TCELL BY RAPID7

WHITEPAPER

Security Report for In-Production Web Applications

TABLE OF CONTENTS

Web Application Attack Highlights

Top Five Most Common Incidents in Q2 2018

- Cross-Site Scripting (XSS)
- SQL Injection (SQLi)
- Automated Threats
- File Path Traversal
- Command Injection (CMDi)

KEY FACTS ON VULNERABILITIES	
Average Time to Patch a Vulnerability (regardless of severity level)	38 days (from discovery to patch)
Average time taken to patch High Severity Vulnerability	34 days
Average time taken to patch Medium Severity Vulnerability	39 days
Average time taken to patch Low Severity Vulnerability	54 days
Oldest unpatched CVE	340 days

Overview

We are proud to share our Security Report for In-Production Web Applications.

The report summarizes our key findings on the most prevalent types of real-world attack that occur inside in-production web applications. The data in the report has been anonymized and aggregated to protect the innocent.

tCell by Rapid7 is an industry leader in protecting web applications at runtime, and as such, we come across a wide array of application attacks, from the simplest to the most complicated. Over Q2 2018, our team analyzed over 316 million incidents across our entire customer base.

REPORT PARAMETERS	
Sample Size	316 million incidents
Period	Q2 2018
Type of Traffic	Production Applications
Audience for Report	"C" Level, Security Professionals, DevOps, Developers and Operations

Executive Summary

This high-level report summarizes the key threats that tCell came across in Q2 2018. The data in this report has been collected, aggregated, compiled, and anonymized from actual production application traffic in the AWS and Azure cloud ecosystems.

tCell protects web applications at runtime and acts as the first line of defense for these applications. As we become a part of the application itself by installing an agent on the application server and browser, tCell is in the unique position of being able to observe attacks against the application first hand and thereby gain an unmatched perspective.

When it comes to cybersecurity breaches, attacks against the application continue to represent a major risk to enterprise. Furthermore, attacks against web applications are growing in volume and sophistication.

We observed two primary dynamics at play across the 60-day observation period:

- **Attempted Cross-Site Scripting (XSS) attacks,** which are aimed at the application user, were the most common type of security incident.
- **SQL Injection,** which is used to access sensitive information or run OS commands to gain further ccess to a system, was the second most popular attack method.

In terms of CVEs (Common Vulnerabilities and Exposures), we detected the following key trends:

- 90% of active applications had a known CVE, while 30% had a critical CVE during this time period
- There were an average of 2,900 orphaned routes or exposed API end-points per application. This represents an attack surface with no current business function, thus representing security blind spots
- On average, it took 34 days for an organization to patch their most critical CVEs (Please note this statistic may be affected by a larger profiled organization taking significantly longer to patch than many smaller ones)

This report was designed to uncover new areas of risk in application security, and confirm the presence of threats, vulnerabilities, and security incidents that teams had previously only suspected. It is our hope that you can leverage the findings to better protect your organizations against these emerging threats.

Industry Trends

The Top Five Incidents: tCell vs. OWASP

The OWASP (Open Web Application Security Project) Top Ten is one of the most popular lists for classifying web application threats and security flaws. According to the 2017 OWASP Top Ten, the leading attack was Injection Flaws, followed by Broken Authentication.

Injection flaws, such as SQL, LDAP, and OS injection, essentially enable attackers to send malicious code through an application to another system. Injection flaws occur when suspicious data is sent to an interpreter as part of a command or query to trick the interpreter into executing unintending commands, or providing access to data without the correct authorization.

Broken authentication occurs when application functions connected to authentication and/ or session management are not correctly implemented, allowing attackers to compromise passwords, session tokens, keys, or to exploit other flaws stemming from incorrect implementation. It can lead to the assumption of other users' identities.

In tCell's analysis, our top five attacks differ from OWASP's top five. The main reason for this is that tCell protects applications in-production that reside in the AWS, Azure and Google cloud environments. This gives us an unique perspective on application security in production, and the nature of the attacks themselves.

	TCELL	OWASP
1	Cross Site-Scripting	Injection Flaws
2	SQL Injection	Broken Authentication
3	Automated Threats	Sensitive Data Exposure
4	File Path Traversals	XML External Entities
5	Command Injection	Broken Access Control

Why is Web Application Security Essential?

In the 2017 Verizon Data Breach Investigations Report, the most common attack pattern associated with an actual breach was web application attacks. Akamai, in its most recent State of the Internet Report, found that web application attacks had gone up by 10% year-on-year, representing a significant security threat to enterprises. There were 128 million alerts in Q4 2017 in the U.S. alone.

Web application attacks had gone up by 10% year-over-year

The majority of web application attacks are the result of overall scanning for vulnerabilities; however, many others are real attempts to compromise a particular target. In our report last year, one of the biggest takeaways we found was that web application attacks are noisy, with an attack to breach ratio of 100k to 1. According to our findings this year, this ratio hasn't changed. Web application attacks are noisy because hackers are using tools to automate attacks, essentially using automation to probe web applications for weak spots. From a security operations standpoint, this makes finding a successful attack much more difficult. We believe the reason technology like runtime application self-protection is gaining popularity is because the need to reduce the signal to noise ratio is crucial to discovering breaches. Additionally, web application threats are so frequent that they can make it challenging for organizations to simply keep their web application firewalls running effectively, and reduce capacity for taking care of updates to security systems.

Web applications, particularly those in the cloud, are vulnerable by their very nature, particularly so as applications become more sophisticated and attacks become both more complex and yet more every day. Anyone with an Internet connection and a web browser can determine a target and wreak chaos on that business or organization. An automated script exposing a loophole can immediately open the door for more advanced attacks, allowing attackers to gain access to an enterprise's network, increase their privileges, and/or gain access to confidential data. A breach of a web application can cause significant damage, particularly if it is not rapidly detected and taken care of.

Most appsec efforts to date have been focused either on creating more secure applications or on attempting to deploy network appliances to protect in production. The rapid growth of DevOps, containerization, microservices and cloud deployments have made it more essential to secure apps in production, yet simultaneously more difficult to do so. DevOps teams have to code securely whilst also keeping up with weekly, daily, or hourly releases. Likewise, security teams struggle to keep up with the required pace of updates while also issuing real-time threat data to help prioritize operations.

At Rapid7, we see security activity in production applications themselves — not theoretical vulnerabilities or possible outcomes, but the actual attacks as they occur. We looked at 316 million incidents over a sixty-day period to determine the trends identified in this report. Our key findings can provide useful insight to our clients, allowing them to measure themselves against what they are seeing in production.

Key Findings

SUMMARY OF ATTACK FINDINGS	
Top 5 Most Common Incidents	1. Cross-Site Scripting (XSS) 2. SQL Injection 3. Automated Threats 4. File Path Traversal 5. Command Injection
Average Time to Patch a Vulnerability (regardless of severity level)	38 days (from discovery to patch)
Average time taken to patch High Severity Vulnerability	34 days
Average time taken to patch Medium Severity Vulnerability	39 days
Average time taken to patch Low Severity Vulnerability	54 days
Oldest Unpatched CVE	340 days
Top Seven Weaknesses (CWEs) Found Most Frequently	1. CWE-264 Permissions, Privileges & Access Controls 2. CWE-284 Improper Access Control 3. CWE-254 Security Features 4. CWE-20 Improper Input Validation 5. CWE-200 Information Exposure 6. CWE-22 Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') 7. CWE-19 Data Handling
Percentage of Apps with Known CVEs	90% of active applications
Routes/API Endpoints Exposed per App	2,900 orphaned routes exposed (average)
Automated Threats	• Geo-Hop • Targeted Attack • Scanning Attack • Distributed Targeted Attack • Distributed Scanning Attack

The Top Five Most Common Incidents

The top five most popular breaches that tCell detected were (in order of frequency):

1. Cross-Site Scripting
2. SQL Injection
3. Automated Threats
4. File Path Traversal
5. Command Injection

Cross Site-scripting (XSS) is one of the most frequent kinds of application-layer web attacks. XSS vulnerabilities allow attackers to target client-side scripts (in the user's web browser) rather than on the server-side. The idea is to manipulate client-side scripts to behave according to the attacker's demands. A malicious script can be embedded into web pages that execute every time the page is loaded. XSS may also allow attackers to bypass access controls.

An SQL injection is a code injection method used in particular against data-driven applications, in which malicious SQL statements are inserted into an entry field for execution. SQL injection attacks enable hackers to create false identities, modify existing data or destroy it, and/or become false administrators of a database.

An Automated Threat is software that is acting on behalf of a user that identifies a requested file when one server requests it from another. As we collect data from the applications, we can tell if the interactions are the result of a user or a script or bot traffic. This helps eliminate false positives for good users and give us controls around blocking automated attacks.

The File Path Traversal attack, if successful, can lead to disclosure of sensitive data, such as the application source code, server configuration, and compromise of OS users' information. This data can then be built upon to further develop an attack.

Command Injection vulnerabilities tend to occur when data enters the application from an unauthorized source. If the application executes the spoof command, it gives an attacker privileges that they would otherwise not have.

JAVA	PYTHON	RUBY
Cross Site Scripting	SQL Injection	Cross Site Request Forgery
SQL Injection	Cross Site Scripting	SQL Injection
File Path Traversal	Cross Site Request Forgery	Cross Site Scripting

Because all applications are built differently, we took a look at the most popular languages across start-ups, small and medium business (SMB), and enterprise applications to distill the top 3 most popular attack attempts by language. As companies improve their software development lifecycles (SDLCs), it is vital to take into account what common real-world attacks are happening against the languages that the application is built on. Having this data can help lead developer training to better code against these attacks.

Cross-Site Scripting (XSS) is Common in Web Applications

XSS attacks were the most popular attack type that tCell detected across the 60-day period. They can represent different degrees of threat, from a minor nuisance to a serious security risk, depending on the confidentiality of the data handled by the site at risk and the nature of security mitigation implemented by the site owner.

However, the vast majority of XSS are merely attack attempts. In our report last year, we found that only one in 1,200 attempts were successful, which made it difficult to separate the successful attack, or breach, from the attempts. Most security operations try to detect this attack on the network or server side; however, the attack lands on the client-side browser. This means, traditionally, it is very hard to know if one of these attempts has

been successful (or not) at getting code to run in the browser. Due to the nature of tCell's instrumentation in the browser, we can see which of our client's users have been compromised by XSS. During the sixty-day period alone, there were five confirmed XSS breaches.

0.31% of users' browsers were infected with malware

tCell's out of the box Browser Security instruments the client- side browser through the delivery of the application and provides protections against browser-based attacks like XSS, Clickjacking, and Cryptomining, and flags end-users whose browsers have been compromised with malware. The impact of this shows that the browser is inherently vulnerable and in return should be incorporated as part of your application security posture. Across all organizations, we are able to see that 0.31% of users' browsers were infected with malware.

At Rapid7, we understand this and inject a Javascript agent into the browser itself to detect when an XSS incident is actually successful. This gives us an unique perspective into what is attempted vs. what is successful, which can offer a security team tangible peace of mind and concrete savings in efficiency as it eliminates the need to sift through thousands of alerts to get to a handful of actual attacks.

Cryptomining is on the Rise

Computers being taken over for illegal cryptocurrency mining is a rising trend worldwide. Applications running unexpected code in the browser could indicate cryptomining activity, and should be investigated. Illegally installed mining tools can cause applications and hardware to crash because of their heavy use of computing power, preventing the CPU from executing other tasks and potentially denying service to the application's users. From an end-user perspective, there's not much that can be done. End-users now need to monitor their system performance to look for websites that take up significant resources. From a company perspective, eliminating the ability to land a Cross Site Scripting attack will dramatically decrease the likelihood of a successful cryptomining attempt. This will provide a way to protect the end-users' browser without requiring them to change their behavior or the way they interact with the application or website.

Endpoint protection, web filtering tools, and Content Security Policies should be enabled (and regularly updated) to detect and block cryptomining scripts. To protect web applications from cryptomining, it is essential that the initial attack is blocked. According to Imperva, 88% of all remote code execution (RCE) attacks in December 2017 sent a request to an external source to attempt to download a cryptomining malware. They are aimed at exploiting vulnerabilities in the web application source code; thus need to be taken seriously and quickly addressed.

The Seven Most Commonly Found Weaknesses

Commonly found weaknesses are designated using the Common Weakness Enumeration (CWE) vulnerability scoring system. They indicate weaknesses that could lead to vulnerabilities (CVEs) in the right conditions. tCell identified the following CWEs as the seven most commonly found weaknesses across apps:

1. **CWE-264 Permissions, Privileges & Access Controls: Low Severity**
   This category relates to access control and its associated security features, including the management of permissions and privileges.

2. **CWE-284 Improper Access Control: Severity Varies by Context**
   This type of CVE is the result of an incorrect implementation of an architectural security tactic. It fails to restrict access to a confidential resource from an unauthorized actor.

3. **CWE-254 Security Features: High Severity**
   Software security is not security software. This category includes miscellaneous security features, such as authentication, access control, confidentiality, cryptography, and privilege management.

4. **CWE-20 Improper Input Validation: High Severity**
   This CVE allows an attacker to generate input in a manner that is not expected by the rest of the application. It is the result of incorrect validation of input at the software level, which impacts the control flow or data flow of a program.

5. **CWE-200 Information Exposure: High Severity**
   A flaw either in architecture, design or implementation, which (intentionally or unintentionally) discloses information to an unauthorized actor.

6. **CWE-22 Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal'): High Severity**
   Many file operations are intended to take place within their given directory; however, this weakness occurs when the software does not properly neutralize elements within the path name, allowing the attacker to access a location outside of the restricted directory. The result is the attacker can access, read, or override, delete, or corrupt critical files.

7. **CWE-19 Data Handling: Severity Varies by Context**
   This category involves weaknesses related to functionality that processes data.

Orphan Routes and APIs Add Risk Exposure

tCell found that 2,900 routes/endpoints were exposed per app (on average) whereas 92% of all routes and API end-points are orphaned routes. An orphan route is essentially when a part of the application has been abandoned by developers. It represents "dead code" with no business purpose and crucially in security terms, an attack surface that is frequently unsecured. These endpoints may not affect the operations of the application itself; however, they represent a blind spot to your security and operations teams that can be attacked and thus should be eliminated. As interconnectivity of businesses and applications grow, understanding that your attack surface area is exponentially growing through the use of APIs is a critical metric to monitor. The bottom line is that developers need to focus their attention on cutting out these orphan routes altogether, reducing attack surface with no reduction in application functionality.

92% of all routes and API end-points are orphaned routes

Automated Threats

The OWASP Foundation classifies attacks as Automated Threats when "web applications are subjected to unwanted automated usage — day in, day out. Often these events relate to misuse of inherent valid functionality, rather than the attempted exploitation of unmitigated vulnerabilities". Common types of automated attacks include, scanning, scraping, automated scripts, etc. We found that 47% of organizations experienced an automated attack within the 60-day period. What's more interesting was that the attacks were highly targeted at specific applications.

47% of organizations experienced an automated attack within the 60-day period

Conclusion

After reviewing our findings, it became quite clear that if you have an application exposed to the internet, you are going to get attacked. There are no exceptions. The question then becomes a matter of how can organizations better position themselves to get the data they need, in the time they need it, to make decisions that thwart real attacks. Very few organizations have the granular level of visibility that tCell can provide on runtime, in-production applications. We remove security blind-spots because our Next-Gen Cloud WAF integrates runtime application self-protection. This allows us to focus on the actual security risks posed to applications rather than on potential or noisy threats, as many security operations teams have to.

Increasing numbers of enterprises want transparency about the types of threats posed to their web apps: they want to know what the top types of attack are, which third-libraries are vulnerable, what represents an attack vs. a breach, where apps are pulling their content from, and so on. By protecting your applications at all levels - in the browser, the web server and the app server, tCell provides a fine level of detail in our monitoring and cloud analytics, allowing you to identify and analyze attacks (and attempted breaches) in depth.

At Rapid7, our goal is to provide Security and DevOps teams with greater monitoring, visibility, detection and mitigation against security breaches of all kinds, particularly those that occur within the app itself. As increasing numbers of application are migrated to and developed for the cloud, this service has become increasingly essential.

Learn more at rapid7.com/tcell.

About Rapid7

Rapid7 (Nasdaq: RPD) is advancing security with visibility, analytics, and automation delivered through our Insight cloud. Our solutions simplify the complex, allowing security teams to work more effectively with IT and development to reduce vulnerabilities, monitor for malicious behavior, investigate and shut down attacks, and automate routine tasks. 7,800 customers rely on Rapid7 technology, services, and research to improve security outcomes and securely advance their organizations. For more information, visit our website, check out our blog, or follow us on Twitter.