Relazione Progetto NSD

Cristiano Cuffaro
matricola: 0299838
cristiano.cuffaro@outlook.com

8 luglio 2022

Indice

1 Descrizione preliminare del problema e dell'attacco **1**
1.1 Boot del sistema . . . 1
1.2 Utilizzo di initrd . . . 1
1.3 Horse Pill . . . 1

2 Implementazione **2**
2.1 Realizzazione del sistema containerizzato . . . 2
2.2 Orchestrazione esterna al sistema containerizzato . . . 2
2.3 Backdoor shell . . . 3
2.4 Intercettazione degli aggiornamenti dell'initrd . . . 3

3 Rilevamento e prevenzione **3**

1 Descrizione preliminare del problema e dell'attacco

La maggior parte dei sistemi operativi convenzionali *Unix-like*, come Linux, durante la fase di boot fa uso di un *initial ram disk*, chiamato *initrd*. Il problema è che ad oggi quest'ultimo non è sicuro e può essere compromesso per ottenere il controllo del sistema.

1.1 Boot del sistema

Tipicamente, quando si fa uso di un initrd, l'avvio del sistema avviene nel seguente modo:

1. il bootloader carica il kernel e l'initrd;
2. il kernel converte l'initrd in un RAM disk "normale" e libera la memoria utilizzata dal precedente;
3. l'initrd viene montato come root;
4. con UID 0 viene eseguito `/linuxrc` (chiamato anche `/init`), che può essere qualsiasi eseguibile valido (inclusi gli shell script);
5. `linuxrc` monta il file system root "reale";
6. `linuxrc` assegna il file system root alla directory root utilizzando la system call `pivot_root`;
7. `linuxrc` esegue la `exec` di `/sbin/init` sul nuovo file system root, consentendo la sequenza di avvio ordinaria;
8. viene rimosso il file system initrd [1].

1.2 Utilizzo di initrd

È possibile avviare un sistema Linux senza utilizzare un initrd, ma questo normalmente richiede la compilazione di un kernel specifico per l'installazione.

Infatti, se non si usa un initrd non si possono caricare moduli del kernel prima di aver montato la partizione radice, e quindi quest'ultima deve poter essere montata utilizzando solo moduli che sono compilati nel kernel. Per cui, un kernel generico dovrebbe contenere il supporto necessario a montare qualsiasi tipo di partizione radice, e finirebbe così per contenere molti moduli non necessari.

L'utilizzo di un initrd permette invece alle distribuzioni Linux di fornire un kernel precompilato con tutte le funzionalità realizzate come moduli, e di costruire per ciascuna installazione un initrd contenente i moduli necessari per montare il filesystem radice su quella particolare installazione [1].

1.3 Horse Pill

L'idea dell'attacco Horse Pill è quella di modificare il file eseguibile `run-init`[1] nell'initrd e quindi ottiene il controllo sul sistema al momento dell'avvio. Poiché l'initrd è generato dinamicamente e non ci sono controlli di integrità su di esso, è improbabile che un tale cambiamento venga notato attraverso un'osservazione casuale.

Il `run-init` modificato, quando viene eseguito, inserisce l'intero sistema in un container creato sfruttando il meccanismo dei *namespace* presente in Linux; avvia anche un processo backdoor al di fuori di quel container. Tutto il resto, incluso il processo `systemd` e tutti i servizi e le applicazioni di sistema regolarmente previsti, sono in esecuzione all'interno del sistema containerizzato. Il `run-init` compromesso crea anche dei processi e li rinomina in modo che sembrino gli autentici thread del kernel. Gli utenti e gli amministratori che sono vittime dell'attacco non possono vedere i processi e la backdoor in esecuzione al di fuori del sistema containerizzato, quindi dall'interno può apparire come un sistema regolare.

La backdoor installata con l'Horse Pill crea una sorta di connessione effimera sfruttando un tunnel DNS, che consente di comunicare con il server attaccante per ricevere comandi e scambiare dati.

[1] All'avvio, una volta montato il disco initrd il kernel esegue `/linuxrc` che, tra le altre cose, lancia il binario `/usr/bin/run-init` per avviare il processo user-mode iniziale del sistema.

2 Implementazione

Ciò che viene fatto dall'initial ramdisk infetto è eseguire i seguenti task:

- **caricamento dei moduli necessari per l'architettura specifica**
- **rispondere agli eventi di hotplug**
- **cryptsetup (opzionale)**
- **ricerca e montaggio del file system rootfs**
- enumerazione dei thread del kernel
- `clone(CLONE_NEWPID, CLONE_NEWNS)`
  - remount di `/proc`
  - creazione dei kernel thread fittizi
  - **clean up e smontaggio dell'initrd**
  - **esecuzione di `init`**
- remount di root
- montaggio di uno spazio di lavoro *scratch*
- `fork()`
  - aggancio agli aggiornamenti dell'initrd
  - esecuzione della shell backdoor
- `waitpid()`
- catch dello shutdown o del reboot

dove in grassetto sono evidenziate le attività eseguite anche da un ramdisk regolare.

2.1 Realizzazione del sistema containerizzato

Per la realizzazione del sistema containerizzato, all'interno della funzione principale dell'attacco (`do_attack()`), la cui invocazione è iniettata nel programma infetto, è utilizzata la system call `clone`, specificando i seguenti flag:

- `CLONE_NEWPID` per creare il processo in un nuovo PID namespace;
- `CLONE_NEWNS` per creare il processo in un nuovo mount namespace, inizializzato con una copia del namespace del parent;
- `SIGCHLD` per segnalare il parent dell'eventuale terminazione del nuovo processo.

Al ritorno dall'invocazione, il processo child svolge il seguente lavoro:

1. enumera i thread del kernel accedendo ai file `/proc/[pid]/stats`;
2. esegue il *remount* di `/proc` per perdere le informazioni sui processi in esecuzione al di fuori del nuovo namespace;
3. esegue la creazione dei kernel thread fittizi[2] bloccando tutti i possibili segnali che possono ricevere, ad eccezione di `SIGTERM`[3], per cercare di mascherare il più possibile la reale natura di tali processi;
4. esce dalla funzione proseguendo il lavoro regolarmente previsto allo startup, i.e. spawn di *init*.

In questo modo, il sistema può proseguire con il normale avvio e senza mostrare evidenze forti sulla containerizzazione realizzata.

2.2 Orchestrazione esterna al sistema containerizzato

Dopo l'esecuzione della `clone`, il processo parent svolge in background il seguente lavoro:

1. installa un gestore di segnale per `SIGINT` che esegue solamente la consegna di tale segnale al processo child;

[2] I processi creati eseguono `pause()` in un ciclo senza fine.
[3] Il segnale `SIGTERM` deve continuare ad essere ricevibile perché è inviato da *init* a tutti i suoi child quando il sistema deve andare in shutdown [2].

2. disabilita la possibilità di riavviare il sistema utilizzando la sequenza CAD (*ctrl-alt-delete*), in modo tale che essa causi solamente l'invio del segnale `SIGINT` al processo *init*;
3. esegue il *remount* della radice del file system (`"/"`) per renderla scrivibile;
4. monta un file system di tipo `tmpfs` su `"/lost+found"` da utilizzare come *scratch space* in cui mantenere:
   - lo script `extractor.sh` che estrae il binario `run-init` infetto dall'initrd (utile per eseguire in maniera rapida future infezioni),
   - il binario `dnscat` corrispondente al client per realizzare una backdoor shell,
   - lo script `infect.sh` che esegue in maniera automatica l'infezione dell'initrd;
5. esegue lo script per l'estrazione del binario infetto dall'initrd e lancia in background i seguenti processi:
   - il client per ricevere comandi da remoto mediante una backdoor shell,
   - un processo *watcher* che intercetta gli aggiornamenti dell'initrd e replica l'infezione;
6. rimane in attesa della terminazione dei processi child.

La terminazione del client *dnscat* e del *watcher* è gestita lanciando nuovamente tali processi, mentre per il child *init* è prevista una gestione più articolata:

- se il processo ha terminato l'esecuzione senza ricevere un segnale di `SIGHUP` o `SIGINT`, allora si esce producendo un messaggio d'errore;
- altrimenti, si esegue il riavvio o lo spegnimento del sistema in accordo alla ricezione del segnale di `SIGHUP` o `SIGINT`, rispettivamente.

  Il motivo è che l'esecuzione di `reboot` o `poweroff` all'interno di un PID namespace diverso da quello iniziale, ha l'effetto di inviare un segnale al processo *init* di tale namespace:
  - `SIGHUP` per riavviare il sistema,
  - `SIGINT` per spegnere il sistema [3].

2.3 Backdoor shell

Per ottenere il controllo del sistema da un server remoto, è stato utilizzato il tool *dnscat2* [4] che implementa un tunnel DNS.

Eseguendo in background il binario `dnscat`, si tenta di stabilire una trasmissione di dati tra la macchina vittima e quella attaccante, creando un canale criptato *C&C* (*command-and-control*) al di sopra del protocollo DNS.

L'uso del DNS tunneling è particolarmente vantaggioso per l'esfiltrazione dei dati basata sulla codifica delle informazioni all'interno dei nomi di dominio (poiché questi possono contenere praticamente qualsiasi cosa) e, inoltre, non viene stabilita alcuna connessione rilevabile con i classici tool di monitoraggio della rete (e.g. `netstat`). Ciò contribuisce a mantenere apparentemente "normale" il sistema containerizzato.

2.4 Intercettazione degli aggiornamenti dell'initrd

Per evitare che gli aggiornamenti del kernel portino alla ricostruzione di un initrd regolare, viene lanciato in background un processo che sfrutta l'API `inotify` per monitorare gli eventi del file system che coinvolgono la directory `/boot`.

Con questo meccanismo è possibile intercettare ogni operazione di creazione o rinomina dei file all'interno della directory osservata e, in caso di modifica dell'initrd, intervenire eseguendo lo script `infect.sh` per iniettare nuovamente il binario `run-init` infetto all'interno dell'initial ramdisk.

3 Rilevamento e prevenzione

Il sistema containerizzato è realizzato facendo uso del meccanismo dei *namespace*, ovvero creando tutti i processi utente all'interno di nuovi PID e mount namespace. Esistono diversi modi per distinguere il sistema containerizzato da quello regolare e alcuni di questi sono descritti di seguito.

- Per ogni processo in Linux esiste la directory `/proc/[pid]/ns/` contenente un link simbolico per ciascun namespace, che riporta il tipo di namespace e l'*inode number* associato [5]. Eseguendo il comando `ls -l /proc/1/ns/` è possibile osservare:

  - al di fuori del sistema containerizzato

```
mnt -> mnt:[4026531840]
pid -> pid:[4026531836]
pid_for_children -> pid:[4026531836]
```

  - all'interno del sistema containerizzato

```
mnt -> mnt:[4026532165]
pid -> pid:[4026532166]
pid_for_children -> pid:[4026532166]
```

  ovviamente i link simbolici riferiti agli altri namespace non differiscono.

- In un sistema regolare il demone `kthreadd`, avente PID `2`, ha `0` come PPID (*Parent PID*), ma in quello containerizzato esso è un processo fittizio creato da *init* e quindi eseguendo `cat /proc/2/status | grep PPid` si ottiene che il PPID corrisponde a `1`.
- In un sistema regolare inviare segnali da terminale ai demoni del kernel non produce alcun effetto apparente (e.g. inviando i segnali tramite `top` non si verificano cambiamenti di stato), mentre nel sistema containerizzato i segnali non bloccabili (i.e. `SIGKILL` e `SIGSTOP`) hanno effetto sui finti thread del kernel.

Quando vengono eseguiti gli aggiornamenti del kernel è necessario anche ricostruire il ram disk, perché nuove versioni del kernel richiedono nuove versioni del ram disk. Si possono avere script compromessi sul disco che vanno ad intercettare questi aggiornamenti (e.g. per Ubuntu è possibile riporli nella directory `/etc/initramfs/post-update.d`) ed è difficile rilevare queste minacce in maniera automatica. Una possibile soluzione potrebbe essere quella di verificare in maniera crittografica gli aggiornamenti, facendo sì che iniezioni di eseguibili infetti (come nell'Horse Pill) siano rilevabili al momento della verifica.

Riferimenti bibliografici

[1] Using the initial RAM disk (initrd), https://www.kernel.org/doc/html/latest/admin-guide/initrd.html

[2] M. Kerrisk, "The Linux Programming Interface, sezione 37.3: Guidelines for Writing Daemons", 2010

[3] reboot(2) - Linux manual page, DESCRIPTION: "Behavior inside PID namespaces"

[4] Ron Bowes (a.k.a. iagox86), "dnscat2", https://github.com/iagox86/dnscat2

[5] namespaces(7) - Linux manual page, DESCRIPTION: "The /proc/[pid]/ns/ directory"