SCRIPT ANALYZER

Shanmukha Agastyaraju (854024207)

Chinmay Rajguru (853964913)

Abstract

This project focuses on finding security vulnerabilities in the given script (JavaScript, PHP, Java, C++ and C#). For example, if a JavaScript code snippet is given as an input to our application, it will read each line, determine the methods and variables used. According to the flow of execution of the program, application will detect the points where the code is vulnerable to attacks. We are focusing first on this part of the project. If this part of the project is functional, we would like to extend our project further on giving recommendations for the detected vulnerability.

1. Introduction

In the current world, we use many programming languages for backend development of websites as well as web applications. When we deploy those programs we should also consider whether they are safe or not to ensure the user's confidentiality and data security. Our project focuses on the backend script (e.g. JavaScript). Input for our program will be a JavaScript file. Our application will analyze JavaScript code and compare it with the vulnerabilities that are mentioned for JavaScript. After the detection of these vulnerabilities, our application will provide the user with all the functions or declarations that are vulnerable to hackers and other various threats. All these detections which are going to be implemented will be based on the vulnerabilities of JavaScript mentioned by various users, security websites and trusted sources. Our main target is to detect where the user is going to pass his/her login credentials. Because, that is the most important information for any user, to gain access of his confidential information. If the username and password are safe, then the rest of his/her data will be secure apparently. To achieve this goal, we will check whether the data is getting passed through HTTP request or HTTPS request. For example: Consider the username and password which are in plaintext passing through HTTP request; then we will recommend the user that it should be encrypted by some kind of algorithm.

2. Importance

In the current threat environment, finding vulnerabilities in any programming language is incredibly important. These findings can serve to better protect users and make software developers and vendors aware of flaws that could put sensitive information at risk of exposure. And with cyber-attacks becoming more prevalent, one would think the industry would welcome any opportunity to mitigate the chances of hackers breaking into commonly-used programs.

Vulnerability research assists the engineering teams in pinpointing flaws in software programs that could lead to security issues. These efforts could include reverse engineering, static and code analysis along with an array of other initiatives to recognize program issues.

In recent years, vulnerability has moved from a white hat hobby to a more pressing need within the industry. As the instances of cyberattacks continue to increase, users are increasingly aware that anyone can become a victim. Because many attacks come due to zero-day exploits, the pressure is on to discover weaknesses and patch them as quickly as possible. This has resulted in both established security vendors as well as startups expanding their ability to discover vulnerabilities in applications and websites. In effect, the ecosystem surrounding vulnerability research has been changed by the need to deal with targeted attacks. As the threat landscape continues to evolve, finding code vulnerabilities will become an increasingly key part of the security and technology sectors.

3. Code Analysis

3.1 Static Analysis

Static Analysis is completed statically without executing the code. In this strategy the tool investigates input program code for conceivable security vulnerabilities and alarms the client.

3.2 Dynamic Analysis

Dynamic Analysis is conveyed amid the execution of a program. Dynamic analyzer screens system memory, response time and general execution of the program to discover security vulnerabilities. Proposes dynamic pollute proliferation systems which encourage discovering input approval mistakes utilizing useful tests. It has been found that static analysis is more appropriate in discovering vulnerabilities. Static analysis tools are intended to distinguish security vulnerabilities in programming code that, if not revised, may prompt exploitable security vulnerabilities. The tools worked for static analysis can be utilized to discover runtime blunders and asset holes and security vulnerabilities statically, i.e. without executing the code.

Presents a diversion to instruct the clients in different Computer security ideas in which members are given arrangement of inquiries to comprehend. Reports that there is as yet a requirement for development in the static investigation apparatuses. This project proposes a technique and a device which statically examinations programs coded in JavaScript, PHP, Java, C++ and C# to discover security vulnerabilities and recommend elective secure coding builds for the same.

4. Diagram

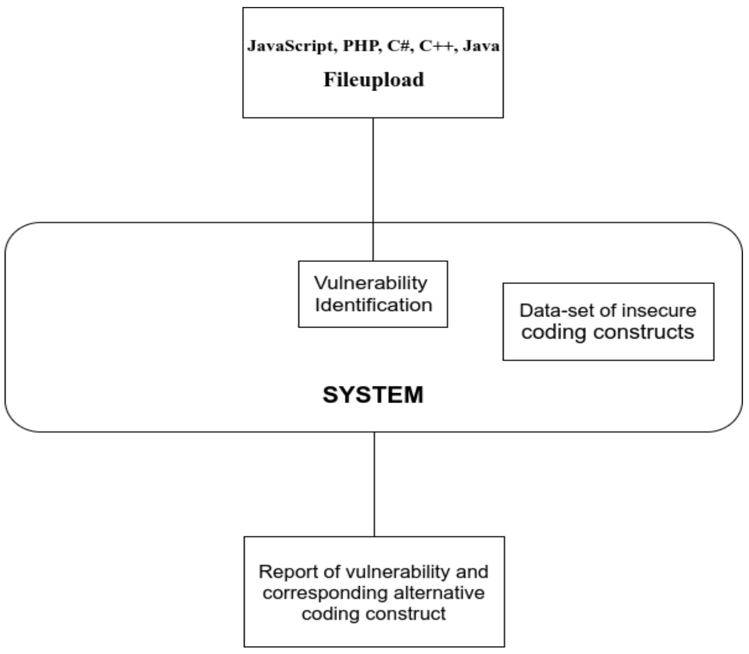

JavaScript, PHP, C#, C++, Java
Fileupload
Vulnerability Identification
Data-set of insecure coding constructs
SYSTEM
Report of vulnerability and corresponding alternative coding construct

Figure 4.1

5. Proposed Solution

Proposed method is intended to let user upload the input program file to be inspected for security vulnerabilities. It inspects the uploaded input source code program file and alerts the user about presence of vulnerabilities if any. The developed tool maintains a list of insecure coding constructs which may lead to security vulnerability in any program written in JavaScript, PHP, Java, C++ and C# languages, along with the corresponding alternative secure coding constructs. As the tool inspects the input program, it checks for the presence of insecure code or functions or keywords from the master list, if found a report is generated listing the insecure coding constructs present in the input program and also providing alternate secure coding constructs that can be used. The tool is developed to overlook vulnerabilities if present in comments section to avoid false positives, thus avoiding false negatives. The source code is given as input to the tool as illustrated in Figure 7.2. The tool may be given the code after clean compilation. Screenshot in Figure 7.3 shows the sample output of tool "Vulnerability Reporter" developed by our team: The tool flags errors when functions which do not carry out input validations such are used in the program.

6. Technical Details:

Our Script Analyzer addresses the following vulnerabilities:

6.1 XSS (Cross-Site Scripting):

Cross-site Scripting (XSS) refers to client-side code injection attack wherein an attacker can execute malicious scripts (also commonly referred to as a malicious payload) into a legitimate website or web application. XSS is amongst the most rampant of web application vulnerabilities and occurs when a web application makes use of invalidated or un-encoded user input within the output it generates.

By leveraging XSS, an attacker does not target a victim directly. Instead, an attacker would exploit a vulnerability within a website or web application that the victim would visit, essentially using the vulnerable website as a vehicle to deliver a malicious script to the victim’s browser.

6.2 LDAP Injection:

DAP Injection is an attack used to exploit web based applications that construct LDAP statements based on user input. When an application fails to properly sanitize user input, it’s possible to modify LDAP statements using a local proxy. This could result in the execution of arbitrary commands such as granting permissions to unauthorized queries, and content modification inside the LDAP tree. The same advanced exploitation techniques available in SQL Injection can be similarly applied in LDAP Injection.

6.3 SQL Injection:

SQL Injection is a code injection technique, used to attack data-driven applications, in which nefarious SQL statements are inserted into an entry field for execution (e.g. to dump the database contents to the attacker). SQL injection must exploit a security vulnerability in an application's software, for example, when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and unexpectedly executed. SQL injection is mostly known as an attack vector for websites but can be used to attack any type of SQL database.

SQL injection attacks allow attackers to spoof identity, tamper with existing data, cause repudiation issues such as voiding transactions or changing balances, allow the complete disclosure of all data on the system, destroy the data or make it otherwise unavailable, and become administrators of the database server.

6.4 Parameter Tampering:

Parameter tampering is a form of Web-based attack in which certain parameters in the Uniform Resource Locator (URL) or Web page form field data entered by a user are changed without that user's authorization. This points the browser to a link, page or site other than the one the user intends (although it may look exactly the same to the casual observer). Parameter tampering can be employed by criminals and identity thieves to surreptitiously obtain personal or business information about the user.

7. Screenshots

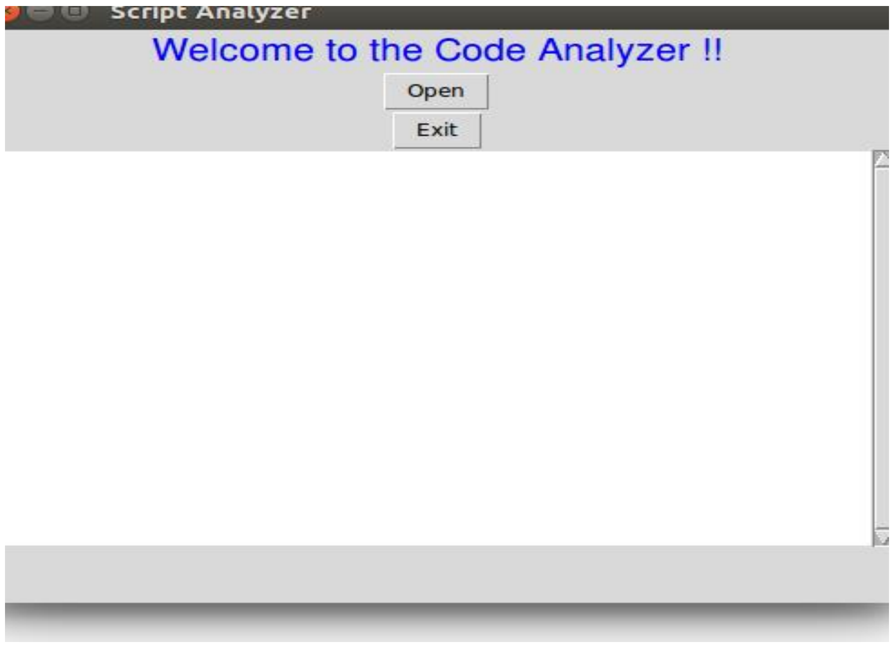
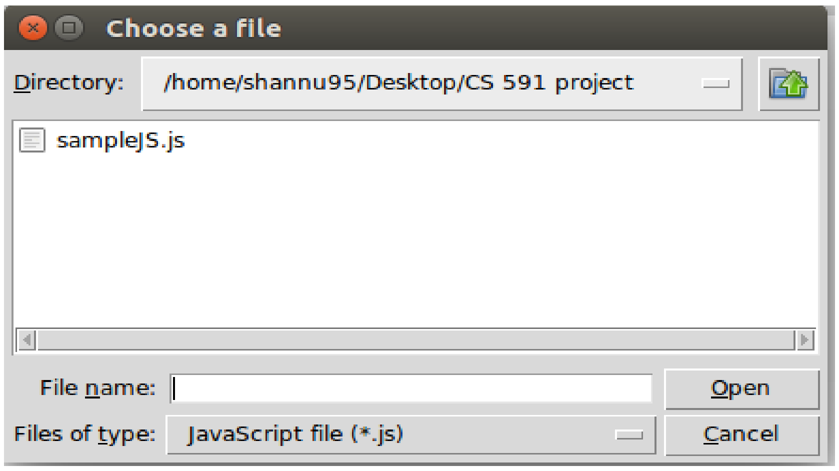


Figure 7.1. Opening Screen of the Program



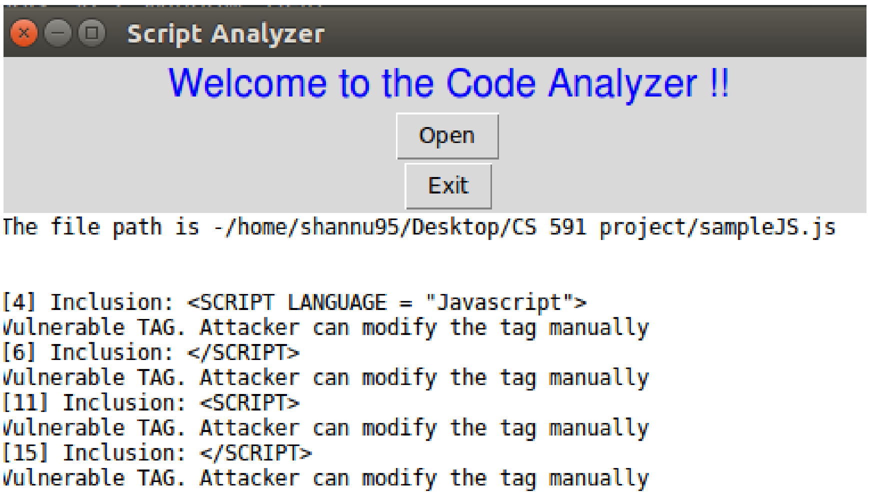
Directory: /home/shannu95/Desktop/CS 591 project

sampleJS.js

File name:

Files of type: JavaScript file (*.js)

Figure 7.2. Select the File with extension of .cs / .java / .js / .cs / .cpp



```
The file path is -/home/shannu95/Desktop/CS 591 project/sampleJS.js


[4] Inclusion: <SCRIPT LANGUAGE = "Javascript">
Vulnerable TAG. Attacker can modify the tag manually
[6] Inclusion: </SCRIPT>
Vulnerable TAG. Attacker can modify the tag manually
[11] Inclusion: <SCRIPT>
Vulnerable TAG. Attacker can modify the tag manually
[15] Inclusion: </SCRIPT>
Vulnerable TAG. Attacker can modify the tag manually
```

Figure 7.3. Output Screen

8. How it works

I. Give Input of .JS /. PHP / .CPP / .CS / .JAVA file
II. System reads the script line by line
III. Compares every line with given dataset
IV. Get output as line number, vulnerable script and its reason and possible solution

9. Conclusion

In this project we highlight the problems of insecure coding in software applications and briefly present features of a tool developed by us that analyses the given code for security vulnerabilities and generate a report of the vulnerabilities present along with suggestions for improving the same. The tool was put to use extensively and the results confirm that there is a significant improvement over manual code review. Since a large number of security incidents occur of late because of insecure applications, the work presented here has profound implications for the future and may be used in peer reviews of software applications.

10. References:

1. https://www.owasp.org/index.php/LDAP_injection
2. https://www.acunetix.com/websitesecurity/sql-injection/
3. https://www.acunetix.com/websitesecurity/cross-site-scripting/
4. https://hdivsecurity.com/owasp-insecure-direct-object-reference