Static Code Analysis Tool for Laravel Framework Based Web Application

1st Ranindya Paramitha
School of Electrical Engineering and Informatics
Institut Teknologi Bandung
Bandung, Indonesia
ranindyananinparamitha@gmail.com

2nd Yudistira Dwi Wardhana Asnar
School of Electrical Engineering and Informatics
Institut Teknologi Bandung
Bandung, Indonesia
yudis@informatika.org

2021 International Conference on Data and Software Engineering (ICoDSE) | 978-1-6654-9453-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICoDSE53690.2021.9648519

***Abstract*—To increase and maintain web application security, developers could use some different methods, one of them is static code analysis. This method could find security vulnerabilities inside a source code without the need of running the program. It could also be automated by using tools, which considered more efficient than manual reviews. One specific method which is commonly used in static code analysis is taint analysis. Taint analysis usually utilizes source code modeling to prepare the code for analysis process to detect any untrusted data flows into security sensitives computations. While this kind of analysis could be very helpful, static code analysis tool for Laravel-based web application is still quite rare, despite its popularity. Therefore, in this research, we want to know how static code (taint) analysis could be utilized to detect security vulnerabilities and how the projects (Laravel-based) should be modeled in order to facilitate this analysis. We then developed a static analysis tool, which models the application's source code using AST and dictionary to be used as the base of the taint analysis. The tool first parsed the route file of Laravel project to get a list of controller files. Each file in that list would be parsed in order to build the source code representation, before actually being analyzed using taint analysis method. The experiments was done using this tool shows that the tools (with taint analysis) could detect 13 security vulnerabilities from 6 Laravel-based projects with one False Negative. An ineffective sanitizer was the suspected cause of this False Negative. This also shows that proposed modeling technique could be helpful in facilitating taint analysis in Laravel-based projects. For future development and studies, this tool should be tested with more Laravel and even other framework based web application with a wider range of security vulnerabilities.**

***Index Terms*—security, vulnerabilities, static code analysis, Laravel, taint analysis**

I. Introduction

Statistic shows that one of three web applications had bad security level. In fact, according to Positive Technology, 67% of web applications had high level security vulnerabilities in 2018 [1]. One of the ways to increase web application's security level is by doing static code analysis. What static code analysis does is analyzing program source codes to find security vulnerabilities without executing the program itself. This kind of analysis could be done manually or by utilizing the help of tools, which are obviously called static code analysis tools.

There are several methods that are usually used in static code analysis. One of them is taint analysis. Taint analysis is a static code analysis method which utilizes data flow analysis to detect security vulnerabilities. Before the analysis process, the source code is usually being modeled into another representation in order to make the analysis process easier. The source code could be modeled into some different representations, depends on the analysis process itself.

One of popular web application development languages is PHP. There are many web application built using this language, and many of them are built using development frameworks based on it. In order to facilitate the need of maintaining their security, some static code analysis tools have been built for PHP based web applications. However, static code analysis tools to detect security vulnerabilities in web applications that based on PHP development frameworks are still quite rare, especially for Laravel framework.

Addressing this issue, this paper would propose about how to do taint analysis in Laravel-based web application. The taint analysis here would be focusing on detecting input injection type security vulnerabilities. To facilitate that discussion, this paper would also propose about how source code should be modeled from Laravel-based web application. In order to validate the proposed method, the end goal of this paper is to build a static code analysis tool that could actually do both of that action to detect security vulnerabilities in Laravel based web applications. After the tool is built, it would be tested to understand its performance, which would be also discussed in this paper.

II. Literature Study

A. Web Application Security Vulnerabilities

According to Cambridge Dictionary [2], vulnerability means the quality of being easily attacked. There are many kinds of web application security vulnerabilities. In 2019, CWE Mitre [3] published their version of top 25 most dangerous software errors. On the other side, OWASP [4] also have been releasing their versions of top 10 web application security risks periodically, which the most recent one was published in 2017.

Among all that security vulnerabilities, there is a type od security vulnerability which is caused by improper sanitation of untrusted data, which are user's inputs. This type of security vulnerability is usually called input injection, where user could exploit applications by injecting input to them. Input injection

978-1-6654-9453-3/21/$31.00 ©2021 IEEE

includes but is not limited to injections, Cross Site Scripting (XSS), and path/directory traversal.

a) Injections: This kind of security vulnerability happens when untrusted data are sent to command or query interpreter, especially the security sensitive ones. There are many kind of injections, such as SQL injection, OS injection, and command injection. Attacker could exploit this vulnerability to get unauthorized data, or even execute malicious commands in the application. Injection has been ranked first in OWASP Top 10 since 2010 until the latest version. CWE Mitre also has 9 different kinds of injections in their Top 25.

b) Cross Site Scripting (XSS): Cross Site Scripting or in short XSS is a vulnerability that an application has when untrusted data gets into the application and the application sends them to web browser without proper validation/ sanitation. Attacker could exploit this vulnerability to execute scripts in victim's browser to hijack their session or any other bad actions that could be harmful for the victim. This security vulnerability has been in OWASP Top 10 since their first released version in 2003. It is also ranked second in CWE Top 25.

c) Path/Directory Traversal: That is common for web applications to redirect and forward their user to other web sites or web pages. However, the implementation is sometimes not done well enough. They use untrusted data to determine the destination of redirects and forwards, without proper validation/ sanitation beforehand. Attacker could exploit this vulnerability to get forwarded into unauthorized pages or even redirect users to malware or scam sites. This vulnerability is included in OWASP Top 10 list in 2013, but they removed it in 2017.

B. PHP and Laravel

PHP stands for a PHP: Hypertext Preprocessor, a recursive acronym [5]. PHP is a programming language which is quite popular in web application development. This language supports programming in procedural and object-oriented paradigm. With features like interface and operator overloading, PHP is a dynamic language, which could determine called operation at run time.

Nowadays, developer tends to use development framework in developing web applications, because of its benefits. Framework itself is a group of integrated components which each component collaborates with each other to produce reusable architecture in the related application scope. As software engineering term, development framework is specifically a conceptual or real base where codes with general functionalities could be reused by developer or even user.

Laravel is a development framework for web application development which uses PHP as its language [6]. Laravel provides expressive syntax and a group of functionalities which help developer implements faster. According to Google Trends [7], Laravel have been quite popular in the last 5 years, compared to other PHP development frameworks, such as CakePHP, Symfony, CodeIgniter, and Zend.

C. Static Code Analysis Methods

Static code analysis is an analysis which is done on the source code of a software in order to determine dynamic execution properties without executing the program directly [8]. The main goal of this analysis is to identify as many code problems as it can automatically even before the program is fully implemented [9]. There are many methods of doing static code analysis. Two of them are control flow analysis and data flow analysis.

a) Control Flow Analysis: This method of analysis usually utilizes control flow graph. In control flow graph, vertexes represent each basic instruction block in a code and edges represent control transition from an instruction block to another [10]. The example of a simple control flow graph is shown in Figure 1.

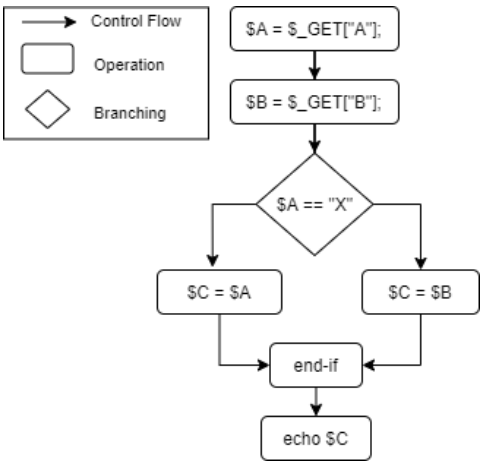

Fig. 1. Simple Control Flow Graph.

b) Data Flow Analysis: This method tracks values and their transitions inside program codes [11]. Data flow analysis could be represented by control flow graph or data flow graph during its process. An example of simple data flow graph is shown in Figure 2. One application of data flow analysis is taint analysis. Taint analysis is a method which automatically detects integrity violations inside program codes. Integrity violation happens when untrusted data flows into security sensitive operations/ computations [12]. In taint analysis, there are 3 crucial terms: source, sanitizer, and sink [13].

- Source is a method which return tainted data or untrusted data which usually come from users.
- Sanitizer is a method which manipulates its inputs and produces taint-free values.
- Sink is a pair of security sensitive method and parameters of that method which are vulnerable to attack from tainted data.

III. Implementation Overview

A. Taint Analysis in Laravel Based Application

Taint analysis in Laravel-based web application should support object oriented paradigm program codes. Taint analysis in

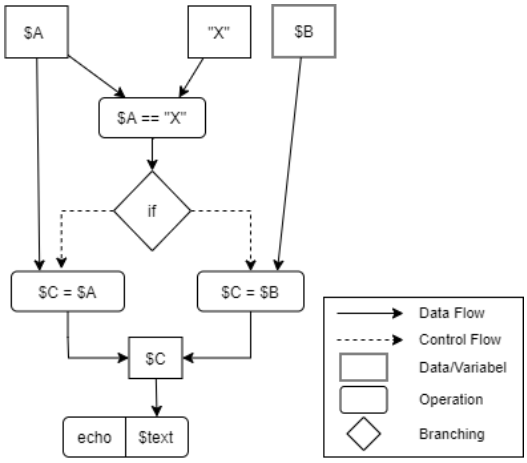

Fig. 2. Simple Data Flow Graph.

object oriented codes is quite different with the one in codes with procedural paradigm. This is because the structure of the program is generally different, with utilization of concepts like class and interface.

```php
<?php
  interface User {
    public function getDataFromDB($userId, $name);
  }
  // Student implements User interface
  class Student implements User {
    public function getDataFromDB($userId) {
      $sql = "SELECT * FROM Users WHERE UserId = ";
      $sql .= $userId;          // Query includes tainted data
      $mysqli = mysqli_connect("H", "U", "P", "D"); // SINK
      $res = mysqli_query($mysqli, $sql);           // SINK
      while ($row = mysqli_fetch_assoc($res)){
        echo implode(" ", $row);  // Printing query result
      }
    }
  }
  // Guest implements User interface
  class Guest implements User {
    public function getDataFromDB($userId) {
      echo "Sorry, please register yourself first.";
    }
  }
  $input_role = $_GET["Role"];     // SOURCE
  $input_id = $_GET["UserId"];   // SOURCE
  if ($input_role == "STUDENT"){ // Role = student
    $person = new Student();
  } else {
    $person = new Guest();
  }
  $person->getDataFromDB($input_id);
?>
```

Fig. 3. PHP Object Oriented Code Sample.

PHP is a dynamic language, which means called methods could be determined at run times. A sample case which should be able to explain more about this could be observed in Figure 3. This program has two classes, “Student” and “Guest”, both implement one interface with a method. Later in that program, it creates an object, which could be either “Student” or “Guest”. The object calls the method. Here, which method that would actually be called would be determined at run times, when the program actually knows what class is the object itself.

Besides of supporting object oriented program, taint analysis in Laravel based web application should pay attention to Laravel-based project structure. Laravel based project has a general directory structure which is shown in Figure 4. Laravel-based project in general has route files, stored in folder routes. These files store a list of endpoints and the controller files that they call. Controller files could be found in the folder Controller inside App and Http. These controllers do many functionalities for the application, including processing user’s input and calling other files, such as views files. Those views files are located in folder views inside resources. Looking at this flow, taint analysis in Laravel-based project should get the list of controller files first, before building source code representation for each of that file. After that, the analysis should be done using each of that model, with a possibility to switch to other files if called.

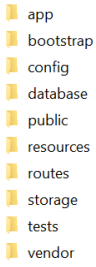

Fig. 4. Laravel-Based Project General Directory Structure.

B. Functional Requirements

The static code analysis tool built has 3 main functionalities, as drawn in use case diagram in Figure 5. First, user could upload their Laravel-based project or application into the tool in .zip format. The tool would do static code analysis on the project uploaded and then show the result back to the user.

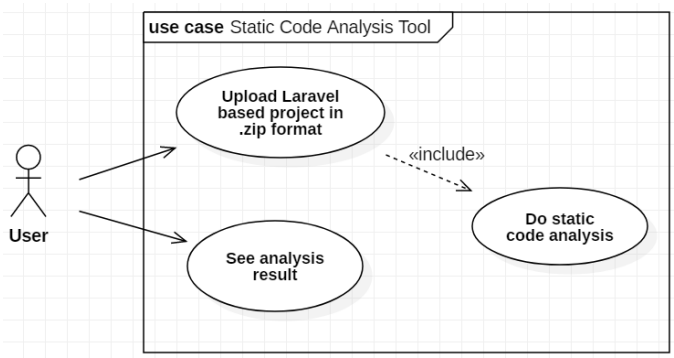

Fig. 5. Use Case Diagram.

C. Tool Workflow and Architecture

In order to detect security vulnerabilities in Laravel-based projects, after getting the project from user, the tool would

parse the route file of the project to get a list of controller files. Then, the tool utilizes PHPLY to generate AST for each files, which later would be traversed to build a nested dictionary structure. This specific structure would be used to store the status of all operations, variables, and objects when the tool runs taint analysis. Lastly, the tool would traverse the AST, storing all necessary status changes in the dictionary structure, and return all vulnerabilities detected (when tainted data are found entering sinks).

The tool that we built has 7 main components: FlaskTemplate, FlaskMainApp, MainController, RouteParser, PHPLY, DictionaryBuilder, and Analyzer. It also has a list of sources, sinks, and sanitizers functions. The component diagram is shown in Figure 6. The tool is implemented using Python language and Flask development framework.

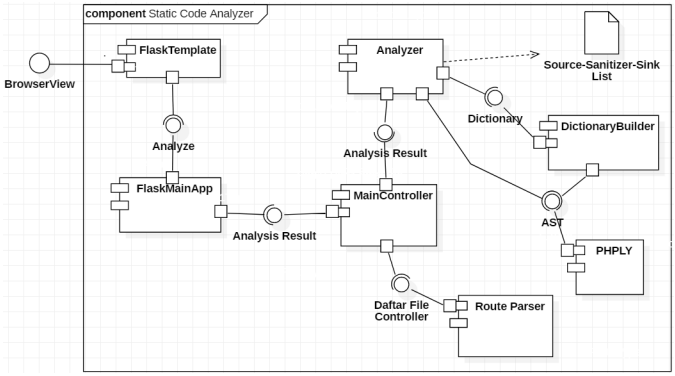

Fig. 6. Component Diagram.

a) FlaskTemplate: FlaskTemplate is a component which provides a browser view for users, where they could upload their Laravel-based application. This component would pass the uploaded file to FlaskMainApp and receive the analysis result from FlaskMainApp. It then shows the analysis result to the user.

b) FlaskMainApp: This component is the main program of Flask, which connect FlaskTemplate and MainController.

c) MainController: MainController is the main program of the tool itself. It passes the Laravel based project to be parsed by RouteParser. After getting the controller file list from RouteParser, it calls Analyzer to analyze each file in that list. MainController gathers the result from analyzer for every file into a result list, which then passed to FlaskMainApp.

d) RouteParser: This component parses the route file of Laravel-based project, which generally named web.php inside routes folder. The parsing process would give back a list of controller files as a result to the MainController.

e) PHPLY: This component is a Python library which produces AST from a PHP file [14].

f) DictionaryBuilder: This component produces an object named NoteDictionary which is used to track all operations/ variables/ objects status during the taint analysis process. It basically traces the AST built by PHPLY and builds a nested dictionary structure, formed by classes shown in Figure 7.

g) Analyzer: This component receives AST from PHPLY and runs taint analysis on it. Every time it detects data flow from one variable/ operation to another, it would record the status in NoteDictionary which it got from DictionaryBuilder. If it finds file calls, such as views files, it would analyze them recursively, even generate AST and NoteDictionary for the file if needed. It uses source-sanitizer-sink list as a reference to determine a behavior of a function. When it detects data flow from source to sink without sanitation process in between, this component would record it as a vulnerability. This component would then sends the list of vulnerabilities it has found back to MainController.

The process starts when a user uploads a Laravel-based project into the FlaskTemplate. FlaskTemplate would pass this file to FlaskMainApp which would extract the .zip file and save all the files inside it. After that, FlaskMainApp would call MainController. MainController would call RouteParser to parse the route file (usually named web.php) to get a list of controller files. After that, for each file in that list, the tool would build a source code representation which are AST and NoteDictionary using PHPLY and DictionaryBuilder. Based on this representation, Analyzer runs taint analysis and returning a list of vulnerabilities. MainController would gather all vulnerabilities from all controller files in a list and return it to FlaskMainApp. FlaskMainApp would pass it to FlaskTemplate to be displayed back to the user.

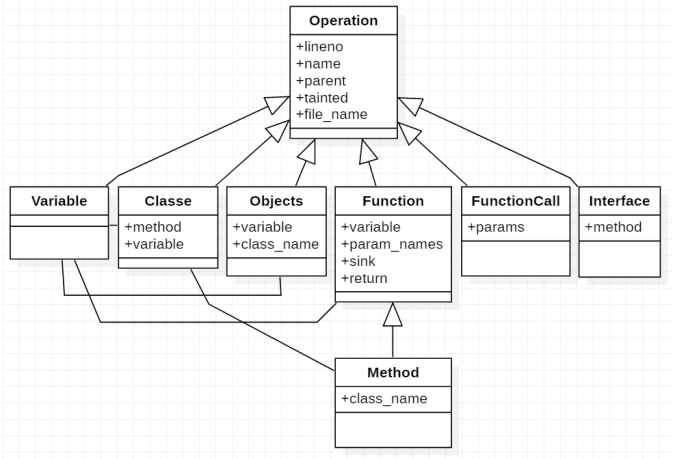

Fig. 7. Operation Class Diagram Showing Classes in *NoteDictionary*.

IV. Experiments and Evaluation

A. Dictionary Structure Validation

In this evaluation, we did manual review to ensure the validity of dictionary structure while running taint analysis. Here we would show one example which uses the code snippet shown in Figure 8. When the tool was creating the dictionary, it would traverse the line 26 (labeled with 1), and create the dictionary structure as shown in Figure 9. The variable *$data* would be considered as tainted (tainted=true).

When the tool runs taint analysis, it would traverse the line 27 (labeled with 2), and find that the variable *$page* is assigned

```
26    public function updatePageDraft(Page $page, $data = [])
27        $page->fill($data);
28        $page->save();
29        return $page;
30    }
31 }
```

Fig. 8. Code Snippet Example

by *$data*. This means, *$page* is also tainted. This status would then be updated in the dictionary as shown in Figure 10.

```
{
  lineno : 26,
  name : $page,
  tainted : False,
  file_name : test-2.php,
  type : Page
  value : []
},
{
  lineno : 26,
  name : $data,
  tainted : True,
  file_name : test-2.php,
  type : None
  value : []
},
```

Fig. 9. The Initial Dictionary Structure

```
{
  lineno : 26,
  name : $page,
  tainted : True,
  file_name : test-2.php,
  type : Page
  value : []
},
{
  lineno : 26,
  name : $data,
  tainted : True,
  file_name : test-2.php,
  type : None
  value : []
},
```

Fig. 10. Dictionary Structure After Line 27 is Traversed

B. Detection Performance Test

The purpose of these experiments and evaluations is understanding the performance of the tool in detecting security vulnerabilities in Laravel-based application. Projects which are used to test the tools along with their CVE vulnerability ID could be observed in Table I. Most detected vulnerability is XSS and the tool also found some new vulnerabilities which are not recorded in CVE Details [15].

In order to get wider range of tested vulnerabilities, the tool is also tested using one additional test application which is a modified version of Laracom [16]. This additional test application also has an SQL injection vulnerability in its code. After some experiments, the result of the experiments were quite satisfactory. The tools could detect a total of 13 security vulnerabilities. The summary of the experiment result could be observed in Table II. It got one False Negative when analyzing application named Microweber [20] and it is suspected to be caused by an ineffective sanitizer method. The sanitizer method has been used in the code, so the tool records the output data as untainted. However, security vulnerability still happens when that output data touch sink method, so it is highly possible that the sanitizer method is ineffective.

TABLE I
TESTED LARAVEL APPLICATIONS

Application Name	CVE ID	Vulnerability Type
Laracom [16]	CVE-2019-15489	XSS
Laravel-bjyblog [17]	CVE-2019-17494	XSS
BookStack [18]	CVE-2017-1000462	XSS
Litecms_Page [19]	CVE-2017-1000467	XSS
Microweber [20]	CVE-2018-1000826	XSS
	CVE-2013-5984	Directory traversal
	CVE-2018-19917	XSS

TABLE II
TEST RESULT

Application Name	CVE ID	Detected/ Not	Additional Information
Laracom	2019-15489	✓	1 new
Laravel-bjyblog	2019-17494	✓	1 new
BookStack	2017-1000462	✓	3 new
Litecms_Page	2017-1000467	✓	1 new
Microweber	2018-1000826	✓	-
	2013-5984	✓	-
	2018-19917	✗	False Negative
Modified Laracom	-	✓	Additional SQL

C. Testing Process Details

The details of analysis process information could be observed in Table III. This tool could do the analysis process in quite a short time with small memory usage. When compared to how many AST that it built and how complex they are based on the node count, it tends to use more time and memories when it builds more complex AST. Microweber is an exception because its files mostly contain InlineHTML which is not being processed that much. On the other side, BookStack [18] has the longer time because its controller files did call to other files often and this kind of call tends to increase the processing time.

TABLE III
TESTING PROCESS INFORMATION

Application Name	AST Built	Average Node/AST	Processing Time (s)	Memory Usage (KB)
Laracom	29	291	1.368	29.4
Modified Laracom	29	292	1.332	29.08
Laravel-bjyblog	18	313	0.852	29.22
BookStack	28	382	7.792	35.52
Litecms_Page	2	655	0.141	28.21
Microweber	504	166	3.85	30.75

V. RELATED WORKS

There are some other related works of static code analysis tools which are definitely worth to check out for. For PHP,

RIPS is quite popular [21]. It uses taint analysis by tokenizing and extracting the PHP codes to detect security vulnerabilities. However, RIPS does not support object oriented codes.

Still in PHP language, Pixy is another static code analysis tool that could detect security vulnerabilities such as XSS, SQL injection, and command injection in PHP based program [22]. It utilizes flow-sensitive, inter procedural, and context-sensitive data flow analysis to detect if there any tainted data reach security sensitive sinks without any sanitation process in the flow. Pixy has 3 analysis processes: aliases analysis, lexical analysis, and taint analysis.

Pixy has a limitation where it could not analyze object oriented code. It also could not handle dynamic inclusion in PHP programs. To deal with those limitations, another tool was built, named OOPixy [23]. This tool is an extension of Pixy which could do analysis in object oriented PHP codes. However, OOPixy only supports PHP 4.3.10 when Laravel application needs at least PHP 5.4. This is why OOPixy would not be that effective for detecting security vulnerabilities in Laravel based applications.

While static code analysis tool for framework in PHP language is still quite rare, there is one tool built for Ruby on Rails, named Brakeman [24]. This tool uses data flow analysis method which traces all variables assignments, branches, arithmetical operations, et cetera. This tool models data flow in Rails application from controller to view in order to detect security vulnerabilities and reduce False Positive that might happen.

VI. Conclusion

Taint analysis could be used for detecting security vulnerabilities in Laravel based web application. Before the analysis process, the source code is modeled using AST and a nested structure. The defined code structure has been validated and found that it could support the taint analysis. The taint analysis traces the AST and records all the status changes in the dictionary. This taint analysis process tends to be more efficient in Laravel-based web application because the source code modeling and the analysis process itself do not need to be done in all files. This is because Laravel-based web application has route file which lists all controller files that should be analyzed. The tool which has been built could detect a total of 13 security vulnerabilities out of 6 Laravel-based web application. It detects one False Negative which is suspected to be caused by ineffective sanitation process.

VII. Future Works

We believed that in the future this work could still be improved by doing more validation tests on more Laravel-based projects. The tool could also be tested and improved to do detection in Laravel-based projects with unusual directory structure. The research itself could still be improved by expanding the scope to other frameworks, or even languages. It could also be expanded to a broader scope of targeted security vulnerabilities.

References

[1] Positive Technologies, “Web application vulnerabilities: statistics for 2018,” Web Application Vulnerabilities: Attacks Statistics for 2018. [Online]. Available: https://www.ptsecurity.com/ww-en/analytics/web-application-vulnerabilities-statistics-2019/. [Accessed: 05-Mar-2019].

[2] Cambridge, “Vulnerability,” Cambridge Dictionary. [Online]. Available: https://dictionary.cambridge.org/dictionary/english/vulnerability. [Accessed: 20-Jun-2020].

[3] The Mitre Corporation, “2019 CWE Top 25 Most Dangerous Software Errors,” CWE: Common Weakness Enumeration, 2019. [Online]. Available: https://cwe.mitre.org/top25/archive/2019/2019_cwe_top25.html. [Accessed: 06-Jun-2019].

[4] OWASP Foundation, Inc., “2017 Top 10,” OWASP. [Online]. Available: https://owasp.org/www-project-top-ten/2017/Top_10. [Accessed: 06-Jun-2019].

[5] Refsnes Data, “PHP Introduction,” W3Schools. [Online]. Available: https://www.w3schools.com/php/php_intro.asp. [Accessed: 13-Jul-2019].

[6] S. McCool, Laravel Starter. Birmingham: Packt Publishing, 2012.

[7] Google Trends, ”Google Trends,” Google, [Online]. Available: https://trends.google.com/trends/explore?q=Laravel,Zend,CodeIgniter, CakePHP,Symfony. [Accessed: 08-Jun-2020].

[8] J. Bergeron, M. Debbabi, J. Desharnais, M. Erhioui, Y. Lavoie, en N. Tawbi, “Static Detection of Malicious Code in Executable Programs”, Int. J. of Req. Eng, 01 2009.

[9] B. Chess and G. Mcgraw, “Static analysis for security,” IEEE Security and Privacy Magazine, vol. 2, no. 6, pp. 76–79, 2004.

[10] F. E. Allen, “Control flow analysis”, Sigplan Notices, vol 5, bll 1–19, 1970.

[11] J. Aldrich, ”Static Analysis,” Analysis of Software Artifacts, 2006.

[12] M. Sridharan, S. Artzi, M. Pistoia, S. Guarnieri, O. Tripp, and R. Berg, “F4F: Taint Analysis of Framework-based Web Applications,” Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications - OOPSLA 11, pp. 1053–1068, 2011.

[13] O. Tripp, M. Pistoia, S. J. Fink, M. Sridharan, and O. Weisman, “TAJ: Effective Taint Analysis of Web Applications,” Proceedings of the 2009 ACM SIGPLAN conference on Programming language design and implementation - PLDI 09, pp. 87–97, 2009.

[14] Viraptor, “PHPLY,” 2020. [Online]. Available: https://github.com/viraptor/phply. [Accessed: 07-Jan-2020].

[15] The Mitre Corporation, “CVE security vulnerability database,” CVE Details. [Online]. Available: https://www.cvedetails.com/. [Accessed: 10-Apr-2020].

[16] J. S. Decena, ”Laracom,” 2020. [Online]. Available: https://github.com/jsdecena/laracom. [Accessed: 13-Jan-2020].

[17] J. Y. Bai, ”Laravel-bjyblog,” 2020. [Online]. Available: https://github.com/baijunyao/laravel-bjyblog. [Accessed: 15-Jan-2020].

[18] BookStackApp, ”BookStack,” 2020. [Online]. Available: https://github.com/BookStackApp/BookStack. [Accessed: 05-Feb-2020].

[19] Litecms, ”Page,” 2020. [Online]. Available: https://github.com/Litecms/Page. [Accessed: 10-Feb-2020].

[20] Microweber, ”Microweber,” 2020. [Online]. Available: https://github.com/microweber/microweber. [Accessed: 12-Feb-2020].

[21] J. Dahse, ”RIPS - A static source code analyser for vulnerabilities in PHP scripts,” Seminar Work at Chair for Network and Data Security, Horst Gortz Institute & Ruhr-University Bochum, 2010.

[22] N. Jovanovic, C. Kruegel and E. Kirda, ”Pixy: a static analysis tool for detecting Web application vulnerabilities,” 2006 IEEE Symposium on Security and Privacy (S&P’06), 2006, pp. 6 pp.-263, doi: 10.1109/SP.2006.29.

[23] M. Nashaat, K. Ali and J. Miller, ”Detecting Security Vulnerabilities in Object-Oriented PHP Programs,” 2017 IEEE 17th International Working Conference on Source Code Analysis and Manipulation (SCAM), 2017, pp. 159-164, doi: 10.1109/SCAM.2017.20.

[24] Brakeman Pro, “What Makes Brakeman Special,” Brakeman Pro. [Online]. Available: https://brakemanpro.com/blog/brakeman/2016/05/25/what-makes-brakeman-special. [Accessed: 23-Feb-2020].