File Attacks

Objectives

- Examine how not properly sanitized server-side rendered content introduces the opportunity to arbitrarily execute code.

- Examine how file uploads and file inclusion vulnerabilities introduce the opportunity to arbitrarily execute and read server-side content.

References

- https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload
- https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/07-Input_Validation_Testing/11.1-Testing_for_Local_File_Inclusion

File Upload Vulnerabilities

- Typically, we allow users to upload user-supplied content like images.

- But what happens if we don’t properly filter that content and a user is allowed to upload and evaluate malicious php code?

File Upload Vulnerabilities

Uploaded files represent a significant risk to applications.... the attack only needs to find a way to get the code executed. Using a file upload helps the attacker accomplish the first step.

Text copied from: https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload

PHP Scripting

- PHP is a hypertext processor for web-based code
- PHP is mostly run server-side (as opposed to client JavaScript)
- For example, we had ChatGPT generate a php script to dynamically generate a webpage with the current date.

```
<!DOCTYPE html>
<html>
<head>
    <title>PHP Example</title>
</head>
<body>
    <h1>Welcome to my website</h1>
    <?php
        echo "Today is " . date("Y-m-d H:i:s");
    ?>
</body>
</html>
```

User Controlled Code Issues

- However, what happens if users are allowed to supply php code and then it is evaluated on the server?
- Well then, its probably good that we don't often allow users to upload php code.

```
<!DOCTYPE html>
<html>
<head>
    <title>Execute whoami Command</title>
</head>
<body>
    <h1>Current User</h1>
    <?php

        $output = shell_exec('whoami');
        echo "<pre>$output</pre>";
    ?>
</body>
</html>
```

Vulnerable Application

Our vulnerable application is pretty bad. Not only does it allow malicious php scripts to be uploaded. But it also puts in a check to go ahead and process them.

```
if file:
        filepath = os.path.join(UPLOAD_FOLDER, file.filename)
        file.save(filepath)

        if filepath.endswith('.php'):
            try:
                result = subprocess.run(['php', filepath], capture_output=True, text=True)
                return render_template('fileupload.html',message=f"Server-side code rendered: {result.stdout}")

            except Exception as e:
                return render_template('fileupload.html',message=f"An error occurred: {e}")
        else:
            return render_template('fileupload.html',message=f"File: {filepath} succesfully uploaded.")
```

File Inclusion Vulnerabilities

The File Inclusion vulnerability allows an attacker to include a file, usually exploiting a “dynamic file inclusion” mechanisms implemented in the target application. The vulnerability occurs due to the use of user-supplied input without proper validation.

https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/07-Input_Validation_Testing/11.1-Testing_for_Local_File_Inclusion

File Inclusion Vulnerability

Our vulnerable application is pretty bad. Can you identify the issue and how we could abuse it?

```
page = request.args.get('page')
file_path = f"templates/{page}"
with open(file_path, 'r') as f:
    template_content = f.read()
    return render_template_string(template_content,message=message)
```

File Inclusion Vulnerability

It works fine if page = “lfi.html”. This will render the local file inclusion template file.

```
page = request.args.get('page')
file_path = f"templates/lfi.html "
with open(file_path, 'r') as f:
    template_content = f.read()
    return render_template_string(template_content,message=message)
```

File Inclusion Vulnerability

But what happens when you render .../../flag.txt?

What directory is /app/templates/../../ ?

```
page = request.args.get('page')
file_path = f"templates/../../flag.txt"
with open(file_path, 'r') as f:
    template_content = f.read()
    return render_template_string(template_content,message=message)
```