



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

INTEGRACE HERNÍHO SCÉNÁŘE A/D DO PLATFORMY BUTCA

INTEGRATION OF A/D GAME SCENARIO INTO THE BUTCA PLATFORM

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE

AUTHOR

Bc. Radek Slaný

VEDOUCÍ PRÁCE

SUPERVISOR

doc. Ing. Zdeněk Martinásek, Ph.D.

BRNO 2024

Diplomová práce

magisterský navazující studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Bc. Radek Slaný

ID: 221570

Ročník: 2

Akademický rok: 2023/24

NÁZEV TÉMATU:

Integrace herního scénáře A/D do platformy BUTCA

POKYNY PRO VYPRACOVÁNÍ:

Hlavním cílem diplomové práce je návrh a implementace nového herního scénáře do platformy BUTCA, který bude zaměřen současně na útok a obranu aplikačního serveru. Scénář bude určen pro týmy hráčů soutěžících proti sobě (př. 2 proti 2), kdy každý tým bude mít k dispozici stejný virtuální stroj. Součástí scénáře bude Game server, který bude kontrolovat a hodnotit jednotlivé týmy. Implementované řešení scénáře důkladně otestujte.

DOPORUČENÁ LITERATURA:

[1] LAZAROV, W.; STODŮLKA, T. Multi-Level Approach to Cybersecurity Education. In Proceedings II of the 29th Student EEICT 2023. 1. Brno: Brno University of Technology, Faculty of Electrical Engineering and Communication, 2023. s. 121-124. ISBN: 978-80-214-6154-3.

[2] STODŮLKA, T.; FUJDIÁK, R. Budování Cyber Range platformy s technologií cloud computingu. Brno: Vysoké učení technické v Brně, 2022. 153 s. ISBN: 978-80-214-6064-5.

Termín zadání: 5.2.2024

Termín odevzdání: 21.5.2024

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

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

UPOZORNĚNÍ:

Autor diplomové práce nesmí při vytváření diplomové práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

ABSTRAKT

Je takřka nemožné vyhnout se informačním technologiím. S tím přichází zvýšené nároky na bezpečnost, které je možné naplnit pouze s pomocí trénovaných profesionálů. BUTCA je Cyber Range platforma, jež studentům i profesionálům poskytuje prostřednictvím výukových scénářů, prostředí, ve kterém si mohou prakticky otestovat své teoretické znalosti. V BUTCA platformě existují scénáře zaměřené buď na defenzivní, nebo ofenzivní kybernetickou bezpečnost. Diplomová práce představuje výukový scénář, který ověřuje současně ofenzivní i defenzivní schopnosti účastníků. Scénář je koncipován formou soutěže dvou týmů, při níž se oba týmy snaží kompromitovat instance druhého týmu, a zároveň zabránit soupeřům v kompromitaci svých instancí. Hlavním přínosem diplomové práce je návrh a implementace scénáře, který je snadno škálovatelný pro různé velké týmy a je také snadno rozšiřitelný díky využití vlastní knihovny zranitelností. Výsledná implementace výukového scénáře byla zavedena do BUTCA prostředí, kde byl scénář otestován za pomoci týmu profesionálních penetračních testerů.

KLÍČOVÁ SLOVA

BUTCA, CTF, Kyber bezpečnost, Výuka, Red Team, Blue Team

ABSTRACT

Nowadays we live in world where it is almost impossible to exist without information technologies. Therefore, higher requirements for cyber security arise and with the requirements grows the need for trained professionals. BUTCA is Cyber Range platform which provides, through educational scenarios, students and professionals environment where they can test their theoretical knowledge in practical examples. Current scenarios in BUTCA are focused either on defensive or offensive cyber security skills. This master's thesis proposes educational scenario that is focused on defensive and offensive security skills at the same time. Scenario is designed for teams competing each other, each team tries to compromise instances of the other team while trying to defend own instances against cyber attacks of other team. Main contribution of this master's thesis is design and implementation of described scenario which will be easily scalable for teams of different number of players and also that would be easily expandable thanks to usage of its own library of vulnerabilities. Final implementation of the educational scenario was imported into BUTCA platform where the scenario was tested with the assistance of team of professional penetration testers.

KEYWORDS

BUTCA, CTF, Cybersecurity, Education, Red Team, Blue Team

SLANÝ, Radek. *Integrace herního scénáře A/D do platformy BUTCA*. Diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2024. Vedoucí práce: doc. Ing. Zdeněk Martinásek, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora: Bc. Radek Slaný
VUT ID autora: 221570
Typ práce: Diplomová práce
Akademický rok: 2023/24
Téma závěrečné práce: Integrace herního scénáře A/D do platformy BUTCA

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno

.....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu doc. Ing. Zdeňkovi Martináskovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Úvod	11
1 Vymezení pojmu penetrační testování	12
1.1 Fáze penetračního testování	12
1.1.1 Pre-Engagement Interactions	14
1.1.2 Intelligence Gathering	14
1.1.3 Threat Modeling	15
1.1.4 Vulnerability Analysis	15
1.1.5 Exploitation	15
1.1.6 Post-Exploitation	15
1.1.7 Reporting	16
1.2 Motivace penetračního testování	16
1.3 Kategorie penetračního testování	16
1.3.1 Dle znalosti	17
1.3.2 Dle umístění testera	18
2 Cyber Range platformy	19
2.1 Cyber Range platformy ve výuce	19
2.2 BUTCA	21
3 Zranitelnosti vlastní knihovny zranitelností	23
3.1 Initial Foothold	23
3.1.1 File Inclusion	23
3.1.2 XML External Entity	25
3.1.3 Server Side Template Injection	27
3.1.4 OS Command Injection	28
3.2 Privilege Escalation	30
3.2.1 SUID spustitelný soubor s relativními cestami	31
3.2.2 SUDO miskonfigurace proměných systému	31
3.2.3 SUDO obecná miskonfigurace	32
4 Implementace herního scénáře do platformy BUTCA	34
4.1 Vlastní návrh scénáře	34
4.2 Vlastní návrh implementační strategie scénáře do BUTCA prostředí	35
4.3 Návrhy a implementace jednotlivých komponent Game Serveru	37
4.3.1 Dílčí nástroj pro tvorbu zranitelných instancí herního scénáře	37
4.3.2 Dílčí nástroj pro provoz herního scénáře	41
4.3.3 Dílčí nástroj pro komunikaci mezi hráči a herním scénářem	43

4.3.4	Nástroj pro spuštění dílčích nástrojů herního scénáře	47
5	Testování scénáře	49
5.1	Lokální testovacího prostředí	49
5.2	Testování v BUTCA prostředí	49
	Závěr	54
	Literatura	55
	Seznam symbolů a zkratk	57
A	Seznam elektronických příloh	59

Seznam obrázků

1.1	Cyklus penetračního testování	13
2.1	Spektrum týmů v oblasti kybernetické bezpečnosti a vývoje	20
2.2	Prostředí výběru scénáře v platformě BUTCA	21
4.1	Návrh implementace	36
4.2	Docker kontejnery s typem sítě macvlan	40
4.3	Webové rozhraní po spuštění scénáře	44
4.4	Webové rozhraní po zadání vlajek	44
4.5	Webové rozhraní při zjištění „nezdravé“ instance	45
4.6	Webové rozhraní v následujícím kole	45
4.7	Webové rozhraní po opravě „nezdravé“ instance	46
4.8	Webové rozhraní po ukončení scénáře	46
5.1	Schéma lokálního testovacího prostředí	50
5.2	Pravidla herního scénáře v prostředí platformy BUTCA	52

Seznam výpisů

3.1	Testovací XXE payload	26
3.2	XML dokument zpracovaný zranitelnou aplikací	26
3.3	Obecný black list	30
3.4	Black list pro Windows	30
3.5	Black list pro Linux	30
3.6	SUDO miskonfigurace systémových proměnných	32
3.7	SUDO obecná miskonfigurace	32
4.1	Ukázka souborů hesel administrátorských účtů obou týmů	38
4.2	Příkaz k vytvoření Docker image	39
4.3	Příkaz k vytvoření Docker kontejnerů	39
4.4	Nápověda nástroje pro tvorbu zranitelných instancí	41
4.5	Ukázka použití nástroje pro tvorbu zranitelných instancí	41
4.6	Ukázka souboru s vlajkami	41
4.7	Nápověda nástroje pro provoz scénáře	43
4.8	Ukázka použití nástroje pro provoz scénáře	43
4.9	Nápověda nástroje pro komunikaci mezi hráči a herním scénářem	47
4.10	Ukázka použití nástroje pro komunikaci mezi hráči a herním scénářem	47
4.11	Nápověda nástroje pro spuštění dílčích nástrojů herního scénáře	48
4.12	Ukázka použití nástroje pro spuštění dílčích nástrojů herního scénáře	48

Úvod

Stále více se spoléháme na informační technologie, ale abychom těmto technologiím mohli věřit je nutné pravidelně ověřovat a testovat jejich bezpečnost. Vhodným přístupem je tzv. penetrační testování, které je věrnou simulací reálného kybernetického útoku. Pro kvalitní penetrační testování je potřeba mnoho znalostí a zkušeností, které ovšem není snadné získávat. Vhodným nástrojem pro prohlubování znalostí jsou tzv. „Cyber Ranges“ a „Cyber Arény“, jež simulují zranitelné prostředí. Může se jednat od jednotlivých zranitelných strojů, přes klasické on-prem řešení *Active Directory* (AD) až po celé lesy hybridního Microsoft Entra ID¹.

Předkládaná diplomová práce má za cíl vytvořit scénář pro platformu BUTCA, který bude kombinovat jak obranou tak i útočnou povahu penetračního testování. V teoretické části budou popsány metodologie pentetračního testování a jeho různé druhy a typy. Dále budou představeny zranitelnosti, které se budou tvořit vlastní knihovnu zranitelností používanou ke tvorbě zranitelných instancí, jež budou součástí scénáře. U každé zranitelnosti bude uvedeno jak danou zranitelnost identifikovat a co ji způsobuje, dále pak dopad této zranitelnosti a také bezpečnostní opatření, jež zabrání zneužití zranitelnosti, nebo alespoň zmírní dopad při jejím případném zneužití. V následující kapitole budou popsány průběh scénáře a implementační strategie do platformy BUTCA.

V rámci diplomové práce bude vytvořen Game server, který bude vytvářet jednotlivé virtuální instance, kontrolovat jejich stav a hodnotit jednotlivé týmy. Game server bude realizován jednotlivými komponentami. První komponenta bude zodpovědná za sestavení zranitelné instance, k tomu bude využívat vlastní knihovnu zranitelností, ze které náhodně vybere zadaný počet zranitelností a vytvoří zranitelnou virtuální instanci, jež bude využita ve scénáři. Druhá komponenta bude mít na starost kontrolu stavu jednotlivých zranitelných instancí a výměnu vlajek v daných časových intervalech. Třetí komponenta bude zodpovědná za bodové hodnocení týmů a komunikaci mezi hráči a scénářem.

¹dříve *Azure Active Directory* (AAD) - <https://www.microsoft.com/cs-cz/security/business/identity-access/microsoft-entra-id>

1 Vymezení pojmu penetrační testování

V této kapitole bude vymezen pojem penetračního testování, popsána motivace penetračního testování, dále budou definovány jednotlivé fáze penetračního testování a také přístupy k penetračnímu testování.

Pojem penetrační testování lze definovat několika způsoby. *The UK National Cyber Security Center* (NCSC) definuje penetrační testování následovně: „Metoda pro získání jistoty o zabezpečení IT systému pokusem o prolomení části nebo celého zabezpečení tohoto systému za použití stejných nástrojů a technik, jaké by mohl použít protivník.“[1]. Jedná se tedy o autorizovanou simulaci kybernetického útoku. Je to proces, který sestává z několika metod a procedur, jejichž cílem je ověřit bezpečnost testovaného systému. Hlavním úkolem penetračního testování je odhalit zranitelnosti testovaného systému, ověřit možnost jejich zneužití a identifikovat jejich možný dopad na aktiva dané společnosti. Kromě pojmu penetrační testování se můžeme setkat také s pojmem *pentest*, jenž vzniknul spojením slov *penetration* a *testing*.

Penetrační testování bývá leckdy zaměňováno s pojmem skenování zranitelností. Tyto dva pojmy jsou však velmi odlišné a rozhodně by zaměňovány být neměly. Pravděpodobně nejstěžejnějším rozdílem mezi skenováním zranitelností a penetračním testováním je ten, že při skenování zranitelností jsou zranitelnosti pouze identifikovány a zákazníkovi je předána zpráva, která je pouhým soupisem nalezených zranitelností. Při penetračním testování jsou zranitelnosti nejen identifikovány, ale jsou provedeny i aktivní pokusy o jejich exploitaci¹. Závěrečná zpráva penetračního testování pak kromě identifikovaných zranitelností obsahuje také popis kroků, jež vedly k úspěšné exploitaci. Popis kroků, které vedly k exploitaci, by měl být srozumitelný a replikovatelný.

1.1 Fáze penetračního testování

Průběh penetračního testování je definován v několika standardech jako jsou *Penetration Testing Execution Standard* (PTES), *The National Institute of Standards and Technology* (NIST), *Open Web Application Security Project* (OWASP), *Open Source Security Testing Methodology Manual* (OSSTMM) nebo *Information System Security Assessment Framework* (ISSAF). Každý ze zmíněných standardů poskytuje mírně odlišný problém na problematiku penetračního testování, lze však mezi nimi nalézt jisté podobnosti. Penetrační testování je rozděleno do několika částí, v závislosti na zvoleném standardu se různí jejich počet od 5 až po 7 fází. Při bližším

¹Exploitací se rozumí úspěšné zneužití dané zranitelnosti v daném prostředí.

1.1.1 Pre-Engagement Interactions

Ve fázi Pre-Engagement Interactions penetračního testování dochází k plánování průběhu penetračního testu. Jedná se pravděpodobně o jednu z nejdůležitějších fází. V této fázi jsou definovány komunikační matice. Mezi oběma stranami je dohodnuto časové rozpětí nadcházejícího testování a to nejen celková délka trvání, ale také časy, ve kterých bude testování probíhat, například zdali bude testování probíhat v pracovních hodinách zaměstnanců klientské firmy. Dále dochází k definování testovaných oblastí tzv. „scope“, je sjednáno jakým způsobem bude testování probíhat, jaké informace budou mít penetrační testéři k dispozici, případně další doplňující detaily v závislosti na daném testování. Celá fáze je nesmírně důležitou součástí penetračního testování, byť bývá mnohdy podceňována [2].

1.1.2 Intelligence Gathering

Fáze Intelligence Gathering penetračního testování je věnována sběru informací a jejich následné analýze. Penetrační testéři se snaží získat co možná nejvíce informací o testovaném počítačovém systému. Tuto fázi lze rozdělit do tří kategorií:

- pasivní,
- semi-pasivní,
- aktivní.

Pasivní a semi-pasivní sběr informací jsou často souhrnně označovány jako *Open Source Intelligence* (OSINT). Při pasivním sběru informací jsou informace získávány bez interakce s testovaným systémem z volně dostupných a veřejně dostupných zdrojů. Mezi využívané zdroje patří webové stránky (např. diskuzní fóra), sociální média, veřejné záznamy (např. whois registr) a další dostupné informace.

Během semi-pasivního sběru informací jsou informace získávány metodami, které vypadají jako běžný internetový provoz a chování, například dotazy na DNS servery vyjma brute force² DNS dotazů. Během semi-pasivního sběru informací je klíčovým aspektem neupoutávat pozornost sledovaného subjektu.

Aktivní sběr informací získává informace pomocí interakce s testovaným počítačovým systémem. Typickým příkladem je skenování otevřených portů a následná aktivní enumerace služeb běžících na nalezených otevřených portech, dále pak tzv. fuzzování³ webových serverů apod. [3].

²Nemusí se jednat pouze o brute force útok, ale může jít také o slovníkový útok s příliš velkým a generickým slovníkem

³Pokusy o odhalení skrytých složek, souborů, nebo také subdomén

1.1.3 Threat Modeling

Fáze Threat Modeling penetračního testování se zaměřuje na vytváření seznamu aktiv a hrozeb, které aktivům hrozí. Fáze může být buď formální, nebo neformální. Formální varianta modelování hrozeb bývá součástí závěrečné zprávy z penetračního testování a může zákazníkovi poskytnout cenné informace pro řízení rizik. Neformální varianta slouží jako mentální kontrolní seznam testera [4].

1.1.4 Vulnerability Analysis

Ve fázi Vulnerability Analysis penetračního testování dochází k ověřování konfigurace jednotlivých služeb identifikovaných během fáze sběru informací, dále pak k identifikaci zranitelností. Může se jednat o veřejně známé zranitelnosti tzv. „*Common Vulnerabilities and Exposures* (CVE)“, nebo o zranitelnosti vyplývající z špatného návrhu služby či aplikace [5].

1.1.5 Exploitation

Fáze Exploitation penetračního testování se vyznačuje tím, že dochází k ověřování možnosti zneužití zranitelností, které byly identifikovány v předešlých fázích. Je při tom využíváno různých technik k obcházení implementovaných bezpečnostních mechanismů testovaného počítačového systému. Mnohdy opomíjenou oblastí bývá lidský faktor. Pro útočníka může být mnohem efektivnější se zaměřit na lidský faktor a pomocí tohoto vektoru útoku docílit přístupu do interní sítě. Je tedy důležité testovat také odolnost zaměstnanců dané organizace proti různým technikám sociálního inženýrství, například phishing, vishing, smishing apod. [6].

1.1.6 Post-Exploitation

Fáze Post-Exploitation má za úkol určit hodnotu kompromitovaného zařízení a udržet nad ním kontrolu pro pozdější využití. To označujeme jako persistenci. Penetrační tester má pak okamžitý přístup k počítačovému systému, aniž by musel znovu zneužít zranitelnost, která vedla k infiltraci zařízení. Hodnota zařízení je určena citlivostí uložených dat a jeho užitečností pro další kompromitaci sítě, např. řadič domény má velmi vysokou hodnotu, počítač běžného zaměstnance má hodnotu nižší.

Pokud nebylo smluveno jinak, neměl by penetrační tester modifikovat služby, které klient označil jako kritické. Důvodem modifikace nekritických služeb je především demonstrace dopadu zneužití dané zranitelnosti klientovi. Jedná se především o:

- Eskalaci privilegií.

- Získání přístup k citlivým datům.
- Znepřístupnění služeb oprávněným uživatelům, tzv. *Denial of Service* (DoS).

Jakékoliv změny v klientově infrastruktuře musí být pečlivě zdokumentovány, aby po ukončení testování mohly být tyto změny vráceny do původního stavu.

Po každé úspěšné kompromitaci počítačového systému opět nastává fáze sběru informací. Poté následují další fáze dokud není dosaženo vytyčeného cíle. Celý cyklus se může několikrát opakovat [7].

1.1.7 Reporting

Pravděpodobně nejvýznamnější fází penetračního testování je Reporting, neboli vytvoření závěrečné zprávy. Závěrečná zpráva je totiž to jediné, co klient z celého penetračního testování vidí.

Struktura závěrečné zprávy obsahuje dvě hlavní části:

- Executive Summary⁴.
- Technical Report⁵.

První část je zaměřena na cíle penetračního testování, způsob provedení, hodnocení rizika a závěry. Druhá část závěrečné zprávy poskytuje technické detaily k jednotlivým nálezům penetračního testu. Typicky je u každé identifikované zranitelnosti uveden výskyt, popis, *Proof of Concept* (PoC), dopad a kroky potřebné k odstranění dané zranitelnosti. Závěrečná zpráva by měla mít všechny náležitosti formálního dokumentu, tedy vhodně zvolené formátování, měla by být přehledná a neměla by obsahovat věcné či gramatické chyby, nebo překlepy [8].

1.2 Motivace penetračního testování

Úkolem penetračního testování je identifikovat bezpečnostní slabiny v počítačovém systému a zajistit jejich opravu předtím, než by mohl potenciální útočník využít těchto nedostatků. Nejčastějším terčem útoků jsou webové aplikace, často spojené s lidským faktorem. Termín „attack vector“ označuje cíle útoku v rámci penetračního testování, a jejich seskupením vzniká tzv. „attack surface“ neboli množina možných vektorů útoku.

1.3 Kategorie penetračního testování

Penetrační testování lze rozdělit do kategorií dle informací, které jsou penetračním testerům poskytnuty před začátkem penetračního testu na:

⁴Manažerské shrnutí

⁵Technické detaily

- Black Box testování.
- White Box testování.
- Gray Box testování.

Dalším možným dělením penetračního testování je dle umístění penetračního testera na:

- externí,
- interní.

1.3.1 Dle znalosti

Black Box penetrační test

Při Black Box penetračním testování začíná penetrační tester bez bližších informací o testovaných počítačových systémech. Sleduje reakce testovaného počítačového systému na různé vstupy, které zadává, a získává tak postupně informace o testovaném systému. Pokud je potřeba získat zdrojový kód aplikace, pokud se nejedná o open-source řešení, je jedinou možností jeho získání využití reverzního inženýrství.

Výhody Jedná se o věrnou simulaci ať už externího či interního útočníka útočníka. Ve srovnání s ostatními metodami jsou náklady na tuto metodu penetračního testování nižší.

Nevýhody Kvůli nedostatku počátečních informací není testování úplné. Některé zranitelnosti nemusí být v časové dotaci penetračního testování odhaleny. Tato metoda penetračního testování je závislá na kvalifikovaných odhadech penetračních testerů a metodě pokus-omyl. S tím bývá spojena i větší délka trvání penetračního testování [9][10][11].

White Box penetrační test

White Box metoda, často označovaná jako „Clear Box“ nebo „Glass Box“, umožňuje penetračnímu testerovi přístup ke všem informacím o testovaných počítačových systémech, včetně zdrojových kódů, konfiguračních souborů a vnitřního fungování aplikací.

Výhody Metoda penetračního testování poskytuje úplný výsledek testování. Další výhodou je možnost automatizace velké části penetračního testování.

Nevýhody Hlavní nevýhodou White Box metody penetračního testování je komplexita a časová náročnost. Dále jsou to pak vyšší náklady ve srovnání s Black Box a Gray Box metodami [9][10][11].

Gray Box penetrační test

Gray Box metoda penetračního testování je kompromisem mezi Black Box a White Box metodou. Tester má omezený přístup k informacím o konfiguraci, zdrojových kódech a síťové infrastruktuře. Typickým příkladem je testování, při kterém byl penetračnímu testerovi poskytnut běžný uživatelský účet. Testování probíhá obdobně jako u Black Box, ale často odstraňuje potřebu kvalifikovaných odhadů a metody pokus-omyl. Pokud má penetrační tester k dispozici část zdrojového kódu či některé konfigurační soubory, může do nich nahlédnout, pokud během testování narazí na zajímavou oblast. Tímto způsobem lze ušetřit čas, oproti Black Box metodě, odstraněním nutnosti kvalifikovaných odhadů.

Výhody Hlavní výhodou této metody penetračního testování je rychlost testování. Obecně lze říct, že kombinuje výhody Black Box a White Box metod a snaží se odstraňovat jejich nevýhody.

Nevýhody Nevýhodou Gray Box metody penetračního testování je omezená možnost automatizace. Dále pak nemožnost procházení celého zdrojového kódu, i v případě, kdy byl klientem poskytnut [9][10][11].

1.3.2 Dle umístění testera

Externí

Externí penetrační test, například infrastruktury, představuje snahu anonymního útočníka o proniknutí do firemní sítě z internetu. Jeho cílem je obejít bezpečnostní kontroly, využít zranitelnosti a získat neoprávněný přístup do sítě, případně narušit dostupnost poskytovaných služeb. Externí testy se zaměřují na oblasti jako DMZ, e-mail, DNS, web server a na něm běžící webové aplikace.

Interní

Cílem interního penetračního testu, například infrastruktury, je ověřit odolnost firemní sítě z vnitřní perspektivy. Tento typ testu se nejčastěji provádí z hlediska anonymního útočníka s fyzickým přístupem do vnitřní sítě bez možnosti přístupu k adresářovým službám nebo z pohledu zaměstnance s přístupem k adresářovým službám. Interní penetrační testy se zaměřují na prvky jako jsou servery ať už fyzické nebo virtuální, koncová zařízení a další softwarové služby, včetně interních webových serverů a databází.

2 Cyber Range platformy

Cyber Range platformy představují klíčový nástroj v oblasti vzdělávání a cvičení v kybernetické bezpečnosti. Platformy poskytují kontrolované, izolované a bezpečné prostředí, ve kterém je možné simulovat reálné kybernetické útoky a obranné reakce, což umožňuje praktické a efektivní vzdělávání v oblasti penetračního testování.

2.1 Cyber Range platformy ve výuce

Cyber Range platformy se staly nezbytným nástrojem pro vzdělávání v oblasti kybernetické bezpečnosti, zejména v rámci penetračního testování. Tyto platformy poskytují prostředí pro simulaci kybernetických útoků a obranných strategií, což umožňuje studentům a profesionálům získat praktické zkušenosti v boji s kybernetickými hrozbami. Cyber Range platformy zahrnují simulované sítě, systémy a aplikace, které umožňují studentům a profesionálům provádět penetrační testy a cvičit obranné strategie bez rizika poškození reálných systémů.

V rámci vývoje kybernetické bezpečnosti a bezpečného vývoje aplikací jsou rozlišovány tři základní týmy:

- Red Team.
- Blue Team.
- Yellow Team.

Yellow Team je zaměřen především na vývoj a architekturu softwaru, sítí a počítačových systémů. Typicky se jedná o programátory nebo síťové architektky.

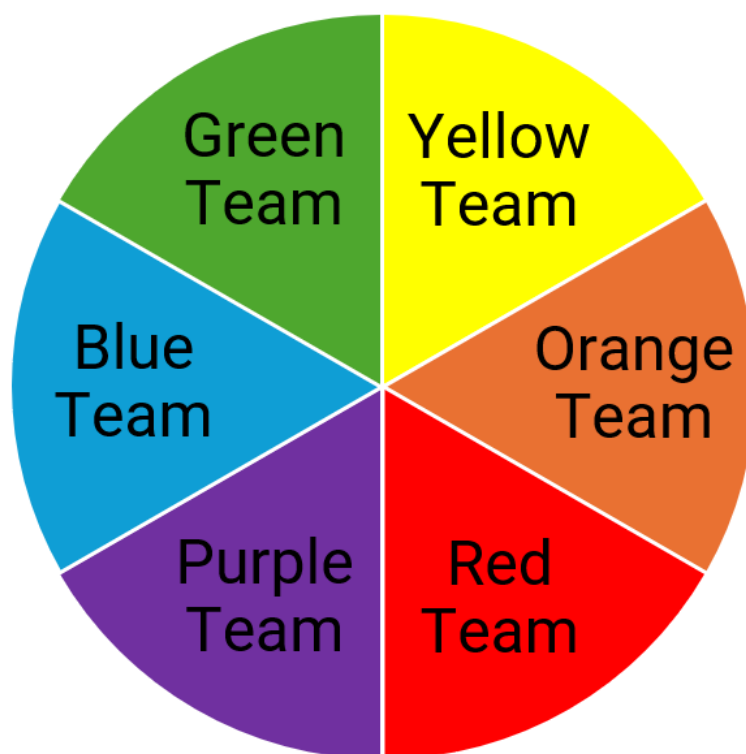
Red Team je tým, který simuluje útočníky a provádí kybernetické útoky na systémy a sítě. Tým je zaměřen na ofenzivní bezpečnost, především pak penetrační testování a etický hacking. Cílem Red Teamu je identifikovat slabá místa v zabezpečení a proniknout do cílových systémů. Naopak, Blue Team je tým, který simuluje obranu a reaguje na kybernetické útoky. Tým je zaměřen na defenzivní aspekt kybernetické bezpečnosti, například ochrana síťové infrastruktury a počítačových systémů, tzv. „Incident Response“ neboli reakce na bezpečnostní incidenty a v neposlední řadě forenzní analýzu. Cílem Blue Teamu je identifikovat a odvrátit kybernetické útoky, minimalizovat škody a zajistit bezpečnost počítačových systémů.

V rychle se rozvíjejícím a měnícím prostředí informačních technologií je velmi obtížné sledovat veškeré aktuální dění. Individuální zaměření jednotlivých týmů jsou natolik odlišná, že je pro jednotlivce nemožné vnímat a zpracovat veškeré dění v daném oboru. Proto vznikly týmy, které stírají rozdíly mezi jednotlivými základními týmy. Vznikly tak tři rozšiřující týmy na rozhraní původních týmů. Jedná se o:

- Orange Team.
- Green Team.

- Purple Team.

Orange Team vyplňuje prostor mezi Yellow Teamem a Red Teamem. Jeho hlavním úkolem je zvýšit povědomí o kybernetické bezpečnosti u členů z řad Yellow Teamu pomocí ukázek možností útoků a demonstrace dopadu kybernetických útoků. Green Team vyplňuje prostor mezi Yellow Teamem a Blue Teamem. Jeho hlavním úkolem je vzdělávat členy Yellow Teamu v oblasti bezpečného designu, logování, apod. Purple Team vyplňuje prostor mezi Red Teamem a Blue Teamem. Typicky provádí kybernetické útoky za účelem zdokonalení detekce těchto útoků, případně automatické mitigace takového útoku. Celé spektrum všech týmů je zachyceno na obrázku 2.1



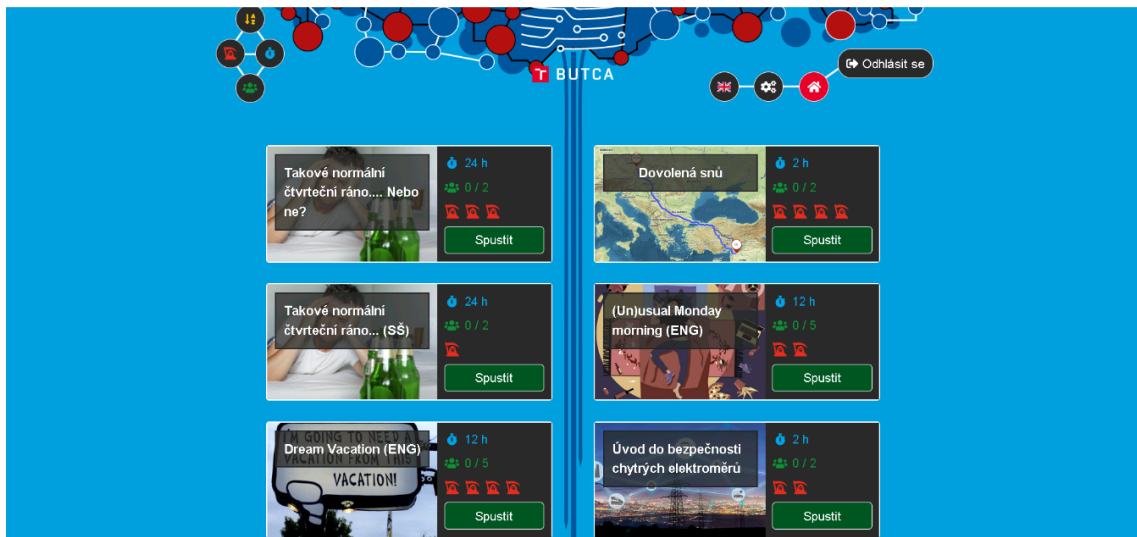
Obr. 2.1: Spektrum týmů v oblasti kybernetické bezpečnosti a vývoje

Cyber Range platformy poskytují ideální prostředí pro vzdělávání jak Red Teamu, tak Blue Teamu. Pro Red Team poskytují prostředí pro simulaci různých typů útoků a umožňují studentům zkoumat různé penetrační techniky a postupy. Studenti mohou provádět simulované útoky na virtuální systémy, a na nich běžících aplikace, a získávat tak cenné zkušenosti s identifikací a využitím zranitelností. Pro Blue Team poskytují Cyber Range platformy prostředí pro cvičení reakcí na kybernetické útoky a obranné strategie. Studenti mohou simulovat situace, ve kterých jsou jejich systémy napadeny, a praktikovat detekci, analýzu a odvracení útoků. Tím získávají dovednosti potřebné k efektivnímu zajištění a obraně sítě.

Cyber Range platformy jsou virtuální nebo fyzické prostředí, jež umožňují simulaci kybernetických útoků a obranných strategií. Tyto platformy poskytují širokou škálu scénářů a cvičení, které umožňují praktickou simulaci různých kybernetických hrozeb a situací. Mezi klíčové prvky Cyber Range patří simulační prostředí, které zahrnuje virtuální sítě, systémy a aplikace, a také nástroje pro monitorování a analýzu provozu.

2.2 BUTCA

Brno University of Technology Cyber Arena (BUTCA) je Cyber Range vyvinutá VUT v Brně. V současném stavu poskytuje několik herních scénářů zaměřených na vzdělávání a trénink Blue Teamu i Red Teamu. Ukázka webového rozhraní BUTCA platformy s výběrem scénářů je ilustrována na obrázku 2.2



Obr. 2.2: Prostředí výběru scénáře v platformě BUTCA

Scénáře jsou však vždy tematicky zaměřené pouze na ofenzivní nebo defenzivní schopnosti účastníků. Pro trénink penetračního testování a etického hackingu existují scénáře, které ověřují znalosti v oblasti testování webových aplikací, skenování zranitelností, používání veřejně dostupných zranitelností a vývoj vlastních skriptů k zneužití některých zranitelností, například zranitelnosti typu Buffer Overflow. Pro trénink defenzivních schopností existuje také několik scénářů, které pokrývají oblasti forenzní analýzy, nebo také analýzu vzorků škodlivého kódu - tzv. „malware analysis“. Každý scénář je však zaměřený pouze na ofenzivní nebo defenzivní schopnosti. Scénář představený v této diplomové práci je koncipován tak, aby umožňoval výuku a trénink zároveň Blue Team i Red Team schopností účastníků.

Účastníci scénáře budou rozděleni do dvou týmů, například o velikosti dvou hráčů. V závislosti na velikosti jednotlivých týmů budou vytvořeny zranitelné instance tak, aby každý člen týmu měl svou vlastní instanci. Každý tým bude mít administrátorský přístup na své vlastní instance. Jednotlivé týmy budou proti sobě soutěžit tak, že se budou snažit zneužít zranitelnosti nacházející se na jednotlivých instancích druhého týmu. Skórování jednotlivých týmů bude stejné jako u *Capture the Flag* (CTF) soutěží. Na zranitelných instancích budou umístěny vlajky („flags“), které budou sloužit jako důkaz o úspěšném průniku. Každá instance bude obsahovat právě dvě vlajky, první bude dostupná po kompromitaci účtu běžného uživatele, druhá bude dostupná po kompromitaci administrátorského účtu. Pokud se danému týmu podaří získat a odevzdat vlajku, získá body. Celý scénář bude probíhat v několika kolech, které na sebe budou plynule navazovat. Během jednoho kola je možné odevzdat každou vlajku pouze jednou. Při začátku nového kola budou vygenerovány nové vlajky, které budou následně rozmístěny na příslušné instance. Aby tým zabránil svým protivníkům v zisku bodů, bude muset identifikovat jakou zranitelnost protivníci zneužili, následně ji opravit. Při opravách však musí dbát na to, aby byla zachována funkcionality dané služby. Funkcionality služeb spuštěných na jednotlivých instancích bude kontrolovat Game Server. Pokud Game Server zjistí nedostupnost, nebo nedostatečnou funkcionality, některé ze služeb, bude tým, jenž tuto instanci spravuje, penalizován bodovou ztrátou. Kontroly provozuschopnosti jednotlivých instancí budou probíhat několikrát během hracího kola. Ztratit body za chybně opravenou instanci bude možné maximálně jednou za hrací kolo. Hra končí ve chvíli kdy skončí poslední herní kolo.

Scénář bude sloužit k výuce a tréninku jak ofenzivních, tak defenzivních schopností účastníků. V nastavení scénáře bude možné upravit bodové zisky za jednotlivé vlajky a také výši penalizace. Tím bude možné docílit většího zaměření na ofenzivní schopnosti v případě nízké penalizace a vysokých odměn za odevzdávání vlajek. To by mělo týmy motivovat strávit více času útočením na cizí instance. V případě většího zaměření na defenzivní schopnosti bude naopak vysoká penalizace a nízké bodové zisky za odevzdávání vlajek. Toto nastavení scénáře by mělo týmy motivovat k vyššímu zaměření na opravování chyb a případnému boji s perzistencí, kterou si útočníci vytvořili.

3 Zranitelnosti vlastní knihovny zranitelností

V následující kapitole budou popsány jednotlivé zranitelnosti, které jsou implementovány ve vlastní knihovně zranitelností. Zranitelnosti z této knihovny jsou pak použity na zranitelných instancích herního scénáře. Implementované zranitelnosti lze rozdělit na dvě kategorie. Těmito kategoriemi jsou:

- Initial Foothold.
- Privilege Escalation.

3.1 Initial Foothold

Tzv. „Initial Foothold“ do zranitelných instancí budou útočícím hráčům poskytovat zranitelnosti z této kategorie. Pojmem Initial Foothold jsou označovány zranitelnosti, které poskytnou útočícím hráčům prvotní přístup k počítačovému systému. Pro zjednodušení herního scénáře budou všechny zranitelnosti z této kategorie poskytovat účastníkům nějakou formou *Remote Code Execution* (RCE) - neboli vzdálené spuštění kódu. Po úspěšném zneužití zranitelnosti získají účastníci přístup na počítačový systém jako běžný uživatel.

V rámci této práce byly vybrány a implementovány čtyři Initial Foothold zranitelnosti:

- File Inclusion.
- *XML External Entity* (XXE).
- *Server-Side Template Injection* (SSTI).
- OS Command Injection.

3.1.1 File Inclusion

Popis

Jednou z použitých zranitelností je tzv. „File Inclusion“. Zranitelnost je způsobena špatnou manipulací se vstupními parametry, které jsou dále zpracovávány back-endovými jazyky jako jsou PHP, JavaScript, nebo Java. To umožňuje vytvářet dynamické webové stránky, snižuje celkovou velikost back-end skriptu a zjednodušuje kód. V těchto případech se parametry používají k určení, který zdroj bude zobrazen na webové stránce. Pokud nejsou tyto funkcionality bezpečně implementovány, může útočník manipulovat těmito parametry tak, aby zobrazil obsah jiného lokálního souboru, než bylo původně zamýšleno. Tato zranitelnost je označována tzv. *Local File*

Inclusion (LFI). V některých případech je dokonce možné načítat soubory ze vzdáleného umístění, v takovém případě se jedná o *Remote File Inclusion* (RFI).

Dopad

Zranitelnost RFI může mít dopad na důvěrnost, integritu i dostupnost. V závislosti na nastavení aplikace a prostředí ji lze zneužít ke spuštění škodlivého kódu, protože útočník má plnou kontrolu nad souborem, který webový server zpracovává. LFI zranitelnost poskytuje útočnickovi možnost číst lokální soubory, které mohou obsahovat citlivé informace. Může se jednat o konfigurační soubory, *Secure Shell* (SSH) klíče apod. Pro útočníka mohou být velmi zajímavými soubory logovací soubory. Při prvním pohledu by se mohlo zdát, že LFI, narozdíl od RFI, neposkytuje RCE. Existují však techniky, jež umožňují dosáhnout RCE pomocí LFI, jednou z takových technik je tzv. „Log poisoning“. Získá-li útočník správnou systémovou cestu k logovacím souborům webového PHP serveru, může pak poslat nevalidní HTTP požadavek obsahující PHP payload, tento PHP payload bude uložen v logovacím souboru webového serveru. Pak už jen útočnickovi stačí využít LFI zranitelnosti, která načte a spustí škodlivý kód z logovacího souboru.

Mitigace

Validace vstupu Prvním možným přístupem je považovat veškeré uživatelské vstupy jako nebezpečné a provádět validaci těchto dat. Doporučovanou strategií pro validaci uživatelských vstupů je tzv. „white list“, což je seznam všech povolených znaků, které se mohou vyskytnout v uživatelských datech. Vstupy, jež obsahují nepovolené symboly jsou zamítnuty.

Při provádění validace vstupů je nutno uvážit všechny potenciální vlastnosti vstupu, včetně jeho délky, typu vstupu, celého rozsahu povolených znaků, syntaxe, konzistence napříč souvisejícími poli a souladu s logikou dané aplikace. Jako příklad souladu vstupu s aplikační logikou uveďme příklad. Uvažujme vstup „lod“, který je syntakticky platný, protože obsahuje pouze alfanumerické znaky, ale není platný, pokud logika aplikace očekává, že vstup bude obsahovat pouze barvy jako „červená“ nebo „modrá“.

Obecně se doporučuje upřednostňovat „white list“ na úkor tzv. „black listu“. Nevýhodou black listu je, že opomenutím byť jen jediného škodlivého znaku, nebude aplikace tímto obranným mechanismem chráněna.

Při ověřování názvů souborů je doporučeno používat white listy, které omezují znakovou sadu. Pokud je to možné, je doporučeno povolit pouze jeden symbol tečky (.) ve jménu souboru, aby bylo zamezeno slabínám aplikace, jako je např. Relative

Path Traversal ¹. Dále je vhodné vyhnout se oddělovačům adresářů jako je např. symbol lomítka / a nebo zpětného lomítka \, což zabraňuje slabíně Absolute Path Traversal².

Důležité je také věnovat pozornost částem aplikace, jež odpovídá za filtraci, která odstraňuje potenciálně nebezpečné znaky ze vstupních dat. Pro demonstraci jsou uvedeny dva příklady nevhodného návrhu filtrace vstupních dat. V prvním příkladu aplikace filtruje symbol lomítka /, ale pokud souborový systém podporuje také použití symbolu zpětného lomítka \ jako oddělovače adresářů, pak je bezpečnost aplikace ohrožena. Ve druhém příkladu aplikace odstraňuje sekvence ../ ze vstupních dat. Pokud útočník vloží vstupní textový řetězec ../...//, pak dvě instance ../ budou odstraněny z původního textového řetězce, ale zbývající znaky textového řetězce budou stále tvořit textový řetězec ../.

Pokud jsou názvy souborů nebo URL předem známé, je vhodné vytvořit mapování na tyto zdroje a odmítat všechny ostatní vstupy. Například ID 1 -> „inbox.txt“; ID 2 -> „profile.txt“, atd. Funkce jako AccessReferenceMap³, jež je součástí *OWASP Enterprise Security API* (ESAPI), poskytuje takovou funkcionalitu.

Sandboxing a Jailing Dalším možným přístupem je spuštění aplikace v izolovaném prostředí, tzv. „sandbox“ nebo také „jail“. Obě tyto prostředí izolují běžící aplikace od operačního systému. Toto opatření však omezuje pouze dopad potenciálního útoku na operační systém, ostatní části postižené aplikace mohou být stále ohroženy. Příklady nástrojů pro sandboxing a jailing: Unix chroot jail, AppArmor a SELinux.

3.1.2 XML External Entity

Popis

Další použitou zranitelností je XXE. Tato zranitelnost postihuje webové aplikace, které zpracovávají neošetřená *Extensible Markup Language* (XML) data. Zranitelnost vzniká v případě, ve kterém aplikace umožňuje uživatelům vkládat XML dokumenty, vhodně neošetřuje tato vstupní data a následně zpracovává tato neošetřená data.

Přítomnost XXE zranitelnosti lze ověřit testovacím XML dokumentem uvedeným ve výpise 3.1. V hlavičce testovacího XML dokumentu je definována proměnná `xxe`, v samotném těle XML dokumentu jsou pak XML tagy „firstName“ a „lastName“.

¹<https://cwe.mitre.org/data/definitions/23.html>

²<https://cwe.mitre.org/data/definitions/36.html>

³<https://github.com/vishank848/owasp-esapi-java/blob/master/src/main/java/org/owasp/esapi/AccessReferenceMap.java>

První zmíněný tag obsahuje textový řetězec „John“, druhý zmíněný tag obsahuje proměnnou definovanou v hlavičce XML dokumentu, pokud by došlo ke zpracování XML dokumentu bude v jeho těle tato proměnná nahrazena textovým řetězcem „Doe“.

Výpis 3.1: Testovací XXE payload

```
<!--?xml version="1.0" ?-->
<!DOCTYPE replace [<!ENTITY xxe "Doe"> ]>
  <userInfo>
    <firstName>John</firstName>
    <lastName>&xxe;</lastName>
  </userInfo>
```

Výsledný XML dokument, by po zpracování aplikací, která obsahuje XXE zranitelnost, je vyobrazen ve výpise 3.2

Výpis 3.2: XML dokument zpracovaný zranitelnou aplikací

```
<!--?xml version="1.0" ?-->
  <userInfo>
    <firstName>John</firstName>
    <lastName>Doe;</lastName>
  </userInfo>
```

Dopad

Zranitelnost XXE může mít dopad na důvěrnost a dostupnost. Útočník může získat citlivá data, jako například konfigurační informace, přihlašovací údaje, nebo dokonce spustit útoky na vnitřní systémy. Tato zranitelnost může vést k odhalení citlivých informací, neoprávněnému přístupu a potenciálně i k úplnému vyřazení aplikace nebo systému z provozu.

Mitigace

Bezpečná konfigurace aplikace Proti XXE se lze bránit bezpečnou konfigurací webové aplikace. Platí zásada, že pokud aplikace danou funkcionalitu nepotřebuje ke svému provozu, je vhodné tuto funkcionalitu zakázat. V konfiguraci aplikace by mělo být zakázáno používání externích entit, obzvláště pak jedná-li se o data přijatá od uživatele. Aplikace by měla být konfigurována tak, aby odmítala externí entity.

Validace vstupů Dalším způsobem ochrany je validace a filtrování vstupních XML dat obdobným způsobem jako u 3.1.1. Aplikace by měla akceptovat pouze validní a důvěryhodné XML struktury.

3.1.3 Server Side Template Injection

Popis

SSTI je variantou tzv. „Code Injection“ útoku. Tento útok je možné provést, pokud webová aplikace používá templatovací engine ke vkládání nebo zpracování externě zadávaných uživatelských dat, ale buď neneutralizuje, nebo nesprávně neutralizuje speciální symboly nebo syntax, které mohou být interpretovány jako výraz templatovacího engine nebo jiné příkazy kódu při zpracování templatovacím engine.

Webové aplikace používající templatovací engine umožňují vývojářům vkládat do webových stránek nebo dokumentů externě zadávané vstupy, jako jsou zprávy nebo text, za účelem generování dynamického obsahu webové stránky, dokumentu atd. Nejčastěji používané templatovací engine jsou Twig, Jinja2, Pug, Java Server Pages, FreeMarker, Velocity, ColdFusion, Smarty a také PHP samotné. Některé *Content Management System* (CMS) také interně používají templatovací engine.

Templatovací engine mají často svůj vlastní jazyk pro interpretaci příkazů a výrazů. Pokud útočník může ovlivnit vstup do templatovacího engine před zpracováním těchto dat, může volat libovolné výrazy, a tedy provést injekční útok. Například v některých jazycích templatovacích engine může útočník vložit výraz `{{7*7}}`, který bude interpretován templatovacím engine a webová aplikace vrátí výsledek zadané operace, tedy 49. Syntaxe vkládaného výrazu závisí na použitém templatovacím engine.

Z povahy této zranitelnosti, tedy že vstup je renderován a zobrazen v odpovědi webové aplikace, lze zranitelnost SSTI snadno zaměnit se zranitelností *Cross-Site Scripting* (XSS), dokud není detailněji prozkoumána opravdová příčina této chyby.

Templatovací engine mohou být použity také na straně klienta, v takovém případě se jedná o *Client-Side Template Injection* (CSTI). Mechanismy útoku nebo postižené technologie se mohou lišit, ale původ chyby je v zásadě stejný.

Dopad

Zranitelnost SSTI může mít dopad na důvěrnost, integritu i dostupnost. Obdobně jako v případě zranitelnosti RFI ji lze v závislosti na nastavení aplikace a prostředí zneužít ke spuštění škodlivého kódu. Útočník, kterému se podaří úspěšně injektovat payload, který je poté interpretován templatovacím engine získává veškerá práva templatovacího engine. Může se jednat o čtení, modifikaci a odstraňování souborů, neoprávněnou manipulaci s daty, spuštění škodlivého kódu atd.

Mitigace

Validace vstupů Hlavním způsobem ochrany proti této zranitelnosti je validace a filtrování vstupních dat obdobným způsobem jako u 3.1.1. Aplikace by měla akceptovat pouze taková data, která nemohou být zneužita k vyvolání nestandardního chování aplikace po interpretaci těchto dat templatovacím enginem.

Sandboxing a omezený režim Ke zmírnění dopadů je vhodné zvolit takový templatovací engine, který nabízí možnost sandboxingu nebo omezeného režimu. V omezeném režimu jsou limitované možnosti templatovacích výrazů, funkcí a příkazů.

3.1.4 OS Command Injection

Popis

OS Command injection je zranitelnost, jež umožňuje spuštění libovolných příkazů v hostitelském operačním systému prostřednictvím zranitelné aplikace. Zranitelnost je přítomna tehdy, předá-li aplikace neošetřená vstupní uživatelská data systémovému shellu, který vykoná libovolný příkaz. Nebezpečný uživatelský vstup může být poskytnut prostřednictvím formulářů, cookie souborů, HTTP hlaviček atd. Při tomto útoku jsou uživatelem dodaná vstupní data obvykle spuštěna s právy zranitelné aplikace. Útoky typu OS Command injection jsou možné v důsledku nedostatečné validace uživatelských vstupů v kombinaci se špatným návrhem aplikace, například použití nebezpečných funkcí namísto využití existujícího API pro daný jazyk.

Je důležité rozlišovat mezi zranitelností OS Command Injection a zranitelností Code Injection. V případě zranitelnosti Code Injection je útočníkovi umožněno přidat vlastní kód daného programovacího jazyka, který je aplikací spuštěn. Při Command Injection útočník rozšiřuje funkčnost zranitelné aplikace, která provádí systémové příkazy. Útočník nemusí injektovat kód, aby dosáhl provádění systémových příkazů.

U zranitelnosti OS Command injection lze rozlišit dvě varianty. V první variantě má aplikace v úmyslu spustit jediný pevně stanovený program, nebo systémový příkaz, který je pod její vlastní kontrolou. Jako argumenty tohoto programu hodlá používat vstupy dodané uživateli zvenčí. PHP aplikace může například používat `system("nslookup <HOSTNAME>")` ke spuštění příkazu `nslookup` a umožňuje uživatelům zadat parametr `HOSTNAME`, který se použije jako argument předem definovaného příkazu. Útočníci nemohou zabránit spuštění `nslookup`. Pokud však aplikace neodstraní z argumentu `HOSTNAME` oddělovače příkazů, jako jsou `;` nebo `&&`, mohou

útočníci oddělovače do argumentů umístit, což jim umožní spustit vlastní program poté, co se vykoná příkaz `nslookup`.

Při druhé variantě aplikace přijímá vstup, který pak používá k úplnému výběru programu, či příkazu, který má být spuštěn. Aplikace celý tento příkaz jednoduše přeměruje operačnímu systému, a ten jej vykoná. Například aplikace může používat `exec(<COMMAND>)` k vykonání příkazu v proměnné `<COMMAND>`. Pokud je útočník schopen ovlivnit obsah proměnné `COMMAND`, může útočník spouštět libovolné příkazy nebo programy. Pokud je navíc příkaz prováděn pomocí funkcí jako jsou `exec()` nebo `CreateProcess()`, pak útočník nemusí kombinovat více příkazů do jednoho řádku.

Dopad

Úspěšné zneužití zranitelnosti OS Command Injection umožňuje útočníkovi spustit na serveru libovolné příkazy. To může vést k úplné kompromitaci zranitelného serveru, a tím pádem k narušení důvěrnosti, integrity či dostupnosti dat používaných aplikací.

Mitigace

Používání API funkcí Při vývoji aplikací by měly být využívány API funkce pro daný programovací jazyk, namísto přímého volání příkazů či programů operačního systému pomocí „nebezpečných funkcí“. Například má-li aplikace v programovacím jazyce Java poskytovat funkcionalitu odesílání mailů, je nevhodné používat funkci `Runtime.exec()`, aby spustila příkaz `mail`, místo toho je vhodné použít existující Java API - `javax.mail.*`. Pokud neexistuje žádné takové API pro daný programovací jazyk, pak je nutné filtrovat a validovat veškerá vstupní uživatelská data.

Nebezpečné funkce pro některé programovací jazyky:

Java

- `Runtime.exec()`

C/C++

- `system()`
- `exec()`
- `ShellExecute()`

Python

- `exec()`
- `eval()`
- `os.system()`
- `os.popen()`
- `subprocess.popen()`

- `subprocess.call()`

PHP

- `system()`
- `shell_exec()`
- `exec()`
- `proc_open()`
- `eval()`

Filtrování a validace vstupu Veškerá vstupní uživatelská data, by měla být považována za potenciálně škodlivá. Existují dva způsoby, kterými lze validovat vstupní data. Těmi jsou white list a black list. Black list kontroluje přítomnost zakázaných symbolů, pokud je zakázaný symbol nalezen, pak je buď odstraněn, nebo je celý vstup zamítnut. Problém black listu je, že je velice těžké obsáhnout všechny symboly, které mohou mít speciální význam v daném kontextu, navíc mohou existovat symboly se speciálním významem, které nebyly v době přípravy black listu veřejně známé. White list obsahuje pouze seznam povolených symbolů, pokud vstup obsahuje jiný symbol než je povoleno, pak je takový symbol odstraněn, nebo je celý vstup zamítnut.

Ve výpisech 3.3, 3.4 a 3.5 jsou uvedeny příklady black listů pro různé případy.

Výpis 3.3: Obecný black list

```
| ; & $ > < ' \ ! >> #
```

Výpis 3.4: Black list pro Windows

```
( ) < > & * ' | = ? ; [ ] ^ ~ ! . " % @ / \
: + , '
```

Výpis 3.5: Black list pro Linux

```
{ } ( ) > < & * ' | = ? ; [ ] $ - # ~ ! . "
% / \ : + , '
```

Oprávnění Webová aplikace by a její komponenty by měly být spuštěny pouze s takovými právy, jež potřebují pro své správné fungování. Omezit webové aplikace lze například pomocí již zmíněných metod Sandboxingu.

3.2 Privilege Escalation

Zranitelnosti z této kategorie budou útočícím hráčům umožňovat tzv. „Privilege Escalation“ neboli eskalaci privilegií z běžného uživatelského účtu na administrátorský účet. Po úspěšném zneužití zranitelnosti získají hráči administrátorský účet.

Pojmem Privilege Escalation jsou označovány takové zranitelnosti, které poskytnou útočícím hráčům možnost eskalovat privilegia běžného uživatelského účtu na administrátorský účet a plně tak kompromitovat daný systém.

V rámci práce byly vybrány a implementovány celkem tři Privilege Escalation zranitelnosti:

- SUID spustitelný soubor s relativními cestami.
- SUDO miskonfigurace proměných systému.
- SUDO obecná miskonfigurace.

3.2.1 SUID spustitelný soubor s relativními cestami

Popis

V souborovém systému se nachází spustitelný soubor s nastaveným *Set owner User ID* (SUID) bitem, který používá relativní cesty. SUID bit umožňuje ostatním uživatelům spustit daný soubor s právy vlastníka souboru. Zranitelnost spočívá v použití relativních cest ke spustitelným souborům v daném skriptu. Pokud jsou ve skriptu použity relativní cesty, například `ip a`, pak útočník může vytvořit soubor se škodlivým kódem, který se jmenuje stejně jako spustitelný soubor (nebo příkaz) volaný zranitelným skriptem. V tomto příkladě by vytvořil soubor `ip`. Poté stačí útočníkovi nastavit proměnou prostředí (environment variable) `PATH` tak, aby při spuštění zranitelného skriptu s nastaveným SUID bitem byl z `PATH` vybrán útočníkem vytvořený soubor se škodlivým kódem.

Dopad

Zranitelnost umožňuje útočníkovi spouštět škodlivý kód v kontextu uživatele, který je vlastníkem zranitelného skriptu.

Mitigace

Zranitelnost lze mitigovat používáním absolutních cest ke spustitelným souborům (příkazům) používaných v daném skriptu. Místo příkazu s relativní cestou `ip a` je nutné použít absolutní cestu k danému souboru, tedy `/usr/bin/ip a`.

3.2.2 SUDO miskonfigurace proměných systému

Popis

Zranitelnost vzniká špatnou konfigurací `/etc/sudoers` souboru. Ve výpise 3.6 je uveden příklad chybné konfigurace. Konfigurace je chybná z toho důvodu, že nejsou

resetovány všechny proměnné prostředí. Příkaz `systemctl status <service>` spustí tzv. „pager“ pokud by se celý výpis příkazu nevešel na jednu stránku terminálového okna. V tomto případě může útočník nastavit proměnné prostředí `SYSTEMD_PAGERSECURE=true` a `PAGER=/usr/bin/more`, spustit příkaz `sudo systemctl status ssh` tak, aby vynutil spuštění pageru, v tomto případě `more`. Z pageru je pak možné spustit interaktivní shell s elevovanými oprávněními.

Výpis 3.6: SUDO miskonfigurace systémových proměnných

```
---8<---
Defaults:user env_keep+=*PAGE*
user ALL=(root) NOPASSWD: /usr/bin/systemctl status ssh
---8<---
```

Dopad

Zranitelnost umožňuje útočnickovi spouštět škodlivý kód v kontextu uživatele, většinou se jedná o administrátorský účet, ale může být nastaveno i jinak.

Mitigace

Zranitelnost lze mitigovat bezpečnou konfigurací proměnných systému.

3.2.3 SUDO obecná miskonfigurace

Popis

Zranitelnost vzniká špatnou konfigurací `/etc/sudoers` souboru, konkrétně jde o příliš obecné nastavení příkazu, který může uživatel spustit jako jiný uživatel, většinou se jedná o administrátorský účet. Příklad chybné konfigurace je uveden ve výpise 3.7. Útočník může využít payload ze stránek <https://gtfobins.github.io> k elevaci privilegií.

Výpis 3.7: SUDO obecná miskonfigurace

```
---8<---
user ALL=(root) NOPASSWD: /usr/bin/sed
user ALL=(root) NOPASSWD: /usr/bin/at
---8<---
```

Dopad

Zranitelnost umožňuje útočnickovi spouštět škodlivý kód v kontextu uživatele, většinou se jedná o administrátorský účet, ale může být nastaveno i jinak.

Mitigace

Zranitelnost lze mitigovat regulárními výrazy v konfiguračním souboru `/etc/sudoers`. Regulární výrazy je nutné zkonstruovat tak, aby příkazy poskytovaly zamýšlenou funkcionalitu, ale aby je nebylo možné zneužít k eskalaci privilegií.

4 Implementace herního scénáře do platformy BUTCA

V kapitole bude popsán návrh jednotlivých komponent Game Serveru a následná implementace jednotlivých komponent. Game Server bude mít za úkol sestavit a spustit jednotlivé zranitelné instance, kontrolovat stav těchto instancí a vůbec celého scénáře. Dále bude hodnotit jednotlivé týmy a komunikovat s hráči - bude jim poskytovat informace o aktuálním stavu instancí, skóre, odevzdaných vlajkách, čísle herního kola a celkovému počtu herních kol.

4.1 Vlastní návrh scénáře

Jak již bylo uvedeno v kapitole 2.2, tak scénář bude sloužit k otestování, jak ofenzivních, tak i defenzivních schopností účastníků v oblasti kybernetické bezpečnosti. Účastníci scénáře budou rozděleni do dvou týmů, každý tým obdrží administrátorský přístup, prostřednictvím vzdáleného terminálu (SSH), na své zranitelné instance. Každému z účastníků bude k dispozici virtuální stroj s operačním systémem Kali Linux, který obsahuje předinstalované některé nástroje, jež může účastník využít k útokům na zranitelné instance druhého týmu. Počet zranitelných instancí se bude odvíjet od počtu účastníků v týmu. Na začátku scénáře budou účastníci seznámeni s jeho pravidly. Některá pravidla budou kontrolována automaticky. Nebude však možné automatizovaně kontrolovat dodržování všech pravidel. Na webovém rozhraní Game Serveru bude uveden seznam spuštěných zranitelných instancí, jejich stav a IP adresa.

Průběh scénáře bude rozdělen na jednotlivá dílčí kola, hrací kola na sebe budou plynule navazovat. Na začátku každého kola budou vygenerovány nové vlajky, které budou následně rozmístěny do jednotlivých zranitelných instancí. V každém kole bude možné zadat každou vlajku právě jednou. Tím bude zajištěno, že bránící tým bude mít čas na útok reagovat. Mimo časovač jednotlivých kol bude spuštěn druhý časovač nezávislý na časovači kol. Druhý časovač bude periodicky provádět kontrolu provozuschopnosti jednotlivých Docker kontejnerů.

Týmy budou soutěžit proti sobě. Tým, který bude mít na konci časového limitu více bodů zvítězí. Body budou týmy získávat odevzdáváním tzv. „vlajek“, obdobně jako je tomu u CTF soutěží. Všechny zranitelné instance budou obsahovat právě 2 vlajky. První bude možné získat po kompromitaci běžného uživatelského účtu, druhá vlajka pak bude účastníkům přístupná po kompromitaci administrátorského účtu. První vlajka bude vždy umístěna v domovském adresáři běžného uživatele, druhá v domovském adresáři administrátorského účtu. Celkový časový limit bude

rozdělen do několika kratších časových intervalů, na začátku každého z intervalů budou vygenerovány nové vlajky a budou umístěny na zranitelné instance. Hráči tak budou moct získat další body, pokud nebyla opravena zranitelnost, kterou použili k získání dané vlajky, nebo pokud nebyla odstraněna perzistence, kterou si do kompromitované instance zavedli.

Každý tým tedy bude muset zároveň zabezpečit své vlastní zranitelné instance a útočit na zranitelné instance druhého týmu. Úkolem každého týmu bude udržovat své instance v provozuschopném stavu. Provozeroschopnost bude ověřovat Game Server jednou ze svých komponent. Pokud Game Server zjistí závadu na některé z instancí, označí ji jako „nezdravou“¹ a strhne danému týmu body. Pokud členové týmu neodstraní závadu do dalšího hracího kola, budou jim body strhnuty znovu.

4.2 Vlastní návrh implementační strategie scénáře do BUTCA prostředí

V současném řešení umožňuje BUTCA vytvářet virtuální stroje², zdánlivě by tak implementace mohla být řešena velmi jednoduše, a to vytvořením daného počtu VM.

Problematická by však byla kontrola provozuschopnosti těchto instancí, pokud by kontrolní skript byl spuštěn na dané instanci. Bránící hráči by mohli modifikovat kontrolní skript, nebo odpozorovat časový interval odesílání kontrolních zpráv i jejich obsah a podvrhnout další odchozí zprávy, tak aby se instance jevila jako zdravá, přitom by skutečnost byla odlišná.

Teoretickým řešením by bylo vytvořit pseudo-administrátorský účet pro účastníky, tak aby nemohli modifikovat kontrolní skript, nebo podvrhnout kontrolní zprávy zasílané skriptem. Tento přístup by však mohl být limitující pro účastníky v tom smyslu, že by neměli plná administrátorská práva, což by omezovalo kreativitu hráčů při vytváření obran proti útokům. Navíc by tento přístup ani nebyl příliš podobný situacím z reálného světa. Dalším možným řešením tohoto problému by bylo kryptografickými prostředky ochránit zprávy proti jejich podvrhnutí. V případě využití symetrické kryptografie³ by skript musel mít přístup ke sdílenému tajnému klíči, v případě použití asymetrické kryptografie⁴ by skript musel mít přístup k privátnímu klíči. V obou těchto případech vyvstává problém, že by účastníci mohli získat klíče potřebné k podvrhnutí kontrolních zpráv.

¹unhealthy

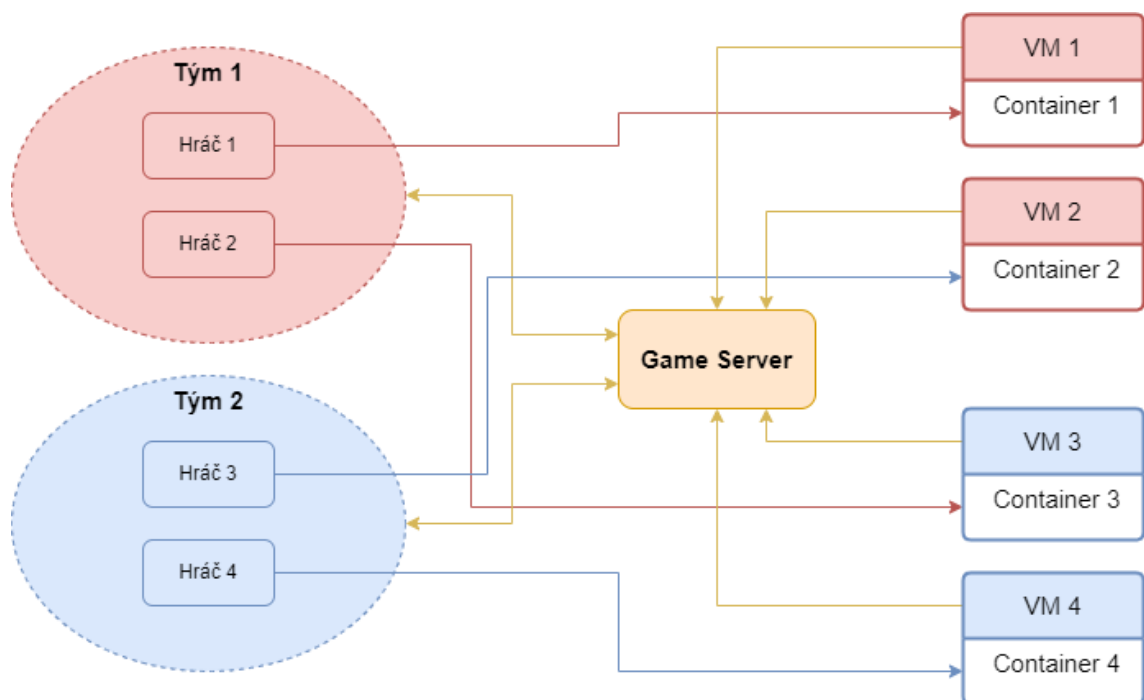
²dále jen *Virtual Machine* (VM)

³šifrování celých zpráv

⁴podpis odesílaných zpráv

Další možností by bylo kontrolní skript spustit na zařízení, které nemají účastníci pod kontrolou. kontrolní skript by se pomocí vzdáleného přístupu připojil na kontrolovanou instanci, provedl kontrolu a vyhodnotil výsledky těchto kontrol. Problematická by však byla situace, v níž se útočníci zmocní administrátorského účtu zranitelné instance. Aby se skript připojil na zranitelnou instanci musel by znát přihlašovací údaje administrátorského účtu a tyto údaje by se během celého scénáře nesměly změnit. Pokud by tedy útočníci získali přístup k administrátorskému účtu získali by persistenci, proti které by se bránci hráči nemohli bránit. Stejná situace by nastala při použití SSH klíčů.

Zajímavou myšlenkou je využití Docker kontejnerů pro tvorbu zranitelných instancí. Docker prostředí totiž umožňuje vykonávat příkazy z hostovacího systému ve zvoleném kontejneru, výsledky těchto příkazů - kontrolní zprávy - jsou pak vráceny do hostovacího systému. Neexistuje tak možnost podvržení kontrolních zpráv. Tato varianta tak umožňuje zpřístupnit bráncím hráčům administrátorský účet a zároveň neposkytuje výhodu „neodstranitelné persistence“ útočícím hráčům.



Obr. 4.1: Návrh implementace

Finální návrh je znázorněn na obrázku 4.1. Protože do současného stavu řešení platformy BUTCA je možné vkládat pouze VM, jsou zranitelné instance realizovány formou VM se zranitelným Docker kontejnerem. Účastníci scénáře budou mít přístup pouze do Docker kontejneru, zatímco VM bude provádět kontrolu stavu Docker kontejneru. Ve finálním návrhu jsou hráči rozděleni do dvou týmů, každý tým získá

od Game Serveru přihlašovací údaje ke svým zranitelným instancím a IP adresy zranitelných instancí druhého týmu. V grafickém znázornění návrhu jsou pro snadnější orientaci týmy a jejich zranitelné instance barevně rozlišeny. Tedy červený tým brání červené instance a útočí na modré, modrý tým naopak. Game Server a veškerá komunikace, která zajišťuje provoz scénáře, je označena oranžově, jde o kontrolní zprávy odesílané jednotlivými VM a o zprávy předávané Game Serverem týmům, například již zmíněné přihlašovací údaje k jednotlivým instancím, ale také informace o stavu jednotlivých instancí, bodovém stavu týmů a o tom, které vlajky týmy odevzdaly.

4.3 Návrhy a implementace jednotlivých komponent Game Serveru

V této části práce jsou popsány návrhy jednotlivých nástrojů pro provoz, kontrolu a údržbu scénáře. Všechny tyto nástroje budou součástí Game Serveru. Těmito nástroji jsou skript pro tvorbu zranitelných instancí, skript pro kontrolu provozuschopnosti zranitelných instancí a skript pro předávání informací hráčům, bodové hodnocení týmů a další provozní záležitosti. Všechny skripty je možné spustět samostatně, pro jednoduchost obsluhy byl vytvořen skript, který spustí jednotlivé dílčí skripty a předá jim patřičné argumenty. Jednotlivými dílčími skripty jsou:

- `deploy.py`,
- `logic.py`,
- `web.py`,
- `utils.py`,
- `run.py`.

Skript `utils.py` obsahuje různé podpůrné funkce používané napříč ostatními skripty. Skript `deploy.py` zodpovídá za vytvoření a spuštění Docker kontejnerů, jež budou používány v dané hře scénáře. Skripty `logic.py` a `web.py` slouží k provozu scénáře, hlídání pravidel a komunikaci scénáře s jeho účastníky. Skript `run.py` byl vytvořen jako jednotné rozhraní pro spuštění `deploy.py`, `logic.py` a `web.py` skriptů, je tedy možné celý scénář spustit jediným příkazem.

4.3.1 Dílčí nástroj pro tvorbu zranitelných instancí herního scénáře

Nástroj pro tvorbu zranitelných instancí, `deploy.py`, plní tři hlavní funkce:

- Vytvoření Dockerfile souboru.
- Sestavení Docker image.
- Spuštění jednotlivých Docker kontejnerů.

Skript vytvoří bezpečné heslo pro administrátorský účet a pro účet běžného uživatele, hesla pro administrátorské účty jsou uloženy do souborů, které provozovatel scénáře poté distribuuje jednotlivým týmům. Obsah ukázkového souboru s hesly pro červený a modrý tým je uveden ve výpise 4.1. Poté skript náhodně vybere zadaný počet zranitelností, z knihovny připravených zranitelností. Použití knihovny zranitelností zajišťuje modulárnost a škálovatelnost celého scénáře. Pro rozšíření scénáře stačí zvolit kategorii, Initial foothold nebo Privilege escalation, dle zvolené kategorie vytvořit složku s názvem zranitelnosti ve složce „Vulns“. Do této nově vytvořené složky pak stačí umístit konfigurační soubory zranitelné služby a soubory `specs.txt` se specifikacemi, které budou použity pro sestavení Dockerfile souboru, a `healthcheck.sh`, který bude kontrolovat provozuschopnost služby.

Skript pro tvorbu zranitelných instancí získá fragment konfiguračního Dockerfile souboru pro každou zranitelnost, každý fragment obsahuje instrukce pro zkopírování veškerých potřebných souborů pro konfiguraci a provoz zranitelných služeb. Skript zkombinuje tyto fragmenty, vloží je do výsledného Dockerfile souboru, v rámci tohoto Dockerfile souboru vytvoří účet běžného uživatele a nastaví heslo jak právě vytvořenému uživatelskému účtu, tak i administrátorskému účtu. Skript také s pravděpodobností $P = \frac{1}{2}$ upraví NGINX konfiguraci, tak aby umožňovala použití HTTP metody PUT, která umožňuje nahrávat soubory na webový server, tuto skutečnost pak mohou hráči využít ve svůj prospěch. Zároveň také upraví výchozí webovou stránku NGINX serveru tak, aby obsahovala všechny názvy dostupných vhostů. Na jednotlivých vhostech se budou nacházet zranitelné webové aplikace. Ostatní zranitelné aplikace, tedy jiné než webové, budou muset hráči nálezt pomocí enumeračních technik. Dále skript zkopíruje z adresáře zranitelností kontrolní skripty jednotlivých zranitelností, které byly pro daný Docker kontejner vybrány. Kontrolní skripty jsou pak umístěny do složky `Game files/checks/<ID>`, kde `<ID>` odpovídá číslu Docker image⁵.

Výpis 4.1: Ukázka souborů hesel administrátorských účtů obou týmů

```
# Game files/blue_passwords.txt
{
    'blue0': 'wVS8vpeHjEfLzoNliXCHNS6uRLhrt8aw',
    'blue1': 'iX76fQh0o8KFmCVkjYcy3mLwErLQqAPW'
}
# Game files/red_passwords.txt
{
    'red0': 'j06rK7PgMaPoyjMwn9SydR3pKNwqYJxH',
    'red1': 'z7ISVxcimcUqJhIYMoDT9v3NJAhnM95d'
}
```

⁵Např. Pro Docker image `butca_bg0` je ID rovno 0, pro Docker image `butca_bg1` je ID rovno 1, a tak dále.

Z takto sestaveného Dockerfile souboru vytvoří skript Docker image pomocí příkazu uvedeného ve výpise 4.2. Počet vytvořených Docker image odpovídá počtu hráčů v jednom týmu. Jako operační systém jednotlivých kontejnerů byl zvolen Ubuntu 22.04 jammy. V rámci sestavení Docker image jsou nainstalovány potřebné balíčky pro chod služeb a také je proveden příkaz `unminimize`, který provede proces inverzní k procesu minimalizace systému, protože direktiva `FROM ubuntu:jammy` v Dockerfile souboru provede načtení pouze minimalizované verze operačního systému Ubuntu 22.04 jammy.

Výpis 4.2: Příkaz k vytvoření Docker image

```
docker build -t <image name> .

# Příklad pro scénář hry 2 proti 2
docker build -t butca_bg0 .
docker build -t butca_bg1 .
```

Z vytvořeného Docker image jsou následně vytvořeny a spuštěny právě 2 Docker kontejnery, jeden pro modrý tým a druhý pro červený tým. Příkaz odpovědný za vytvoření spuštění Docker kontejnerů je uveden ve výpise 4.3. Přepínače `--ip` a `--network` přidělí každému kontejneru IP adresu a umístí jej do předem vytvořené sítě typu `macvlan`⁶. `Macvlan` síť byla zvolena, protože umožňuje umístit Docker kontejnery na stejnou síť, na které se nachází hostující systém. Pro ostatní stanice ve stejné síti se pak jednotlivé kontejnery jeví jako samostatné zařízení. Přepínače `--privileged` a `-d` zajišťují spuštění kontejneru na pozadí v privilegovaném režimu. Privilegovaný režim umožňuje spustit `systemd` jako hlavní program kontejneru a tím simulovat běžný počítač, dále jsou použity přepínače `--hostname` a `--name`, které nastavují jméno a `hostname` daného kontejneru. Situace po vytvoření a spuštění Docker kontejnerů je zachycena na obrázku 4.2.

Výpis 4.3: Příkaz k vytvoření Docker kontejnerů

```
docker run --ip <IP> --network macvlan_network \
  --privileged -d --name <container name> \
  --hostname <container name> <image name>

# Příklad pro scénář hry 2 proti 2
docker run --ip 192.168.67.33 --network macvlan_network \
  --privileged -d --name red0 \
  --hostname red0 butca_bg0

docker run --ip 192.168.67.34 --network macvlan_network \
```

⁶Popis sítě typu `macvlan` se nachází v oficiální dokumentaci dostupné z <https://docs.docker.com/network/drivers/macvlan/>. V dokumentaci jsou rovněž uvedeny příklady příkazů k vytvoření sítě typu `macvlan`

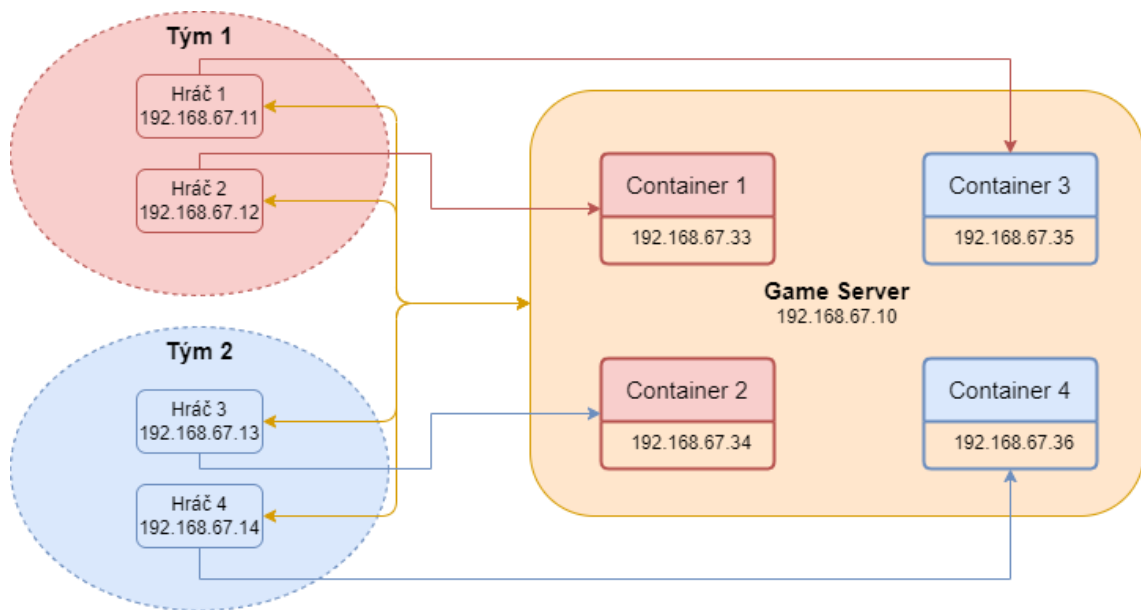
```

--privileged -d --name blue0 \
--hostname blue0 butca_bg0

docker run --ip 192.168.67.35 --network macvlan_network \
--privileged -d --name red1 \
--hostname red1 butca_bg1

docker run --ip 192.168.67.36 --network macvlan_network \
--privileged -d --name blue1 \
--hostname blue1 butca_bg1

```



Obr. 4.2: Docker kontejnery s typem sítě macvlan

Ve výpise 4.4 je zachycena nápověda pro spuštění nástroje pro tvorbu zranitelných instancí herního scénáře. Příkazem `python deploy.py` lze snadno spustit tento nástroj ve výchozím nastavení, tedy:

- 1 proti 1,
- se 3 zranitelnostmi typu Initial foothold,
- se 2 zranitelnostmi typu Privilege escalation.

Pro změnu nastavení nástroje pro tvorbu zranitelných instancí herního scénáře lze využít příslušné přepínače při volání skriptu. Ve výpise 4.5 je uveden příkaz pro spuštění nástroje s nastavením hry 2 proti 2 se 4 zranitelnostmi typu Initial foothold a 3 zranitelnostmi typu Privilege escalation.

Výpis 4.4: Nápopvěda nástroje pro tvorbu zranitelných instancí

```
$ python deploy.py -h
usage: deploy.py [-h] [--team-size INT] [--foothold-count INT]
[--privesc-count INT]

options:
-h, --help            show this help message and exit
--team-size INT
Number of players in each team (default: 1)
--foothold-count INT
Number of foothold paths (default: 3)
--privesc-count INT
Number of privesc paths (default: 2)
```

Výpis 4.5: Ukázka použití nástroje pro tvorbu zranitelných instancí

```
$ python deploy.py --team-size 2 --foothold-count 4 --privesc-
count 3
```

4.3.2 Dílčí nástroj pro provoz herního scénáře

Nástroj pro provoz scénáře, `logic.py`, plní dvě hlavní funkce:

- Obměna vlajek.
- Kontrola provozuschopnosti jednotlivých kontejnerů.

První zmíněná funkce, je prováděna na začátku každého kola, skript vygeneruje nové vlajky a umístí je do souborů `/root/root.txt` a `/home/john/user.txt` pro administrátorský účet, respektive účet běžného uživatele, v jednotlivých Docker kontejnerech. Vlajky jsou náhodné textové řetězce složené z pouze hexadecimálních znaků, celková délka textového řetězce s vlajkou je 30 znaků. Poté co jsou vlajky umístěny do jednotlivých Docker kontejnerů jsou uloženy do složky se soubory pro aktuální hru scénáře do souboru s vlajkami. Struktura souboru s vlajkami je zobrazena ve výpise 4.6. Název vlajky je vždy složen z jména daného Docker kontejneru a typu vlajky, buď se jedná o „user“ v případě vlajky získané kompromitací účtu běžného uživatele, nebo o „root“ v případě vlajky získané kompromitací administrátorského účtu.

Výpis 4.6: Ukázka souboru s vlajkami

```
{
  'red0_user': 'f5cbda1aee8fb14c3ea295c761a684',
  'red0_root': '236663eb5870b22cfe3abd38c0f012',
  'blue0_user': 'c6c0b2034ae4026dbd22cf0e1a0c53',
  'blue0_root': 'd536563b7c89a3c091836b476b3eef',
  'red1_user': 'b0807affc76409770354ca65bd7fda',
```

```
'red1_root': '8a61a10e40017c34c518c94f3c808b',
'blue1_user': '5d9c427065038f5d348bb0be864bb9',
'blue1_root': 'b35fafc9122a86277d1d31767b6e71'
}
```

Druhá hlavní funkce skriptu, tedy kontrola provozuschopnosti jednotlivých Docker kontejnerů, je prováděna nezávisle na délce probíhajících kol. Je žádoucí, aby tato kontrola proběhla minimálně dvakrát za kolo, aby bránící tým dostal zpětnou vazbu, jestli jejich řešení, které má útoku zabránit, je v souladu s pravidly. Kontrola provozuschopnosti jednotlivých Docker kontejnerů probíhá na pozadí, hráči tak explicitně neznají jaké kontroly skript provádí. Za pomoci nástroje `pspy`⁷ je možné získat představu o tom, jaké kontroly jsou prováděny, a podle toho mohou účastníci scénáře vytvořit obranné mechanismy, které mitigují zranitelnosti, jež se na daném Docker kontejneru nacházejí. Obecně však platí, že není nutné znát přesné kontroly prováděné na jednotlivých Docker kontejnerech, při opravách zranitelných aplikací stačí zachovat jejich funkcionalitu.

Kontrola provozuschopnosti jednotlivých Docker kontejnerů probíhá ve dvou fázích. V první fázi jsou zkontrolovány vlajky na jednotlivých Docker kontejnerech. Je kontrolováno, že se vlajky nachází v předem vyhrazených souborech. Jedná se o soubory `/root/root.txt` pro vlajku administrátorského účtu a `/home/john/user.txt` pro vlajku účtu běžného uživatele. Dále je kontrolováno, že vlajky nebyly modifikovány a také to, že jsou správně nastavena oprávnění na souborech s vlajkami. Ve druhé fázi probíhá kontrola jednotlivých služeb běžících na daném Docker kontejneru. Tato kontrola je realizována `bash` skripty, které jsou při vytváření zranitelných Docker kontejnerů umístěny do složky `Game files/checks/<ID>`. Tyto skripty jsou pak spuštěny v patřičných Docker kontejnerech. Na základě výstupu těchto skriptů je pak rozhodnuto, jestli je daný Docker kontejner „zdravý“. Pokud je zjištěno, že některá ze služeb není schopna vykonávat svou funkcionalitu je prohlášena za „nezdravou“ a danému týmu jsou za tento Docker kontejner strženy body. Body mohou být strženy každému týmu pouze jednou za hrací kolo za Docker kontejner.

Ve výpise 4.7 je zachycena nápověda pro spuštění skriptu pro provoz scénáře. Příkazem `python logic.py` lze snadno spustit tento nástroj ve výchozím nastavení, tedy:

- 1 proti 1,
- 11 hracích kol, každé o délce 5 minut,
- s kontrolou provozuschopnosti zranitelných instancí každé 2 minuty.

Pro změnu nastavení nástroje pro provoz scénáře lze využít příslušné přepínače při volání skriptu. Ve výpise 4.8 je uveden příkaz pro spuštění nástroje s nastavením

⁷Dostupného z <https://github.com/DominicBreuker/pspy>

hry 2 proti 2 na 10 kol (každé o délce 4 minut) s kontrolou provozuschopnosti zranitelných instancí každou jednu minutu.

Výpis 4.7: Návod nástroje pro provoz scénáře

```
$ python logic.py -h
usage: logic.py [-h] [--rounds INT] [--round-duration INT] [--health-check-sleep INT] [--team-size INT]

options:
-h, --help            show this help message and exit
--rounds INT          Number of game rounds (default: 11)
--round-duration INT  Duration of single round (in seconds) (default: 300)
--health-check-sleep INT
Sleep time for health check (in seconds) (default: 120)
--team-size INT       Number of players in each team (default: 1)
```

Výpis 4.8: Ukázka použití nástroje pro provoz scénáře

```
$ python logic.py --team-size 2 --rounds 10 --round-duration 240 --health-check-sleep 60
```

4.3.3 Dílčí nástroj pro komunikaci mezi hráči a herním scénářem

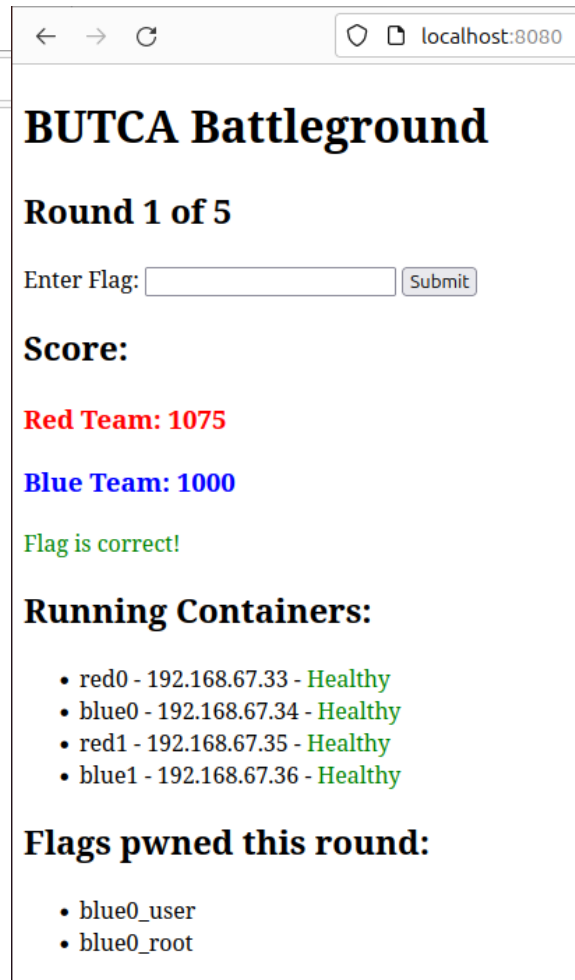
V současném stavu řešení platforma BUTCA vyžaduje v každém scénáři předem definované otázky a odpovědi, což nevyhovuje dynamické povaze scénáře. Proto byl navržen a vyvinut nástroj, který bude sloužit pro komunikaci hráčů a scénáře. Komunikace bude probíhat pomocí jednoduchého webového serveru hostujícího jednoduchou webovou stránku, jež bude obsahovat informace o aktuálním skóre jednotlivých týmů, seznam vlajek zadaných v daném kole, počítadlo herních kol, informace o stavu jednotlivých Docker kontejnerů a jejich IP adresy. Dále bude webová stránka obsahovat pole pro zadávání vlajek. Webová stránka bude dostupná na portu 8080 Game serveru.

Ukázka webové stránky při spuštění scénáře je zachycena na obrázku 4.3, poté co červený tým zadá vlajky, které získal kompromitací jednoho z Docker kontejnerů modrého týmu nastane situace, jež je zachycena na obrázku 4.4. Na obrázku 4.5 je zachycena situace, v níž je jeden z Docker kontejnerů modrého týmu prohlášen za „nezdravý“ a tento tým je penalizován ztrátou bodů. Na začátku dalšího hráčského kola je vyprázdněn seznam s odevzdanými vlajkami a je tak možné odevzdat nově vygenerované vlajky a získat tak další body. Situace po resetování vlajek je zobrazena na obrázku 4.6. Poté, co modrý tým opraví pokažený Docker kontejner

a proběhne kontrola provozuschopnosti, je zranitelná instance opět prohlášena za zdravou a modrý tým tak přestává být bodovou ztrátou penalizován. Situace je zachycena na obrázku 4.7. Po uplynutí všech hracích kol je zobrazen výsledek scénáře, který je možné pozorovat na obrázku 4.8.



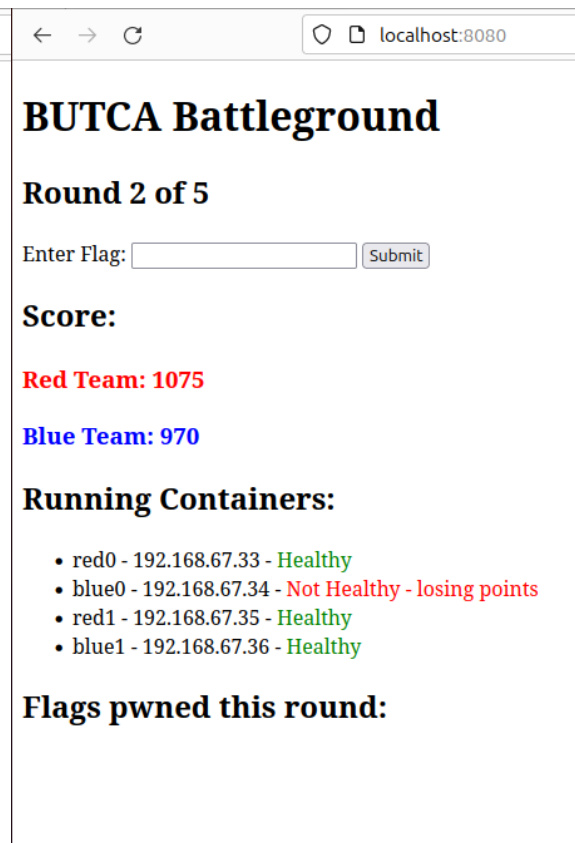
Obr. 4.3: Webové rozhraní po spuštění scénáře



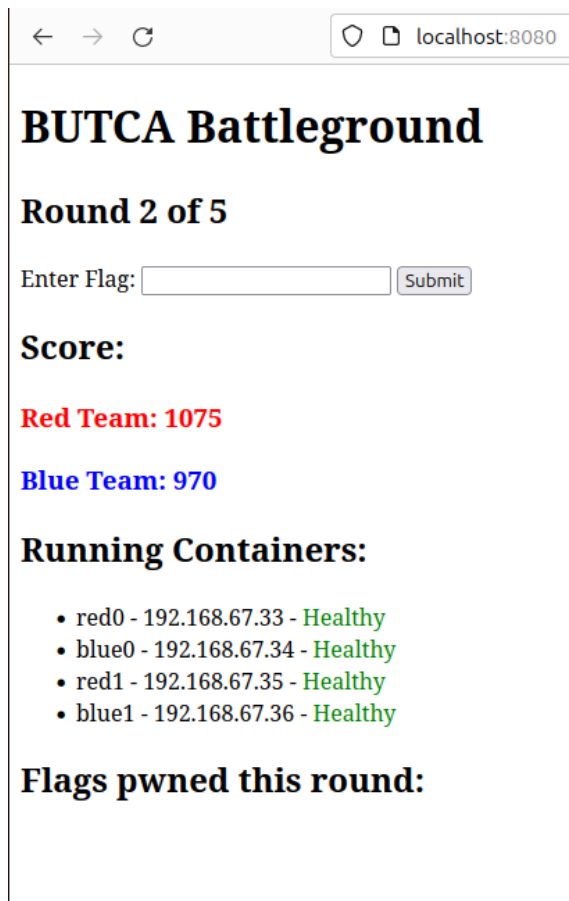
Obr. 4.4: Webové rozhraní po zadání vlajek



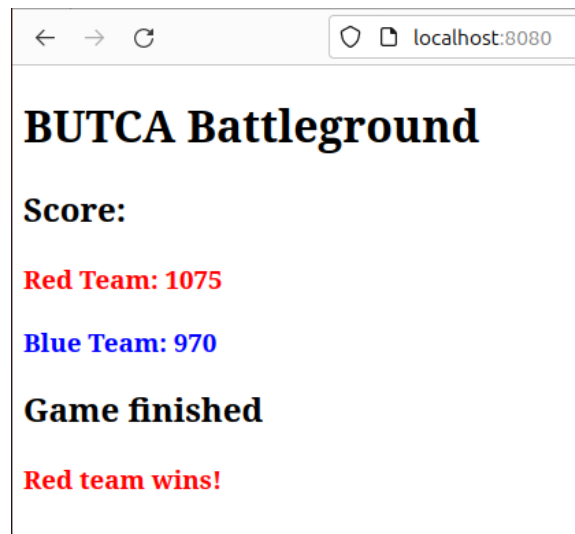
Obr. 4.5: Webové rozhraní při zjištění „nezdravé“ instance



Obr. 4.6: Webové rozhraní v následujícím kole



Obr. 4.7: Webové rozhraní po opravění „nezdravé“ instance



Obr. 4.8: Webové rozhraní po ukončení scénáře

Ve výpise 4.9 je zachycena nápověda pro spuštění skriptu pro komunikaci mezi hráči a herním scénářem. Příkazem `python web.py` lze snadno spustit tento nástroj ve výchozím nastavení, tedy:

- 11 hracích kol, každé o délce 5 minut,
- s bodovým ohodnocením 50 bodů za vlajku administrátorského účtu a 25 bodů za vlajku běžného uživatelského účtu,
- s penalizací 15 bodů za špatný stav zranitelné instance,
- s počátečním skóre 1000 bodů.

Pro změnu nastavení nástroje pro komunikaci mezi hráči a herním scénářem je možné použít příslušné přepínače při volání skriptu. Ve výpise 4.10 je uveden příkaz pro spuštění webového rozhraní na 10 kol o délce 4 minut, s bodovým ziskem 100 bodů za vlajku administrátora a 50 bodů za vlajku běžného uživatele, penalizací 35 bodů a počátečním skóre 1500.

Výpis 4.9: Náповěda nástroje pro komunikaci mezi hráči a herním scénářem

```
$ python web.py -h
usage: web.py [-h] [--rounds INT] [--root-points INT] [--user-
points INT] [--penalization INT] [--round-duration INT]
           [--initial-score INT]

options:
  -h, --help            show this help message and exit
  --rounds INT          Number of game rounds (default: 11)
  --root-points INT     Points for submitting root flag (
default: 50)
  --user-points INT     Points for submitting user flag (
default: 25)
  --penalization INT    Penalization for un-healthy machine (
default: 15)
  --round-duration INT  Duration of single round (in seconds) (
default: 300)
  --initial-score INT   Initial score of each team (default:
1000)
```

Výpis 4.10: Ukázka použití nástroje pro komunikaci mezi hráči a herním scénářem

```
$ python web.py --rounds 10 --round-duration 240 --root-points
100 --user-porints 50 --penalization 35 --initial-score 1500
```

4.3.4 Nástroj pro spuštění dílčích nástrojů herního scénáře

Nástroj pro spuštění dílčích nástrojů herního scénáře poskytuje sjednocené rozhraní pro spuštění všech dílčích skriptů a tím zajišťuje spuštění celého herního scénáře. Zpráva s nápovědou je zobrazena ve výpise 4.11. Příkazem `python run.py` lze snadno spustit celý scénář ve výchozím nastavení, tedy:

- 1 proti 1,
- 11 hracích kol, každé o délce 5 minut,
- s kontrolou provozuschopnosti zranitelných instancí každé 2 minuty,
- s bodovým ohodnocením 50 bodů za vlajku administrátorského účtu a 25 bodů za vlajku běžného uživatelského účtu,
- s penalizací 15 bodů za špatný stav zranitelné instance,
- s počátečním skóre 1000 bodů,
- se 3 zranitelnostmi typu Initial foothold,
- se 2 zranitelnostmi typu Privilege escalation.

Pro změnu nastavení herního scénáře lze využít příslušné přepínače při volání skriptu. Ve výpise 4.12 je uveden příkaz pro spuštění celého scénáře s nastavením hry 2 proti 2 na 10 kol, každé o délce 4 minut, s kontrolou zranitelných instancí

každou jednu minutu, s bodovým ziskem 100 bodů za vlajku administrátora a 50 bodů za vlajku běžného uživatele, s penalizací 35 bodů, s počátečním skóre 1500, 4 zranitelnostmi typu Initial foothold a 3 zranitelnostmi typu Privilege escalation.

Výpis 4.11: Náповěda nástroje pro spuštění dílčích nástrojů herního scénáře

```
$ python run.py -h
usage: run.py [-h] [--team-size INT] [--rounds INT] [--round-
duration INT] [--health-check-sleep INT] [--root-points INT]
           [--user-points INT] [--penalization INT] [--
initial-score INT] [--foothold-count INT] [--privesc-count INT]

options:
  -h, --help            show this help message and exit
  --team-size INT       number of players in each team (default
: 1)
  --rounds INT          number of game rounds (default: 11)
  --round-duration INT  duration of single round (in seconds) (
default: 300)
  --health-check-sleep INT
                        sleep time for health check (in seconds
) (default: 120)
  --root-points INT     points for submitting root flag (
default: 50)
  --user-points INT     points for submitting user flag (
default: 25)
  --penalization INT    penalization for un-healthy machine (
default: 15)
  --initial-score INT  initial score of each team (default:
1000)
  --foothold-count INT  Number of foothold paths (default: 3)
  --privesc-count INT  Number of privesc paths (default: 2)
```

Výpis 4.12: Ukázka použití nástroje pro spuštění dílčích nástrojů herního scénáře

```
$ python run.py --team-size 2 --rounds 10 --round-duration 240
--health-check-sleep 60 --root-points 100 --user-points 50 --
penalization 35 --initial-score 1500 --foothold-count 4 --
privesc-count 3
```

5 Testování scénáře

V této kapitole bude popsáno testování vytvořeného herního scénáře pro platformu BUTCA, dále budou v této kapitole uvedeny výsledky provedeného testování.

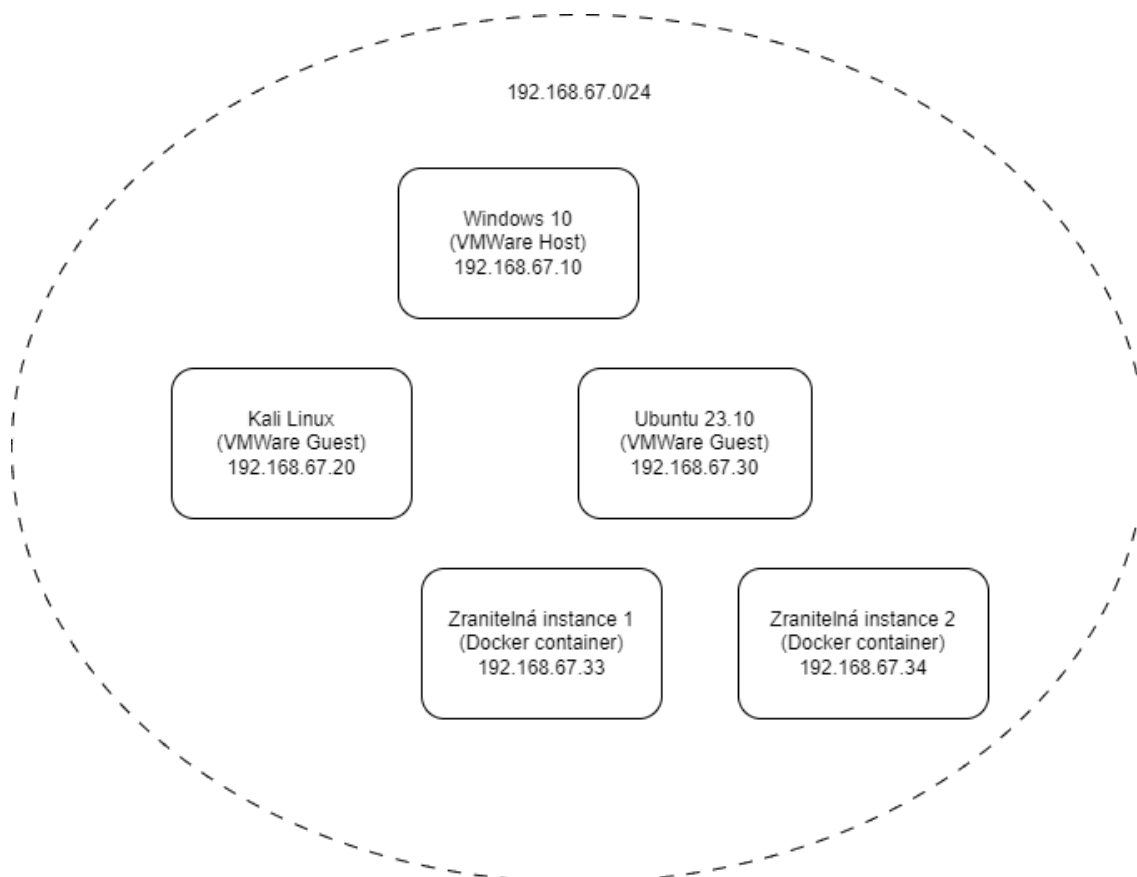
5.1 Lokální testovacího prostředí

Za účelem průběžného testování bylo vytvořeno lokální testovací prostředí, jež sestávalo z virtuálního stroje Linux Ubuntu 23.10, který plnil roli Game Serveru, a virtuálního stroje Kali Linux, který zastával roli stroje, jenž bude účastníkům k dispozici.

Testovací prostředí bylo vytvořeno lokálně pomocí virtualizačního hypervisoru VMWare. V rámci lokálního testovacího prostředí byl vytvořena virtuální síť, do níž byly umístěny oba virtuální stroje používané pro testování vlastního herního scénáře. Schéma zapojení jednotlivých částí lokálního testovacího prostředí je zachyceno na obrázku 5.1. V diagramu je vyobrazena lokální síť 192.168.67.0/24, ve které se nachází počítač s operačním systémem Windows 10, na kterém je spuštěn virtualizační hypervisor VMWare, s IP adresou 192.168.67.10. Dále se v této síti nachází oba virtuální stroje - Kali Linux s IP adresou 192.168.67.20 a Ubuntu 23.10 s IP adresou 192.168.67.30. Spuštěné Docker kontejnery, neboli zranitelné instance herního scénáře, se nachází na IP adresách 192.168.67.33 a 192.168.67.34. Docker kontejnery zranitelných instancí se díky Docker síti typu `macvlan` jeví ostatním zařízením na síti jako další unikátní zařízení. Ve skutečnosti však virtuální stroj Ubuntu 23.10 (Game Server), provádí transparentní směrování na příslušné IP adresy jednotlivých kontejnerů na základě jejich MAC adres. Pozice jednotlivých bloků reprezentující počítačové systémy umístěné v síti naznačuje zanoření virtualizace. Počítačový systém Windows 10 je fyzickým počítačem, počítačové systémy Ubuntu 23.10 a Kali Linux jsou stroje virtualizované pomocí virtualizačního hypervisoru VMWare a počítačové systému jsou Docker kontejnery běžící na virtuálním stroji Ubuntu 23.10. V takto vytvořeném lokálním testovacím prostředí byly vyvinuty a odladěny všechny jednotlivé komponenty Game Serveru. Po dokončení vývoje byl virtuální stroj Ubuntu 23.10 exportován a nasazen do prostředí platformy BUTCA.

5.2 Testování v BUTCA prostředí

Testování implementovaného řešení scénáře do prostředí BUTCA platformy probíhalo ve dvou fázích. V první fázi testovali herní scénář profesionální penetrační testeři, ve druhé fázi testovali herní scénář studenti oboru Informační bezpečnost z Fakulty elektrotechniky a komunikačních technologií Vysokého učení technického



Obr. 5.1: Schéma lokálního testovacího prostředí

v Brně. V obou fázích byli účastníci nejprve seznámeni s průběhem scénáře a s jeho pravidly, během toho měli prostor pro seznámení se s poskytnutými Kali Linux VM a jejich nastavením. Pravidla jsou uvedena na obrázku 5.2

V první fázi testování bylo nastavení scénáře následovné:

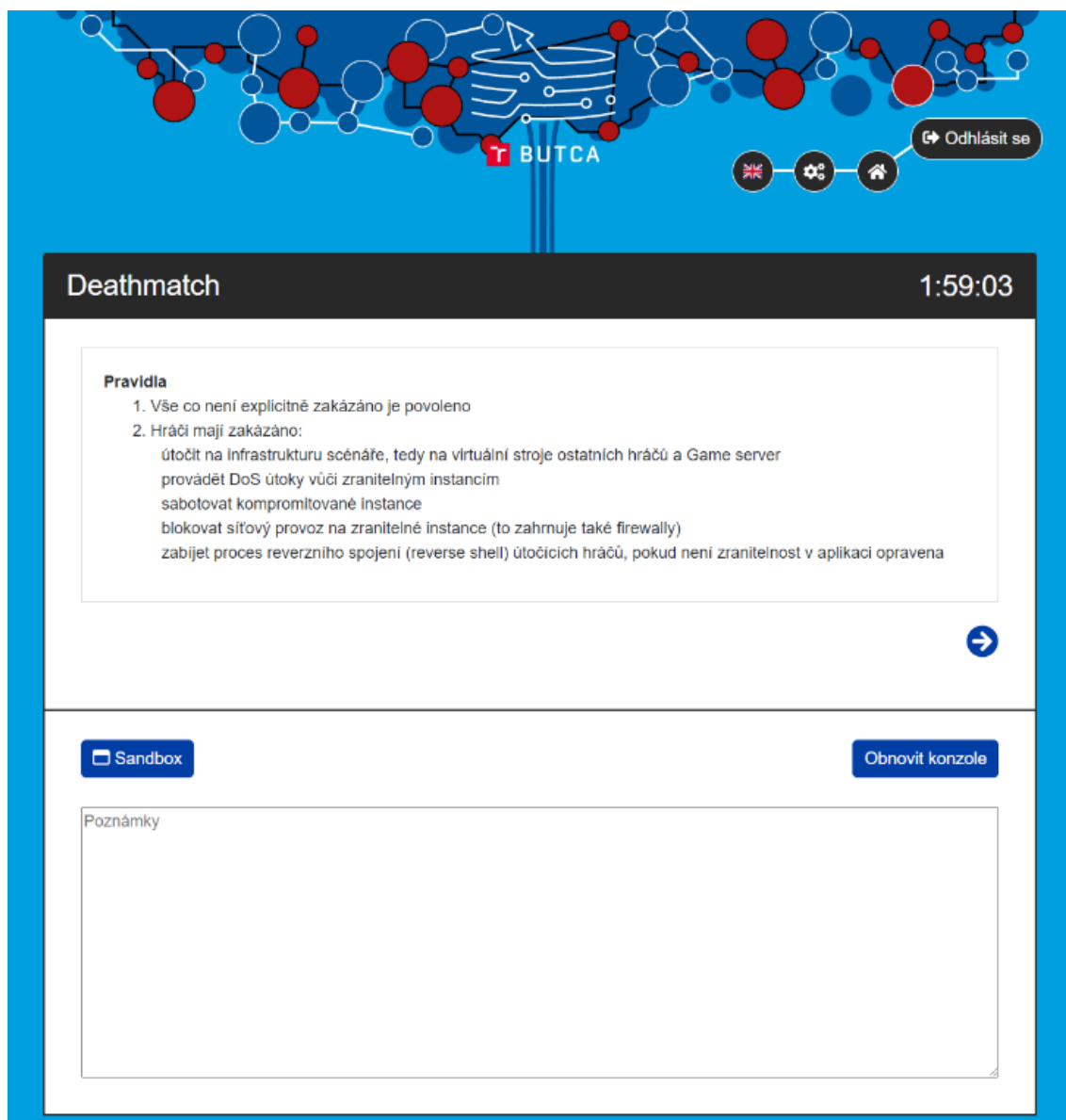
- 1 proti 1,
- 12 hracích kol, každé o délce 5 minut,
- kontrola provozuschopnosti zranitelných instancí každé 2 minuty,
- bodové ohodnocení 50 bodů za vlajku administrátorského účtu a 25 bodů za vlajku běžného uživatelského účtu,
- penalizace 15 bodů za špatný stav zranitelné instance,
- počáteční skóre 1000 bodů,
- 4 zranitelnosti typu Initial foothold,
- 3 zranitelnosti typu Privilege escalation.

Zranitelné instance tedy obsahovaly všechny zranitelnosti z vlastní knihovny zranitelností, aby došlo k otestování všech těchto zranitelností. Během této fáze testování byl odhalen nedostatek, který způsoboval, že nebylo možné komunikovat ani z jedné VM Kali Linux účastníků s žádnou ze zranitelných instancí spuštěných na Game Serveru. Problém se za asistence síťového administrátora BUTCA platformy podařilo odstranit vypnutím Port Security v nastavení sítě OpenStacku, ve které scénář probíhal. Dle vyjádření síťového administrátora BUTCA platformy však není možné automatizovat tento proces v současném stavu řešení BUTCA platformy. Při každém spuštění scénáře tedy bude nutné vypnout Port Security v nastavení sítě OpenStacku. Zpětná vazba profesionálních penetračních testerů k hernímu scénáři byla velmi pozitivní. Jedinou výtkou byl krátký časový úsek poskytnutý pro nastavení poskytnutých Kali Linux VM.

Ve druhé fázi testování proti sobě soutěžili studenti oboru Informační bezpečnost z Fakulty elektrotechniky a komunikačních technologií Vysokého učení technického v Brně. Nastavení scénáře bylo následující:

- 2 proti 2,
- 6 hracích kol, každé o délce 5 minut,
- kontrola provozuschopnosti zranitelných instancí každé 2 minuty,
- bodové ohodnocení 50 bodů za vlajku administrátorského účtu a 25 bodů za vlajku běžného uživatelského účtu,
- penalizace 15 bodů za špatný stav zranitelné instance,
- počáteční skóre 1000 bodů,
- 3 zranitelnosti typu Initial foothold,
- 2 zranitelnosti typu Privilege escalation.

Do scénáře byly navíc implementovány poznatky z první fáze testování, došlo tedy k vypnutí Port Security v nastavení sítě scénáře a účastníkům byl poskytnut delší čas



Obr. 5.2: Pravidla herního scénáře v prostředí platformy BUTCA

na přípravu poskytnutých Kali Linux VM. Během druhé fáze testování implementovaného řešení herního scénáře nedošlo k žádným problémům, a lze tedy prohlásit implementaci herního scénáře do BUTCA platformy za úspěšnou.

Závěr

Cílem předkložené diplomové práce bylo navrhnout a implementovat nový herní scénář do platformy BUTCA a důkladně otestovat implementované řešení scénáře.

V diplomové práci byly popsány metodologie pentetračního testování, stejně tak jeho různé druhy a typy. Dále je v práci popsán význam Cyber Ranges pro výukové účely v oblasti informační bezpečnosti. Dále jsou v práci uvedeny a přiblíženy zranitelnosti z vlastní knihovny zranitelností, které se budou nacházet ve zranitelných virtuálních strojích, jež budou součástí herního scénáře. Následně byl popsán průběh scénáře. Dále byly v práci popsány problémy s různými přístupy k různým řešením implementace scénáře do platformy BUTCA. Na základě popsáných problémů byla vyvinuta implementační strategie do platformy BUTCA, která odstraňuje tyto problémy a zároveň je kompatibilní se současným stavem platformy BUTCA.

V rámci diplomové práce byly vytvořeny jednotlivé komponenty realizující Game Server. Tyto komponenty obstarávají tvorbu zranitelných instancí, kontrolu stavu jednotlivých instancí, hodnocení hráčů a komunikaci mezi hráči a samotným herním scénářem. Pro tvorbu zranitelných instancí používaných v jednotlivých hrách herního scénáře byla v rámci diplomové práce vytvořena vlastní knihovna zranitelností, která zajišťuje jedinečnost jednotlivých spuštění scénáře. Díky použití knihovny pro tvorbu zranitelných instancí je scénář snadno rozšiřitelný a škálovatelný.

Scénář byl implementován do platformy BUTCA a důkladně otestován. Testování proběhlo za pomoci studentů oboru informační bezpečnosti Fakulty elektrotechniky a komunikačních technologií Vysokého učení technického v Brně a týmu profesionálních penetračních testerů. Testování ověřilo funkcionality implementovaného řešení scénáře do platformy BUTCA. Tím byl splněn poslední z vytyčených cílů diplomové práce, všechny cíle diplomové práce tak byly splněny.

Literatura

- [1] *The UK National Cyber Security Center*. Online. 2017, 10.1.2022. Dostupné z: https://www.ncsc.gov.uk/guidance/penetration-testing#section_2. [cit. 2023-11-05].
- [2] *Pre-engagement Interactions*. Online. PTES-s documentation. 2016, 11.04.2022. Dostupné z: https://pentest-standard.readthedocs.io/en/latest/pre-engagement_interactions.html. [cit. 2023-11-06].
- [3] *Intelligence Gathering*. Online. PTES-s documentation. 2016. Dostupné z: https://pentest-standard.readthedocs.io/en/latest/intelligence_gathering.html. [cit. 2023-11-06].
- [4] *Threat Modeling*. Online. PTES-s documentation. 2016, 16.06.2021. Dostupné z: https://pentest-standard.readthedocs.io/en/latest/threat_modeling.html. [cit. 2023-11-06].
- [5] *Vulnerability Analysis*. Online. PTES-s documentation. 2016. Dostupné z: https://pentest-standard.readthedocs.io/en/latest/vulnerability_analysis.html. [cit. 2023-11-06].
- [6] *Exploitation*. Online. PTES-s documentation. 2016, 12.04.2022. Dostupné z: <https://pentest-standard.readthedocs.io/en/latest/exploitation.html>. [cit. 2023-11-06].
- [7] *Post Exploitation*. Online. PTES-s documentation. 2016, 12.04.2022. Dostupné z: https://pentest-standard.readthedocs.io/en/latest/post_exploitation.html. [cit. 2023-11-06].
- [8] *Reporting*. Online. PTES-s documentation. 2016. Dostupné z: <https://pentest-standard.readthedocs.io/en/latest/reporting.html>. [cit. 2023-11-06].
- [9] *Types of Pen Testing: Black Box, White Box & Grey Box*. Online. RedScan. 2023. Dostupné z: <https://www.redscan.com/news/types-of-pen-testing-white-box-black-box-and-everything-in-between/>. [cit. 2023-11-06].
- [10] *What are black box, grey box, and white box penetration testing*. Online. Infosec Institute. 2023. Dostupné z: <https://resources.infosecinstitute.com/topics/penetration-testing/what-are-black-box-grey-box-and-white-box-penetration-testing/>. [cit. 2023-11-06].

- [11] *Black-Box vs Grey-Box vs White-Box Penetration Testing*. Online. Packetlabs. 2022. Dostupné z: <https://www.packetlabs.net/posts/types-of-penetration-testing/>. [cit. 2023-11-06].
- [12] *Penetrační testování - co to je, jak na ně vč. odkazů a zdrojů*. Online. CESNET. 2021. Dostupné z: <https://www.packetlabs.net/posts/types-of-penetration-testing/>. [cit. 2023-11-06].

Seznam symbolů a zkratek

AAD	Azure Active Directory
AD	Active Directory
BUTCA	Brno University of Technology Cyber Arena
CMS	Content Management System
CSTI	Client-Side Template Injection
CTF	Capture the Flag
CVE	Common Vulnerabilities and Exposures
DoS	Denial of Service
ESAPI	OWASP Enterprise Security API
HTTP	Hyper-Text Transfer Protocol
ISSAF	Information System Security Assessment Framework
LFI	Local File Inclusion
NCSC	The UK National Cyber Security Center
NIST	The National Institute of Standards and Technology
OSINT	Open Source Intelligence
OSSTMM	Open Source Security Testing Methodology Manual
OWASP	Open Web Application Security Project
PTES	Penetration Testing Execution Standard
PoC	Proof of Concept
RCE	Remote Code Execution
RFI	Remote File Inclusion
SSH	Secure Shell
SSTI	Server-Side Template Injection
SUID	Set owner User ID

VM	Virtual Machine
XSS	Cross-Site Scripting
XXE	XML External Entity
XML	Extensible Markup Language

A Seznam elektronických příloh

