GALAXY INTERNATIONAL INTERDISCIPLINARY RESEARCH JOURNAL (GIIRJ)
ISSN (E): 2347-6915
Vol. 10, Issue 5, May. (2022)

STEPS TO BUILD STATIC CODE ANALYSIS TOOL TO DETECT VULNERABILITIES IN PHP PROGRAMMING LANGUAGE

Utepbergenov Yunus Qutlimuratovich
Master's Student of "Information Security" Faculty,
Tashkent University of Information Technologies named after Muhammad al-Khwarizmi
yunusinha@gmail.com

ABSTRACT

This article discusses the steps required to create an analytical tool that performs static analysis to identify vulnerabilities in the code of programs written in the Php programming language, which is one of the most widely used programming languages.

Keywords: static analysis, lexical analysis, semantic analysis, control flow analysis, data flow analysis, potential vulnerability functions, Taint analysis, remote system command execution vulnerability, internal process analysis, external process analysis

ANNOTATSIYA

Ushbu maqolada eng ko'p foydalaniladigan dasturlash tillaridan biri hisoblangan php dasturlash tilida yozilgan dasturlarning kodidagi zaifliklarni aniqlash uchun statik tahlilni amalga oshiruvchi tahliliy vositani yaratish uchun talab etiladigan bosqichlar haqida so'z yuritiladi.

Kalit so'zlar: statik tahlil, leksik tahlil, semantik tahlil, boshqarish oqimi tahlili, ma'lumotlar oqimi tahlili, potentsial zaiflik funksiyalari, Taint tahlili, masofadan tizim buyruqlarni amalga oshirish zaifligi, ichki jarayonlar tahlili, tashqi jarayonlar tahlili

INTRODUCTION

When analyzing program code using the static method, the source code of the computer program is analyzed without running it. The static analysis method allows you to assess all of the activities that the program can take at the same time as evaluating the influence of data entered by users on the program's performance. The static analysis method is widely used for a variety of purposes, such as defining syntax, checking variable types, optimizing a program, and identifying program code errors and vulnerabilities that affect program security. In general, we can consider any tool that checks the program code without running it as a static analysis tool. There are four basic steps in building static code analysis tool: configuring, model construction, analyzing, and processing the results.

CONFIGURATION

First of all, a static analysis tool needs a clear set of rules to know what to look for in a program code. In order to analyze the vulnerabilities found in program code, it is necessary to identify which types of vulnerabilities are identified and how these vulnerabilities can be found during analyis. Static analysis tool also needs to identify sources that could corrupt the data, such as information that users can enter. Finally, it is necessary to adjust the appropriate settings that can be taken by the developer of the program and to prevent the safe parts of the program from appearing in the analysis results.

Model Construction

The static analysis tool should convert the source code to an program model. This is an abstract internal view of the source code. The quality of the analysis of the analytics tool depends in many ways on the quality of the tool model, so it is important that the analytics tool has a good understanding of the semantics of the programming language. Creating a program model consists of the following steps:

- **Lexical Analysis**
  The source code must be in the tokens to correctly identify the language constructs of the analytics tool. Insignificant tokens, such as spaces and comments in the program code, are usually removed to identify the connected tokens correctly.

- **Semantic Analysis**
  The analysis tool checks the appearance of each token. For example, it is important to decide between the call to the print() function and the same word used on the line. You can also define the types of variables.

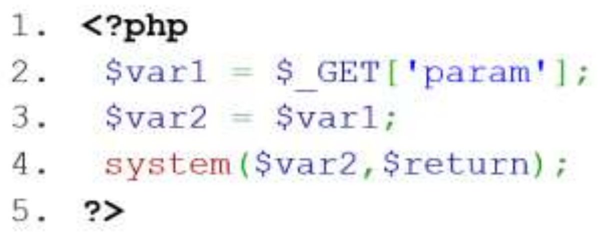
- **Control Flow Analysis**.
  The program identifies all available actions that can be performed. These paths are then combined into multiple control flow charts that represent all possible data flow paths.

- **Data Flow Analysis**.
  This tool uses data flow analysis on each control flow chart to determine how data moves throughout the program. The Security Vulnerability Analyzer uses Taint analysis to determine where vulnerabilities occur. Data flow analysis is based on the results of semantic and control flow analysis. Therefore, it is important that the previous results are as accurate as possible.

ANALYSIS

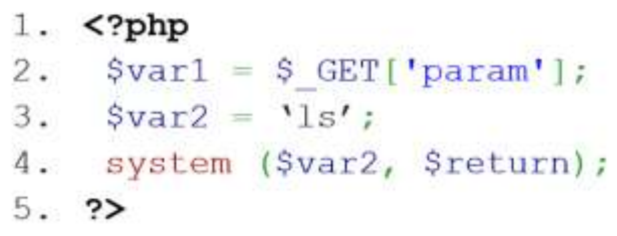
When a model is created, the analysis tool performs a Taint analysis on each potential vulnerability function found. It should also perform internal process analysis to analyze the potential vulnerability function call in an individual function and external process analysis to analyze interactions between multiple functions.

During the Taint analysis, we can detect that the compromised data could reach vulnerable parts of the program and result in a security vulnerability. The analytics tool should be able to differentiate whether the compromised data reaches the vulnerable parts of the program during the data flow analysis phase.



```php
<?php
 $var1 = $_GET['param'];
 $var2 = $var1;
 system($var2,$return);
?>
```

Figure 1. Remote command execution vulnerability



```php
<?php
 $var1 = $_GET['param'];
 $var2 = 'ls';
 system ($var2, $return);
?>
```

Figure 2. Harmless command execution

Figures 1 and 2 show two php scripts that use the system() function, which is a potential vulnerability function that executes system commands. Although the code in Figure 1 indicates a vulnerability in remote command execution where the user can execute any command on a parameter called a param, the code shown in Figure 2 is not a vulnerability because the executed command is static and cannot be controlled by an attacker or hacker. By analyzing the data flow of the potential vulnerability function parameter, the analytics tool can detect whether unreliable data has reached parts that could cause software vulnerabilities, or whether the potential vulnerability function is invoked by harmless static data.

When performing taint analysis, the static analysis tool should also perform internal process analysis. This analysis involves tracking data within a function called by the user. If a potential vulnerability is found within a function, the analyzer should be able to determine under what circumstances the vulnerability could be caused when the function is called. This is mainly due to the fact that the vulnerability depends on the parameters of the user-created functions that called with malicious information.

The analysis tool should also determine whether the malicious data is affecting the function or whether it is being cleared when the function is processed. In addition, the function can return malicious data even if it is not called with malicious data (for example, when the data entered by the user while performing the function is accepted).

External process analysis is the analysis of a non-functional context when a function is called. Depending on the state of the program, calling a function can cause the program to have unexpected events or change external variables on a global scale. Because static source code analysis requires the analysis of all possible control flow chart data, it can be very difficult to perform both analyzes without increasing the analysis time.

Processing the Results

The last important part of static source code analysis is to present the results to the user in a way that can quickly identify vulnerabilities. Showing only the relevant parts of the weak code, as well as highlighting the syntax, should help you review and understand the problem. It will also be helpful to provide automated information on how to fix this bug.

If the problem-free part of the code is incorrectly marked as weak, the condition is called a false positive or a false signal. If a tool cannot detect a vulnerability, but actually has a vulnerability, the condition is false negative and the correctly identified vulnerability is called true positive. The better the value of the analysis tool, the more real positives can be identified and the fewer false positives. Static code analysis tools often generate a lot of false positives and warnings. This may be due to incorrect semantics or data flow analysis or situations that are necessary to correctly identify vulnerabilities but have not been properly evaluated during the analysis. Therefore, it is important to present the results in such a way that the user can easily make a decision between correctly identifyed vulnerability and a false positive.

REFERENCES

1. Nico L. de Poel, Automated Security Review of PHP Web Applications with Static Code Analysis, 2010.
2. Get Started with PHP Static Code Analysis [Elektron resurs]. -Kirish tartibi: https://deliciousbrains.com/php-static-code-analysis/
3. Interprocedural analysis (IPA) [Elektron resurs]. -Kirish tartibi: https://www.ibm.com/docs/en/i/7.2?topic=techniques-interprocedural-analysis-ipa
4. Jiazhen Zhao et al, WTA: A Static Taint Analysis Framework for PHP Webshell, 2021.