comes from actual code-level analysis of billions of lines of code and tens of thousands of websites, web applications and mobile apps. Sentinel inspects the full spectrum of applications, including components and shared libraries. Industry information for websites and applications is provided by customers.

The report's statistical analysis focuses exclusively on assessment and remediation data for custom applications. 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).

Some biases may be intrinsic to the data source – for instance, a site that was recently on-boarded to Sentinel will have a much lower Open Vulnerability Age for any of its vulnerabilities than a site that has been using Sentinel longer. Different service levels for different sites also mean that certain vulnerabilities may be more likely to be discovered than others. Finally, tests are continually evolving, and change in tests over time may affect apparent trends.

APPENDIX: GLOSSARY OF TERMS

Web Application Vulnerability Classes

ABUSE OF FUNCTIONALITY

Abuse of Functionality is an attack technique that uses a web site'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 (DoS) attacks attempt to prevent an application from serving normal user activity. DoS 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 CROSSDOMAIN CONFIGURATION

The crossdomain.xml file is used to determine from which resources a Flash application is allowed to access data. Insufficient Crossdomain Configuration reflects a poorly configured Flash application that can be compromised to allow an attacker access to all the resources allowed in the crossdomain file. This error often occurs because a crossdomain 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 preform 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 a 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.

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 an 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.

Android Mobile App Vulnerability Classes

ALLOWS BACKUP

If the “allow backup” flag is enabled, it could allow for easier access to the app files stored on the mobile device.

DYNAMIC CODE LOADING

While this is not a vulnerability per se, it can lead to code injection or malicious side-loading of code.

JAVASCRIPT INTERFACE CHECK

Using `addJavascriptInterface()` can allow an attacker to intercept network traffic sent by the app and/or interact with the JavaScript interface.

WEAK/INADEQUATE KEY SIZE

The key used to sign the app is less-than-or-equal to 1024 bits, which leaves the app vulnerable to forged digital signatures.

LACKS BASIC OBFUSCATION

If an app’s source code is not obfuscated, it’s easier for an attacker to reverse engineer the app and puts intellectual property at risk.

SECURERANDOM IMPLEMENTATION ISSUES

An app that uses the Java Cryptography Architecture (JCA) for key generation and does not properly initialize the pseudo-random number generator (PRNG) may not receive cryptographically strong values on Android devices.

SENSITIVE DATA IN TRANSIT (no encryption)

Traffic included sensitive data (e.g., username, password, GPS coordinates, WiFi MAC address, IMEI, Device Serial Number, Phone Number, etc.) transmitted over the network without encryption (i.e., via HTTP, not HTTPS).

WORLD READABLE FILES CHECK

Files with world-readable permissions enabled can allow other apps on a device to read app files which may include sensitive data. Google applied a protection measure in Android 6.0 (“Marshmallow”) to prevent apps from reading another app’s files. We continue to test for this issue because as of March 6, 2017, 65.9 percent of Android devices use a version of Android prior to 6.0 (and therefore apps on those devices will be vulnerable).

WORLD WRITABLE FILES CHECK

Files with world-writable permissions were identified, which can grant other apps write access to those files.

ZIP FILES SENT IN TRANSIT

The app transmits ZIP files over the network, potentially in an insecure manner which, in combination with other issues (e.g., Writeable Executable issues), could result in a remote code execution vulnerability.

iOS Mobile App Vulnerability Classes

COOKIE WITHOUT SECURE FLAG

A cookie within the app was not marked “Secure” and therefore may be transmitted over HTTP even if the session with the host is secure.

COOKIE WITHOUT HTTPONLY FLAG

A cookie within the app was not marked “HTTPonly” and therefore may make the cookie accessible via the client-side and leave the app vulnerable to cross-site scripting.

SENSITIVE DATA IN TRANSIT (no encryption)

Traffic included sensitive data transmitted over the network without encryption (i.e., via HTTP, not HTTPS). Sensitive data might include username, password, GPS coordinates, WiFi MAC address, IMEI, Device Serial Number, Phone Number, etc.

About NowSecure

NowSecure is the mobile app security software company that enterprises trust to continuously secure mobile applications. NowSecure provides coverage for all mobile apps, superior testing and analysis, and multiple delivery models for automated or analyst-driven products. NowSecure customers benefit by making their app security process significantly faster and their apps more secure. For more information about NowSecure, visit www.nowsecure.com.

About WhiteHat Security

WhiteHat Security has been in the business of securing applications for over 15 years. In that time, we've seen applications evolve and become the driving force of the digital business, permeating every aspect of our lives. As a result, it's more important than ever to ensure that security experts and software developers work hand-in-hand to secure the applications that drive our daily digital experiences. The WhiteHat Application Security Platform is a cloud service that allows organizations to bridge the gap between security and development to deliver secure applications at the speed of business. This innovative platform is one of the reasons why WhiteHat has won numerous awards and been recognized by Gartner as a Leader in application security testing four times in a row. The company is headquartered in Santa Clara, Calif., with regional offices across the U.S. and Europe. For more information on WhiteHat Security, please visit www.whitehatsec.com, and follow us on Twitter, LinkedIn and Facebook.

SECURE YOUR DIGITAL BUSINESS

WHITEHAT SECURITY, INC.

3970 Freedom Circle, Santa Clara, CA 95054 • 1.408.343.8300 • www.whitehatsec.com

WhiteHat SECURITY®