Static Code Analysis Tools with the Taint Analysis Method for Detecting Web Application Vulnerability

Achmad Fahrurrozi Maskur
School of Electrical Engineering and Informatics
Institut Teknologi Bandung
Bandung, Indonesia
fahrurrozi31@gmail.com

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

***Abstract*—Information technology is developing very fast. Along with rapid development, more and more people are interested in becoming a software engineer. But many software engineers are not aware of the quality of a code, especially for security aspects. A solution is needed to solve the problem, such as using automated code scanning technology or often called static code analysis. The purpose of this paper is to build a tool that functions to carry out static code analysis. Static code analysis is carried out using the Taint Analysis method, namely by identifying variables that are suspected of being dangerous (tainted), because they originate from user input. Then do a tracking of these variables to a dangerous function which is then called as sink. If the tainted variable enters the sink before filtering or sanitizing, it is considered as a vulnerability. Evaluation of the constructed tools showed satisfactory results in finding vulnerabilities. From the 20 open source projects listed on the official CVE website, 12 of them were found or as many as 60%. The type of vulnerability that can be found by the taint analysis method is injection type vulnerability.**

Keywords—static code analysis, taint analysis, web vulnerability

I. Introduction

Nowadays information technology is developing very fast. Daily life is inseparable from things that smell of technology. Ranging from transportation, marketplace, entertainment, and others. Many of these technologies are developed with web-based technology. Along with these rapid developments, more and more people are interested in becoming a Web Software Engineer or commonly called a web developer.

But many software engineers are not aware of the quality of a code, especially for security aspects. Based on the official CVE (Common Vulnerabilities and Exposures) Details website, there were around 14,600 vulnerabilities reported in 2017, a drastic increase compared to 2016 there were 6,447 reports [1]. This is also influenced by the complexity of an application that has thousands and even more source code which causes less effectiveness if it is done manually review by a developer. This will end in bad code quality. Bad code causes lots of bugs and ends up with a security hole in an application.

With so many problems that can occur due to negligence of a web developer, we need a method or tools that can minimize such negligence, such as by utilizing automatic source code scanning technology or so-called static code analysis. With the aim of web developers to know if there is a bad code.

This paper will discuss several issues related to how to detect vulnerability with the taint analysis method. Then how to model the source code so that vulnerability detection accuracy is better. Next is what types of security holes can be detected by the taint analysis method. And last is what methods are appropriate for evaluating the taint analysis.

The purpose of this paper is to create a static code analysis tools for web applications with the taint analysis method in detecting security holes (vulnerability).

In this paper, there are some limitations on the scope of work. First, static code analysis is performed only on web applications that are built with the PHP programming language. And for the evaluation object is the Open Source Software PHP project whose vulnerability is listed on the official CVE Details website.

II. Literature Study

A. Web Application Security

Web application is a concept introduced with the Java programming language through Servlet Specification version 2.2 [2]. A web-based application is a computer program that consists of clients and servers where the program runs on a web browser application using an internet network.

Web-based applications are very popular and are often encountered because of the ease of access by the client. In addition there is also the ability to maintain and develop a web application without having to redistribute (centralized on the server). Another interesting thing from a web-based application that is not dependent on a particular platform so it can be accessed by anywhere.

The programming language that is often used in making a web application is PHP (PHP Hypertext Preprocessor). PHP is a programming language that has been used by millions of users throughout the world [3]. Learning PHP is very easy but to build a PHP code that is "clean" is very difficult.

Security is one important aspect in developing a web application. If the security level of a web application is low, it is very possible to have a vulnerability in the web application. The Open Web Application Security Project (OWASP) even released a document "OWASP Top 10 Most Critical Web Application Security Risks" to raise awareness of the importance of security aspects in developing a web application.

978-1-7281-4992-9/19/$31.00 ©2019 IEEE

B. Web Vulnerability

In Oxford Living Dictionaries, vulnerability can be interpreted as a quality or status that is likely to be attacked or harmed, both physically and emotionally. Likewise in computer system security, vulnerabilities cause a system can be attacked by people who are not responsible so that it can cause harm.

According to RFC 2828, vulnerability is a weakness in the design, implementation, or operation and management of systems that can be exploited to violate system security policies[4].

Referring to Seven Pernicious Kingdoms [5], a taxonomy created by Tsipenyuk, Chess, and McGraw, there are seven classifications of vulnerability, namely Input Validation and Representation, API Abuse, Security Features, Time and State, Error Handling, Code Quality, and Encapsulation. This paper will focus on Input Validation and Representation. That is because the most related to static code analysis.

Problems that occur in input validation are caused by untrusted input by user (or attacker), such as the presence of meta-characters, encoding, and other representations. Errors in validating input can cause an injection attack even up to memory leakage. Therefore, an application should check and validate the input to minimize unwanted things. Referring to OWASP Top 10 - 2017, common vulnerabilities include Injection, and Cross-Site Scripting (XSS) [6].

The main problem that often occurs is injection, which is caused by unfiltered user input. This can be seen in OWASP Top 10 starting from 2010, 2013, until 2017 where injection was ranked first in the most common vulnerabilities.

1) Command Injection

Command injection can cause the execution of arbitrary code on the server or commonly referred to as Remote Code Execution (RCE). For example, the PHP code that functions as a calculator on the server side.

```
$input = $_GET['exp'];
eval('$answer = '.$input.';');
```

If the user enters the exp parameter which is `10; system ('rm -rf /')` then the PHP `eval` function will execute two commands with the second command which is to delete all files on the server. Therefore, it is better to filter or sanitize user input.

2) SQL Injection

Similar to command injection, SQL injection is caused by not filtering or sanitizing user input so that users can change the results of an SQL query and even change the database itself. For example, in the following PHP code.

```
$query = mysql_query("SELECT * FROM user
  WHERE user='". $_POST['username']."'
  AND pass='".$_POST['password']."'");
$count = mysql_num_rows($query);
```

If the user enters the username parameter which is `admin' or 1 = 1 --` then the SQL query will stop up to the part `--` because the rest will be considered as comments on the query.

C. Static Code Analysis

To avoid errors or vulnerabilities that have been mentioned, it is necessary to find errors in the program from the beginning of development to the production stage. There are several ways, including code review, static code analysis, unit tests, vulnerability scans, etc.

One advantage of static code analysis compared to other methods of finding errors in programs is that vulnerability can be found without executing the program. Vulnerability is found only by scan the source code of the program with various approaches.

Taint Analysis is done by identifying variables that are suspected of being dangerous (tainted), because they from user input. Then do a track of these variables for functions that are also suspected of being vulnerable. If the tainted variable executed by sink before filtering or sanitizing, it is marked as vulnerability. Broadly speaking, taint analysis consists of three important components, sources, sinks, and sanitizers [8].

Sources are all sources that can be used by attackers to insert dangerous malicious code. In a web-based PHP application, some of which become sources are HTTP request parameters, sessions, uploaded files, etc.

Sinks are places or end functions that execute actions that are vulnerable to attack. In a web-based PHP application, some of which become sinks include the SQL query function, include file functions, functions related to the operating system, etc.

Sanitizers are anything that can protect or filter the given sources. In a web-based PHP application, some of which become sanitizers include the html escape tag function, SQL query escape function, and others.

III. Implementation Overview

A. Taint Analysis in Finding Vulnerabilities

If there is a sources that passes through a sinks without going through a sanitizers first, it can be said that the web application might be vulnerable to an attack.

```
<?php
1
2 $name = $_GET["name"];
3 echo "Hello, ". $name;
4
```

Fig. 1. Vulnerable code example

Fig 1 above is an example of code that can cause vulnerabilities, which is Cross-site Scripting. This is because the variable `$name` is assigned by user input, therefore it marked as tainted. The tainted variable is then executed by the `echo` which is one of sink function. However, if the input from the user first passes through the sanitize function, then the user input is considered safe from the danger of injection. Examples of safe codes can be seen in Fig 2.

```php
<?php

$name = $_GET["name"];
$name = htmlentities($name)
echo "Hello, ". $name;

```

Fig. 2. Safe code example

With taint analysis, a vulnerability that can be detected is an injection type vulnerability because it is only checked for dangerous functions or sinks, also for user input. Simply, a program is said to have vulnerability if and only if there is a variable that is tainted, not sanitized and then executed by a malicious or sink function. To be more easily understood, the following visualizations in the form of state diagrams can be seen in Fig 3.

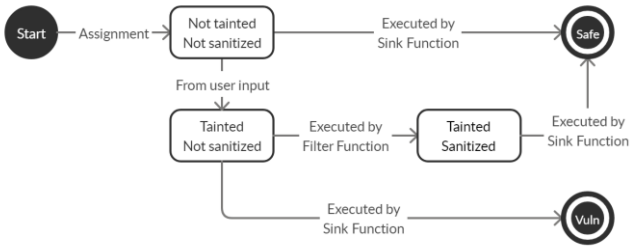

Fig. 3. Variable state diagram taint analysis

The three components of sources, sanitizers and sinks will be defined using a knowledge base. The knowledge base used is a config file that aims to register all the knowledge we have. The knowledge includes which variables are included in sources, as well as what functions are included in sanitizers or sinks.

In checking the given source code, preprocessing is first done in the form of parsing the source code which then produces an abstract syntax tree for further processing. Then for each node in the tree is processed according to the type of token of each node.

B. Preprocessing: Parsing and Abstract Syntax Tree

Preprocessing is done with the help of an open source library tools which name is PHPLY, a parser of the PHP programming language using the help of PLY (Python Lex-Yacc) [7]. It changes the source code in the form of lexical or tokens. For example, a simple source code as shown in Fig 1 will produce an abstract syntax tree as follows.

```
('Assignment',
 {'expr': ('ArrayOffset',
           {'expr': u'name',
            'lineno': 2,
            'node': ('Variable', {'lineno': 2, 'name': u'$_GET'})}),
  'is_ref': False,
  'lineno': 2,
  'node': ('Variable', {'lineno': 2, 'name': u'$name'})})
('Echo',
 {'lineno': 3,
  'nodes': [('BinaryOp',
             {'left': u'Hello, ',
              'lineno': 3,
              'op': u'.',
              'right': ('Variable', {'lineno': 3, 'name': u'$name'})})]})
```

The resulting abstract syntax tree consists of several nodes according to the order in the given source code. Each node consists of the type and expression of the code itself. In processing each node, it needed to checks the type of each node.

There are several types of nodes, including Assignment, Function, Function-Call, Include, If, Switch, For, Foreach, etc. The type of node to be explained is what affects the taint analysis

Assignment, is one type of node that plays an important role in the taint analysis process. This is because user input will be saved into a variable list. For each variable, the stored parameters are `value (object), line number (int), tainted (bool), and sanitized (bool)`. Examples of source code with Assignment node types can be seen in Fig 4.

```php
<?php

# Variable Assignment
$myVariable = $anotherVar;

# Array Assignment
$myArray = array(1, 2, 3);

# Constant Assignment
$myConst = true;

```

Fig. 4. Example of source code with Assignment type

A variable is tainted when the variable is assigned by user input that has been previously defined via a knowledge base. Examples of user input in PHP are `$_GET`, `$_POST`, `$_COOKIE`, `$_REQUEST`, etc.

In addition to tainted status, sanitized status is also obtained on the Assignment node type, which is when a variable is assigned by a function that is included in the sanitizers. The functions included in sanitizers are also defined via a knowledge base. Examples of functions included in sanitizers are `htmlentities`, `addslashes`, `escapeshellcmd`, `urlencode`, etc. Example of source code with Assignment node type that can lead to tainted state and tainted-sanitized can be seen in Fig 5.

```php
<?php

# Varible name Tainted
$name = $_GET["name"];

# Varible name Tainted-Sanitized
$name = htmlentities($name);

```

Fig. 5. Example of Assignment can lead to tainted-sanitized state

FunctionCall, is also one type of node that plays an important role in the taint analysis process. FunctionCall can be a registered function or a default function. The default sink functions are determined from the knowledge base. Example of source code which calls a function that is sinks can be seen in Fig 6.

```php
<?php

$name = $_GET["name"];

# FunctionCall Sinks
print("Hello, " + $name);

```

Fig. 6. Example of FunctionCall which is a sink

In addition to the type of nodes that affect the taint analysis, there are also types of nodes that affect the control flow of the given source code. Some examples of node types that affect the control flow are Include, If, Switch, For, Foreach, etc.

C. Functional Requirements

There are two types of actors associated with this tools, namely software engineers and security engineers. Software engineer is a user who can do static code analysis by inputting source code or uploading files. Software engineers can of course also see the results of the static code analysis. Security engineer is an actor who has experience with existing vulnerabilities, so that security engineer is a user who can add knowledge owned by tools to enrich the knowledge base. This functionality is illustrated by the use case diagram which can be seen in Fig 7.

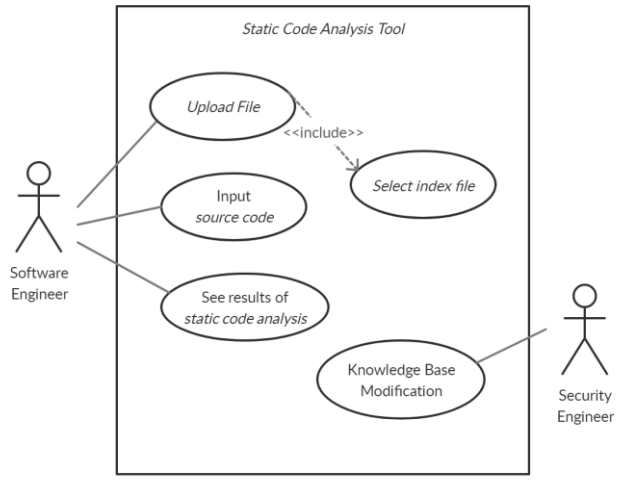

Fig. 7. Use case diagram of static code analysis tools

There are five requirements in implementing the tool. First requirement is being able to do static code analysis. This functionality is the main requirement of the static code analysis tools that will be built. The tool is expected to be able to detect vulnerabilities if there are vulnerabilities in the source code provided. Second is the tools has the "input source code" feature, so users can type or paste the source code easily. Third is the tools has the "upload file" feature, so users can upload files with certain types of extensions (only php and zip). With the file upload function, users can also upload multiple files with zip extension and then choose the 'index' file. Fourth is the tool can display the results, so users can see the results of static code analysis. There are two possibilities: no vulnerability or there is a vulnerability. And the last is security engineer can modify the knowledge base by the config file.

D. Static Code Analysis: Taint Analysis

Based on the previous sections, the work flow of the constructed tools will be illustrated in the form of a flow chart, the illustration can be seen in Fig 8.

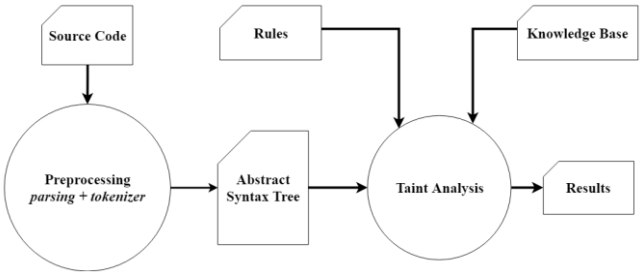

Fig. 8. Work flow of static code analysis tools

In this paper, the main focus is on the AST process to Taint Analysis. That is because the preprocessing process is done with the help of PHPLY. In addition, the taint analysis also receives input from the rules and knowledge base that have been explained in previous sections. From these three inputs, it is hoped that the tools can produce good accuracy in detecting vulnerabilities.

Next, the implementation of architecture is carried out in accordance with the work flow design described as shown in Fig 8. Implementation of architecture the technology used will be describe by using component diagram that can be seen in Fig 9.

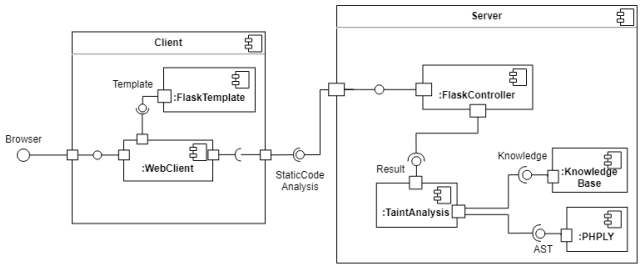

Fig. 9. Component diagram of static code analysis tools

The implementation of architecture using the principle of client and server side using Flask technology. On the client-side, the flask-materialize library is used to facilitate front-end implementation. In the server-side, there are two different features namely input source code, and upload files. Then from the source code or file provided, a taint analysis is carried out which will produce results in the presence or absence of detected vulnerabilities. In addition, taint analysis is also carried out with the help of PHPLY and its knowledge base.

IV. Experiment and Evaluation

The static code analysis tools is tested to ensure the success of the tools. The main objective in testing is the implementation of tools successfully detecting vulnerabilities with the taint analysis method. Before testing complex applications, first do a simple source code design that has been designed to have vulnerabilities in it. There are three important components in testing. The three components are list of variables, control flow program, and the truth of detecting vulnerability. In testing with a simple source code that has been designed as can be seen in Fig 10, the tools has successfully tested the three components that have been mentioned. The results of taint analysis can be seen in Fig 11.

```php
<?php


echo $_GET["vuln"];
echo htmlentities($_GET["safe"]);

```

Fig. 10. Simple source code that have vulnerabilites

Fig. 11. Results of taint analysis with simple source code

Furthermore, testing is done with an open source PHP project that has a CWE (Common Weakness Enumeration) ID related to Injection and Cross-site Scripting (XSS). From the existing CWE ID, detection is done on the corresponding CVE ID. List of applications tested can be seen in Table 1.

TABLE I. THE CVE IDS BEING TESTED

No	CWE ID	CVE ID	Open Source Project
1	79	CVE-2019-1010287	Timesheet Next Gen
2	79	CVE-2019-13186	MiniCMS
3	79	CVE-2019-13339	MiniCMS
4	79	CVE-2019-13340	MiniCMS
5	79	CVE-2019-13341	MiniCMS
6	79	CVE-2019-11359	I, Librarian
7	79	CVE-2019-11428	I, Librarian
8	79	CVE-2019-11449	I, Librarian
9	79	CVE-2019-9839	VFront
10	77	CVE-2019-12585	Pfsense
11	78	CVE-2018-1000019	OpenEMR
12	78	CVE-2017-1000235	I, Librarian
13	78	CVE-2016-10709	Pfsense
14	89	CVE-2019-1010153	Zzcms
15	89	CVE-2019-1010104	TechyTalk Quick Chat
16	89	CVE-2019-13578	GiveWP Give plugin
17	89	CVE-2019-13575	WPEverest Forms
18	89	CVE-2019-13086	CSZ CMS
19	89	CVE-2019-14529	OpenEMR
20	89	CVE-2019-14313	10Web Photo Gallery

From the CVE list, the results were quite satisfactory. 12 out of 20 vulnerabilities have been detected by the tool or as many as 60%.. There are eight that are not found, three of them require special techniques, with bypassing the sanitizers function. Two of them are Object Oriented Programming (OOP) which is not yet supported. And the rest is due to functions that are not known by the tools. For more details, can be seen in Table II below.

TABLE II. TEST RESULT

No	CWE ID	Found	Description
1	CVE-2019-1010287	V	*Vulnerability* detected
2	CVE-2019-13186	V	*Vulnerability* detected
3	CVE-2019-13339	V	*Vulnerability* detected
4	CVE-2019-13340	V	*Vulnerability* detected
5	CVE-2019-13341	V	*Vulnerability* detected
6	CVE-2019-11359	V	*Vulnerability* detected
7	CVE-2019-11428	V	*Vulnerability* detected
8	CVE-2019-11449	V	*Vulnerability* detected
9	CVE-2019-9839	X	Tainted parameter of unknown function
10	CVE-2019-12585	V	*Vulnerability* detected
11	CVE-2018-1000019	V	*Vulnerability* detected
12	CVE-2017-1000235	X	Required special techniques (insert "`||<payloads>||`")
13	CVE-2016-10709	X	Required special techniques (*bypass via '`|`' character*)
14	CVE-2019-1010153	V	*Vulnerability* detected
15	CVE-2019-1010104	X	Required special techniques (*bypass* `like_escape` function)
16	CVE-2019-13578	X	Not supported (OOP)
17	CVE-2019-13575	X	Not supported (OOP)
18	CVE-2019-13086	V	*Vulnerability* detected
19	CVE-2019-14529	X	SyntaxError (Recursive ArrayOffset)
20	CVE-2019-14313	X	Not supported (*database controller*)

There are still false negatives caused by OOP features that are not yet supported by the tools. However, the tool still managed to detect vulnerabilities in files that use the OOP feature because the vulnerability found was in the section that did not use the OOP feature. Also, some of sanitizer functions are not effective in "cleaning" harmful input, which also causes several cases of vulnerability to occur, so that there are still false negatives.

V. CONCLUSION

Detecting web application vulnerabilities using taint analysis is possible. From the source code provided, then modeled to becomes an Abstract Syntax Tree to make it easy to do the taint analysis. Taint analysis successfully detects a vulnerability if in the source code it is clear there is a tainted variable that is executed by sinks without going through the sanitizer function first. However, the results of the proposed method can be improved by supporting the Object Oriented Programming. The results can also be improved by adding more knowledge used in the tools.

REFERENCES

[1] "Browse CVE vulnerability by date", *CVE Details*, 2018. [Online]. Available: https://www.cvedetails.com/browse-by-date.php. [Accessed: 21- Oct- 2018].

[2] A. Chaffe, "What is a web application (or "webapp")?", *Jguru.com*, 2012. [Online]. Available: http://www.jguru.com/faq/view.jsp?EID=129328. [Accessed: 21- Oct- 2018].

[3] W. Huiren, "Introduction to PHP", *Woo Huiren's Blog*, 2015. [Online]. Available: https://woohuiren.me/blog/introduction-to-php. [Accessed: 21- Oct- 2018].

[4] R. Shirey, "Internet Security Glossary", *Ietf.org*, 2000. [Online]. Available: https://www.ietf.org/rfc/rfc2828.txt. [Accessed: 21- Oct- 2018].

[5] K. Tsipenyuk, B. Chess and G. McGraw, "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors", *IEEE Security and Privacy Magazine*, vol. 3, no. 6, pp. 81-84, 2005. Available: 10.1109/msp.2005.159.

[6] *The Ten Most Critical Web Application Security Risks*. The OWASP Foundation, 2017.

[7] D. Benjamin, phply, (2018), GitHub repository. Available: https://github.com/viraptor/phply

[8] M. Shannon, "Is your code tainted? Finding security vulnerabilities using taint tracking", in EuroPython Conference 2018, Edinburgh, United Kingdom, 2018.