Criptografía y Seguridad *(72.44)*

GUÍA 10: SEGURIDAD EN APLICACIONES (SOLUCIONES)

Actividad 1: SQL injection

Situación 1.

1) La búsqueda se ejecutaría así:

```
Select * FROM ID_pwd where pwd = 'a' or '1=1'
```

Lo que seguramente le daría acceso no autorizado a un atacante (aún si éste no conoce la contraseña real).

Situación 2.

2) "`Alicia'; DROP TABLE usuarios; SELECT * FROM datos WHERE '-' = '-`", se generaría la siguiente consulta SQL, (el color verde es lo que pretende el programador, el azul es el dato, y el rojo, el código SQL inyectado):

```
SELECT * FROM usuarios WHERE nombre = 'Alicia';
DROP TABLE usuarios;
SELECT * FROM datos WHERE '-' = '-';
```

3) Se debe de restringir la entrada de datos de manera que se acepte únicamente lo necesario y se filtren caracteres de control.

- Restringir la entrada de datos únicamente a aquellos caracteres que son válidos (por ejemplo, números en teléfonos, letras en nombres y apellidos).
- Evitar en la medida de lo posible caracteres como son: ' \ ? ; : . , * ” | ~ [ ] { }
- Evitar interpretar directamente las entradas de datos como instrucciones, o concatenar directamente las cadenas de entrada como instrucciones (por ejemplo, evitar pegar directamente la cadena de un campo de entrada a una instrucción de búsqueda de SQL, sin antes validar que la cadena no contiene caracteres inválidos).

Actividad 2: Cross Site Scripting: Failure to preserve web page structure

Situación 1.

1) Almacenó las cookies del que invoco la url.

2) La variable que contiene las cookies del usuario será almacenada en un documento de texto llamado cookies.txt alojado en el servidor del atacante y eso expone peligrosamente la seguridad de nuestro usuario.

Situación 2.

1)

2) Al cargar la pagina nos aparecerá la misma ventanita de saludo pero esta vez al darle click volverá a aparecer, inundando nuestro navegador, ya que siempre que demos click en el botón aceptar del cuadro de dialogo nos seguirá apareciendo la misma ventanita, quitándonos toda posibilidad de interactuar con el sitio ya que la única forma de cerrar este cuadro de dialogo es cerrando la web que lo ejecuta, como vemos ya dejo de ser chistoso, pero hasta aquí llegaron los principiantes.

3) Ahora veamos lo que puede hacer alguien mas experimentado en el XSS.

```
<iframe src=http://mipagina.com/pagina.htm>
```

Y en el código de pagina.htm algo como esto:

```
<SCRIPT TYPE="text/javascript" LANGUAGE=JAVASCRIPT>
if (top.frames.length!=0)
top.location=self.document.location; </SCRIPT>
```

Como podemos observar cada vez que se ingrese a la página donde se refleja esta inyección se mostrara pagina.htm como frame superior.

Ahora veamos algo más dañino, vamos a eliminar todo el contenido de la página de post y vamos a remplazarlo por un mensaje nuestro y se logra de una manera tan sencilla como esta:

```
<script>document.documentElement.innerHTML="Borrada";</script>
```

4) Para prevenir esta vulnerabilidad:

- Usar librerías y frameworks que facilitan la generación de salidas de código apropiadas (librerías anti xss)

- Entender el contexto en el que los datos serán usados y qué codificación se espera. Para cualquier dato que será enviado a otra página web y proviene de fuentes externas, utilizar la codificación adecuada para caracteres no alfanuméricos, usando secuencias de escape cuando corresponda. (Ver XSS Prevention Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html)
- Duplicar controles (para cualquier control de seguridad que se haga del lado del cliente, hacerlos también del lado del servidor)
- Usar mecanismos que automáticamente fuercen la separación de datos y código.
- Asumir que cualquier entrada es potencialmente maligna: aceptar únicamente lo que es estrictamente seguro. Para esto, establecer una lista de entradas aceptadas. Lo que no esté en esa lista, no se acepta.

Actividad 3. Cross-Site Request Forgery (CSRF)

CSRF no se puede rastrear porque es llevada a cabo por la dirección IP de un usuario legítimo. Esto puede llevar a confusión en investigación forense. Los exploits de este tipo de ataque apenas se conocen públicamente.

Actividad 4. Unrestricted upload of file with dangerous type

En este ejemplo, el archivo es movido a un directorio más permanente: `/pictures`.
El problema con el código anterior es que no hay un chequeo en relación al tipo de archivo que está siendo cargado. Asumiendo que el directorio pictures/ está disponible en el web document root, un atacante puede subir un archivo con el nombre **malicious.php**.
Como este archivo termina en ".php", puede ser ejecutado por el web server. En los contenidos de este archivo, el atacante puede usar:

Example Language: **PHP**

```
<?php
  system($_GET['cmd']);
?>
```

Una vez que este código ha sido instalado, el atacante puede ingresar arbitrariamente comandos para ejecutar usando una URL del tipo:
Que ejecuta "ls-l" comando u otro tipo de comando que el atacante desea especificar

```
http://server.example.com/upload_dir/malicious.php?cmd=ls%20-l
```

Actividad 5. Information Exposure Through an Error Message

Ingresando un nombre de usuario que haga que $file no exista, un atacante podrá obtener el pathname y luego usarlo para hacer un path traversal.

Actividad 6. Open Redirect: URL Redirection to Untrusted Site

Modificando el valor URL hacia un sitio malicioso, un atacante puede robar los números de las tarjetas de credito o claves.

Actividad 7. Race condition

En una aplicación de comercio electronico se tiene este código para soportar transferencias entre cuentas.
Toma la cantidad total a transferir, la envía a una nueva cuenta y la deduce de la cuenta original.
Una condición de carrera puede ocurrir entre las llamadas a GetBalanceFromDatabase() y SendNewBalanceToDatabase().
Supongamos que el mismo usuario invoca al programa varias veces en forma simultánea, como por ejemplo haciendo varios requests en una aplicación web. Un ataque puede construirse así:

Balance inicial: 100.00

El atacante hace dos llamadas simultáneas al programa: CALLER 1 y CALLER 2. Ambos para la misma cuenta.

CALLER-1 (atacante) asociado a PROGRAM-1 (la instancia que maneja CALLER-1). CALLER-2 asociado a PROGRAM-2.

CALLER-1 requiere una transferencia de 80.00.

PROGRAM-1 llama a GetBalanceFromDatabase and setea $balance a 100.00

PROGRAM-1 calcula $newbalance en 20.00, y llama a SendNewBalanceToDatabase().

Acá la invocación a SendNewBalanceToDatabase() sufre un retardo.

CALLER-2 pide una transferencia de 1.00.

PROGRAM-2 llama a GetBalanceFromDatabase() y setae $balance a 100.00. Esto ocurre porque el requerimiento anterior de PROGRAM-1 todavía no fue procesado.

PROGRAM-2 determina que el nuevo balance es 99.00.

Después del retardo inicial, PROGRAM-1 graba su balance en la base de datos seteandolo a 20.00.

PROGRAM-2 envía un request de actualización a la base de datos seteando el balance a 99.00

En este momento, el atacante debería tener un balance de 19 pero tiene 99.

Para prevenir esto, el programador tiene varias opciones, incluyendo un lock para evitar multiples requests a una aplicación web, o usar un mecanismo de sincronización que incluya todo el código entre GetBalanceFromDatabase() y SendNewBalanceToDatabase().

Actividad 8. Buffer Overflow. Causas.

- No chequear el tamaño de la entrada.
- No tener en cuenta el tamaño del buffer o calcularlo mal.
- No validar adecuadamente el índice de un arreglo.

Actividad 9. Buffer Overflow.

S.O.	Compilador	Depurador
Linux (pampero)	Gcc 13.2.1	Gdb

Situación1

Compilado normalmente: `gcc -o buffer buffer.c`

Argumento	Salida del programa
AnitaMaria	Ok
AnitaMariaArias	*** stack smashing detected ***: terminated Aborted (core dumped)

```
[mroig@pampero guia10]$ gcc -o buffer buffer.c
[mroig@pampero guia10]$ ./buffer AnitaMaria
[mroig@pampero guia10]$ ./buffer AnitaMariaArias
*** stack smashing detected ***: terminated
```

Compilado con opción para no controlar stack: `gcc -o buffer buffer.c -fno-stack-protector`

Argumento	Salida del programa
AnitaMaria	Ok
AnitaMariaArias	Ok

```
[mroig@pampero guia10]$ gcc -o buffer buffer.c -fno-stack-protector
[mroig@pampero guia10]$ ./buffer AnitaMaria
[mroig@pampero guia10]$ ./buffer AnitaMariaArias
[mroig@pampero guia10]$
```

Solución para evitar el buffer overflow:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int
main(int argc, char *argv[])
{
      char buffer [10];
      if (argc > 1)
      {
            strncpy(buffer, argv[1], sizeof(buffer));
```

```
            buffer[sizeof(buffer)-1] = '\0';
      }
      return EXIT_SUCCESS;
}
```

mroig@pampero:~/cys/2024/guia10

```
#include <stdio.h>
#include <string.h>
int main(int argc, char *argv[]){
        char buffer[10];
        if (argc > 1){
                strncpy(buffer, argv[1], sizeof(buffer));
                buffer[sizeof(buffer)-1] = '\0';
                printf("Se copió: %s\n", buffer);
        }

        return 0;
}
[mroig@pampero guia10]$ gcc -o buffer buffer.c -fno-stack-protector
[mroig@pampero guia10]$ ./buffer AnitaMariaArias
Se copió: AnitaMari
```

Situación 2:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
      short int cookie;
      char buf[4];
      printf("direccion de buf: %p\n direccion de cookie: %p\n", &buf, &cookie);
      gets(buf);
      if (cookie == 0x41424344)
            printf("Explotado!!");
      return EXIT_SUCCESS;
}
```

Primero, se compila con opción para no controlar stack: `gcc -o buffer buffer.c -fno-stack-protector.`
Luego, se ejecuta usando gdb y se obtienen los siguientes datos:

```
6               printf("direccion de buf: %p\n direccion de cookie: %p\n",&buf,&cookie);
(gdb) n
direccion de buf: 0x7fffffffe938
 direccion de cookie: 0x7fffffffe93c
7               gets(buf);
(gdb) print &buf[3]
$1 = 0x7fffffffe93b "\367\377\177"
```

VARIABLE	Dirección	Valor ingresado	Fin programa
cookie	0x7fffffffe93c	ABCD	ok
buf	Buf[0] 0x7fffffffe938 Buf[3] 0x7fffffffe93b		

Es decir, tenemos:

0x7fffffffe938	buf[0]	A
0x7fffffffe939	buf[1]	B
0x7fffffffe93a	buf[2]	C
0x7fffffffe93b	buf[3]	D
0x7fffffffe93c	cookie	

Si ahora se ejecuta poniendo AAAADCBA

```
[mroig@pampero guia10]$ ./buffer
direccion de buf: 0x7ffdbd8f8d68
 direccion de cookie: 0x7ffdbd8f8d6c
AAAADCBA
Explotado!![mroig@pampero guia10]$
```

VARIABLE	Dirección	Valor ingresado	Fin programa
cookie	0x7fffffffe93f	AAAADCBA	**Explotado!**
buf	Buf[0] 0x7fffffffe93b Buf[3] 0x7fffffffe93e		

Explotó porque lo que tenemos es:

0x7fffffffe938	buf[0]	A	
0x7fffffffe939	buf[1]	A	
0x7fffffffe93a	buf[2]	A	
0x7fffffffe93b	buf[3]	A	
0x7fffffffe93c	cookie	D	Es el entero 0x41424344 (en Little Endian)
0x7fffffffe93d		C	
0x7fffffffe93e		B	
0x7fffffffe93f		A	

Situación 3:

Esta situación puede darse si el stack se usa de tal manera que las variables buffer1 y buffer2 están en direcciones más bajas que la dirección de retorno de la función. En esta situación, si en esos buffers se coloca más información de la reservada, se puede pisar el punto de retorno y alterar el correcto funcionamiento del programa.
Si el stack se maneja de otra manera (como actualmente sucede) **esta situación no se presenta**.

Por ejemplo:

DESPLAZAMIENTO = 8
INCREMENTO = 7

	ret	buffer2	buffer1	Dir Retorno	a	b	c	x
Address	0x22FF0C	0x22FF10	0x22FF14	0x22FF1C	0x22FF20	0x22FF24	0x22FF28	0x22FF44
content	0x22FF1C			**0x40136C**	1	2	3	0

	ret	buffer2	buffer1	Dir Retorno	a	b	c	x
Address	0x22FF0C	0x22FF10	0x22FF14	0x22FF1C	0x22FF20	0x22FF24	0x22FF28	0x22FF44
content	0x22FF1C			**0x401373**	1	2	3	0

Las direcciones del programa son:
`Frame address de main: 0022FF50`
La llamada a function es:

```
      00401367    call    0x4012e0 <function>

Frame address de Function: 0x22FF20
```

Muestra: 1, 2, 3 y 0
Por que saltea la instrucción: `0040136C  movl  $0x1,0xfffffffc(%ebp)`
(Move 0x1 into the memory pointed to by ebp - 4)

DESPLAZAMIENTO = 12

INCREMENTO = 7
Muestra: 8, 2, 3 y 1

	ret	buffer2	buffer1	Dir Retorno	a	b	c	x
Address	0x22FF0C	0x22FF10	0x22FF14	0x22FF1C	0x22FF20	0x22FF24	0x22FF28	0x22FF44
content	0x22FF20			0x40136C	1	2	3	0

Después del incremento:

	ret	buffer2	buffer1	Dir Retorno	a	b	c	x
Address	0x22FF0C	0x22FF10	0x22FF14	0x22FF1C	0x22FF20	0x22FF24	0x22FF28	0x22FF44
content	0x22FF20			0x40136C	8	2	3	1

Otro ejemplo:
La llamada a function es:

```
0x080483a9 <main + 33>  call 0x8048354 <function>
```

La siguiuente a esa es:

```
0x080483ae <main + 38>  add  $0x10,%esp
```

La instrucción de movl es:

```
0x080483b1 <main + 41>  movl $0x1,-0x8(%ebp)
```

Y la siguiente a esa es:

```
0x080483b8 <main + 48>  sub $0x8,%esp
```

Así que el desplazamiento debe ser: 48 - 38 = 10
Para encontrar la dirección donde se guarda la dirección de la próxima instrucción:

DESPLAZAMIENTO = 12
INCREMENTO = 10

	buffer2	buffer1	ret	Dir Retorno	a	b	c	x
Address	BF87D65C	BF87D660	BF87D664	BF87D66C	BF87D670	BF87D674	BF87D678	
content			BF87D66C	**80483AE**	1	2	3	0

	buffer2	buffer1	ret	Dir Retorno	a	b	c	x
Address	BF87D65C	BF87D660	BF87D664	BF87D66C	BF87D670	BF87D674	BF87D678	
content			BF87D66C	**80483B8**	1	2	3	0

Actividad 10. Path Traversal.

Un atacante puede ingresar: `../../../etc/passwd`

El programa generará un path de perfil del tipo: `/users/cwe/profiles/../../../etc/passwd`

Cuando el archivo se abre, el sistema operativo resuelve el ../ y accede.

Actividad 11. Integer Overflow o wraparound.

Example Language: **C**

```
nresp = packet_get_int();
if (nresp > 0) {
  response = xmalloc(nresp*sizeof(char*));
  for (i = 0; i > nresp; i++) response[i] = packet_get_string(NULL);
}
```

Actividad 12. Allocation of Resources Without Limits or Throttling

Este código reserva un socket y crea un proceso cada vez que recibe una nueva conexión

El programa no lleva la cuenta de cuántas conexiones se han hecho y no limita el número de las mismas.
Como la creación de un proceso es una operación relativamente cara, un atacante puede, haciendo un número grande de conexiones, hacer que el sistema se quede sin capacidad de CPU, procesos, memoria. (Denial of Service)

Actividad 13.

Explicación en documento aparte.