

Sponsored by:

LA SALLE 












Universitat Ramon Llull




Escalada de privilegios mediante infección ELF





Jesús Olmos González

-  I/O vs Memory
-  Comprobaciones iniciales
-  Cabecera Elf
-  Tabla de segmentos
-  Cálculo de entry en disco
-  Backup de `_start`
-  Código Relocatable y Payload
-  Regeneración
-  Retorno al huesped



Lectura y escritura en disco

-  Mediante syscalls `read()` se llena una estructura de datos `elf`
-  Los cambios se aplican a la estructura `elf`
-  Se guardan mediante syscall `write()`



Mapa en memoria

-  Se mapea el fichero a memoria llamando la syscall `mmap()` contra el descriptor.
-  Se guardan los cambios con `sync()` o `msync()`

Lectura y escritura en disco.

-  En c es comodo, no se necesitan usar punteros
-  En asm es un poco engorroso, hay que crear toda la estructura elf en vez de acceder por distancias relativas.

Mapa en memoria.

-  Es rápido, pero en c se apunta la estructura a la dirección del mapa y se trabaja todo con punteros.
-  En asm no es necesario tener toda la estructura elf, basta con saber el offset donde empiezan los campos que se necesitan.

Comprobaciones iniciales

- ❌ No se debe infectar cualquier objeto, se recomiendan unos requisitos.
 - Que presente formato ELF
 - Que sea ejecutable
 - Que no esté infectado

- La ejecución comienza en el inicio de la sección `.text` (generalmente `_start`), la dirección es en ejecución y no en mapa.
 - `e_entry` → `0x18`
- Número de segmentos.
 - `e_phnum` → `0x2c`
- Inicio de la tabla de segmentos.
 - `e_phoff` → `0x1c`

Tabla de Segmentos

 Inició del segmento en disco

 p_offset → 0x04

 Inicio del segmento en ejecución

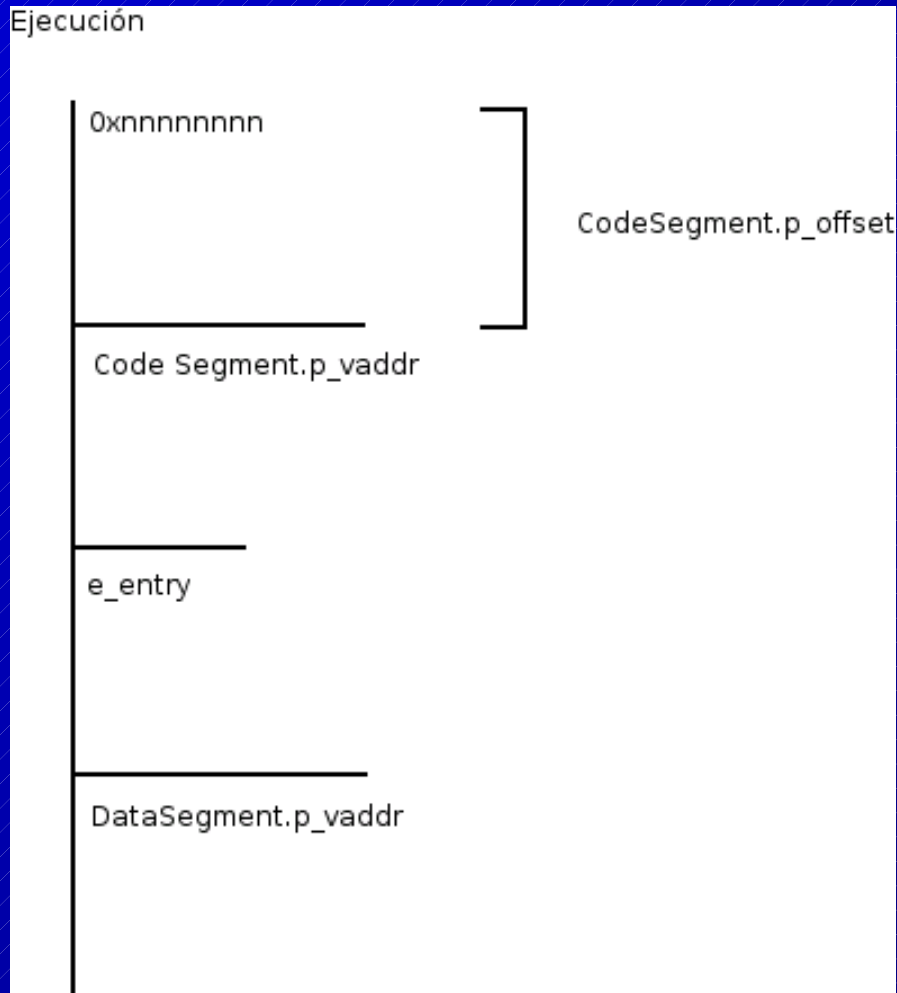
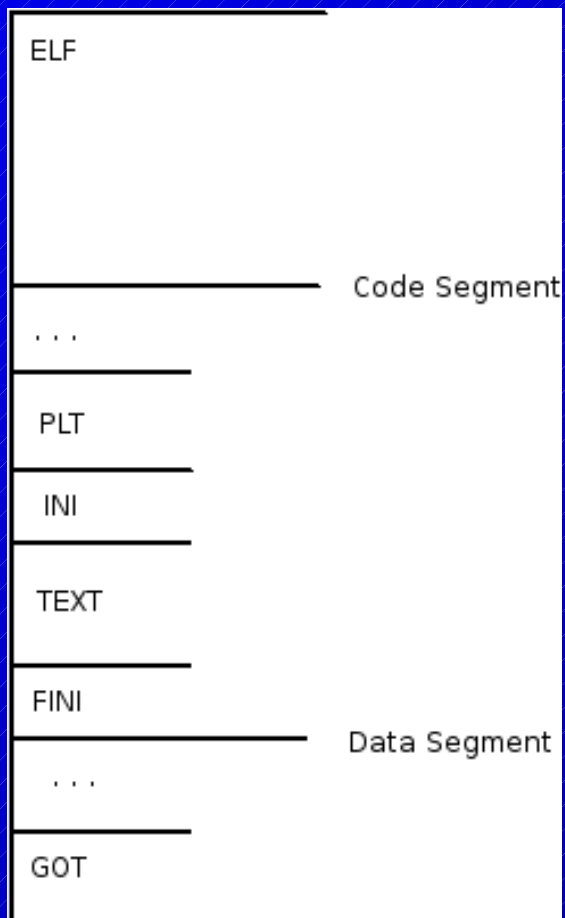
 p_vaddr → 0x08



 Tamaño de un registro de segmento

 0x20 bytes

Cálculo de entry en mapa

$$\text{e_entry} = \text{p_vaddr} + \text{p_offset}$$



-  De cara a la posterior regeneración del código sobrescrito se realiza un backup del mismo tamaño que el código a insertar.
-  Este backup se guarda al final del fichero haciéndolo crecer unos bytes.

Código Relocatable

- El código infectado ha de ser relocatable
 - Delta offset o jump / call
- Ha de saber el path y nombre del huesped para abrirlo y regenerarlo
 - Los datos estan en pila

Creacion de shushi

-  Inconveniente: se necesita un fork para hacerlo con pocas instrucciones

chmod del chmod?

-  Chequeo de root mediante getuid()
-  Setuid al chmod mediante chmod()

Regeneración del huesped

- Se necesita abrir el huesped para recuperar el backup de `_start`
 - syscalls `open()` `lseek()` y `mmap()`
- Se necesita dar permisos de escritura
 - desproteccion con syscall `mprotect()`
- Un codigo no se puede borrar a si mismo
 - copia de la rutina de regeneración a pila, desprotección de pila mediante `mprotect()`

Retorno al huésped

- Tanto los win32 PE como los ELF no necesitan un valor inicial en %ebp
- Existen diversas técnicas de retorno al huésped:
 - push / ret
 - jmp %ebp
- Problema: regeneración de contexto inicial registro a registro

Retorno al huesped

 Localizar dirección donde hay que saltar.

 `leal start_vir(%ebp), %eax`

 Volver a la posición inicial de pila

 `addl $424, %esp`

 Colocar el retorno en saved-ebp

 `movl %eax, 8(%esp)`

 Recuperar contexto inicial

 `popal y jmp *%ebp`

```
gcc infR3.s -o infR3  
./infR3 /usr/bin/binario-escribible
```

 Cuando root ejecute el binario infectado el chmod tendrá setuid y por tanto . . .

 código:

 www.badchecksum.com

 presentación:

 www.7a69ezine.org

 www.fistconference.org

 Contacto

 jolmos@isecauditors.com

 jolmos@7a69ezine.org

 sha0@badchecksum.com

Creative Commons Attribution-ShareAlike 2.0

You are free:

- to copy, distribute, display, and perform this work
- to make commercial use of this work

Under the following conditions:



Attribution. You must give the original author credit.



Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

For any reuse or distribution, you must make clear to others the license terms of this work.

Any of these conditions can be waived if you get permission from the author.

Your fair use and other rights are in no way affected by the above.

This work is licensed under the Creative Commons Attribution-ShareAlike License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/2.0/> or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

Conferencias **FIST**

with the sponsorship of:



FIST Conference



Jesús Olmos

**Barcelona,
Diciembre, 2006**

www.fistconference.org