UNIVERSITÀ
DEGLI STUDI
DI BERGAMO

Dipartimento di

Ingegneria Gestionale, dell'Informazione e della Produzione

Corso di Laurea in

Ingegneria Informatica

Classe LM-32

Integrazione di eBPF in ambiente Android:

sfide e opportunità per l'ispezione di sistema

Candidato:

Mattia Ferrari

Matricola n. 1083721

Relatore:

Chiar.mo Prof. Mattehw Rossi

ANNO ACCADEMICO

2024/2025

Sommario

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Indice

Capitolo 1

Introduzione

1.1 Contesto

Negli ultimi anni i dispositivi mobili hanno assunto un ruolo centrale nell'ambito dell'informatica personale e professionale. Tra i sistemi operativi per dispositivi mobili, Android rappresenta una delle piattaforme più diffuse a livello mondiale e viene utilizzato in una grande varietà di dispositivi, dagli smartphone ai sistemi embedded. La complessità crescente delle applicazioni e del sistema operativo rende sempre più importante disporre di strumenti in grado di analizzare e comprendere il comportamento delle applicazioni durante l'esecuzione.

Il monitoraggio delle attività delle applicazioni è fondamentale in diversi ambiti, tra cui il debugging del software, l'analisi delle prestazioni e la sicurezza informatica. In particolare, la possibilità di osservare le interazioni tra le applicazioni e il sistema operativo consente di individuare anomalie di funzionamento, colli di bottiglia prestazionali e potenziali comportamenti malevoli.

Tradizionalmente il tracing del sistema operativo viene realizzato mediante strumenti specifici che permettono di raccogliere informazioni sull'esecuzione dei processi e sulle attività del kernel. Tali strumenti possono tuttavia introdurre un significativo overhead computazionale oppure richiedere modifiche al sistema operativo, rendendone difficile l'utilizzo in ambienti reali. Inoltre, molti strumenti di analisi disponibili nei sistemi operativi tradizionali, e in particolare in ambiente Linux, non sono direttamente utilizzabili su Android senza adattamenti specifici.

Dal 2014, con l'introduzione dell'extended Berkeley Packet Filter (eBPF) in Linux, è stata resa disponibile una tecnologia che permette di eseguire piccoli programmi in modo sicuro ed efficiente all'interno del sistema operativo. Questo consente di osservare il comportamento delle applicazioni e del sistema senza la necessità di modifiche al codice sorgente o dello sviluppo di moduli dedicati. Per queste ragioni eBPF è diventato uno strumento sempre più utilizzato per attività di tracing e osservabilità nei sistemi Linux.

1.2 Problema

Tra gli strumenti che permettono di utilizzare eBPF in modo pratico, bpftrace costituisce una soluzione particolarmente efficace per il tracing delle applicazioni e del sistema operativo. bpftrace mette a disposizione un linguaggio di scripting ad alto livello che consente di definire sonde e azioni associate agli eventi osservati, permettendo di raccogliere informazioni durante l'esecuzione senza sviluppare programmi eBPF completi. Questo approccio semplifica notevolmente la realizzazione di strumenti di analisi dinamica.

Tuttavia, l'utilizzo di bpftrace in ambiente Android presenta diverse difficoltà pratiche. In molti dispositivi l'accesso alle funzionalità di basso livello del sistema è limitato e spesso non sono disponibili gli strumenti e le librerie necessari per l' esecuzione di programmi basati su eBPF. Inoltre, l'installazione diretta di tali strumenti sul sistema Android può risultare complessa o non praticabile, soprattutto in ambienti di test controllati.

Queste limitazioni rendono difficile l'impiego delle tecnologie basate su eBPF per l'analisi del comportamento delle applicazioni Android, nonostante il loro potenziale per il monitoring del sistema. Risulta quindi necessario definire un ambiente di lavoro che consenta l'utilizzo di bpftrace in modo stabile e riproducibile, permettendo allo stesso tempo l'osservazione delle attività delle applicazioni durante l'esecuzione.

Il problema affrontato in questa tesi consiste quindi nel rendere possibile l'utilizzo di bpftrace per il tracing delle applicazioni in ambiente Android, individuando una

configurazione che consenta di superare le limitazioni tipiche del sistema e di ottenere dati significativi sull'esecuzione delle applicazioni.

1.3 Idea di Soluzione

Per superare le difficoltà legate all'utilizzo degli strumenti basati su eBPF in ambiente Android, in questo lavoro è stata sviluppata una soluzione che permette di eseguire lo strumento bpftrace in un ambiente controllato e riproducibile, mantenendo allo stesso tempo la possibilità di osservare il comportamento delle applicazioni Android.

La soluzione proposta si basa sulla creazione di un ambiente di sviluppo virtualizzato in cui sia possibile eseguire Android e utilizzare strumenti tipicamente disponibili nei sistemi Linux tradizionali. A questo scopo è stato utilizzato l'emulatore Android Cuttlefish, eseguito su sistema operativo Ubuntu, che consente di disporre di un ambiente Android completo e configurabile per attività di sviluppo e sperimentazione.

All'interno dell'ambiente Android è stata installata una distribuzione Linux minimale basata su Debian. Questa distribuzione è stata utilizzata come ambiente di lavoro per l'esecuzione di bpftrace e degli strumenti necessari al tracing. L'utilizzo di una distribuzione Linux separata ha permesso di evitare la compilazione nativa di bpftrace su Android e di semplificare l' installazione delle dipendenze necessarie.

In questo modo è stato possibile realizzare un ambiente che combina la flessibilità degli strumenti Linux con la possibilità di osservare il comportamento di applicazioni eseguite su Android. Su questa base è stato sviluppato un insieme di script e strumenti software che permettono di eseguire sessioni di tracing e di raccogliere automaticamente i dati prodotti da bpftrace.

Capitolo 2

Tecnologie di base

2.1 Linux come base per il tracing

Linux è un sistema operativo di tipo Unix-like ampiamente utilizzato sia in ambito accademico sia industriale grazie alla sua stabilità, flessibilità e natura open source. Il sistema è basato su un'architettura in cui il kernel gestisce le risorse hardware e fornisce ai programmi un'interfaccia standardizzata attraverso le system call, mentre l'insieme dei programmi in spazio utente costituisce l'ambiente operativo vero e proprio. Questa organizzazione rende Linux particolarmente adatto allo sviluppo software e alle attività di sperimentazione, poiché consente un controllo dettagliato del sistema e un facile accesso agli strumenti di analisi e debugging.

Nel presente lavoro è stato utilizzato come sistema operativo di base Ubuntu 22.04 LTS, una distribuzione Linux ampiamente diffusa e supportata a lungo termine. Ubuntu è stata scelta per la disponibilità di pacchetti software aggiornati e per la semplicità di configurazione dell'ambiente di sviluppo. Il sistema Ubuntu ha costituito la piattaforma principale su cui sono stati installati gli strumenti necessari allo svolgimento della tesi, tra cui l'ambiente di virtualizzazione Android Cuttlefish e i componenti utilizzati per il tracing basato su eBPF. L'utilizzo di una distribuzione Linux standard ha permesso di disporre di un ambiente stabile e riproducibile, facilitando l'installazione delle dipendenze e la gestione degli strumenti software impiegati nelle sperimentazioni.

2.2 Android Cuttlefish

Android è un sistema operativo per dispositivi mobili basato sul kernel Linux e progettato per supportare l'esecuzione di applicazioni in un ambiente isolato e controllato. Il sistema è organizzato in una struttura a livelli che comprende il kernel Linux, le librerie di sistema, il runtime Android, il framework delle applicazioni e il livello delle applicazioni utente. Il kernel Linux costituisce il livello più basso dell'architettura e si occupa della gestione delle risorse hardware, della memoria, dei processi, dei dispositivi di input/output e delle comunicazioni di rete. I livelli superiori forniscono invece i servizi necessari all'esecuzione delle applicazioni e definiscono l'ambiente software tipico della piattaforma Android.

Ogni applicazione Android viene eseguita in uno spazio isolato rispetto alle altre applicazioni, generalmente associato a un identificatore utente distinto. Questo meccanismo di isolamento contribuisce alla sicurezza del sistema ma limita l'accesso diretto alle risorse e alle funzionalità di basso livello. Inoltre, molte componenti tipiche delle distribuzioni Linux tradizionali non sono presenti oppure risultano fortemente ridotte, rendendo più complesso l'utilizzo di strumenti di analisi progettati per ambienti Linux standard.

Nonostante Android utilizzi il kernel Linux, l'ambiente software differisce in modo significativo da quello di una distribuzione Linux tradizionale. In particolare, l'organizzazione del file system, la disponibilità delle librerie e la gestione dei permessi rendono spesso necessario un adattamento degli strumenti sviluppati per sistemi Linux convenzionali. Questo aspetto è particolarmente rilevante nel caso degli strumenti di tracing, che richiedono generalmente l'accesso a funzionalità di basso livello del sistema operativo.

Per lo sviluppo e la sperimentazione è stato utilizzato Android Cuttlefish, un emulatore ufficiale della piattaforma Android progettato per l'esecuzione su sistemi Linux. Cuttlefish permette di eseguire una o più istanze complete del sistema operativo Android all'interno di un ambiente virtualizzato, consentendo di simulare il comportamento di un dispositivo reale senza la necessità di utilizzare hardware fisico.

Cuttlefish utilizza tecnologie di virtualizzazione disponibili in ambiente Linux per eseguire il sistema Android in una macchina virtuale che riproduce i principali componenti di un dispositivo reale. L'ambiente virtualizzato include il kernel Android, il sistema operativo e i servizi necessari all'esecuzione delle applicazioni. Questo approccio permette di ottenere un ambiente di test controllato e riproducibile, caratteristica particolarmente importante nelle attività di sviluppo e sperimentazione.

Nel presente lavoro Cuttlefish è stato utilizzato come ambiente di esecuzione del sistema Android su cui effettuare le attività di tracing. L'emulatore è stato eseguito su sistema operativo Ubuntu, che ha costituito la piattaforma principale per l'installazione e la configurazione degli strumenti necessari allo svolgimento della tesi. L'utilizzo di un ambiente virtualizzato ha permesso di effettuare le sperimentazioni in modo riproducibile e di controllare la configurazione del sistema senza intervenire su dispositivi fisici.

2.3 Tracing, eBPF e bpftrace

2.3.1 Tracing delle applicazioni

Il tracing delle applicazioni è una tecnica di analisi che consiste nella raccolta sistematica di informazioni durante l'esecuzione di un programma o di un sistema operativo, con l'obiettivo di osservare il comportamento dinamico del software e le sue interazioni con il sistema. A differenza dell'analisi statica, che si basa sull'esame del codice sorgente o dei file eseguibili, il tracing consente di studiare ciò che avviene realmente durante l'esecuzione, permettendo di ottenere informazioni temporali sugli eventi generati dal sistema.

Nel contesto dei sistemi operativi moderni, il tracing può essere effettuato a diversi livelli di osservazione. A livello applicativo è possibile monitorare le operazioni effettuate da un processo, come la creazione di nuovi processi, l'accesso ai file o le comunicazioni di rete. A livello di sistema operativo è invece possibile osservare eventi gestiti dal kernel, come le chiamate di sistema, la schedulazione dei processi o le operazioni di input/output. Questa possibilità di osservare eventi a diversi livel-

li rende il tracing uno strumento fondamentale per comprendere il funzionamento complessivo di un sistema software.

Le tecniche di tracing vengono utilizzate principalmente per tre scopi. Il primo è il debugging del software, dove l'osservazione dettagliata delle operazioni eseguite da un programma permette di individuare errori o comportamenti inattesi. Il secondo è l'analisi delle prestazioni, in cui il tracing consente di identificare colli di bottiglia e di comprendere come le risorse del sistema vengono utilizzate dalle applicazioni. Il terzo ambito è quello della sicurezza informatica, dove l'osservazione delle attività dei processi permette di individuare comportamenti anomali o potenzialmente malevoli.

Tradizionalmente il tracing in ambiente Linux è stato realizzato tramite strumenti basati su meccanismi come ptrace, utilizzati ad esempio da programmi come strace, oppure tramite infrastrutture di tracing integrate nel kernel come ftrace e perf. Questi strumenti permettono di ottenere informazioni dettagliate sul comportamento del sistema, ma possono risultare complessi da utilizzare oppure introdurre un overhead significativo durante l'esecuzione.

2.3.2 Tracing mediante eBPF

Una delle tecnologie più recenti per il tracing nei sistemi Linux è rappresentata dall'extended Berkeley Packet Filter (eBPF). eBPF permette di eseguire piccoli programmi all'interno del sistema operativo in modo controllato e sicuro, consentendo di intercettare eventi durante l'esecuzione senza modificare il codice del kernel o delle applicazioni.

Il funzionamento di eBPF si basa sull'inserimento dinamico di programmi che vengono eseguiti quando si verificano determinati eventi, come l'ingresso in una funzione del kernel o l'attivazione di un tracepoint. Prima di essere caricati, i programmi eBPF vengono verificati da un componente del sistema operativo chiamato verifier, che ne controlla la sicurezza e garantisce che non possano compromettere la stabilità del sistema. Questo meccanismo permette di eseguire codice in modo sicuro anche all'interno di parti critiche del sistema operativo.

Rispetto alle tecniche di tracing tradizionali, eBPF presenta alcune caratteristiche distintive. In primo luogo consente di raccogliere informazioni con un overhead

generalmente ridotto, poiché i programmi vengono eseguiti direttamente nel contesto dell'evento osservato. Inoltre, le sonde possono essere attivate e disattivate dinamicamente senza riavviare il sistema operativo. Un ulteriore vantaggio è la grande flessibilità, che permette di definire nuove sonde e nuovi tipi di analisi senza modificare il sistema.

Grazie a queste caratteristiche eBPF rappresenta oggi una tecnologia particolarmente adatta per il tracing dinamico dei sistemi Linux e costituisce la base di numerosi strumenti moderni di osservabilità.

2.3.3 bpftrace

Per utilizzare eBPF in modo pratico sono stati sviluppati diversi strumenti software. Tra questi, `bpftrace` rappresenta uno degli strumenti più diffusi per la realizzazione di script di tracing basati su eBPF.

`bpftrace` è un linguaggio di scripting e un interprete che permette di definire programmi di tracing in forma compatta e leggibile. Gli script vengono tradotti automaticamente in programmi eBPF ed eseguiti dal sistema operativo, semplificando notevolmente lo sviluppo rispetto alla scrittura diretta di codice eBPF.

Un programma `bpftrace` è costituito da una o più probe, cioè punti di osservazione associati a eventi specifici. Ogni probe è composta da due parti principali:

- la definizione dell'evento da osservare
- le azioni da eseguire quando l'evento si verifica

La struttura generale di una probe è la seguente:

```
probe
{
    azioni
}
```

Un esempio semplice è il seguente:

```
tracepoint:syscalls:sys_enter_openat
{
    printf("openat called by pid %d\n", pid);
}
```

In questo caso lo script stampa un messaggio ogni volta che un processo esegue la system call `openat`.

Tipi di probe in bpftrace

`bpftrace` supporta diversi tipi di probe, che permettono di osservare eventi a diversi livelli del sistema.

Tracepoint I tracepoint sono punti di osservazione statici definiti all'interno del sistema operativo. Essi rappresentano eventi predisposti per il tracing e forniscono informazioni strutturate sui parametri dell'evento osservato.

Un esempio di tracepoint è:

```
tracepoint:syscalls:sys_enter_execve
{
    printf("execve called\n");
}
```

I tracepoint sono generalmente il metodo più stabile per il tracing, perché la loro interfaccia tende a rimanere compatibile tra diverse versioni del sistema.

kprobe I kprobe permettono di inserire sonde dinamiche all'ingresso o all'uscita di funzioni del sistema operativo. Questo consente di osservare il comportamento interno del sistema anche in punti dove non esistono tracepoint predefiniti.

Un esempio di kprobe è:

```
kprobe:vfs_read
{
    printf("read operation\n");
}
```

Esistono anche le kretprobe, che permettono di osservare il valore restituito da una funzione:

```
kretprobe:vfs_read
{
    printf("return value: %d\n", retval);
}
```

I kprobe offrono grande flessibilità ma possono essere meno stabili dei tracepoint, perché dipendono dai nomi delle funzioni interne del sistema operativo.

uprobe Le uprobe permettono di inserire sonde all'interno di programmi in spazio utente. Questo consente di tracciare funzioni di librerie o applicazioni senza modificarne il codice.

Un esempio di uprobe è:

```
uprobe:/bin/bash:readline
{
    printf("readline called\n");
}
```

Sono disponibili anche le uretprobe, che permettono di osservare il valore di ritorno di una funzione:

```
uretprobe:/bin/bash:readline
{
    printf("function returned\n");
}
```

Le uprobe sono particolarmente utili quando si vuole osservare il comportamento interno di una specifica applicazione.

Struttura degli script bpftrace

Gli script `bpftrace` possono includere variabili, mappe associative e operazioni di aggregazione dei dati.

Le variabili temporanee sono indicate con il simbolo `$`, mentre le mappe globali sono indicate con il simbolo `@`.

Ad esempio:

```
tracepoint:syscalls:sys_enter_read
{
    @count[pid] = count();
}
```

Questo script conta il numero di chiamate `read` effettuate da ogni processo.

È inoltre possibile stampare i risultati al termine dell'esecuzione tramite una probe speciale:

```
END
{
    print(@count);
}
```

Questa probe viene eseguita automaticamente quando il programma termina.

Capitolo 3

Overview del programma

Capitolo 4

Test e Validazione

Capitolo 5

Limiti e possibili miglioramenti

Capitolo 6

Conclusioni

Bibliografia

[1] John Doe e John Smith. «Paper title». In: *Proceedings of the Big Conf*. Giu. 2017, pp. 185–200. DOI: https://doi.org/doi/reference/number.

[2] *Url title*. URL: the%20actual%20url (visitato il 05/2022).