Chipin' In

Reported By:
Jawad Al-Fuheid

Reported To:
Academy of Learning

I have created this challenge for Hackathon 2023, and I want to clarify that while this challenge may not be particularly difficult, it does require a basic understanding of web frameworks in general. This challenge is based on CVE-2020-14343.

To get started, let's delve into the core concept of this challenge. First, take a look at the source code. You'll notice that the index page handles both GET and POST requests.

What's interestus here is that the server accepts your data from a POST request without applying any filtering and then stores it in a YAML file called subs.yml. It might not seem very challenging at first glance, but there's more to it.

If you attempt something like a cross-site scripting (XSS) or server-side template injection (SSTI) attack, you won't make any headway. It's not that straightforward. However, there's another, more clever way to exploit it.

In this challenge, we're using PyYAML version 3.13 to store user data on the server. And here's where the vulnerability comes into play. In PyYAML 3.13, we can leverage CVE-2020-14343. In essence, this vulnerability allows us to execute Python code within YAML files.

So, in simple terms, we'll craft a payload and send it to the server to retrieve the flag. The payload should look like this:

```
!!python/object/new:Warning
state:
  extend: !!python/name:exec
listitems: 'Any Python Code'
```

Let's break down what this YAML payload means:

- **!!python/object/new:Warning:** This YAML tag indicates that the following data should be interpreted as a Python object, specifically, an instance of the Python Warning class.

- **state**: This is a key within the object being created.

- **extend: !!python/name:exec:** This is an attempt to execute Python code. It invokes the exec function using the YAML tag !!python/name:exec, meaning it executes the Python code defined in the **listitems** field.

- **listitems: 'Any Python Code':** Here, you can insert any Python code you desire.

By creating a YAML file and adding this code to it, you can execute arbitrary Python code if done correctly.

Now, once we have the ability to execute code on the server, we can locate the flag or browse the files on the machine. This can be achieved by executing Linux commands and retrieving the results:

```
import os; os.system("ls -la > log.txt; mv log.txt /static/log.txt")
```

Let's break down this code:

- **os.system:** This function is used to execute commands on both Linux and Windows. In this case, it runs the Linux command:

- **ls -la > log.txt; mv log.txt /static/log.txt**
  ls: Lists all files in the current directory, including hidden files **(-la)**.

- **>:** Redirects the output of the previous command to a file named **log.txt**.

- **mv:** Moves the **log.txt** file to the **/static/** folder.

But why the static folder?

In Flask's layout, which you can learn more about here: Flask Layout, static files include CSS, JavaScript, images, and more. Therefore, it's always possible to access static files in the static folder using your browser to display images or retrieve any other necessary files.

Once you're certain that the flag resides in the current directory, you can simply move it to the static folder:

mv flag.txt /static/flag.txt

Afterward, you can access the flag by visiting http://host:port/static/flag.txt and submit it to complete the challenge.

You can also find the exploit script attached with this writeup.