SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-5382-111266

VYUŽITIE BPF NA ZABEZPEČENIE OS LINUX

BAKALÁRSKA PRÁCA

2023 **Lukáš Grúlik**

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-5382-111266

VYUŽITIE BPF NA ZABEZPEČENIE OS LINUX

BAKALÁRSKA PRÁCA

Študijný program:	Aplikovaná informatika
Názov študijného odboru:	Informatika
Školiace pracovisko:	Ústav informatiky a matematiky
Vedúci záverečnej práce:	Ing. Roderik Ploszek
Konzultant:	Ing. Roderik Ploszek

Bratislava 2023 **Lukáš Grúlik**

SÚHRN

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný program:	Aplikovaná informatika
Autor:	Lukáš Grúlik
Bakalárska práca:	Využitie BPF na zabezpečenie OS Linux
Vedúci záverečnej práce:	Ing. Roderik Ploszek
Konzultant:	Ing. Roderik Ploszek
Miesto a rok predloženia práce:	Bratislava 2023

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean et est a dui semper facilisis. Pellentesque placerat elit a nunc. Nullam tortor odio, rutrum quis, egestas ut, posuere sed, felis. Vestibulum placerat feugiat nisl. Suspendisse lacinia, odio non feugiat vestibulum, sem erat blandit metus, ac nonummy magna odio pharetra felis. Vivamus vehicula velit non metus faucibus auctor. Nam sed augue. Donec orci. Cras eget diam et dolor dapibus sollicitudin. In lacinia, tellus vitae laoreet ultrices, lectus ligula dictum dui, eget condimentum velit dui vitae ante. Nulla nonummy augue nec pede. Pellentesque ut nulla. Donec at libero. Pellentesque at nisl ac nisi fermentum viverra. Praesent odio. Phasellus tincidunt diam ut ipsum. Donec eget est. A skúška mäkčeňov a dĺžnov.

Kľúčové slová: eBPF, Linux, bezpečnosť

ABSTRACT

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY

Study Programme:	Applied Informatics
Author:	Lukáš Grúlik
Bachelor's thesis:	Use of BPF for Linux OS security
Supervisor:	Ing. Roderik Ploszek
Consultant:	Ing. Roderik Ploszek
Place and year of submission:	Bratislava 2023

On the other hand, we denounce with righteous indignation and dislike men who are so beguiled and demoralized by the charms of pleasure of the moment, so blinded by desire, that they cannot foresee the pain and trouble that are bound to ensue; and equal blame belongs to those who fail in their duty through weakness of will, which is the same as saying through shrinking from toil and pain. These cases are perfectly simple and easy to distinguish. In a free hour, when our power of choice is untrammelled and when nothing prevents our being able to do what we like best, every pleasure is to be welcomed and every pain avoided. But in certain circumstances and owing to the claims of duty or the obligations of business it will frequently occur that pleasures have to be repudiated and annoyances accepted. The wise man therefore always holds in these matters to this principle of selection: he rejects pleasures to secure other greater pleasures, or else he endures pains to avoid worse pains.

Keywords: eBPF, Linux, security

Poďakovanie

I would like to express a gratitude to my thesis supervisor.

Obsah

Úvod **1**

1 Teoretické základy eBPF **2**

1.1 História eBPF 2

1.2 Princípy eBPF 3

1.2.1 Ako funguje eBPF 3

1.2.2 Vstupné body pre eBPF 4

1.2.3 Ako sa tvoria programy eBPF 5

1.3 Architektúra eBPF 5

1.3.1 Kompilátor JIT 5

1.3.2 eBPF Maps 6

1.3.3 eBPF Helpers 6

1.4 eBPF nástroje 6

1.4.1 Kompilátory 7

1.4.2 Debuggery 7

1.5 Knižnice eBPF 8

1.5.1 Knižnica libbpf 8

1.5.2 Knižnica BCC 8

1.5.3 Knižnica eBPF Go 8

1.6 Použitie eBPF 9

1.6.1 Bezpečnosť a eBPF 10

1.6.2 Seccomp 11

1.6.3 LSM 12

1.6.4 KRSI 13

2 Aplikácie eBPF **14**

2.1 Detekcia útokov 14

3 Porovnanie eBPF s inými nástrojmi **15**

4 Implementácia pomocou eBPF **16**

5 Výsledky implemetácie **17**

Záver **18**

Zoznam použitej literatúry **19**

Úvod

Tu bude krasny uvod s diakritikou atd.
A mozno aj viac riadkovy uvod.

1 Teoretické základy eBPF

Táto kapitola je vypracovaná podľa internetových stránok ebpf.io, kernel.org, oracle.com, sysdig.com a knihy od spisovateľky Liz Rice [1, 2, 3, 4, 5, 6]

V súčasnej dobe sa operačné systémy stávajú čoraz komplexnejšími, s rastúcim počtom aplikácií a služieb, ktoré sa na týchto systémoch spúšťajú, rastie aj potreba zabezpečenia pred rôznymi bezpečnostnými rizikami a hrozbami. Jedným z nástrojov, ktorý sa používa na zlepšenie bezpečnosti v operačných systémoch Linux, je eBPF (*extended Berkeley Packet Filter*). Je to flexibilný a mocný nástroj, ktorý sa čoraz častejšie používa na rôzne účely v operačných systémoch, ako je napríklad filtrovanie sieťovej aktivity, optimalizáciu výkonu operačného systému, alebo detegovanie útokov a škodlivého kódu. Taktiež je čoraz častejšie používaný pre implementáciu rôznych nástrojov slúžiacich na analýzu dát a monitorovanie systémov, ako napríklad nástroj *perf*, ktorý umožňuje profilovanie výkonu aplikácií v operačnom systéme. Pre maximálny výkon a bezpečnosť systému je dôležité, aby bol eBPF správne nakonfigurovaný a používaný pretože môže spôsobiť veľké bezpečnostné riziká pri zlom zaobchádzaní.

1.1 História eBPF

História eBPF sa začína s technológiou nazývanou *Berkeley Packet Filter* v skratke BPF, ktorá vznikla 19. Decembra 1992 kedy ju opísali Steven McCanne a Vano Jacobson. [7] BPF bol navrhnutý ako jednoduchý jazyk pre filtrovanie sieťových paketov, ktorý bol implementovaný ako rozšírenie jadra operačného systému. Jeho hlavným cieľom bolo umožniť používateľom filtrovať sieťové pakety bez nutnosti používať externé nástroje ako napríklad `tcpdump`. BPF bol úspešne implementovaný v operačných systémoch ako *BSD* a Linux, a stal sa jedným z kľúčových nástrojov pre sieťovú diagnostiku a analýzu. Avšak, s rastúcimi požiadavkami na funkcionalitu a výkon, bolo potrebné rozšíriť BPF o ďalšie možnosti. Začiatkom 21. storočia sa začal vývoj zameraný na vylepšenie technológie BPF. Nová verzia nazvaná eBPF (*extended Berkeley Packet Filter*) vznikla ako rozšírenie k BPF, ktoré bolo navrhnuté tak, aby poskytlo viac funkcií a umožňovalo vykonávať komplexnejšie filtračné skripty na sieťových paketoch. Tento nástroj bol prebratý Linuxovou komunitou, kde sa stal veľmi populárnym pre rôzne účely a v roku 2014 bol implementovaný do jadra Linuxu. Medzi hlavné rozdiely BPF a eBPF patrí podpora pre *x86* a *arm* architektúry, možnosť spustiť aplikácie v jadre operačného systému a možnosť vykonávať viac operácií ako len filtrovanie sieťových paketov. V súčasnosti eBPF umožňuje používateľom načítať a spustiť vlastné programy vo virtuálnom priestore v rámci jadra operačného systému. To

znamená, že môže rozšíriť alebo dokonca upraviť spôsob, akým sa jadro správa bez zmeny zdrojového kódu jadra. [5] Programy spustené týmto spôsobom sú schopné monitorovať systém, zhromažďovať metriky, a dokonca aj vykonávať rôzne úlohy ako napríklad sledovanie a upravovanie sieťovej aktivity. Vďaka týmto rozšíreniam sa eBPF stal veľmi flexibilným nástrojom pre riešenie rôznych problémov v oblasti sieťovej diagnostiky, monitorovania systému a izolácie kontajnerov.

1.2 Princípy eBPF

Z úvodu vieme, že eBPF je veľmi silný nástroj pre vývojárov operačných systémov, ktorý poskytuje flexibilitu a vysoký výkon. Umožňuje spustenie malých programov priamo v jadre operačného systému. Tieto programy sa nazývajú *ëBPF programy"* a sú napísané v bajtkóde. Následujúce podsekcie sú zamerané na hlavné princípy eBPF, funkciu eBPF, vstupné body a spôsoby akými je možné písať samotné eBPF programy. Vo všeobecnosti eBPF prináša možnosť jednoduchého a flexibilného prístupu k jadru operačného systému, ktorá umožňuje vytvoriť nástroje pre monitorovanie, analýzu a riadenie systému bez potreby úpravy samotného jadra operačného systému. Medzi hlavné princípy eBPF patrí:

- **Bezpečnosť**: všetky eBPF programy sú spúšťané vo virtuálnom prostredí, vďaka čomu nespôsobujú poškodenie systému ani iných programov.
- **Flexibilita**: eBPF programy môžu byť spustené v rôznych častiach jadra, ako napríklad pri sledovaní systémových volaní, pri filtrovaní sieťových paketov, alebo pri sledovaní využitia pamäte.
- **Vysoký výkon**: eBPF programy sú spustené priamo v jadre, čo v kombinácií s dobre optimalizovaným bajtkódom, špecializáciou programu na konkrétny účel a možnosťou využívať existujúce dátové štruktúry v jadre (*in-kernel datastructures*) umožňuje vysoký výkon a malú latenciu.
- **Dynamická úprava kódu**: eBPF programy môžu byť dynamicky upravené, alebo nahradené bez nutnosti reštartovať systém, čo umožňuje rýchlu a jednoduchú úpravu správania operačného systému.
- **Kontrola jadra OS**: eBPF programy umožňujú prístup k interným dátam jadra OS a tým pádom lepšie pochopenie a kontrolu nad chodom celého systému.

1.2.1 Ako funguje eBPF

eBPF je nástroj, ktorý sa používa na filtrovanie sieťovej aktivity, sledovanie výkonu a implementáciu bezpečnostných politík v operačných systémoch Linux. Jeho kľúčovým

princípom je používanie *bytecode* interpreteru, pomocou ktorého je schopný vykonávať rôzne operácie. eBPF programy sú riadené udalosťami a spúšťajú sa, keď jadro alebo aplikácia prejde určitým bodom ktorý nazývame vstupný bod alebo *hook*. Po úspešnej kompilácií programu do bajtkódu a validácií sa eBPF program nahrá, do jadra Linuxu. Tento program sa potom spustí pri každom prechode zvoleným bodom/hookom. eBPF program sa skladá z viacerých častí, ktoré sú zodpovedné za rôzne úlohy. Program môže filtrovať pakety, ukladať informácie do máp, vykonávať výpočty a rozhodovať o tom, či má byť paket ponechaný alebo zahodený. Mapou sa označuje jedna z dôležitých funkcií eBPF, ktorá umožňuje ukladať dáta v kerneli do hash tabuliek, ktoré sa potom dajú použiť na rôzne účely, napríklad sledovanie výkonu aplikácií, alebo na implementáciu rôznych bezpečnostných politík. O eBPF mapách a eBPF pomocných funkciách (*eBPF helpers*) sa budeme viac zaoberať v sekcií 1.3s. Ďalej tento nástroj umožňuje implementovať rôzne bezpečnostné politiky, ako napríklad firewall, izolácia kontajnerov, a podobne.

1.2.2 Vstupné body pre eBPF

eBPF hooks a vstupné body sú súčasťou eBPF. Slúžia na spustenie eBPF programu pri vyvolaní konkrétnej udalosti v jadre. Aj keď sa na prvý pohľad môže zdať, že ide o tú istú vec, nie je to tak. Ich funkcionalita sa môže líšiť v konkrétnom kontexte. Vstupné body sú **miesta v jadre**, kde sa môže eBPF program spustiť. Sú to napríklad:

- **kprobe**: Programy sa spustia, **pri vstupe** na konkrétnu adresu v pamäti. Táto adresa sa označuje ako *probe point*. Kprobe je veľmi flexibilný, pretože umožňuje sledovanie akcie v jadre aj v užívateľskom priestore.
- **kretprobe**: Spustí eBPF programu **pri návrate** z konkrétneho kódu v jadre.
- **tracepoint**: Programy sú spustené **pri dosiahnutí** konkrétnych bodov v jadre, ktoré sú označené ako *tracepoints*. Tracepointy sú najmä využívané na sledovanie systémových udalostí, ako sú volania systémových volaní alebo prijímanie a odosielanie sieťových paketov.

eBPF Hooks, na druhej strane, sú mechanizmy, ktoré sa používajú na **pridanie vlastného kódu** do existujúceho kódu v jadre. To znamená, že keď sa kód v jadre vykoná, môže sa volať vlastný kód, ktorý bol pridaný cez hook. Hooks sú definované v jadre operačného systému a poskytujú eBPF programu prístup k rôznym systémovým udalostiam. Preddefinované hooky zahŕňajú systémové volania, vstup a výstup funkcií, sledovacie body jadra, sieťové udalosti a niekoľko ďalších. Pokiaľ pre konkrétnu požiadavku neexistuje preddefinovaný

hook, je možné vytvoriť *kernel probe* (`kprobe`) alebo *user probe* (`uprobe`) na pripojenie eBPF programov takmer kdekoľvek v užívateľských aplikáciách alebo jadre.

1.2.3 Ako sa tvoria programy eBPF

eBPF programy sa píšu pomocou jazyka s nízkou úrovňou, ako je napríklad C. Keďže eBPF programy sú spustené v jadre systému, je dôležité, aby boli bezpečné a nekonfliktné s ostatnými časťami jadra. Preto existujú špeciálne pravidlá a obmedzenia pre písanie eBPF programov, ktoré musia byť dodržiavané. V mnohých scenároch sa eBPF nepoužíva priamo, ale nepriamo prostredníctvom projektov ako je napríklad **Cilium** alebo **KRSI** (*Kernel runtime security instrumentation*). Pre zjednodušenie tvorby eBPF programov vzniklo veľa rôznych knižníc a nástrojov, ktoré umožňujú písať eBPF programy v jazykoch s vyššou úrovňou, ako napríklad **Python**, a potom ich kompilujú do bajtkódu. Tieto nástroje pomáhajú vývojárom vytvárať eBPF programy bez potreby hlbokých znalosti jazyka C a pravidiel pre písanie bezpečných eBPF programov. Pokiaľ, ale neexistuje abstrakcia vyššej úrovne, je potrebné programy písať priamo. Aj keď je samozrejme možné napísať bajtkód priamo, bežnejšou vývojovou praxou je využitie kompilátora, ako je **LLVM**, na kompiláciu pseudo-C kódu do eBPF bajtkódu. Po naprogramovaní a skompilovaní eBPF programu sa následne tento program dá nahrať do jadra linuxu pomocou komunikačného rozhrania *bpf()*.

1.3 Architektúra eBPF

Táto sekcia je spracovaná podľa [1, 8, 5] a zaoberá sa rôznymi mechanizmami, ktoré eBPF ponúka. Pozrieme sa bližšie na preklad kódu cez kompilátor JIT, eBPF mapy a pomocné eBPF funkcie.

1.3.1 Kompilátor JIT

Kompilátor JIT (Just-In-Time) je typ kompilátora, ktorý generuje strojový kód počas behu, a nie vopred ako tradičné kompilátory. [1] Kompilátor zoberie vloženú reprezentáciu kódu väčšinou vo forme bajtkódu alebo assembleru a konvertuje ju na strojový kód, ktorý môže byť vykonaný priamo procesorom. Hlavnou výhodou používania JIT je, že umožňuje efektívnejšie vykonávanie kódu, pretože tento kompilátor môže optimalizovať kód špeciálne pre prostredie behu, napríklad pre konkrétnu architektúru procesora a dostupné pamäťové zdroje. Vďaka tomu sa programy eBPF spúšťajú rovnako efektívne ako natívne skompilovaný kód jadra alebo ako kód načítaný, ako modul jadra. [1] Taktiež má schopnosť dynamicky prekompilovať kód počas behu programu, vďaka čomu sa dokáže prispôsobiť meniacim podmienkam behu a ďalej zvyšovať výkon. V prípade eBPF, JIT kompilácia znamená, že eBPF programy sa kompilujú do strojového kódu priamo v jadre Linuxu, keď

sa spúšťajú. Tým sa dosiahne vyššia rýchlosť a nižšia spotreba pamäte oproti interpretácii eBPF programu v jadre.

1.3.2 eBPF Maps

Dôležitým aspektom eBPF programov je schopnosť zdieľať zhromaždené informácie a ukladať stav. [1] Na tento účel môžu programy eBPF využívať koncept máp. eBPF mapami sa označuje jedna z dôležitých funkcií eBPF, ktorá umožňuje ukladať dáta v kerneli do hash tabuliek, ktoré sa potom dajú použiť na rôzne účely, napríklad na sledovanie výkonu aplikácií, alebo na implementáciu rôznych bezpečnostných politík, ako napríklad firewall, izolácia kontajnerov a podobne. Tiež sa dajú použiť pre implementáciu rôznych nástrojov pre analýzu dát a monitorovanie systémov. K mapám možno pristupovať z eBPF programov, ako aj z aplikácií v používateľskom priestore prostredníctvom systémového volania.

1.3.3 eBPF Helpers

eBPF programy nemôžu volať ľubovoľné funkcie jadra. Ak by sa to povolilo, programy eBPF by sa viazali na konkrétne verzie jadra a skomplikovala by sa kompatibilita programov. Namiesto toho môžu programy eBPF uskutočňovať volania funkcií do pomocných (*helper*) funkcií. eBPF pomocníci (angl. *Helpers*) sú rozšírenia jadra Linuxu, ktoré umožňujú programovať eBPF programy jednoduchšie a bezpečnejšie. [1] Pomáhajú vytvárať a spravovať eBPF objekty, ako sú programy, mapy a tracepointy, a tiež umožňujú komunikovať s jadrom bez nutnosti používať príkazy ako napr. netlink. eBPF Helpre tiež poskytujú vysokú úroveň abstrakcie pre rôzne úlohy, ako napríklad sieťovú filtráciu, tj. riadenie prístupu sieťových paketov, a vyhodnocovanie štatistík. To umožňuje vývojárom vytvárať aplikácie, ktoré využívajú silné a flexibilné možnosti eBPF, bez nutnosti mať hlboké znalosti jadra Linuxu. Príklady dostupných pomocných volaní:

- Prístup k mape eBPF
- Získať aktuálny čas a dátum
- Získať kontext procesu/skupiny
- Manipulácia so sieťovými paketmi a logika presmerovania

Súbor obsahujúci všetky dostupné helper funkcie pre eBPF nájdete priamo v manuáli Linuxu. `https://man7.org/linux/man-pages/man7/bpf-helpers.7.html`

1.4 eBPF nástroje

Tvorba eBPF programov je náročný proces. Pre jeho zjednodušenie vzniklo a stále vzniká veľa nástrojov, ktoré napomáhajú k ľahšiemu a hlavne rýchlejšiemu používaniu a samotnému

vývoju eBPF. Tieto nástroje poskytujú rôzne funkcie, ako napríklad kompiláciu eBPF programov, ladenie eBPF programov, získavanie štatistík o eBPF programe alebo vizuálne zobrazovanie dát získaných prostredníctvom eBPF. V súčasnosti existuje veľa rôznych typov eBPF nástrojov, ktoré poskytujú rôzne funkcie pre spravovanie a využívanie eBPF. Medzi niektoré z najbežnejších typov eBPF nástrojov patria:

- **Kompilátory eBPF**: Tieto nástroje umožňujú kompilovať eBPF programy zdrojového kódu v jazyku C, aj z jazykov s vyššou úrovňou, ako napríklad Python.
- **Debuggery eBPF**: umožňujú ladenie eBPF programov. Umožňujú napríklad sledovať premenné, prezerať hodnoty máp a tracepointov, zastavovať a spúšťať program na konkrétnych bodoch (angl. *breakpoints*).
- **eBPF nástroje pre sieť**: Existujú nástroje ktoré sú špecializované na sieťovú diagnostiku, monitoring a filtráciu pomocou eBPF. Patria sem napríklad nástroje ako tc, iproute2, bpftrace, bcc a iné. Tieto nástroje umožňujú napríklad zistiť *bottleneck* v sieťovej prevádzke, zaznamenávať zdroj a cieľ sieťovej prevádzky, a umožňujú vykonávať sieťovú filtráciu na úrovni jadra systému.

1.4.1 Kompilátory

- **LLVM/Clang**: je kompilátor, ktorý využíva technológiu LLVM (*Low Level Virtual Machine*) pre kompiláciu eBPF programov. Clang je C/C++ kompilátor, ktorý je súčasťou LLVM projektu a umožňuje kompilovať eBPF programy napísané v jazyku C/C++.
- **BCC (BPF Compiler Collection)**: Je súbor nástrojov pre kompiláciu eBPF programov, ktorý poskytuje rôzne možnosti pre kompiláciu eBPF programov z jazykov s vyššou úrovňou, ako Python a Lua. [9]
- **Bpftool**: Tento nástroj je súčasťou jadra Linuxu a umožňuje kompilovať eBPF programy, spravovať mapy eBPF, inštalovať a odstraňovať eBPF programy z jadra.

Každý eBPF program musí prejsť overovačom nato aby mohol byť spustený. O to sa stará *eBPF verifier* ktorý je taktiež súčasťou jadra Linuxu a umožňuje overiť bezpečnosť eBPF programov skôr než sa spustia v jadre. Toto zabezpečuje, že eBPF program neprinesie nežiaduce zmeny v jadre a nebude sa spúšťať nekontrolovateľne.

1.4.2 Debuggery

- **bpftrace**: je nástroj pre dynamické sledovanie eBPF programov, ktorý umožňuje okrem sledovania aj ladenie eBPF programov v reálnom čase. Bpftrace umožňuje

napríklad sledovať systémové volanie (*syscall*), kontext vstupu/výstupu, prístupy k eBPF mapám a podobne. [10]

- **bpf_dbg**: je otvorený-zdrojový eBPF debugger, ktorý umožňuje prezeranie a ladenie eBPF programov v reálnom čase.
- **BCC**: obsahuje nástroj pre ladenie eBPF kódu a umožňuje sledovať a analyzovať eBPF program v reálnom čase.

Programovanie eBPF je výkonné, ale aj zložité. Z toho dôvodu vzniklo niekoľko projektov a dodávateľov, ktorí stavajú na platforme eBPF s cieľom vytvoriť novú generáciu nástrojov, ktoré budú pokrývať monitorovanie, bezpečnosť, sieťovanie a ďalej.

1.5 Knižnice eBPF

eBPF knižnice predstavujú sadu funkcií a rozhraní, ktoré uľahčujú prácu s eBPF, poskytujúc rôzne funkcie pre vývoj a správu eBPF programov. [1] Tieto knižnice umožňujú vývojárom a administrátorom systémov pracovať s eBPF bez potreby hlbokých znalostí jadra Linuxu, pričom poskytujú silné a flexibilné možnosti pre prácu s eBPF programami.

1.5.1 Knižnica libbpf

je generická knižnica eBPF založená na jazyku C/C++, ktorá pomáha oddeliť načítavanie objektových súborov eBPF generovaných kompilátorom clang/LLVM do jadra a vo všeobecnosti abstrahuje interakciu so systémovým volaním BPF poskytovaním ľahko použiteľných API knižníc pre aplikácie. [1] Dôležitá poznámka k tejto knižnici. Keďže obsahuje informácie o formáte typu BPF (*BTF*). Je potrebné použiť konfiguráciu jadra, ktoré má pri kompilácii nastavené `CONFIG_DEBUG_INFO_BTF=y`. V prípade, že vaša distribúcia Linuxu nemá toto nastavenie povolené je potrebná manuálna konfigurácia.

1.5.2 Knižnica BCC

BCC je sada nástrojov na vytváranie efektívnych programov na sledovanie a manipuláciu s jadrom. [9] Väčšina funkcií ktorú BCC používa, vyžaduje Linux 4.1 a vyšší. BCC uľahčuje písanie programov BPF vďaka inštrumentácii jadra v jazyku C a front-endom v jazykoch Python a lua. Je vhodná na mnohé úlohy vrátane analýzy výkonu a riadenia sieťovej prevádzky.

1.5.3 Knižnica eBPF Go

eBPF Go je knižnica jazyka Go, ktorá poskytuje nástroje na načítanie, kompiláciu a ladenie programov eBPF. Má minimálne externé závislosti a je určená na používanie v dlhodobo bežiacich procesoch.

1.6 Použitie eBPF

eBPF programy môžu byť použité pre rôzne účely, ako napríklad pre sieťovú diagnostiku, sledovanie výkonu, filtrovanie sieťovej aktivity, a implementáciu rôznych sieťových a bezpečnostných politík. Táto časť je zameraná na to kde sa dá eBPF použiť a aké bezpečnostné riziká, toto použitie prináša.

- **Monitoring systému**: eBPF sa často používa na sledovanie systémových metrík ako je využitie CPU, pamäte a sieťovej komunikácie.
- **Sledovanie kódu**: eBPF sa môže použiť na sledovanie a profilovanie kódu v reálnom čase.
- **Ochrana pred útokmi**: eBPF sa môže použiť na detekciu a blokovanie škodlivého kódu v reálnom čase.
- **Firewall**: eBPF sa môže použiť ako firewall, ktorý je schopný vykonávať pokročilé filtrovanie sieťového provozu.

Pre zabezpečenie operačného systému Linux sú to napríklad:

- **Kontrola sieťovej komunikácie**: eBPF môže byť použitý na filtrovanie sieťovej aktivity, aby sa zabránilo nežiadúcemu prístupu alebo útoku. Môže tiež byť použitý na vyhľadávanie nebezpečných alebo podozrivých sieťových aktivít a upozornenie na ne.
- **Kontrola súborov**: eBPF môže byť použitý na monitorovanie a kontrolu súborov v operačnom systéme, aby sa zabránilo neoprávnenému prístupu alebo útoku.
- **Kontrola procesov** eBPF môže byť použitý na monitorovanie a kontrolu procesov v operačnom systéme, aby sa zabránilo neoprávnenému použitiu alebo útoku na systém. Môže tiež byť použitý na detekciu a blokovanie nebezpečných procesov alebo aktivít.
- **Kontrola pamäte**: eBPF môže byť použitý na monitorovanie a kontrolu používania pamäte v operačnom systéme, aby sa zabránilo zneužitiu alebo útoku na pamäť.
- **Riadenie systémových volaní**: eBPF môže byť použitý na monitorovanie a riadenie systémových volaní v operačnom systéme, aby sa zabránilo neoprávnenému použitiu alebo útoku.

1.6.1 Bezpečnosť a eBPF

Sekcia je vypracovaná podľa [1]

Počas vývoja eBPF bola bezpečnosť najdôležitejším aspektom pri zvažovaní začlenenia eBPF do jadra Linuxu. Bezpečnostné riziká eBPF súvisia s tým, že eBPF programy môžu spúšťať arbitrárny kód v jadre operačného systému. To znamená, že eBPF programy musia byť správne navrhnuté, implementované a testované, aby sa predišlo bezpečnostným problémom. Kritickou časťou je validácia eBPF pred spustením, aby sa zabezpečilo, že eBPF program neobsahuje nebezpečný kód. Zraniteľnosti v eBPF programoch sú tiež jedným z potenciálnych bezpečnostných rizík. Tieto zraniteľnosti môžu byť využité pre preniknutie do systému, napríklad pre získanie prístupu k root právam alebo pre spustenie vlastného škodlivého kódu. Je nevyhnutné preto, aby boli eBPF programy pravidelne skúmané na prítomnosť zraniteľností a aby boli rýchlo aktualizované pri jej objavení. Ďalším dôležitým aspektom bezpečnosti eBPF je kontrola prístupu k eBPF programom. Je potrebné zabezpečiť, aby iba oprávnené osoby mali prístup k eBPF programom a ich konfigurácii. Je tiež dôležité zaznamenávať a sledovať akúkoľvek neoprávnenú aktivitu s eBPF programami. V konečnom dôsledku, bezpečnosť eBPF je kritickou témou pre úspešné používanie eBPF. Je nevyhnutné, aby boli eBPF programy správne navrhnuté, implementované a testované, a aby bolo zabezpečené ich chránenie pred potenciálnymi bezpečnostnými rizikami. eBPF bezpečnosť je zabezpečená prostredníctvom niekoľkých vrstiev: Začneme s požadovanými oprávneniami. Pokiaľ nie je povolený neprivilegovaný eBPF, všetky procesy, ktoré majú v úmysle načítať programy eBPF do jadra Linuxu, musia byť spustené v privilegovanom režime (root) alebo musia vyžadovať schopnosť `CAP_BPF`. To znamená, že nedôveryhodné programy nemôžu načítať programy eBPF. V prípade, že je povolený neprivilegovaný režim eBPF, neprivilegované procesy môžu načítať určité programy eBPF s výhradou obmedzenej sady funkcií a s obmedzeným prístupom k jadru. Skôr ako bude procesu povolené načítať program eBPF musí tento program prejsť procesom overenia cez takzvaný overovač (angl. *Verifier*). Overovač eBPF zabezpečuje bezpečnosť samotného programu. Pri overení sa kontroluje či program neobsahuje funkcie ktoré by mohli spôsobiť prerušenie systému. Kontroluje sa:

- Či programy eBPF neobsahujú neinicializované premenné, a taktiež či nenastáva prístup do pamäte mimo hraníc.
- Či programy eBPF môžu obsahovať tzv. ohraničené cykly. Program je prijatý len vtedy, ak overovač môže zabezpečiť, že cyklus obsahuje výstupnú podmienku, ktorá sa zaručene stane pravdivou.

- Veľkosť programov eBPF, nakoľko je potrebné aby sa programy zmestili do požiadaviek na veľkosť operačného systému. Nie je možné načítať ľubovoľne veľké programy eBPF.
- Každý program musí mať konečnú zložitosť. Overovač vyhodnotí všetky možné cesty vykonávania a musí byť schopný dokončiť analýzu v medziach nakonfigurovanej hornej hranice zložitosti.

Po úspešnom dokončení overovania program eBPF prejde procesom tvrdenia (angl. *Hardening*) podľa toho, či je program načítaný z privilegovaného alebo neprivilegovaného procesu. Tento krok zahŕňa:

- **Ochranu vykonávania programu**:
  - Pamäť jadra, v ktorej sa nachádza program eBPF, je chránená a je určená len na čítanie. Pokiaľ sa program pokúsi niečo modifikovať, jadro sa zrúti aby neumožnilo pokračovať vo vykonávaní poškodeného/manipulovaného programu.
- **Zmiernenie proti Spectre**:
  - Pri špekulácii môžu procesory nesprávne predpovedať vetvy a zanechať pozorovateľné vedľajšie efekty, ktoré by sa mohli extrahovať prostredníctvom bočného kanála.
- **Konštantné zaslepenie**:
  - Všetky konštanty v kóde sú zaslepené, aby sa zabránilo útokom JIT spraying.

1.6.2 Seccomp

Následujúca sekcia je vypracovaná podľa [11]

Jedným z prvých použití virtuálneho stroja BPF mimo siete bola implementácia politík kontroly prístupu pre systémové volanie seccomp(). *Secure computing mode* alebo seccomp je bezpečnostná funkcia jadra Linux, ktorá umožňuje obmedzenie systémových volaní (*syscalls*), ktoré môžu byť vykonávané procesom. Jedná sa o jednoduchý, ale flexibilný mechanizmus sandboxingu, ktorý sa široko používa. Účelom programu BPF pod funkciou seccomp() je rozhodovať o tom, či má byť dané systémové volanie povolené. Filtračné programy seccomp bežia na "klasickom"virtuálnom stroji BPF, a nie na eBPF. To ale neznamená, že nie je možné skombinovať seccomp s eBPF, aj keď v čase písania tejto práce nie je možné úplne integrovať eBPF do systému seccomp kvôli bezpečnostným problémom. Prechodom na eBPF by sa seccomp() programom sprístupnilo množstvo nových funkcií

vrátane máp, pomocných funkcií, ukladania na jednotlivé úlohy, expresívnejšej inštrukčnej sady a ďalších. Kombinácia eBPF s filtrom seccomp je možná, vyžaduje však úpravu predvolených nastavení a konfiguráciu, ktorá by sa mala testovať a zohľadniť konkrétne požiadavky a potreby aplikácie predtým, než sa nasadí do produkčného prostredia. eBPF môže byť použitý na implementáciu filtrov, ktoré kontrolujú a ovplyvňujú systémové volania, ktoré sú vykonávané procesom. Ďalej poskytuje viacúrovňovú ochranu pre systém, pretože eBPF filtre môžu byť použité na hlbšiu kontrolu systémových volaní, pokiaľ seccomp filtre poskytnú základnú ochranu pred ostatými nebezpečnými systémovými volaniami. Ochrana systému musí byť komplexná a zahŕňať viacero prvkov a implementáciu dobrej bezpečnostnej politiky.

1.6.3 LSM

Následujúca sekcia je vypracovaná podľa internetovej stránky [12, 13]

Linux Security Modules v skratke LSM je rozšírenie operačného systému Linux, ktoré umožňuje pridávať rôzne moduly pre zlepšenie zabezpečenia. BPF je jedným z týchto modulov, ktorý môže byť použitý na rôzne účely, vrátane zlepšenia bezpečnosti operačného systému. BPF môže byť nakonfigurovaný tak, aby filtroval sieťový prenos na základe rôznych kritérií, ako napríklad IP adresy, porty alebo protokoly. Týmto spôsobom môžete zabrániť nežiaducim sieťovým prenosom, ako sú napríklad útoky DDoS, a zlepšiť zabezpečenie siete. Ďalším spôsobom, ako je možné použiť LSM BPF na zlepšenie bezpečnosti operačného systému, je kontrolou prístupu k systémovým objektom, ako sú súbory alebo procesy. Nastavením eBPF tak, aby kontroloval prístup napríklad na základe užívateľského mena alebo skupiny. Týmto spôsobom je možné zlepšiť kontrolu prístupu k dôležitým systémovým objektom a zvýšiť bezpečnosť operačného systému. Framework bezpečnostného modulu Linuxu (LSM) poskytuje mechanizmus na pripojenie rôznych bezpečnostných kontrol pomocou rozšírení jadra. Primárnymi používateľmi rozhrania LSM sú rozšírenia **MAC** (*Mandatory Access Control*), ktoré poskytujú komplexnú bezpečnostnú politiku. Okrem väčších rozšírení MAC možno pomocou rozhrania LSM vytvárať aj ďalšie rozšírenia, ktoré poskytujú špecifické zmeny fungovania systému, ak tieto úpravy nie sú k dispozícii v základnej funkcii samotného systému Linux. Z pohľadu bezpečnostného správania sa lepšie mapuje na LSM ako na filtre seccomp, ktoré sú založené na zachytávaní systémových volaní. Pri filtri seccomp sa bezpečnostné správanie môže realizovať prostredníctvom viacerých systémových volaní, pričom je ľahké jedno alebo viacero z nich prehliadnuť, zatiaľ čo LSM hooks zachytávajú správanie, ktoré je predmetom záujmu. Zámerom je, aby pomocné eBPF funkcie boli "presné a granulárne".

1.6.4 KRSI

Následujúca sekcia je vypracovaná podľa internetovej stránky [14, 15]

Kernel Runtime Security Instrumentation, alebo v skratke KRSI je koncepcia, ktorá umožňuje vykonávať vlastný kód v jadre operačného systému v reálnom čase, čo umožňuje vykonávať rôzne bezpečnostné úlohy. Hlavným cieľom je sledovať celkové správanie systému za účelom odhalenia útokov. eBPF je jazyk a platforma pre KRSI, ktorá umožňuje vývojárom naprogramovať vlastné eBPF programy, ktoré môžu byť spustené v jadre operačného systému. Tieto programy môžu byť využité na rôzne úlohy, ako napríklad monitorovanie siete, detekciu útokov, bezpečnostnú kontrolu prístupu k súborom, a podobne.

Ako používať KRSI

Prototyp KRSI je implementovaný ako bezpečnostný modul Linuxu (LSM), ktorý umožňuje pripojenie programov eBPF k bezpečnostným hookom jadra. Pre používanie KRSI, je potrebné mať verziu jadra, ktorá podporuje eBPF. Po načítaní eBPF programov do jadra ich možno pripojiť k rôznym funkciám jadra pomocou mechanizmu KRSI. To umožňuje, aby sa programy eBPF spustili vždy, keď sa tieto funkcie zavolajú, čo umožňuje dynamické presadzovanie bezpečnostných politík. V prípade zabezpečenia operačného systému sa KRSI dá použiť na monitorovanie systému a zhromažďovanie údajov o akomkoľvek podozrivom správaní, ako je napríklad pokus o neoprávnený prístup. KRSI exportuje novú hierarchiu súborového systému pod `/sys/kernel/security/bpf` s jedným súborom pre každý hook. Na daný hook môže byť pripojených viac ako jeden program a pri každom volaní sa postupne zavolajú všetky pripojené programy BPF. Ak niektorý program BPF vráti chybový stav, požadovaná akcia sa zamietne.

2 Aplikácie eBPF

V sieťovom priemysle sa eBPF používa na monitorovanie a filtrovanie sieťovej premávky, čo umožňuje analýzu a optimalizáciu sieťovej infraštruktúry. Tiež sa využíva na implementáciu bezpečnostných opatrení, ako napr. kontrola prístupu k sieťovej premávke, detekcia útokov a ochrana pred útokmi. V oblasti kontroly procesov v priemysle, eBPF umožňuje kontrolu a riadenie procesov na úrovni jadra operačného systému, čo poskytuje presné a rýchle sledovanie využívania zdrojov a umožňuje lepšiu analýzu výkonu procesov. V oblasti telekomunikácií eBPF sa využíva na monitorovanie a riadenie sieťovej aktivity, detekciu a odstránenie *bottleneckov*, ako aj na implementáciu bezpečnostných opatrení pre telekomunikačné siete. Všeobecne je eBPF využívaný v rôznych odvetviach, kde potrebujeme kontrolovať a riadiť procesy na úrovni jadra operačného systému alebo poskytovať presné a rýchle sledovanie využívania zdrojov. Taktiež umožňuje lepšiu analýzu výkonu a optimalizáciu procesov.

2.1 Detekcia útokov

Existujú rôzne spôsoby, ako môže byť eBPF použitý na detekciu útokov, sú to napríklad:

- **Monitorovanie siete**: eBPF môže byť použitý na monitorovanie sieťovej premávky v reálnom čase.
- **Riadenie systémových volaní**: eBPF môže byť použitý na riadenie systémových volaní, ktoré sú vykonávané procesom, a na detekciu neobvyklého využívania systémového volania, ktoré by mohlo byť spôsobené útokom.
- **Analýza údajov**: eBPF môže byť použitý na analyzovanie údajov získaných z monitorovania siete a systémových volaní, a na detekciu neobvyklého správania, ktoré by mohlo byť spôsobené útokom.

Použitie eBPF ako jedného zo zabezpečovacích nástrojov môže byť veľmi efektívne pri detekcii útokov, ale musí sa kombinovať s ďalšími opatreniami pre zabezpečenie systému.

3 Porovnanie eBPF s inými nástrojmi

V dnešnej dobe sa môžeme stretnúť s veľkou ponukou nástrojov slúžiacich na zabezpečenie operačného systému. Pár najznámejších si uvedieme v zozname nižšie.

- **SELinux (Security Enhanced Linux)**: je klasický nástroj na zabezpečenie operačného systému Linux, ktorý poskytuje mechanizmy izolácie a ochrany prostredníctvom rolí a politík.
- **AppArmor**: je nástroj pre dynamickú kontrolu prístupu, ktorý umožňuje nastavenie profilov pre jednotlivé aplikácie, aby sa zabránilo nežiadúcemu prístupu k systémovým zdrojom.
- **FirewallD**: je firewall pre Linux, ktorý umožňuje správu firewallu prostredníctvom rozhrania D-Bus.
- **Iptables**: je klasický firewall nástroj pre Linux, ktorý poskytuje základnú kontrolu prístupu k sieťovej premávke.
- **OpenSSH**: je nástroj pre zabezpečenie sieťového pripojenia, ktorý umožňuje bezpečné prihlásenie a komunikáciu prostredníctvom šifrovaných protokolov.
- **Tripwire**: je nástroj na detekciu zmien v súborovom systéme, ktorý umožňuje detegovať nežiaduce zmeny v systéme a prijímať opatrenia na ich odstránenie.

4 Implementácia pomocou eBPF

Predbežným návrhom na implemetáciu sú malé programy zamerané na:

1. **Kontrolu súborov**: eBPF môže byť použitý na monitorovanie a kontrolu súborov v operačnom systéme, aby sa zabránilo neoprávnenému prístupu alebo útoku.

2. **Kontrolu procesov**: eBPF môže byť použitý na monitorovanie a kontrolu procesov v operačnom systéme, aby sa zabránilo neoprávnenému použitiu alebo útoku. Môže tiež byť použitý na detekciu a blokovanie nebezpečných procesov alebo aktivít.

3. **Kontrolu pamäte**: eBPF môže byť použitý na monitorovanie a kontrolu používania pamäte v operačnom systéme, aby sa zabránilo zneužitiu alebo útoku na pamäť.

4. **Kontrolu systémových volaní**: eBPF môže byť použitý na monitorovanie a kontrolu systémových volaní v operačnom systéme, aby sa zabránilo neoprávnenému použitiu alebo útoku.

5. **Ochranu pred fileless malware**: eBPF môže byť použité pre demonštratívnu implementáciu ochrany pred fileless malwarom.

5 Výsledky implemetácie

Záver

Conclusion is going to be where?
Here.

Zoznam použitej literatúry

1. BORKMANN, Daniel a STAROVOITOV, Alexei. *eBPF* [online]. 2022. [cit. 2022-11-24]. Dostupné z : `https://ebpf.io/`.
2. KENISTON, Jim, PANCHAMUKHI, Prasanna S a HIRAMATSU, Masami. *Kernel Probes (Kprobes)* [online]. 2019. [cit. 2023-01-11]. Dostupné z : `https://www.kernel.org/doc/Documentation/kprobes.txt`.
3. MAGUIRE, Alan. *Taming Tracepoints in the Linux Kernel* [online]. 2020. [cit. 2023-01-11]. Dostupné z : `https://blogs.oracle.com/linux/post/taming-tracepoints-in-the-linux-kernel`.
4. CARTER, Eric. *Introducing Container Observability with eBPF* [online]. 2019. [cit. 2023-01-10]. Dostupné z : `https://sysdig.com/blog/introducing-container-observability-with-ebpf-and-sysdig/`.
5. RICE, Liz. *What Is eBPF?* 1st ed. O'Reilly Media, Inc., 2022. Dostupné tiež z: `https://www.oreilly.com/library/view/what-is-ebpf/9781492097266/`.
6. GREGG, Brendan. *Linux Extended BPF (eBPF) Tracing Tools* [online]. 2021. [cit. 2023-02-10]. Dostupné z : `https://www.brendangregg.com/ebpf.html`.
7. MCCANNR, Steven a JACOBSON, Van. *The BSD Packet Filter: A New Architecture for User-level Packet Capture* [online]. 2023. [cit. 2023-01-10]. Dostupné z : `https://www.tcpdump.org/papers/bpf-usenix93.pdf`.
8. FRENCH, Steve. *BPF Documentation* [online]. 2022. [cit. 2023-02-10]. Dostupné z : `https://www.kernel.org/doc/html/latest/bpf/index.html`.
9. PROJECT, IO Visor. *BPF Compiler Collection (BCC)* [online]. 2023. [cit. 2023-02-10]. Dostupné z : `https://github.com/iovisor/bcc`.
10. PROJECT, IO Visor. *bpftrace* [online]. 2023. [cit. 2023-02-10]. Dostupné z : `https://github.com/iovisor/bpftrace`.
11. CORBET, Jonathan. *eBPF seccomp() filters* [online]. 2021. [cit. 2023-01-11]. Dostupné z : `https://lwn.net/Articles/857228/`.
12. SMALLEY, Stephen, FRASER, Timothy a VANCE, Chris. *Linux Security Modules: General Security Hooks for Linux* [online]. 2023. [cit. 2023-02-10]. Dostupné z : `https://docs.kernel.org/security/lsm.html`.

13. FRENCH, Steve. *Linux Security Module Usage* [online]. 2018. [cit. 2023-02-10]. Dostupné z : `https://www.kernel.org/doc/html/v4.15/admin-guide/LSM/index.html`.

14. EDGE, Jake. *Kernel runtime security instrumentation* [online]. 2019. [cit. 2023-01-11]. Dostupné z : `https://lwn.net/Articles/798157/`.

15. CORBET, Jonathan. *KRSI — the other BPF security module* [online]. 2019. [cit. 2023-02-10]. Dostupné z : `https://lwn.net/Articles/808048/`.