Scientific Research Publishing

Communications and Network, 2021, 13, 36-49
https://www.scirp.org/journal/cn
ISSN Online: 1947-3826
ISSN Print: 1949-2421

Analysis and Evaluation of Performance Related to Java and PHP Security Codes

Fontaine Rafamantanantsoa*, Rabetafika Louis Haja, Randrianomenjanahary Lala Ferdinand

University of Fianarantsoa, Fianarantsoa, Madagascar
Email: *fontainerafamant@yahoo.fr

How to cite this paper: Rafamantanantsoa, F., Haja, R.L. and Ferdinand, R.L. (2021) Analysis and Evaluation of Performance Related to Java and PHP Security Codes. *Communications and Network*, **13**, 36-49.
https://doi.org/10.4236/cn.2021.131004

Received: January 15, 2021
Accepted: February 23, 2021
Published: February 26, 2021

Abstract

In recent years, Internet exposure of applications continuously engenders new forms threats that can endanger the security of the entire system and raises many performance issues related to code security. The safety of information systems becomes essential. This is why the performance linked to security codes is of importance essential in the security systems of all companies. Indeed, as contribution, to carry out measurements, it appropriates tools that are the JMH tool (Java Microbenchmark Harness) and the PHP Benchmark script tool which include unsecure java and PHP codes and secured against SQL (Structured Query Language) injection, XSS (Cross Site Scripting) *i.e.*, using prepared requests, stored procedures, validation of input from white lists, reinforcement of minimum privilege, when sending requests from the last ones to MySQL databases and Postgresql. We recover the times of response to his requests. From java codes and PHP (Hypertext Preprocessor) secure, we also retrieve the response time for requests to databases MySQL and PostgresqL data. We then obtain the curves and interpretations comparing performance related to security and non-security of codes. The goal is to analyze and evaluate the performance comparing secure Java and PHP code against unsecure java and PHP code using MySQL and Postgresql databases. In Section 1, we presented the performance of the code Java and PHP. The configuration of the experiments and the experimental results are discussed in Sections 2 and 3, respectively. Use of suitable tool which is the JMH tool and the PHP Benchmark script tool, we have developed in Java 1.8 and PHP 7.4 secure and non-secure codes that send the queries to the MySQL or Postgresql database to carry out the measurements which led to the conclusion that the insecure PHP and Java codes are faster in terms of response time compared to the PHP and Java secure codes as the number of tables linked to the query increases because the blocking times of SQL injection and XSS preventions linked to its secure codes are increasing.

Keywords

Applications, Attacks, XSS, Security, Java, PHP, Performances

1. Introduction

The web has become increasingly popular. The number of internet users continues to increase therefore the number of sources malicious or malware and hacking becomes considerable. This is why the security codes such as Java codes and PHP codes are of importance primordial. Various studies [1] have proposed the safety of web applications: analysis, modeling and detection of learning attacks automatic, work [2] presented the modeling and automatic classification of security information. [3] [4] [5] [6] have studied computer hacking, testing security, penetration testing and basic testing. In the studies [7]-[18], authors showed hacking and cyber security, quantitative of computer security, security holes discovery, web hacking, systems information performance and hardening java security.

Few of the studies on performance related to code security. This is why researchers are urged to analysis and evaluation of related performance security codes. The methodology is based on the recovery time when launching codes that perform requests to the databases data, then we recover the time after executing its codes to calculate response times and obtain the curves related to security or not codes. The expected results are to have the results of analyzes on the impact of codes compared to response times to using the results of the measurements carried out making it possible to draw the curves for each type of database. We are not satisfied with the measurements of performance of a code, by calculating the difference between the start of the process and its end, because it was necessary to modify the code of the method to add the measurement elements, the method is not the same as what was initially predicted. it is therefore advisable to use a suitable tool which is the JMH tool (Java Microbenchmark Harness) and the PHP tool Benchmark script which include java codes and PHP secure or not.

2. Code Performance

The security of information systems (ISS) or more simply computer security is all the technical, organizational, legal and human resources necessary for the implementation of means aimed at preventing unauthorized use, misuse, modification or misappropriation of the information system.

To measure the execution time of a method reliably, we isolate the execution of the method to be measured, then by stimulating the JVM with the code which one wishes to measure (one often speaks about Warmup). This step allows the JVM time to optimize the code if necessary. Then, you have to run the code to be measured repeatedly (also called iterations). It is during these iterations that the

measurements will be carried out so as to obtain a reliable average execution time. Obviously, it is preferable to carry out these measures with the least intrusive means possible. That is, by modifying the code of the method to be tested as little as possible.

To carry out measurements taking into account the previous points, we use a suitable tool which is the JMH tool and the PHP Benchmark script tool.

Secure codes are codes that block the following vulnerabilities:

- Block SQL injection using prepared queries, using stored procedures, performing validations on whitelist entries, avoiding special characters in SQL queries, strengthening minimum privilege.
- XSS (Cross Site Scripting) prevention by avoiding the insertion of data in the codes. The system must meet the following conditions:

A computer where is installed:

- A MySQL or Postgresql database.
- A secure and unsecure java 1.8 or PHP 7.4 code that makes requests to the MySQL or Postgresql database.
- An apache web server 2.

Eight measurements are used to measure the performance related to code security:

- Secure code written in java 1.8 executing queries to the MySQL database.
- Secure code written in PHP 7.4 executing queries to the MySQL database.
- Insecure code written in java 1.8 executing queries to the MySQL database.
- Insecure code writes PHP 7.4 executing queries to the MySQL database.
- Secure code written in java 1.8 executing queries to the Postgresql database
- Secure code written in PHP 7.4 executing queries to the Postgresql database.
- Insecure code written in java 1.8 executing queries to the Postgresql database.
- Insecure code written in PHP 7.4 executing queries to the Postgresql database.

2.1. Methodology for Performance Analysis

The main steps in performance evaluation are:

- Isolate running the method at measure, then by stimulating the JVM with the code that we are going to measure.
- Execute the code to be measured repeatedly (also called iterations). It is during these iterations that the measurements will be carried out so as to obtain a reliable average execution time.

Figure 1 shows the experimental Java Mysql setup.

Figure 1 shows that the time t is the average response time for a request sent from a secure and non-secure Java code to the MySQL database.

Figure 2 shows the experimental Java Postgresql setup.

Figure 2 shows that the time t is the average response time for a request sent from a secure and non-secure Java code to the Postgresql database.

Figure 3 shows the experimental PHP MySQL configuration.

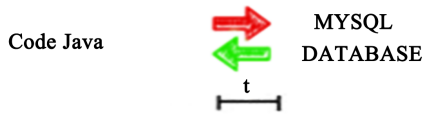

Figure 1. Java MySQL experimental setup.

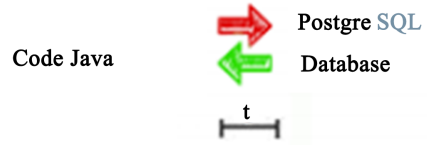

Figure 2. Java Postgresql experimental setup.

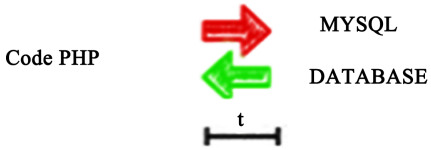

Figure 3. Experimental PHP MySQL configuration.

Figure 3 shows that the time t is the average response time for a request sent from a secure and non-secure PHP code to the MySQL database.

Figure 4 shows the experimental PHP Postgresql setup.

Figure 4 shows that the time t is the average response time for a request sent from a secure and non-secure PHP code to the Postgresql database.

2.2. Architecture Related to Code Security

SQL injection attack is very common because it is quick to set up, can cause irreversible damage to your database or, if used in a more subtle way, it allows passwords to be recovered discreetly and identifiers. The hacker hijacks your request by injecting code into the form fields: hence the term SQL injection.

Java security is provided at two main levels: at compile time and at interpretation. Although the security provided at the compiler level is very interesting. On the other hand, the security provided at the level of the Java virtual machine is directly affected by the three articles synthesized. We must escape our values for render inert and harmless. The magic_quotes are part of a PHP directive aimed at ensuring the security of SQL queries without its knowledge by systematically escaping the following characters:

- double quotes ";
- the slashes /;
- NULL characters;
- single quotes '.

3. Configuration of Experiments

Figure 5 shows the architecture related to code security.

The characteristics of the system used are shown in **Table 1**.

Table 1 summarizes the hardware and software used for the experiments.

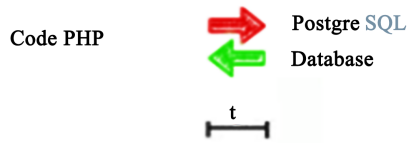

Figure 4. PHP Postgresql experimental setup.

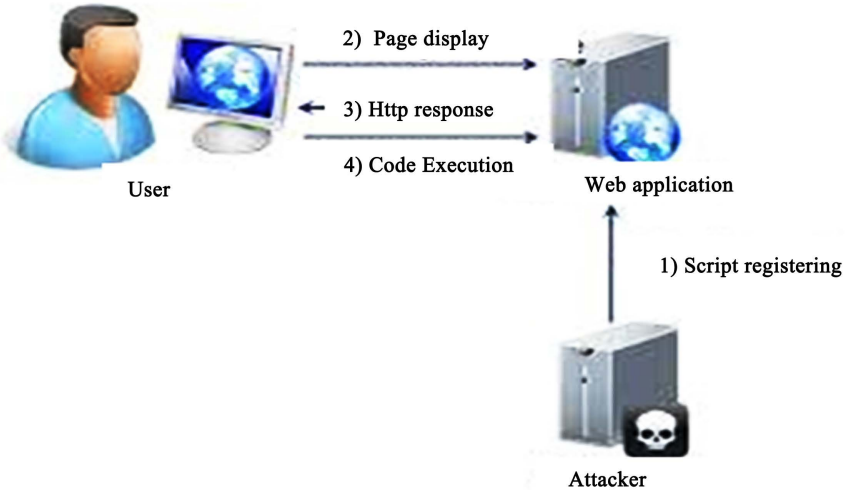

Figure 5. Architecture related to code security.

Table 1. Characteristics of the hardware and software used in the experiments.

Mark	DELL Vostro 3400
CPU	2.40 GHz
NIC	RealTek Semiconductor
RAM	1 GB
Operating System	Ubuntu 16.04
Java	1.8
PHP	7.4
MySQL	14.14
Postgresql	9.6

The operating system Ubuntu 16.04, Java 1.8, PHP 7.4, MySQL and Postgresql 9.6 were installed on the same machine without any modification material.

Presentation of measurement methods Java and PHP are programming in object language, multi-paradigm and multiplatform. It promotes structured and object-oriented imperative programming.

We developed in Java 1.8 and PHP 7.4. Its codes send queries to the MySQL or Postgresql database. Its queries make joins on tables and after responses, the execution time is calculated.

To escape a character, this directive adds backslashes to strings that pass through the PHP script. In fact, it plays the same role as the addslashes () func-

tion.

Extract from Java source code

```
package org.sample;
import
org.openjdk.jmh.annotations.Benchmark;
import java.util.concurrent.TimeUnit;
import
org.openjdk.jmh.annotations.BenchmarkMode;
import org.openjdk.jmh.annotations.Fork;
import
org.openjdk.jmh.annotations.Measurement;
import org.openjdk.jmh.annotations.Mode;
import
org.openjdk.jmh.annotations.OutputTimeUnit;
import org.openjdk.jmh.annotations.Scope;
import org.openjdk.jmh.annotations.State;
import
org.openjdk.jmh.annotations.Warmup;
import java.sql.*;
@Warmup(iterations = 3, time = 1000, TimeUnit = TimeUnit.MILLISECONDS)
@Measurement(iterations = 3, time = 1000, TimeUnit = Time-
Unit.MILLISECONDS)
@BenchmarkMode (Mode.AverageTime)
@OutputTimeUnit (TimeUnit.MILLISECONDS)
@Fork (1)
@State (Scope.Benchmark)
public class MyBenchmark {
@Benchmark
public void testMethod () {Connection connection = null; PreparedStatement
pstmt = null; String sql = "select usr.username as
name, usr.firstname as firstname, usr.email
as email, usr.id as id from group_member,
usr, group, usr_account_preference,
usr_activity_preference, usr_custom_layout
where ((group_member.groupe = 1) AND
(group_member.member = usr.id) AND
(group_member.groupe = group.id) AND
(group_member.member = usr.id) AND
(usr.id = usr_account_preference.usr) AND
(usr.id = usr_activity_preference.usr) AND
(usr.id = usr_custom_layout.usr))";
try
{
```

```
connection =
DriverManager.getConnection ("jdbc: postgresql: //localhost:
5432/BDmesure", "postgres", "postgres");
pstmt = connection.prepareStatement (sql);
ResultSet rs = pstmt.executeQuery ();
rs.close ();
pstmt.close ();
connection.close ();
}
catch (Exception e) {
e.printStackTrace ();
} finally {
try {
pstmt.close ();
connection.close ();
} catch (Exception e) {
e.printStackTrace ();
}
}
}
```

Extract from PHP source code

```
public static function run ($ echo = true) {$ db = new
PDO ('mysql: host = localhost; dbname = BDmesure', "root", "admin");
$sql = $db->prepare ("select usr.username as name, usr.firstname as
firstname, usr.email as email, usr.id as id from group_member, usr, 'group'
where ((group_member.group = 1) AND (group_member.member = usr.id)
AND (group_member.group = 'group'.id) AND (group_member.member =
usr.id))");
$sql->execute ();
$rows = $sql->fetchAll (PDO :: FETCH_ASSOC);
$total = 0;
$server =
(isset($_SERVER['SERVER_NAME'])? $_SERVER ['SERVER_NAME']: '?').
'@'. (isset ($_SERVER['SERVER_ADDR'])? $_SERVER ['SERVER_ADDR']: '?');
$methods = get_class_methods ('benchmark');
$line = str_pad ("-", 38, "-");
$return = "<pre> $ line \ n |" .str_pad ("PHP BENCHMARK SCRIPT", 36, "",
STR_PAD_BOTH). "| \ n $ line \ nStart:" .date ("Ymd H: i: s ")." \ nServer:
$ server \ nPHP version: ".PHP_VERSION." \ nPlatform: ".PHP_OS. "\ n $ line \
n";
for each ($ methods as $ method) {if (preg_match ('/ ^ test_ /',
$method)) {
$total + = $result = self ::
$method ();
```

```
$return. =
str_pad ($method, 25). ":". $result. "sec. \ n";
}
}
$return. = str_pad ("-",
38, "-"). "\n" .str_pad ("Total time:", 25). ":".
$total. "sec. </pre>";
if ($echo) echo $return;
return $return;
}
}
benchmark :: run ();
```

Notes:

"Warmup (itérations = 3, time = 1000, timeUnit = time-Unit.MILLISECONDS)"

Iterations: Warmup iterations are intended to bring the JVM into a steady state (e.g., execute all applicable just-in-time compilations). We can configure the code being benchmarked to run a specified number of "warmup" iterations first before any measurement actually begins. This allows the JVM optimizations to take place before we are actually going to benchmark it. We should also measure it after some warmup iterations to simulate "real" production conditions. Each iteration takes a defined amount of time (typically 1 s), during which the framework repeatedly calls the method annotated with @Benchmark (a single invocation in JMH parlor) and records all configured performance counters (e.g., throughput, execution time, latency). That's why, we set the value 3 to the iteration parameter

time: Sets how much time every warmup iteration will take in specified timeUnit. That's why, we set the value 1000 to the time parameter

time Unit: Almost every possible unit starting from TimeUnit.NANOSECONDS up to TimeUnit.DAYS (SI standard). That's why, we set the value TimeUnit.MILLISECONDS to the timeUnit parameter to have more precision of the results.

4. Experimental Results

Using as MySQL Database

Table 2 shows the query times as a function of the number of tables in the database for secure and non-secure java code.

Figure 6 shows the curves of the time of queries as a function of the number of tables in the MySQL database for secure and non-secure java code approximate a quadratic curve which is given by the formula simulink:

$$y = 0.31004 * x^2 + 0.18869 * x + 14.05633 \quad (1)$$

Figure 6 shows that in the interval of time [15.2; 18.4], the deadline is almost same for secure java code and code unsecure java using as base MySQL data. By

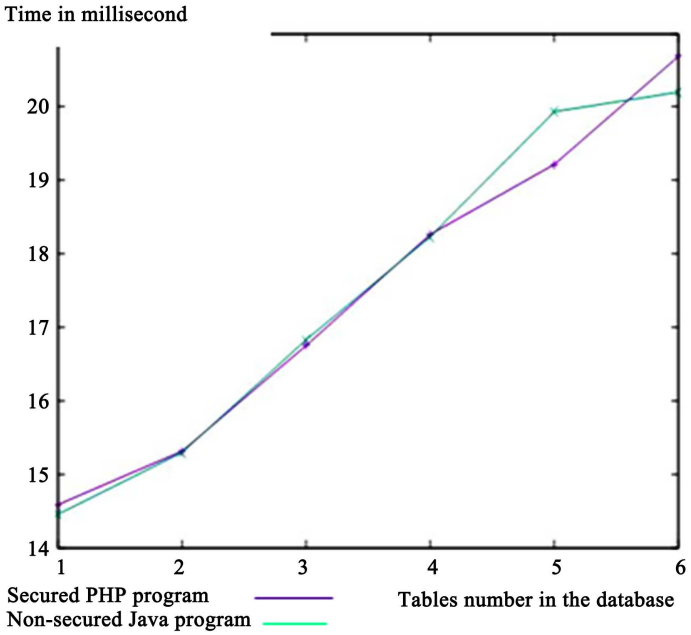

Figure 6. Curve of query time as a function of number of tables in the database for secure and non-secure java code.

Table 2. Query time depending on the number of tables in the database for the secure and unsecured java code.

Non-secured Java Code		Secured Java Code	
Table Number	Time t in millisecond	Table Number	Time t in millisecond
1	14.452	1	14.583
2	15.295	2	15.312
3	16.830	3	16.752
4	18.235	4	18.265
5	19.936	5	19.220
6	20.203	6	20.692

increasing the number of tables used by the query, the java code secure considerably loses its performance, which means that the queries according to the number of tables in the database for secure java code slow down as the number of tables in the database increases. This can be explained by increased access time to tables caused by the increase in selection of the lines linked to each table because the sizes of the tables slow down its query operations.

Table 3 shows the query times as a function of the number of tables in the database for secure and non-secure PHP code.

Figure 7 shows the curves of the time of queries as a function of the number of tables in the MySQL database for secure and non-secure PHP code approximate a quadratic curve which is given by the formula simulink:

$$y = 0.05396 * x^2 + 0.10211 * x + 1.93333 \quad (2)$$

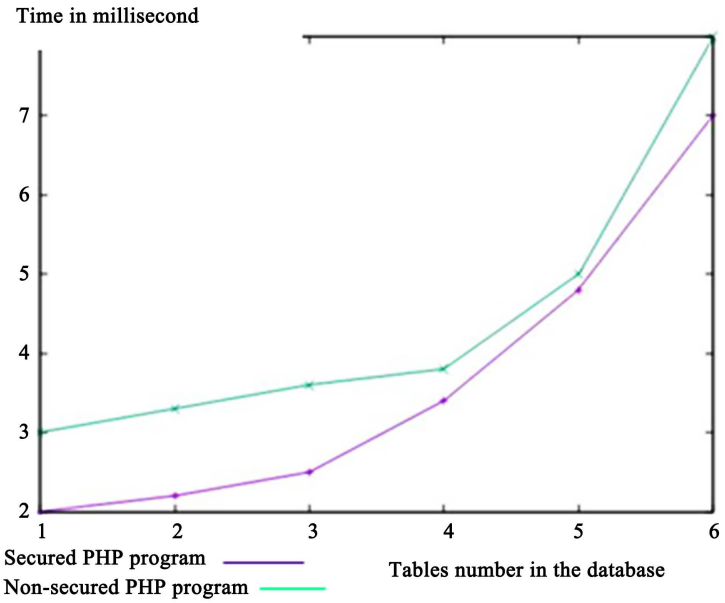

Figure 7. Curve of query time as a function of number of tables in the database for secure and non-secure PHP code.

Table 3. Request time according to the number of tables in the database for secure and non-secure PHP code.

Non-secured PHP Code		Secured PHP Code	
Table Number	Time t in millisecond	Table Number	Time t in millisecond
1	2	1	3
2	2.2	2	3.3
3	2.5	3	3.6
4	3.4	4	3.8
5	3.8	5	5
6	7	6	8

Figure 7 shows that query times as a function of the number of tables in the database for secure and unsafe PHP code using as base MySQL data slows down as the number of tables in the database increases. This can be explained by the increase in the access time to the tables caused by the increase in the selection times of the rows linked to the queries of each table because the sizes of the tables slow down its query operations.

Using as Postgresql Database

Table 4 shows query times as a function of the number of tables in the database for secure and non-secure java code.

Figure 8 shows the curves of the query time as a function of the number of tables in the Postgresql database for secure and non-secure java code approximate a quadratic curve which is given by the simulink formula:

$$y = 0.09484 * x^2 + 1.26220 * x + 4.24133 \quad (3)$$

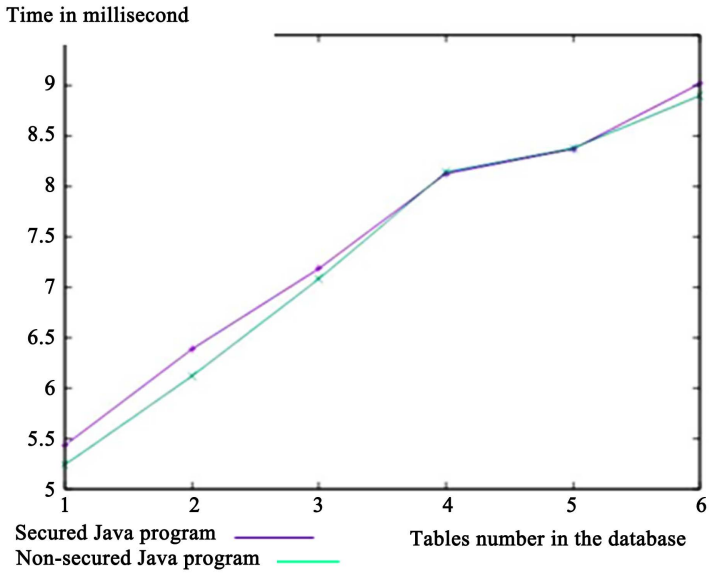

Figure 8. Curve of query time as a function of number of tables in the database for secure and non-secure java code.

Table 4. Query time depending on the number of tables in the database for secure and non-secure java code.

Non-secured Java Code		Secured Java Code	
Table Number	Time t in millisecond	Table Number	Time t in millisecond
1	5.234	1	5.428
2	6.114	2	6.382
3	7.079	3	7.182
4	8.139	4	8.122
5	8.374	5	8.366
6	8.896	6	9.015

Figure 8 shows that in the interval of time [7, 8; 8.5], the deadline is almost same for secure java code and code unsecure java using as base Postgresql data. By increasing the number of tables used by the query, the java code secure considerably loses its performance, which means that the queries according to the number of tables in the database for secure java code slow down as the number of tables in the database increases. This can be explained by increased access time to tables caused by the increase in selection of the lines linked to each table because the sizes of the tables slow down its query operations.

Table 5 shows query times as a function of the number of tables in the database for secure and non-secure PHP code.

Figure 9 shows the curves of the query time as a function of the number of tables in the Postgresql database for secure and non-secure PHP code approximate a quadratic curve which is given by the formula simulink:

$$y = 0.30873 * x^2 + 0.92513 * x + 2.33333 \quad (4)$$

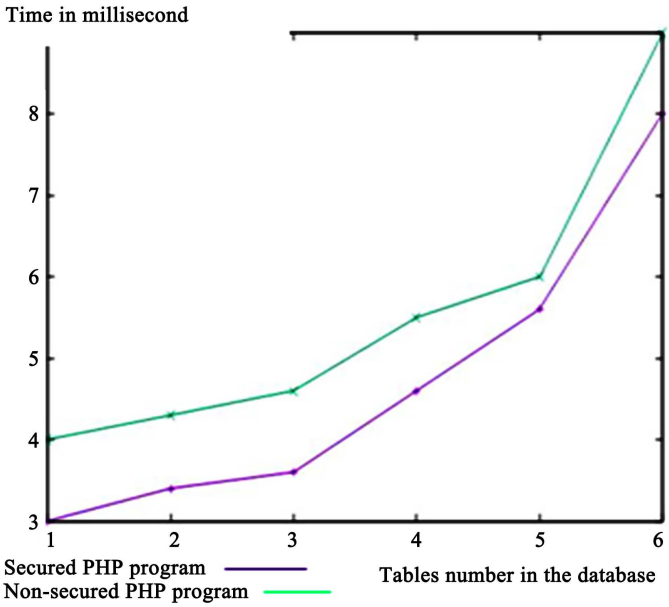

Figure 9. Curve of query time as a function of number of tables in the database for secure and non-secure PHP code.

Table 5. Request time according to the number of tables in the database for secure and non-secure PHP code.

Non-secured PHP Code		Secured PHP Code	
Table Number	Time t in millisecond	Table Number	Time t in millisecond
1	3	1	4
2	3.4	2	4.3
3	3.6	3	4.6
4	4.6	4	5.5
5	5.6	5	6
6	8	6	9

Figure 9 shows that query times as a function of the number of tables in the database for secure and unsafe PHP code using as base Postgresql data slows down as the number of tables in the database increases. This can be explained by the increase in the access time to the tables caused by the increase in the selection times of the rows linked to the queries of each table because the sizes of the tables slow down its query operations.

Secured Java code

We run the JMH (Java Microbench Harnessmark) tool code repeatedly (also known as iterations) containing the secure connection and queries to the database. It is during these iterations that measurements will be made in such a way as to obtain an average of reliable query execution time to the database tables. Because the codes are secure, they block vulnerabilities like SQL injections by using prepared queries, using stored procedures, validating whitelist ens entered,

avoiding special characters in SQL queries, and strengthening minimum privilege. Codes do XSS (Cross Site Scripting) preventions by avoiding the insertion of data into codes. After running its codes contained in the JMH tool, we get the response times in milliseconds based on the fixed table number of its queries.

Non-secured Java code

We run the codes of the JMH (Java Microbench harnessmark) tool repeatedly (also called iterations) containing the unsecured connection and queries to the database. It is during these iterations that measurements will be made in such a way as to obtain an average of reliable query execution time to the database tables. Because the codes are insecure, they don't block vulnerabilities like SQL injections, and don't do XSS (Cross Site Scripting) preventions, *i.e.* by running queries directly. After running its codes contained in the JMH tool, we get the response times in milliseconds based on the fixed table number of its queries.

Secured PHP code

We launch the codes of the Benchmark script PHP tool containing the secure connection and queries to the database. It is during the repeated launch that the measurements will be made in such a way as to obtain an average of reliable query execution time to the tables in the database. Because the codes are secure, they block vulnerabilities like SQL injections by using prepared queries, using stored procedures, validating whitelist ens entered, avoiding special characters in SQL queries, and strengthening minimum privilege. Codes do XSS (Cross Site Scripting) preventions by avoiding the insertion of data into codes. After launching its codes from the Benchmark script PHP tool, we get the response times in milliseconds based on the fixed table number of its queries.Code.

Non-secured PHP code

We launch the codes of the Benchmark script PHP tool containing the unsecured connection and queries to the database. It is during the repeated launch that the measurements will be made in such a way as to obtain an average of reliable query execution time to the tables in the database. Because the codes are insecure, they don't block vulnerabilities like SQL injections. Codes do not do XSS (Cross Site Scripting) preventions *i.e.* launch queries directly. After launching its codes from the Benchmark script PHP tool, we get the response times in milliseconds based on the table number of its queries.

5. Conclusion

This article presents the analysis and evaluation of performance related to both Java and PHP security codes. In Section 1, we presented the performance of the code. The configuration of the experiments and the experimental results are discussed in Sections 2 and 3, respectively. To carry out measurements, it is necessary to use a suitable tool which is the JMH tool (Java Microbenchmark Harness) and the PHP Benchmark script tool, we have developed in Java 1.8 and PHP 7.4 secure and non-secure codes that send the queries to the MySQL or Postgresql database. Its queries make joins on tables and after responses, the ex-

ecution time is calculated. The measurements led to the conclusion that the insecure codes are faster in terms of response time compared to the secure codes as the number of tables linked to the query increases because the blocking times of SQL injection and XSS preventions linked to its secure codes are increasing.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Makiou, A. (2016) Web Application Security: Analyzing, Modeling, and Detecting Machine Learning Attacks.

[2] Benali, F. (2009) Modeling and Automatic Classification of Security Information.

[3] Hall, G. and Watson, E. (2016) Computer Hacking, Security Testing Penetration Testing and Basic Testing.

[4] Engebretson, P. (2011) The Basics of Hacking and Penetration Testing: Ethical Hacking and Penetration Testing Made Easy. Elsevier, Amsterdam. https://doi.org/10.1016/B978-1-59749-655-1.00001-5

[5] Weidman, G. (2018) Penetration Testing: A Hands-On Introduction to Hacking.

[6] Allsopp, W. (2018) Advanced Penetration Testing: Hacking the World's Most Secure Networks. John Wiley & Sons, Inc., Hoboken. https://doi.org/10.1002/9781119367741

[7] Sahay, U. (2013) Hack-x-Crypt. A Straight Forward Guide towards Ethical Hacking and Cyber Security.

[8] Stuttard, D. and Pinto, M. (2018) The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws. 2nd Edition.

[9] Vache-Marconato, G. (2009) Quantitative Evaluation of Computer Security: Approach by Vulnerabilities.

[10] Anley, C., Heasman, J., Lindner, F. and Richarte, G. (2018) The Shellcoder's Handbook: Discovering and Exploiting Security Holes. 2nd Edition.

[11] Yaworski, P. (2018) Web Hacking 101.

[12] Elien, F. (2013) The Performance of Information Systems.

[13] Yende, R. (2018) Safety Course Support.

[14] Mazri, C. (2015) Safety Management by Performance Indicators.

[15] El Hamzaoui, M., Bensalah, F. and Rachid, H. (2017) Contribution of the Management of Computer Networks to the Performance of Business Management: A New Theoretical Model for Effective Business Management.

[16] Holzinger, P.A. (2019) A Systematic Analysis and Hardening of the Java Security Architecture.

[17] Kahanwal, B. (2013) Performance Evaluation of Java File Security System.

[18] Babatunde, J. (2015) Evaluating the Impact of Security Measures on Performance of Secure Web Applications Hosted on Virtualized Platforms.