CS3511 - P2 - WEB AUTHENTICATION PRACTICAL

Overview

In this practical we are looking at web authentication; the aim is to explore common authentication flaws and to get a feeling for how hackers work to circumvent authentication methods. The aim of this practical is not to teach you how to become a hacker, instead it is the aim to understand what hackers do so that you can implement appropriate defence methods.

In this practical we use *WebGoat* and *WebWolf*, which has been used as well in the previous practical. In addition, a web browser is required and you might also use python.

WebGoat

From the WebGoat's description page: WebGoat is a deliberately insecure application that allows interested developers just like you to test vulnerabilities commonly found in Java-based applications that use common and popular open source components. Now, while we in no way condone causing intentional harm to any animal, goat or otherwise, we think learning everything you can about security vulnerabilities is essential to understanding just what happens when even a small bit of unintended code gets into your applications. What better way to do that than with your very own scapegoat? Feel free to do what you will with him. Hack, poke, prod and if it makes you feel better, scare him until your heart's content. Go ahead, and hack the goat. We promise he likes it. Thanks for your interest! The WebGoat Team.

For this practical WebGoat is installed on the server X.X.X.X on port 8080. Using a web browser you can navigate to *http://X.X.X.X:8080/WebGoat/* to interact with the service. As a first step you have to create an account by providing a username and password. After logging in you will be provided with a bunch of hacking exercises. In this practical we will look at a subset of these exercises as explained later.

WebWolf

From the WebWolf's description page: Some (WebGoat) challenges requires to have a local web server running. WebWolf is for you the attacker it helps you while solving some of the assignments and challenges within WebGoat. An assignment might for example require you to serve a file or connect back to your own environment or to receive an e-mail. In order to not let you run WebGoat open and connected to the internet we provided these tools in this application, called WebWolf.

For this practical WebWolf is installed on the server X.X.X.X on port 9090 (Both, WebGoat and WebWolf are local installations which can be reached from the machines in the lab but not from the wider Internet).

PART1: Setup

This part is only required if you have not done this in the practical 2 weeks ago.

Use a web browser to navigate to WebGoat: *http://X.X.X.X:8080/WebGoat/*. Create an account on the system by providing a username and password.

PART2: Authentication Bypasses

Navigate to the *Broken Authentication* section and in there to the subsection *Authentication Bypasses.*

Complete the 2 lesson steps.

Step 2 can be completed using the web browser development tools or by using a proxy intercepting and modifying the POST message.

PART3: Password Reset

Navigate to the *Broken Authentication* section and in there to the subsection *Password reset.*

Complete the 7 lesson steps.

PART4: JWT Tokens

Navigate to the *Broken Authentication* section and in there to the subsection *JWT Tokens.*

Complete the first 5 lesson steps.

Step 4 can be completed using the web browser development tools or by using a proxy intercepting and modifying the POST message. You might have to find a Base64 encoder/decoder to complete this task.

Step 5 is quite challenging. You need to find (or create) a tool that can run a dictionary attack against the token signature. A python3 program that performs this task is given below. Once the password is recovered, a new JWT token should be created and then submitted.

```python
import base64
import hashlib
import hmac

def hmac_base64(key, message):
        return base64.urlsafe_b64encode(bytes.fromhex(hmac.new(key,
                            message, hashlib.sha256).hexdigest()))

token = '[token goes here]'.split('.')

unsigned_token = (token[0] + '.' + token[1]).encode()

# signature is base64 URL encoded and padding
# has been removed, so we must add it
signature = (token[2] + '=' * (-len(token[2]) % 4)).encode()

with open('google-10000-english.txt', 'r') as fd:
        lines = [line.rstrip('\n').encode() for line in fd]

for line in lines:
        test = hmac_base64(line, unsigned_token)
        if test == signature:
                print('Key: {}'.format(line.decode()))
```

CS3511 Continuous Assessment - PART 2

Please submit an answer to the following question with your CS3511 Continuous Assessment. Your answer should not be longer than half a page (You can use figures or code pieces to illustrate your answer).

Question P2 [2 MARKS]: Web Authentication

How many parts has a JSON Web Token? How are these parts encoded? Describe the function of each of these parts. Highlight the different parts of the JWT shown below.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiw
ibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2Q
T4fwpMeJf36POk6yJV_adQssw5c
```