tinexta
defence

Usi e abusi di eBPF: minacce, tecniche di rilevamento e contromisure
parte 1

#TinextaDefenceBusiness

DFIR

Il Gruppo DFIR di Tinexta Defence è una Threat Response Unit specializzata in Digital Forensics & Incident Response che supporta imprese e pubbliche amministrazioni nella gestione di incidenti di sicurezza e nella produzione di evidenze digitali con valore probatorio.

L'attività del Gruppo integra competenze multidisciplinari e si articola in quattro aree principali:

- **Incident Response:** capacità di intervento rapido per contenere, eradicare e mitigare incidenti, riducendo l'impatto operativo;
- **Consulenze Tecniche d'Ufficio (CTU) e di Parte (CTP):** perizie informatiche conformi alle best practice di catena di custodia, a supporto del contesto giudiziario;
- **Forensic Readiness:** predisposizione preventiva di processi, tecnologie e standard per garantire che i dati raccolti siano accurati, integri e attestabili;
- **Ricerca e innovazione:** sperimentazione di tecnologie avanzate (eBPF, kernel telemetry, AI per anomaly detection, container forensics) per anticipare le minacce e sviluppare strumenti di nuova generazione.

In qualità di Threat Response Unit, il Gruppo DFIR non si limita alla fase di indagine post-evento, ma affianca i Security Operations Center (SOC) e le organizzazioni nella detection proattiva, nel threat hunting e nella gestione di crisi cyber.

La missione del Gruppo è elevare i livelli di sicurezza e resilienza delle infrastrutture critiche e dei sistemi informativi, coniugando rigore scientifico, innovazione tecnologica e capacità operativa a supporto della difesa digitale e del contesto giudiziario.

Sommario

Abstract

Le avanzate capacità di eBPF nell'estensione delle funzionalità del kernel Linux rendono questa tecnologia un'arma a doppio taglio nel mondo della sicurezza informatica.

Se da un lato è abilitante a soluzioni per osservabilità, networking avanzato e sicurezza runtime ad alto potenziale, dall'altro si sta dimostrando purtroppo uno strumento molto versatile per lo sviluppo di impianti malevoli con funzionalità tipiche dei rootkit.

Grazie a specifiche helper functions è possibile implementare funzionalità di offuscamento, manipolazione dei processi, manipolazione di attività di rete leggendo e modificando pacchetti e alterazione di programmi nello spazio utente.

Il presente lavoro si propone come primo di una serie di pubblicazioni volta a:

- esplorare le potenzialità malevole dei programmi eBPF;
- delineare metodologie di rilevamento e mitigazione;
- presentare tecniche e strumenti utili alle analisi condotte da gruppi DFIR e centri SOC per rilevare attività riconducibili a rootkit eBPF.

In particolare, in questo report viene condotta una panoramica teorica sui campi di applicazione in cui l'abuso di programmi eBPF può rappresentare un grande vantaggio per attori malevoli, analizzando studi condotti sia su ricerche accademiche che su malware realmente utilizzati.

Successivamente viene fornita una descrizione di approcci in-kernel e userland per il rilevamento degli impianti malevoli discussi, concludendo con un'esposizione di possibili contromisure adottabili al fine di prevenire la minaccia.

Autori:

- Marco Tinti: Team Leader DFIR
- Riccardo Luzi: Forensics Analyst
- Andrea Mura: Cyber Security Analyst

Introduzione

La tecnologia eBPF *(extended Berkeley Packet Filter)*[1] è emersa come uno strumento all'avanguardia per il monitoraggio e la protezione dei sistemi Linux da attacchi sempre nuovi e sofisticati.

Versatilità, privilegi di esecuzione e verifica del codice sono alcuni dei punti di forza che rendono eBPF un elemento estremamente utile per la raccolta dettagliata di informazioni in tempo reale, senza rischiare di compromettere la stabilità del kernel.

Tuttavia, sono proprio queste caratteristiche a rendere eBPF una tecnologia preziosa per la realizzazione di impianti malevoli, come i *rootkit*.

Un programma eBPF non può modificare arbitrariamente le informazioni del sistema su cui sta operando ma, grazie a delle specifiche helper functions, può:

- scrivere in aree di memoria dell'utente;
- specificare il valore di ritorno di una chiamata di sistema;
- scrivere in un pacchetto di rete;
- leggere da un pacchetto di rete;
- inviare un segnale a un processo.

Sebbene tali funzionalità siano state progettate per abilitare tecniche legittime di introspezione e protezione, esperimenti di ricerca e Proof-of-Concept hanno dimostrato come possano essere abusate per realizzare impianti persistenti e furtivi (rootkit) che ostacolano le attività di rilevamento e di analisi forense.

Sono disponibili approcci di mitigazione e tecniche di rilevamento specifiche per ridurre l'impatto dei rootkit basati su eBPF. Tali strumenti risultano particolarmente efficaci nel momento del caricamento dei programmi malevoli, fase in cui l'impianto non ha ancora alterato le strutture di sistema o interferito con gli strumenti di analisi.

Una volta caricati, individuare questi impianti risulta significativamente più complesso, poiché possono alterare o nascondere informazioni cruciali per le attività di analisi forense, come l'elenco dei processi attivi, le connessioni di rete o persino il contenuto dei file di log.

Per questo motivo, il presente lavoro si propone come primo di una serie sul tema degli impianti malevoli basati su questa tecnologia e su quali metodi possano permetterne il rilevamento in un sistema basato su Linux. Le analisi fanno riferimento a versioni del kernel Linux 5.15+ (serie LTS) su architettura x86_64.

Gli obiettivi che ci siamo prefissi per questo lavoro sono i seguenti:

- fornire una panoramica tecnica degli abusi noti e delle aree di rischio;
- descrivere strumenti e contromisure pratiche per la mitigazione e il monitoraggio;
- proporre approcci investigativi e tecniche di correlazione utili ai gruppi DFIR e ai SOC per identificare attività riconducibili a rootkit eBPF.

Nei capitoli successivi verranno analizzati esempi pratici, strumenti di rilevamento (in-kernel e userland), e raccomandazioni operative per integrare queste precauzioni nei processi di sicurezza e governance aziendali.

Possibili abusi di eBPF

Le motivazioni che portano a un abuso di eBPF per lo sviluppo di impianti malevoli risiedono principalmente nelle sue estese potenzialità e nella possibilità di eseguire codice al livello del kernel, senza rischiare di compromettere il sistema.

Grazie alle varie tipologie di programmi eBPF esistenti è possibile operare sia su pacchetti di rete (a diversi livelli dello stack TCP/IP) che su varie funzioni, siano esse chiamate di sistema, funzioni interne al kernel o funzioni di librerie nello spazio utente.

Inoltre, la presenza dell'*eBPF Verifier* e della *eBPF Virtual Machine* permettono ad agenti malevoli di eseguire impianti con privilegi elevati e capacità aumentate, senza rischiare di comprometterlo o causarne interruzioni a causa di problemi di compatibilità o bug.

Allo stato dell'arte possiamo vedere vari esempi di abusi, sia in campo di ricerca che in malware realmente utilizzati che sfruttano eBPF in diverse aree di applicazione.

Offuscamento

Una delle caratteristiche principali che rendono eBPF uno strumento tanto potente è la capacità di poter osservare e interagire con i dati utilizzati da molte funzioni del sistema, siano esse chiamate di sistema (utilizzabili quindi da programmi nello spazio utente) o interne al kernel e destinate al solo utilizzo da parte di esso.

Queste capacità permettono di realizzare soluzioni ottime per aumentare l'osservabilità di un sistema, ma anche di realizzare impianti capaci di manipolare la realtà resa visibile allo spazio utente.

I programmi eseguiti nello spazio utente devono fare uso delle chiamate di sistema per molte delle operazioni che compiono ordinariamente (accedere a un file, eseguire un comando ecc.), e ciò riguarda anche gli strumenti di analisi utilizzati nello spazio utente per rilevare anomalie e potenziali minacce.

Quindi, un impianto in grado di modificare i dati ottenuti utilizzando una chiamata di sistema o di modificare direttamente l'esito di una sua invocazione può compromettere completamente l'analisi effettuata.

In tal senso è stato dimostrato che è possibile:

- nascondere la presenza di specifici processi andando ad alterare i dati ottenuti da strumenti di enumerazione come **ps**;
- nascondere la presenza di specifici programmi eBPF e mappe eBPF andando ad alterare i dati ottenuti da strumenti che usano la chiamata di sistema **bpf** per visualizzare queste informazioni, come ad esempio **bpftool**[14];
- nascondere la presenza di file e cartelle andando ad alterare i dati ottenuti da strumenti come **ls** o bloccando l'accesso diretto a essi, alterando il risultato della chiamata di sistema utilizzata (ovvero, restituendo un errore per indicare che la cartella o il file non sono stati trovati);
- modificare tutte le occorrenze di una stringa all'interno di un file nel momento in cui questo viene letto, cambiando solo la percezione del suo contenuto e non i dati effettivamente presenti al suo interno. Questo può essere usato, ad esempio, per nascondere la presenza di un modulo del kernel cambiandone il nome o nascondere l'esistenza di specifici messaggi di log;
- intercettare le chiamate di sistema per la scrittura di dati su disco prendendo di mira specifici processi, impedendo quindi al processo *rsyslogd* di scrivere sul file di log */var/log/auth.log* così da non lasciare tracce riguardanti accessi via SSH.

Manipolazione dei processi

Manipolare l'esito di una chiamata di sistema o i dati processati e restituiti da essa permette di ottenere altri risultati oltre all'offuscamento di artefatti che possono rivelare la presenza di un programma malevolo.

Prendendo come bersaglio il processo *sudo* (creato dall'omonimo comando per eseguire operazioni con privilegi elevati) è possibile fornire privilegi più alti a un utente che non ne dispone, senza che ciò possa essere rilevato utilizzando i comuni strumenti di lettura dei file presenti sul sistema.

In particolare, è possibile intercettare la lettura del file */etc/sudoers* per sovrascriverne la prima riga con *<nome utente> ALL=(ALL:ALL) NOPASSWD:ALL #* andando così a garantire privilegi maggiori all'utente indicato, senza che questo debba inserire una password all'utilizzo del comando *sudo*[3]. In questo caso è importante notare che influenzando solo la lettura da parte di *sudo*, la lettura del file con altri strumenti come *cat* e *sudoedit* non riporterebbe questa modifica.

Unendo questa tecnica a una precisa modifica dei parametri passati alla chiamata di sistema *execve* è possibile eseguire programmi arbitrari con permessi elevati senza che l'utente riesca a notarlo.

La chiamata di sistema *execve* viene utilizzata per eseguire un programma specificando il suo percorso. Intercettando la sua invocazione e sostituendo il percorso con quello di un altro programma (in questo caso, malevolo) è possibile eseguirlo al posto di quello desiderato. Inoltre, raffinando la tecnica, è possibile anche realizzare il programma malevolo in modo che abbia le informazioni necessarie per eseguire comunque il programma desiderato, così che l'utente non sia in grado di accorgersene.

Se invece non fosse possibile eseguire nuovi programmi, è stato dimostrato che a partire dagli indirizzi di memoria ottenuti dagli argomenti passati a una chiamata di sistema, si è in grado di attuare tecniche di process injection prendendo di mira la memoria del processo nello spazio utente che ha invocato la chiamata.

Senza modificare i dati provenienti dallo spazio utente o diretti verso di esso è comunque possibile bloccare l'esecuzione di una chiamata di sistema o anche inviare un segnale di terminazione verso un processo che utilizza una specifica chiamata di sistema. Questo permette, ad esempio, di bloccare chiamate come *kill* o *ptrace* quando usate verso un processo di interesse per l'attore malevolo o direttamente terminare il processo che sta cercando di invocarle.

Manipolazione della rete

L'interazione con chiamate di sistema e processi in esecuzione non è l'unico aspetto a rendere eBPF uno strumento appetibile per la produzione di impianti malevoli.

Un altro aspetto fondamentale riguarda le sue capacità di operare con i pacchetti di rete a diversi livelli dello stack TCP/IP. A partire dall'interfaccia di rete stessa, in cui l'interazione avviene prima che i pacchetti vengano processati dal kernel, è possibile scartare, accettare, modificare e ritrasmettere i pacchetti, sia in ingresso che in uscita.

Un uso comune di queste capacità, osservato sia in letteratura che in esempi di malware realmente utilizzati, è di intercettare i pacchetti prima del processamento effettuato dal kernel per identificate la presenza di caratteristiche specifiche che identificano l'innesco progettato per l'attivazione di alcune capacità dell'impianto malevolo e della comunicazione C2[6] [7].

Ad esempio, il rootkit *boopkit*[4] è stato realizzato per dimostrare che è possibile eseguire comandi su una macchina remota utilizzando un solo pacchetto SYN, opportunamente intercettato da un impianto malevolo basato su eBPF.

Il client di questo sistema utilizza due tecniche:

- invia un pacchetto TCP SYN con un checksum malformato che viene riconosciuto dal rootkit indipendentemente dal tipo di servizio TCP in esecuzione sul server compromesso;
- se la scheda di rete della macchina scarta il pacchetto con checksum malformato (cosa che può accadere in schede più recenti), il client completa un handshake TCP con un qualsiasi servizio TCP in ascolto.

Successivamente, il client instaura una nuova connessione TCP in cui inverte il valore del flag RST nel pacchetto SYN, causando così un reset da parte del servizio in ascolto ed innescando il rootkit.

Nonostante le grandi capacità di questa tecnologia, un programma eBPF non può creare nuovi pacchetti, ma può sfruttare quelli creati da altri processi.

Per attuare un'esfiltrazione di dati e comunicare con un server C2 remoto, un impianto malevolo può sfruttare la **ritrasmissione dei pacchetti TCP**[2]: nel protocollo TCP, un pacchetto che non arriva a destinazione viene ritrasmesso. Questo avviene se il mittente non riceve un nuovo ACK entro il timeout avviato all'invio del pacchetto o se riceve tre ACK duplicati entro lo scadere del timeout.

Questa caratteristica del protocollo TCP può essere sfruttata per esfiltrare dati senza causare evidenti cambiamenti nel servizio. In particolare, è stato dimostrato che si possono realizzare programmi eBPF che modificano un pacchetto TCP in uscita alterandone la destinazione ed il contenuto, senza interrompere la comunicazione originale, grazie alla successiva ritrasmissione del pacchetto modificato.

Un altro uso dei pacchetti esistenti è quello di modificare i pacchetti inviati dal server C2 con l'obiettivo di eseguire scansione attiva della rete in cui si trova la macchina compromessa[5]. Modificando i pacchetti inviati dal C2 per renderli pacchetti ARP o pacchetti SYN è possibile individuare quali sono le macchine raggiungibili da quella infetta, anche se queste non comunicano attivamente tra loro.

Se invece la scansione attiva non è una tecnica impiegabile nel contesto specifico in cui viene utilizzato l'impianto malevolo, è comunque possibile osservare passivamente i pacchetti in ingresso con vari programmi eBPF per collezionare informazioni sull'ambiente circostante, limitandosi però alle sole macchine che comunicano attivamente con quella d'interesse.

Per nascondere le tracce dei pacchetti utilizzati come innesco e delle connessioni sospette è stato osservato l'utilizzo di *hijacking* di eventuali processi di filtro dei pacchetti presenti nel sistema[9]. Anteponendo il bytecode eBPF malevolo a quello utilizzato dal programma di filtro nel momento del caricamento è possibile scartare specifici pacchetti prima che questi raggiungano il bytecode del programma, così da impedire che questo possa rilevarli.

Manipolazione dei programmi nello spazio utente

Finora abbiamo visto le potenzialità malevole dei programmi eBPF per operare su funzioni del kernel, chiamate di sistema, processi in esecuzione e connessioni di rete.

Un ulteriore campo di applicazione riguarda la possibilità di agganciare programmi eBPF a funzioni specifiche dei programmi che si trovano nello spazio utente.

Ad esempio, conoscendo la versione della libreria OpenSSL utilizzata nel sistema bersaglio è possibile intercettare con precisione i dati trasmessi nel momento in cui vengono cifrati e decifrati dalla libreria, così da poter vedere in chiaro il contenuto di connessioni cifrate.

Utilizzi di questo tipo di programmi eBPF individuati allo stato dell'arte comprendono:

- cattura delle credenziali in chiaro durante il processo di autenticazione di applicazioni che utilizzano la libreria Pluggable Authentication Modules (PAM)[8];
- bypass di tecnologia RASP (Runtime Application Self-Protection), utilizzata ad esempio per osservare specifiche funzioni di librerie SQL o di server HTTP, al fine di verificare che gli input vengano sanificati correttamente, così da prevenire minacce come *SQL Injection*. In particolare, è possibile modificare dei dati malevoli passati in input subito prima del controllo effettuato dal RASP per farli sembrare innocui, per poi riportarli allo stato originale subito prima che la query venga effettivamente eseguita;
- implementare meccanismi di accesso persistente a un applicativo (come un server PostgreSQL) andando a intercettare la funzione di verifica delle credenziali per poter garantire l'accesso a utenti non autorizzati;
- iniettare comandi all'interno di un container Docker nel momento in cui un processo *bash* interno a esso sta leggendo da una *pipe*;
- cambiare immagini Docker a runtime intercettando la funzione *ParseNormalizedName*.

Rilevamento di abusi

Il modo migliore per provare a rilevare la presenza di impianti malevoli basati su eBPF è quello di monitorare l'invocazione della chiamata di sistema **bpf()** utilizzata per il loro caricamento.

In questo modo è possibile generare allarmi di sicurezza quando vengono caricati programmi eBPF inattesi, così da attuare prontamente le dovute contromisure.

Un aspetto chiave nel rilevamento di impianti eBPF malevoli è il livello di osservazione adottato dallo strumento di monitoraggio, distinguibile in due categorie:

- *In-kernel monitoring*: soluzioni come Falco[10] e Tracee[11] utilizzano eBPF stesso per intercettare eventi direttamente dal kernel, garantendo visibilità e accuratezza elevate con minimo overhead. Tuttavia, un attacco in grado di compromettere il kernel può manipolare anche questi meccanismi di controllo;
- *Userland monitoring*: strumenti come *auditd*[15], *perf*[16] o framework basati su *ptrace* operano nello spazio utente e non sono immuni a un kernel compromesso, ma possono fornire dati di confronto o di validazione rispetto agli eventi osservati in kernel-space seppur con minore granularità e un impatto prestazionale più alto.

La combinazione dei due approcci – in-kernel per la tempestività e userland per la validazione incrociata – consente di ridurre la probabilità che un rootkit eBPF eluda completamente il monitoraggio.

Inoltre, Tracee fornisce anche un evento chiamato **bpf_attach** che si avvale di diversi hooks eBPF per fornire maggiori informazioni di contesto riguardo il programma caricato, tra cui: nome, tipo, tipo di probe e funzioni helper utilizzate.

Nel caso in cui non sia stato opportunamente configurato un sistema di monitoraggio per il rilevamento di questo tipo di minacce, la situazione diventa più complessa.

Come illustrato nel capitolo precedente, un impianto malevolo può utilizzare programmi eBPF per distorcere la realtà del sistema per come viene vista dagli strumenti utilizzati nello spazio utente.

Quindi, una volta che un programma eBPF progettato per nascondere delle informazioni è stato caricato, rilevarne la presenza sulla macchina non è un compito semplice.

Il primo strumento utile all'analisi dei vari elementi che fanno parte dell'ecosistema eBPF presenti in un sistema è **bpftool**, che tramite la chiamata di sistema bpf() consente di ottenere varie informazioni, tra cui dettagli riguardanti *programmi*, *link* e *mappe* presenti sulla macchina, fornendo anche la possibilità di formattare il risultato in json per facilitarne eventuali processamenti. Dato l'id di un programma eBPF precedentemente caricato è possibile scaricare il bytecode per visualizzarne il contenuto e verificare se sono presenti funzioni helper potenzialmente pericolose.

Oltre a bpftool, altre verifiche manuali possono includere:

- identificare la presenza di eventuali *kprobes* inattesi controllando il file */sys/kernel/debug/kprobes/list*;
- identificare la presenza di programmi XDP e TC inattesi con i comandi *ip link show* e *tc filter show*;
- ispezionare il **filesystem virtuale** di eBPF (o **bpffs**). Tradizionalmente montato al percorso */sys/fs/bpf*, questo contiene riferimenti a diversi oggetti eBPF – come programmi o mappe – che sono stati "appuntati" al loro caricamento. Tale pratica, nota anche come *pinning*[12], è utile perché altrimenti gli oggetti vengono mantenuti nel kernel finché esiste almeno un riferimento a quell'oggetto nel sistema. Se, ad esempio, il programma che utilizza una mappa terminasse, non esisterebbero più riferimenti a quella mappa e di conseguenza verrebbe cancellata; tuttavia, appuntandola continuerebbe a esistere ed essere utilizzabile finché non esplicitamente rimossa o fino allo spegnimento del sistema;
- controllare i log di sistema per messaggi relativi all'utilizzo di chiamate di sistema come bpf() o funzioni helper specifiche. Funzioni come **bpf_probe_write_user** (fondamentale per molte delle operazioni descritte nel capitolo precedente) causano la generazione di un messaggio di log specifico quando un programma che le usa viene caricato.

Vista la potenziale pericolosità di impianti malevoli basati su eBPF, sono stati studiati degli strumenti per automatizzare l'analisi di una macchina potenzialmente infetta. Unendo le tecniche appena descritte ad altre tecniche di analisi di artefatti legati sia a rootkit generici che a quelli basati su eBPF, le possibilità di rilevare la presenza di queste minacce aumentano.

Questi strumenti permettono di automatizzare il processo di indagine sfruttando le funzionalità messe a disposizione da bpftool e inserendo capacità di analisi di altri artefatti per ricercare possibili discrepanze tra lo stesso tipo di informazioni raccolte da diverse fonti, così da poter identificare una loro possibile manomissione.

Come detto in precedenza, queste metodologie di rilevamento possono soffrire di eventuali tecniche di offuscamento utilizzate da un impianto malevolo una volta attivato.

Per affrontare questo problema, in un recente studio[17] è stato presentato un sistema composto da quattro componenti:

- uno strumento basato su un hypervisor per l'acquisizione di un'immagine di memoria della macchina in esecuzione;
- un framework per l'analisi dell'immagine acquisita;
- uno strumento per l'estrazione di programmi eBPF;
- un classificatore di immagini forensi.

Questo sistema permette di analizzare una macchina potenzialmente infetta senza dover eseguire strumenti su di essa, né doverne interrompere l'esecuzione.

Una volta acquisita l'immagine, l'analisi viene condotta utilizzando il framework *Volatility* (specializzato per l'analisi forense della memoria) unito a un plugin opportunamente progettato e realizzato per individuare ed estrarre codice e informazioni utili (es. nome, lunghezza, tipo) riguardanti eventuali programmi eBPF precedentemente caricati sulla macchina.

Successivamente, nel codice disassemblato si procede a identificare quali funzioni helper vengono utilizzate da ciascun programma eBPF individuato, segnalando la presenza di eventuali funzioni considerate sospette. Le funzioni helper sospette sono quelle utilizzate dai rootkit open source per implementare funzionalità malevole:

- **probe_write_user:** utilizzata per alterare i dati restituiti da una chiamata di sistema;
- **override_return:** utilizzata per falsificare un errore o un successo sostituendo il codice di ritorno di una funzione;
- **skb_store_bytes** e **skb_pull_data:** utilizzate per leggere e modificare dati di un pacchetto di rete, permettendo di implementare canali nascosti per l'esfiltrazione e la comunicazione con un server C2;
- **send_signal:** utilizzata per terminare l'esecuzione di un altro processo.

Infine, per cercare di contrastare il problema di impianti che abusano di eBPF, in un altro studio[18] è stata proposta una soluzione che si integra con l'*hypervisor* di una macchina virtuale al fine di implementare un meccanismo di validazione dei programmi eBPF durante il loro caricamento.

Per utilizzarla è necessario modificare opportunamente il kernel del sistema operativo *guest*, in modo da sostituire il codice responsabile della verifica e della compilazione "Just-In-Time" dei programmi eBPF con delle chiamate a questo sistema di validazione.

Una volta apportate le modifiche, tutti i programmi eBPF caricati nel *guest* vengono sottoposti al processo di verifica implementato da questo sistema che comprende:

- controlli effettuati dal Verifier eBPF presente nel kernel Linux (es. validità delle istruzioni, limite di accesso alla memoria);
- controlli personalizzabili per limitare le istruzioni e le funzioni helper utilizzabili, il tipo di programmi e il tipo dei possibili hooks;
- controllo del codice per identificare eventuali parti malevole sulla base di un database di impianti malevoli noti.

Contromisure

La contromisura più drastica, ma anche la più efficace, per proteggere un sistema da potenziali abusi di eBPF consiste nel disabilitarne completamente il supporto a livello di kernel. Questa misura è raccomandabile solo in contesti ad alta sicurezza o in ambienti air-gapped, dove non sono utilizzati strumenti dipendenti da eBPF (come osservabilità, networking avanzato o sicurezza runtime).

Per ottenere tale risultato è necessario ricompilare il kernel Linux disabilitando esplicitamente le opzioni di *kconfig* correlate al supporto BPF. Le principali sono:

- **CONFIG_BPF:** abilita il supporto base per BPF;
- **CONFIG_BPF_SYSCALL:** abilita la chiamata di sistema bpf();
- **CONFIG_BPF_JIT:** abilita compilatore JIT per BPF;
- **CONFIG_BPF_JIT_ALWAYS_ON:** se presente, abilita permanentemente BPF JIT e rimuove l'interprete BPF;
- **BPF_UNPRIV_DEFAULT_OFF:** disabilita l'utilizzo non privilegiato di BPF;
- **CONFIG_BPF_EVENTS:** consente agli utenti di attaccare programmi BPF a kprobes, uprobes e tracepoints;
- **CONFIG_NET_ACT_BPF:** abilita l'esecuzione di codice BPF sui pacchetti di rete;
- **CONFIG_NET_CLS_BPF:** abilita la classificazione di pacchetti di rete tramite programmi BPF;
- **CONFIG_CGROUP_BPF:** permette di attaccare programmi eBPF su un *cgroup*;
- **CONFIG_KPROBES:** abilita l'utilizzo dei kprobes;
- **CONFIG_BPF_KPROBE_OVERRIDE:** consente a BPF di sovrascrivere l'esecuzione di una funzione sondata e impostarne un valore di ritorno diverso. Viene utilizzato per l'iniezione di errori. Risulta fondamentale disattivare questa opzione se non strettamente necessaria.

Tuttavia, una disabilitazione totale del supporto eBPF rappresenta una misura estremamente vincolante e, nella pratica, difficilmente applicabile: molti sistemi e piattaforme moderne – tra cui strumenti di osservabilità, networking e sicurezza runtime – fanno ampio uso di questa tecnologia. In tali contesti è quindi preferibile adottare un approccio di *hardening* selettivo, intervenendo unicamente sulle funzionalità non necessarie e sui profili di privilegio associati.

Hardening selettivo del supporto a eBPF

Un'alternativa consiste in un'azione più mirata alla disabilitazione delle sole funzionalità che non sono necessarie. Una delle misure più efficaci è limitare l'utilizzo non privilegiato di eBPF, configurando il parametro del kernel *kernel.unprivileged_bpf_disabled*[13] con uno dei seguenti valori:

- "1" disabilita il caricamento di programmi eBPF da parte di utenti non privilegiati, impedendone la riabilitazione fino al prossimo riavvio del sistema;
- "2" disabilita il caricamento di programmi eBPF da parte di utenti non privilegiati, consentendone la riabilitazione direttamente a runtime, senza che il sistema necessiti di riavvio.

Tale parametro puo' essere impostato con il seguente comando:

sudo sysctl kernel.unprivileged_bpf_disabled=<valore>

L'avvenuta applicazione puo' essere verificata eseguendo:

sudo cat /proc/sys/kernel/unprivileged_bpf_disabled

Per rendere persistente tale configurazione, ad esempio dopo un riavvio, è necessario aggiungere la seguente riga al file */etc/sysctl.conf*:

kernel.unprivileged_bpf_disabled=<valore>

Protezione delle aree sensibili del filesystem

Oltre alla disattivazione delle funzionalità che non vengono utilizzate è opportuno assicurare che parti critiche del filesystem come */sys/fs/bpf* e */sys/kernel/tracing* siano adeguatamente configurate per permettere l'accesso solo a utenti e processi fidati. Tali cartelle contengono riferimenti a oggetti eBPF "pinnati" nel kernel e un accesso non controllato potrebbe consentire a un attore malevolo di alterare o sostituire elementi già caricati.

Monitoraggio della chiamata di sistema bpf()

Nel caso in cui non sia possibile disabilitare completamente o parzialmente le funzionalità che consentono l'utilizzo di eBPF, è opportuno monitorare la chiamata di sistema bpf() adottando soluzioni basate su eBPF – come Tracee e Falco – per generare alert di sicurezza quando questa viene invocata, o predisporre strumenti come AppArmor[19] per limitarne l'utilizzo.

Restrizione delle capabilities dei processi

Un'ulteriore contromisura consiste nel restringere i privilegi dei processi in esecuzione assegnando solo le *capabilities* strettamente necessarie, specialmente in contesti in cui si usano ambienti containerizzati dove è più complesso tenere sotto controllo la postura di sicurezza. L'impiego di sistemi di controllo come AppArmor o SELinux[20] consente di definire politiche granulari per limitare l'uso della chiamata di sistema bpf() ai soli processi autorizzati, mitigando il rischio di bytecode eBPF malevolo.

Conclusioni e sviluppi futuri

Le qualità che rendono eBPF una valida tecnologia per realizzare strumenti capaci di fronteggiare le nuove minacce informatiche sono le stesse che potrebbero renderla sempre più presente nei malware con capacità avanzate.

Sebbene questi rootkit non abbiano effetto sui sistemi in cui eBPF è disabilitato, l'adozione sempre più diffusa di strumenti basati su tale tecnologia (per il monitoraggio, il networking e la protezione runtime) amplia progressivamente la superficie d'attacco disponibile. Ciò implica che la minaccia non è più teorica, ma concretamente applicabile in ambienti operativi moderni, soprattutto in contesti containerizzati e multi-tenant.

Diventa quindi essenziale iniziare a comprendere come l'utilizzo di eBPF possa contribuire alle varie fasi di un attacco e come possa essere sfruttato per contrastare le operazioni di analisi condotte su una macchina infetta. Questa duplice natura richiede un cambio di paradigma: dalla semplice mitigazione alla gestione consapevole e controllata dell'uso di eBPF all'interno delle infrastrutture critiche.

Solo attraverso la consapevolezza del rischio e di come poterlo fronteggiare dispiegando gli strumenti di sicurezza più efficaci risulta possibile adottare le contromisure più adeguate a ogni contesto, potendo così fare affidamento su analisi post incidente più mirate grazie a una contezza degli artefatti legati alla presenza di impianti dipendenti da questa tecnologia.

Nelle future pubblicazioni di questa serie verranno affrontate particolari metodologie di rilevamento e di analisi che siano in grado di fornire un aiuto nell'individuazione di attività potenzialmente riconducibili alla presenza di rootkit eBPF all'interno di un sistema.

Riferimenti

[1] https://ebpf.io/

[2] https://github.com/h3xduck/TripleCross

[3] https://github.com/pathtofile/bad-bpf

[4] https://github.com/krisnova/boopkit

[5] https://github.com/Gui774ume/ebpfkit

[6] https://www.synacktiv.com/en/publications/linkpro-ebpf-rootkit-analysis

[7] https://sandflysecurity.com/blog/bpfdoor-an-evasive-linux-backdoor-technical-analysis

[8] https://github.com/citronneur/pamspy

[9] https://blogs.blackberry.com/en/2022/06/symbiote-a-new-nearly-impossible-to-detect-linux-threat

[10] https://github.com/falcosecurity/falco

[11] https://github.com/aquasecurity/tracee

[12] https://docs.ebpf.io/linux/concepts/pinning/

[13] https://support.scc.suse.com/s/kb/Security-Hardening-Use-of-eBPF-by-unprivileged-users-has-been-disabled-by-default?language=en_US

[14] https://github.com/libbpf/bpftool

[15] https://man7.org/linux/man-pages/man8/auditd.8.html

[16] https://perfwiki.github.io/main/

[17] Nezer Zaidenberg, Michael Kiperberg, Eliav Menachi, and Asaf Eitani. 2024. Detecting eBPF Rootkits Using Virtualization and Memory Forensics. In Proceedings of the 10th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP. INSTICC, SciTePress, 254 - 261.

[18] Y. Wang, D. Li, and L. Chen, "Seeing the invisible: Auditing ebpf programs in hypervisor with hyperbee," in Proceedings of the 1st Workshop on EBPF and Kernel Extensions (eBPF '23), 2023.

[19] https://www.apparmor.net/

[20] https://github.com/selinuxproject

tinexta
defence

Next | Donexit | Foramil | Innodesi

Via Giacomo Peroni, 452 – 00131 Roma
tel. 06.45752720 – info@defencetech.it
www.tinextadefence.it

#TinextaDefenceBusiness