Alexander Lotero

Squid Security - Web Challenge Report

Vulnerability 1

Name: Reflected Cross-Site Scripting (XSS) on /profile

Risk Rating: Medium (6.5)[1]

Exploitation Likelihood: High

Potential Impact: Medium

Description: The application is not properly sanitizing user input within the /profile page, allowing for arbitrary JavaScript execution. This provides malicious actors with the opportunity to run scripts that, when run as other users, can lead to session hijacking and sensitive information disclosure.

Remediation: We advise that the developers implement strict input sanitization through encoding practices like HTML escape which would turn malicious JavaScript into harmless jargon:

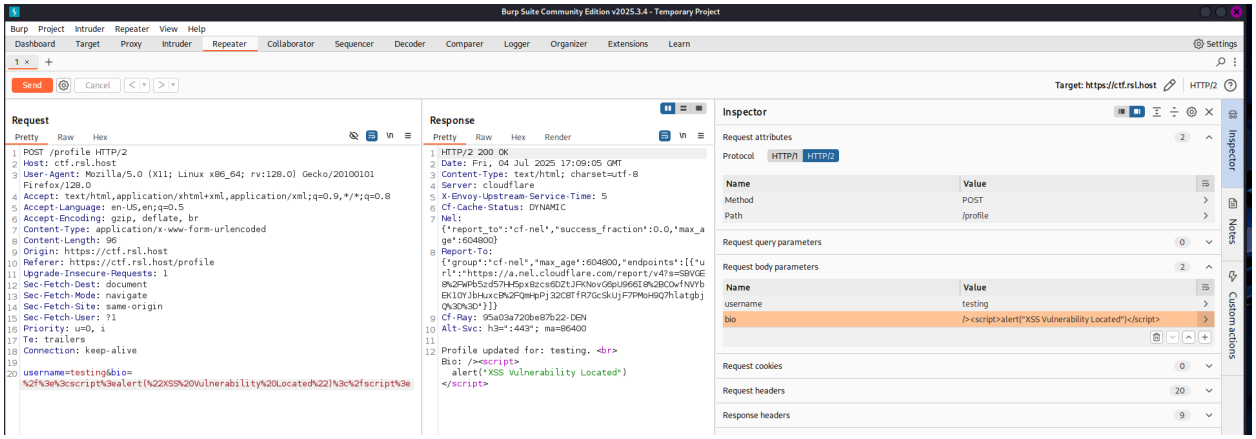
```
>>> import html
>>> unsanitized = '/><script>alert("XSS Vulnerability Located")</script>'
>>> sanitized = html.escape(unsanitized)
>>> print("Before sanitization: ", unsanitized)
Before sanitization:  /><script>alert("XSS Vulnerability Located")</script>
>>> print("After sanitization: ", sanitized)
After sanitization:  /&gt;&lt;script&gt;alert(&quot;XSS Vulnerability Located&quot;)&lt;/script&gt;
>>>
```

Testing Process:

1. Navigate to the site's /profile section that features two text entry fields, "username" and "bio," as well as a submit button to update the provided profile.
2. Enter the JavaScript payload, either within the webpage itself or, in our case, within BurpSuite's repeater tool[2]. Our payload here, as a proof-of-concept, is a simple JavaScript alert - in which an "alert" box is displayed with the provided message when executed. Our BurpSuite payload:

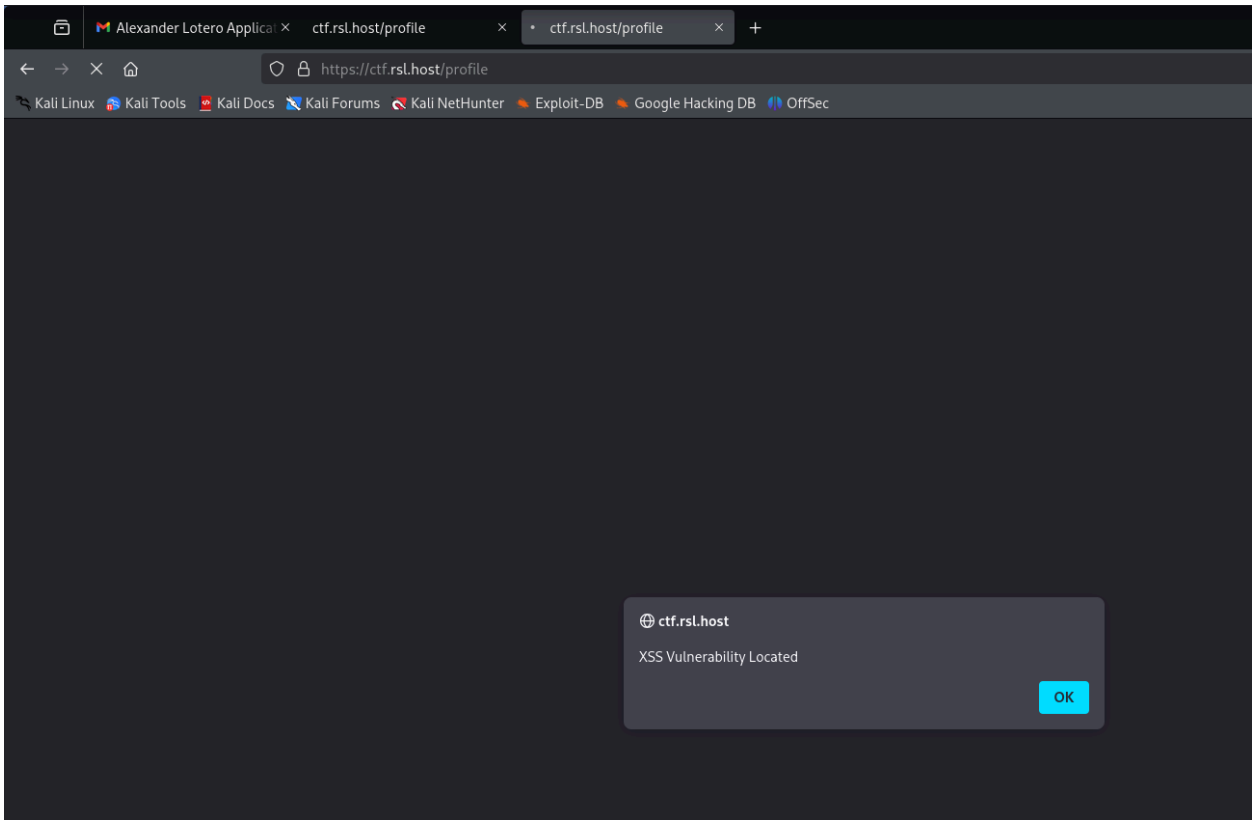
[1] Common Vulnerability Scoring System Version 3.0 Calculator - https://www.first.org/cvss/calculator/3-0#CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:N

[2] "Burp Repeater is a tool that enables you to modify and send an interesting HTTP or WebSocket message over and over." - https://portswigger.net/burp/documentation/desktop/tools/repeater



As seen in the screenshot, the website's response includes our JavaScript alert method within the "Bio" section.

3. Within a webpage, submit the payload and observe the behavior:



4. The screenshot shows our alert message has appeared, demonstrating arbitrary JavaScript execution.

Vulnerability 2

Name: Reflected Cross-Site Scripting (XSS) on /search

Risk Rating: Medium (6.4)

Exploitation Likelihood: High

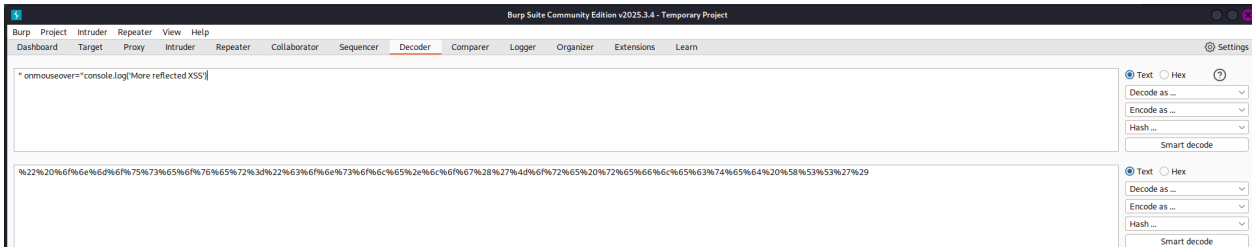
Potential Impact: Medium

Description: A continuation of vulnerability 1. In addition to the /profile page, user input is also not properly sanitized within the /search page. Once again, this vulnerability enables arbitrary JavaScript (JS) execution. Specifically, the backend code seems to run user input as part of executing JavaScript script code, but through " escaping and HTML encoding, we are able to execute additional JS.

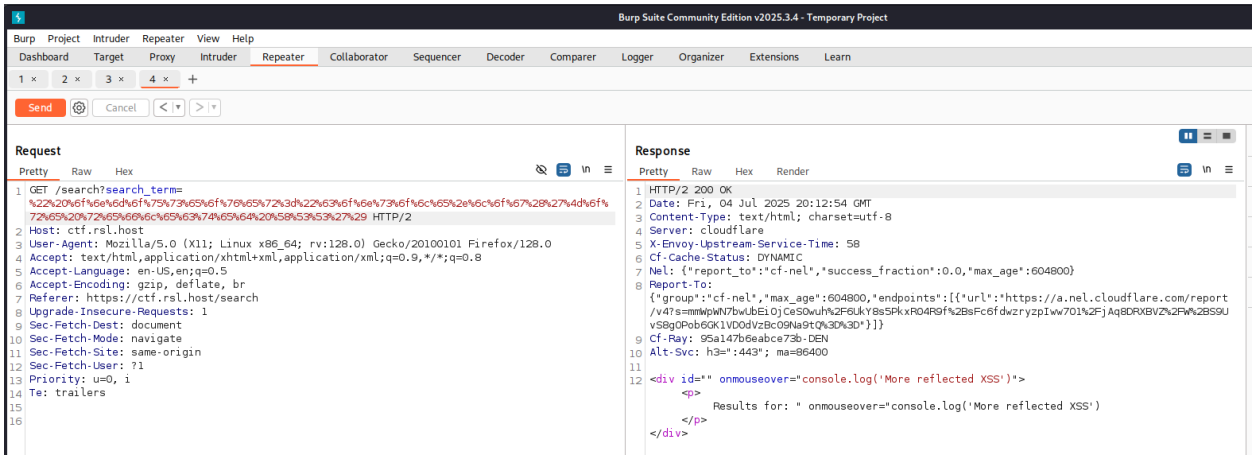
Remediation: Again, we would advise that the developers implement stricter sanitization methods of user input. This case in particular takes advantage of " as an allowed character and HTML encoding to obfuscate malicious JavaScript. We advise that the developers utilize stricter blacklisting or preferably whitelisting to define allowed characters including their encoded counterparts.

Testing Process:

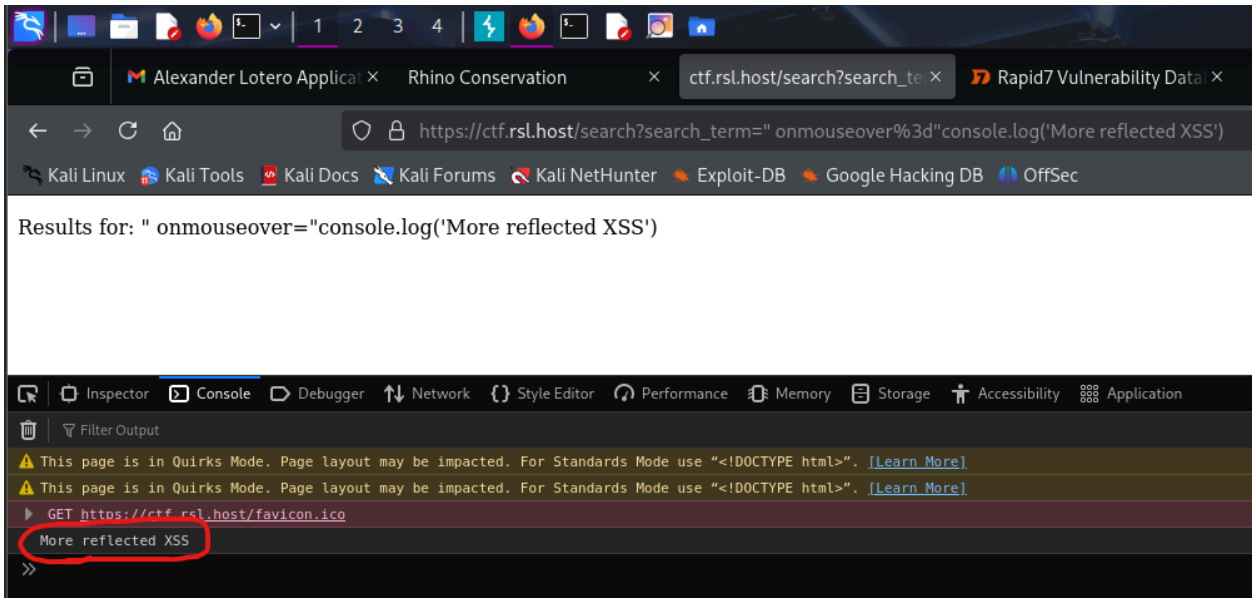
1. Navigate to the site's /search page in which users are able to enter and submit a search term.
2. As mentioned above, our payload must be HTML encoded in order to actually work as intended. Specifically, we deploy BurpSuite's built in decoder functionality to HTML encode our new payload:



   To avoid an exact duplicate of our previously reported vulnerability 1, we instead use JavaScript's "mouseover" event paired with the console log method as our proof-of-concept payload.
3. Once HTML encoded, we submit our payload. As before, this can be done through the browser itself, or through BurpSuite's repeater functionality:



4. We proceed to the webpage with our submitted payload in order to observe its effects. Specifically, this payload requires that our mouse “hovers over” the printed text before it actually executes the command and prints our message to the console log:



5. We see that our message is printed to the console log, thus proving another case of a reflected XSS vulnerability.

Vulnerability 3

Name: Directory Traversal on Squid Data Page (/data)

Risk Rating: High (8.6)

Exploitation Likelihood: High

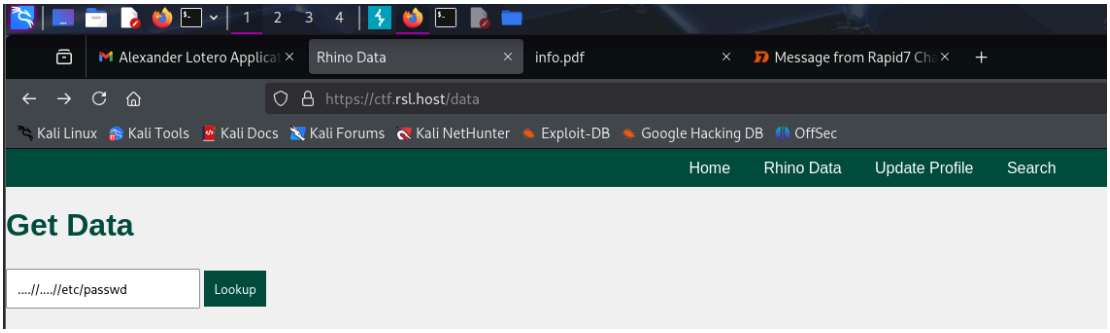
Potential Impact: High

Description: Once again, the application is failing to do proper sanitization checks on user-supplied input. In this case, we abuse the “Get Data” lookup functionality within the /data endpoint. The page includes a text entry field that is submitted via a clickable “lookup” button. Some trial-and-error with different kinds of inputs (SQLi, XSS, etc.) ultimately led us to directory traversal. Specifically, our submitted payload is fully utilized to traverse the server’s file system. Doing so allows us to view potentially sensitive files living in the server’s file system. The payload “....//….//etc/passwd” fully bypasses the existing regex sanitizer to view the /etc/passwd file.

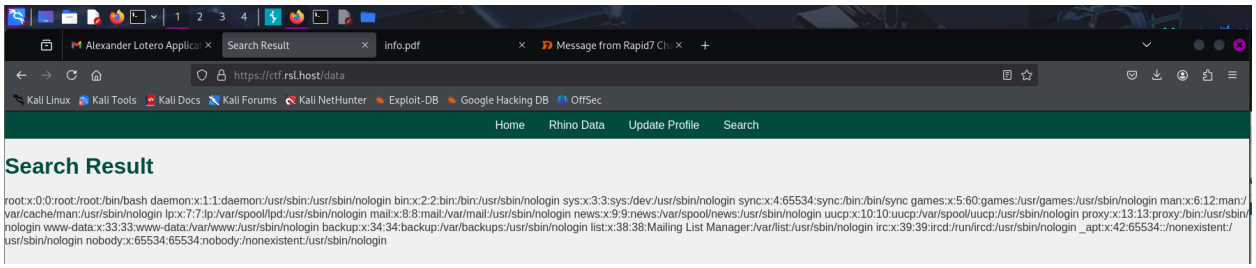
Remediation: We recommend that the developers deploy stronger input validation than the naive regex replacement that currently exists. Specifically, the current validator replaces any occurrences of the exact pattern “../”, which on face value would make directory traversal difficult. However, accounting for that specific pattern’s removal makes bypassing this protection trivial: “..../...//etc/passwd” becomes “../../etc/passwd”. We recommend a strict allowlist to define expected files and paths.

Testing Process:

1. We begin by navigating to the “Get Squid Data” page that is linked within the site’s homepage.
2. Once there, like before, we can submit our payload via the text entry field visible on the webpage, or through BurpSuite’s repeater tab. We demonstrate the webpage case:



3. As mentioned above, our proof-of-concept payload “..../////etc/passwd” is used to bypass the naive regex sanitizer and be run as a command along the lines of “cat /data/../../etc/passwd” or “open("data/" + “../../etc/passwd”)” in python:



4. The successful output of the /etc/passwd file’s contents including all of the machine’s user accounts, demonstrates unrestricted directory traversal and the information disclosure that it enables.

Vulnerability 4

Name: Server Side Request Forgery (SSRF)/ Local File Read (LFI) via HTTP Leading to Internal Admin Interface Disclosure

Risk Rating: High (7.3)

Exploitation Likelihood: High

Potential Impact: High

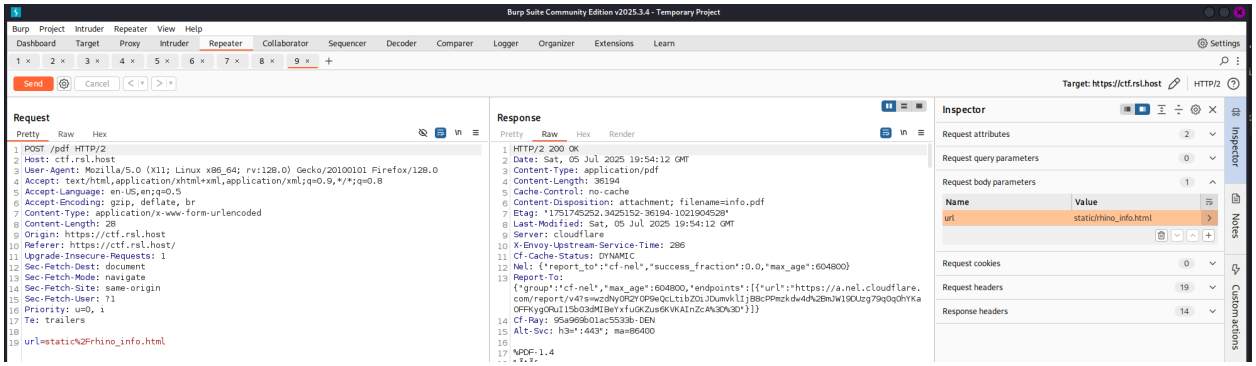
Description: The website's homepage features a "Download Info" link. Inspecting the packet transfer with BurpSuite's proxy functionality reveals that clicking the link results in a call to the /pdf endpoint. Further, that /pdf endpoint accepts a url parameter intended to generate PDFs from a provided resource. Additionally, a directory buster run against the server reveals a /admin endpoint, which when we attempt to navigate to, we are met with an HTTP 403 "Forbidden" code. We utilize the PDF generator to bypass this "forbidden" restriction. By supplying specific inputs, it was possible to make the server-side PDF generation tool (wkhtmltopdf[3]) issue HTTP requests to internal services that we as external users were not meant to access. This feature ultimately enabled SSRF. Specifically, we are able to download that /admin page as a PDF to reveal what is ultimately an admin command console.

Remediation: We once again recommend that the developers introduce stricter limits on user supplied input. A whitelist should be created for the url parameter to ensure that only expected, safe paths and domains are accepted. Further, the localhost, 127.0.0.1, and internal IPs in general should be blocked to prevent requests to internal services.

Testing Process:

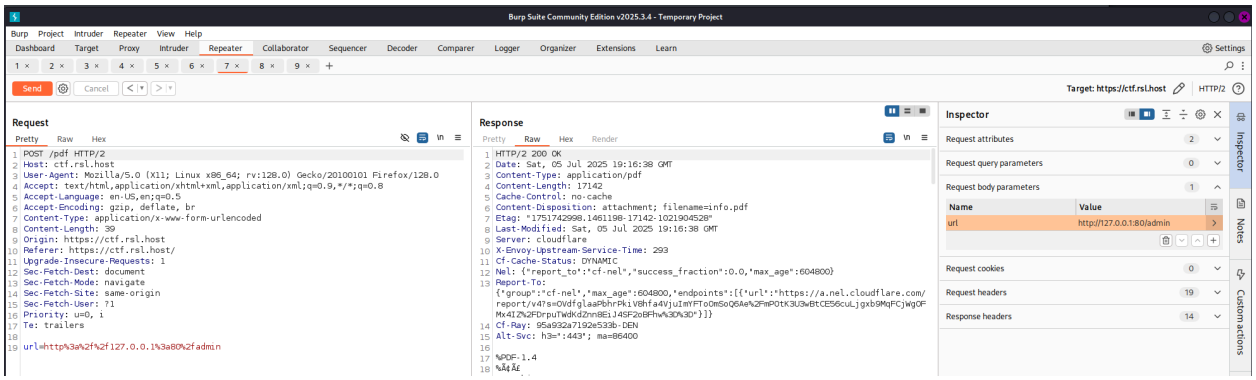
1. Navigate to the site's home page that includes the "Download Info" link.
2. We intercept the POST request sent to the server when the "Download Info" link is clicked by utilizing the BurpSuite proxy. We send that legitimate request to the BurpSuite repeater to incorporate our payload:

[3] "A command line tool to render HTML into PDF" - https://wkhtmltopdf.org/



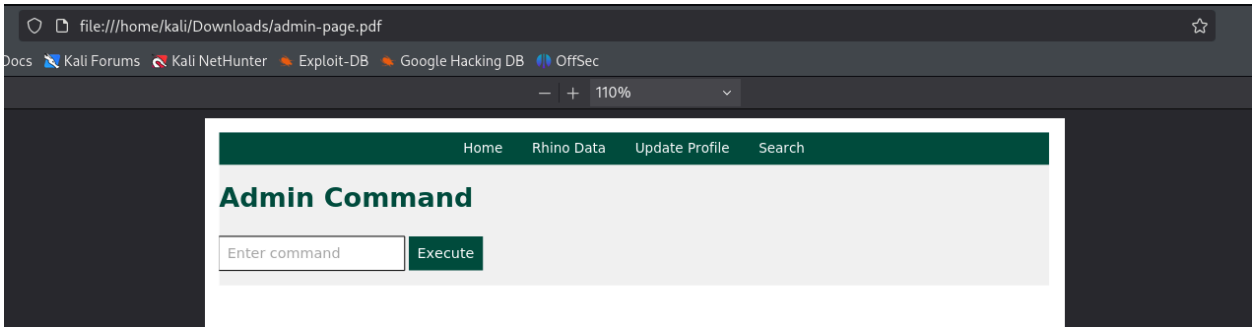
The url parameter here is where we will ultimately include our payload.

3. As described above, we point the wkhtmltopdf PDF generator back at the server's local resources, specifically the /admin page. To do so, we change the url parameter to "http://127.0.0.1:80/admin":



This instructs the generator to grab the resource found at 127.0.0.1/admin (the server itself) and provide it to us as a download.

4. Finally, we inspect the downloaded file that we fittingly renamed to "admin-page.pdf":



5. The /admin page is revealed to be a command console presumed to execute with admin privileges. At this stage, we can only see the /admin page, but not yet interact with it. More importantly, this same attack is likely to work with any

resource available on the server’s file system. This attack represents a full disclosure of the contents of the server’s file system.

Vulnerability 5

Name: Full Remote Code Execution (RCE) with Root Privileges via Server Side Request Forgery (SSRF)

Risk Rating: Critical (9.8)

Exploitation Likelihood: High

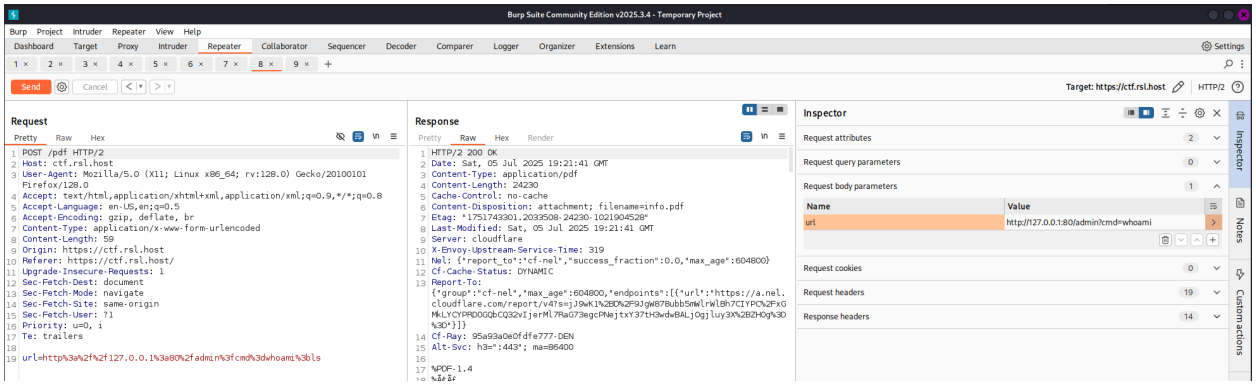
Potential Impact: Critical

Description: We take the /admin page discovered in vulnerability 4, and actually run commands through the admin command console with full root privileges. Specifically, we can see a text input field that resembles that of the /search page discussed earlier. As such, we can assume that the packet sent via the /admin page to execute a command is similar to a packet sent to the /search page to perform a search, but instead of a “search_term” parameter, it will feature a “command” parameter. Some trial-and-error of common names like “command,” “exec”, and “run” eventually reveals the parameter “cmd.” Thus, using that “cmd” parameter added onto our payload from the previous vulnerability, we can achieve full remote code execution as the root user.

Remediation: In addition to the remediation steps listed in vulnerability 4, we advise that the developers also reduce the privileges with which the wkhtmltopdf generator is run as part of a least privilege policy. Finally, avoid incorporating user input directly into system commands under any circumstances.

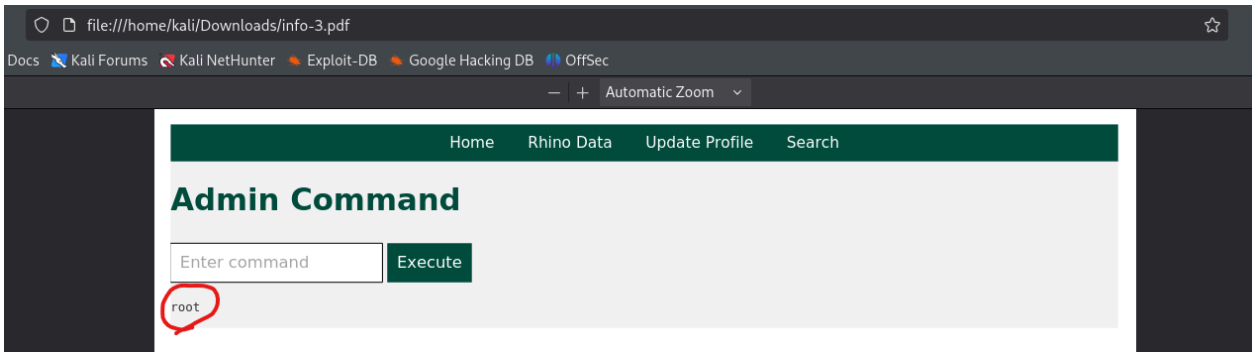
Testing Process:

1. We repeat steps 1 and 2 from vulnerability 4 to load a legitimate PDF download request into BurpSuite repeater.
2. Next, we include the system command (“cmd”) to execute as part of our malicious url parameter:



We selected “whoami[4]” as our proof-of-concept command because it makes it immediately clear that our command is being run as root.

3. Submitting this new payload once again provides us with a PDF download of the same /admin page. However, this time the output of the “whoami” command is printed to the screen:



This “whoami” output of “root” confirms that we have full remote code execution (RCE) with root privileges. The “ls” and “cat” commands were also tested to validate the broad command capabilities.

[4] “Display the current Username in Linux” - https://www.geeksforgeeks.org/linux-unix/whoami-command-linux-example/