Nástroj pro tvorbu reportu z automatizovaného penetračního testu

Bc. Michal Martinek

Diplomová práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2023/2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Michal Martinek**
Osobní číslo:	**A22592**
Studijní program:	**N0613A140022 Informační technologie**
Specializace:	**Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Nástroj pro tvorbu reportu z automatizovaného penetračního testu**
Téma práce anglicky:	**A Tool for Creating a Report from an Automated Penetration Test**

Zásady pro vypracování

1. Vypracujte rešerši na problematiku penetračního testování.
2. Analyzujte aktuální možnosti automatizovaných penetračních testů a k tomu určených specializovaných zařízení.
3. Navrhněte aplikaci pro kompletaci reportů z automatizovaných penetračních testů.
4. Zvolte vhodný programovací jazyk a zdokumentujte vývoj.
5. Vytvořte odpovídající testy a otestujte vytvořenou aplikaci.
6. Zhodnoťte výsledný report z pohledu uživatelské přívětivosti a doporučte budoucí možný vývoj.

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. HERZOG, Pete. OSSTMM3: The Open Source Security Testing Methodology Manual. ISECOM. [online]. Dostupné z: https://www.isecom.org/OSSTMM.3.pdf.
2. SELECKÝ, Matúš. Penetrační testy a exploitace. Brno: Computer Press, 2012. ISBN 978-80-251-3752-9.
3. WEIDMAN, Georgia. Penetration testing: a hands-on introduction to hacking. San Francisco, CA: No Starch Press, c[2014]. ISBN 978-15-932-7564-8.
4. KIM, Peter. Hacking: praktický průvodce penetračním testováním. Přeložil Jan POKORNÝ. Encyklopedie Zoner Press. Brno: Zoner Press, 2015. ISBN 978-80-741-3313-8.
5. LYON, Gordon. Nmap: the Network Mapper – Free Security Scanner. [online]. Dostupné z: https://nmap.org/.
6. PECINOVSKÝ, Rudolf. Začínáme programovat v jazyku Python. Začínáme s.. Praha: Grada Publishing, 2020. ISBN 978--80-271-1237-1.
7. BORY, Pavel. C# bez předchozích znalostí. 2. vydání. V Brně: Computer Press, 2022. ISBN 978-80-251-5061-0.
8. What is Java? [online]. Dostupné z: https://opensource.com/resources/java.

Vedoucí diplomové práce: **Ing. Lukáš Králík, Ph.D.**
Ústav bezpečnostního inženýrství

Datum zadání diplomové práce: **5. listopadu 2023**
Termín odevzdání diplomové práce: **13. května 2024**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 5. ledna 2024

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 13. května 2024

Bc. Michal Martinek v.r.
…….
podpis studenta

ABSTRAKT

Tato práce se zabývá tvorbou aplikace pro automatizovaný běh penetračních testů a reportu z výsledků testování. Cílem je vytvořit nástroj v jazyce C#, jež při spuštění vykoná dvě dílčí operace. První z nich je spuštění skriptů jazyka PowerShell, které provedou testy a výsledky uloží v podobě textových souborů. Druhou je zpracování těchto textových souborů a vytvoření jednotného reportu z testování ve formátu PDF.

Aplikace je vytvořena ve dvou verzích. Pro spouštění testů uživatelem slouží verze s grafickým rozhraním (GUI) postaveném na frameworku Winforms. Konzolová verze pak umožní také automatizované spuštění na pozadí bez přímé účasti uživatele.

Klíčová slova: penetrační testy (počítačová bezpečnost), desktopová aplikace, PDF (souborový formát), C# (programovací jazyk), PowerShell (programovací jazyk), regulární výrazy

ABSTRACT

This thesis deals with the creation of an application for the automated run of penetration tests and a report from the test results. The goal is to create a C# tool that performs two basic operations when started. The first is to run PowerShell scripts that will run the test and save the results to a text file. The second step is to process these text files and create a unified test report in PDF format.

The application is created in two versions. A version with a graphical interface (GUI) built on the Winforms framework is used for running tests by the user. The console version will also allow automated background execution without direct user involvement.

Keywords: penetration tests (computer security), desktop application, PDF (computer file format), C# (programming language), PowerShell (programming language), regular expressions

Tímto bych chtěl poděkovat vedoucímu diplomové práce, panu Ing. Lukáši Králíkovi Ph.D., za odborné vedení a rady při tvorbě diplomové práce. Dále pak patří dík i mé rodině, jež mě po celou dobu studia podporovala a bez níž bych ho dokončit nedokázal.

Prohlašuji, že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

V dnešní době je kybernetická bezpečnost nezbytnou součástí každé organizace, která využívá moderní informační technologie. S narůstajícím množstvím internetových hrozeb a útoků je klíčové zajistit ochranu informačních systémů před potencionálními zranitelnostmi a útoky.

Cílem této diplomové práce je poskytnout uživatelům nástroj, který umožní snadno vytvořit přehled o zranitelnostech v jejich systémech. Tato aplikace umožňuje na pozadí systému spustit sérii penetračních testů ve formě PowerShell skriptů a z výstupů vytvořit přehledný dokument ve formátu PDF.

Pro maximální použitelnost je aplikace zhotovena ve dvou verzích. Konzolová verze umožňuje spouštět aplikaci s danou konfigurací jako proces bez zásahu uživatele. Verze s grafickým rozhraním je určena pro nastavení a spuštění přímo uživatelem.

Teoretická část se zabývá druhy a metodikami penetračního testování. Dále jsou zde popsány jednotlivé nástroje, které aplikace využívá k provádění penetračních testů a také formy výstupů z těchto nástrojů.

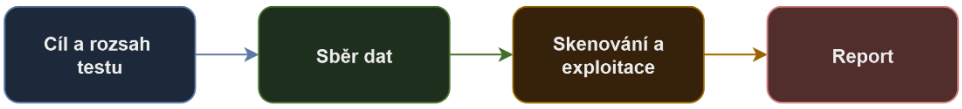
V praktické časti jsou popsány zvolené testy a vývoj samotné aplikace. Mimo jiné, je zde znázorněno praktické použití knihovny PDFsharp, jež slouží k tvorbě dokumentu. Jsou zde popsány důležité části zdrojového kódu a práce se soubory projektu. Dále se praktická část věnuje tvorbě uživatelského rozhraní (GUI) a na závěr jsou také přiloženy ukázky některých výsledných reportů.

I. TEORETICKÁ ČÁST

1 PENETRAČNÍ TESTY

Penetrační testy, známé také jako etické hackování, představují klíčový prvek v oblasti kybernetické bezpečnosti. Jedná se o systematický a řízený proces hodnocení bezpečnosti informačního systému či sítě prostřednictvím simulace skutečného útoku. Cílem penetračního testu je identifikovat a následně odstranit potenciální slabiny a zranitelnosti, které by mohly být využity neautorizovanými útočníky. Penetrační testy jsou tedy nástrojem pro organizace, které chtějí zajistit, že jejich IT infrastruktura je odolná vůči hrozbám kybernetických útoků. V době, kdy kybernetické útoky stále přibývají na intenzitě a sofistikaci, jsou penetrace klíčové pro udržení bezpečnosti a integrity informačních systémů. [1] [2]

1.1 Fáze penetračního testování



Obrázek 1: Fáze penetračního testování

Fáze 1: Cíl a rozsah testu

V této fázi se na základě obecných zadání a cílů určí detailnější cíle testu a také jeho rozsah. Hlavní je vymezit prioritní cíle, protože nelze vše ověřit zcela jistě. To zahrnuje identifikaci systémů, aplikací nebo sítí, které mají být testovány. [1]

Fáze 2: Sběr dat

V této fázi jsou, na základě výstupu z fáze 1, shromažďovány veškeré relevantní informace o cílovém prostředí. To může zahrnovat sběr informací o síťové topologii, identifikaci aktivních systémů a služeb nebo zjišťování konfigurace systémů a verzí softwaru. Cílem této fáze je získat co nejúplnější obraz o prostředí, které bude podrobeno testování. [1]

Fáze 3: Skenování a exploatace

V této fázi jsou použity různé nástroje a techniky k identifikaci zranitelností a slabých míst v systémech nebo sítích. To může zahrnovat hledání otevřených portů, zranitelných aplikací a služeb nebo pokusy o exploataci za účelem získání přístupu a kontroly nad systémy. Cílem je ověřit, zda jsou identifikované zranitelnosti skutečnými hrozbami a jak by mohly být zneužity.

Žádný systém není dokonalý a pokud doteď nebyl prolomen, tak to neznamená, že prolomen být nemůže. Pouze to znamená, že způsob, jak to provést, ještě nebyl odhalen. A to platí nejen v oblasti informačních systémů, ale obecně. [1]

Fáze 4: Report

Po dokončení testování je připravena a prezentována zpráva obsahující všechny nalezené zranitelnosti, jejich závažnost a doporučení pro zlepšení zabezpečení. Tato zpráva poskytuje podrobný přehled o stavu bezpečnosti prostředí. Cílem poslední fáze je seznámení zainteresovaných stran, včetně správců informační bezpečnosti a vedení organizace, s výsledky testování, aby mohli přijmout nezbytná opatření k ochraně systémů a dat. [1]

1.2 Objekty penetračních testů

Objekty penetračních testů se odvíjejí od cílů a rozsahu testování. Penetrační testy mohou mít široký záběr a mohou být zaměřeny na různé aspekty informačních systémů.

a) **Fyzická bezpečnost**: Testy zabezpečení spojené s fyzickým přístupem k zařízením a infrastruktuře, včetně testování bezpečnosti budov, uzamykání serverovny, kamerového dohledu a dalších prvků fyzické bezpečnosti.
b) **Operační systémy**: Testy konkrétních OS (např. Windows, Linux) k identifikaci zranitelností v jádru systému, oprávněních a dalších kritických prvcích.
c) **Mobilní aplikace**: Testy zaměřené na bezpečnost mobilních aplikací pro různé platformy (Android, iOS). Zahrnuje analýzu kódu, ověřování komunikace s backendem a testování zabezpečení dat uložených na zařízení.
d) **Webové aplikace**: Testy zabezpečení zaměřené na webové aplikace, včetně analýzy zranitelností v kódu, správě relací a ochraně před SQL injection, cross-site scripting (XSS) a dalšími webovými hrozbami.
e) **Bezdrátové sítě**: Testování zabezpečení bezdrátových sítí, včetně ověřování kvality šifrování, identifikace možných útoků typu Man-in-the-Middle (MitM) a dalších
f) **Sociální inženýrství**: Simulace útoků na lidský faktor. To může zahrnovat testování odpovědi zaměstnanců na phishingové e-maily, pokusy o získání citlivých informací a další formy sociálního inženýrství.
g) **Cloudové služby**: Testování bezpečnosti cloudových prostředí, včetně konfigurace a správy cloudových služeb, ověřování přístupových práv a ochrany dat v cloudu.

1.3 Druhy penetračních testů

Penetrační testy se dělí na základě znalostí o testovaném systému na: Black-box, White-box, Grey-box. Existuje však více hledisek, podle kterých lze testy dělit. Dalším může být způsob provedení testu. Podle provedení se testy dělí na manuální, automatizované nebo semiautomatické. [1]

Dělení podle znalostí:

a) Black-box

V tomto případě má tester minimální nebo žádnou předchozí znalost o testovaném systému. Tester má přístup pouze k veřejně dostupným informacím, stejně jako by to bylo pro potenciálního útočníka bez interních znalostí o systému. U tohoto typu testu je nutný poměrně rozsáhlý průzkum. Samotná funkcionalita systému je tzv. černou skřínkou (black-box).

Výhodou je zde realistický pohled na to, jak by vnější útočník mohl napadnout systém. Poskytuje objektivní perspektivu a umožňuje identifikovat zranitelnosti, ke kterým by externí útočník mohl mít přístup. Zároveň není třeba, aby byl testerovi zpřístupněn zdrojový kód, takže není ohroženo know-how zákazníka (průmyslová špionáž apod.).

Nevýhodou black-box přístupu může být chybějící interní kontext o systému, což může omezit hloubku testování. [1]

b) White-box

Protiklad black-box testů jsou tzv. white-box testy. V tomto případě má tester úplnou znalost o testovaném systému, včetně interní architektury, zdrojového kódu a konfigurace. Tester může používat všechny dostupné informace k identifikaci a vyhodnocení zranitelností. [1]

Výhodou je hluboký vhled do vnitřních struktur systému, což umožňuje identifikovat zranitelnosti, které by mohly být jinak objeveny obtížněji.

Za nevýhodu se dá považovat riziko nedostatečného testování vnějších hrozeb a zanedbání faktorů, které by mohl znát pouze externí útočník. Dále může být problém relativně úzké zaměření na kód a architekturu. Je zde nutná znalost programovacího jazyka testerem, což může ovlivnit cenu testu, jelikož tester může potřebovat vyšší kvalifikaci. [1]

c) Grey-box

Tento přístup k testům kombinuje prvky black-box a white-box testování. Tester má omezenou znalost o systému (často s částečným přístupem k interním informacím). Toto omezené množství znalostí může simulovat situaci, kdy má útočník částečnou informační nebo insiderskou perspektivu.

Kombinuje výhody obou předchozích přístupů. Umožňuje testovat systém z více perspektiv a poskytuje komplexnější pohled na celkovou bezpečnost.

Nevýhodou může být obtížné dosažení správné rovnováhy mezi úrovní znalostí testera a skutečným povědomím útočníka. Vyžaduje dobrou komunikaci mezi testerem a týmem spravujícím systém. [1]

Dělení podle způsobu provedení

a) Manuální testy

V tomto případě jsou testy prováděny lidským testerem, který manuálně provádí analýzu a vyhledává zranitelnosti ve zkoumaném systému. Tester využívá své znalosti, intuici a kreativitu k identifikaci potenciálních bezpečnostních nedostatků.

Výhodou tohoto přístupu je flexibilita a schopnost přizpůsobit se měnícím se podmínkám nebo neobvyklým situacím.

Nevýhodou je časová náročnost a závislost na dovednostech konkrétního testera. [1]

b) Automatizované testy

Jsou testy prováděny pomocí automatizovaných nástrojů a skriptů, které automaticky skenují, identifikují zranitelnosti a provádějí některé útoky na testovaném systému. Testovací nástroje mohou být naprogramovány k opakování rutinních úloh a identifikují tak zranitelnosti rychleji, než kdyby to dělal člověk.

Hlavní výhodou je tedy efektivita, rychlost provedení a snadná rozšiřitelnost. Testy mohou být spuštěny opakovaně a konzistentně. Jsou ideální pro rutinní testování velkých systémů.

Nevýhodou je, že může mít omezenou schopnost identifikovat některé typy složitějších zranitelností. Vyžaduje údržbu a aktualizace skriptů. [1]

c) Semi-automatické testy

Tyto testy kombinují prvky manuálních a automatizovaných testů. Tester používá automatizované nástroje pro určité fáze testování, ale také provádí ruční analýzu a hodnocení zranitelností. Může například používat automatizované skenování a následně provádět ruční ověření nalezených zranitelností.

Kombinují výhody obou přístupů. Poskytují flexibilitu ručního testování a rychlost automatizovaných nástrojů.

Nevýhodou může být potřeba vyšší úrovně dovedností testera (ačkoliv menší než úplně manuální přístup). [1]

1.4 Metodiky penetračního testování

V dnešní době je mnoho metodik provádění penetračních testů. Existují komerční firmy, které se specializují na testování bezpečnosti nebo nabízejí školení a certifikace. Ne všechny tyto metodiky jsou však chráněny obchodním tajemstvím. Některé jsou volně k dispozici v rámci open source komunity. [1]

a) OSSTMM

OSSTMM je standardem pro penetrační testování vyvinutý organizací ISECOM. Zabývá se měřením bezpečnostních rizik v rámci organizací a zdůrazňuje systematický přístup k testování bezpečnosti. OSSTMM se zaměřuje se na široké spektrum bezpečnostních aspektů jako např. informační zabezpečení, fyzické zabezpečení nebo také lidský faktor. [1] [3]

b) OWASP

Metodika OWASP je zaměřena na zabezpečení webových aplikací. Jmenuje se podle mezinárodní neziskové organizace zaměřené na zlepšení bezpečnosti softwaru, která je známa svým otevřeným a komunitním přístupem k vývoji. [4]

1.5 Právní aspekty testování

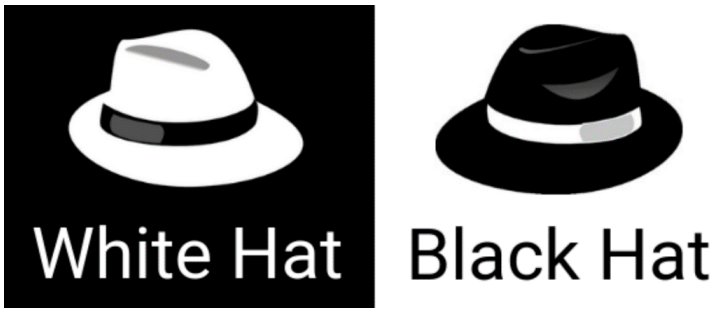
Právní a etické aspekty penetračního testování jsou klíčovými faktory, které je důležité zohlednit při provádění těchto testů. Je velmi důležité, aby tester měl povolení klienta k provádění testů na cílovém systému. V případě, kdy klient není vlastníkem daného sytému, je důležité, aby měl formální zmocnění od majitele. Dále je vhodné, aby smlouva mezi stranami obsahovala prohlášení, které zbavuje testera odpovědnosti pro případ neočekávaných situací. V neposlední řadě by také měla být předem smluvena cena a časový rozsah testu.

Přesný rozsah testování by měl být stanoven, aby se minimalizovalo riziko náhodného zjištění citlivých informací, které nejsou předmětem testování. V některých případech může být vhodné používat falešné nebo anonymizované údaje během testování. Pokud však tester, z nějakého důvodu, již má přístup k citlivým informacím, měl by být vázán mlčenlivostí a jakmile je testování dokončeno, všechny získané informace by měly být bezpečně smazány nebo vráceny organizaci. Není vhodné ukládat citlivá data bez souhlasu. [1] [4]

1.6 Typy útočníků

a) White hat

White hat útočníci jsou etičtí profesionálové, kteří mají za cíl zlepšit bezpečnost systémů. Pracují na objednávku vlastníka systému nebo organizace a své dovednosti využívají k identifikaci a opravě zranitelností. Jejich práce spočívá v provádění penetračních testů, auditů bezpečnosti a vytváření bezpečnostních politik. [1]



Obrázek 2: typy hackerů [5]

b) Black hat

Black hat útočníci jsou neetičtí a nemají žádnou oprávněnou autoritu k tomu, aby pronikali do cílového systému. Jejich motivací je obvykle osobní prospěch nebo financování nelegálních aktivit. Mezi jejich cíle patří zneužívání systémů, krádeže citlivých dat, vydírání, šíření malwaru nebo páchání jiných kybernetických zločinů. Často působí skrytě, využívají sofistikované metody a snaží se uniknout identifikaci a spravedlnosti. [1]

c) Gray hat

Gray hat útočníci kombinují vlastnosti předešlých typů. Mají schopnosti k průniku do systémů bez oprávnění, ale obvykle tak činí s dobrými úmysly. Mohou mít za cíl posílit bezpečnost systému, ale často jednají bez formálního povolení vlastníka systému. Zveřejňují nalezené zranitelnosti a informují organizaci nebo vlastníka systému o slabých místech. Někdy také mohou pomoci s opravami, i když to není formálně požadováno. [1]

1.7 Nástroje pro penetrační testování

1.7.1 Nmap

Nmap, zkratka pro "Network Mapper", je bezplatný a open-source nástroj určený pro skenování a mapování sítě. Jeho hlavním účelem je poskytnout uživatelům detailní informace o zařízeních a službách běžících v jejich síti. S pomocí Nmapu mohou administrátoři sítí identifikovat připojená zařízení, zjistit otevřené porty a určit, jaké služby na těchto portech běží. To je užitečné pro diagnostiku síťových problémů, zabezpečení sítě a detekci potenciálních bezpečnostních hrozeb. [6]

Obrázek 3:Nmap [7]

Jednou z nejvýznamnějších vlastností Nmapu je jeho schopnost provádět různé typy skenování sítě. Patří sem TCP skenování, UDP skenování, SYN skenování, a další. Každý typ skenování má své výhody a nevýhody a může být použit v závislosti na konkrétní situaci.

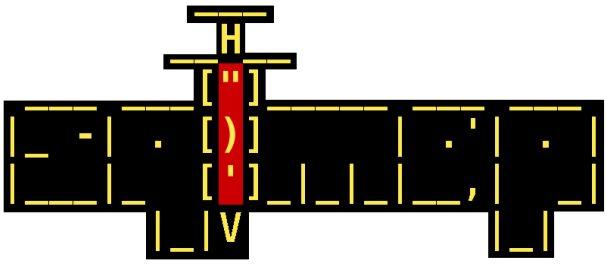
Další klíčovou vlastností Nmapu je jeho schopnost detekovat operační systémy na vzdálených strojích. Tento proces, nazývaný OS fingerprinting, umožňuje uživatelům získat informace o operačním systému, který běží na cílovém zařízení, což je užitečné pro plánování bezpečnostních opatření a optimalizaci síťových služeb.

Nmap také nabízí širokou škálu funkcí, které umožňují uživatelům přizpůsobit skenování podle svých potřeb. To zahrnuje možnost použití vlastních skriptů pro automatizaci úloh nebo možnost exportu výsledků do různých formátů pro další analýzu.

Nicméně Nmap může být také použit k nelegitimním účelům, jako je skenování cizích sítí bez povolení. Každý, kdo nástroj používá, by si toho měl být vědom a měl by svou činnost provádět pouze v souladu s platnými zákony a etickými standardy. [6]

1.7.2 sqlmap

Sqlmap je open-source nástroj specializovaný na testování zranitelností webových aplikací, které využívají databáze SQL. Jeho hlavním účelem je automaticky detekovat zranitelnosti spojené s databázovými systémy, jako je například MySQL, Microsoft SQL Server a Oracle. [8]



Obrázek 4: sqlmap [8]

Tento nástroj je oblíbený mezi bezpečnostními profesionály, pentestery a hackery díky své schopnosti rychle identifikovat různé typy SQL injection útoků. SQL injection je technika, která umožňuje útočníkovi vložit nežádoucí SQL kód do dotazu, který je prováděn na webové stránce. sqlmap může identifikovat tuto zranitelnost a umožnit útočníkovi získat nelegitimní přístup k databázi a citlivým informacím, jako jsou hesla, osobní údaje nebo dokonce celá tabulka dat.

Hlavními funkcemi sqlmapu jsou automatické detekce, detekce databázových typů, extrakce dat, zjišťování uživatelských přístupových údajů a mnoho dalších. sqlmap poskytuje uživatelům možnost testovat zabezpečení svých webových aplikací a databází a identifikovat potenciální bezpečnostní hrozby.

Je však důležité si uvědomit, že použití sqlmapu k testování webových aplikací, na kterých nemáte povolení, může být nelegální a eticky neakceptovatelné.

Celkově je sqlmap mocným nástrojem pro testování zabezpečení webových aplikací a identifikaci zranitelností spojených s databázovými systémy. Jeho použití může pomoci organizacím zlepšit zabezpečení svých webových aplikací a minimalizovat rizika spojená s SQL injection útoky. [8]

1.7.3 Mimikatz

Mimikatz je nástroj určený pro získávání citlivých údajů, jako jsou hesla, klíče, tokeny a certifikáty z paměti operačního systému Windows. Jeho hlavním účelem je demonstrovat zranitelnosti v autentizačních mechanismech operačního systému a umožnit uživatelům analyzovat zabezpečení svých systémů. [9]

Obrázek 5: Mimikatz [10]

Tento nástroj se stal známým v bezpečnostní komunitě díky své schopnosti extrahovat hesla a jiné citlivé údaje přímo z paměti systému, což je způsob, jakým může útočník získat nelegitimní přístup k systému. Mimikatz dokáže pracovat s různými verzemi operačního systému Windows a podporuje různé techniky pro získání citlivých dat, včetně "Pass-the-Hash" útoků, které umožňují útočníkovi autentizovat se na systému pomocí hashe hesla, aniž by potřeboval znát samotné heslo.

Mezi hlavní funkce Mimikatzu patří získávání hesel, tokenů a certifikátů z paměti systému, procházení lokálních uživatelů a skupin, manipulace s autentizačními tokeny a mnoho dalších. Mimikatz je oblíbeným nástrojem mezi penetračními testery a bezpečnostními profesionály, kteří používají jeho schopnosti k testování zabezpečení a detekci možných zranitelností v systémech Windows.

Jeho použití by však mělo být pouze za účelem testování zabezpečení a pro demonstraci zranitelností v autentizačních mechanismech. [9]

1.7.4 Nikto

Nikto je open-source nástroj určený pro detekci a testování webových serverů na různé zranitelnosti a potenciální bezpečnostní hrozby. Jeho hlavním účelem je poskytnout uživatelům informace o možných slabých místech ve webových aplikacích a infrastruktuře, které by mohly být zneužity útočníky k neoprávněnému přístupu nebo poškození systému. [11]

Obrázek 6: Nikto [12]

Nikto je populárním nástrojem mezi bezpečnostními profesionály a penetračními testery, kteří ho používají k automatickému skenování webových serverů. Mezi typy skenování, které Nikto provádí, patří detekce neaktualizovaného softwaru, hledání známých bezpečnostních chyb a analýza konfigurace serveru.

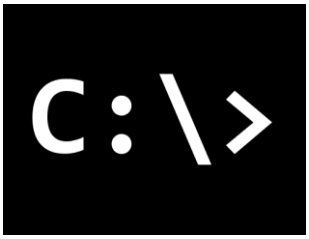
Nikto podporuje různé typy webových serverů a technologií, což mu umožňuje být efektivním nástrojem pro skenování široké škály webových aplikací.

Je však důležité si uvědomit, že použití Nikta by mělo být provedeno s dohledem a souhlasem vlastníka webových serverů. Neoprávněné skenování může být v některých případech nelegální a může mít právní důsledky. Používání Nikta by mělo být v souladu s platnými zákony a etickými standardy.

Celkově je Nikto užitečným nástrojem pro detekci a skenování webových serverů na možné zranitelnosti a bezpečnostní hrozby. Jeho použití může pomoci organizacím zlepšit zabezpečení svých webových aplikací a minimalizovat rizika spojená s útoky na webové servery. [11]

1.7.5 Windows Commands

Windows Commands, často označované jako příkazy příkazového řádku nebo příkazy PowerShellu, jsou integrované nástroje operačního systému Windows, které umožňují uživatelům získat různé informace o systému, síti a aplikacích. Tyto příkazy jsou užitečné pro diagnostiku problémů, správu systému a monitorování výkonu.



Obrázek 7: cmd

Zde jsou některé z nejčastěji používaných příkazů pro získání informací o systému:

- **whoami**: Tento příkaz zobrazuje aktuálního přihlášeného uživatele v rámci aktuálního kontextu.
- **systeminfo**: Příkaz systeminfo poskytuje detailní informace o hardwaru a softwaru na počítači, včetně informací o operačním systému, verzi, instalovaných aktualizacích, fyzické a virtuální paměti a mnoho dalšího.
- **ipconfig**: Tento příkaz zobrazuje informace o síťových adaptérech na počítači, včetně IP adres, masky podsítě, výchozí brány a dalších informací.
- **tracert**: Diagnostický nástroj *tracert* (Trace Route) je používán k určení trasy, kterou data procházejí z jednoho bodu v síti k druhému. Pro zjištění této trasy TRACERT využívá hodnoty IP Time-To-Live (TTL). Každý směrovač na trase sníží hodnotu TTL paketu minimálně o 1 před jeho předáním dál. TTL tedy efektivně funguje jako čítač směrování. Je-li hodnota TTL paketu nula a není v cílové IP síti, pak je paket zahozen a odesílateli poslána chybová zpráva.
- **nslookup**: Zobrazuje informace, které můžete použít k diagnostice infrastruktury DNS (Domain Name System).
- **netstat**: Tento příkaz zobrazuje síťové spojení, směrovací tabulky, statistiky rozhraní a další síťové informace.

2 VHODNÉ FORMÁTY PRO VÝSTUPY Z TESTU

Existuje celá řada formátů pro ukládání textu, z nichž některé jsou specializované pro určité účely nebo mají specifické vlastnosti. Zde je několik formátů, které stojí za zmínku.

2.1 TXT

Formát .txt je základní formát pro ukládání textových dat. Zkratka „TXT“ znamená „textový soubor“. Jedná se o jednoduchý a univerzální formát, který umožňuje ukládání textu bez jakýchkoli formátovacích prvků, jako jsou styly písma, barvy nebo grafika. Text je uložen jako série znaků ASCII (American Standard Code for Information Interchange) nebo Unicode, které představují písmena, číslice a speciální znaky.

Hlavní výhodou formátu .txt je jeho jednoduchost a široká podpora napříč různými operačními systémy a aplikacemi. Textové soubory .txt lze otevřít a upravit v prakticky jakémkoli textovém editoru, včetně Notepadu (Windows), TextEditu (Mac) nebo Vim (Unix/Linux). To zajišťuje snadnou přenositelnost a sdílení textových dat mezi různými uživateli a systémy.

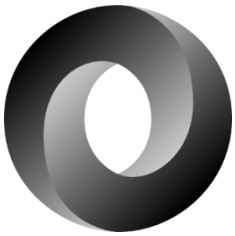
Použití formátu .txt je rozmanité. Může se jednat o ukládání poznámek, vytváření jednoduchých konfiguračních souborů, zálohování údajů nebo ukládání zdrojového kódu. Textové soubory .txt jsou také často používány pro vytváření jednoduchých webových stránek, zejména pro soubory s kódem HTML, CSS nebo JavaScriptu.

I přesto, že formát .txt nabízí jednoduchost a přenositelnost, má i své omezení. Chybí mu možnost formátování textu, vkládání obrázků nebo jiných multimediálních prvků, což může být nevýhodou při ukládání složitějších dokumentů nebo prezentací.

Celkově lze říci, že formát .txt je ideální volbou pro ukládání a sdílení čistého textu bez formátování. Jeho jednoduchost a univerzální podpora ho činí důležitým nástrojem pro mnoho uživatelů a aplikací po celém světě.

2.2 JSON

Formát JSON (JavaScript Object Notation) je lehký a snadno čitelný datový formát, který se používá k výměně dat mezi aplikacemi. Byl vyvinut jako alternativa k XML s cílem poskytnout jednoduchý a efektivní způsob serializace strukturovaných dat. JSON je založen na JavaScriptu a používá syntaxi, která je snadno srozumitelná pro programátory. [13]



Obrázek 8: JSON logo [14]

Hlavní charakteristikou formátu JSON je jeho schopnost reprezentovat různé typy dat, včetně objektů (sady klíč-hodnota), pole (seznamy hodnot) a primitivní datové typy (čísla, řetězce, logické hodnoty a null). Data jsou zaznamenána ve formě páru klíč-hodnota, kde je klíč identifikátorem a hodnota obsahuje samotná data. [13]

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

Kód 1: příklad JSON [15]

JSON je často používán ve webovém vývoji a API pro přenos dat mezi klientem a serverem. Jeho jednoduchá syntaxe a snadné zpracování z něj činí ideální volbu pro přenos strukturovaných dat v rámci moderních webových aplikací. JSON je podporován v mnoha programovacích jazycích a existuje mnoho knihoven pro jeho zpracování.

Další výhodou formátu JSON je jeho lidská čitelnost. Data v JSON dokumentu jsou zapisována pomocí běžných znaků, což zjednodušuje ladění a ruční úpravy datových souborů. Také to usnadňuje práci s JSON daty jak pro programátory, tak i pro ostatní uživatele. [13]

Přestože JSON nabízí mnoho výhod, má i několik omezení. Například nemá vestavěnou podporu pro komentáře, což může být nevýhoda při psaní rozsáhlejších konfiguračních souborů. Dále také nedokáže zachytit cyklické odkazy mezi objekty, což může být problém při serializaci složitějších datových struktur. [13]

2.3 XML

Formát XML (eXtensible Markup Language) je značkovací jazyk, který se používá k ukládání a přenosu strukturovaných dat. XML umožňuje definovat vlastní značky a hierarchii dat, což poskytuje flexibilitu při organizaci informací. Každý prvek dat je obvykle zaznamenán pomocí značek, které jsou definovány v syntaxi podobné HTML.

Může být použit pro ukládání konfiguračních informací, výměnu dat mezi aplikacemi, serializaci objektů pro přenos mezi systémy a mnoho dalších účelů. XML je nezávislý na platformě a jazyku, což z něj činí vhodný formát pro interoperabilitu mezi různými systémy.

Struktura XML dokumentu je obvykle hierarchická, skládá se z kořenového elementu, který obsahuje další vnořené elementy. Každý element může obsahovat atributy a textový obsah. Díky své flexibilitě a hierarchické povaze je formát XML často používán v různých oblastech, včetně webových služeb, databází, elektronického obchodování, zpracování dokumentů a mnoha dalších. Nicméně, při ukládání velkého množství dat může XML vykazovat větší režii než jiné formáty, jako je například JSON (JavaScript Object Notation). [16]

```
<menu id="file" value="File">
      <popup>
            <menuitem value="New" onclick="CreateNewDoc()" />
            <menuitem value="Open" onclick="OpenDoc()" />
            <menuitem value="Close" onclick="CloseDoc()" />
      </popup>
</menu>
```

Kód 2: příklad XML [15]

Celkově lze říci, že formát XML je mocný nástroj pro ukládání a přenos strukturovaných dat mezi různými aplikacemi a systémy. Jeho schopnost definovat vlastní značky a hierarchii ho činí univerzálně použitelným formátem. [16]

3 VHODNÉ PROGRAMOVACÍ JAZYKY

3.1 Python

Python je všeobecný, vysokoúrovňový programovací jazyk. Narazit na něj můžeme snad ve všech odvětvích. Díky jeho jednoduché syntaxi a bohatému ekosystému knihoven je častou volbou hlavně pro začátečníky. Profesionálové ho však také hojně využívají pro menší projekty nebo otestování nových konceptů a myšlenek. [17]

Obrázek 9: Python [18]

Historie

Python byl poprvé vytvořen v roce 1991 Guidem van Rossumem, který chtěl vytvořit jazyk, který by byl intuitivní a efektivní zároveň. Název „Python“ byl inspirován oblíbeným televizním seriálem Monty Python's Flying Circus, což naznačuje, že tvůrci měli smysl pro humor. Python postupně získal popularitu a stal se jedním z nejuznávanějších programovacích jazyků dneška. [17]

Vlastnosti

- **Jednoduchá Syntaxe**: Jedna z hlavních výhod Pythonu spočívá v jeho jednoduché a čitelné syntaxi. S minimálními závorkami a speciálními znaky je kód napsaný v Pythonu snadno čitelný a srozumitelný.
- **Dynamická Typování**: Python je dynamicky typovaný jazyk, což znamená, že nemusíte explicitně deklarovat typ proměnných. Interpreter Pythonu automaticky určuje typ proměnných během běhu programu.

- **Rozsáhlá Knihovna**: Python disponuje obrovskou knihovnou modulů, které usnadňují práci ve všech možných oblastech, od webu až po vědecké výpočty. Knihovny jako NumPy, Pandas, Django a Flask jsou jen některé z mnoha dostupných, které rozšiřují možnosti Pythonu.
- **Multiplatformní Podpora**: Python je multiplatformní jazyk, což znamená, že můžete psát kód na jednom operačním systému a spouštět ho na jiném bez jakýchkoli změn. Tato vlastnost je velmi užitečná pro vývojáře pracující na různých platformách. [17]

Použití

- **Data Science**: Python se stal velmi významným v oblasti data science díky knihovnám jako NumPy, Pandas a Matplotlib, které umožňují analýzu a vizualizaci dat.
- **Automatizace**: Automatizace je dalším oborem, kde Python exceluje. Díky jednoduché syntaxi a bohatým knihovnám můžete snadno vytvořit skripty pro automatizaci opakujících se úkolů.
- **Webové Aplikace**: Díky frameworkům jako Django a Flask je Python populární volbou pro vývoj webových aplikací. Jeho čistá syntaxe a silná podpora knihoven usnadňují tvorbu robustních a škálovatelných webových aplikací.

3.2 Java

Java je silně typovaný, objektově orientovaný programovací jazyk, který se stal jedním z nejrozšířenějších nástrojů pro vývoj softwaru po celém světě. [19]

Obrázek 10: Java [20]

Historie

Java byla vyvinuta v roce 1995 Jamesem Goslingem a jeho týmem ve společnosti Sun Microsystems (nyní součástí Oracle Corporation). Cílem bylo vytvořit jazyk, který by byl nezávislý na platformě. Původně byla Java navržena pro použití v domácích spotřebičích, ale rychle se stala populární pro webové aplikace a další software. [19]

Vlastnosti

- **Objektová orientace**: Java je plně objektově orientovaný jazyk, což znamená, že vše v Javě je objekt, který má vlastnosti a metody. Tento přístup usnadňuje strukturování a správu kódu.
- **Přenositelnost**: Jednou z hlavních výhod Javy je její přenositelnost. Java kód může být napsán jednou a spuštěn na jakémkoli zařízení s JVM, což znamená, že aplikace mohou běžet na různých platformách bez nutnosti úprav.
- **Bezpečnost**: Bezpečnost je integrální součástí Javy. Systém řízení přístupu (Access Control) a mechanismy jako sandbox umožňují spouštění kódu v bezpečném prostředí, což chrání uživatele před nebezpečným kódem.
- **Velká standardní knihovna**: Java disponuje rozsáhlou standardní knihovnou, která obsahuje tisíce tříd a metod pro různé účely, včetně zpracování souborů, práce se sítěmi, GUI a mnoho dalšího. Tato knihovna snižuje potřebu psaní kódu od nuly a urychluje vývoj aplikací. [19]

Použití

- **Webové Aplikace**: Java je široce používána pro vývoj webových aplikací. Framework Spring poskytuje robustní a škálovatelné nástroje pro tvorbu moderních webových aplikací.
- **Mobilní Aplikace**: Java často používána pro vývoj mobilních aplikací na platformě Android (která je na Javě postavena).
- **Desktopové Aplikace**: Java je také stále populární pro vývoj desktopových aplikací díky svému silnému GUI frameworku Swing a dalším nástrojům, které umožňují tvorbu uživatelsky přívětivých aplikací.

3.3 C#

C# je vysokoúrovňový, objektově orientovaný programovací jazyk vyvinutý společností Microsoft jako klíčová součást C# .NET Framework. [21]

Obrázek 11: C# [22]

Historie

C# byl poprvé představen v roce 2000. Tým kolem Andersa Hejlsberga stál za vývojem jazyka, který by kombinoval sílu jazyka C++ s jednoduchostí jazyka Visual Basic. C# se rychle stal klíčovým jazykem pro vývojáře na platformě Windows. [21]

Vlastnosti

- **Objektová orientace**: C# je plně objektově orientovaný jazyk, který umožňuje vytváření modulárního a strukturovaného kódu pomocí tříd, dědičnosti, polymorfismu a dalších principů OOP.
- **Typová Bezpečnost**: Jedna z významných vlastností C# je jeho silná typová bezpečnost. C# umožňuje odhalit chyby v kódu již při kompilaci, což vede k méně chybám za běhu.
- **Moderní Syntaxe**: Syntaxe C# je elegantní a čistá, což usnadňuje čtení a psaní kódu. Funkce jako lambdy, LINQ a async/await poskytují moderní nástroje pro efektivní vývoj aplikací.
- **Integrace s .NET Framework**: C# je pevně integrován s .NET Framework, což poskytuje rozsáhlou knihovnu tříd pro různé účely včetně práce se soubory, práce se sítěmi, vytváření GUI a mnoho dalšího. [21]

Použití

- **Desktopové Aplikace**: C# je často používána pro vývoj desktopových aplikací pomocí platformy Windows Presentation Foundation (WPF) nebo Windows Forms.
- **Webové Aplikace**: S frameworkem ASP.NET může být C# použita pro vývoj webových aplikací. ASP.NET poskytuje silné nástroje pro vytváření dynamických a škálovatelných webových stránek a služeb.
- **Mobilní Aplikace**: Pomocí platformy Xamarin může být C# použita pro vývoj mobilních aplikací pro platformy iOS a Android.

3.4 PowerShell

PowerShell je skriptovací jazyk vyvinutý společností Microsoft, který je primárně určen pro automatizaci a správu systémů a služeb. Jedná se o výkonný nástroj, který umožňuje vývojářům a administrátorům provádět širokou škálu úkolů z příkazového řádku nebo pomocí skriptů. [23]

Obrázek 12: PowerShell [24]

Historie

PowerShell byl poprvé uveden v roce 2006 jako nástupce starších skriptovacích jazyků, jako je například cmd.exe. Byl vyvinut s cílem poskytnout vývojářům a administrátorům mocný nástroj pro automatizaci rutinních úkolů a správu systémů. [23]

Vlastnosti

- **Objektová orientace**: PowerShell je objektově orientovaný jazyk, což umožňuje jednoduché manipulace s daty a výstupy z příkazů.
- **Integrace s .NET Framework**: PowerShell je postaven na .NET Framework, což umožňuje ve skriptech plně využívat sílu .NET knihoven a propojení s jinými technologiemi, jako je C#. [23]

4 REGULÁRNÍ VÝRAZY

Regulární výrazy jsou mocným nástrojem pro manipulaci s textovými řetězci. Jsou to vzory, které definují soubor řetězců. Tyto vzory mohou být použity k různým účelům při práci s textem.

výraz	význam	výraz	význam	výraz	význam
\w	znak (A-Z)	.	cokoli	()	výběr
\s	mezera	+	1 nebo více	[]	rozsah
\d	číslice	*	0, 1 nebo více	^	začátek
\n	nový řádek	?	0 nebo 1	$	konec

Tabulka 1: seznam základních regulárních výrazů

Použití regulárních výrazů

- **Vyhledávání v textu**: Regulární výrazy jsou často používány k vyhledávání konkrétních vzorků v textu. Například, hledání všech e-mailových adres v textu může být provedeno pomocí regulárního výrazu.
- **Nahrazování textu**: Mohou být také použity k nahrazení částí textu jinými řetězci nebo prázdným řetězcem, pokud chceme daný vzor odstranit.
- **Validace vstupních dat**: Regulární výrazy jsou běžně používány k ověřování formátu vstupních dat jako jsou jména, emaily, telefonní čísla apod.
- **Extrakce dat**: Mohou být použity k extrakci určitých částí textu.

Nástroje pro práci s regulárními výrazy

Existuje mnoho nástrojů a knihoven pro práci s regulárními výrazy v různých programovacích jazycích. Některé z nich zahrnují:

- **Python**: Python má vestavěnou knihovnu *re* pro práci s regulárními výrazy.
- **JavaScript**: V JavaScriptu lze použít objekt *RegExp* nebo metody řetězce jako match, test a replace.
- **C#**: V C# je regulární výrazy snadné používat pomocí třídy *Regex* z .NET frameworku. Tato třída poskytuje metody pro vyhledávání a manipulaci s textem.
- **Online nástroje**: Existují také různé online nástroje, které vám umožní testovat a ladit regulární výrazy, jako je například Regex101 nebo RegExr.

II. PRAKTICKÁ ČÁST

5 VOLBA PROGRAMOVACÍHO JAZYKA

A) Python

Hlavní výhodou Pythonu je jeho jednoduchost. Python nabízí snadnou syntaxi a rychlý vývoj díky své čitelnosti a jednoduchosti syntaxe, což usnadňuje rychlou tvorbu aplikací. Má také bohatý ekosystém knihoven pro práci se síťovými protokoly, datovou analýzou a manipulací s daty. Může tedy výrazně zjednodušit vývoj aplikace. Jeho schopnost rychlého prototypování a interaktivní práce může být výhodná pro tvorbu a testování bezpečnostních mechanismů.

Nicméně, Python je interpretovaný jazyk, což může znamenat pomalejší běh aplikace ve srovnání s kompilovanými jazyky.

B) Java

Java nabízí širokou podporu knihoven, které mohou být užitečné pro vývoj. Díky své dobré rovnováze mezi bezpečností a výkonem může být Java vhodná volba pro aplikace, které manipulují s citlivými daty. Navíc, Java je platformě nezávislý jazyk, což znamená, že aplikace napsané v Javě mohou být spuštěny na různých operačních systémech.

Je zde však problém v podobě složitější syntaxe, něž je u jazyků jako Python nebo C#. Dále také aplikace v Javě obvykle vyžadují více paměti a mají vyšší nároky na výkon.

C) C#

C# nabízí bezpečnost typů a statickou typovou kontrolu, což může snížit počet chyb v kódu a zlepšit celkovou bezpečnost aplikace. Díky úzké integraci s platformou Windows je C# vhodný pro tvorbu aplikací, které budou používány v prostředí Windows. To může být výhodné pro penetrační testy, které cílí na zařízení nebo aplikace na platformě Windows.

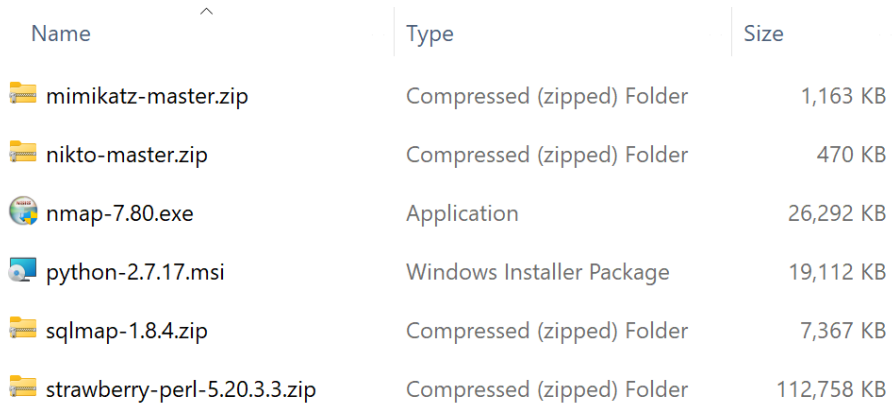
Nicméně, C# může mít menší ekosystém knihoven pro penetrační testy ve srovnání s Pythonem nebo Javou, což může znamenat více vlastní implementace funkcí.

Po zvážení těchto možností volím pro tvorbu aplikace jazyk C#. Penetrační testy budou vytvořeny prostřednictvím skriptovacího jazyka PowerShell. Aplikace v jazyce C# bude pak tyto testy spouštět a z výstupu tvořit přehledný report ve formátu PDF.

6 POWERSHELL SKRIPTY

Pro efektivní správu skriptů a nástrojů byla vytvořena specifická struktura složek. Skripty, které jsou používány pro automatizaci různých úloh testu, jsou umístěny v adresáři s názvem *Scripts*. Tento adresář slouží jako domovská složka pro všechny testy, jež aplikace používá.

Ve stejném základním adresáři se nachází také adresář nazvaný *Tools*, který slouží k uchování nástrojů, jež používám jako součást automatizačních procesů. Adresář obsahuje instalační soubory a portable aplikace (v balících ZIP) a další potřebné nástroje, které jsou klíčové pro naši práci.



Obrázek 13: obsah adresáře Tools

Základní adresář je umístěn na specifické cestě, jež bude popsána později (Kapitola 8). Tato struktura umožňuje snadný přístup k výchozím souborům a nástrojům, které jsou třeba k dokončení testu.

```
$scriptsDirectory = Split-Path -Path $MyInvocation.MyCommand.Path -Parent
$baseDirectory = Split-Path -Path $scriptsDirectory -Parent
$toolsDirectory = "$baseDirectory\Tools"
```

Kód 3: přesun základním adresářem

Aby bylo možné se skriptem v kódu efektivně pracovat, obsahuje vstupní parametry, které lze při jeho zavolání měnit. Jejich definici a defaultní nastavení lze vidět zde:

```
param (
    [string]$desktopPath = [System.Environment]::GetFolderPath('Desktop'),
    [string]$target = "127.0.0.1"
)
```

Kód 4: parametry skriptu

Když je skript spuštěn, prvním krokem je prohledání adresáře *Tools* a identifikace potřebných nástrojů. Skript poté provede instalaci nebo rozbalení (v případě zip archivů) těchto nástrojů tak, aby byly připraveny k použití. Pokud byla tato přípravná fáze úspěšně dokončena, skript spustí konkrétní nástroj s definovanými argumenty.

Nakonec je výstup z provedených operací přesměrován do souboru. Tento soubor slouží jako záznam provedených akcí a výsledků.

6.1 Nmap

Nástroj Nmap (Network Mapper) nemá portable verzi. Je tedy třeba jej nainstalovat. Pokud Nmap ještě nainstalován není (nenachází se na definované cestě), skript se jej pokusí nainstalovat a vypíše hlášku o úspěchu, či neúspěchu.

Zde nastal problém s poslední verzí nástroje – *Nmap 7.94*. Ta totiž v základu neumožnuje instalaci na pozadí (tzv. silent install). Tuto možnost obsahuje pouze OEM Edition, která je zpoplatněná. Z toho důvodu jsem se rozhodl použít starší verzi – *Nmap 7.80*, která ještě na pozadí instalovat lze.

```
$nmapPath = "C:\Program Files (x86)\Nmap\nmap.exe"

if (-not (Test-Path $nmapPath)) {
    $installerPath = "$toolsDirectory\nmap-7.80.exe"
    try {
        # silent install nmap
        Start-Process -FilePath $installerPath /S -Wait
        Write-Host "$(Split-Path $installerPath -Leaf) successfully installed."
    } catch {
        Write-Host "$(Split-Path $installerPath -Leaf) installation failed."
    }
}
```

Kód 5: instalace Nmap

Dále je nutné určit cestu k výstupu z nástroje a pokud adresář na cestě neexistuje, pak jej vytvořit, aby mohly být výsledky úspěšně zaznamenány.

```
$outputsDirectory = "$desktopPath\Outputs"
$outputPath = "$outputsDirectory\nmap.xml"

if (-not (Test-Path $outputsDirectory)) {
    New-Item -Path $outputsDirectory -ItemType Directory
}
```

Kód 6: vytvoření cesty k výstupu

Nyní už stačí jen nástroj zavolat s danými argumenty.

```
$nmapArgs = "-p- -sS -A -oX $outputPath"
# -p- : search all 65535 ports
# -sS : scan of ports TCP SYN
# -A : finds services and operating system info
# -oX : gives output in XML format
$args = "$target $nmapArgs"


if (Test-Path $nmapPath) {
    Start-Process -FilePath $nmapPath -ArgumentList $args -NoNewWindow -Wait
}
```

Kód 7: spuštění nástroje Nmap

6.2 sqlmap

Stejně jako v předchozím případě i zde je třeba si nástroje pro spuštění testu nejprve připravit. Na rozdíl od Nmapu, je však sqlmap dostupný ve verzi portable. Není jej tedy třeba instalovat, stačí ho rozbalit a spustit.

```
$sqlmapPath = "$toolsDirectory\sqlmap\sqlmap.py"

if (-not (Test-Path $sqlmapPath)) {
    $zipPath = "$toolsDirectory\sqlmap-1.8.4.zip"
    try {
        Add-Type -AssemblyName System.IO.Compression.FileSystem
        [System.IO.Compression.ZipFile]::ExtractToDirectory($zipPath,
        "$toolsDirectory\sqlmap")
        Write-Host "$(Split-Path $zipPath -Leaf) successfully extracted."
    } catch {
        Write-Host "$(Split-Path $zipPath -Leaf) Extraction failed."
    }
}
```

Kód 8: rozbalení nástroje sqlmap

V tomto případě se však jedná o Python script (.py) a k jeho spuštění je třeba doinstalovat interpreter jazyka Python. Pro menší objem dat a jednodušší instalaci jsem volil starší verzi Python 2.7.

```
$pythonPath = "C:\Python27\python.exe"

if (-not (Test-Path $pythonPath)) {
    $installerPath = "$toolsDirectory\python-2.7.17.msi"
    $args = "/qn /i `"$installerPath`""
    try {
        Start-Process -FilePath msiexec.exe -ArgumentList $args -Wait
        Write-Host "$(Split-Path $installerPath -Leaf) successfully installed."
    } catch {
        Write-Host "$(Split-Path $installerPath -Leaf) installation failed."
    }
}
```

Kód 9: instalace Pythonu

Po úspěšné instalaci Pythonu zbývá pouze nástroj zavolat a přesměrovat výstup.

```
if ((Test-Path $pythonPath) -and (Test-Path $sqlmapPath)) {
    $command = "$pythonPath $sqlmapPath -u $target -crawl=1 --batch"
    Invoke-Expression $command | Out-File -FilePath $outputPath -Append
}
```

Kód 10: spuštění nástroje sqlmap

6.3 Mimikatz

U nástroje Mimikatz je přípravná část obdobná jako v předchozím případě (stačí jej rozbalit). Musí se však myslet na jednu věc. Nástroj Mimikatz může být antiviry vyhodnocen jako škodlivý software. Proto je pro správnou funkci nutné vypnout antivirovou ochranu. [25]

V PowerShellu je sice možnost vypnout monitoring antiviru MS Defender příkazem, ten však neprojde, pokud je zapnutá tzv. Ochrana před falšováním (Tamper Protection).

```
Set-MpPreference -DisableRealtimeMonitoring $true
```

Kód 11: vypnutí ochrany

Jakmile je ochrana vypnuta nástroj je možné spustit.

```
if (Test-Path $mimikatzPath) {
    Start-Process -FilePath $mimikatzPath -ArgumentList $args -NoNewWindow -Wait
-RedirectStandardOutput $outputPath
}
```

Kód 12: spuštění nástroje Mimikatz

6.4 Nikto

Ke spuštění nástroje Nikto je třeba nejprve nainstalovat interpreter jazyka perl a stejně jako v předchozím případě rozbalit nástroj samotný.

Opět zde platí, že antivirus musí být vypnutý, jinak pravděpodobně dojde k přesunu Nikta do karantény nebo rovnou k jeho smazání.

```
if ((Test-Path $perlPath) -and (Test-Path $niktoPath)) {
    $command = "$perlPath $niktoPath -h $target"
    Invoke-Expression $command | Out-File -FilePath $outputPath -Append
}
```

Kód 13: spuštění nástroje Nikto

6.5 Windows cmd

V posledním skriptu již není třeba nic instalovat ani jinak připravovat, jelikož cmd (příkazový řádek) se všemi jeho funkcemi je integrován v operačním systému Windows.

Stačí zde pouze definovat cestu pro výstup a dále jen volat příkazy jeden za druhým.

```
$commands = @(
    "whoami",
    "systeminfo",
    "ipconfig /all",
    "tracert $target",
    "nslookup $target",
    "netstat -a"
)

foreach ($cmd in $commands) {
    "`n# Command: $cmd" | Out-File -FilePath $outputPath -Append
    Invoke-Expression $cmd | Out-File -FilePath $outputPath -Append
}
```

Kód 14: vyvolání příkazů cmd

7 TVORBA DOKUMENTU

Pro vytváření dokumentu byla využita knihovna (NuGet) PDFSharp.

7.1 PDFSharp

PDFsharp je open-source knihovna napsaná v jazyce C#, která umožňuje tvorbu a manipulaci s PDF soubory. Slouží k vytváření, úpravě a zpracování PDF dokumentů přímo z aplikací vyvinutých v jazyce C# nebo .NET Framework. Tato knihovna poskytuje širokou škálu funkcí, včetně přidávání textu, obrázků, grafů, formátování, rozdělování stránek, tvorbu tabulek a mnoho dalšího. [25]

Obrázek 14: PDFsharp [26]

7.1.1 Příklady použití

Inicializace dokumentu zahrnuje vytvoření instance dokumentu, stránky a grafiky na dané stránce.

```
PdfDocument document = new();
PdfPage page = document.AddPage();
XGraphics gfx = XGraphics.FromPdfPage(page);
document.Save(outputPath);
document.Close();
```

Kód 15: tvorba dokumentu

Pokud chceme vypsat text, je nejprve nutné vytvořit font, brush („štětec“), a bod umístění začátku. Samotný text se pak vypisuje do instance grafiky (přiřazené dané straně dokumentu) voláním metody *DrawString*.

```
XFont font = new("Arial", 12, XFontStyle.Regular);
XBrush brush = XBrushes.Black;
XPoint position = new XPoint(50, 50);
string text = "Hello world";
gfx.DrawString(text, font, brush, position);
```

Kód 16: výpis textu

Pro vykreslení obrázku (na určité pozici) slouží metoda *DrawImage*.

```
XImage image = XImage.FromFile("image.png");
gfx.DrawImage(image, 100, 100);
```

Kód 17: vykreslení obrázku

Pro kreslení geometrie existuje také řada metod jako: *DrawLine*, *DrawRectangle*, *DrawEllipse*, *DrawPolygon* atd.

```
XPoint point1 = new XPoint(100, 100);
XPoint point2 = new XPoint(page.Width - 100, page.Height - 100);

XPen pen = new(XColors.Coral, 1);
gfx.DrawLine(pen, point1, point2);
```

Kód 18: vykreslení čáry

```
XRect rectangle1 = new(point1, point2);
gfx.DrawRectangle(pen, rectangle1);

double width = 50;
double height = 50;
XRect rectangle2 = new(100, 100, width, height);
gfx.DrawRectangle(brush, rectangle2);
```

Kód 19: vykreslení obdélníku

7.1.2 Nutná nastavení projektu:

Aby PDFsharp fungoval správně, je třeba do projektu zahrnout následující knihovny:

```
<PackageReference Include="PDFsharp" Version="1.50.5147" />
<PackageReference Include="System.Drawing.Common" Version="8.0.4" />
<PackageReference Include="System.Text.Encoding.CodePages" Version="8.0.0" />
```

Kód 20: knihovny

```
// Ensuring that necessary encoding support is available for PDFsharp
Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);
```

Kód 21: zajištění podpory kódování

Je nutné zajistit, že *PDFsharp* dokáže správně kódovat text pomocí různých znakových sad, což umožní generovat soubory PDF se správným vykreslením textu, zejména pro jazyky nebo znaky, které nepokrývá výchozí kódování.

7.2 Tvorba stránek

Tvorbu a úpravu jednotlivých stránek reportu zajišťuje třída s názvem *PDFCreator*. Metody této třídy jsou odpovědné za vypisování textu, vytváření nových stránek, řádkování, stránkování a mnoho dalšího. Hlavní metoda této třídy se jmenuje *Create* a právě tato metoda je zavolána po úspěšném dokončení všech testů.

Parametry metody jsou pracovní složka, tedy místo, kam se ukládají výstupy z testů i výsledný report, a binární proměnná rozhodující o tom, zda se dokument po vytvoření otevře.

```
public static void Create(string workDirectory, bool postOpen)
{
    Initialize();
    ArrangeFirstPage();

    string outputsDirectory = Path.Combine(workDirectory, "Outputs");

    WriteReports(outputsDirectory);
    Save(workDirectory, postOpen);
}
```

Kód 22: metoda *Create*

Na začátku je nutné provést inicializaci dokumentu, podobně jako to bylo v příkladech použití *PDFsharp*. Titulní strana má specifický vzhled oproti zbytku dokumentu, takže se musí vytvářet samostatně.

```
private static void WriteReports(string workDirectory)
{
    string[] xmlPaths = Directory.GetFiles(workDirectory, "nmap*.xml");
    foreach (string path in xmlPaths)
    {
        Nmap.AddReport(path);
    }

    string[] txtPaths = Directory.GetFiles(workDirectory, "*.txt");
    foreach (string path in txtPaths)
    {
        if (path.Contains("sqlmap"))
        {
            Sqlmap.AddReport(path);
        }
        ...
```

Kód 23: metoda *WriteReports*

Metoda *WriteReports* prohledá složku s výstupy testů a pokud je nalezne, pak zavolá *AddReport* příslušné třídy. Každý nástroj má vlastní třídu pro zpracování daného výstupu a výpis dat do dokumentu.

Důležité metody třídy PDFCreator:

a) Write

Asi nejdůležitější metodou celé aplikace je metoda *Write*. Tato metoda se stará o výpis textu, a proto ji volají téměř všechny ostatní metody.

V úvodu je zde přiřazení hodnot volitelných parametrů. Tato část zde musí být, protože výchozí hodnotou parametru nemůže být objekt. Následně jsou definovány pomocné proměnné a kontroluje se, zda aktuální pozice nepřesahuje prostor vymezený pro text na stránce. Pokud ano, vytvoří se nová stránka, na kterou je text vypsán.

V neposlední řadě, je zde také možnost udělat odrážku (*bullet*) před textem. Volba je řízena binární proměnnou, přičemž pokud se má odrážka udělat, zavolá se metoda, která ji vypíše a zbylý text odsadí.

```
public static void Write(string text, bool bullet = false, XFont? font = null,
    XBrush? brush = null, double posX = 0)
{
    posX = posX == 0 ? lastPosX : (padding + posX);
    font = font == null ? normalFont : font;
    brush = brush == null ? XBrushes.Black : brush;

    double stringWidth = 0;
    double charWidth;
    double lineWidth;
    double lineHeight = font.GetHeight();

    if (posY >= page.Height - padding)
    {
        NewPage();
        NewLine(lineHeight);
    }

    if (bullet) { posX = WriteBullet(posX); }
```

Kód 24: metoda *Write*, část 1

Druhá část metody *Write* se věnuje vypisování samotných znaků textu. Text se vypisuje po znacích kvůli řádkování. Pokud při výpisu dojde text až na hranici prostoru, přejde se na nový řádek.

Pozice v řádku je definována jako proměnná třídy, tedy mimo metodu *Write*. Díky tomu lze metodu opakovaně volat a výpis textu pokračuje tam, kde předchozí skončil.

```
    lineWidth = page.Width - posX - padding;

    foreach (char character in text)
    {
        charWidth = gfx.MeasureString(character.ToString(), font).Width;

        if (stringWidth + charWidth > lineWidth)
        {
            NewLine(lineHeight);

            if (posY >= page.Height - padding)
            {
                NewPage();
                NewLine(lineHeight);
            }

            stringWidth = 0;
        }

        gfx.DrawString(character.ToString(), font, brush,
            new XPoint(posX + stringWidth, posY));

        stringWidth += charWidth;
    }
    lastPosX = posX + stringWidth;
}
```

Kód 25: metoda *Write*, část 2

b) WriteLine

Další neméně důležitou metodou je *WriteLine*. Je „dvojčetem“ metody *Write* a sama ji také volá, aby kód nebyl zbytečně redundantní. Rozdíl je zde pouze v tom, že *WriteLine* vždy začíná na začátku nového řádku.

```
public static void WriteLine(string text, bool bullet = false,
    XFont? font = null, XBrush? brush = null, double posX = 0)
{
    font = font == null ? normalFont : font;
    double lineHeight = font.GetHeight();

    if (posY >= page.Height - padding) { NewPage(); }
    NewLine(lineHeight);
    lastPosX = padding; // reset X

    Write(text, bullet, font, brush, posX);
}
```

Kód 26: metoda *WriteLine*

c) NewPage a NewLine

Tyto 2 metody, jak napovídá jejich název, se starají o vytvoření nové stránky a přechod na nový řádek. Při vytváření nové stránky je také přidána hlavička (*WriteHeader*) a řádkování (*WritePaging*). Obojí zde bude popsáno později.

```
public static void NewPage(bool firstPage = false)
{
    page = document.AddPage();
    gfx.Dispose();
    gfx = XGraphics.FromPdfPage(page);
    posY = padding; //reset Y

    WriteHeader(firstPage);
    WritePaging();
}
```

Kód 27: metoda *NewPage*

```
public static void NewLine(double lineHeight)
{
    posY += (lineHeight * 1.25);
}
```

Kód 28: metoda *NewLine*

d) WriteHeader a WritePaging

WriteHeader má možné 2 scénáře. Pokud je aktuální strana první stranou reportu, vypíše velký nadpis a čas vytvoření. V případě, že není první, vypíše pouze název malým písmem do rohu.

```
private static void WriteHeader(bool firstPage)
{
    if (firstPage)
    {
        WriteLine($"{champterName} scan report", false,
            new XFont("Arial", 20, XFontStyle.Bold), champterColor);
        WriteLine($"Created: ", false, boldFont);
        Write(dateTime);
        WriteSeparator(5);
    }
    else
    {
        XSize size = gfx.MeasureString(champterName, boldFont);

        gfx.DrawString(champterName, boldFont, champterColor,
        new XPoint(
            page.Width - padding - size.Width,
            padding - size.Height * 0.5));
        gfx.DrawLine(new XPen(XColors.Black, 0.5),
            new XPoint(padding, padding),
            new XPoint(page.Width - padding, padding));
    }
}
```

Kód 29: metoda *WriteHeader*

Jelikož je tento text vypisován až na hranici oblasti, nemůže zde být použita metoda *Write* (text by měl tendenci přesunout se na nový řádek).

Totéž platí i u stránkování, které se opět vypisuje do pravého rohu.

```
private static void WritePaging()
{
    paging++;
    string text = $"page {paging}";
    XSize size = gfx.MeasureString(text, boldFont);

    gfx.DrawString(text, boldFont, XBrushes.Black,
        new XPoint(
            page.Width - padding - size.Width,
            page.Height - padding + size.Height * 1.25));
    gfx.DrawLine(new XPen(XColors.Black, 0.5),
        new XPoint(padding, page.Height - padding),
        new XPoint(page.Width - padding, page.Height - padding));
}
```

Kód 30: metoda *WritePaging*

7.3 Zpracování výstupů z testů

7.3.1 XML

Nástroj Nmap má, jako jediný z použitých testovacích programů, možnost výstupu ve formátu XML, tedy strukturovaný text. Takový výstup je možné snadno a elegantně zpracovat bez procházení celého dokumentu po řádcích a hledání vzorů. Stačí zde nalézt uzel, jež obsahuje požadované informace a v cyklu vypisovat hodnoty všech jeho atributů.

AddReport je hlavní metoda třídy Nmap a je odpovědná za výpis dat z XML do PDF reportu. Na začátku se upraví nastavení hlavičky první strany. Dále je načten výstup z testu, jež je zpracován metodami jako *WriteRunInfo* nebo *WritePorts*.

```
public static void AddReport(string path)
{
    PDFCreator.champterName = "Nmap";
    PDFCreator.champterColor = XBrushes.Red;
    PDFCreator.NewPage(true);

    xml.Load(path);
    WriteRunInfo();
    WritePorts();
    ...
```

Kód 31: metoda *AddReport* (Nmap)

Metoda *WriteRunInfo* vypisuje základní informace o testu jako verze nástroje nebo čas běhu. Voláním pomocné metody *GetAttributeValue* získává hodnotu konkrétního atributu v daném uzlu a tu následně vypíše pomocí *Write*.

```
private static void WriteRunInfo()
{
    string version = GetAttributeValue("nmaprun", "version");
    string args = GetAttributeValue("nmaprun", "args");
    string elapsed = GetAttributeValue("runstats/finished", "elapsed");
    string exit = GetAttributeValue("runstats/finished", "exit");

    XBrush color;
    if (exit == "success") { color = XBrushes.Green; }
    else { color = XBrushes.Red; }

    PDFCreator.WriteTitle("Nmap: ");
    PDFCreator.WriteLine("Version: ", true, PDFCreator.boldFont);
    PDFCreator.Write(version);
    ...
```

Kód 32: metoda *WriteRunInfo* (Nmap)

Metoda *GetAttributeValue* je tzv. přetížená (overload) metoda, což znamená, že má více variant sebe sama. Díky tomu se jí může předat jak název uzlu, tak i samotný uzel, ve kterém se nachází potřebný atribut a kompilátor si podle typu parametru vybere správnou variantu této metody.

Této vlastnosti využívá následující metoda *WritePorts*, která vypisuje hodnoty portů v cyklu iterujícím po uzlech.

```
private static void WritePorts()
{
    XmlNodeList portNodes = xml.GetElementsByTagName("port");
    XBrush stateColor;

    if (portNodes.Count > 0)
    {
        PDFCreator.WriteTitle("Ports: ");
        foreach (XmlNode node in portNodes)
        {
            string protocol = GetAttributeValue(node, "protocol");
            string portid = GetAttributeValue(node, "portid");
            XmlNode? stateNode = node.SelectSingleNode("state");
            string state = stateNode != null ? GetAttributeValue(stateNode,
               "state") : "none";
            XmlNode? serviceNode = node.SelectSingleNode("service");
            string service = serviceNode != null ? GetAttributeValue(serviceNode,
               "name") : "none";

            // Writing to document
            ...
```

Kód 33: metoda *WritePorts* (Nmap)

7.3.2 TXT

Při zpracovávání výstupu ze souboru .txt, je nejprve nutné vytvořit z prostého textu vlastní strukturu. K tomuto účelu mi zde poslouží regulární výrazy a pomocná třída *Element*.

```
public class Element
{
    public string Name { get; set; }
    public string Value { get; set; }
    public List<Element> Children { get; set; }
    public int childrenCount { get; set; }

    public Element(string name = "", string value = "")
    {
        Name = name;
        Value = value;
        Children = new List<Element>();
    }

    public void AddChild(Element child)
    {
        Children.Add(child);
        childrenCount++;
    }
}
```

Kód 34: pomocná třída *Element*

Pomocí regulárních výrazů jsou textu vyhledávány specifické vzory. Tyto části textu jsou následně ukládány do instancí třídy *Elements*, a jelikož třída obsahuje i seznam potomků, lze vytvořit vícevrstvou strukturu podobnou jako ve formátu XML.

Tento převod dat má na starost metoda *ReadTxt*. Na začátku je definován seznam elementů, který představuje požadovanou strukturu a jeden element pro aktivní záznam (dočasná data).

```
public static List<Element> ReadTxt(string sqlmapPath)
{
    List<Element> records = new();
    string[] lines = File.ReadAllLines(sqlmapPath);

    Element activeRecord = new();
    bool hasUrl = false;

    string pattern_end = @"\[\d+\/\d+\]\s\w+\:";
    string pattern_url = @"GET http://(.*)";
    string pattern_message = @"\[[^\]]+\]\s\[([^\]]+)\]\s(.*)";
    string pattern_version = @"\{([\d.]+)\#\w+\}";
```

Kód 35: metoda *ReadTxt*, část 1 (sqlmap)

Cyklus iteruje po řádcích a porovnává shodu se všemi vzory. Při shodě se vzorem pro adresu (*match_url*) se naplní aktivní záznam. V následujících iteracích bude shoda se vzorem pro zprávu (*match_message*), čímž se naplní potomstvo aktivního záznamu a jakmile nastane shoda s koncem záznamu (*match_end*), aktivní záznam se přidá do výsledné struktury a postup se znovu opakuje.

```
    foreach (string line in lines)
    {
        Match match_version = Regex.Match(line, pattern_version);
        Match match_end = Regex.Match(line, pattern_end);
        Match match_url = Regex.Match(line, pattern_url);
        Match match_message = Regex.Match(line, pattern_message);

        if (match_end.Success)
        {
            if (hasUrl)
            {
                records.Add(activeRecord);
            }
        }
        else if (match_url.Success)
        {
            activeRecord = new Element("", match_url.Groups[1].Value);
            hasUrl = true;
        }
        else if (match_message.Success)
        {
            Element child = new Element(match_message.Groups[1].Value,
                match_message.Groups[2].Value);
            activeRecord.AddChild(child);
        }
        else if (match_version.Success)
        {
            version = match_version.Groups[1].Value;
        }
    }
    return records;
}
```

Kód 36: metoda *ReadTxt*, část 2 (sqlmap)

Výstup z testu *sqlmap* byl vybrán záměrně pro demonstraci této logiky, protože zde je struktura pouze 2 vrstvá (seznam v seznamu). Lze však jít i hlouběji jako je tomu u zpracování reportu z *Mimikatz*.

Jakmile jsou data strukturovaná, zbývá je jen vypsat do dokumentu. O to se zde stará metoda *WriteRecords*, která je spolu s *ReadTxt* volána při vytváření reportu.

```
public static void AddReport(string path)
{
    PDFCreator.champterName = "SQLmap";
    PDFCreator.champterColor = XBrushes.Green;
    PDFCreator.NewPage(true);

    List<Element> urls = ReadTxt(path);

    PDFCreator.WriteTitle("SQLmap: ");
    if (version != null)
    {
        PDFCreator.WriteLine("Version: ", true, PDFCreator.boldFont);
        PDFCreator.Write(version);
    }

    WriteRecords(urls);
}
```

Kód 37: metoda *AddReport* (sqlmap)

Metoda *WriteRecords* postupně ve dvou cyklech prochází strukturou a všechna data jsou vypisována do dokumentu. Tímto způsobem lze text snadno odsazovat podle úrovně ve struktuře a report je tak mnohem lépe čitelný.

```
public static void WriteRecords(List<Element> records)
{
    int counter = 0;
    PDFCreator.WriteTitle("Records: ");
    foreach (var url in records)
    {
        counter++;
        PDFCreator.WriteLine($"URL {counter}: ", true, PDFCreator.boldFont);
        PDFCreator.Write(url.Value);

        foreach (Element child in url.Children)
        {
            XBrush color;
            if (child.Name == "INFO") { color = XBrushes.DodgerBlue; }
            else if (child.Name == "WARNING") { color = XBrushes.Goldenrod; }
            else if (child.Name == "ERROR") { color = XBrushes.Red; }
            else { color = XBrushes.Black; }

            PDFCreator.WriteLine($"[{child.Name}] ",
                false, PDFCreator.normalFont, color, 15);
            PDFCreator.Write(child.Value);
        }
        PDFCreator.NewLine(5);
    }
}
```

Kód 38: metoda *WriteRecords* (sqlmap)

8 PRÁCE SE SOUBORY

Aplikace pro svou činnost vyžaduje další externí soubory jako jsou nástroje pro penetrační testy nebo skripty. Bylo tedy nutné tyto soubory nějakým způsobem připojit k samotné aplikaci.

Pro textové soubory či obrázky se běžně používají tzv. *embedded resources*, jež se při publikování (vytvoření spustitelného souboru) zahrnou do výsledného .exe souboru. V kódu se k nim přistupuje pomocí třídy *ResourceManager* nebo *Assembly*.

Zde mi však přišlo vhodnější přidat soubory do projektu jako tzv. *content files*. Na rozdíl od resources se k nim přistupuje jednoduše přes specifickou cestu v adresářové struktuře buildu. Build se pak vytváří při prvním spuštění aplikace v dočasných souborech uživatele včetně naších *content files*. Cesta adresáře je následující:

C:\Users\USERNAME\AppData\Local\Temp\.net\ASSEMBLY\BUILD

Soubory *content files* jsou definovány v nastavení projektu (.csproj). Jejich automatizované přidání spolu s tvorbou spustitelného souboru (.exe) řeší pomocný program Publisher, jež zde bude popsán později.

```
<Content Include="ÚPLNÁ_CESTA_K_PŘIDÁVANÉMU_SOUBORU">
  <CopyToOutputDirectory>Always</CopyToOutputDirectory>
  <Link>REALTIVNÍ_CESTA_SOUBORU_V_ADRESÁŘI_BUILDU</Link>
</Content>
```

Kód 39: definice contentu v projekru

V kódu se k adresáři přistupuje takto:

```
string baseDirectory = AppDomain.CurrentDomain.BaseDirectory;
```

Kód 40: baseDirectory

9 PUBLIKOVÁNÍ PROJEKTU

Pro vytvoření spustitelného souboru byla vytvořena jednoduchá konzolová aplikace s názvem *Publisher*. Hlavní metoda se nazývá *PublishProject*. Jejím úkolem je změnit požadovaná nastavení v projektu a vytvořit spustitelný soubor ve výstupním adresáři.

```
private static void PublishProject(string projectDirectory,
    string contentDirectory, string assemblyName)
{
    string? projectFile = Directory.GetFiles(projectDirectory, "*.csproj").
    FirstOrDefault();
    if (projectFile != null)
    {
        AddContent(projectFile, contentDirectory);
        AddPublishSettings(projectFile, assemblyName);
        string result = CreateExecutable(projectDirectory, outputDirectory);
        Console.WriteLine(result);
    }
}
```

Kód 41: metoda *PublishProject*

Projektový soubor C# aplikace je vlastně dokument ve formě XML a skládá se tedy z jednotlivých uzlů (nodes). K přidání těchto uzlů slouží následující metoda *AddNode*.

Metoda *AddNode* nejpve ověří, zda se uzel s daným jménem v dokumentu již nenachází. Pokud daný uzel nenalezne, přidá jej. Pokud ale ano, změní pouze jeho hodnotu. Díky tomu nehrozí vznik duplicit v dokumentu.

```
private static XmlNode? AddNode(XmlNode parentNode, string name,
    string? value = null)
{
    XmlDocument? document = parentNode.OwnerDocument;
    if (document != null)
    {
        XmlNode? node = parentNode.SelectSingleNode(name);
        if (node == null)
        {
            node = document.CreateElement(name);
            parentNode.AppendChild(node);
        }
        if (value != null)
        {
            node.InnerText = value;
        }
        return node;
    }
    return null;
}
```

Kód 42: metoda *AddNode*

Jak již bylo zmíněno, pro vytvoření samostatného .exe souboru je třeba změnit některá nastavení projektu. Za tyto změny je zodpovědná metoda *AddPublishSettings*. Tato metoda načte soubor .csproj a přidá potřebná nastavení.

```
private static void AddPublishSettings(string projectFile, string assemblyName)
{
    XmlDocument document = new XmlDocument();
    document.Load(projectFile);

    XmlNode? propertyGroupNode = document.SelectSingleNode("Project/
    PropertyGroup");
    if (propertyGroupNode != null)
    {
        AddNode(propertyGroupNode, "AssemblyName", assemblyName);
        AddNode(propertyGroupNode, "PublishSingleFile", "true");
        AddNode(propertyGroupNode, "SelfContained", "true");
        AddNode(propertyGroupNode, "IncludeAllContentForSelfExtract", "true");
        AddNode(propertyGroupNode, "IncludeNativeLibrariesForSelfExtract",
        "true");
    }

    document.Save(projectFile);
}
```

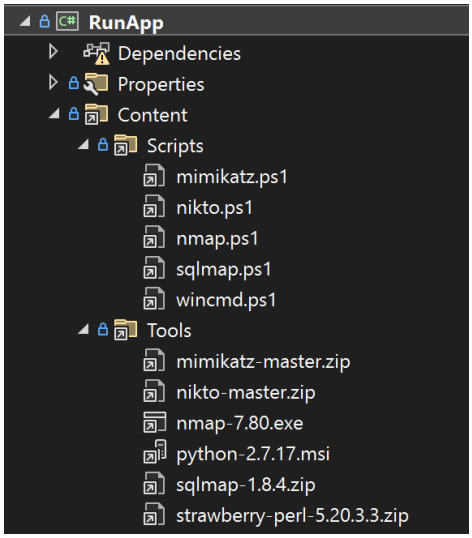
Kód 43: metoda *AddPublishSettings*

A konečně jsme u přidávání content files. Metoda *AddContent* pracuje obdobně jako předchozí *AddPublishSettings*. Nejprve načte soubor .csproj. Pokud v projektu není uzel *ItemGroup*, přidá ho. Pokud jsou v uzlu *ItemGroup* nějaké uzly s názvem *Content*, odstraní je. Následně prohledá *contentDirectory* a přidá uzly nové, což lze vidět v kódu níže.

```
string[] content = Directory.GetFiles(contentDirectory, "*",
    SearchOption.AllDirectories);

if (content.Length > 0)
{
    Console.WriteLine("Content files: ");
    foreach (string sourcePath in content)
    {
        Console.WriteLine(sourcePath);

        XmlElement contentElement = document.CreateElement("Content");
        contentElement.SetAttribute("Include", sourcePath);
        itemGroupNode.AppendChild(contentElement);

        AddNode("CopyToOutputDirectory", "Always", contentElement);

        // Destination in build directory
        string relativePath = Path.GetRelativePath(contentDirectory, sourcePath);
        AddNode("Link", $"Content\\{relativePath}", contentElement);
    }
}
```

Kód 44: část metody *AddContent* (přidání souborů)

Výsledek vypadá nějak takto:



Obrázek 15: struktura *content files* v projektu

Do struktury projektu přibyly soubory, které se v něm fyzicky nenacházejí. Projekt má v sobě pouze cesty k těmto souborům a až při publikování je zahrne do výsledného .exe souboru. Samotné vyvolání procesu publikace je uvedeno zde:

```
private static string CreateExecutable(string projectDirectory,
    string outputDirectory)
{
    string command = $"dotnet publish -r win-x64 --output {outputDirectory}";

    Process process = new Process
    {
        StartInfo = new ProcessStartInfo
        {
            FileName = "powershell.exe",
            UseShellExecute = false,
            CreateNoWindow = true,
            WorkingDirectory = projectDirectory,
            RedirectStandardOutput = true,
            RedirectStandardError = true,
            Arguments = $"-ExecutionPolicy Bypass -Command \"{command}\""
        }
    };

    Console.WriteLine("Creating executable...");
    process.Start();

    string output = process.StandardOutput.ReadToEnd();
    string error = process.StandardError.ReadToEnd();

    process.WaitForExit();

    return output + "\n" + error;
}
```

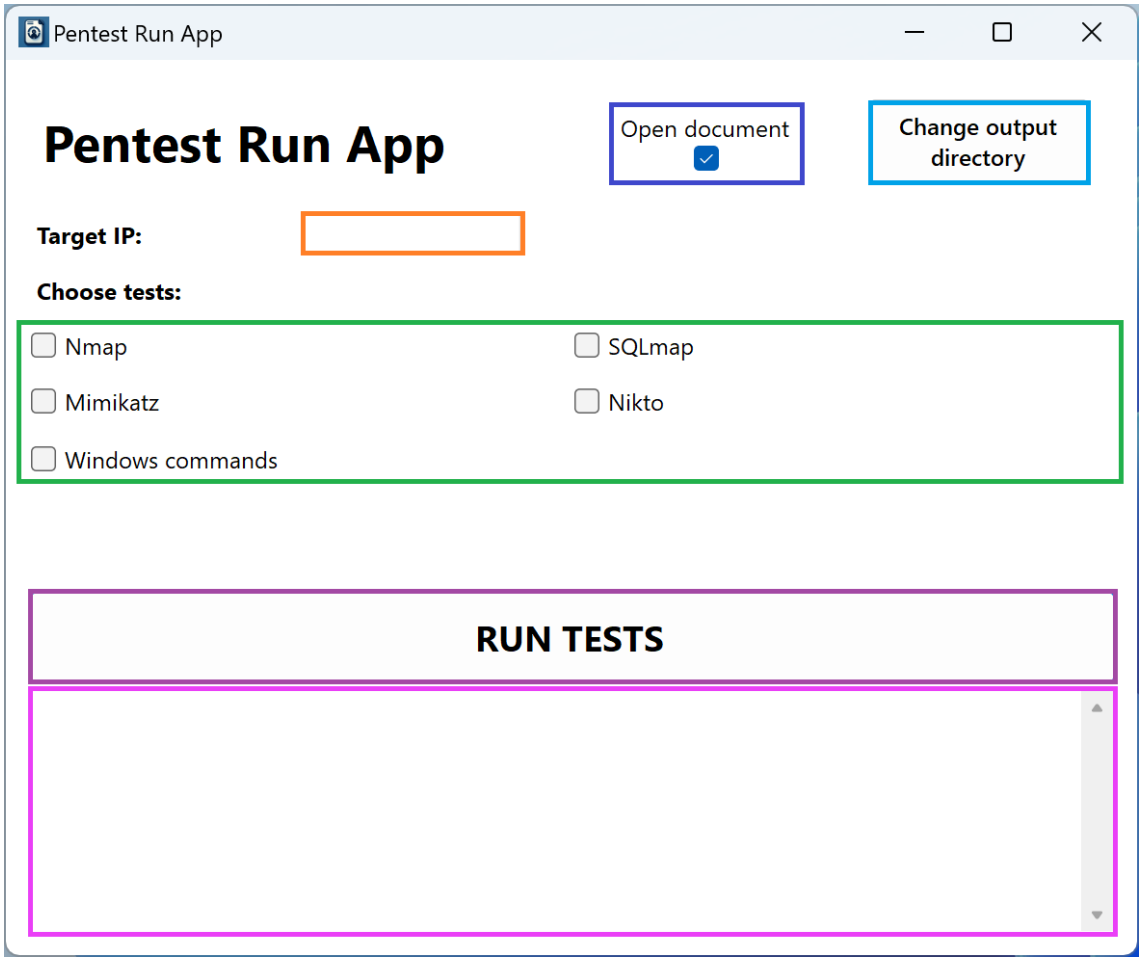
Kód 45: metoda *CreateExecutable*

10 APLIKACE PRO SPOUŠTENÍ TESTŮ

Uživatelské prostředí programu bylo vytvářeno pomocí frameworku Winforms.

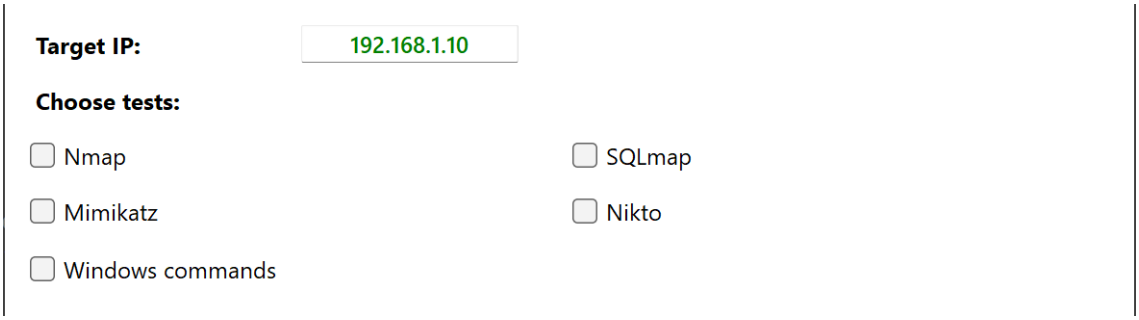
Jeho základními prvky jsou:

1. Pole pro zadávání IP adresy cíle (výchozí 127.0.0.1)
2. Přepínače pro výběr testů
3. Přepínač otevření dokumentu po dokončení testů
4. Tlačítko pro změnu výstupního adresáře
5. Tlačítko pro spuštění testů
6. Konzole



Obrázek 16: rozložení uživatelského prostředí (GUI)

Součástí pole pro zadávání adresy je validace vstupu. Pokud je adresa zelená, pak je v pořádku a předá se příslušné proměnné.



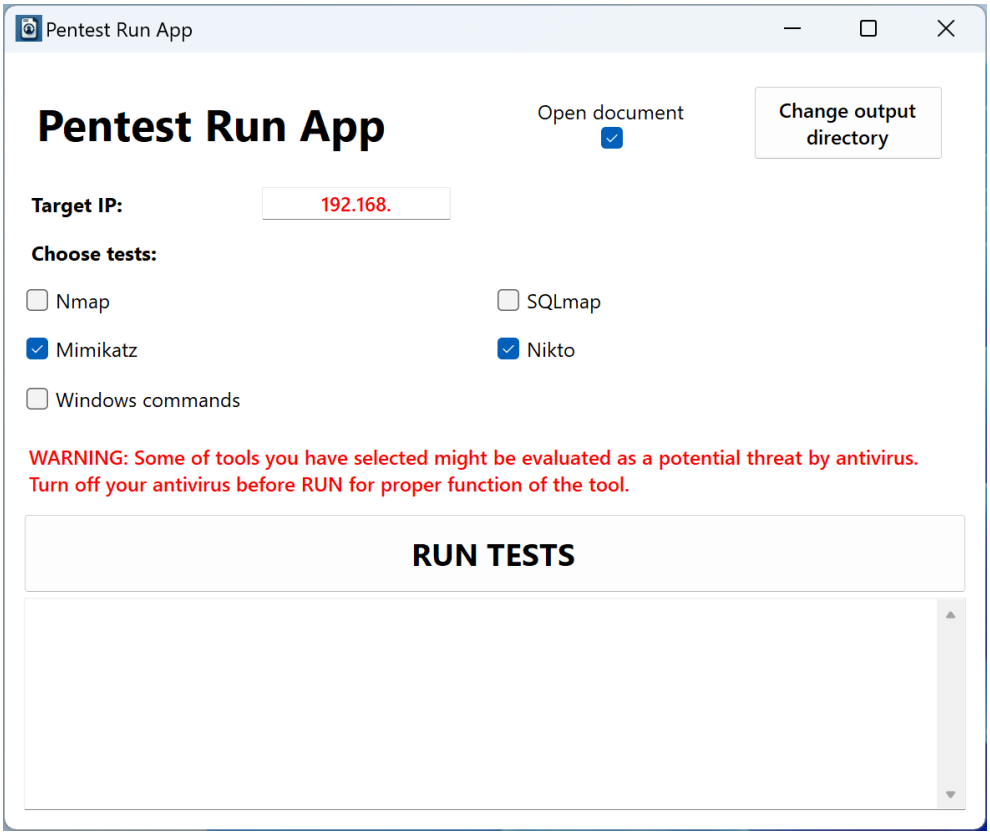
Obrázek 17: validace IP adresy

Validace proběhne v případě změny vstupního textu. Využívá se zde regulárního výrazu pro porovnávání. Pokud je vstup shodný s definovaným vzorem (*pattern*), pak se text předá proměnné *target*. Pokud však není shodný, nastaví se výchozí hodnota 127.0.0.1 (localhost).

```
private void target_TextChanged(object sender, EventArgs e)
{
    string text = textBox_target.Text.Trim();

    string pattern = @"^(?:\d{1,3}\.){3}\d{1,3}$";
    Regex regex = new Regex(pattern);

    if (regex.IsMatch(text))
    {
        textBox_target.ForeColor = Color.Green;
        target = textBox_target.Text;
    }
    else
    {
        textBox_target.ForeColor = Color.Red;
        target = "127.0.0.1";
    }
}
```

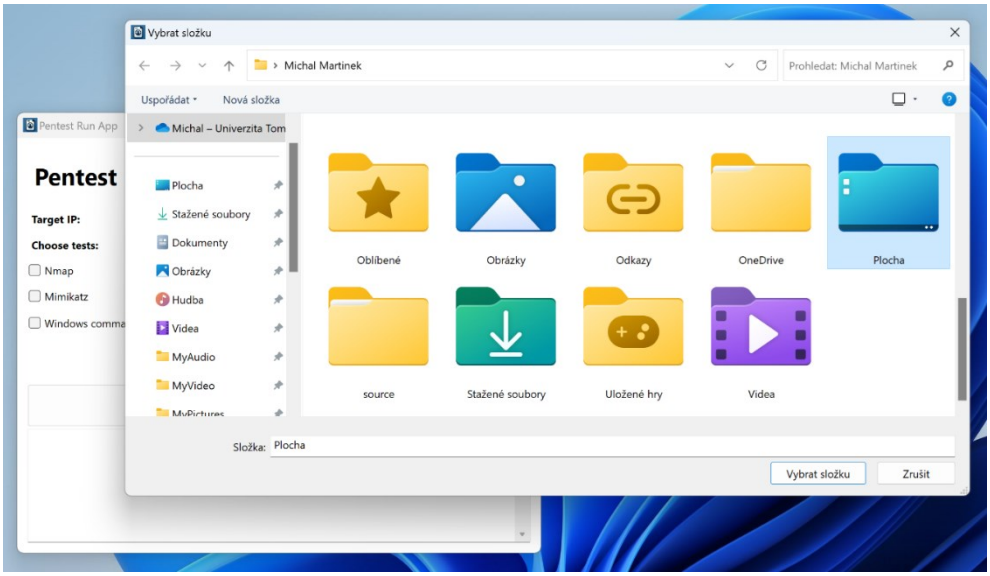
Kód 46: validace IP adresy

Jak je již uvedeno v Kapitole 6, nástroje *Mimikatz* a *Nikto* mohou být antivirem vyhodnoceny jako škodlivý software. Z toho důvodu je při jejich výběru uživatel informován, že pokud ochranu nevypne, testy nemusí probíhat korektně.



Obrázek 18: výpis varovné hlášky

Při stisku tlačítka pro změnu výstupního adresáře se otevře dialog s aktuálně nastavenou cestou. Uživatel ji může změnit svým výběrem jiného adresáře, nebo dialog zavřít a nechat tak cestu výchozí (adresář *Desktop*, česky *Plocha*).



Obrázek 19: změna pracovní složky

```
private void changeWorkDirectory_Click(object sender, EventArgs e)
{
    using (FolderBrowserDialog dialog = new FolderBrowserDialog())
    {
        dialog.SelectedPath = workDirectory;

        if (dialog.ShowDialog() == DialogResult.OK)
        {
            workDirectory = dialog.SelectedPath;
        }
    }
}
```

Kód 47: změna pracovní složky

Po stisku tlačítka Run je zavolána metoda *HandleRun* pro všechny testy. Teprve uvnitř této metody se určí, zda byl test vybrán či ne. Pokud byl, pak je sestavena cesta k příslušnému skriptu a tento skript je následně spuštěn. Aby nedocházelo k „zamrznutí“ uživatelské aplikace, je metoda definována jako asynchronní.

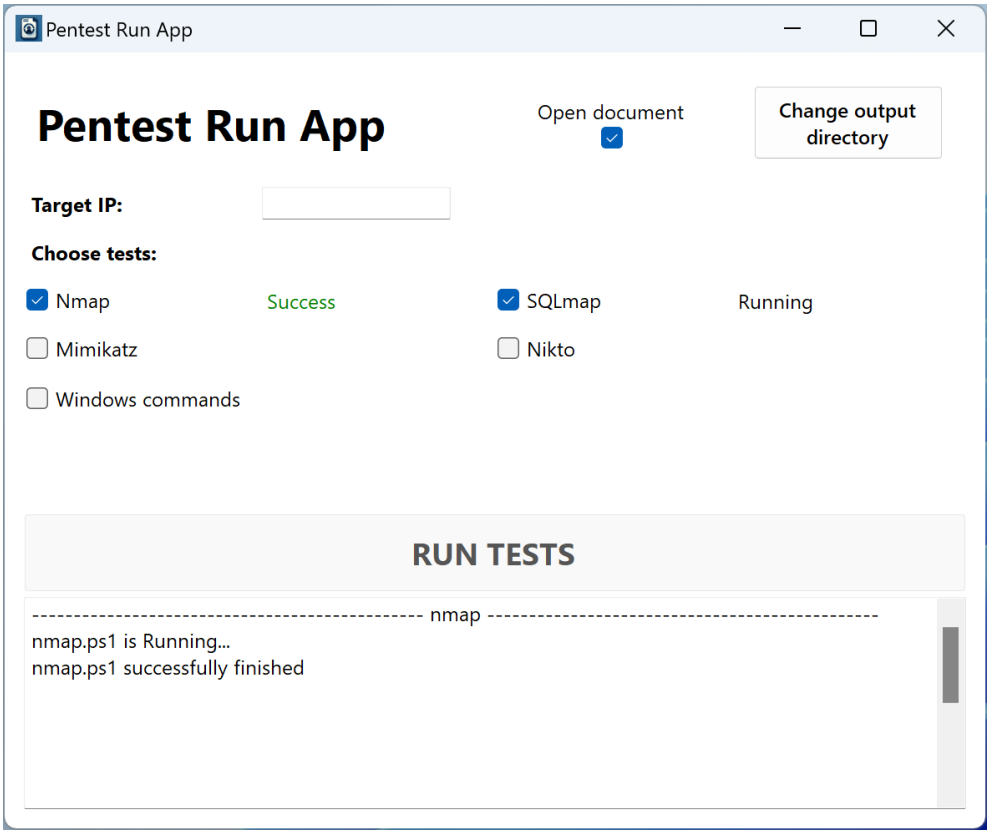
```
private async Task HandleRun(string scriptName, CheckBox checkBox, Label label,
    TextBox consoleBox)
{
    runSuccess = false;

    if (checkBox.Checked)
    {
        string scriptPath = Path.Combine(Program.baseDirectory, "Content",
        "Scripts", $"{scriptName}.ps1");

        if (File.Exists(scriptPath))
        {
            label.Text = "Running";
            label.ForeColor = Color.Black;
            consoleBox.Text += GetSeparator(scriptName) + Environment.NewLine;
            consoleBox.Text += $"{scriptName}.ps1 is Running..."
            + Environment.NewLine;

            string result = await Task.Run(() => RunScript(scriptPath));
            consoleBox.Text += result;
        }
        else
        {
            consoleBox.Text += $"Script {scriptName} was not found.";
        }

        label.Text = runSuccess ? "Success" : "Failed";
        label.ForeColor = runSuccess ? Color.Green : Color.Red;
    }
}
```

Kód 48: metoda *HandleRun*

Metoda odpovědná za spuštění testových skriptů se nazývá *RunScript*. Kód této metody je velice podobný jako *CreateExecutable* u publikování (Kapitola 9), proto není nutno, aby zde byl celý. Jediný rozdíl spočívá v argumentech pro spuštění procesu PowerShell, které vypadají následovně:

```
Arguments = $"-File \"{path}\" \"{workDirectory}\" \"{target}\"",
```

Kód 49: argumenty *RunScript*



Obrázek 20: stisk tlačítka Run

Tlačítko *Run* se po dobu běhu testu zablokuje. Dále se všechny stavové popisy (*labels*) vyresetují a zkontroluje se, zda je vybrán nějaký test. Pokud ano, tak se nejprve „uklidí“ složka pro výstup (vymažou se soubory z předchozího běhu) a následně se volá *HandleRun* pro každý test. Po doběhnutí všech testů už zbývá pouze zavolat metodu *Create*, která vytvoří z výstupů jeden PDF report.

```
private async void run_Click(object sender, EventArgs e)
{
    buttonRun.Enabled = false;
    ResetLabels();

    bool anyChecked = IsAnythingChecked(tableLayout);
    if (anyChecked)
    {
        CleanupOutputDirectory();

        await HandleRun("nmap", checkBox_nmap, label_nmap, consoleBox);
        await HandleRun("sqlmap", checkBox_sqlmap, label_sqlmap, consoleBox);
        await HandleRun("mimikatz", checkBox_mimikatz, label_mimikatz,
        consoleBox);
        await HandleRun("nikto", checkBox_nikto, label_nikto, consoleBox);
        await HandleRun("wincmd", checkBox_wincmd, label_wincmd, consoleBox);

        PDFCreator.Create(workDirectory, postOpen.Checked);
    }
    else
    {
        consoleBox.Text += "Nothing is checked. Please select some tool."
        + Environment.NewLine;
    }

    buttonRun.Enabled = true;
}
```

Kód 50: stisk tlačítka Run

10.1 Konzolová verze

Výsledná aplikace byla vyhotovena ve dvou verzích. Kromě verze s prostředím GUI, byla také vytvořena verze pro spuštění prostřednictvím konzole, čímž se rozšířila použitelnost.

Konzolová verze má 4 možné druhy argumentů:

Argument	Popis
`-h, -help`	Zobrazit možnosti (nápovědu)
`-tools T1 T2 ...`	Vybrat nástroje které budou spuštěny
`-target X.X.X.X`	Změnit IP adresu cíle
`-output DIRECTORY`	Změnit adresář pro uložení výstupu

Tabulka 2: argumenty konzolové verze

Pokud nejsou zadány žádné argumenty, budou spuštěny všechny testy s cílovou IP 127.0.0.1 (localhost) a výstup se uloží to složky *Desktop* (*Plocha*)

Zpracování argumentů řeší metoda *HandleArgs*. Metoda je obsáhla a dělá totéž co je již vysvětleno v popisu verze s GUI (Kapitola 10), takže zde bude jen naznačena struktura a tělo podmínek bude nahrazeno třemi tečkami.

```
private static bool HandleArgs(string[] args)
{
    if (args[0] == "-h" || args[0] == "-help")
    {
        DisplayHelp();
        return false;
    }
    string currentKey = string.Empty;
    foreach (var arg in args)
    {
        if (arg.StartsWith("-")) { currentKey = arg; }
        else
        {
            if (currentKey == "-tools") { ... }
            else if (currentKey == "-target") { ... }
            else if (currentKey == "-output") { ... }
            else
            {
                Console.WriteLine("Wrong argument.");
                return false;
            }
        }
    }
    WriteRunParams();
    return true;
}
```

Kód 51: metoda *HandleArgs*

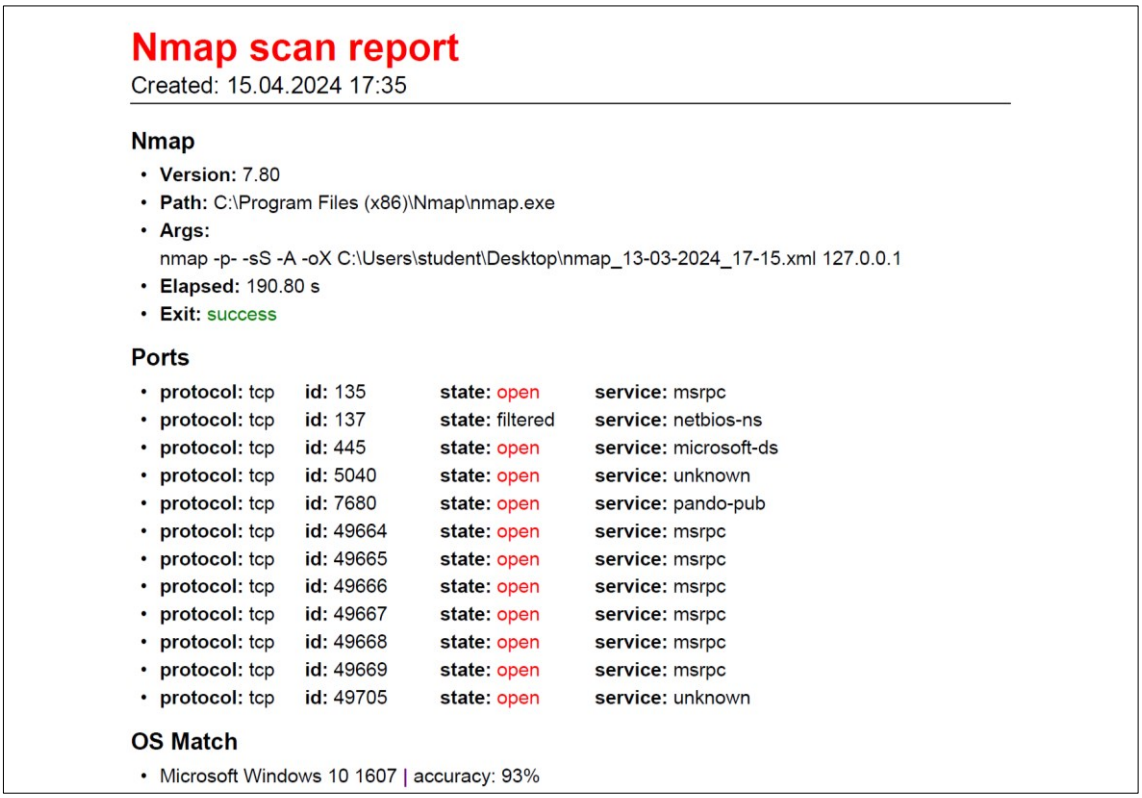
Oproti verzi s GUI jsou zde další 2 nové metody:

Metoda *WriteRunParams* vypisuje upravenou konfiguraci po provedení změn a metoda *DislayHelp*, jak napovídá její název, vypisuje nápovědu a dostupné nástroje.

```
private static void DisplayHelp()
{
    Console.WriteLine("\nOptions:");
    Console.WriteLine("  -h, -help           Display options");
    Console.WriteLine("  -tools T1 T2 ...    Select tools that will be used");
    Console.WriteLine("  -target X.X.X.X     Select target ip address");
    Console.WriteLine("  -output DIRECTORY   Select output directory");

    Console.WriteLine("\nAvailable tools:");
    foreach (string tool in availableTools)
    {
        Console.WriteLine($"  {tool}");
    }
}
```

Kód 52: Metoda *DisplayHelp*

10.2 Ukázky reportů



Nmap scan report

Created: 15.04.2024 17:35

Nmap

- **Version:** 7.80
- **Path:** C:\Program Files (x86)\Nmap\nmap.exe
- **Args:**
  nmap -p- -sS -A -oX C:\Users\student\Desktop\nmap_13-03-2024_17-15.xml 127.0.0.1
- **Elapsed:** 190.80 s
- **Exit:** success

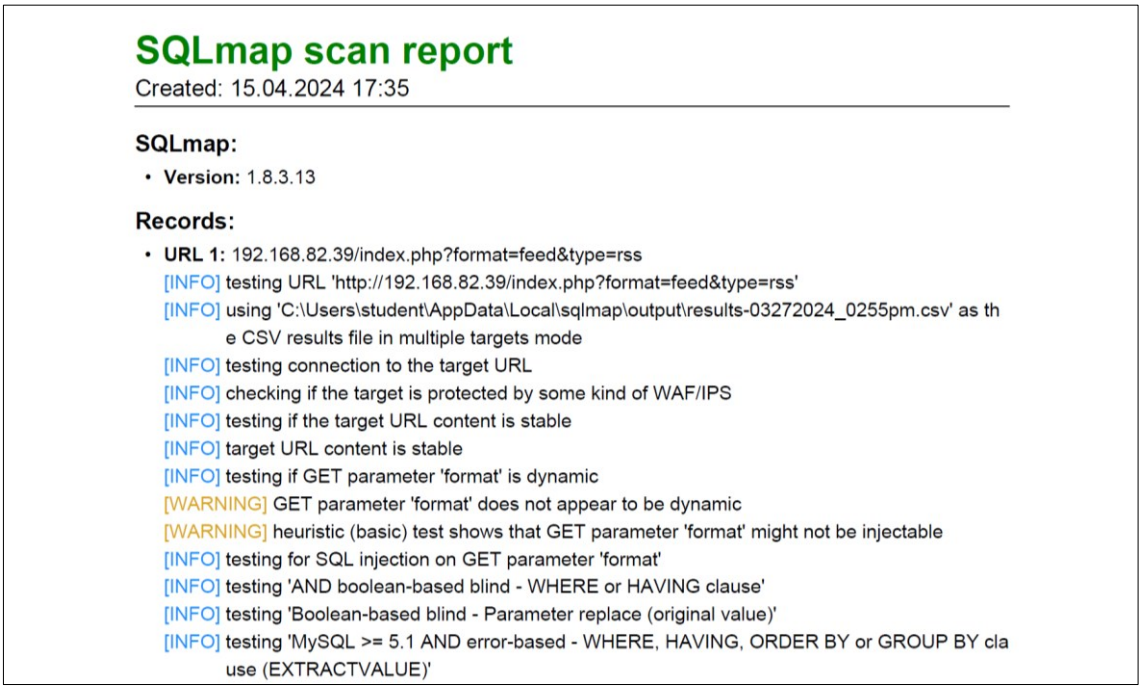
Ports

- **protocol:** tcp **id:** 135 **state:** open **service:** msrpc
- **protocol:** tcp **id:** 137 **state:** filtered **service:** netbios-ns
- **protocol:** tcp **id:** 445 **state:** open **service:** microsoft-ds
- **protocol:** tcp **id:** 5040 **state:** open **service:** unknown
- **protocol:** tcp **id:** 7680 **state:** open **service:** pando-pub
- **protocol:** tcp **id:** 49664 **state:** open **service:** msrpc
- **protocol:** tcp **id:** 49665 **state:** open **service:** msrpc
- **protocol:** tcp **id:** 49666 **state:** open **service:** msrpc
- **protocol:** tcp **id:** 49667 **state:** open **service:** msrpc
- **protocol:** tcp **id:** 49668 **state:** open **service:** msrpc
- **protocol:** tcp **id:** 49669 **state:** open **service:** msrpc
- **protocol:** tcp **id:** 49705 **state:** open **service:** unknown

OS Match

- Microsoft Windows 10 1607 | accuracy: 93%

Obrázek 21: ukázka reportu z Nmap



SQLmap scan report

Created: 15.04.2024 17:35

SQLmap:

- **Version:** 1.8.3.13

Records:

- **URL 1:** 192.168.82.39/index.php?format=feed&type=rss
  [INFO] testing URL 'http://192.168.82.39/index.php?format=feed&type=rss'
  [INFO] using 'C:\Users\student\AppData\Local\sqlmap\output\results-03272024_0255pm.csv' as th
  e CSV results file in multiple targets mode
  [INFO] testing connection to the target URL
  [INFO] checking if the target is protected by some kind of WAF/IPS
  [INFO] testing if the target URL content is stable
  [INFO] target URL content is stable
  [INFO] testing if GET parameter 'format' is dynamic
  [WARNING] GET parameter 'format' does not appear to be dynamic
  [WARNING] heuristic (basic) test shows that GET parameter 'format' might not be injectable
  [INFO] testing for SQL injection on GET parameter 'format'
  [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
  [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
  [INFO] testing 'MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY cla
  use (EXTRACTVALUE)'

Obrázek 22: ukázka reportu z sqlmap

10.3 Ikony aplikací

Pro tuto aplikaci byla vytvořena dvojice ikon, které se zobrazují u jejich spustitelných souborů a v případě verze s GUI i v prostředí aplikace.

Obrázek 23: ikona aplikace s GUI

Obrázek 24: ikona konzolové aplikace

ZÁVĚR

Aplikace pro automatizované spouštění penetračních testů představují důležitý krok směrem k zajištění bezpečnosti informačních systémů. Tato práce si kladla za cíl poskytnout uživatelům nástroj, který umožňuje provádět penetrační testy snadno a efektivně.

S ohledem na různé typy zařízení a preference uživatelů přináší aplikace možnost volby mezi grafickým rozhraním a spuštěním z konzole, čímž se zvyšuje její použitelnost v praxi.

Podařilo se tedy vytvořit nástroj pro administrátory, který lze použít na serverech i běžných uživatelských stanicích.

Výsledný report je strukturovaný a přehledný. Důležité informace jsou barevně zvýrazněny. Celkově hodnotím dokument jako dobře srozumitelný a po estetické stránce čistý a moderní.

Věřím, že tato aplikace přispěje ke zvýšení úrovně kybernetické bezpečnosti a pomůže svým uživatelům identifikovat potenciální zranitelnosti a slabiny v jejich systémech. Dále se domnívám, že dalším vývojem a zdokonalováním může tato aplikace posílit svou pozici jako efektivní nástroj v boji proti nadcházejícím kybernetickým hrozbám.

SEZNAM POUŽITÉ LITERATURY

[1] SELECKÝ, Matúš. *Penetrační testy a exploitace*. Brno: Computer Press, 2012. ISBN 978-80-251-3752-9.

[2] KIM, Peter. *Hacking: praktický průvodce penetračním testováním*. Encyklopedie Zoner Press. Brno: Zoner Press, 2015. ISBN ISBN978-80-741-3313-8.

[3] HERZOG, Pete. *OSSTMM 3 – The Open Source Security Testing Methodology Manual*. online. ISECOM, 2010. Dostupné z: https://www.isecom.org/OSSTMM.3.pdf. [cit. 2024-04-19].

[4] WEIDMAN, Georgia. *Penetration testing: a hands-on introduction to hacking*. San Francisco, CA: No Starch Press, 2014. ISBN 978-1-59327-564-8.

[5] White-Hat-vs-Black-Hat. online. In: *Mindsmapped*. Dostupné z: https://www.mindsmapped.com/wp-content/uploads/2019/01/White-Hat-vs-Black-Hat.png. [cit. 2024-04-19].

[6] LYON, Gordon. *Nmap: the Network Mapper - Free Security Scanner*. online. Dostupné z: https://nmap.org/. [cit. 2024-04-19].

[7] Nmap-logo-256x256. online. In: *Nmap: the Network Mapper - Free Security Scanner*. Dostupné z: https://nmap.org/images/nmap-logo-256x256.png. [cit. 2024-04-19].

[8] *Sqlmap*. online. Dostupné z: https://sqlmap.org/. [cit. 2024-04-19].

[9] *Mimikatz Wiki*. online. In: Github. Dostupné z: https://github.com/gentilkiwi/mimikatz/wiki. [cit. 2024-04-19].

[10] Mimikatz-logo. online. In: *Kali*. Dostupné z: https://www.kali.org/tools/mimikatz/images/mimikatz-logo.svg. [cit. 2024-04-19].

[11] *Nikto Wiki*. online. In: Github. Dostupné z: https://github.com/sullo/nikto/wiki. [cit. 2024-04-19].

[12] Word-image. online. In: *Secuneus*. Dostupné z: https://www.secuneus.com/wp-content/uploads/2021/09/word-image.jpeg. [cit. 2024-04-19].

[13] *JSON*. online. Dostupné z: https://www.json.org/json-en.html. [cit. 2024-04-20].

[14] Json160. online. In: *JSON*. Dostupné z: https://www.json.org/img/json160.gif. [cit. 2024-04-19].

[15] *JSON Example*. online. In: JSON. Dostupné z: https://json.org/example.html. [cit. 2024-04-19].

[16] *Xml_whatis*. online. In: W3schools. Dostupné z: https://www.w3schools.com/xml/xml_whatis.asp. [cit. 2024-04-20].

[17] PECINOVSKÝ, Rudolf. *Začínáme programovat v jazyku Python*. Začínáme s.. Praha: Grada Publishing, 2020. ISBN ISBN978-80-271-1237-1.

[18] Python-logo-notext. online. In: *Wikipedia*. Dostupné z: https://en.wikipedia.org/wiki/Python_%28programming_language%29#/media/File:Python-logo-notext.svg. [cit. 2024-04-19].

[19] *What is Java?*. online. In: Opensource. Dostupné z: https://opensource.com/resources/java. [cit. 2024-04-19].

[20] Java-logo. online. In: *Kinsta*. Dostupné z: https://kinsta.com/wp-content/uploads/2023/01/Java-logo.png. [cit. 2024-04-19].

[21] BORY, Pavel. *C# bez předchozích znalostí*. 2. vydání. V Brně: Computer Press, 2022. ISBN ISBN978-80-251-5061-0.

[22] Csharp_Logo. online. In: *Wikimedia*. Dostupné z: https://upload.wikimedia.org/wikipedia/commons/4/4f/Csharp_Logo.png. [cit. 2024-04-19].

[23] *Co je PowerShell?*. online. In: Microsoft. Dostupné z: https://learn.microsoft.com/cs-cz/powershell/scripting/overview. [cit. 2024-04-19].

[24] Powershell-2. online. In: *Ldlnet*. Dostupné z: https://itblog.ldlnet.net/wp-content/uploads/2019/01/powershell-2.png. [cit. 2024-04-19].

[25] *Anti-virus Interrupts Installation*. online. In: Rapid7. Dostupné z: https://docs.rapid7.com/metasploit/anti-virus-interrupts-installation/. [cit. 2024-04-24].

[26] *PDFsharp*. online. Dostupné z: https://www.pdfsharp.net/?AspxAutoDetectCookieSupport=1. [cit. 2024-04-20].

[27] PDFsharp-80x80. online. In: . Dostupné z: https://www.pdfsharp.net/Themes/PDFsharp/images/PDFsharp-80x80.png. [cit. 2024-04-20].

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

ASCII	American Standard Code for Information Interchange
CMD	Command
CSS	Cascading Style Sheets
GUI	Graphical User Interface
HTML	Hypertext Markup Language
IP	Internet Protocol
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
LINQ	Language Integrated Query
MITM	Man In The Middle
MS	Microsoft
OOP	Object Oriented Programming
OS	Operating System
OSSTMM	Open Source Security Testing Methodology Manual
OWASP	Open Web Application Security Project
PS	PowerShell
SQL	Structured Query Language
TCP	Transmission Control Protocol
TTL	Time To Live
TXT	Text
UDP	User Datagram Protocol
WPF	Windows Presentation Foundation
XML	Extensible Markup Language
XSS	Cross-Site Scripting

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA PI: SEZNAM PŘILOŽENÝCH SOUBORŮ

\PentestPDF-master.zip	repozitář obsahující obě verze aplikace, odkaz: https://github.com/M4rt3l/PentestPDF
\Content.zip	obsahuje obrázky (ikony aplikací a logo do hlavičky reportu), skripty pro penetračními testy, pomocné programy a instalátory
\report_04-05-2024_21-09.pdf	výsledný report