Mitigación

Hacking Ético

©Ismael Ripoll &
Hector Marco

Universidad Politècnica de València

March 21, 2021

Índice

Qué vamos a trabajar

- Es imposible escribir código libre de fallos.
- Podemos tratar de limitar o mitigar el impacto de los fallos.
- Impedir/dificultar la explotación del fallo.
- Las técnicas de mitigación se desarrollaron a partir de los métodos de explotación.
- Es un área en continua actualización.

Non-eXecutable (I)

Wikipedia

Prevenir que una aplicación o servicio se ejecute desde una región de memoria no ejecutable.

- También recibe los nombres de

  DEP: Data Execution Prevention.

  W⊗X: Write xor Execute.

- El procesador, la MMU en concreto, gestiona varios permisos de las tablas de páginas, entre los que se encuentran los de lectura, escritura y ejecución.
- El procesador emite una trap cuando se intenta ejecutar una instrucción que reside en páginas de memoria que no tiene el flag de ejecución activo.

Non-eXecutable (II)

- Por defecto, nunca están activados a la vez los flags de escritura y ejecución.
- Se puede ver en los mapas de memoria estos permisos.
- También se puede mirar en los flags de los segmentos en el fichero ELF.

```
$ readelf -lW /bin/bash
     .....
Encabezados de Programa:
  Tipo        Desplaz DirVirt DirFísica TamFich TamMem Opt Alin
  PHDR        0x000040 0x0000040 0x0000040 0x0001f8 0x0001f8 R 0x8
  INTERP      0x000238 0x0000238 0x0000238 0x00001c 0x00001c R 0x1
  LOAD        0x000000 0x0000000 0x0000000 0x103038 0x103038 R E 0x200000
  LOAD        0x103d90 0x0303d90 0x0303d90 0x00b7d4 0x015378 RW 0x200000
  DYNAMIC     0x106630 0x0306630 0x0306630 0x000210 0x000210 RW 0x8
  NOTE        0x000254 0x0000254 0x0000254 0x000044 0x000044 R 0x4
  GNU_EH_FRAME 0x0e8300 0x00e8300 0x00e8300 0x0042ac 0x0042ac R 0x4
  GNU_STACK   0x000000 0x0000000 0x0000000 0x000000 0x000000 RW 0x10
  GNU_RELRO   0x103d90 0x0303d90 0x0303d90 0x003270 0x003270 R 0x1
```

Non-eXecutable (III)

- Si apareciera una “E” en el segmento `GNU_STACK`, entonces este ejecutable no tendría la protección NX.
- Esta técnica es ampliamente utilizada en todos los sistemas de usuario, servidores y en parte de los dispositivos IoT.
- Con esto se han eliminado todas las explotaciones basadas en shellcodes.
- Pero todavía quedan nichos donde no se puede usar. Por ejemplo los programas que usan técnicas de JIT (Just In Time compilation).

Stack Smashing Protector (I)

OsDev.org

The Stack Smashing Protector (SSP) compiler feature helps detect stack buffer overrun by aborting if a secret value on the stack is changed.

- Se sitúa un valor aleatorio por proceso, llamado "*canario*" o "*guarda*", en cada marco de pila que usa vectores, entre los vectores y la dirección de retorno.
- Este valor es comprobado antes de retornar de la función. El proceso se aborta en caso de detectar que ha sido modificado.
- Esta guarda la inserta el compilador, y la librería debe gestionar el fallo.

Stack Smashing Protector (II)

- Ejemplo de código insertado por el compilador:

```
$ objdump -d simple-ssp | less
00000000004005ed <protegida>:
  4005ed:  55                    push  %rbp
  4005ee:  48 89 e5              mov   %rsp,%rbp
  4005f1:  48 83 ec 20           sub   $0x20,%rsp
  4005f5:  64 48 8b 04 25 28     mov %fs:0x28,%rax
  4005fe:  48 89 45 f8           mov %rax,-0x8(%rbp)
  400602:  31 c0                 Bla
  400604:  48 8d 45 e0           Bla bla
  [. . . . Cuerpo de la funcion . . . . .]
  400624:  49 8d                 Bla bla
  400626:  90                    Bla
  400627:  48 8b 45 f8           mov -0x8(%rbp),%rax
  40062b:  64 48 33 04 25 28     xor %fs:0x28,%rax
  400634:  74 05                 je 40063b <protegida+0x4e>
  400636:  e8 75 fe ff ff        callq 4004b0 <__stack_chk_fail@plt>
  40063b:  c9                    leaveq
  40063c:  c3                    retq
```

Stack Smashing Protector (III)

- GCC utiliza el flag `-fstack-protector` para activar la generación de código protegido. Podemos desactivar el SSP con el flag `-fno-stack-protector`.
- En las últimas versiones han añadido un control más fino para:
  - `-fstack-protector-all`: Se añade código de protección a todas las funciones, aunque no declaren vectores.
  - `-fstack-protector-strong`: Se añade código a funciones que declaran y usan vectores de pila, aunque sea de otras funciones.
  - `-fstack-protector-explicit`: Deja que sea el programador el que indique qué funciones se deben proteger con el atributo `stack_protect`.

  Se puede encontrar información sobre el SSP en la hoja de manual del compilador.

Análisis

- El SSP transforma las explotaciones de ejecución de código en denegaciones de servicio.
- Añade una pequeña sobrecarga en la ejecución.
- Si el atacante es capaz de averiguar el valor del canario (la guarda), entonces se lo podrá saltar, sobre escribiendo la memoria con el mismo valor.
- El valor del canario es heredado por los procesos hijo y solo se renueva cuando se llama a `exec()`.
- Algunas implementaciones ponen a cero uno de los bytes del canario para frenar los fallos causados por copias de cadenas.
- En función del tipo de fallo, puede que sea posible averiguar el valor del canario por prueba y error, probando bytes de forma secuencial.
- En general, el SSP es una protección muy efectiva.

Address Space Layout Randomization (I)

Wikipedia

Address space layout randomization (ASLR) is a computer security technique involved in preventing exploitation of memory corruption vulnerabilities. In order to prevent an attacker from reliably jumping to, for example, a particular exploited function in memory, ASLR randomly arranges the address space positions of key data areas of a process, including the base of the executable and the positions of the stack, heap and libraries. is changed.

- Tal como hemos visto, la mayoría de ataques de ejecución de código necesitan conocer las direcciones donde se ejecuta el proceso objetivo.
- El ASLR hace que cada vez que se carga un programa en memoria (`exec()`), se carge en direcciones aleatorias.

Address Space Layout Randomization (II)

- Para ello es necesario que el código sea PIC (o PIE).

  PIC: Position Independend Code. Usado en la librerías, para poder ser compartidas en cualquier dirección de memoria virtual. PIC es código que se puede situar en cualquier dirección y que a demás puede ser compartido.

  PIE: Position Independent Executable. Es como el PIE pero no está preparado para ser compartido, por lo que es más eficiente que el PIC. Se utiliza para los ejecutables.

- Esto es, las referencias a memoria se hacen respecto al contador de programa. No debe utilizar direcciones absolutas.
- La mayoría de los procesadores disponen del modo de direccionamiento relativo al contador de programa. Pero Intel i386 no! Hubo que esperar a los procesadores de 64 bits.

Address Space Layout Randomization (III)

- Por tanto, implementar PIE o PIC en i386 es bastante costoso. Tuvo que llegar AMD con los 64bits para añadir este tipo de direccionamiento.
- Hoy en día el código PIE y PIC es tan eficiente como el de direcciones fijas y por tanto el ASLR no añade ningún sobrecoste de ejecución.
- Un proceso está compuesto de un conjunto de zonas de memoria (stack, libs, heap, rodata, text, etc.), cada una de ellas puede estar randomizada o no.
- Otro elemento importante es **cuánta entropía** tienen estas zonas.
- El ASLR es un concepto simple pero con muchas implementaciones y versiones.

Ejecutable randomizables (I)

- Para saber si un ejecutable está preparado para ser randomizado:

```
$ readelf -h /bin/cat
Encabezado ELF:
Mágico:  7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
Clase:                             ELF64
Datos:                             complemento a 2, little endian
Versión:                           1 (current)
OS/ABI:                            UNIX - System V
Versión ABI:                       0
Tipo:                              DYN (Fichero objeto compartido)
Máquina:                           Advanced Micro Devices X86-64
Versión:                           0x1
Dirección del punto de entrada:        0x2710
......
```

- Si el tipo es `DYN`, si que es randomizable, pero si es `EXEC` entonces utiliza direcciones absolutas.

Ejecutable randomizables (II)

- Otra forma de verlo es ejecutándolo varias veces y observar sus mapas:

```
$ for i in {0..10} ; do cat /proc/self/maps; done | grep heap
55d2a74bb000-55d2a74dc000 rw-p 00000000 00:00 0                  [heap]
55fa877d9000-55fa877fa000 rw-p 00000000 00:00 0                  [heap]
557541dbb000-557541ddc000 rw-p 00000000 00:00 0                  [heap]
56056b793000-56056b7b4000 rw-p 00000000 00:00 0                  [heap]
556193fd4000-556193ff5000 rw-p 00000000 00:00 0                  [heap]
55b730628000-55b730649000 rw-p 00000000 00:00 0                  [heap]
5605f3bbb000-5605f3bdc000 rw-p 00000000 00:00 0                  [heap]
55ed1c1ed000-55ed1c20e000 rw-p 00000000 00:00 0                  [heap]
5614b405f000-5614b4080000 rw-p 00000000 00:00 0                  [heap]
5630b76e2000-5630b7703000 rw-p 00000000 00:00 0                  [heap]
55b39a95f000-55b39a980000 rw-p 00000000 00:00 0                  [heap]
```

Desactivar el ASLR (I)

- Es posible desactivar el ASLR de dos formas:

1. Por proceso, usando el comando `setarch`:

```
$ for i in {0..5} ; do
   setarch x86_64 -R cat /proc/self/maps;
done | grep heap
55555575d000-55555577e000 rw-p 00000000 00:00 0                  [heap]
55555575d000-55555577e000 rw-p 00000000 00:00 0                  [heap]
55555575d000-55555577e000 rw-p 00000000 00:00 0                  [heap]
```

   El flag "`-R`": Disables randomization of the virtual address space.

2. Del sistema, escribiendo un cero en `/proc/sys/kernel/randomize_va_space`:

```
# echo "0" > /proc/sys/kernel/randomize_va_space
```

   Para esto es necesario ser root.

Análisis

- El ASLR es la técnica que cubre o mitiga más tipos de ataques.
- Su eficacia depende de que todas las zonas de memoria estén aleatorizadas y cuán aleatorias sean.
- En sistemas de 32bits, solo se aleatorizan 8 bits de las direcciones. Esto permite realizar ataques de prueba y error con éxito casi garantizado en unos pocos segundos.
- Los procesos hijo que no han llamado a `exec()` heredan el mapa de memoria de su padre. Por tanto todos lo hijos tienen el mismo mapa.
- Los ataques para bypassear el ASLR son avanzados.

Nuevas técnicas

Indirect Branch Tracking (I)

- Es una nueva característica del procesador.
- Los saltos (calls y jmps) solo pueden saltar en posiciones conocidas.
- La instrucción destino de un salto “DEBE” ser `ENDBRANCH`.
- El compilador debe generar código con instrucciones de `ENDBRANCH` en todos los destinos de los saltos.
- Cuando el prococesador salta a un instrucción “normal” se produce una excepción.
- Protege contra el secuestro de saltos forward. Pero no los returns de las funciones.
- Intel implementa una versión de esta solución con el nobre de “Control-Flow Enforcement Technology” CET.

Indirect Branch Tracking (II)

- En Intel el opcode de `ENDBRANCH` es el mismo que el de una sintrucción de nop, para conseguir compatibilidad con procesadores anteriores.
- *The ENDBRANCH instruction is a new instruction added to ISA to mark legal target for an indirect branch or jump. Thus if ENDBRANCH is not target of indirect branch or jump, the CPU generates an exception indicating unintended or malicious operation. This specific instruction has been implemented as NOP on current Intel processors for backwards compatibility (similar to several MPX instructions) and pre-enabling of software.*
- La versión 8 de GCC ya incorpora este mecanismo, controlado con los flags `-fcf-protection=[full | branch | return | none]`.

Indirect Branch Tracking (III)

```
/* gcc-8 -fcf-protector=full
         cfi-test.c */
void solo(){
  return ;
}
int main(){
  solo();
  return 0;
}
```

Listing 1: cfi-test.c

```
00000000000005d5 <solo>:
 5d5: f3 0f 1e fa   endbr64
 5d9: 55            push  %rbp
 5da: 48 89 e5      mov   %rsp,%rbp
 5dd: 90            nop
 5de: 5d            pop   %rbp
 5df: c3            retq

00000000000005e0 <main>:
 5e0: f3 0f 1e fa   endbr64
 5e4: 55            push  %rbp
 5e5: 48 89 e5      mov   %rsp,%rbp
 5e8: b8 00 00 00 00 mov  $0x0,%eax
 5ed: e8 e3 ff ff ff callq 5d5 <solo>
 5f2: b8 00 00 00 00 mov  $0x0,%eax
...
```

Listing 2: cfi-test.c

Shadow Stacks (I)

- El objetivo es detectar por hardware cualquier alteración de la dirección de retorno que se guarde en la pila.
- El procesador mentiene automáticamente una segunda pila sólo con las direcciones de retorno.
- Cuando se retorna de una función se comprueba que los valores de las dos pilas conicidan.
- *A shadow stack is a second stack for the program that is used exclusively for control transfer operations. This stack is separate from the data stack and can be enabled for operation individually in user mode or supervisor mode. When shadow stacks are enabled, the CALL instruction pushes the return address on both the data and shadow stack. The RET instruction pops the return address from both stacks and compares them. If the return addresses from the two stacks do*

Shadow Stacks (II)

not match, the processor signals a control protection exception (#CP). Note that the shadow stack only holds the return addresses and not parameters passed to the call instruction. See Figure 1 for an illustration of shadow stack operations on near call and ret instruction.

- *The shadow stack is protected from tamper through the page table protections such that regular store instructions cannot modify the contents of the shadow stack. To provide this protection the page table protections are extended to support an additional attribute for pages to mark them as "Shadow Stack" pages. When shadow stacks are enabled, control transfer instructions/flows like near call, far call, call to interrupt/exception handlers, etc. are allowed to store return addresses to the shadow stack. However stores from instructions like MOV, XSAVE, etc. will not be allowed. Likewise*

Shadow Stacks (III)

control transfer instructions like near ret, far ret, iret, etc. when they attempt to read from the shadow stack the access will fault if the underlying page is not marked as a "Shadow Stack" page. This paging protection detects and prevents conditions that cause an overflow or underflow of the shadow stack or any malicious attempts to redirect the processor to consume data from addresses that are not shadow stack addresses.

Links (I)

- intel-release-new-technology-specifications-protect-rop-attacks
- control-flow-enforcement-technology-preview.pdf
- gnu-compiler-collection-8-gcc-8-transitioning-new-compiler