



**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. Matteo

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

1	Introduzione	1
1.1	Contesto	1
1.2	Problema	2
1.3	Idea di Soluzione	3
2	Tecnologie di base	5
2.1	Linux come base per il tracing	5
2.2	Example	7
2.3	Example	7
3	Overview del programma	9
4	Test e Validazione	11
5	Limiti e possibili miglioramenti	13
6	Conclusioni	15
	Bibliografia	17

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 bpfttrace 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 bpfttrace 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 bpfttrace 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 bpfttrace e degli strumenti necessari al tracing. L'utilizzo di una distribuzione Linux separata ha permesso di evitare la compilazione nativa di bpfttrace 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

Il lavoro di tesi si colloca nel contesto dei sistemi operativi di tipo Unix-like, dove il kernel svolge il ruolo di componente centrale per la gestione delle risorse (processi, memoria, file system, rete) e fornisce alle applicazioni un'interfaccia standardizzata tramite le system call. In Linux, gran parte delle attività “interessanti” dal punto di vista osservativo (creazione di processi, accesso ai file, comunicazioni di rete, scheduling) attraversa inevitabilmente il kernel o viene mediata da esso, rendendo possibile analizzare il comportamento delle applicazioni osservando eventi a livello kernel o al confine user–kernel.

Per predisporre un ambiente di sviluppo riproducibile e vicino a un contesto reale, è stato utilizzato un sistema Ubuntu come base. In particolare, il root filesystem adottato è Ubuntu Base 22.04.4 LTS per architettura amd64, nella variante “ubuntu-base-22.04.4-base-amd64”. Ubuntu Base è una distribuzione minimale (rootfs) pensata come punto di partenza per immagini custom e ambienti vincolati o integrati; fornisce uno user-space funzionale e la possibilità di installare pacchetti aggiuntivi dai repository tramite apt. Nel presente lavoro non sono state introdotte modifiche particolari a questo ambiente, se non l'installazione dei componenti necessari alle attività di sperimentazione e tracing descritte nei capitoli successivi.

Con il termine “tracing” si intende la raccolta di informazioni (eventi) durante l’esecuzione di un sistema o di un’applicazione, con l’obiettivo di ricostruire “cosa è successo” e “quando”. A differenza del logging applicativo tradizionale, che dipende dal codice sorgente e dalle scelte dello sviluppatore, il tracing a livello di sistema può osservare l’esecuzione anche senza modificare l’applicazione, sfruttando punti di osservazione presenti nel kernel o meccanismi di ispezione dei processi.

Nel caso del tracing delle applicazioni, gli obiettivi principali sono tipicamente tre:

Debugging funzionale. Tracciare eventi come chiamate di sistema e segnali permette di capire perché un programma fallisce (errori di permessi, file mancanti, chiamate non attese) e di riprodurre il comportamento in modo deterministico a partire da una sequenza di eventi osservati.

Analisi delle prestazioni e latenza. Molti problemi prestazionali non derivano dalla logica applicativa in sé, ma dall’interazione con il sistema operativo (I/O, scheduling, lock, contesa di risorse). Strumenti di tracing e profiling consentono di misurare tempi, frequenze di eventi e contatori, collegando carichi di lavoro a fenomeni di sistema (ad esempio cambi di contesto, tempi di run-queue, utilizzo CPU).

Osservabilità e sicurezza. In ambito security e forensic, osservare attività come apertura di file, creazione di processi o comunicazioni di rete consente di identificare pattern anomali o potenzialmente malevoli. Anche quando l’obiettivo non è “bloccare” un comportamento, la tracciatura può fornire evidenze utili per classificare attività sospette e ricostruire catene causali.

In sintesi, il tracing è utile quando si vuole passare da una visione “black box” dell’applicazione (input/output) a una visione “glass box” dell’interazione tra applicazione e sistema operativo, riducendo l’incertezza nelle analisi e migliorando la riproducibilità dei risultati sperimentali.

2.2 Example

2.3 Example

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).