2022

Automatically Generating Searchable Fingerprints For WordPress Plugins Using Static Program Analysis

Chuang Li
Wright State University

AUTOMATICALLY GENERATING SEARCHABLE FINGERPRINTS FOR WORDPRESS PLUGINS USING STATIC PROGRAM ANALYSIS

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science

by

CHUANG LI
B.E., Xi'an University, China, 2018

2022
Wright State University

Wright State University
GRADUATE SCHOOL

April 27, 2022

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Chuang Li ENTITLED Automatically Generating Searchable Fingerprints for WordPress Plugins Using Static Program Analysis BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science.

Junjie Zhang, Ph.D.
Thesis Director

Michael Raymer, Ph.D.
Chair, Department of Computer
Science and Engineering

Committee on Final Examination:

Junjie Zhang, Ph.D.

Krishnaprasad Thirunarayan, Ph.D.

Bin Wang, Ph.D.

Barry Milliagan, Ph.D.
Vice Provost for Academic Affairs
Dean of the Graduate School

ABSTRACT

Li, Chuang. M.S., Department of Computer Science, Wright State University, 2022. *Automatically Generating Searchable Fingerprints For WordPress Plugins Using Static Program Analysis.*

This thesis introduces a novel method to automatically generate fingerprints for WordPress plugins. Our method performs static program analysis using Abstract Syntax Trees (ASTs) of WordPress plugins. The generated fingerprints can be used for identifying these plugins using search engines, which have support critical applications such as proactively identifying web servers with vulnerable WordPress plugins. We have used our method to generate fingerprints for over 10,000 WordPress plugins and analyze the resulted fingerprints. Our fingerprints have also revealed 453 websites that are potentially vulnerable. We have also compared fingerprints for vulnerable plugins and those for vulnerability-free plugins.

Contents

List of Figures

List of Tables

Listings

Acknowledgment

I want to take this chance to express my sincere appreciation to my advisor, Dr. Junjie Zhang. The completion of this thesis and my dissertation would not have been possible without his support and nurturing. His extensive knowledge and insight have been invaluably helpful in my study. I would like to thank my colleague Dr. Jin Huang, who has extended a great amount of assistance in my work. I would also like to thank Dr.Krishnaprasad Thirunarayan and Dr. Bin Wang for taking their time to evaluate my work. I am very thankful to my family for always supporting me in my long-term education career.

Introduction

We propose a system to generate fingerprints for WordPress plugins that can be used as searchable keys to finding what websites have installed the respective plugin (s). Our system makes use of syntax analysis, to analyze functions that generate the front-end content including HTML tags, themes, and JavaScript from the source code, then abstract the outcomes with regular expressions to generate fingerprints for every scanned plugin. We later use these fingerprints to search through web crawlers and find involved websites, especially vulnerable webs. Then we verify the correctness of those results and analyze the distribution and pattern of our fingerprints in vulnerable plugins compare to regular plugins.

1.1 Motivation

WordPress is one of the most common and popular content management system (CMS) tools in the world, used by more than 40 percent of the top 10 million websites as of May 2021 [1]. What makes WordPress so popular is the ease with which users can do customization and extension through WordPress plugins [2]. However, such popularity has also made it a significant target for web attackers, a single vulnerable WordPress plugin could lead to a variety of attacks and unexpected losses to web owners. Besides, the WordPress plugin itself may expose the information of the current plugin to the external world. That information, including HTML tags of the plugin, the current version of the plugin and JavaScript, and the full path of the plugin on the server, could potentially be collected by attackers and lead to other more severe problems once there are vulnerabilities exposed in those plugins.

It is, therefore, necessary to check WordPress plugins and see what information will be exposed, then use that information to find what websites have installed vulnerable plugins. However, manually extracting the random front-end content from webs could be tough, especially when handling large search works. For this purpose, we have built a system, that can automatically scan plugins and produce easily searchable fingerprints for them.

1.2 Background

Before illustrating the detailed design of the system, we would like to discuss how plugins work on WordPress websites and what factors are important for analyzing the front-end behavior of plugins.

1.2.1 Load Plugins in WordPress

Briefly speaking, a plugin is an application programming interface (API), WordPress features various APIs for use. Every API supports users to interact with WordPress in different ways [2]. One of the very commonly used APIs is the plugin. The plugin provides a set of hooks, that enable functions to access different sections of WordPress. Hooks are special functions in WordPress, that allow users to execute their customized functions of a plugin at some very points during the loading of the website. For instance, a developer can write a function to display a welcome message after a user logs into the website, then use hooks to attach that function to the web.

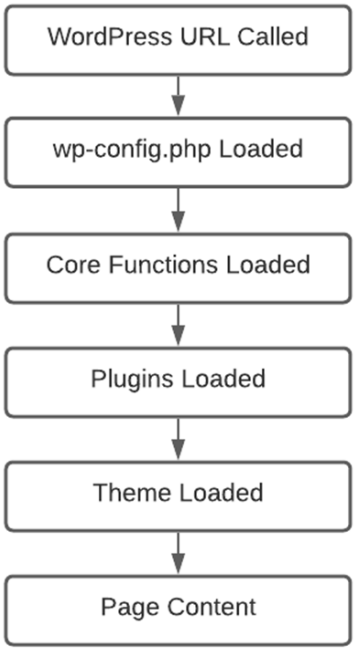

Figure 1.1: The Loading Process of WordPress Plugins

Figure 1.1 presents a brief loading process of WordPress website. When visiting a WordPress website, after the web URL is called, some core functions in 'wp-config.php' are loaded, then it will go through the current plugin folder to load all active plugins. After that, WordPress will start loading themes, in the process of loading themes, a variety of customized functions from the active plugins will be called through hooks to do their corresponding works to the front-end page. For example, display some customized themes or enqueue scripts to the page content.

1.2.2 Hooks in WordPress

When loading a WordPress page, before HTML is returned to the client, WordPress goes through some code that determines what the client-side needs to see. During this process, hooks are executed. The plugin adds required callback functions for every hook in a queue, based on the order of their priority. When a hook gets triggered, it calls all callback functions that are attached to that hook. Then all the resulting HTML and CSS make up what is ultimately returned to the browser and seen by the users. From the perspective of

the WordPress front-end page, three hooks are virtually required in any theme because most plugins use them. [3] Although two of those hooks, 'wp_head()' and 'wp_footer()', look like function calls, buried underneath the function are hooks. 'wp_head()' and 'wp_footer' are very popular for developers who want to add some customized themes to the head or the foot on their page. The third one is 'comment_form', positioned in the WordPress comment form and commonly used when developers need to customize the comment section.

1.2.3 Action Hooks

These three hooks: 'wp_head()' ,'wp_footer()' and 'comment_form', are loaded by default when a WordPress page runs. Besides, these hooks need further action hooks to call functions with them to customize the page. As shown in Figure 1.2, the underline in red marks the front-end default hooks, and the underline in black marks action hooks that connect the customized functions with the front-end default hooks. Next, we introduce three action hooks that are significant to us in analyzing the front-end behavior of WordPress plugins.

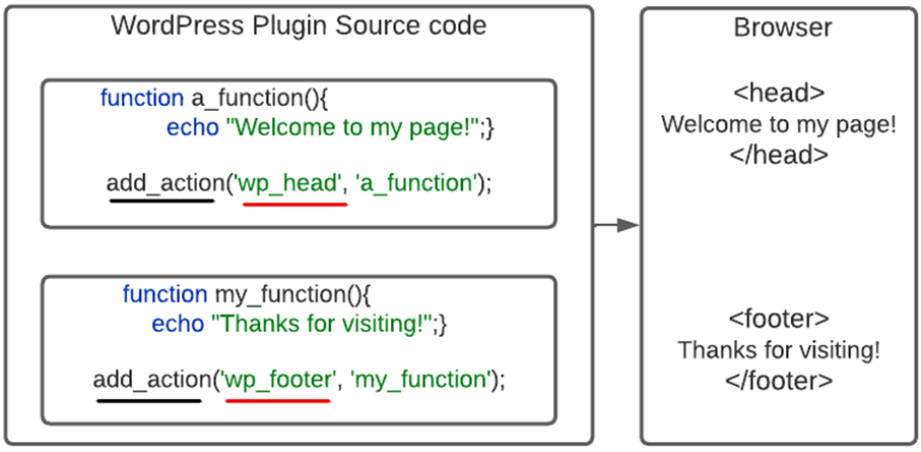

Figure 1.2: How Customized Functions Makes Changes to Page Content via Hooks

- 'add_action': Many hooks are called by the 'add_action()' function, in the most basic case, 'add_action()' takes two arguments. The first argument is the hook name, for example, 'wp_head'. The second argument is the callback function that passes to the first hook, for example, 'my_function', is a string that presents the name of the function. Mostly, the second argument is a customized function written by users in a plugin [4].

- 'add_shortcode': Shortcodes in WordPress are added by inserting a tag in brackets to the content editor. For example, a 'display' shortcode in WordPress is written like: [display]. When the content of the post is parsed, shortcodes are extracted and related functions are triggered in their place [3]. Similar to 'add_action()', 'add_shortcode()' also takes two arguments, the first argument refers to the shortcode tag and the second argument refers to the customized callback function. Continue with the [display] example, in the source code, it will be written as 'add_shortcode ('display', 'function_display')', where 'function_display' is the name of the callback function.

- 'wp_enqueue_scripts': WordPress supports developers to add style sheets and JavaScript files dynamically by using a simple function hook—'wp_enqueue_scripts()' [3].

  'wp_enqueue_scripts()' takes multiple parameters, including the name of the script, the full URL of the script, script dependencies, and the current version information. Among those parameters, the most significant two are the script name and script full URL, because they will display the current JavaSript version to the browser which also includes the full path of the plugin in the URL.

Therefore, by analyzing the customized functions that are called back through these action hooks, we will be able to determine what information a plugin will display to the client-side.

1.2.4 Abstract Syntax Tree

We have designed the system to perform static analysis by leveraging an abstract syntax tree (AST) derived from WordPress plugins. The source code of plugins will be parsed into ASTs through PHP Parser, a library that takes PHP source code and analyzes the code with a lexical analyzer, then generate its corresponding syntax tree. [5]

1.3 Goals and Challenges

There are three general goals that we wish to accomplish with our system. First, the system needs to produce fingerprints automatically without extra manual costs. Besides, the system is expected to be highly scalable considering the increasing growth number of new WordPress plugins and potential vulnerabilities. Last, the fingerprints, generated by the system, are supposed to be able to precisely identify the respective plugin. Consequently, as discussed above, there are three main challenges in building this system, automation, scalability, and effectiveness. To overcome these challenges, we have made the following contributions:

- We designed the system to analyze the Abstract syntax tree of the plugins, the tree structure allows us to create a recursive analysis technique that readily overcomes the constraints of dealing with program scope diversity and variable repetition in all potential conditional outcomes.
- By recursively analyzing the AST, if further new plugins need to be analyzed, we just simply add new conditions to the core function without making any other changes, hence keeping the scalability of the system.
- For every set of fingerprints generated by our system, we compared the Intersection over Union for every two sets of them and proved their effectiveness.

1.4 Organization

We have described the motivation, background, and goals in this thesis. In Chapter 2, we will discuss other similar works which are related to this thesis and how our system differs from them. Then in chapter 3, we introduce the detailed design of the system and the approaches we have used to overcome the challenges in building the system. Chapter 4 introduces the experiments we have accomplished with the generated fingerprints. Chapter 5 discusses the weakness of our current system and potential approaches to solve them, and finally, Chapter 6 concludes the overall thesis.

Related Work

According to Daniel et al., [6], the sensitivity of security scanner plugins in WordPress to vulnerabilities in vulnerable plugins varies significantly, some plugins that are developed as security scanners performed even more expressive than others in the reporting detected vulnerable plugins. Although the WordPress core is considered relatively secure, a large number of WordPress plugins are not safe. Besides, none of the security scanner plugins tested in the study were able to appropriately address those vulnerabilities. Given the reporting result of the paper, WordPress is not only the most famous CMS in the world but also possibly most attacked CMS tool.

On the other hand, several studies have also focused on scanning web applications. Next, we will discuss some of the existed approaches as well as the difference between our system and these tools.

2.1 Creating Semantic Fingerprints for Web Documents

Katrin et al. [7] introduced an approach to generate semantic fingerprints for web documents. Their system generates semantic fingerprints by retrieving the web content in the DOM tree. The system detects all nouns from the content using NLP methods and discovers the stems of these words in the content, computes the term frequency tf, and stores the distinct noun stems in a set K = k1,..., kn along with their computed tf value. The list of terms, referred to as keywords, will be used as input for the algorithm later, then it generates a semantic fingerprint. The fingerprint in this paper is represented as a graph that

illustrates the concepts and the relations of the respective resource.

2.2 Patcher

Patcher [8] is a web service platform that assists users in detecting and patching potential vulnerabilities. The tool takes two inputs, web application source code and attack patterns that describe malicious strings for certain vulnerabilities. It uses static string analysis and integrates the service front end with a 3D program visualization interface so that users can access and examine the analyzed results as well as how vulnerabilities are patched. However, Compared to our system, the way that Pacther detects vulnerabilities for webs is relatively harder in use where it requires providing the malicious strings for every certain vulnerability.

2.3 Anomaly Detection Using Negative Security Model

Auxilia et al.[9] proposed a negative security model based on web application misuse, it supports a Web Application Firewall(WAF) engine that ensures vital protection through different web architectures and allows HTTP traffic monitoring without modifications to the current infrastructure. Unfortunately, although it could detect all kinds of common attacks as an anomaly detection system, simultaneously, it will cause a high false-positive rate.

Design

In this chapter, we introduce the design of our system and the approaches we have used to accomplish our goals.

3.1 System Structure

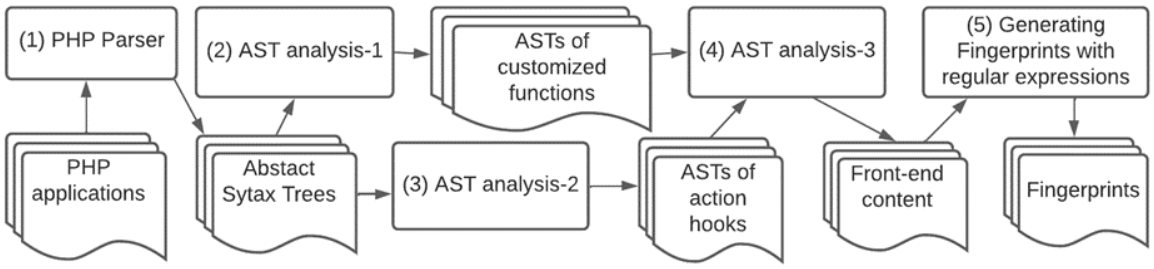

Figure 3.1: Architecture of System

Figure 3.1 shows a general overview of our system, which illustrates 5 major steps.

- Parse Source Code with PHP-Parser: The original input of the system is a set of PHP applications (WordPress plugins), the first step of our analysis is to parse the PHP applications and then construct the Abstract Syntax Trees (ASTs) by PHP Parser.
- AST analysis-1: In the AST analysis-1, the system traverses the whole AST to get all user-defined functions from the plugin, then saves them.
- AST analysis-2: In this phase, the system gets the action hooks, extracts the name of the callback functions, and saves it.

- AST analysis-3: In the AST analysis-3, the system analyzes the results from the previous two steps, gets ASTs of the custom functions that will be triggered through the front-end action hooks, then save all these contents to an array.

- Generating Fingerprints with Regular Expression: In the last phase, our system handles the unprocessed contents that are generated from the previous step with Perl-Compatible Regular Expressions (PCRE) of PHP, then generate the searchable fingerprints that can be easily used later.

3.2 Detailed AST Analysis

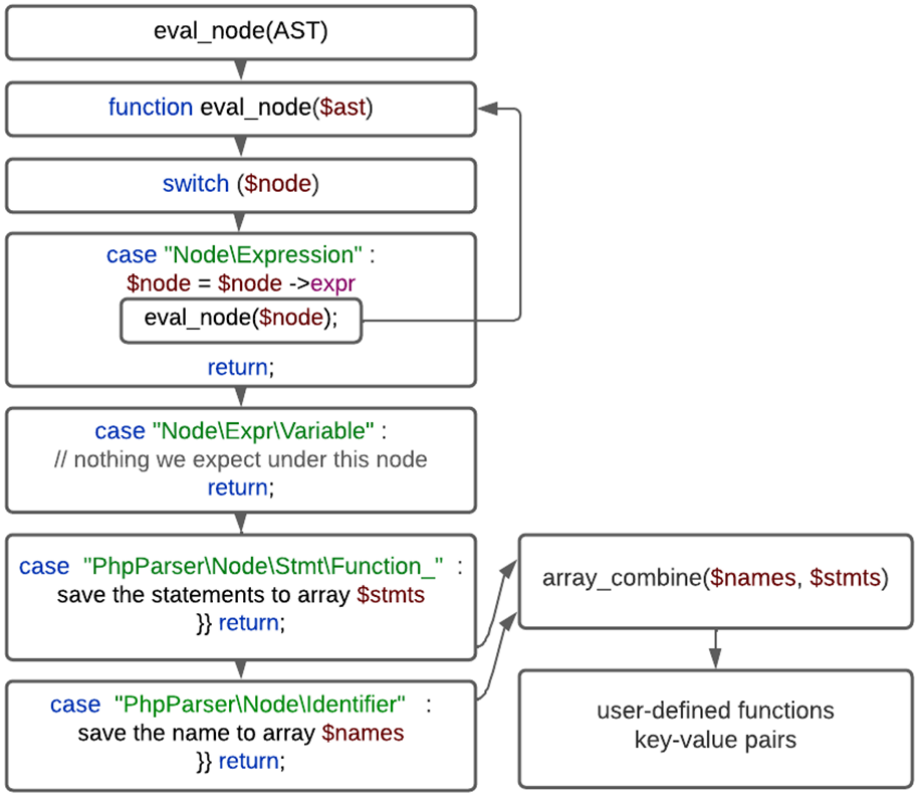

Figure 3.2: Save User-defined Functions as Key-value Pairs

As illustrated in the previous section, the system implements three times AST analysis to get the expecting result. Figure 3.2 shows a brief process of AST analysis-1, after

getting AST (s) from PHP Parser, the system calls its main function 'eval_node' to start the AST analysis. In the main function, the system recursively evaluates every node of the AST by a switch statement, every case represents a node of the tree, if the current node does not have any expected information, the system will call itself to evaluate the next node, until there is no more next node, it will just return. When a node meets the expected conditions, the system will then save it and return. All user-defined functions are under node 'PhpParser\Node\Stmt\Function_', its sub-node 'PhpParser\Node\Identifier' has all the user-defined function names and '[stmts]' contains all the corresponding function statements. Then the system saves the user-defined function names as keys and their corresponding statements as values into an array as key-value pairs.

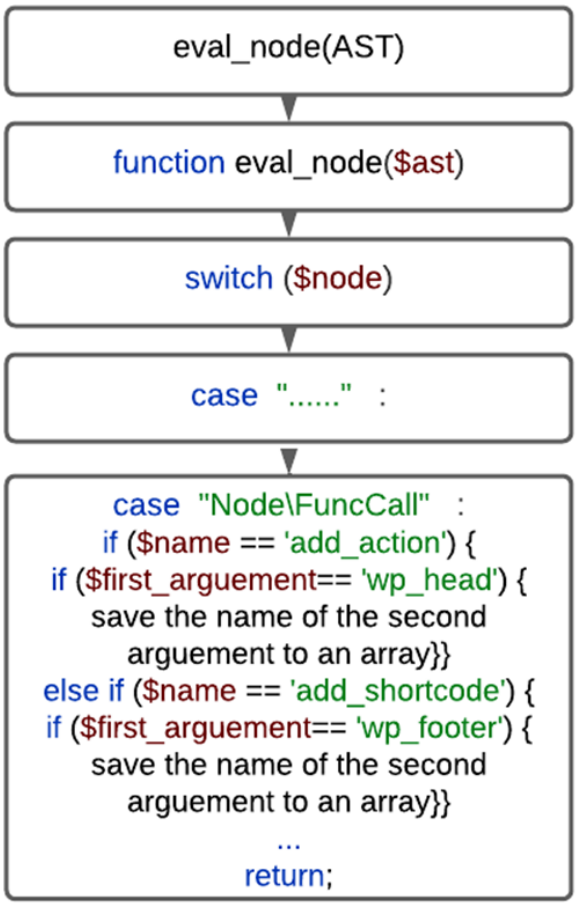

Figure 3.3: Save Functions That Are Called by Default Hooks

In the phase of AST analysis-2, the system traverses the tree to check all the action

hooks. Figure 3.3 presents a simple example of saving expected function names from AST. At the syntax level, action hooks are function calls, hence they are all represented under the node of "Node\FuncCall". The system checks the names of those function calls, if any of them match with the three action hooks that we have discussed previously ('add_action', 'add_shortcode', 'wp_enqueue_script'), the system continues to check whether their first argument matches with the WordPress default theme hooks ('wp_head', 'wp_footer', 'comment_form'). If both conditions are satisfied, the second argument should be the name of a callback function, the system saves them in an array for the next phase.

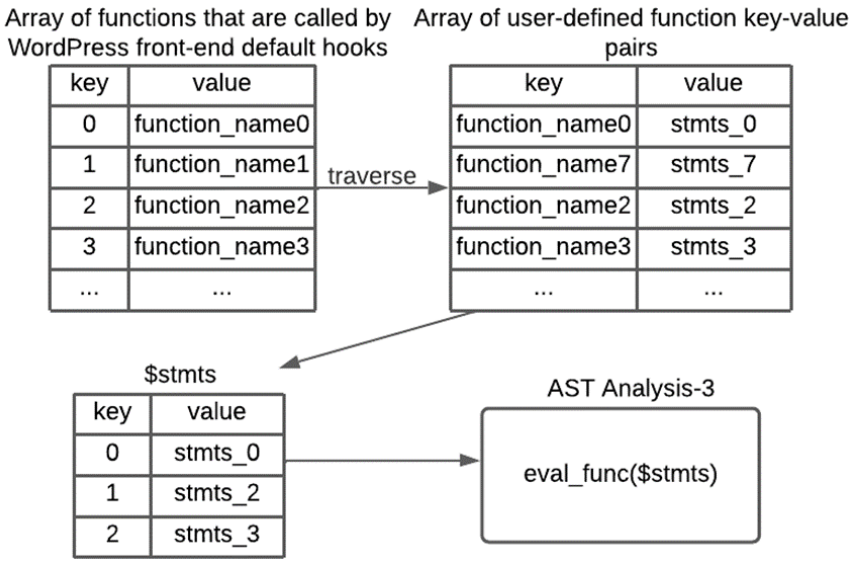

Figure 3.4: Save Functions That Are Called by Default Hooks

After the previous two phases, the system now has the key-value pair of user-defined functions, as well as the name of functions that are called back by WordPress front-end default hooks. As shown in Figure 3.4, the system uses those called function names to traverse the user-defined function key-value pairs, if a name matches a key, we pass the corresponding value (statements) to the AST analysis-3 by calling the function 'eval_func()'.

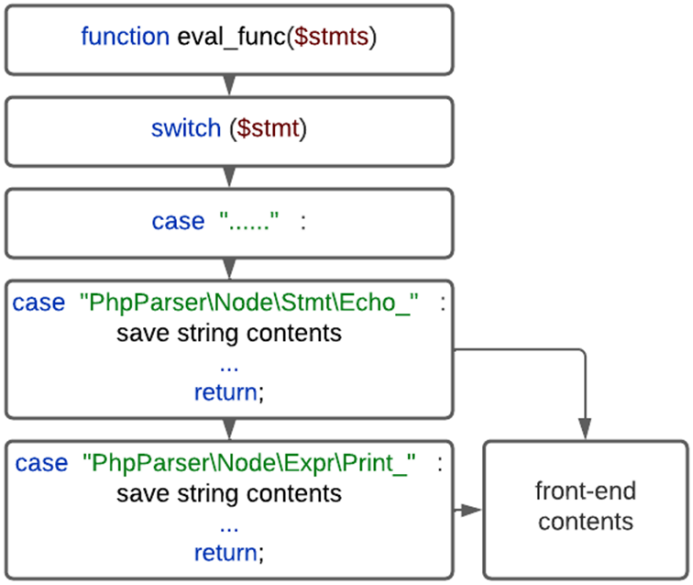

Figure 3.5: AST Analysis-3

In the AST analysis-3, the system will simply check if there are function calls that will output any information to the front-end of the client-side, for example, HTML code, JavaScript, and plain text.

As the example presented in Figure 3.5. 'PhpParser\Node\Stmt\Echo_' presents the function call 'Echo' in AST, 'Echo' does nothing but prints some contents. Similar to 'Echo', 'Print' is another function call that prints certain contents when executed, presented as 'PhpParser\Node\Expr\Print_' . Hence under these two conditions of the AST, the system extracts those contents that will be output to the page for later to finally generate fingerprints.

3.3 Generate Fingerprints with Regular Expression

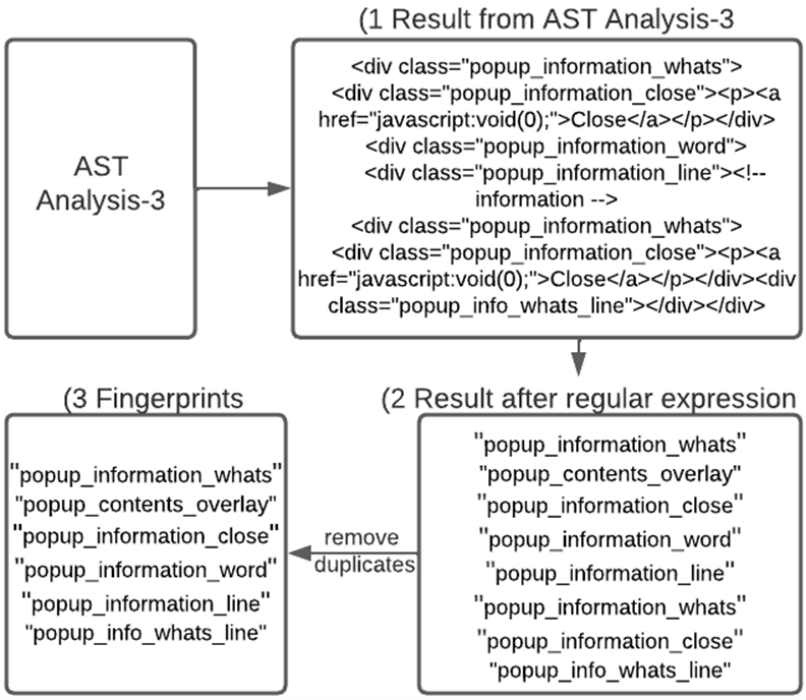

Figure 3.6: Sample of Regular Expression Match

As shown in phase (5 of Figure 3.6, the system finally accomplishes regular expression to produce the fingerprints. The outcomes we have got from phase (4 are unprocessed contents, they are either arbitrary plain texts or HTML tags, and they may contain unexpected information hence not easy to use. Consequently, the system simplifies the result by using PHP PCRE functions. As shown in Figure 3.6, phase (1 is the original outcome from AST analysis-3, we only extract the class names and IDs from phase (1 with regular expression, the result are shown in (2 of Figure 3.6. As we discussed in chapter 1, considering the effectiveness of the system, the fingerprints should not contain duplicates, therefore, the system next checks and removes all duplicates from the fingerprints and finally produce the result as shown in (3.

3.4 Fingerprints Types

There are two general types of fingerprints, one of them are as presented in phase (3 Figure 3.6, they are mainly from the branch of two action hooks, ‘add_action’ and ‘add_shortcode’. Another type of fingerprint is ‘wp_enqueue_scripts’, it contains nothing but a single line of URL. For example, one WordPress plugin named ‘imagements’ uses ‘wp_enqueue_scripts’ to leverage some JavaScript to the page, which will result in a line of code to the browser as:

<script src=”http://mywordpress/wp-content/plugins/imagements/js/form_tag.js”.>

Since it is required to use a unique name for every plugin in WordPress, the URL in the code above is sufficient to precisely identify the plugin ‘imagements’, and the system just simply saves its name as a fingerprint. It seems unnecessary in the ’wp_enqueue_script’ branch since what the system has produced is just the plugin name, one could have simply copied the plugin name for use rather than analyzing it. However, it is still very meaningful because analyzing it significantly lowers the false negatives in detecting vulnerable websites, especially when handling a large amount of searching work.

Experiments

In this chapter, we will discuss some experimental results that we have accomplished with our system.

4.1 Dataset

To prove the functionality of our system, we have downloaded 45725 plugins from WordPress's official site and generated fingerprints for 10053 plugins with our system. The current generating rate reached approximately 22 percent. As we discussed in chapter 3, this rate could be further improved by simply adding more filtering conditions to the system. Besides, the 10053 sets of fingerprints are relatively sufficient for the current experiments. On the other hand, we have collected 40 vulnerable plugins with verified CVEs and generated the respective fingerprints for each of them, we will discuss the details of the analysis in the following sections.

4.2 Evaluating The Similarity of Generated Fingerprints

It could be possible that the fingerprints are duplicated within two different plugins and therefore not accurate to identify each of them. Consequently, we did a similarity comparison by using the intersection of two sets of fingerprints divided by their union to present their similarity. Figure 4.1 shows the distribution of similarities for 10053 analyzed WordPress plugins, X-axis represents the similarity percentage and Y-axis represents their

probability, among more than 4.9 million times of comparisons, only 2470 times are 100 percent duplicated, the majority of similarity comparisons are lower than 5 percent. Hence proved the accuracy and effectiveness of the fingerprints.

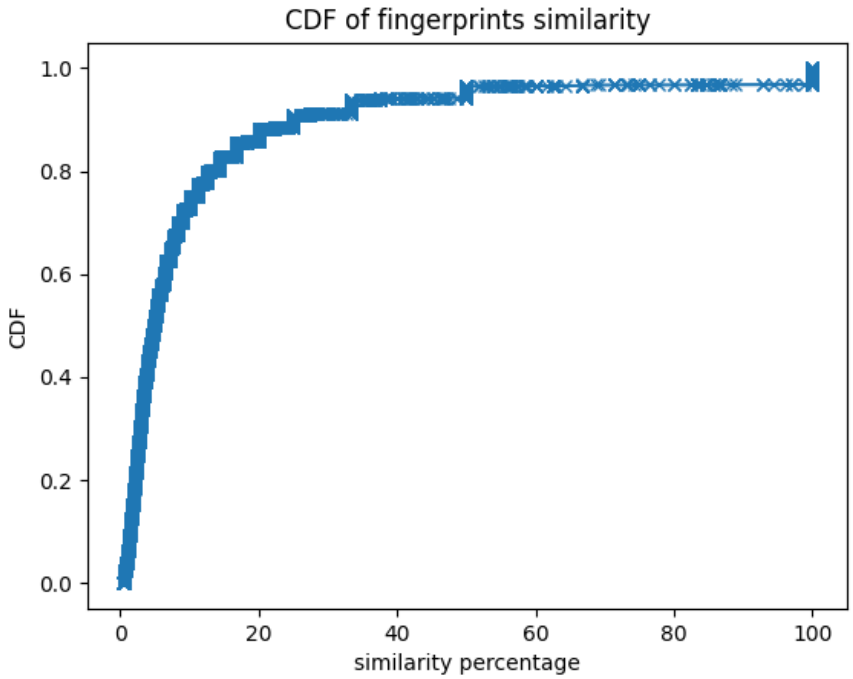

Figure 4.1: CDF of Fingerprints Percentage

4.3 Applications of Fingerprints

We have proved the fingerprints are in good effectiveness for identifying their plugins, Now we introduce the experiments we have finished with fingerprints. First, we illustrate an example of one set of fingerprints.

"popup_information_whats"
"popup_contents_overlay"
"popup_information_close"
"popup_information_word"
"popup_information_line"
"popup_info_whats_line"

Figure 4.2: Fingerprints for ”whats-new-popup-generator”

Listing 4.1: Partial HTML of a Sample Web

```
<div class="popup_contents-overlay"></div>
<div class="popup_information_whats"></div><!-- information -->
    <div class="popup_information_whats">
        <div class="popup_information_close"><p><a ...
            href="javascript:void(0);">Close</a></p></div>
        <div class="popup_information_word">
            <div class="popup_information_line">
                <a href="https://najapan.org/meeting">...</a>
                    <div class="popup_info_whats_line"></div></div>
        </div>
    </div>
```

As shown above, Figure 4.2 is the fingerprints set of a plugin named “whats-new-popup-generator”, we used those fingerprints as the keyword to search in the web crawler to get the corresponding result. When we got the result from the data crawler, we visit the listing webs and analyze their source code. Listing 4.1 is the front-end source code we have

copied from one of the webs we have got.

The data crawler we have used in searching has large WordPress websites source code database and supports direct source code search, namely NerdyData. As presented in Figure 4.3, we put all the tags of the signature using logic AND, hence we will get webs with all those fingerprints in their front-end source code. The result for "whats-new-popup-generator" is shown in Table 4.1.

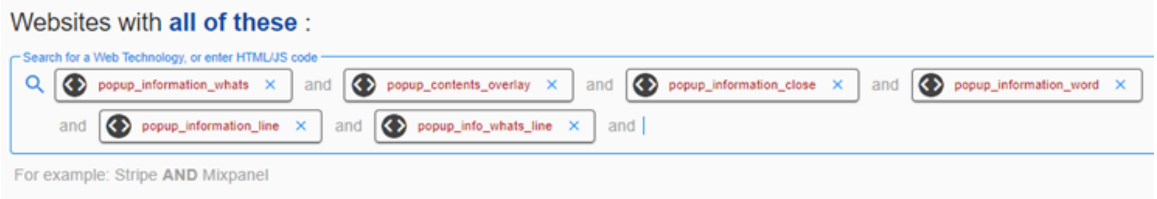

Figure 4.3: A Screenshot of Using NerdyData

In Table 4.1, page rank represents the popularity of the websites based on the domain's backlinks, in verifying the results, we first verify if the websites are WordPress websites or not, if it is, we continue to compare if the plugin's interface is matched or there are some functions in the client-side source code that are matched with in the plugin, then we mark the result as verified.

Table 4.1: Result for "what-new-popup-generator" from NerdyData, Date 03/09/2022

Rank	**Domain**	**WordPress page**	**Prove of evidence**
1	najapan.org	yes	Plugin functions matched
2	yozhizumikaihatsu.com	yes	Plugin functions matched

4.4 Result Verification

In the previous example of ”what-new-popup-generator”, we have found two websites that have installed the tested plugin. However, further evidence is needed to prove the result. As discussed in the previous section, we searched on the web crawler with the fingerprints, if there are other functions that are matched to the plugin, then we can prove the correctness of the result..

Listing 4.2: Partial Source Code from ”what-new-popup-generator”

```
1  <script type="text/javascript">
2  jQuery(function($){
3      $('.popup_contents-overlay').css('background-color', '
4      <?php echo ...
           esc_js(get_option('popup_plugin_value_whatscolor')); ?>');
5  });
6  </script>
7  <script type="text/javascript">
8  jQuery(function($){
9      $('.popup_information_box').click(function(){
10         $('.popup_contents-overlay, ...
               .popup_information_whats').fadeIn();
11     });
12 $('.popup_contents-overlay,.popup_information_close').click(function()
13     {
14     $('.popup_contents-overlay, .popup_information_whats').fadeOut();
15             });
16         });
17     </script>
```

Listing 4.2 shows a part of the source code from the tested plugin, and Figure 4.4 is a screenshot from one of the websites we have got by the previous search in the data crawler.

We can see that the website has the same function as the tested plugin, hence proving the correctness of our results.

```
<script type="text/javascript">
jQuery(function($){
    $('a[href="pop_void"]').click(function(){
        return false;
    })
});
</script>


        <script type="text/javascript">
    jQuery(function($){
        $('.popup_contents-overlay').css('background-color', '#000000');
    });
</script>

<script type="text/javascript">
    jQuery(function($){
        $('.popup_information_box').click(function(){
            // ブラウザ全体を暗くする
            $('.popup_contents-overlay, .popup_information_whats').fadeIn();
        });

        $('.popup_contents-overlay,.popup_information_close').click(function(){
            // ブラウザ全体を戻す
            $('.popup_contents-overlay, .popup_information_whats').fadeOut();
        });
    });
</script>
```

Figure 4.4: A Screenshot of Searched Website

4.5 Vulnerable Plugin Example

Table 4.2 presents the result for a vulnerable plugin named “wp-curriculo-vitae” [10]. Although the plugin has already been reported as vulnerable, there are 11 websites verified by us that still have it installed, which could be a significant risk for those web owners.

Table 4.2: Result for Plugin “wp-curriculo-vitae” from NerdyData 4/09/2022

Rank	Domain	WordPress page	Prove of evidence
1	http://www.williamluis.com.br/	yes	Plugin functions matched
2	http://www.idbr.org.br/	yes	Not verified
3	https://facol.br/	yes	Plugin functions matched
4	http://www.scale.com.br/	yes	Plugin functions matched
5	https://gutemberg.com.br/	yes	Plugin functions matched
6	https://www.brmfundicoes.com.br/	yes	Plugin functions matched
7	http://ebrasilenergia.com.br/	yes	Plugin functions matched
8	http://www.barradecomercio.org/	yes	Plugin functions matched
9	http://www.korum.com.br/	yes	Plugin functions matched
10	https://rimarcontabilidade.com.br/	yes	Plugin functions matched
11	https://instaloengenharia.com.br/	no	Not verified
12	https://indiceconsultoria.com.br/	yes	Not Verified
13	https://inwall.com.br/	yes	Plugin functions matched
14	https://sienaffari.it/	yes	Plugin functions matched
N/A	http://sindcomcf.com.br/	yes	Not verified

4.6 Searching Vulnerable Plugins

WordPress records all the reported vulnerable plugins on its official security website wpscan.com [11], including some common and popular vulnerabilities, cross-site scripting, SQL injection, and upload. Therefore, we have collected 100 vulnerable plugins with different types of security risks from wpscan.com, then generated 35 sets of fingerprints that are ready to use on the data crawler. With those fingerprints, we explored that various websites have still been running different kinds of vulnerable plugins, the details will be illustrated in the following subsections.

4.6.1 Upload

As presented in Table 4.3. As the date of March 9th 2022, There are 42 websites we have explored with fingerprints and 34 of them have been verified that vulnerable plugins with upload issues are installed on their server. [12] [13] [10] [14] [15] [16] [17] [18].

Table 4.3: Result for Vulnerable Plugins (upload) from NerdyData Date 03/09/2022

Application	Version	Explored Websites	Verified Websites	Reference
gallerio	v1.2 No known fix	5	1	N/A
scroll-baner	v1.0 No known fix	3	2	CVE-2021-24642
simple-schools-staff-directory	v1.1 No known fix	4	4	CVE-2021-2466
wp-curriculo-vitae	v6.3 No known fix	17	15	CVE-2021-24222
imagements	v1.2.5 No known fix	2	2	CVE-2021-24236
art-picture-gallery	v1.2.9 No known fix	4	4	CVE-2018-9206
baggage-freight	v0.1.0 No known fix	1	1	CWE-434
automatic-grid-image-listing	v1.0 No known fix	1	1	CVE-2021-25119
wbcom-designs-buddypress-search	v1.2 No known fix	1	1	CWE-862
sermon-browser	v0.45.22 No known fix	4	3	CWE-434

4.6.2 Cross-Site Scripting

As shown in Table 4.4, there are 134 websites we have explored from the data crawler and 123 Websites are at risk of Cross-Site Scripting issues with 10 vulnerable plugins. [19]

[20] [21] [22] [23] [24] [25] [26] [27] [28].

Table 4.4: Result for Vulnerable Plugins (Cross-Site Scripting) from NerdyData Date 03/09/2022

Application	Version	Explored Websites	Verified Websites	Reference
4k-icon-fonts-for-visual-composer	v1.21 No known fix	3	3	CVE-2021-24435
crazy-bone	v0.6.0 No known fix	5	4	CVE-2022-0385
youtube-feeder	v2.0.1 No known fix	25	24	CVE-2021-34633
sola-newsletters	v4.0.2 No known fix	53	50	CVE-2021-34634
cleeng	v2.3.2 No known fix	1	1	CVE-2013-1808
comment-attachment	v1.0 No known fix	1	1	CVE-2013-6010
contact-form-generator	v2.1.86 No known fix	15	12	CVE-2015-6965
my-chatbot	v1.1 No known fix	26	23	CWE-79
light-messages	v1.0 No known fix	2	2	CVE-2021-24535
social-tape	v1.0 No known fix	3	3	CVE-2021-24411

4.6.3 SQL Injection

There are 73 websites we have explored through the data crawler and 64 Websites are at the risk of SQL injection issues with 10 vulnerable plugins, shown in Table 4.5. [29] [30] [31] [32] [33] [34] [35] [31] [36] [37].

Table 4.5: Result for Vulnerable Plugins (SQL injection) from NerdyData Date 03/09/2022

Application	Version	Explored Websites	Verified Websites	Reference
wpcalc	v2.1 No known fix	10	10	CVE-2021-25054
chameleon-css	v1.2 No known fix	11	10	CVE-2021-24626
game-server-status	v1.0 No known fix	1	1	CVE-2021-24662
wp-display-users	v2.0.0 No known fix	4	4	CVE-2021-24400
dbox-slider-lite	v1.2.2 No known fix	25	22	CVE-2018-5374
wp-email-users	v1.7.6 No known fix	3	3	CVE-2021-24959
mwp-forms	v3.1.3 No known fix	2	2	CVE-2021-24628
g-auto-hyperlink	v1.0.1 No known fix	7	7	CVE-2021-24627
author-chat	Fixed in v2.0.0	3	1	CWE-89
wicked-folders	Fixed in v2.8.10	7	6	CVE-2021-24919

4.6.4 Bypass

Among the total number of 259 explored websites, 222 webs have been verified that installed plugins with bypass vulnerability. The details are presented in Table 4.6. [38] [16] [39] [40]

Table 4.6: Result for Vulnerable Plugins (Bypass) from NerdyData Date 04/11/2022

Application	Version	Explored Websites	Verified Websites	Reference
woo-nmi-three-step	v1.6.11 No known fix	10	10	CWE-352
123contactform-for-wordpress	v1.5.6 No known fix	10	8	CWE-434
spam-free-wordpress	v1.9.2 No known fix	50	47	CWE-287
rw-divi-unite-gallery	v1.0.0 No known fix	3	3	CWE-287
opal-estate	v1.6.11 No known fix	37	36	CWE-352
rays-grid	v1.2.2 No known fix	68	50	CWE-352
wp-symposium	v12.7.7 No known fix	19	19	CWE-287
mac-dock-gallery	v3.1 No known fix	6	4	CWE-287
sell-media	Fixed in v2.5.7.3	28	19	CWE-352
simple-student-result	Fixed in v1.6.4	28	26	CVE-2017-14766

4.7 Result Analysis

After the previous experiments, we collected 40 vulnerable plugins and found 446 websites that have installed and activated those vulnerable plugins. Next, we move forward to analyze the numbers of fingerprints in vulnerable plugins compared to all plugins.

Figure 4.5 and Figure 4.6 present the CDF of fingerprints numbers in all plugins and vulnerable plugins. From the tables we can see that there is no significant difference in comparing the fingerprints numbers between two sets of plugins.

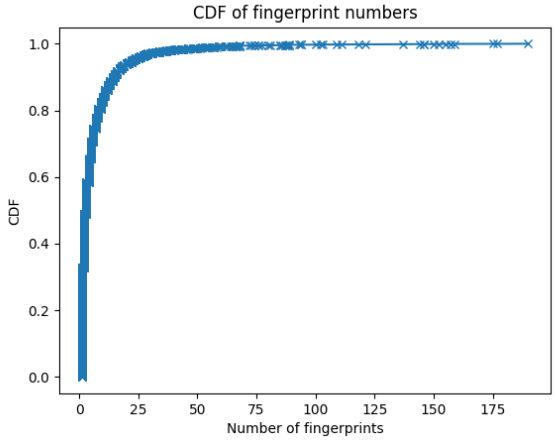

Figure 4.5: CDF of Fingerprints Numbers in Regular Plugins

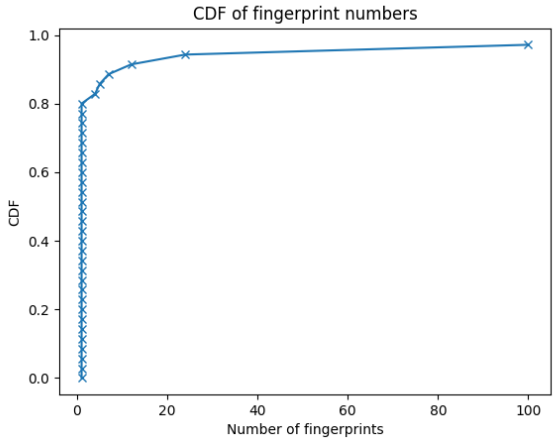

Figure 4.6: CDF of Fingerprints Numbers in Vulnerable Plugins

However, as shown in Table 4.7, both average and median fingerprints numbers in all plugins are greater than in vulnerable plugins, illustrating that vulnerable plugins tend to have fewer numbers of fingerprints based on our current experiments.

Table 4.7: Fingerprints Numbers Comparison

	Vulnerable plugins	**All plugins**
Average of fingerprints numbers	5.17	11.35
Median of fingerprints numbers	1	4

Discussion

Given that our system can successfully generate fingerprints for WordPress plugins and we have explored various websites with the generated fingerprints. However, in the experiment of searching vulnerable websites, there are also considerable numbers of websites that either failed for verification or had wrong results. According to Table 5.1, among the total number of 508 explored websites, 445 of them are verified that have installed vulnerable plugins, 63 of them are failed for verification and hence false positives. The overall false-positive rate (FPR) is 12.4 %, this number is relatively high due to the large random content of websites, some code matches the fingerprints but not actually installed the plugins.

Table 5.1: Vulnerable Plugins Evaluation Results

	Explored webs	**True Positive**	**False Positive**
Upload	42	34	8
Cross-Site Scripting	134	123	11
SQL injection	73	66	7
Bypass	259	222	37

With the current experiments, we have two challenges left for the potential extension of our future work. First, we need to improve the generating rate for overall plugins. Among the total number of 45725 plugins, we currently have generated 10053 plugins with fingerprints. As we discussed in the previous chapter, the generating rate could be further improved by adding more conditions in the AST analysis. However, this would be tough when some complex conditions need to be satisfied with syntax analysis. We believe this

problem could be solved by leveraging semantic analysis in AST. Besides, plugins with a large number of fingerprints are not easy for searching, in those fingerprints, many of them are relatively too common and not suitable as searchable keys. This problem could be solved by improving our regular expression matching. we can filter some of the very common fingerprints, for example, 'a', 'form', 'here', 'table' etc.

Conclusion

We have discussed our system: Fingerprints Generator in this thesis, We illustrated the design of the system by extending AST analysis. Then we have proved that the system performs properly with effectiveness and scalability. Besides, we have analyzed over 48000 WordPress plugins and generated fingerprints for over 10000 plugins. We also explored and verified 446 websites that have installed vulnerable plugins. Then we have discussed the current weakness of the system and possible ways to overcome it. We believe there could be more significant results to be produced by extending our work in the future.

Bibliography

[1] Web Technology Surveys. https://w3techs.com/.

[2] Williams Brad. *Professional WordPress Plugin Development*. Wrox, second edition, 2020.

[3] Onishi Adam. *Pro WordPress Theme Development*. Apress, first edition, 2013.

[4] Brazell Aaron. *WordPress Bible*. Wiley, second edition, 2011.

[5] Sikora Martin. *PHP Reactive Programming*. Packt, first edition, 2017.

[6] Daniel T. Murphy, Minhaz F. Zibran, and Farjana Z. Eishita. Plugins to detect vulnerable plugins: An empirical assessment of the security scanner plugins for wordpress. pages 39–44, 2021.

[7] Katrin Krieger, Jens Schneider, Christian Nywelt, and Dietmar Rösner. Creating semantic fingerprints for web documents. 2015.

[8] Fang Yu and Yi-Yang. Tung. Patcher: An Online Service for Detecting, Viewing and Patching Web Application Vulnerabilities. pages 4878–4886, 2014.

[9] Auxilia. D and Tamilselvan.D. Anomaly detection using negative security model in web application. pages 4878–4886, 2010.

[10] CVE-2021-24222. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-24222.

[11] WPScan. https://wpscan.com/plugins.

[12] CVE-2021-24642. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-24642.

[13] CVE-2021-2466. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-2466.

[14] CVE-2021-24236. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-24236.

[15] CVE-2018-9206. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-9206.

[16] CWE-434. https://cwe.mitre.org/data/definitions/434.html.

[17] CVE-2021-25119. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-25119.

[18] CWE-862. https://cwe.mitre.org/data/definitions/862.html.

[19] CVE-2021-24435. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-24435.

[20] CVE-2022-0385. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-0385.

[21] CVE-2021-34633. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-34633.

[22] CVE-2021-34634. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-34634.

[23] CVE-2013-1808. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-1808.

[24] CVE-2013-6010. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-6010.

[25] CVE-2015-6965. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-6965.

[26] CWE-79. https://cwe.mitre.org/data/definitions/79.html.

[27] CVE-2021-24535. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-24535.

[28] CVE-2021-24411. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-24411.

[29] CVE-2021-25054. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-25054.

[30] CVE-2021-24626. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-24626.

[31] CVE-2021-24627. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-24627.

[32] CVE-2021-24400. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-24400.

[33] CVE-2018-5374. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-5374.

[34] CVE-2021-24959. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-24959.

[35] CVE-2021-24628. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-24628.

[36] CWE-89. https://cwe.mitre.org/data/definitions/89.html.

[37] CVE-2021-24919. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-24919.

[38] CWE-352. https://cwe.mitre.org/data/definitions/352.html.

[39] CWE-287. https://cwe.mitre.org/data/definitions/287.html.

[40] CVE-2017-14766. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-14766.