

AUTOMATED CODE ANALYSIS WEB APPLICATION VULNERABILITIES IN 2017

CONTENTS

Introduction..... 3

Testing methods and classification 3

1. Executive summary..... 4

2. How PT AI works 4

 2.1. Verifying vulnerabilities..... 5

 2.2. Data flow diagrams..... 5

 2.3. Vulnerability tree 6

3. Results 7

 3.1. Participant portrait 7

 3.2. Overall statistics 7

 3.3. Most common vulnerabilities 8

 3.4. Threat breakdown 9

 3.5. Statistics by industry 10

 3.5.1. Banks and other financial institutions..... 10

 3.5.2. Government..... 12

 3.5.3. E-commerce..... 14

Conclusion..... 15

INTRODUCTION

Positive Technologies regularly performs research and audits in the field of web application security. Findings from our in-the-field experience paint a sobering picture of the state of security. While our findings in 2017 were concerning, the results in prior years were not assuring either:

- + In 77 percent of external penetration tests, we found vulnerabilities that attackers could use to obtain access to a company's internal network.¹
- + 26 percent of all cyberincidents in Q3 2017 involved attacks on web applications.²

These numbers confirm that web applications are a big and tempting target: large numbers of unfixed, easily exploitable vulnerabilities give attackers free reign to do everything from stealing sensitive information to accessing internal systems.

Fortunately, most vulnerabilities can be discovered long before an attack even starts. And by analyzing source code, it is possible to identify a much larger number of critical vulnerabilities in web applications than is otherwise possible.

This report provides statistics on vulnerabilities in 33 web applications that were analyzed with PT Application Inspector (PT AI) in automated security assessments in 2017.

TESTING METHODS AND CLASSIFICATION

Vulnerability assessment was conducted via automated white-box testing using PT AI. White-box scanning refers to testing that makes use of all relevant information about the application, including its source code.

The vulnerability classification in this document is the same as used in PT AI. This classification differs from the Web Application Security Consortium Threat Classification ([WASC TC v.2](#)) due to its more detailed breakdown of weaknesses, which in the WASC classification are grouped in more general categories such as Application Misconfiguration and Improper Filesystem Permissions.

Some of the tested applications are publicly available on the Internet, while others are used for internal business purposes.

Our statistics only include code and configuration vulnerabilities. Other widespread information security weaknesses, such as failure to install software updates, are not considered here.

PT AI built-in mechanisms were used to evaluate vulnerability severity levels.

¹ [Security trends & vulnerabilities review: corporate information systems](#)

² [Cybersecurity threatscape: Q3 2017](#)

1. EXECUTIVE SUMMARY

Key findings:

All analyzed web applications contained vulnerabilities. Automated source code analysis revealed vulnerabilities in every web application that we analyzed. Only six percent of applications were free of high-severity vulnerabilities.

Web application users are at risk. 85 percent of the web applications had vulnerabilities that allow attacks against users. A hacker can exploit these vulnerabilities to steal users' cookies, implement phishing attacks, or infect user computers with malware.

Finance web applications are the most vulnerable. High-severity vulnerabilities were found in all tested banking and other finance web applications. The reason is that the operating logic for these applications is more complex than in other industries. Greater complexity results in more opportunities for critical vulnerabilities to arise. By exploiting these vulnerabilities, an attacker may be able to bring an application offline or run arbitrary code on a target system, which can lead to gaining control over the server hosting the web application.

All tested government web applications can be leveraged to attack users. All government web applications tested by Positive Technologies contained vulnerabilities that facilitate attacks against users. In addition, security awareness among the users of these applications is low, likely increasing the success rate of phishing attacks.

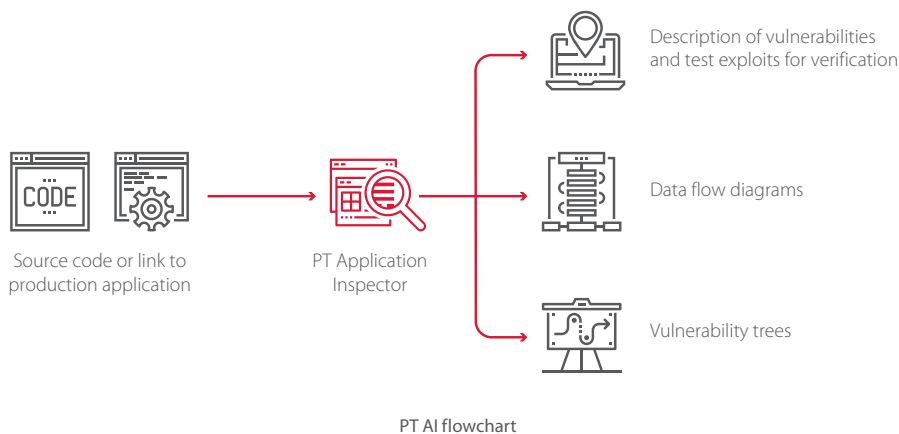
Denial of service is the most common threat for e-commerce. In 75 percent of e-commerce web applications, assessment revealed vulnerabilities enabling denial of service. DoS attacks have the potential to cause significant financial losses for owners of such web applications.

2. HOW PT AI WORKS

As the tool used to perform testing, PT AI analyzes source code (or a compiled web application) by applying abstract interpretation. This method builds upon classic static application security testing (SAST) while providing more precise results. Abstract interpretation allows generating attack vectors for any suspected vulnerability, creating specific guidance for remediation, generating test exploits, and determining any additional conditions that an attacker would need to exploit the vulnerability. If a web application is already deployed, PT AI can use dynamic application security testing (DAST) for analysis.

The output of PT AI analysis contains the following details for each vulnerability:

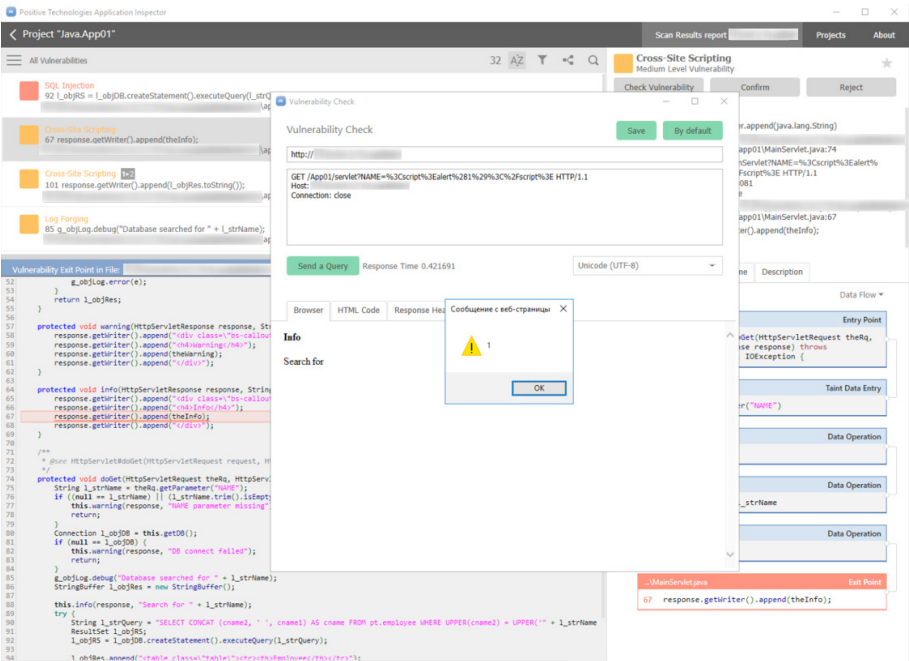
- + Vulnerability type, line number, and code fragment containing the vulnerability, description, and guidance for remediation of vulnerabilities of the relevant type
- + Test exploit to confirm or disprove existence of the vulnerability
- + Data flow diagram
- + Vulnerability tree



Context is given for each vulnerability, in the form of test exploits, data flow diagrams, and vulnerability trees. This comprehensive information gives a full picture of identified vulnerabilities and provides the details needed for fixing the source code of the web application.

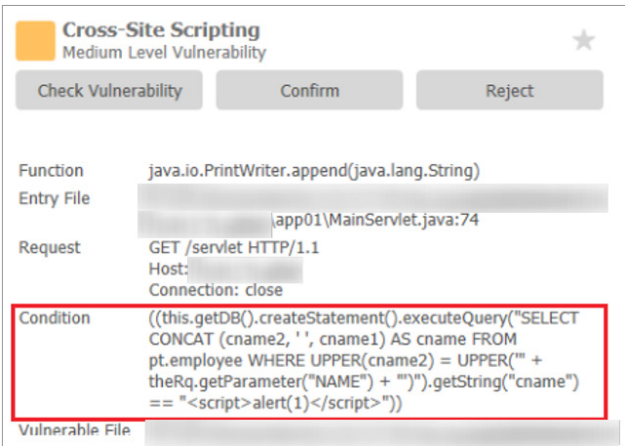
2.1. Verifying vulnerabilities

PT AI applies abstract interpretation methods in order to verify vulnerabilities and rule out false positives. PT AI generates special test HTTP requests (exploits), which are intended to exploit vulnerabilities of the web application as deployed on a test server. These exploits are used both for manual and automated (Autocheck) testing. If necessary, an exploit can be used to make a virtual patch: in this case, PT Application Firewall applies the rules needed to block attacks against the active web application until the application source code can be fixed in a more permanent way.



Vulnerability verification using a test exploit

Test exploits also can be generated for cases involving extra requirements for successful vulnerability exploitation, such as being logged in to a system. Such cases require a partial exploit combined with extra requirements for exploitation, which are defined using abstract interpretation. These extra requirements, when met, allow verifying that a vulnerability is exploitable. This method allows detecting and verifying second-order vulnerabilities.



Vulnerability exploitation conditions

2.2. Data flow diagrams

PT AI uses scanning results to plot data flow diagrams, which show a sequence of transformations of user-controlled input, from when the input appears in the application up to the vulnerability exit point (a potentially dangerous operation). Each diagram shows the path leading from entry point to exit point corresponding to a vulnerability exploitation variant.

Data flow diagrams have a standard structure and consist of the following items:

- + Entry point: starting point in the execution flow
- + Taint data entry: file and line of code describing the location of user-controlled input
- + Data operation: a description of one or more functions that modify potentially dangerous input
- + Exit point: the line where a potentially vulnerable function is run

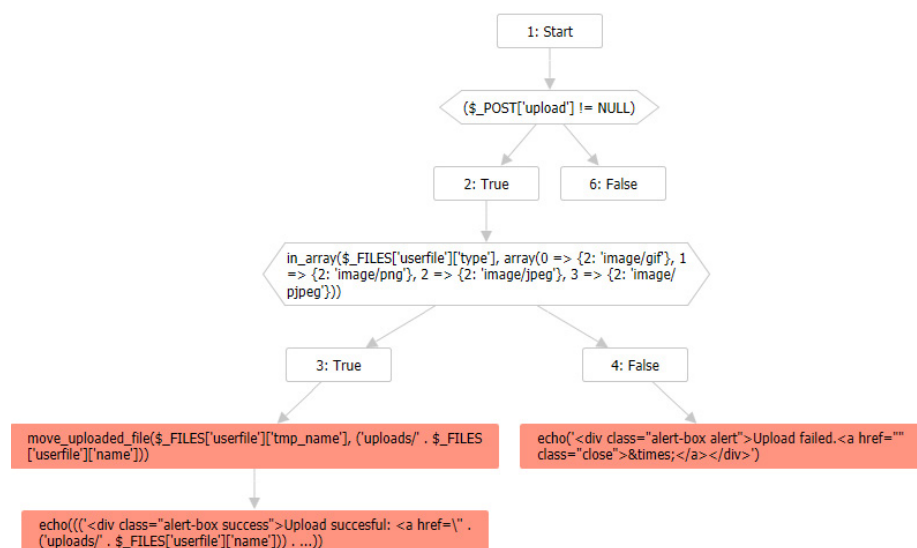
[illegible]

Data flow diagram

Data flow diagrams visualize the flow of data from a vulnerability entry point to its exit point in order to facilitate the verification process.

2.3. Vulnerability tree

A vulnerability tree is a diagram corresponding to the control flow of the application, containing branching operations and potentially dangerous operations (highlighted in red). The following example contains a simple vulnerability tree; however, most vulnerabilities have a more complex tree with a far greater number of blocks and branching operations.



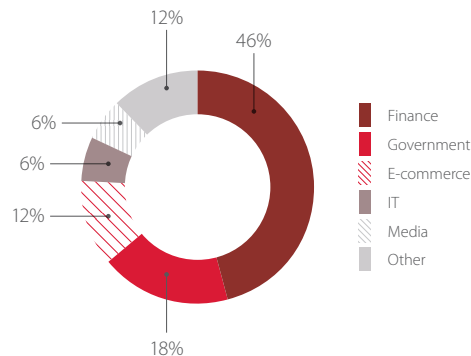
Simple vulnerability tree

3. RESULTS

3.1. Participant portrait

The web applications in our dataset tested with PT AI represent a variety of industries. Banks and other financial institutions, as well as government agencies, tend to be the most interested in source code analysis.

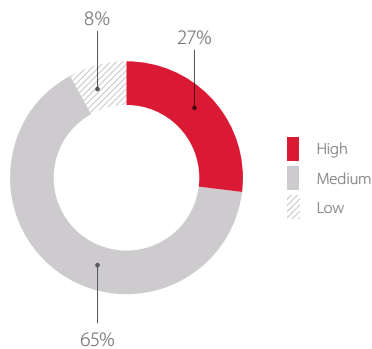
These statistics reinforce the research results of the SANS Institute:³ banking and government are particularly concerned with securing their web applications, since these web applications are the top priorities for attackers. This is also consistent with the statistical findings of Positive Technologies quarterly research on web application attacks.⁴



Web applications, by industry (percentage of tested web applications)

3.2. Overall statistics

Using automated security assessment, PT AI found vulnerabilities of various severity levels in all tested web applications. The majority of these vulnerabilities were of medium severity (65%).

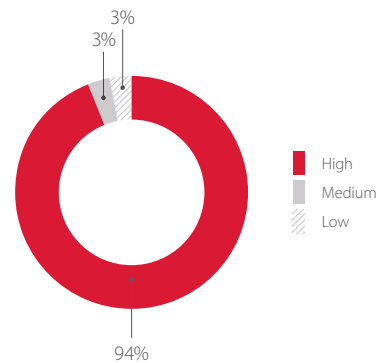


Severity of vulnerabilities

Looking at the most dangerous flaws, only six percent of tested web applications contained zero high-severity vulnerabilities. While the web applications in our dataset are not necessarily representative of all web applications (the tested applications are not standard CMS platforms and contain large amounts of custom code), even one critical vulnerability is enough to fully compromise an application or entire server.

³ [State of Application Security: Closing the Gap](#)

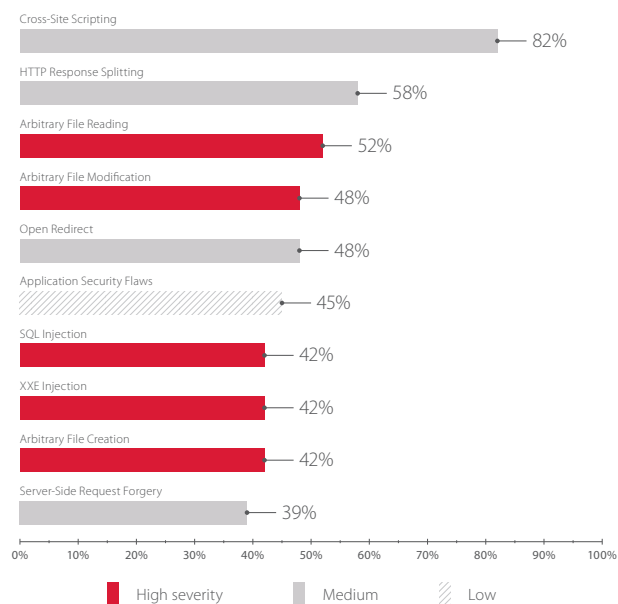
⁴ Quarterly web application attack statistics are available at: ptsecurity.com/ww-en/analytics/



Maximum severity of detected vulnerabilities (percentage of tested web applications)

3.3. Most common vulnerabilities

The most common vulnerability found in automated source code analysis was Cross-Site Scripting, which allows attackers to perform phishing attacks against users or infect their computers with malware. This same vulnerability also heads the list of vulnerabilities found in manual web application testing.⁵ Next in frequency was HTTP Response Splitting. If successfully exploited, this vulnerability allows attacks against web application users by sending a double HTTP response to a browser, with the header and field contents partially controlled by the attacker. Arbitrary File Reading, which facilitates unauthorized access to contents of arbitrary files on a server, rounds out the top three vulnerabilities. Thus an attacker can obtain web application source code, credentials, and other sensitive information processed by a system.



Most common vulnerabilities (percentage of tested web applications)

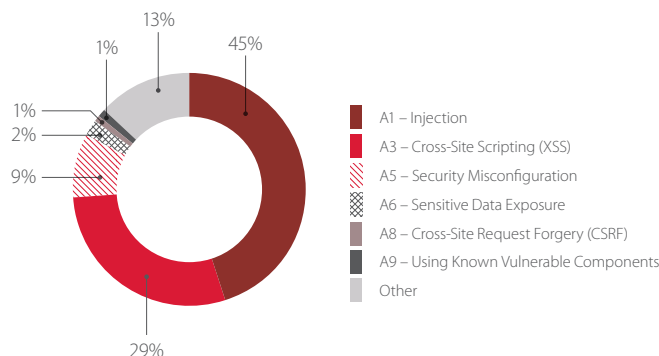
Out of the ten most common vulnerabilities, five are of high severity and, if exploited, may cause severe consequences. For example, by exploiting an Arbitrary File Creation vulnerability, an attacker may be able to execute arbitrary code on a target system and fully compromise the server.

Security flaws include missing or incorrect values of properties and directives. For example, the `requireSSL` property value was not set for some applications. This property enables or disables the `SECURE` attribute in the `Set-Cookie` HTTP header, which is responsible for requiring a secure HTTPS connection for transmission of cookies. In the OWASP classification,⁶ this flaw falls under category A5: Security Misconfiguration.

⁵ [Security trends & vulnerabilities review: web applications \(2017\)](#)

⁶ owasp.org/index.php/Top_10_2017-Top_10

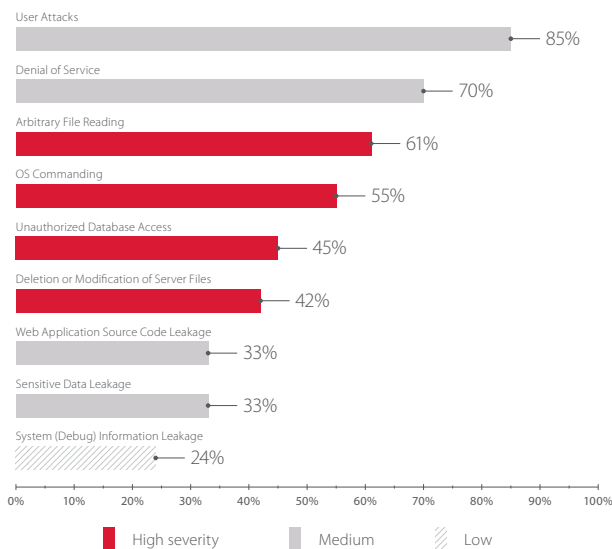
The following diagram categorizes vulnerabilities based on the OWASP Top 10 (2017). Any vulnerabilities outside of the ten most critical web application security risks are in the Other category.



Vulnerabilities categorized per the OWASP Top 10 classification

3.4. Threat breakdown

After assessing the potential impact of each web application vulnerability, we created a list of the most common threats. At the head of the list: attacks targeting web application users.

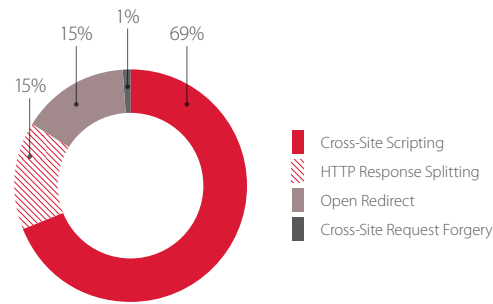


Most frequent threats (percentage of tested web applications)

Of the nine top threats, four are of high severity. By taking advantage of these vulnerabilities, an attacker could obtain unauthorized access to sensitive information on a server (61%) or database (46%), run arbitrary OS commands on a server (55%), and delete or modify files (42%). To access contents of arbitrary files on a server, an attacker must successfully take advantage of such vulnerabilities as Arbitrary File Reading and XML External Entity. SQL Injection makes it possible to steal or modify information from databases, or even delete all data. The most dangerous threat is OS Commanding. With this technique, an attacker can gain total control over a server and execute OS commands with the privileges of the web application. If a LAN interface is found on the target server, the attacker can gain access to local systems of the web application owner, amplifying the attack to a full compromise of the entire corporate infrastructure. As found in our testing, vulnerabilities often make these potentially devastating attacks feasible for real attackers.⁷

85 percent of the web applications contained vulnerabilities that allow attacks against users. Most attacks against users are the result of Cross-Site Scripting. However, many of the tested web applications contained other vulnerabilities that enable attacks: HTTP Response Splitting, Open Redirect, and Cross-Site Request Forgery.

⁷ [Corporate information system penetration testing: attack scenarios](#)

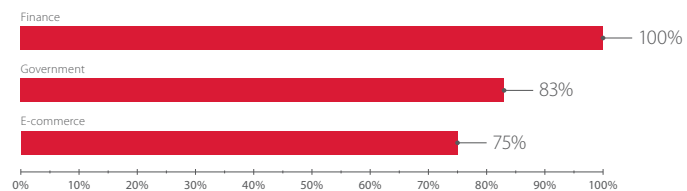


Vulnerabilities allowing attacks against clients

The results clearly show the necessity to search for vulnerabilities in web application source code both during development and on an ongoing basis.

3.5. Statistics by industry

This section contains the results of source code analysis for finance, government, and e-commerce web applications. Due to small sample size, IT and media web applications are not considered here.

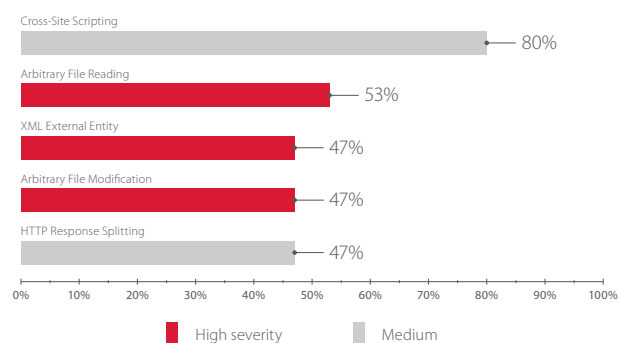


Web applications containing critical vulnerabilities, by industry (percentage of tested web applications)

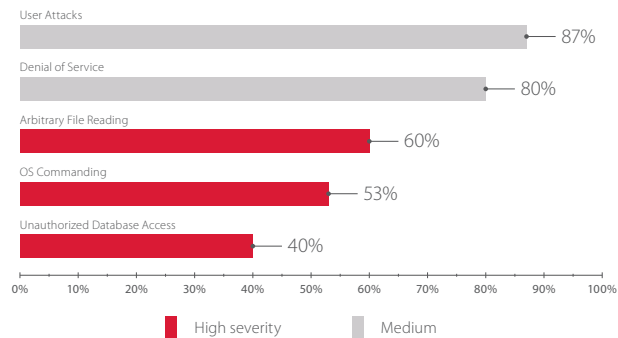
Industries differ in the vulnerabilities in their web applications, as well as their threat profiles.

3.5.1. Banks and other financial institutions

80 percent of the web applications for financial institutions were vulnerable to Cross-Site Scripting, and almost half of them to HTTP Response Splitting. These vulnerabilities are a primary factor in why 87 percent of applications allowed attacks on web application users. A risk of denial of service is the result of XML External Entity and Arbitrary File Modification. The latter can also be useful for running arbitrary code on a target system, which may lead to total control of the web application server. If the attacked application is an online banking system, or a compromised server hosting applications that involve financial transactions, vulnerabilities can bring a very large payday to the attacker.



Most common vulnerabilities in finance web applications (percentage of tested web applications)



Most common threats for finance web applications (percentage of tested web applications)

Example: Security assessment of one banking web application revealed that due to mistakes made during deployment, test and demo files were uploaded as part of a framework. This error caused numerous vulnerabilities. For instance, the index.php module contained a Cross-Site Scripting flaw, which made it possible for an attacker to craft a link to a specific web page and trigger execution of malicious JavaScript code. Such JavaScript code would create an HTTP GET request to the cookies.php module, pretending to be the user in order to obtain the user's cookie.

Medium

Cross-Site Scripting

✓

Vulnerable Code:

26

<?php print_r(\$_POST); ?>

Function:

print_r

Vulnerable File:

\\vendor\\codeception\\codeception\\tests\\data\\app\\view\\index.php

Entry File:

\\vendor\\codeception\\codeception\\tests\\data\\app\\view\\index.php : 1

Exploit:

POST /vendor/codeception/codeception/tests/data/app/view/index.php HTTP/1.1
Host:
Accept-Encoding: identity
Connection: close
Content-Length: 49
Content-Type: application/x-www-form-urlencoded

someparam=%3Cscript%3Ealert%281%29%3C%2Fscript%3E

OWASP - A3

CWE-79

Show Data Flow

Example of Cross-Site Scripting vulnerability

Analysis also revealed that the modules in the \filebrowser directory contained a demo application. The demo application allowed performing core file management functions in the \root directory using a web interface. Source code analysis found numerous Arbitrary File Creation and Arbitrary File Modification vulnerabilities that could be exploited for unlimited copying and renaming of files in the \filebrowser directory. These vulnerabilities would also make it possible to deplete free space on the web server local disk, causing denial of service.

High

Arbitrary File Modification

✓

Vulnerable Code:

112

file_put_contents(\$dir . DIRECTORY_SEPARATOR . \$name, "");

Function:

file_put_contents

Vulnerable File:

\\vendor\\vakata\\jstree\\demo\\filebrowser\\index.php

Entry File:

\\vendor\\vakata\\jstree\\demo\\filebrowser\\index.php : 1

Exploit:

GET /vendor/vakata/jstree/demo/filebrowser/index.php?id=93513789000&operation=create_nodes&text=%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2Ftmp%2Ftest.php&type=file HTTP/1.1
Host:
Accept-Encoding: identity
Connection: close
Condition:
(!preg_match('([^\a-zA-Z-_0-9.]+)ui', \$_GET['text']))
realpath('\\\\vendor\\vakata\\jstree\\demo\\filebrowser\\data\\root\\' .
trim(str_replace('/', '\\', \$_GET['id']), '\\'))
strlen(\$_GET['text'])

OWASP - A1

CWE-73

Show Data Flow

Example of Arbitrary File Modification vulnerability

High

Arbitrary File Creation

✓

Vulnerable Code:

112

file_put_contents(\$dir . DIRECTORY_SEPARATOR . \$name, "");

Function:

file_put_contents

Vulnerable File:

\\vendor\\vakata\\jstree\\demo\\filebrowser\\index.php

Entry File:

\\vendor\\vakata\\jstree\\demo\\filebrowser\\index.php : 1

Exploit:

GET /vendor/vakata/jstree/demo/filebrowser/index.php?id=%23&operation=create_nodes&text=%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2Ftmp%2Ftest.php&type=file HTTP/1.1
Host:
Accept-Encoding: identity
Connection: close
Condition:
(!preg_match('([^\a-zA-Z-_0-9.]+)ui', \$_GET['text']))
strlen(\$_GET['text'])

OWASP - A1

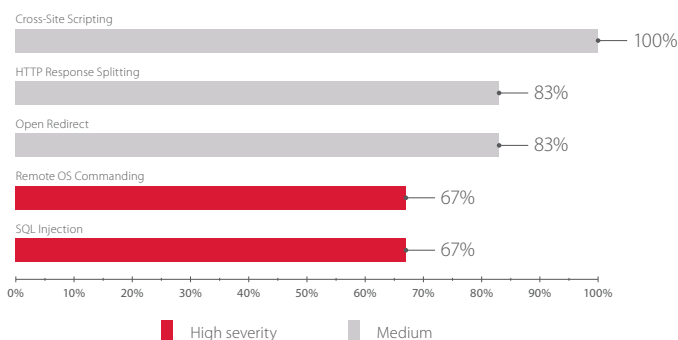
CWE-73

Show Data Flow

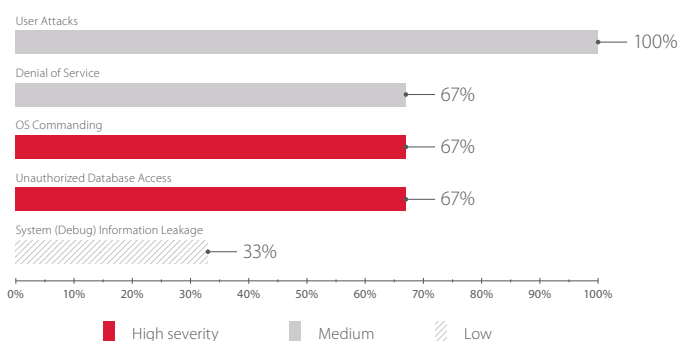
Example of Arbitrary File Creation vulnerability

3.5.2. Government

All government web applications tested by Positive Technologies contained vulnerabilities that facilitate attacks against users. Users of government web applications tend to not be security-savvy, which makes them more likely to fall for fraud.



Most common vulnerabilities in government web applications (percentage of tested web applications)



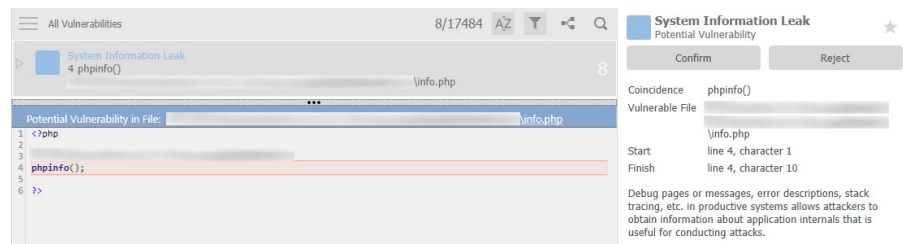
Most common threats for government web applications (percentage of tested web applications)

Example: Security assessment of a web application for a local government revealed SQL Injection, a critical vulnerability that, if exploited, would allow obtaining sensitive information from a database.

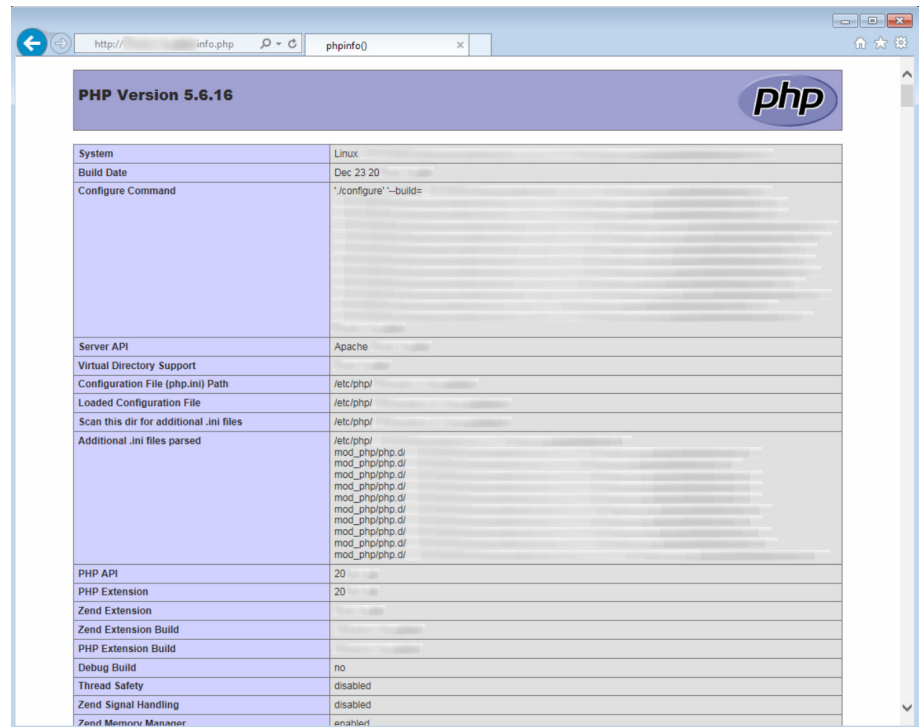
The screenshot shows a security tool interface with a list of vulnerabilities on the left and a detailed view of a SQL Injection vulnerability on the right. The vulnerability is titled 'SQL Injection' and is classified as a 'High Level Vulnerability'. The source code on the left shows a file named 'error.php' with a line of code that constructs a SQL query: `mysql_query("INSERT INTO 'a_support' ('SUPPORT_USER', 'TYPE', 'DATE_CREATED', 'NAME', 'EMAIL', 'LINK') VALUES ('" . $USER->GetID() . "', 4, NOW(), '" . $NAME . "', '" . $EMAIL . "', '" . $POST['URL'] . "'))");`. The right pane shows the request details and a data flow diagram illustrating the flow of data from the user input to the database query.

Data flow diagram for SQL Injection

Further source code analysis showed that any attacker could access the file info.php, which contains details about the system configuration. The vulnerability was manually verified by accessing the file contents. This information helps an attacker to plan and conduct additional attacks on the application.

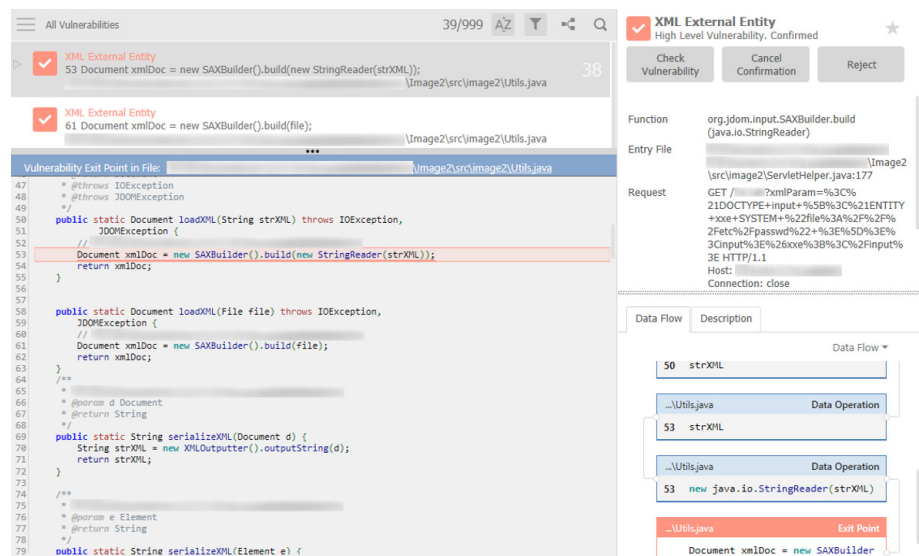


Example of Debug Information Leakage



Example of exploiting Debug Information Leakage

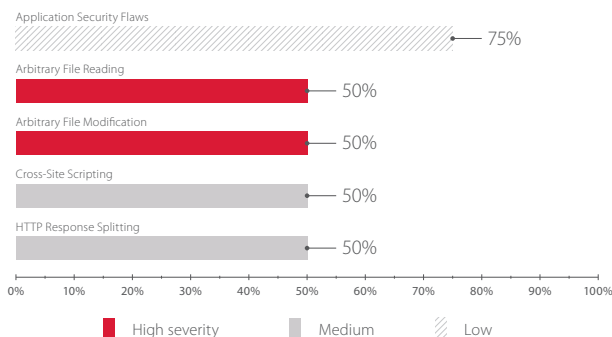
In another case, analysis of web application code revealed a critical XML External Entity vulnerability.



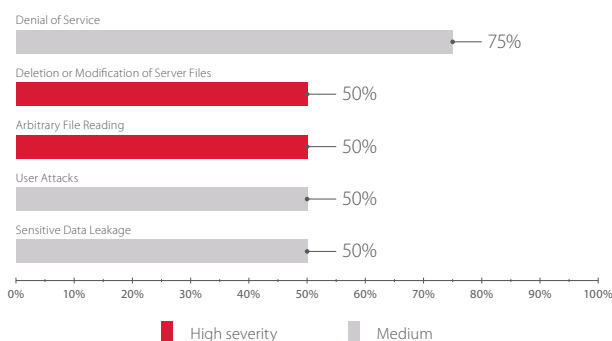
Data flow diagram for XML External Entity

3.5.3. E-commerce

Denial of service is especially threatening for e-commerce web applications, because any downtime means missed business and lost customers. High-profile e-commerce web applications receive large amounts of daily visits, increasing the motivation for attackers to find vulnerabilities to turn against users.

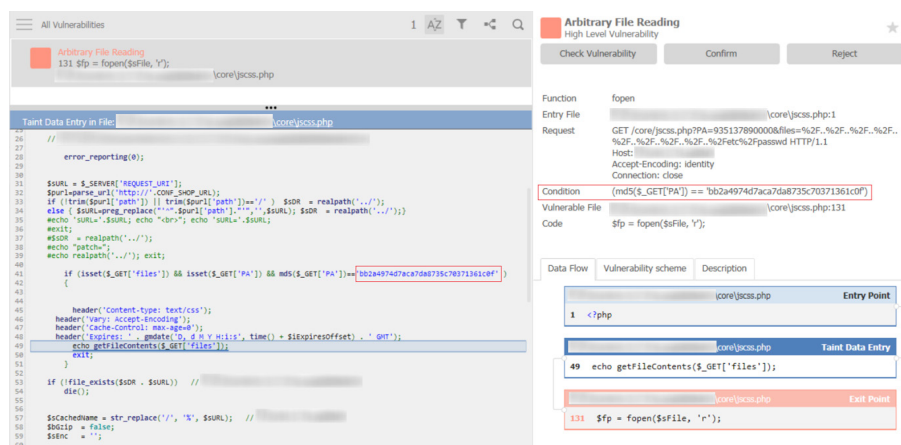


Most common e-commerce vulnerabilities (percentage of tested web applications)



Most common threats for e-commerce (percentage of tested web applications)

Example: When testing a content management platform for e-commerce, we found a critical Arbitrary File Reading vulnerability. It was impossible to use a test HTTP request to verify this vulnerability, because exploiting it required that the attacker be logged in. The conditions for vulnerability exploitation include the MD5 hash of the required password, which was found during source code analysis. The MD5 hash can be used to bruteforce the password, log in to the system, and successfully exploit this vulnerability. This case is a typical example of the danger of undeclared features left by web application developers.



Data flow diagram for Arbitrary File Reading

CONCLUSION

The results of source code analysis speak for themselves: this approach enables detecting a large number of vulnerabilities of various severity levels. Such an ability is critical for increasing the end-product security of web applications. Automated tools for source code analysis should be used at multiple stages of development, because analyzers are much quicker than manual analysis.

Merely detecting vulnerabilities, of course, is not enough: developers have to make fixes to code and roll them out to production systems. Any delay in remediation means more opportunities for attackers. Therefore, effective protection requires a multipronged approach built on periodic white-box security assessment of web applications including automated tools, complemented by proactive protection such as a web application firewall (WAF) to detect and prevent attacks against web applications.

About Positive Technologies

Positive Technologies is a leading global provider of enterprise security solutions for vulnerability and compliance management, incident and threat analysis, and application protection. Commitment to clients and research has earned Positive Technologies a reputation as one of the foremost authorities on Industrial Control System, Banking, Telecom, Web Application, and ERP security, supported by recognition from the analyst community. Learn more about Positive Technologies at ptsecurity.com.

© 2018 Positive Technologies. Positive Technologies and the Positive Technologies logo are trademarks or registered trademarks of Positive Technologies. All other trademarks mentioned herein are the property of their respective owners.