PLT and GOT

binary exploitation

Marco Garlet
LaSER
University of Milan

March 16, 2022

Overview

1. Basics

2. bypass NX and ASLR

PLT e GOT

- sezioni di un eseguibile ELF utilizzate per risolvere l'indirizzo di una funzione di libreria.
- GCC disabilita di default lazy binding '-z now' (ELF binary hardening feature introdotta in Ubuntu 16.10) (sebbene per il linker è ancora abilitato).
- Runtime lazy binding: migliora le startup performance "rinviando" la risoluzione degli indirizzi di libreria.

PLT e GOT

- Tutte le call associate alle chiamate di funzioni di libreria avranno una forma del tipo *nome_funzione@plt*
- L'indirizzo a cui queste *call* si riferiscono (prima dell'esecuzione dell'eseguibile) è una entry della ProcedureLinkingTable il cui indice differisce in base alla funzione che si sta chiamando.

Examples

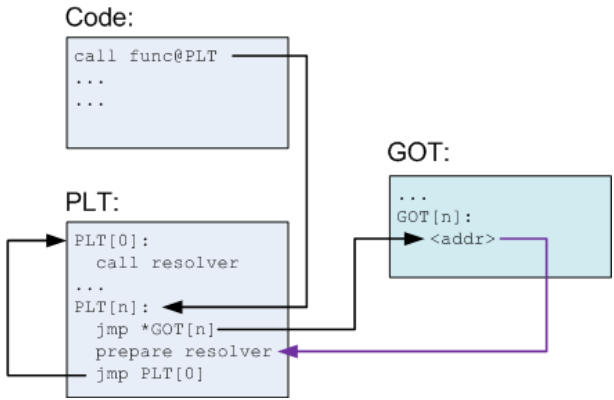
Tutte le printf avranno una call alla stessa entry della PLT.

PLT

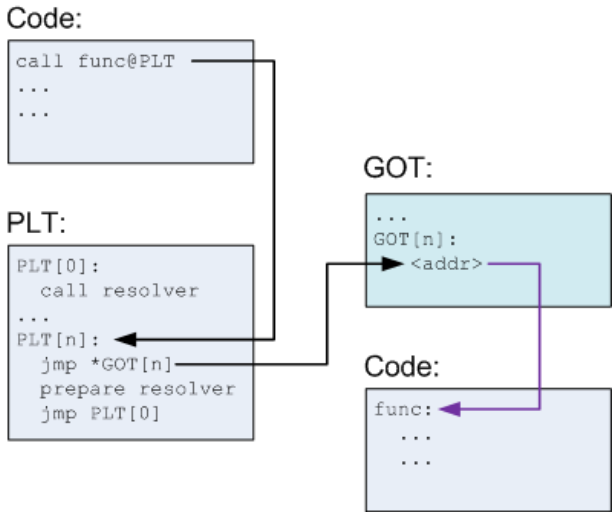
Nella entry della PLT avrò 3 istruzioni principali:

1. Un salto all'indirizzo contenuto nella entry della GOT associata.
2. La push di un parametro nello stack.
3. Un salto a PLT[0] che chiama il resolver.

PLT&GOT


Code:
call func@PLT
...
...
PLT:
PLT[0]:
call resolver
...
PLT[n]:
jmp *GOT[n]
prepare resolver
jmp PLT[0]
GOT:
...
GOT[n]:
<addr>

PLT&GOT


Code:
call func@PLT
...
...
PLT:
PLT[0]:
call resolver
...
PLT[n]:
jmp *GOT[n]
prepare resolver
jmp PLT[0]
GOT:
...
GOT[n]:
<addr>
Code:
func:
...
...

Esempio

Examples

1. la prima volta che si esegue una printf verrà fatta una prima call alla entry della PLT associata a printf.
2. questa ci rimanda al contenuto della entry della GOT indicata nell'istruzione di jmp in quella entry della PLT.
3. La GOT, in fase iniziale, non ha l'indirizzo di printf ma contiene l'indirizzo della seconda istruzione della entry della PLT associata.
4. Vengono eseguite le istruzioni 2 e 3 di PLT[n] (dove n assumo sia associato a printf) che esegue la push di un parametro e salta a PLT[0] che esegue una call a resolver.
5. Il nostro resolver fornisce l'indirizzo della funzione che stiamo cercando e la mette nella GOT.
6. Alle successive chiamate di printf, nella entry PLT associata alla funzione, l'istruzione 1 salterà direttamente alla funzione printf essendo stata modificata la GOT.

Esercizio 1

- git pull `https://github.com/MarcoGarlet/PLT-GOT`
- sfruttare la vulnerabilità per dirottare il flusso di esecuzione (enabled ASLR, NX bit)

Mitigation

- Partial RELRO: forces the GOT to come before the BSS in memory, avoiding BOF attack.
- full RELRO: makes the entire GOT read-only which removes the ability to perform a "GOT overwrite" attack.

Spawn a shell

Con ASLR abilitato:

- La libc è uno *shared object* caricata da un processo a runtime ad un indirizzo ignoto
- gli offset sui simboli e funzioni caratterizzano la libc (si può capirne la versione)
- fare leak di un indirizzo di libreria significa risolvere tutti gli indirizzi della libc
- nella libc a offset prestabiliti, oltre alle funzioni sono presenti i *magic gadeget* i cui offset, data una libc, sono ottenuti mediante *one_gadget*

Esercizio 2

- git pull `https://github.com/MarcoGarlet/PLT-GOT`
- dirottare il flusso di esecuzione (enabled ASLR, NX bit) e spawnare una shell

References

- https://chowdera.com/2021/08/20210823175601610F.html
- https://cseweb.ucsd.edu/~ricko/CSE131/the%20inside%20story%20on%20shared%20libraries%20and%20dynamic%20loading.pdf

The End