Software Security 06

Analisi dinamica

Gianluca Amato
Università di Chieti-Pescara

https://cybersecnatlab.it

License & Disclaimer

License Information

Disclaimer

- We disclaim any warranties or representations as to the accuracy or completeness of this material.
- Materials are provided “as is” without warranty of any kind, either express or implied, including without limitation, warranties of merchantability, fitness for a particular purpose, and non-infringement.
- Under no circumstances shall we be liable for any loss, damage, liability or expense incurred or suffered which is claimed to have resulted from use of this material.

Analisi statica

- Si chiama "**analisi statica di programmi**" l'analisi di software che viene compiuta **senza eseguire i programmi**.
- L'analisi statica si può eseguire:
  - sul codice sorgente;
  - sul codice oggetto.
- Tutti gli strumenti visti fin'ora (`readelf`, `objdump`, `strings`, *Ghidra*) sono strumenti per l'analisi statica.

Analisi dinamica

- Si chiama “**analisi dinamica di programmi**” l’analisi di software che viene compiuta **mentre il programma è in esecuzione**.
- Talvolta l’analisi statica non è sufficiente a capire come funziona il programma.
  - Osservarlo mentre è in esecuzione può aiutarci.

Strumenti per l'analisi dinamica

- Vediamo in questa breve lezione una carrellata di alcuni strumenti di analisi dinamica:
  - `ltrace`
  - `strace`
  - `preeny`
- Si tratta di strumenti minori… rimandiamo alla prossima lezione lo studio dello strumento principale per l'analisi dinamica che è il *debugger*.

ltrace

- `ltrace` consente di tracciare tutte le chiamate alle funzioni di libreria dei programmi C.
- Uso:
  - `ltrace -o <file_traccia> <programma>`
- Installazione:
  - Fedora e derivati: `dnf install ltrace`
  - Debian e derivati: `apt install ltrace`

Esempio di `ltrace`

esempio-trace.c

```c
#include <stdio.h>
#include <stdlib.h>

void main()
{
    char buffer[100];

    printf("Inserisci un numero: ");
    fgets(buffer, sizeof(buffer), stdin);
    int n = atoi(buffer);
    printf("Il numero inserito è: %d\n", n);
}
```

```
printf("Inserisci un numero: ")                          = 21
fgets("12\n", 100, 0x7fee2e2d48e0)                       = 0x7ffd0ed3ec90
atoi(0x7ffd0ed3ec90, 2610, 0x7fee2e2d67c0, 0x7fee2e2d67c0) = 12
printf("Il numero inserito \303\250: %d\n", 12)          = 26
+++ exited (status 26) +++
```

Esercizi

Svolgere le challenge

Software 08 – Dynamic 1

Software 10 – Dynamic 3

strace

- `strace` è simile ad `ltrace`.
  - Non traccia le chiamate alle librerie dinamiche.
  - Traccia invece le **chiamate di sistema**.
- In generale l'output è meno leggibile di `ltrace`, perché spesso le chiamate di sistema sono ad un livello più basso.
- Installazione:
  - Fedora e derivati: `dnf install strace`
  - Debian e derivati: `apt install strace`

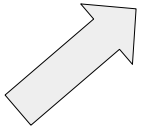
Esempio di `strace`

esempio-trace.c

```c
#include <stdio.h>
#include <stdlib.h>

void main()
{
    char buffer[100];

    printf("Inserisci un numero: ");
    fgets(buffer, sizeof(buffer), stdin);
    int n = atoi(buffer);
    printf("Il numero inserito è: %d\n", n);
}
```



```
execve("./esempio-trace", ["./esempio-trace"], 0x7ffd4f534d90 /* 66 vars */) = 0
brk(NULL)                               = 0x3d985000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f4a30718000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=114675, ...}) = 0
mmap(NULL, 114675, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f4a306fc000
close(3)                                = 0
openat(AT_FDCWD, "/lib64/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

...omissis...

fstat(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x4), ...}) = 0
write(1, "Inserisci un numero: ", 21)   = 21
read(0, "12\n", 1024)                   = 3
write(1, "Il numero inserito \303\250: 12\n", 26) = 26
exit_group(26)                          = ?
+++ exited with 26 +++
```

Esercizi

Svolgere le challenge

Software 09 – Dynamic 2

Software 11 – Dynamic 4

Variabili di ambiente

- L'*ambiente* consiste di un insieme di *variabili di ambiente* che contengono informazioni come
  - nome utente corrente (`USER`)
  - la cartella principale dell'utente (`HOME`)
  - elenco di cartelle dove cercare i programmi (`PATH`)
- Dalla shell Linux è possibile vedere le variabili di ambiente con il comando `env`.

Il comando `env`


```
amato@banzai:~$ env
SHELL=/bin/bash
SESSION_MANAGER=local/unix:@/tmp/.ICE-unix/2253,unix/unix:/tmp/.ICE-unix/2253
CAML_LD_LIBRARY_PATH=/home/amato/.opam/default/lib/stublibs:/home/amato/.opam/defaul
OCAML_TOPLEVEL_PATH=/home/amato/.opam/default/lib/toplevel
COLORTERM=truecolor
HISTCONTROL=ignoredups
XDG_MENU_PREFIX=gnome-
PTYXIS_PROFILE=cca001b7f0ff5a8a9c470cc1672b4522
HOSTNAME=banzai
HISTSIZE=1000
GUESTFISH_OUTPUT=\e[0m
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
MEMORY_PRESSURE_WRITE=c29tZSAyMDAwMDAgMjAwMDAwMAA=
XMODIFIERS=@im=ibus
```

...e continua…..

Modificare le variabili d'ambiente

- Dalla shell di Linux, si può modificare una variabile d'ambiente con il comando
  - `export <variabile>=<valore>`
  - la modifica si applica alla shell corrente e a tutti i programmi da essa lanciati.
- Se invece si vuole lanciare un programma con una variabile di ambiente modificata
  - `<variabile>=<valore> <programma>`
  - quando il programma termina, la variabile ritorna al valore originario.

La variablile LD_PRELOAD

- In Linux, è possibile dire al *caricatore* dei programmi di utilizzare delle librerie di nostra scelta invece di quelle standard.
- Si usa la variabile d'ambiente `LD_PRELOAD`
  - `LD_PRELOAD=<libreria> <programma>`
- Il programma viene caricato normalmente, ma quando accede a funzioni esterne:
  - le cerca prima di tutto nel file di libreria indicato;
  - solo se lì non le trova, le cerca nelle librerie standard di sistema.

preeny

- `preeny` è una collezione di librerie dinamiche da usare con `LD_PRELOAD`
  - utile per disabilitare o alterare il funzionamento di alcune funzioni di libreria;
  - le funzioni modificate possono rendere più facile l'analisi del software.
- Installazione:
  - Fedora e derivati: `dnf install preeny`
  - Debian e derivati: `preeny` non è normalmente disponibile come pacchetto, ma va compilato seguendo le istruzioni nel repository git.

Esercizi

Usando esclusivamente `preeny` ed `ltrace`, svolgere (*di nuovo*) la challenge

SS_1.02 – Slow Printer

Gianluca Amato
Università di Chieti-Pescara

Software Security 06

Analisi dinamica

FINE