Relazione Progetto NSD

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

25 giugno 2022

Indice

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.

1.2 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.

1.3 Horse Pill

L'idea dell'attacco è che il rootkit Horse Pill sostituisce il file `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.

Quando il rootkit viene eseguito, a valle dell'infezione, 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 processo `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 del rootkit 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 dal rootkit Horse Pill crea una sorta di connessione effimera sfruttando un tunnel DNS, che consente di connettersi al server attaccante per ricevere comandi e scambiare dati.

2 Implementazione

Ciò che viene fatto dall'initial ramdisk infetto sono 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()`

[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.

  - 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 `runinitlib`, 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 montaggio 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 da `runinitlib`, 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:

[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 quando il sistema deve andare in shutdown.

1. installa un gestore di segnale per `SIGINT` che esegue solamente la consegna di tale segnale al processo child;
2. disabilita la possibilità di riavviare il sistema utilizzando la sequenza CAD (*ctrl-alt-delete*);
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 ottenere una backdoor shell;
   - lo script `infect.sh` che esegue in maniera automatica l'infezione dell'initrd;