Network and System Defense

Alessandro Chillotti*

A.A. 2021/2022

Indice

Indice 1

1 Traccia del progetto 2

2 Contesto dell'attacco 2

3 Implementazione dell'attaco 2

3.1 Costruzione del *init ramdisk* infetto . . . 2

3.2 Definizione della patch utilizzata per la costruzione dell'*init ramdisk* infetto . . . 3

3.2.1 Struttura della patch `horsepill-attack.patch` . . . 3

3.2.2 Modifica del file `klibc-x.x.x/usr/kinit/run-init/runinitlib.c` . . . 4

3.2.3 Modifica al file `klibc-x.x.x/usr/kinit/run-init/Kbuild` . . . 4

3.2.4 File `klibc-x.x.x/usr/kinit/run-init/dnscat.h` . . . 4

3.2.5 File `klibc-x.x.x/usr/kinit/run-init/reinfect.h` . . . 4

3.2.6 File `klibc-x.x.x/usr/kinit/run-init/horsepill.c` . . . 5

3.3 Preparazione del server `dnscat2` . . . 5

4 Simulazione dell'attacco 6

5 Struttura della directory di progetto 6

*alessandro.chillotti@outlook.it

1 Traccia del progetto

È richiesto produrre un payload (utilizzando qualunque vettore) che è in grado di installare una DNS shell (e.g. guardare qui) utilizzando un horsepill attack. Il payload deve essere in grado di sopravvivere agli update del kernel.

2 Contesto dell'attacco

L'attacco implementato all'interno del progetto si basa sull'idea di sfruttare l'*init ramdisk* (*initrd*). Esso è un memory-only file system che viene caricato in memoria dal kernel con lo scopo di svolgere alcune funzioni importanti:

1. Caricare i moduli necessari
2. Rispondere ad eventi di hotplug
3. Cryptsetup
4. Trovare e montare *rootfs*
5. Fare il clean up di *initrd*
6. Eseguire l'*init* process

L'attacco implementato cerca di costruire un *init ramdisk* infetto che svolge le seguenti operazioni:

1. Caricare i moduli necessari
2. Cryptsetup
3. Trovare e montare *rootfs*
4. Enumerare i thread kernel
5. Invocare `clone(CLONE_NEWPID, CLONE_NEWNS)`
6. Effetuare il remount di root
7. Montare uno scratch space
8. Effettuare una `fork()` nella quale si può fare l'hook degli aggiornamenti e si può implementare una backdoor shell
9. Invocare `waitpid()` per attendere l'*init* process nel namespace
10. Effettuare lo shutdown/reboot del sistema

Al passo 5 viene invocata una variante della `fork` che consente di specificare dei flag come `CLONE_NEWPID` e `CLONE_NEWNS` ed in questo caso si sta creando un namespace. All'interno del container vengono eseguite le seguenti operazioni:

1. Effettuare il remount di `/proc`
2. Costruire kernel thread fake
3. Fare il clean up di *initrd*
4. Eseguire l'*init* process fittizio

3 Implementazione dell'attaco

3.1 Costruzione del *init ramdisk* infetto

Per la costruzione dell'*init ramdisk* infetto sono stati implementati i seguenti passaggi:

1. All'interno della directory `/tmp` viene estratto l'*init ramdisk* presente all'interno del sistema originale con il seguente comando

   ```
   unmkinitramfs path-of-original-initrd path-to-initrd-extracted
   ```

2. Si effettua il download del codice sorgente della libreria `klibc`
3. Si applica la patch al sorgente scaricato al passo precedente attraverso l'utilizzo di `quilt`
4. Si utilizza il comando `dpkg-buildpackage` per costruire il pacchetto binario *run-init*
5. Il binario *run-init* generato al passo precedente viene inserito sotto il path `main/usr/bin/run-init` all'interno della directory in cui è stato inserito il risultato dell'estrazione performata al passo 1

6. Viene costruito l'*init ramdisk* considerando l'estrazione eseguita al passo 1 modificata al passo 5 e, per eseguire questa costruzione, sono stati eseguiti i seguente comandi:

   (a) Aggiunta del primo microcode firmware

   ```
   cd early
   find .  -print0 | cpio -null -create -format=newc > /tmp/infected-initrd
   ```

   (b) Aggiunta del secondo microcode firmware

   ```
   cd ../early2
   find kernel -print0 | cpio -null -create -format=newc » /tmp/infected-initrd
   ```

   (c) Aggiunta del ram filesystem

   ```
   cd ../main
   find .  | cpio -create -format=newc | lz4 -l -c » /tmp/infected-initrd
   ```

   Si può notare che è stato utilizzato l'algoritmo di compressione `lz4` con il flag legacy abilitato. Si è scelto di utilizzate l'algoritmo `lz4` piuttosto che l'algoritmo `lzma` perché attraverso l'analisi, effettuata mediante lo strumento `binwalk`, dell'*init ramdisk* originale si è notato l'uilizzo di `lz4`. Quindi, questo ha permesso di mantenere un'aderenza significativa con l'*initrd* originale. Inoltre, l'algoritmo `lz4` è molto più rapido nella computazione rispetto a `lzma`.

7. Si sostituisce l'*init ramdisk* originale con l'*init ramdisk* infetto

8. Si effettua il reboot

L'intera sequenza di passaggi viene eseguita da uno script bash che permette di automatizzare la generazione di un *init ramdisk* infetto. Il file eseguibile bash è il file `infect.sh`.

3.2 Definizione della patch utilizzata per la costruzione dell'*init ramdisk* infetto

3.2.1 Struttura della patch `horsepill-attack.patch`

La patch utilizzata al passo 3 della sotto-sezione precedente è rappresentata dal file `horsepill-attack.patch` ed è composta dalle seguenti parti:

- Modifica del file `klibc-x.x.x/usr/kinit/run-init/runinitlib.c`
- Modifica del file `klibc-x.x.x/usr/kinit/run-init/Kbuild`
- Inserimento del nuovo file `klibc-x.x.x/usr/kinit/run-init/horsepill.c`
- Inserimento del nuovo file `klibc-x.x.x/usr/kinit/run-init/horsepill.h`
- Inserimento del nuovo file `klibc-x.x.x/usr/kinit/run-init/dnscat.h`
- Inserimento del nuovo file `klibc-x.x.x/usr/kinit/run-init/reinfect.h`

3.2.2 Modifica del file `klibc-x.x.x/usr/kinit/run-init/runinitlib.c`

Il file `runinitlib.c` è stato modificato in modo tale che, prima dell'invocazione dell'*init* process si passi ad eseguire in una porzione di codice facente parte dell'attacco. Infatti, oltre all'aggiunta della seguente linea di codice

```
#include "horsepill.h"
```

il file `runinitlib.c` è stato modificato nel seguente modo:

```
perform_hacks();
execv(init, initargs);
return init;
```

In questa maniera, prima di eseguire l'esecuzione dell'*init* process viene invocata la funzione `perform_hacks()` facente parte del file `horsepill.c`.

3.2.3 Modifica al file `klibc-x.x.x/usr/kinit/run-init/Kbuild`

È stato modificato il file `Kbuild`, ovvero è il `Makefile` ed esso contiene delle regole di compilazione come i file oggetto da includere. Infatti, poiché è stato inserito il file `horsepill.c`, il file `Kbuild` ha subito i seguenti cambiamenti:

```
-objs := run-init.o runinitlib.o
-objs := run-init.o runinitlib.o horsepill.o
```

3.2.4 File `klibc-x.x.x/usr/kinit/run-init/dnscat.h`

Il file `dnscat.h` è un file header che contiene solamente due variabili:

- la variabile `dnscat` che è un vettore di `unsigned char` ed esso contiene il codice binario del file eseguibile del client di `dnscat2`;
- la variabile `dnscat_len` che è un `unsigned int` e contiene la dimensione del vettore di `unsigned char`.

L'esistenza di questo file permette, a run-time, la scrittura del contenuto (byte per byte) del file client di `dnscat2`. Il file client originale è disponibile all'interno del repository GitHub di `dnscat2`[1].

3.2.5 File `klibc-x.x.x/usr/kinit/run-init/reinfect.h`

Il file `reinfect.h`, in modo similare al file `dnscat.h`, contiene un array di `unsigned char` che rappresenta il contenuto, byte per byte, di un file eseguibile ed una variabile che contiene il numero di byte dell'array. L'array rappresenta il contenuto byte per byte dello script `reinfect.h`, ovvero uno script creato ad-hoc per permettere la reinfezione della macchina a valle di un aggiornamento.

Per la conversione da file eseguibile a contenuto byte per byte è stato scritto un comando ad-hoc:

```
hexdump -e '16/1 "0x%02x, " "\n"' file
```

Lo script `reinfect.sh` è simile allo script `infect.sh`, ma utilizza il *run-init* salvato all'interno dello scratch space e quindi non necessita di utilizzare la rete per scaricare la libreria `klibc`.

[1] https://github.com/iagox86/dnscat2

3.2.6 File `klibc-x.x.x/usr/kinit/run-init/horsepill.c`

Il file `horsepill.c` contiene gli elementi principali dell'attacco. L'entrypoint è rappresentato dalla funziona chiamata `perform_hacks()` ed essa performa le seguenti operazioni:

1. Enumerazione dei thread kernel

   In questo passaggio si sono acquisite informazioni tramite il file system `proc` poiché contiene una gerarchia di file speciali che rappresentano lo stato corrente del kernel. Per ogni file si va a leggere il contenuto di `/proc/#proc/stat` e si ottengono informazioni come `pid` e `pid name`.

2. Invocazione della `clone(CLONE_NEWPID, CLONE_NEWNS)`

   Questa invocazione ritorna il valore del `pid` ed in base al suo valore ci sono due possibili casi distinti.

Nel caso in cui stia eseguendo il reale *init* process vengono eseguite le seguenti operazioni:

1. Si monta lo scratch space nel path `/lost+found` con permessi `755`
2. Viene effettuato il remount del root

   Questo viene effettuato perché solitamente il root è read-only a questo stage.

3. Si salva il *run-init* infetto all'interno dello scratch space
4. Si scrive il contenuto byte per byte del file `reinfect.sh` nel path `/lost+found/reinfect`
5. Viene spawnato il processo che si occupa dell'implementazione di hook sugli update
6. Si scrive il contenuto byte per byte del client di `dnscat2` nel path `/lost+found/dnscat`
7. Viene spawnato il processo che si occupa dell'implementazione della backdoor shell passandogli come argomenti l'indirizzo IP del server, la porta e la secret

Nel caso in cui stia eseguendo l'*init* process vittima eseguirà i seguenti passaggi:

1. Viene effettuato il remount di `proc`
2. Vengono creati i thread kernel fake

   Vengono eseguite una serie di `fork` ed attraverso una funzione ad-hoc, per ogni thread, vengono eseguite le seguenti operazioni:

   (a) Si apre, in modalità lettura, il file `/proc/self/stat` per fare il retrieve delle informazioni `arg_start` (l'indirizzo sopra il quale gli argomenti di command line sono posizionati) e `arg_end` (l'indirizzo sotto il quale gli argomenti di command line sono posizionati)

   (b) Se l'area di memoria non è sufficiente ne viene allocato una nuova

   (c) Vengono settate le informazioni di cui è stato fatto il retrieve tramite la funzione `prctrl` con l'utilizzo dei flag `PR_SET_MM_ARG_START` e `PR_SET_MM_ARG_END`

   (d) Se le operazioni al passo precedente sono andare a buon fine, viene settato il nome del thread sempre con l'invocazione della funzione `prctl`, ma con il flag `PR_SET_NAME`

Inoltre, l'hook per gli update è stato realizzato grazie al costrutto `inotify`. Per implementare l'hook è stato definitio un `watch` che, nel path `/boot/`, notifica gli eventi di tipo `IN_MOVE`. Nel momento in cui viene eseguita la `MOVE TO` dell'*init ramdisk* aggiornato viene lanciato lo script `reinfect` in modo tale da infettare il nuovo *initrd* andando ad utilizzare il *run-init* salvato all'interno dello scratch space.

3.3 Preparazione del server `dnscat2`

Dalla repository GitHub di `dnscat2`[2] sono stati scaricati i file necessari per l'esecuzione del server. Il server `dnscat2` viene messo in esecuzione, una volta che si risiede all'interno della directory `server`, con il seguente comando:

```
bundle exec ruby dnscat2.rb -dns 'host=X.X.X.X,port=53531'
```

[2] https://github.com/iagox86/dnscat2

4 Simulazione dell'attacco

L'attacco è stato eseguito sfruttando l'interconnesione fra due macchine virtuali collegate alla stessa rete locale.
Le macchine in questione presentano le seguenti caratteristiche:

- l'utilizzo del sistema operativo *Ubuntu 20.04.2* con la versione kernel *5.13.0-40-generic*;
- una svolge il ruolo di client, ovvero rappresenta la macchina vittima dell'attacco;
- una svolge il ruolo di server, ovvero rappresenta la macchina che attende connessioni sulla shell `dnscat2` da parte della macchina client.

5 Struttura della directory di progetto

Il repository GitHub[3] ha la seguente struttura:

- `infect.sh`: la sua esecuzione implementa l'infezione della macchina vittima;
- `reinfect.sh`: la sua esecuzione implementa l'infezione della macchina vittima a valle di un aggiornamento;
- `horsepill-attack.patch`: è il file che contiene la patch che, se applicata, consente la costruzione del *run-init* infetto;
- `src`: contiene i file sorgenti che costituiscono l'attacco
  - `dnscat.h`
  - `horsepill.c`
  - `horsepill.h`
  - `reinfect.h`
- `README.md`: mostra un how-to per svolgere l'attacco.

[3] https://github.com/alessandrochillotti/horsepill-attack