ATTACK DEFENSE

by PentesterAcademy

Name	RIPS: Statically Scanning PHP Code
URL	https://www.attackdefense.com/challengedetails?cid=2158
Type	DevSecOps Basics: Static Application Security Testing

Important Note: This document illustrates all the important steps required to complete this lab. This is by no means a comprehensive step-by-step solution for this exercise. This is only provided as a reference to various commands needed to complete this exercise and for your further research on this topic. Also, note that the IP addresses and domain names might be different in your lab.

Challenge Description

The RIPS scanner will audit the PHP code statically for vulnerabilities.

A RIPS scanner instance is provided to the user and the source code for two sample applications is provided in the root directory.

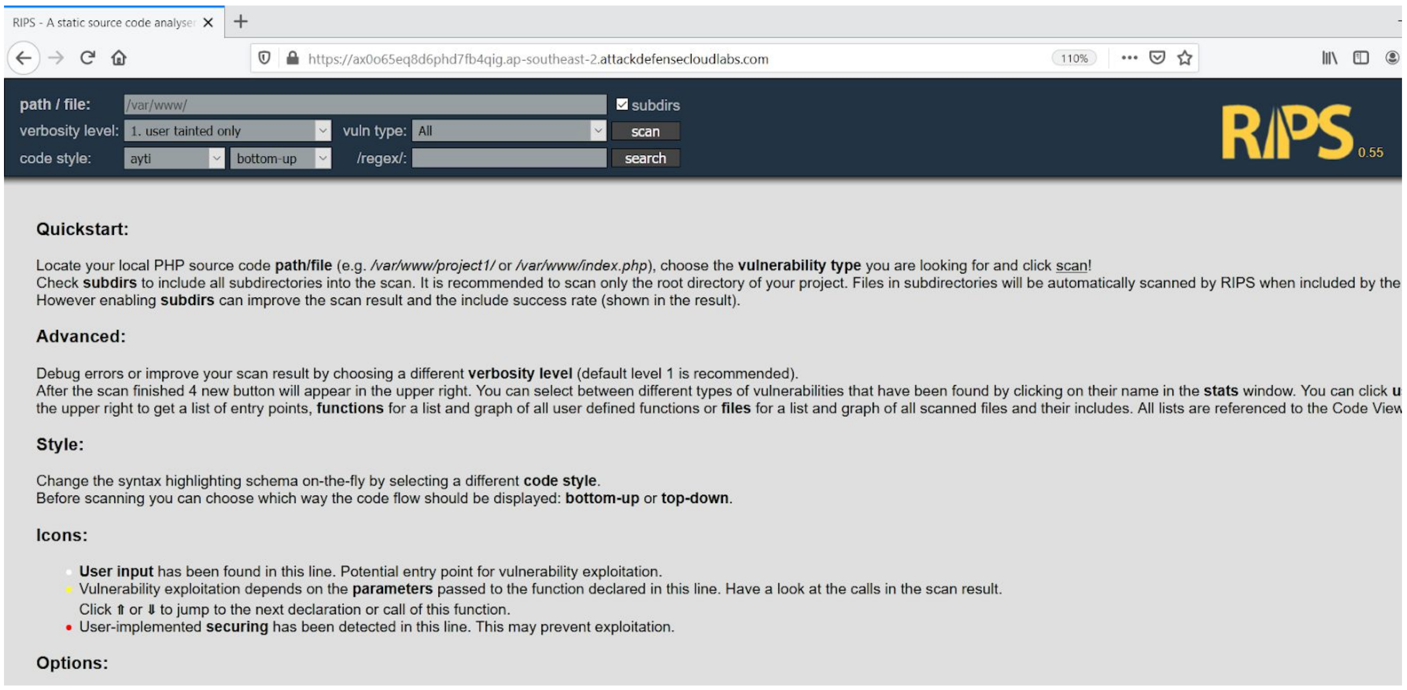
Objective: Use the rips scanner utility to find vulnerabilities in the application!

Instructions:

- The source code of applications is provided at /github-repos
- The names of the projects present in the github-repos are pagekit and mejiro

Solution

Step 1: Open the RIPS scanner website.



Quickstart:

Locate your local PHP source code **path/file** (e.g. */var/www/project1/* or */var/www/index.php*), choose the **vulnerability type** you are looking for and click scan!
Check **subdirs** to include all subdirectories into the scan. It is recommended to scan only the root directory of your project. Files in subdirectories will be automatically scanned by RIPS when included by the
However enabling **subdirs** can improve the scan result and the include success rate (shown in the result).

Advanced:

Debug errors or improve your scan result by choosing a different **verbosity level** (default level 1 is recommended).
After the scan finished 4 new button will appear in the upper right. You can select between different types of vulnerabilities that have been found by clicking on their name in the **stats** window. You can click **u**
the upper right to get a list of entry points, **functions** for a list and graph of all user defined functions or **files** for a list and graph of all scanned files and their includes. All lists are referenced to the Code View

Style:

Change the syntax highlighting schema on-the-fly by selecting a different **code style**.
Before scanning you can choose which way the code flow should be displayed: **bottom-up** or **top-down**.

Icons:

- **User input** has been found in this line. Potential entry point for vulnerability exploitation.
- Vulnerability exploitation depends on the **parameters** passed to the function declared in this line. Have a look at the calls in the scan result.
  Click ⇑ or ⇓ to jump to the next declaration or call of this function.
- User-implemented **securing** has been detected in this line. This may prevent exploitation.

Options:

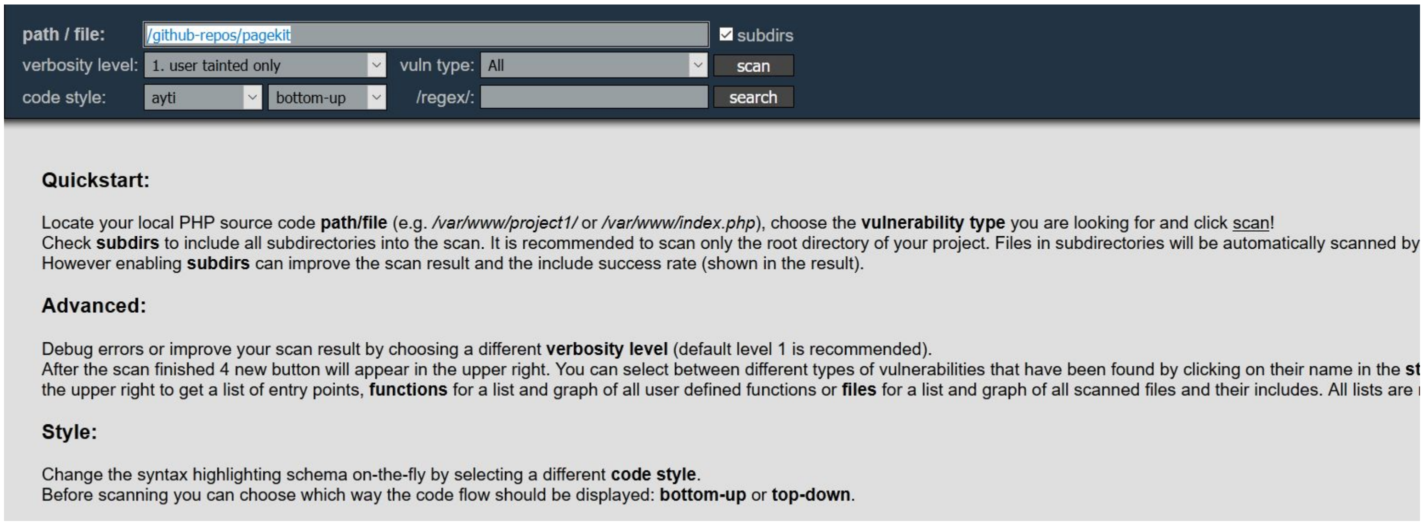
The path/file will accept the path to the project directory and the names of projects have been provided in the challenge description. The base directory of the projects is /github-repos/

We will take one example at a time and run the tool on that.

Example 1: pagekit

Step 1: Pass the project directory of pagekit in the path/file section.

PATH: /github-repos/pagekit


path / file: /github-repos/pagekit ☑ subdirs
verbosity level: 1. user tainted only vuln type: All scan
code style: ayti bottom-up /regex/: search

Quickstart:

Locate your local PHP source code **path/file** (e.g. */var/www/project1/* or */var/www/index.php*), choose the **vulnerability type** you are looking for and click scan!
Check **subdirs** to include all subdirectories into the scan. It is recommended to scan only the root directory of your project. Files in subdirectories will be automatically scanned by
However enabling **subdirs** can improve the scan result and the include success rate (shown in the result).

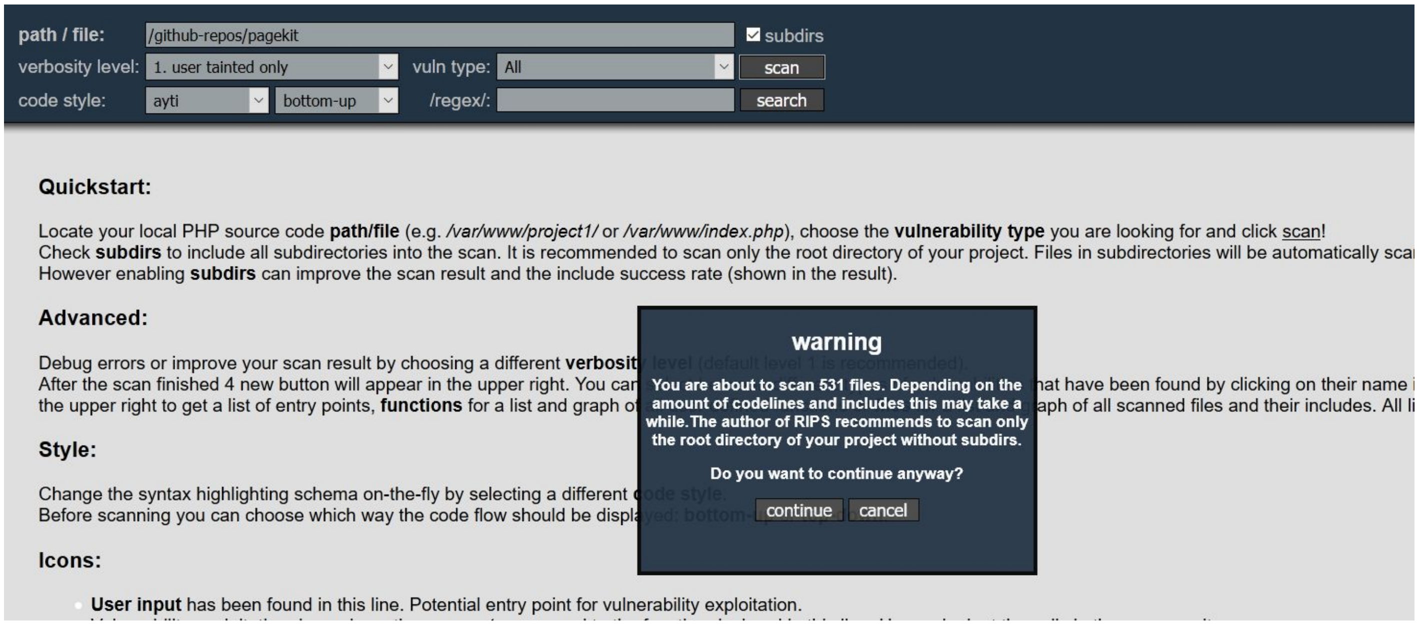
Advanced:

Debug errors or improve your scan result by choosing a different **verbosity level** (default level 1 is recommended).
After the scan finished 4 new button will appear in the upper right. You can select between different types of vulnerabilities that have been found by clicking on their name in the **st**
the upper right to get a list of entry points, **functions** for a list and graph of all user defined functions or **files** for a list and graph of all scanned files and their includes. All lists are

Style:

Change the syntax highlighting schema on-the-fly by selecting a different **code style**.
Before scanning you can choose which way the code flow should be displayed: **bottom-up** or **top-down**.

Step 2: Press the scan button.


path / file: /github-repos/pagekit ☑ subdirs
verbosity level: 1. user tainted only vuln type: All scan
code style: ayti bottom-up /regex/: search

warning

You are about to scan 531 files. Depending on the amount of codelines and includes this may take a while.The author of RIPS recommends to scan only the root directory of your project without subdirs.

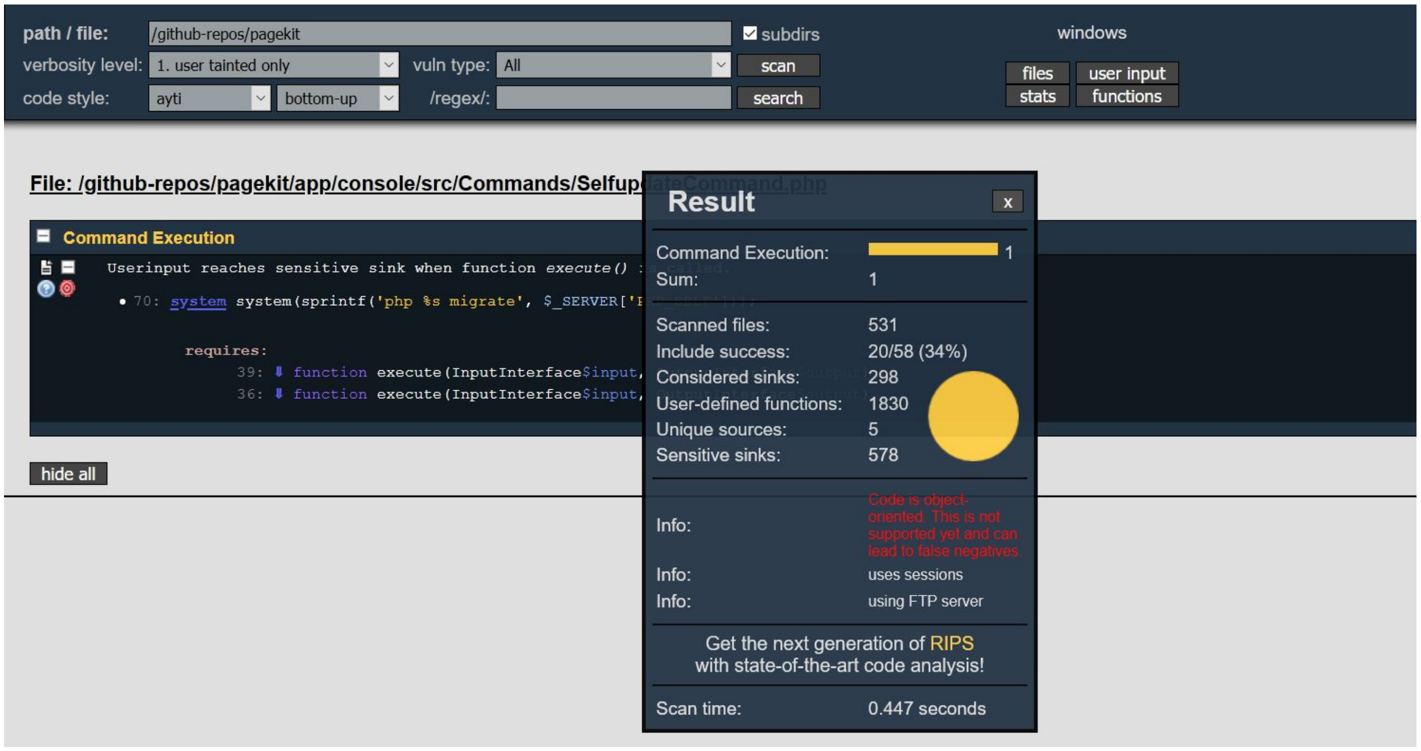
Do you want to continue anyway?

continue cancel

Icons:

User input has been found in this line. Potential entry point for vulnerability exploitation.

Click on the ‘continue’ button to scan the project source code.


path / file: /github-repos/pagekit ☑ subdirs
verbosity level: 1. user tainted only | vuln type: All | scan
code style: ayti | bottom-up | /regex/: | search

File: /github-repos/pagekit/app/console/src/Commands/SelfupdateCommand.php

Command Execution

```
Userinput reaches sensitive sink when function execute() is called.

• 70: system system(sprintf('php %s migrate', $_SERVER['PHP_SELF']));

    requires:
        39: function execute(InputInterface$input, OutputInterface$output)
        36: function execute(InputInterface$input, OutputInterface$output)
```

Result

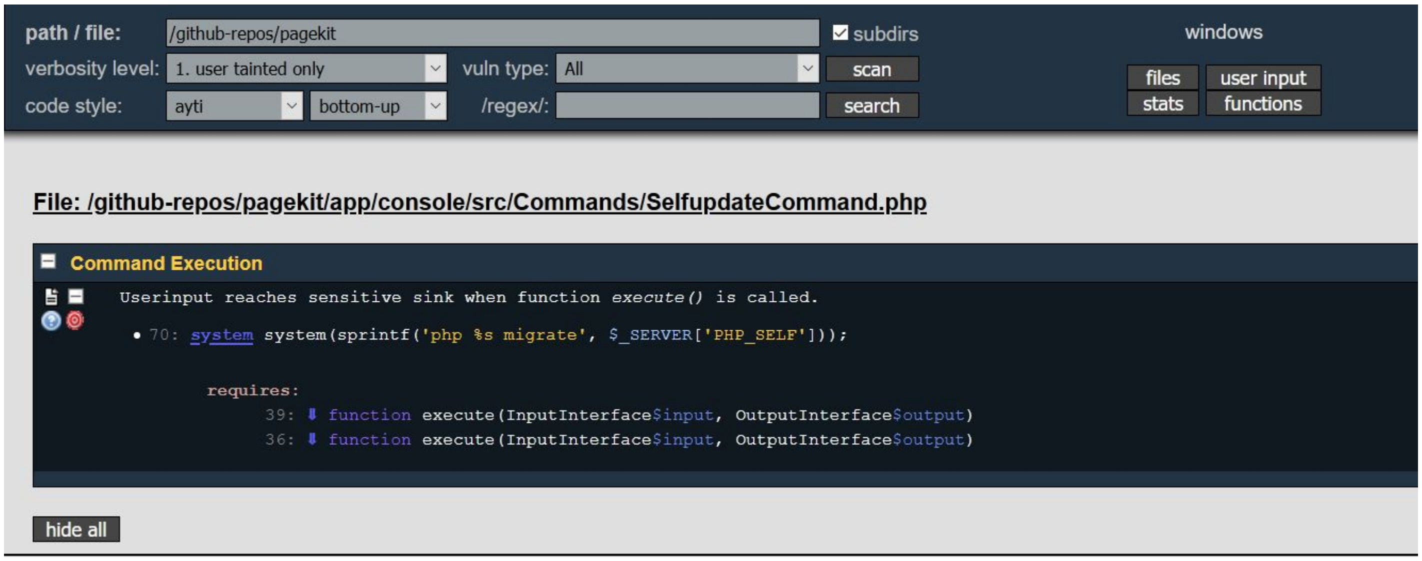
Command Execution:	1
Sum:	1
Scanned files:	531
Include success:	20/58 (34%)
Considered sinks:	298
User-defined functions:	1830
Unique sources:	5
Sensitive sinks:	578
Info:	Code is object-oriented. This is not supported yet and can lead to false negatives.
Info:	uses sessions
Info:	using FTP server

Get the next generation of RIPS with state-of-the-art code analysis!

Scan time: 0.447 seconds

The scan has been completed and the RIPS scanner found one vulnerability in the project.

Step 3: Close the dialog box and check the source code displayed by RIPS scanner.


path / file: /github-repos/pagekit ☑ subdirs
verbosity level: 1. user tainted only | vuln type: All | scan
code style: ayti | bottom-up | /regex/: | search

File: /github-repos/pagekit/app/console/src/Commands/SelfupdateCommand.php

Command Execution

```
Userinput reaches sensitive sink when function execute() is called.

• 70: system system(sprintf('php %s migrate', $_SERVER['PHP_SELF']));

    requires:
        39: function execute(InputInterface$input, OutputInterface$output)
        36: function execute(InputInterface$input, OutputInterface$output)
```

The function “execute” is getting the user-input directly without any sanitization which could lead to command injection.

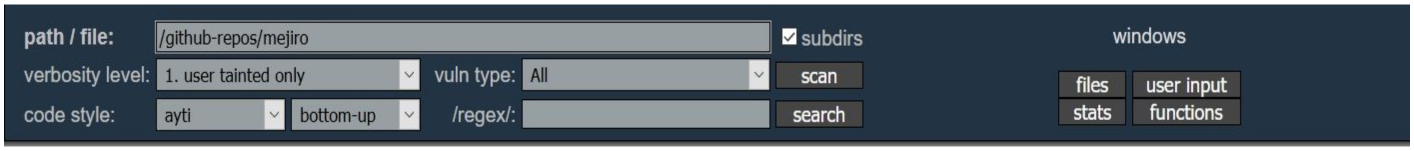
Issues Detected

- Command Injection

Example 2: mejiro

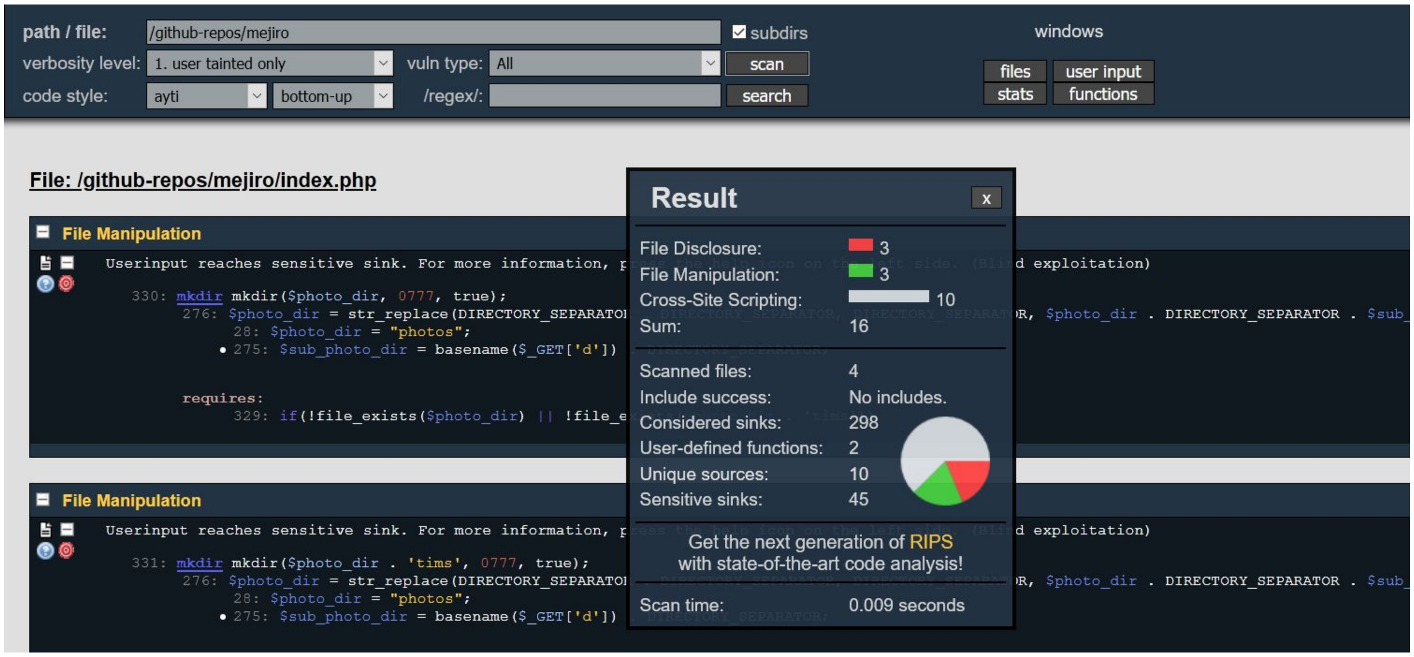
Step 1: Change to the project directory in the path/file section.

PATH: /github-repos/mejiro



```
path / file:       /github-repos/mejiro                       ☑ subdirs
verbosity level:   1. user tainted only     vuln type: All
code style:        ayti        bottom-up    /regex/:
```

Step 2: Press the scan button.



```
path / file:       /github-repos/mejiro                       ☑ subdirs
verbosity level:   1. user tainted only     vuln type: All
code style:        ayti        bottom-up    /regex/:
```

File: /github-repos/mejiro/index.php

File Manipulation

```
Userinput reaches sensitive sink. For more information, p                                          d exploitation)

    330: mkdir mkdir($photo_dir, 0777, true);
        276: $photo_dir = str_replace(DIRECTORY_SEPARATOI                              R, $photo_dir . DIRECTORY_SEPARATOR . $sub_
            28: $photo_dir = "photos";
          • 275: $sub_photo_dir = basename($_GET['d'])

        requires:
            329: if(!file_exists($photo_dir) || !file_e
```

File Manipulation

```
Userinput reaches sensitive sink. For more information, p                                          d exploitation)

    331: mkdir mkdir($photo_dir . 'tims', 0777, true);
        276: $photo_dir = str_replace(DIRECTORY_SEPARATOI                              R, $photo_dir . DIRECTORY_SEPARATOR . $sub_
            28: $photo_dir = "photos";
          • 275: $sub_photo_dir = basename($_GET['d'])
```

Result

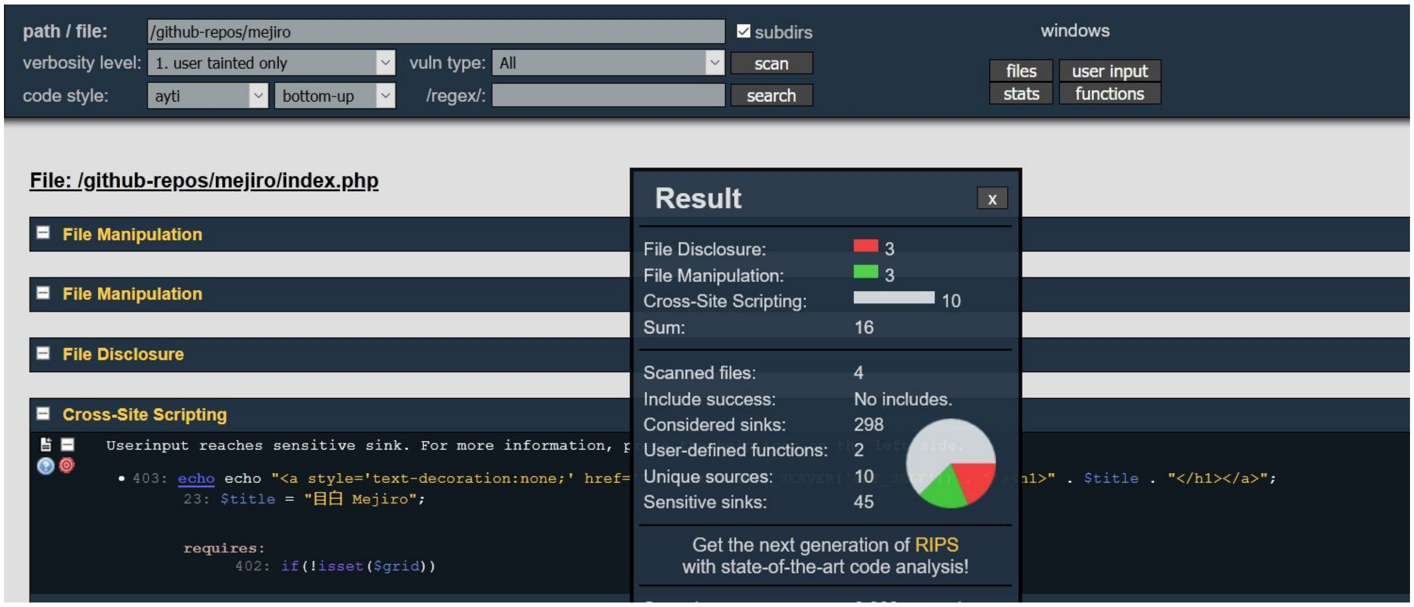
File Disclosure:	🟥 3
File Manipulation:	🟩 3
Cross-Site Scripting:	▭ 10
Sum:	16
Scanned files:	4
Include success:	No includes.
Considered sinks:	298
User-defined functions:	2
Unique sources:	10
Sensitive sinks:	45

Get the next generation of RIPS
with state-of-the-art code analysis!

Scan time:	0.009 seconds

The RIPS scanner revealed multiple vulnerabilities in the project's source code.

Step 3: Click on any vulnerability to check the source code. In this case, Cross-site scripting is selected



path / file: /github-repos/mejiro
verbosity level: 1. user tainted only
vuln type: All
code style: ayti, bottom-up

File: /github-repos/mejiro/index.php

File Manipulation

File Manipulation

File Disclosure

Cross-Site Scripting

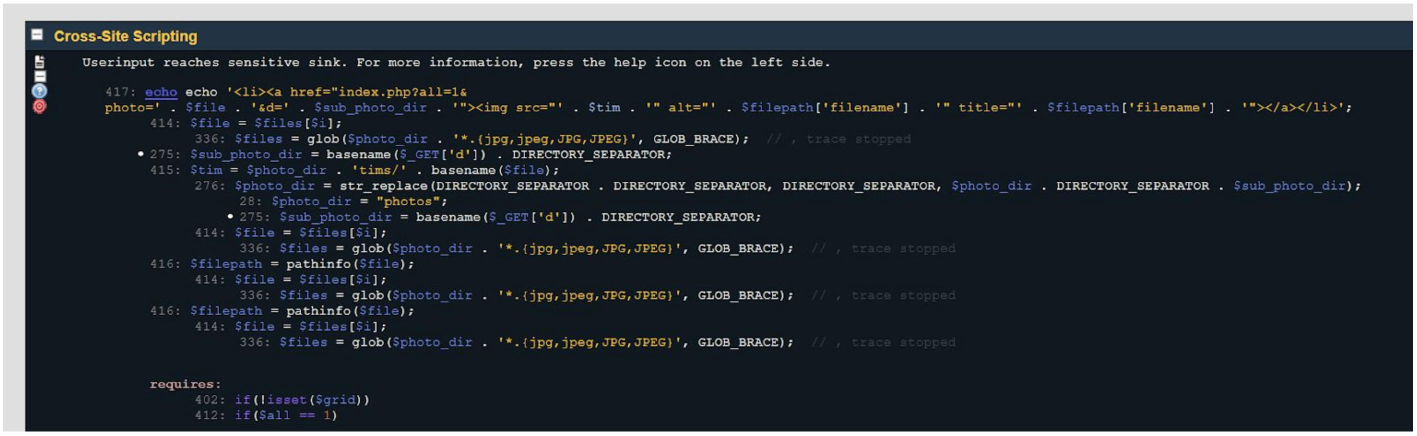
```
Userinput reaches sensitive sink. For more information, p

• 403: echo echo "<a style='text-decoration:none;' href=                                    h1>" . $title . "</h1></a>";
     23: $title = "目白 Mejiro";


     requires:
          402: if(!isset($grid))
```

Result

File Disclosure:	🟥 3
File Manipulation:	🟩 3
Cross-Site Scripting:	10
Sum:	16
Scanned files:	4
Include success:	No includes.
Considered sinks:	298
User-defined functions:	2
Unique sources:	10
Sensitive sinks:	45

Get the next generation of RIPS with state-of-the-art code analysis!

Close the result dialogue box to check the vulnerable source code.



Cross-Site Scripting

```
Userinput reaches sensitive sink. For more information, press the help icon on the left side.

  417: echo echo '<li><a href="index.php?all=1&
  photo=' . $file . '&d=' . $sub_photo_dir . '"><img src="' . $tim . '" alt="' . $filepath['filename'] . '" title="' . $filepath['filename'] . '"></a></li>';
        414: $file = $files[$i];
              336: $files = glob($photo_dir . '*.{jpg,jpeg,JPG,JPEG}', GLOB_BRACE);  // , trace stopped
      • 275: $sub_photo_dir = basename($_GET['d']) . DIRECTORY_SEPARATOR;
        415: $tim = $photo_dir . 'tims/' . basename($file);
              276: $photo_dir = str_replace(DIRECTORY_SEPARATOR . DIRECTORY_SEPARATOR, DIRECTORY_SEPARATOR, $photo_dir . DIRECTORY_SEPARATOR . $sub_photo_dir);
                    28: $photo_dir = "photos";
                  • 275: $sub_photo_dir = basename($_GET['d']) . DIRECTORY_SEPARATOR;
              414: $file = $files[$i];
                    336: $files = glob($photo_dir . '*.{jpg,jpeg,JPG,JPEG}', GLOB_BRACE);  // , trace stopped
        416: $filepath = pathinfo($file);
              414: $file = $files[$i];
                    336: $files = glob($photo_dir . '*.{jpg,jpeg,JPG,JPEG}', GLOB_BRACE);  // , trace stopped
        416: $filepath = pathinfo($file);
              414: $file = $files[$i];
                    336: $files = glob($photo_dir . '*.{jpg,jpeg,JPG,JPEG}', GLOB_BRACE);  // , trace stopped


        requires:
              402: if(!isset($grid))
              412: if($all == 1)
```

In this case, the filename is getting passed without being checked or sanitised, hence a malicious user can trigger XSS attack by uploading an image with the malicious file name.

Issues Detected:

- File Disclosure
- File Manipulation
- Cross-Site Scripting

Learnings

Perform Static Code Analysis using the RIPS tool.

References:

- Pagekit (https://github.com/pagekit/pagekit.git)
- Mejiro (https://github.com/dmpop/mejiro.git)