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

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

1 Teoretické základy eBPF

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 monitorovanie a 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 problémy pri zlom zaobchádzaní. [1]

1.1 História eBPF

História eBPF sa začína s technológiou nazývanou *Berkeley Packet Filter* v skratke BPF, ktorá vznikla v národnom laboratóriu Lawrenca Berkeleyho, kde ju 19. Decembra 1992 opísali Steven McCanne spolu s Vanom Jacobsonom. 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 medzi 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 (*sandboxe*) v rámci jadra operačného systému. To znamená, že môže rozšíriť alebo dokonca upraviť spôsob, akým sa jadro správa bez zmeny zdrojového kódu

jadra. 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, 3]

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 bytekóde. V nasledujúcich pod sekciách sa oboznámime s hlavnými princípmi eBPF, vysvetlíme si ako funguje eBPF, aké má vstupné body a ako sa píšu 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 v *sandboxovanom* 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 bytekódom, 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]

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*(hák). Po úspešnej kompilácií programu do bytekódu a validácií sa eBPF program nahrá, do jadra Linuxu. Tento program sa potom spustí pri každom prechode zvoleným hákom. 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 pomocníkoch (*helpers*) sa budeme viac zaoberať v sekcií 1.3 Architektúra eBPF. Ďalej tento nástroj umožňuje implementovať rôzne bezpečnostné politiky, ako napríklad firewall, izolácia kontajnerov, a podobne. V skratke by sme mohli funkciu eBPF popísať ako virtuálny počítač, ktorý beží v jadre OS a na základe vloženého bytekódu vykonáva operácie v rôznych častiach systému ako napríklad ukladanie dát do máp, implementáciu bezpečnostných politík, alebo optimalizáciu výkonu systému. [1]

1.2.2 Vstupné body pre eBPF

Vstupné body a háky (*hooks*) 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.

Háky, 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 hák. Háky sú definované v jadre operačného systému a poskytujú eBPF programu prístup k rôznym systémovým udalostiam. Preddefinované háky zahŕňajú systémové volania, vstup a výstup funkcií, sledovacie body jadra, sieťové udalosti a niekoľko ďalších. Pokiaľ pre konkrétnu požiadavku neexistuje preddefinovaný hák, je možné vytvoriť *kernel probe* (`kprobe`) alebo *user probe* (`uprobe`) na pripojenie eBPF programov takmer kdekoľvek v užívateľských aplikáciách alebo jadre. [1, 4, 5, 6]

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 bytekódu. 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 *bytecode* 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]

1.3 Architektúra eBPF

V tejto sekcií sa budeme zaoberať rôznymi mechanizmami ktoré nám eBPF ponúka. Pozrieme sa bližšie na preklad kódu cez kompilátory JIT a LLVM, eBPF mapy a eBPF helpre.

1.4 Still in Progress

1.4.1 Kompilácia cez JIT

Krok kompilácie `Just-in-Time` (`JIT`) prekladá všeobecný bajtový kód programu do inštrukčnej sady špecifickej pre stroj s cieľom optimalizovať rýchlosť vykonávania programu. Vďaka tomu sa programy eBPF spúšťajú rovnako efektívne ako natívne skompilovaný kód jadra alebo ako kód načítaný ako modul jadra.

Požadované oprávnenia

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. Ak je zapnutý neprivilegovaný režim eBPF, neprivilegované procesy môžu načítať určité programy eBPF s výhradou obmedzenej sady funkcií a s obmedzeným prístupom k jadru.

Overovač (Verifier)

Ak je procesu povolené načítať program eBPF, všetky programy stále prechádzajú cez overovač eBPF. Overovač eBPF zabezpečuje bezpečnosť samotného programu. To znamená, že napr:

- Programy eBPF môžu obsahovať tzv. ohraničené slučky, ale program je prijatý len vtedy, ak overovateľ môže zabezpečiť, že slučka obsahuje výstupnú podmienku, ktorá sa zaručene stane pravdivou.
- Programy nesmú používať žiadne neinicializované premenné ani pristupovať do pamäte mimo hraníc.
- Programy sa musia zmestiť do požiadaviek na veľkosť systému. Nie je možné načítať ľubovoľne veľké programy eBPF.
- Program musí mať konečnú zložitosť. Overovač vyhodnotí všetky možné cesty vykonávania a musí byť schopný dokončiť analýzu v medziach nakonfigurovanej hornej hranice zložitosti.

Hardening (Tvrdenie)

Po úspešnom dokončení overovania program eBPF prejde procesom „tvrdenia“ 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.

1.4.2 Kompilácia cez LLVM

1.4.3 eBPF Maps

Dôležitým aspektom programov eBPF je schopnosť zdieľať zhromaždené informácie a ukladať stav. Na tento účel môžu programy eBPF využívať koncept máp pre ukladanie a načítavanie údajov v širokom súbore dátových štruktúr. K mapám možno pristupovať z eBPF programov, ako aj z aplikácií v používateľskom priestore prostredníctvom systémového volania.

1.4.4 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 funkcií, čo je dobre známe a stabilné API, ktoré jadro ponúka. Súbor dostupných pomocných volaní sa neustále vyvíja. Príklady dostupných pomocných volaní:

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

1.5 eBPF nástroje

Programovanie eBPF je neuveriteľne 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 pozorovateľnosť, bezpečnosť, sieťovanie a ďalej.

1.5.1 Knižnica libbpf C/C++

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.

1.5.2 Knižnica BCC

Umožňuje používateľom písať programy v jazyku python s vloženými programami eBPF. Tento framework je primárne zameraný na prípady použitia, ktoré zahŕňajú profilovanie-/sledovanie aplikácií a systémov, kde sa program eBPF používa na zber štatistík alebo generovanie udalostí. V používateľskom priestore zbiera údaje a zobrazuje ich v ľudsky čitateľnej forme.

1.5.3 Knižnica eBPF Go

poskytuje všeobecnú knižnicu eBPF, ktorá oddeľuje proces získania bajtkódu eBPF, načítanie a správu programov eBPF. Programy eBPF sa zvyčajne vytvárajú napísaním jazyka vyššej úrovne a potom sa pomocou kompilátora clang/LLVM skompilujú do bajtkódu eBPF.

1.6 Použitie eBPF

1.6.1 Bezpečnosť a eBPF

Počas vývoja eBPF bola bezpečnosť najdôležitejším aspektom pri zvažovaní začlenenia eBPF do jadra Linuxu. eBPF bezpečnosť je zabezpečená prostredníctvom niekoľkých vrstiev:

1.6.2 Seccomp

Mechanizmus seccomp() umožňuje procesu načítať BPF program na obmedzenie jeho budúceho používania systémových volaní. Jedná sa o jednoduchý, ale flexibilný mechanizmus sandboxingu, ktorý sa široko používa. Tieto filtračné programy však bežia na "klasickom"virtuálnom stroji BPF, a nie na rozšírenom stroji eBPF, ktorý sa používa na iných miestach jadra. Účelom programu BPF pod funkciou seccomp() je rozhodovať o tom, či má byť dané systémové volanie povolené. Prechodom na eBPF by sa seccomp() programom sprístupnilo množstvo nových funkcií vrátane máp, pomocných funkcií, ukladania na jednotlivé úlohy, expresívnejšej inštrukčnej sady a ďalších. Programy pre eBPF možno písať v jazyku C, čo nie je možné pre programy klasického BPF. Tento problém, viedol k vytvoreniu špeciálnych jazykov, ako je easyseccomp. Kvôli bezpečnostným problémom nie je zatiaľ možné integrovať eBPF do systému seccomp()

* Jedným z prvých použití virtuálneho stroja BPF mimo siete bola implementácia

politík kontroly prístupu pre systémové volanie seccomp().

1.6.3 LSM (Linux Security Modules)

Framework bezpečnostného modulu Linuxu (LSM) poskytuje mechanizmus na pripojenie rôznych bezpečnostných kontrol pomocou nových 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 syscallov. Rôzne bezpečnostné správanie sa môže realizovať prostredníctvom viacerých systémových volaní, takže by bolo ľahké jedno alebo viacero z nich prehliadnuť, zatiaľ čo hooky LSM zachytávajú správanie, ktoré je predmetom záujmu. Zámerom je, aby eBPF helpre boli "presné a granulárne". Na rozdiel od API sledovania BPF nebudú mať všeobecný prístup k vnútorným dátovým štruktúram jadra. KRSI vyžaduje na svoju prácu `/CAP_SYS_ADMIN`. * `/CAP_SYS_ADMIN` je potrebný na vykonávanie celého radu administratívnych operácií, ktoré je ťažké z kontajnerov vypustiť, ak sa v kontajneri vykonávajú privilegované operácie.

1.6.4 KRSI

Prototyp KRSI je implementovaný ako bezpečnostný modul Linuxu (LSM), ktorý umožňuje pripojenie programov eBPF k bezpečnostným hákom jadra. Hlavným cieľom KRSI je sledovať celkové správanie systému za účelom odhalenia útokov. KRSI exportuje novú hierarchiu súborového systému pod `/sys/kernel/security/bpf` s jedným súborom pre každý hák. K danému háku môže byť pripojených viac ako jeden program. Pri každom volaní bezpečnostného háku sa postupne zavolajú všetky pripojené programy BPF, a ak niektorý program BPF vráti chybový stav, požadovaná akcia sa zamietne.

2 Aplikácie eBPF

2.1 Detekcia útokov

3 Porovnanie eBPF s inými nástrojmi

4 Implementácia vlastného riešenia pomocou eBPF

5 Vyhodnotenie a porovnanie výsledkov implementácie

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. 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/`.
3. WIKIPEDIA. *Berkeley Packet Filter* [online]. 2023. [cit. 2023-01-10]. Dostupné z : `https://en.wikipedia.org/wiki/Berkeley_Packet_Filter`.
4. 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/`.
5. 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`.
6. 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`.