|  |  |
| --- | --- |
| Logo  Description automatically generated | Challenge Writeup  V4ln Design  by Kouretas Panagiotis |

# Main Concept

# Discover the elegance of V4ln Interior Design in this captivating challenge. Dive into the world of luxury and creativity, as you search for clues seamlessly woven into the fabric of this intricately designed website.

# Exploitation

First of all, we deploy the challenge from CTFLib.

Then we are redirected at <http://localhost:8081/> which is the home page of the challenge as seen in Figure 4.1. This appears to be the webpage for the "Vuln" interior design firm.

A close-up of a staircase

Description automatically generated

Figure 4 - Challenge's Home Page - Interior Design website.

At the first glance, the only thing that seems possible to apply some exploits on this website is the “Get in Touch” form at the bottom of the page.

A screenshot of a computer

Description automatically generated

Figure 4 - "Get In Touch" form.

Challenge Description:

"Discover the elegance of V4ln Interior Design in this captivating challenge. Dive into the world of luxury and creativity, as you search for clues seamlessly woven into the fabric of this intricately designed website."

Since we can’t find any useful hints in the challenge description, we hop on inspecting the source code of the website (pressing “CTRL + U” on the page) as depicted in Figure 4.3.

There I realized that the contact form is not functional. We can cross validate it from two significant points. The form's action attribute points to "index.html", an HTML file, which cannot process or store form data in a database. HTML files are static and do not contain server-side processing capabilities.

A black background with text on it

Description automatically generated

Figure 4 - Form's source code.

Likewise, there is no visible AJAX code provided that would handle the form submission asynchronously to send the data to a server or database, indicating the form's submission process does not interact with any backend system.

Every time I submitted the form, I was just redirected to the homepage without any message displayed on my screen. Now it makes sense why this was happening.

While checking the network traffic from the browser developer tools (by pressing the “F12” button on the browser), I distinguished a clue that could be proved crucial. The server used for this website was “Apache/2.4.49” as depicted in Figure 4.4.

A screenshot of a computer

Description automatically generated

Figure 4 - Browser Developer tools / Server Used

So, let's do some googling to see if there are any possible vulnerabilities for this specific version of Apache.

Just typed “Apache/2.4.49” and as depicted in the figure, we understand that something goes wrong with this version.

A screenshot of a computer

Description automatically generated

Figure 4 – Searching for Apache 2.4.49 vulnerabilities.

The first article I found from “ExlploitDB”, indicates that this version of Apache server is vulnerable to Path Traversal and Remote Code Execution (RCE) constituting “CVE-2021-41773”. (Exploit Database is a well-known website including Exploits, Shellcode, 0days etc).

A screenshot of a computer

Description automatically generated

Figure 4 - ExploitDB Apache 2.4.49 vulnerabilities article.

Upon further investigation, I found some very helpful articles and videos ([article 1](https://www.picussecurity.com/resource/blog/simulate-apache-cve-2021-41773-exploits-vulnerability), [article 2](https://blog.qualys.com/vulnerabilities-threat-research/2021/10/27/apache-http-server-path-traversal-remote-code-execution-cve-2021-41773-cve-2021-42013), [video](https://www.youtube.com/watch?v=m6pW_8LFzNk)) that analyse extensively this CVE’s vulnerability, explain everything that someone has to know about it and demonstrate how to exploit it.

In Figure 4.7, there is a snippet of the [article 1](https://www.picussecurity.com/resource/blog/simulate-apache-cve-2021-41773-exploits-vulnerability) that explains briefly how this vulnerability arised.

A close-up of a computer screen

Description automatically generated

Figure 4 - Snippet that explains how the vulnerability caused.

Now let's apply this exploit in our situation. To do that I will use “Burp Suite Proxy” tool. Utilizing this tool, I will capture the website’s requests using “Burp’s Browser”.

A screenshot of a computer

Description automatically generated

Figure 4 - Capturing website's traffic using BurpSuite Proxy.

Then I will modify one of them with “Burp’s Repeater” and resend the new malformed.

The payload I used on this request is the following and it consists of two different parts:

First part: “/cgi-bin/.%2e/.%2e/.%2e/.%2e/.%2e/bin/sh”

Second part: “echo;id”

So, lets break it down explaining what happens while executing this payload.

In the first part this URI (Uniform Resource Identifier) attempts to exploit the path traversal vulnerability.

/cgi-bin/: This is the directory on the server where CGI (Common Gateway Interface) scripts are typically stored and executed.

.%2e: This is URL encoding for . (dot), a directory level in Unix-like file systems. In this context, ”.%2e” represents a "dot" character, attempting to bypass security mechanisms by encoding.

The sequence /.%2e/.%2e/.%2e/.%2e/.%2e: Tries to traverse up five directory levels from the /cgi-bin/ directory. The goal is to reach the root directory of the filesystem.

/bin/sh: After reaching the filesystem's root, the request attempts to access the shell (sh) located typically in the /bin directory. This would theoretically allow the execution of shell commands.

In the second part, the payload contains the commands we want to execute on the server. The “echo;id” is a Unix command that prints the real and effective user and group IDs. Now that’s executed, it demonstrates that arbitrary commands can be run on the server, showing the user context under which the web server's process is running as seen in Figure 4.9.

A screenshot of a computer

Description automatically generated

Figure 4 - Payload execution using Burp’s Repeater.

So, since we can execute arbitrary commands on the server, I did some testing trying to determine which commands could be proved useful.

As depicted in Figure 4.10, the command I executed is “echo;cd /var/www/html;ls”.  
This command lists the contents of the “/var/www/html” directory. Before ending up in this directory I was going into each directory and listing its contents separately so I see every possible files I could access.

A screenshot of a computer

Description automatically generated

Figure 4 - Listing the contents of "/var/www/html" directory.

In the “/var/www/html” directory I noticed that there is a “flag.txt” file that’s most possible to include the hidden flag.

So let’s execute “echo;cd /var/www/html;cat flag.txt” command to view the contents of “flag.txt”.

A screenshot of a computer

Description automatically generated

Figure 4 - Viewing the contents of flag.txt file / Hidden flag successfully retrieved.

NAILED IT, we found the hidden FLAG after successfully exploiting the path traversal and the remote command execution vulnerabilities.

CTFLIB{D3f!n!t3ly\_@\_v4ln3r@bl3\_d3s!gn}