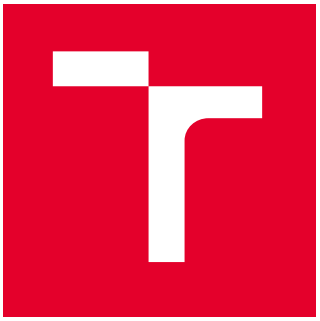


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

NÁSTROJ PRO PENETRAČNÍ TESTOVÁNÍ APLIKAČNÍHO SERVERU

TOOLS FOR APPLICATION SERVER PENETRATION TESTING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Tomáš Vašíček**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Zdeněk Martinásek, Ph.D.**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Bc. Tomáš Vašíček ***ID:*** 221581

Ročník: 2 ***Akademický rok:*** 2023/24

NÁZEV TÉMATU:

Nástroj pro penetrační testování aplikačního serveru

POKYNY PRO VYPRACOVÁNÍ:

Hlavním cílem diplomové práce je návrh a implementace komplexního automatizovaného nástroje pro bezpečnostní penetrační testování protokolů aplikační vrstvy. Konkrétně se jedná o testovací případy aplikačních protokolů FTP, SSH, IMAP, POP3 a SMTP. Výsledný nástroj bude modulární a bude se skládat z individuálních modulů pro testování jednotlivých zranitelností. V teoretické části nastudujte současný stav problematiky, navrhněte diagram obsahující postup penetračního testu jednotlivých protokolů. Na základě analýzy navrhněte a implementujte nástroj v jazyku Python, který otestujte pro různé instalace aplikačního serveru.

DOPORUČENÁ LITERATURA:

[1] KONDO, Tabu S.; MSELLE, Leonard J. Penetration testing with banner grabbers and packet sniffers. Journal of Emerging Trends in computing and information sciences, 2014, 5.4: 321-327.

[2] VISOOTTIVISETH, Vasaka, et al. PENTOS: Penetration testing tool for Internet of Thing devices. In: TENCON 2017-2017 IEEE Region 10 Conference. IEEE, 2017. p. 2279-2284.

Termín zadání: 5.2.2024 ***Termín odevzdání:*** 21.5.2024

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

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

ABSTRAKT

Tato diplomová práce se zabývá problematikou penetračního testování aplikačních protokolů. V rámci práce jsou představeny aplikační protokoly FTP, SSH, SMTP, POP3 a IMAP a jsou prozkoumány jejich možné zranitelnosti. Informace o zranitelnostech jsou získávány z veřejně dostupných sbírek typu HackTricks a The Hacker Recipes, ale rovněž vlastním studiem RFC dokumentů jednotlivých protokolů. Na základě nalezených zranitelností jsou sestrojeny kontrolní seznamy sloužící pro návodné provedení penetračního testera procesem testování daného protokolu. Hlavním přínosem práce je vývoj modulárního automatizovaného nástroje ptapptest a dalšího pomocného nástroje ptntlmauth, které slouží pro penetrační testování zmíněných aplikačních protokolů. Závěrem práce je provedeno testování nástroje ptapptest na reálných aplikačních serverech, které byly nalezeny pomocí vyhledávače Shodan.

KLÍČOVÁ SLOVA

FTP, IMAP, POP3, Penterep, Python, SMTP, SSH, aplikační protokol, penetrační testování, ptapptest, ptntlmauth

ABSTRACT

This thesis explores the field of penetration testing of application protocols. The thesis introduces the application protocols FTP, SSH, SMTP, POP3 and IMAP and explores their possible vulnerabilities. Information about vulnerabilities is obtained from publicly available collections such as HackTricks and The Hacker Recipes, but also by studying the RFC documents of each protocol. Based on the vulnerabilities found, penetration testing checklists are constructed to provide guidance through the process of testing a given protocol. The main contribution of the work is the development of a modular automated tool ptapptest and another auxiliary tool ptntlmauth, which are used for penetration testing of the mentioned application protocols. Finally, the thesis concludes by testing the ptapptest tool on application servers discovered using the Shodan search engine.

KEYWORDS

FTP, IMAP, POP3, Penterep, Python, SMTP, SSH, application protocol, penetration testing, ptapptest, ptntlmauth

Vysázeno pomocí balíčku `thesis` verze 4.09; `https://latex.fekt.vut.cz/`

VAŠÍČEK, Tomáš. *Nástroj pro penetrační testování aplikačního serveru*. Diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2024. Vedoucí práce: doc. Ing. Zdeněk Martinásek, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Bc. Tomáš Vašíček
VUT ID autora:	221581
Typ práce:	Diplomová práce
Akademický rok:	2023/24
Téma závěrečné práce:	Nástroj pro penetrační testování aplikačního serveru

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

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

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Děkuji vedoucímu diplomové práce panu doc. Ing. Zdeňkovi Martináskovi, Ph.D. za odborné vedení, konzultace a podnětné návrhy k práci. Rovněž děkuji společnosti HACKER Consulting s.r.o. za spolupráci na vývoji jednoho z modulů nástroje ptapptest.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

Kybernetická bezpečnost a kybernetické útoky jsou aktuálním a důležitým tématem. Podle Zprávy o stavu kybernetické bezpečnosti ČR za rok 2022[1] došlo v roce 2022 k téměř dvojnásobnému nárůstu kyberkriminálních aktivit a k dvojnásobnému nárůstu počtu kybernetických incidentů v rámci kritické informační infrastruktury.[1] Je zřejmé, že trend kybernetických útoků je na vzestupu a s ním je zapotřebí zvyšovat i úroveň kybernetické bezpečnosti. Toho lze docílit více cestami, jednou z nichž je proaktivní přístup k ochraně informačních systémů formou penetračního testování. Penetrační testování simuluje reálné útoky a hrozby a pomáhá tak identifikovat slabá místa, která by mohla být zneužita skutečnými útočníky. Po identifikaci těchto slabin dochází k jejich nápravě, a tím ke zvýšení celkové kybernetické bezpečnosti testovaných systémů. Penetrační testování je tedy efektivním způsobem pro zesílení odolnosti vůči kybernetickým útokům, avšak pro jeho kvalitní a důsledné provedení je zapotřebí zkušených a znalých odborníků. Alternativou k manuálnímu penetračnímu testování je testování automatizované. Tento typ testování sice neklade tak vysoké nároky na jeho vykonavatele, ale má také svá negativa. Automatizované penetrační testování zpravidla není schopné odhalit složitější slabiny, které by bylo možné odhalit odborným manuálním přístupem.[2]

V současné době na trh penetračního testování ovšem vstupuje nový nástroj, a to platforma Penterep[2]. Tato platforma nabízí provedení penetračního testera procesem poloautomatického testování. Během testování tester následuje předem připravené kontrolní seznamy obsahující veškeré důležité kroky, které by měly být vykonány. Některé z těchto kroků rovněž platforma automatizuje pomocí automatických testovacích modulů. Platforma touto cestou zpřístupňuje penetrační testování a snižuje míru znalostí a zkušeností, které jsou od testerů vyžadovány. Platforma je ovšem stále ve fázi vývoje a pro její plošné využití je nutné ji dále rozšířit o více schopností a možností.

[1] `https://www.nukib.cz/download/publikace/zpravy_o_stavu/Zprava_o_stavu_kybernetic ke_bezpecnosti_CR_za_rok_2022.pdf`

[2] Platforma Penterep: `https://www.penterep.com/`

Cíle práce

Hlavním cílem této diplomové práce je navrhnout a implementovat rozšíření platformy Penterep, a to v podobě modulárního automatizovaného nástroje pro penetrační testování protokolů aplikační vrstvy FTP, SSH, SMTP, POP3 a IMAP. Dílčí cíle vedoucí k sestrojení automatizovaného nástroje jsou následující:

- analýza aplikačních protokolů FTP, SSH, SMTP, POP3 a IMAP a jejich zranitelností,
- návrh kontrolních seznamů (diagramů) pro analyzované aplikační protokoly,
- návrh modulárního automatizovaného nástroje pro penetrační testování analyzovaných protokolů,
- implementace automatizovaného nástroje pro analyzované protokoly v jazyce Python,
- otestování výsledného nástroje pro různé instalace aplikačního serveru.

1 Představení základních pojmů

V této kapitole jsou vysvětleny pojmy *penetrační testování* a *aplikační vrstva*. Rovněž je zde představena platforma *Penterep*, její kontrolní seznamy a automatické nástroje.

1.1 Penetrační testování

Pojem *penetrační testování* označuje proces testování, který se zaměřuje na odhalení zranitelností ohrožujících bezpečnost testovaného cíle. Penetrační testování má více druhů, v závislosti na testovaném cíli, kterým může být například bezdrátová síť Wi-Fi, klientská aplikace (webová, desktopová apod.), hardwarové zařízení, ale i fyzické zabezpečení objektů (zámky, pohybové detektory apod.). Tato diplomová práce se věnuje pouze penetračnímu testování síťové infrastruktury a síťových služeb. [3]

1.1.1 Dělení penetračního testování

Dle znalostí o testovaném systému

V závislosti na množství dostupných informací (o testovaném cíli lze penetrační testování rozdělit do 3 kategorii. Konkrétně se jedná o tzv. *black-box*, *white-box* a *grey-box* penetrační testování.

Black-box penetrační testování simuluje proces, kterým by se skutečný útočník snažil prolomit testovaný cíl, jelikož během tohoto typu penetračního testování nejsou dostupné žádné interní informace o testovaném cíli. Penetrační testování probíhá pouze na základě pozorování chování testovaného cíle.

White-box penetrační testování je opakem typu black-box, jelikož během testování jsou dostupné veškeré informace o testovaném cíli, jako například zdrojové kódy aplikací či detailní informace o síťových zařízeních a jejich službách. Testování může probíhat formou testování síťových služeb se znalostí jejich interního fungování, nebo i formou pouhé bezpečnostní analýzy zdrojového kódu testované aplikace (tzv. source code review).

Gray-box penetrační testování se nachází na pomezí typů black-box a white-box. Během gray-box testování je dostupné určité množství informací o testovaném cíli, jako například seznam koncových bodů webové aplikace či znalost topologie síťové infrastruktury. [2, 4, 5]

Dle míry automatizace testování

Penetrační testování je rovněž možné rozdělit na manuální a automatizované testování. Oba tyto typy mají své výhody a nevýhody a během penetračního testování

je často vhodné využít kombinaci obou typů.

Manuální penetrační testování je prováděno manuálně penetračním testerem a vyžaduje vysokou míru znalostí a zkušeností. Tester musí znát komplexní techniky a být schopen ovládat potřebné nástroje. Výhodou tohoto přístupu je možnost hlubšího a kreativnějšího testování, což může vést k nalezení složitějších zranitelností, které by mohly být automatizovanými nástroji přehlédnuty. Nevýhodami manuálního přístupu jsou zvýšené časové i cenové nároky na provedení testu.

Při automatizovaném penetračním testování jsou využívány automatické softwarové nástroje pro skenování testovaného cíle a identifikaci jeho zranitelností. Výhodou automatizovaného testování jsou vyšší rychlost, nižší cenové a časové nároky a škálovatelnost pro testování velkého množství cílů. Nevýhodou toto přístupu je ohlašování falešné positivních nálezů a nemožnost nalezení některých zranitelností. [6]

1.1.2 Fáze penetračního testování

Proces penetračního testování se standardně dělí na 5 fází, z nichž první je sběr informací. Jedná se o úvodní fázi testování, jejíž cílem je poskytnout penetračním testerů co nejvíce relevantních informací o testovaném prostředí. Informace jsou sbírány ze všech dostupných zdrojů, častokrát s využitím *Open Source Intelligence* (OSINT)[1] technik. Informace získané během této fáze mohou zahrnovat údaje o cílové doméně či IP adresách, jména či e-mailové adresy uživatelů, operační systémy zařízení nebo softwary vyskytující se v testovaném prostředí a jejich verze.

Druhou fází je skenování testovaného cíle. Během skenování jsou využívány různé metody a nástroje k odhalení a identifikaci síťových zařízení a služeb, které tato zařízení provozují. Detailní identifikace a prozkoumání nalezených síťových služeb jsou důležitými navazujícími kroky, jelikož mohou poskytnout důležité informace nutné v dalších fázích penetračního testování.

Po identifikaci síťových služeb následuje fáze identifikace zranitelností. Cílem této fáze je identifikování zranitelností nalezených síťových služeb s využitím předešle získaných informací. Pokud síťové služby provozují veřejně známý software, je možné zjistit, zda pro jeho verzi existují veřejně známé zranitelnosti. Tyto zranitelnosti je možné vyhledávat v databázích veřejně známých zranitelností, jako jsou například databáze *Common Vulnerabilities and Exposures* (CVE)[2] či *National Vulnerability Database* (NVD)[3].

[1] OSINT: `https://www.sans.org/blog/what-is-open-source-intelligence/`

[2] CVE databáze: `https://www.cve.org/`

[3] NVD databáze: `https://nvd.nist.gov/`

V případě úspěšné identifikace zranitelností následuje fáze zneužití zranitelností, neboli „exploitace“. Během této fáze je ověřována zneužitelnost a reálný bezpečnostní dopad identifikovaných zranitelností. Výsledkem úspěšné exploitace zranitelností může být například získání neoprávněného přístupu k testovanému cíli či jeho zdrojům, elevace oprávnění v rámci dané síťové služby či narušení dostupnosti dané služby či systému.

Pátou a poslední fází je shrnutí výsledků penetračního testu do podoby výstupní zprávy, neboli „reportu“. Výstupní zpráva obsahuje zadání penetračního testu, harmonogram a popis jeho průběhu, popis nalezených zranitelností a doporučení k jejich nápravě. [7, 8, 2]

1.2 Síťový model TCP/IP

Pojem *aplikační protokol* označuje síťový protokol, který je provozován na aplikační vrstvě síťového modelu *TCP/IP*. V této sekci je představen síťový model TCP/IP a jsou zde uvedeny příklady aplikačních protokolů.

Síťový model TCP/IP[4] je síťovou architekturou, která klade důraz na jednoduchost komunikační sítě a koncovým zařízením ukládá odpovědnost za zajištění spolehlivosti síťové komunikace. Tímto přístupem se model TCP/IP odlišuje od referenčního modelu *ISO/OSI*[5], který se naopak zaměřuje na spolehlivou komunikační síť s jednoduchými koncovými zařízeními. Model dělí síťovou komunikaci do 4 vrstev dle jejich účelu a funkce. Těmito vrstvami jsou: vrstva síťového rozhraní, internetová vrstva, transportní vrstva a aplikační vrstva.

Vrstva síťového rozhraní je nejnižší vrstvou modelu TCP/IP a její funkcí je vysílání a příjem data do a z přenosové sítě. Model TCP/IP tuto vrstvu blíže nespecifikuje, její implementace je plně závislá na konkrétní přenosové technologii.

Internetová vrstva, rovněž označována jako síťová, je druhou vrstvou modelu TCP/IP a zajišťuje nespolehlivý (bez garance bezchybného přenosu) přenos dat přes mezilehlá síťová zařízení. Model TCP/IP specifikuje, že na této vrstvě je provozován síťový protokol *Internet Protocol* (IP)[6]. Součástí protokolu IP je adresace síťových zařízení pomocí IP adres.

Transportní vrstva je třetí vrstvou modelu TCP/IP. Tato vrstva zajišťuje přenos dat mezi koncovými aplikačními programy, které jsou provozovány na vyšší vrstvě

[4]Název modelu se odvíjí od označení protokolů, které model využívá. Konkrétně se jedná o transportní protokol TCP a síťový protokol IP.

[5]ISO/OSI: `https://www.cloudflare.com/learning/ddos/glossary/open-systems-interconnection-model-osi/`

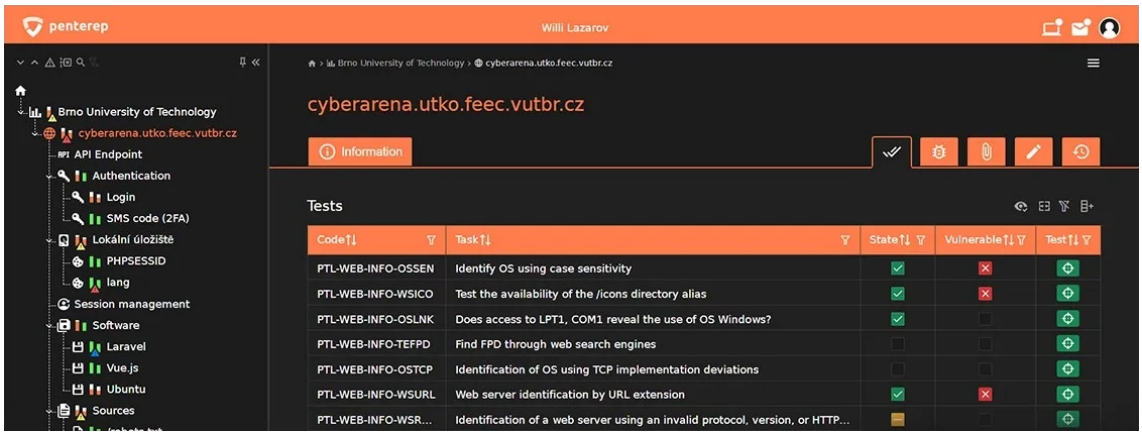
[6]IP: `https://www.ibm.com/docs/en/aix/7.3?topic=protocols-internet-protocol`

modelu. Nejčastěji je zde využíván spolehlivý protokol *Transmission Control Protocol* (TCP)[7], ale hojně využíván je také nespolehlivý protokol *User Datagram Protocol* (UDP)[8]. Transportní vrstva rovněž zajišťuje adresaci komunikujících aplikačních programů pomocí *síťových portů*. Síťové porty jsou číselná označení v rozsahu 1 až 65535.

Aplikační vrstva je čtvrtou a nejvyšší vrstvou modelu TCP/IP. Tato vrstva je rozhraním koncových programů umožňující přístup ke komunikační síti. Je zde provozována celá řada aplikačních protokolů, jako například protokoly FTP, SSH, SMTP, POP3 či IMAP. Zmíněné aplikační protokoly jsou v této práci popsány podrobněji v kapitole č. 2. [9, 10]

1.3 Platforma Penterep

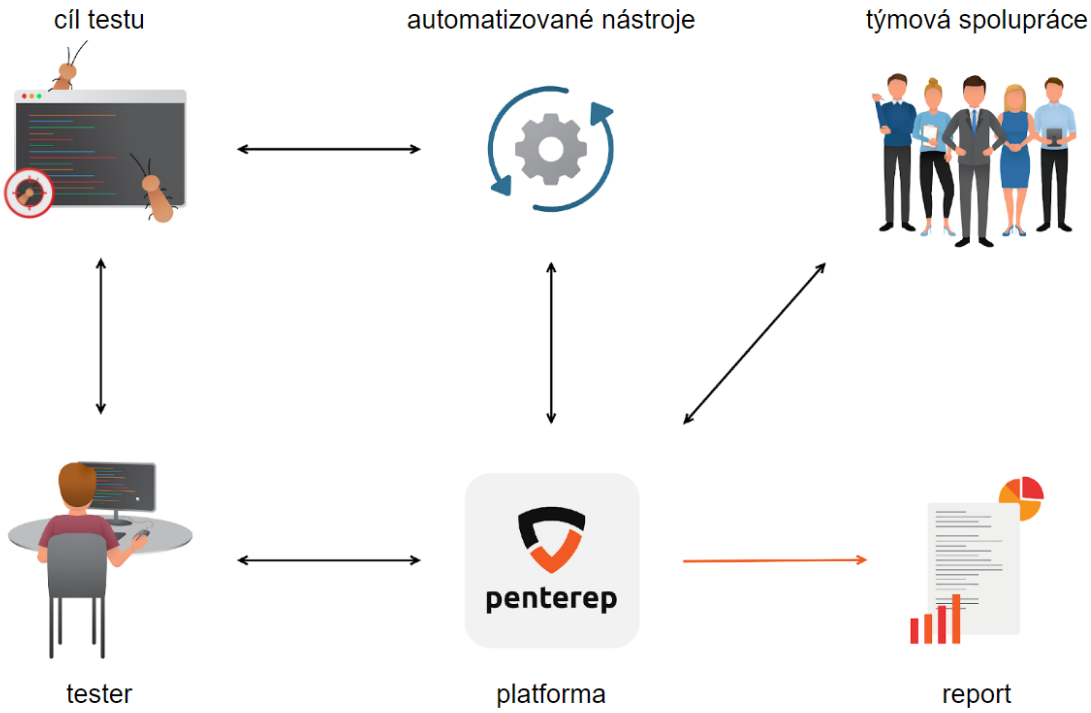
Platforma Penterep je komplexním nástrojem pro podporu penetračního testování. Penetračním testerům poskytuje kontrolní seznamy, obsahující jednotlivé kroky, které by měly být testery vykonány. Poskytnutí návodných kontrolních seznamů celý proces penetračního testování usnadňuje a zpřístupňuje, a to i méně zkušeným pracovníkům. Na obrázku 1.1 je uveden příklad kontrolního seznamu. Některé z bodů těchto kontrolních seznamů rovněž nabízí možnost spuštění automatizovaných testovacích modulů pro automatickou či poloautomatickou kontrolu daného bodu seznamu, čímž platforma Penterep dosahuje efektivního propojení manuálního a automatického testování. Automatizované moduly jsou veřejně dostupné na *Python Package Index* (PyPI) repozitáři platformy Penterep[9].



Obr. 1.1: Kontrolní seznam platformy Penterep [11]

[7]TCP: `https://datatracker.ietf.org/doc/html/rfc9293`
[8]UDP: `https://datatracker.ietf.org/doc/html/rfc768`
[9]PyPI Penterep repozitář: `https://pypi.org/user/penterep/`

Platforma dále nabízí například správu jednotlivých projektů a uložení relevantních informací, umožňuje týmovou spolupráci několika penetračních testerů, nebo nabízí automatickou tvorbu závěrečných reportů. Poskytnutím těchto možností platforma šetří čas strávený na jednotlivých projektech, a umožňuje tak testerům věnovat více času samotnému penetračnímu testování. [12, 13]



Obr. 1.2: Schéma užití platformy Penterep [14]

2 Výzkum aplikačních protokolů a jejich zranitelností

Náplní této kapitoly je představení vybraných aplikačních protokolů, a to konkrétně FTP, SSH, SMTP, POP3 a IMAP. Pro každý z protokolů je zde popsán jeho účel a proces fungování, společně se zranitelnostmi nalezenými průzkumem veřejně dostupných dokumentů a informací. Informace o zranitelnostech jednotlivých protokolů byly čerpány primárně ze sbírek typu *HackTricks*[1] a *The Hacker Recipes*[2]. Tyto sbírky obsahují známé zranitelnosti a postupy, které mohou být užitečné při procesu penetračního testování. Některé zranitelnosti byly nalezeny i mimo tyto sbírky, například vlastním studiem *RFC*[3] dokumentů daných aplikačních protokolů.

2.1 Protokol přenosu souborů FTP

Protokol *File Transfer Protocol* (FTP) slouží ke spolehlivému a efektivnímu síťovému přenosu souborů. Jedná se o textový protokol typu klient-server využívající transportní protokol TCP. Komunikace protokolu v jeho základní verzi není zabezpečena a data jsou tak přenášena v nešifrované podobě. Existuje však i rozšířená varianta protokolu *File Transfer Protocol Secure* (FTPS), která využívá protokoly *Secure Sockets Layer* (SSL)[4] či *Transport Layer Security* (TLS)[5] k zabezpečení přenášené komunikace. Varianta FTPS může být provozována v implicitním či explicitním režimu. Při explicitním režimu dochází nejprve k navázání spojení bez zabezpečení, ale následně je v tomto spojení klientem iniciováno zabezpečení pomocí příkazu `AUTH`. Explicitní režim pro svoji komunikaci využívá identické síťové porty jako nezabezpečená varianta protokolu FTP. Při implicitním režimu server již při prvotním navázání spojení očekává zabezpečení protokoly SSL/TLS a pro komunikaci standardně využívá odlišné síťové porty.

FTP při svém běhu používá dvě souběžná TCP spojení, a to kontrolní a datové. Kontrolní spojení slouží k výměně příkazů a odpovědí mezi klientem a serverem a je realizováno pomocí protokolu TELNET[6]. Na straně serveru toto spojení standardně využívá síťový port 21 (990 v případě implicitního FTPS). Datové spojení slouží k vlastnímu přenosu dat.

[1] HackTricks: `https://book.hacktricks.xyz/`
[2] The Hacker Recipes: `https://www.thehacker.recipes/`
[3] RFC: `https://datatracker.ietf.org/`
[4] SSL: `https://www.cloudflare.com/learning/ssl/what-is-ssl/`
[5] TLS: `https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/`
[6] TELNET: `https://datatracker.ietf.org/doc/html/rfc854`

Při připojení klienta k FTP serveru dojde k ustanovení kontrolního spojení. Prvním krokem, který klient standardně musí provést, je autentizace. Ta probíhá pomocí příkazů `USER` a `PASS`, kterými klient specifikuje uživatelské jméno a heslo. Po úspěšném přihlášení může klient využívat dalších příkazů, jako jsou například `LIST` (výpis obsahu adresáře), `RETR` (stažení souboru ze serveru) či `STOR` (nahrání souboru na server).

Pro přenos dat existují v protokolu FTP dva režimy, a to aktivní a pasivní. V aktivním režimu je sestavení datového spojení iniciováno serverem. Klient si zvolí libovolný síťový port (standardně zdrojový port použitý klientem pro kontrolní spojení), tento port společně se svojí IP adresou pomocí příkazu `PORT` sdělí serveru a server se k tomuto portu připojí, standardně s využitím zdrojového síťového portu 20 (989 v případě implicitního FTPS). Po navázání datového spojení může probíhat přenos dat. V pasivním režimu zahajuje datové spojení klient. Server náhodně zvolí síťový port a společně s odpovídající IP adresou jej skrze kontrolní spojení sdělí klientovi. Klient se následně na obdržený síťový port připojí a naváže tak datové spojení. [15, 16, 17]

Následující text popisuje zranitelnosti protokolu, které byly nalezeny studiem veřejně dostupných informací a dokumentů.

Zranitelná verze

Existuje mnoho implementací FTP serveru, například *Pure-FTPd*[7], *vsftpd*[8] či *FileZilla*[9]. Každé z těchto řešení může v některé ze svých verzí obsahovat veřejně známé zranitelnosti. Pokud je při penetračním testu odhalen konkrétní software a verze FTP serveru, je možné tyto zranitelnosti efektivně vyhledat. [18, 19]

Anonymní přihlášení

FTP servery mohou podporovat tzv. anonymní přihlášení, které umožňuje přihlášení k serveru pomocí uživatelského jména „anonymous“ a libovolného hesla. Při této formě autentizace klient zpravidla získá pouze omezený přístup k serveru, nicméně i tato úroveň přístupu může během penetračního testování poskytnout cenné informace, či rozvinout další možnosti. [20, 18, 19]

Útok FTP bounce

Útok FTP bounce využívá příkaz `PORT`, aby přiměl server k navázání datového spojení s libovolným síťovým zařízením na libovolném síťovém portu. Tento útok může

[7] Pure-FTPd: `https://www.pureftpd.org/project/pure-ftpd/`
[8] vsftpd: `https://security.appspot.com/vsftpd.html`
[9] FileZilla: `https://filezilla-project.org/`

být využit za účelem skenování síťových zařízení a jejich otevřených síťových portů, případně k interakci se síťovými službami vyskytujícími se na těchto portech. Pomocí tohoto útoku je možné interagovat i s odlišnými aplikačními protokoly. Ukázkový postup pro komunikaci s HTTP serverem je následující:

1. Klient na FTP server nahraje textový soubor obsahující validní HTTP žádost.
2. Klient pomocí příkazu `PORT` sdělí serveru, aby datové spojení cílil na IP adresu HTTP serveru a síťový port, na kterém HTTP server naslouchá.
3. Klient pomocí příkazu `RETR` iniciuje přenos souboru s HTTP žádostí na cílový HTTP server, kde dojde ke zpracování této žádosti.

[18, 19]

Slabé přihlašovací údaje

Zabezpečení uživatelských účtů přímo závisí na síle použitých přihlašovacích údajů. Pokud uživatelské účty nacházející se na testovaném FTP serveru používají často vyskytující se či výchozí přihlašovací údaje, mohou být potenciálně kompromitovány s využitím slovníkového útoku či útoku hrubou silou. Pokud byly během penetračního testu již nalezeny nějaké validní přihlašovací údaje, měly by být vyzkoušeny také. [18, 19]

Odposlech nešifrované komunikace

Při provozu nešifrované verze protokolu FTP probíhá komunikace v čitelné nešifrované podobě. Pokud je možné odposlouchávat síťovou komunikaci testovaného FTP serveru, je možné číst klientskou FTP komunikaci, jako je přenos jména a hesla, nebo obsah přenášených souborů. Tato zranitelnost se vyskytuje rovněž u explicitní varianty protokolu FTPS, jelikož klient nemusí využít přechodu na zabezpečené spojení a může se pokusit dále komunikovat bez zabezpečení. [18, 19, 21]

Neaktuální verze protokolů SSL/TLS a jejich podporovaných algoritmů

V případě kdy testovaný server provozuje šifrovanou komunikaci (standardně porty 990 a 989) a pro její realizaci podporuje neaktuální verze protokolů SSL/TLS či jejich kryptografických algoritmů, je potenciálně možné narušit integritu přenášené komunikace pozměněním obsahu přenášených zpráv, nebo komunikaci dešifrovat, a tím získat neoprávněný přístup k jejímu obsahu. [22, 23]

2.2 Protokol vzdálené správy SSH

Protokol *Secure Shell* (SSH) je aplikačním protokolem umožňujícím bezpečnou komunikaci zařízení skrze nezabezpečenou síť. Jedná se o binární protokol typu klient-server využívající transportní protokol TCP a standardní síťový port s číslem 22.

Typickým využitím protokolu SSH je bezpečné vzdálené přihlášení umožňující interakci s operačním systémem, kde se nachází SSH server. Tímto způsobem využití protokol SSH nahrazuje starší nezabezpečené protokoly jako TELNET či rlogin[10]. Kromě vzdáleného přihlášení SSH dále umožňuje tzv. tunelování síťového provozu a přesměrování síťových portů. Při těchto procesech dochází k ustanovení zabezpečeného SSH spojení a následnému přeposílání síťového provozu skrze toto spojení. Dalším typickým využitím protokolu SSH je přenos souborů.

Bezpečnost protokolu SSH spočívá v zajištění autentizace komunikujících stran pomocí asymetrické kryptografie (s využitím tzv. SSH klíčů), šifrování přenášených dat a zajištění jejich integrity. Autentizace serveru probíhá vždy pomocí asymetrické kryptografie s využitím jeho veřejného klíče. Pro autentizaci klienta existuje více způsobů, z nichž nejpoužívanějšími jsou použití přihlašovacího jména a hesla a použití klíčů asymetrické kryptografie. Při navazování SSH spojení server klientovi oznámí dostupné algoritmy pro ustanovení šifrovacího klíče a samotné algoritmy pro následné šifrování komunikace. Následně dojde ke zvolení vhodných algoritmů a k dohodnutí symetrického šifrovacího klíče. Navazující komunikace pak již probíhá šifrovaně s využitím předešle dohodnutých parametrů. [24, 25, 26, 27]

Následující text popisuje zranitelnosti protokolu, které byly nalezeny studiem veřejně dostupných informací a dokumentů.

Podpora slabých kryptografických algoritmů

Pokud SSH server podporuje starší kryptografické algoritmy, které jsou v dnešní době již považovány za nebezpečné, je potenciálně možné narušit integritu přenášené komunikace pozměněním obsahu přenášených zpráv, nebo komunikaci dešifrovat, a tím získat neoprávněný přístup k jejímu obsahu. [28]

Známý veřejný klíč serveru

Pokud testovaný SSH server využívá staticky nastavený a veřejně známý SSH klíč (respektive kombinaci veřejného a soukromého klíče), je možné vůči tomuto serveru provést útok typu Man-in-the-Middle. [28, 29]

[10] rlogin: `https://datatracker.ietf.org/doc/html/rfc1282`

Slabé přihlašovací údaje

Při autentizaci s využitím jména a hesla se jedná o zranitelnost totožnou se zranitelností již popsanou u protokolu FTP v sekci 2.1. [28]

Při využití autentizace pomocí SSH klíčů existují obdobné možnosti. SSH servery mohou být nakonfigurovány tak, aby umožňovaly přihlášení pomocí staticky definovaných a veřejně známých SSH klíčů. Pokud tomu tak je, je možné využít některý z dostupných seznamů takových klíčů a pokusit se s jeho pomocí k serveru přihlásit. Rovněž pokud byly během penetračního testování již nalezeny nějaké existující soukromé SSH klíče, měly by být vyzkoušeny. [29]

Zranitelná verze

Totožné se zranitelností již popsanou u protokolu FTP v sekci 2.1. [28]

2.3 Protokol přenosu elektronické pošty SMTP

Protokol *Simple Mail Transfer Protocol* (SMTP) je protokolem typu klient-server sloužícím pro síťový přenos elektronické pošty (e-mail). Jedná se o textový protokol využívající transportní protokol TCP, standardně naslouchající na síťovém portu 25. Vyskytují se i takové SMTP servery, které místo portu 25 využívají port 2525, ale tento síťový port není standardizován. Samotný protokol SMTP neimplementuje zabezpečení přenášené komunikace, ovšem existuje i jeho rozšířená varianta *Simple Mail Transfer Protocol Secure* (SMTPS), která přináší zabezpečení provozu pomocí protokolů SSL/TLS. Zabezpečení u SMTPS může být zaváděno v implicitním či explicitním režimu. Při implicitním režimu je zabezpečení relace realizováno ihned při prvotním připojení klienta k serveru. SMTPS servery pracující v tomto režimu standardně využívají síťový port 587, ale vyskytují se i takové SMTPS servery, které místo portu 587 využívají nestandardizovaný port 465. Při explicitním režimu je naopak nejprve navázáno nezabezpečené spojení, v rámci kterého klient využije příkaz `STARTTLS` pro zahájení ustanovení zabezpečeného spojení. Dalším rozšířením protokolu SMTP je jeho verze *Extended Simple Mail Transfer Protocol* (ESMTP), která protokolu přidává nové možnosti, jako jsou například posílání příloh či podpora protokolu TLS.

Po navázání spojení klienta k serveru dojde k výměně úvodních zpráv a informací. Server potvrdí navázání spojení odpovědí s kódem 220. Součástí této odpovědi mohou být i informace o softwaru a verzi implementace serveru. Klient na tuto zprávu reaguje vlastní úvodní zprávou `HELO`, kterou žádá o navázání SMTP relace, či zprávou `EHLO`, kterou žádá o navázání ESMTP relace. Po této úvodní výměně zpráv může klient serveru zasílat příkazy, jako jsou například `HELP` (zjištění dostupných

příkazů), `MAIL` (zahájení poštovní transakce), `RCPT` (identifikace příjemce e-mailu) či `AUTH` (autentizace s využitím specifikovaného protokolu). Autentizace není povinným krokem a některé SMTP servery mohou umožňovat odesílání e-mailových zpráv i bez jejího provedení.

E-mailové zprávy přenášené protokolem SMTP pomocí příkazu `MAIL` se skládají z obálky a obsahu. Obálka obsahuje primárně adresy odesílatele a příjemců e-mailové zprávy a slouží pouze pro síťový přenos e-mailové zprávy. Obsah e-mailové zprávy se dále dělí na hlavičky a tělo zprávy. Hlavičky zprávy obsahují informace jako jsou adresa odesílatele, adresy příjemců či předmět zprávy a slouží pro informování příjemce e-mailové zprávy o jejích detailech. Tělo zprávy pak obsahuje samotný obsah zprávy, kterým kromě obyčejného textu mohou při využití rozšíření MIME být i přílohy. [21, 30, 31, 32, 33, 34, 35, 36]

Mechanismy SPF, DKIM a DMARC

Odesílatel může do obálky i hlaviček e-mailové zprávy zadat libovolné informace, což bez implementace dalších bezpečnostních opatření umožňuje snadné podvržení informací o odesílateli e-mailové zprávy a jeho zdrojové doméně. Pro zabránění podvržení e-mailových zpráv byly vytvořeny bezpečnostní mechanismy *Sender Policy Framework* (SPF), *DomainKeys Identified Mail* (DKIM) a *Domain-based Message Authentication, Reporting, and Conformance* (DMARC).

Mechanismus SPF zabraňuje podvržení adresy odesílatele v obálce e-mailové zprávy. Provozovatel domény vloží do *Doman Name System* (DNS) záznamů nový záznam typu TXT, který definuje IP adresy serverů, které mohou odesílat e-mailové zprávy z této domény. SMTP servery pak mohou kontrolovat příchozí e-mailové zprávy tím, že porovnají IP adresu klienta s IP adresou uvedenou v DNS záznamech domény, do které náleží odesílatel uvedený v obálce zprávy.

Mechanismus DKIM zajišťuje integritu hlaviček a těla zprávy. Provozovatel domény opět vloží do DNS záznamů nový záznam typu TXT, který obsahuje veřejný klíč asymetrické kryptografie. Při odesílání e-mailové zprávy je vytvořena a do zprávy přidána nová hlavička obsahující digitální podpis důležitých částí zprávy (např. informace o odesílateli a příjemci). Tento podpis je vytvořen s využitím soukromého klíče domény. Při přijetí zprávy SMTP serverem je pak možné ověřit digitální podpis získáním veřejného klíče z DNS záznamů domény, která je uvedena v DKIM hlavičce.

Využití mechanismů SPF a DKIM neumožňuje zcela zabránit podvržení e-mailových zpráv. Mechanismus SPF zabraňuje podvržení obálky, ovšem nezabraňuje podvržení hlaviček, které jsou zobrazeny koncovému uživateli zprávy. Mechanismus DKIM zachovává integritu obsahu zprávy, ovšem opět nezabraňuje podvržení hlaviček, jelikož

umožňuje podepisovat i zprávy, u nichž se neshoduje doména odesílatele uvedeného v hlavičce zprávy s doménou uvedenou v DKIM hlavičce. Mechanismus DMARC umožňuje stanovit politiku zacházení s výsledky kontrol SPF a DKIM mechanismů a rovněž stanovit, co se má stát s e-mailovými zprávami, které stanovenou politiku nesplní (např. označení jako spam či úplné blokování zprávy). Vhodným nastavením DMARC politiky je možné vynutit kontrolu SPF a DKIM mechanismů, ale rovněž i shodu domény odesílatele uvedené v obálce a hlavičkách zprávy (hlavička odesílatele i hlavička DKIM). Nastavení DMARC politiky je opět provedeno vložením nového TXT záznamu do DNS záznamů domény. [37, 38, 39, 40, 41, 42].

Následující text popisuje zranitelnosti protokolu, které byly nalezeny studiem veřejně dostupných informací a dokumentů.

Chybějící autentizace

Některé SMTP servery nemusí vyžadovat klientskou autentizaci k tomu, aby umožnili klientům odesílat e-mailové zprávy. Pokud testovaný SMTP server autentizaci nevyžaduje, je možné skrze něj odesílat e-mailové zprávy bez znalosti přihlašovacích údajů. [43]

Open relay

Open relay se nazývá SMTP server, který nevyžaduje autentizaci a zároveň umožňuje odesílat e-mailové zprávy s libovolnými zdrojovými a cílovými e-mailovými adresami. [43, 44]

NTLM autentizace

Protokol SMTP podporuje několik autentizačních mechanismů. Při využití mechanismu NTLM dochází během přihlášení k výměně specifických zpráv mezi klientem a serverem, z nichž některé mohou prozrazovat informace o operačním systému, interním názvu či doméně serveru. Pokud testovaný SMTP server podporuje NTLM autentizaci, je možné skrze tento mechanismus odhalit interní informace. [43]

Automatické doplnění e-mailové adresy

Pokud je SMTP serveru zaslán příkaz `MAIL FROM` bez specifikace doménové části e-mailové adresy, může server za chybějící doménu dosadit vlastní doménové jméno. Pokud SMTP server toto chování podporuje, může tím odhalovat doposud neznámé interní informace. [43]

Enumerace e-mailových adres

Při zaslání příkazu `RCPT TO`, `VRFY` či `EXPN` a e-mailové adresy či uživatelského jména SMTP server kontroluje, zda uživatel se zadanou adresou či jménem existuje. Pokud uživatel neexistuje, SMTP server klienta o této skutečnosti informuje. Server tuto kontrolu provádí pouze v případě, kdy zadaná e-mailová adresa či uživatelské jméno spadá do domény, jejíž uživatele SMTP server zná. Při penetračním testování SMTP serveru lze tohoto chování využít k enumeraci existujících e-mailových adres či uživatelských jmen. [43]

Slabá či chybějící konfigurace mechanismů SPF, DKIM a DMARC

Protokol SMTP vyžaduje implementaci mechanismů SPF, DKIM a DMARC, aby bylo možné zabránit odesílání podvržených e-mailových zpráv. Pokud testovaná doména tyto mechanismy neimplementuje, nebo je provozuje ve slabé či nevhodné konfiguraci, je možné využít SMTP server k odeslání podvržené e-mailové zprávy (např. pro účely phishingu). [43]

SMTP smuggling

Zranitelnost SMTP smuggling spočívá v rozdílné interpretaci znaků představujících konec e-mailové zprávy příchozím a odchozím SMTP serverem. Konec zprávy standardně obsahuje kombinaci znaků *Carriage Return* (CR) a *Line Feed* (LF), tečku a opět kombinací znaků CR a LF. Pokud odchozí SMTP server vyžaduje použití obou znaků pro ukončení zprávy, ale příchozí SMTP server akceptuje i takové ukončení zprávy, ve kterém nejsou použity oba znaky, nastává zranitelnost SMTP smuggling. V takovém případě je možné odeslat e-mailovou zprávu, která po neúplné kombinaci znaků CR a LF obsahuje druhou e-mailovou zprávu, která již je zakončena úplnou kombinací těchto znaků. Ve chvíli, kdy příchozí SMTP server tyto data přijme, vyhodnotí je jako 2 e-mailové zprávy a pokusí se doručit obě. Zneužitím této zranitelnosti může dojít k obejití některých bezpečnostních mechanismů (SPF, DMARC), jelikož druhá zpráva již neprojde jejich kontrolou na odchozím SMTP serveru. [45]

Obcházení multifaktorové autentizace

Klientské SMTP aplikace mohou od uživatelů vyžadovat splnění multifaktorové autentizace. Pokud jsou během penetračního testu odhaleny přihlašovací údaje používané pro přihlášení do takovéto aplikace, je možné je využít pro přímý přístup k testovanému SMTP serveru, a tedy k obejití multifaktorové autentizace aplikace SMTP klienta. [46]

SMTP injection

Pokud se v testovaném prostředí nachází kromě SMTP serveru i jiná aplikace (například webová aplikace), která s SMTP serverem zřejmě interaguje, je vhodné se u této aplikace zaměřit na hledání zranitelnosti typu SMTP injection. [47]

Zranitelná verze

Totožné se zranitelností již popsanou u protokolu FTP v sekci 2.1. [43]

Slabé přihlašovací údaje

Totožné se zranitelností již popsanou u protokolu FTP v sekci 2.1. [43]

Odposlech nešifrované komunikace

Totožné se zranitelností již popsanou u protokolu FTP v sekci 2.1. [43]

2.4 Protokol příjmu elektronické pošty POP3

Protokol *Post Office Protocol 3* (POP3) je textový protokol typu klient-server sloužící pro příjem elektronické pošty. Využívá transportní protokol TCP se standardním síťovým portem 110. Samotný protokol POP3 neimplementuje žádné zabezpečení přenášené komunikace, ovšem existuje i jeho rozšířená varianta *Post Office Protocol 3 Secure* (POP3S), která pro zabezpečení provozu využívá protokoly SSL či TLS. Tuto variantu lze provozovat v implicitním či explicitním režimu. Při implicitním režimu je již prvotní spojení klienta a serveru navazováno zabezpečeně pomocí protokolů SSL/TLS. Pro tento režim je standardně využíván síťový port 995. V explicitním režimu je nejprve navázáno nezabezpečené spojení, ve kterém je následně klientem zahájeno zabezpečení příkazem `STLS`. Explicitní režim využívá standardní síťový port protokolu POP3, tedy 110.

Po připojení klienta POP3 server standardně odpoví uvítací zprávou, po které očekává autentizaci klienta. Protokol POP3 podporuje několik způsobů autentizace. Pomocí příkazů `USER` a `PASS` se klient může autentizovat poskytnutím uživatelského jména a hesla. Příkaz `APOP` umožňuje autentizaci prokázáním znalosti tajemství sdíleného mezi klientem a serverem. POP3 servery rovněž mohou podporovat příkaz `SASL`, který umožňuje využít další rozšiřující autentizační mechanismy, jako jsou anonymní autentizace či autentizace pomocí protokolu Kerberos[11]. Po úspěšné autentizaci je klientovi poskytnut přístup k jeho e-mailovým zprávám. Pro operace se zprávami má klient k dispozici řadu příkazů, například `STAT` (výpis počtu a velikosti

[11]Kerberos: `https://www.kerberos.org/software/tutorial.html`

e-mailových zpráv), `RETR` (přístup k e-mailové zprávě) či `DELE` (označení e-mailové zprávy pro smazání). Po dokončení požadovaných operací klient standardně ukončuje relaci příkazem `QUIT`. Po ukončení relace jsou ze serveru odstraněny veškeré e-mailové zprávy, které byly předešle označeny příkazem `DELE`[12]. [21, 30, 48, 49, 50]

Následující text popisuje zranitelnosti protokolu, které byly nalezeny studiem veřejně dostupných informací a dokumentů.

Anonymní přihlášení

Jedním z autentizačním mechanismům protokolu POP3 je anonymní autentizace. pokud testovaný POP3 server tento mechanismus podporuje, je možné jej využít a bez znalosti validních přihlašovacích údajů získat přístup k serveru. [51]

POP3 injection

Pokud se kromě POP3 serveru v testovaném prostředí nachází i další aplikace (například webové aplikace), které s POP3 serverem zřejmě komunikují, je vhodné se u těchto aplikací zaměřit na hledání zranitelností typu POP3 injection. [52, 47]

Zranitelná verze

Totožné se zranitelností již popsanou u protokolu FTP v sekci 2.1. [51]

NTLM autentizace

Totožné se zranitelností již popsanou u protokolu SMTP v sekci 2.3. [51]

Slabé přihlašovací údaje

Totožné se zranitelností již popsanou u protokolu FTP v sekci 2.1. [51]

Odposlech nešifrované komunikace

Totožné se zranitelností již popsanou u protokolu FTP v sekci 2.1. [51]

Neaktuální verze protokolů SSL/TLS a jejich podporovaných algoritmů

Totožné se zranitelností již popsanou u protokolu FTP v sekci 2.1. [22, 23]

Obcházení multifaktorové autentizace e-mailových klientů

Totožné se zranitelností již popsanou u protokolu SMTP v sekci 2.3. [46]

[12]Zprávy mohou být serverem odstraňováno i bez jejich explicitního označení, a to po uplynutí doby stanové serverem (např. 7 dní pro přečtené zprávy a 60 dní pro nepřečtené zprávy).

2.5 Protokol přístupu k elektronické poště IMAP

Protokol *Internet Message Access Protocol* (IMAP) je protokolem typu klient-server a slouží pro vzdálený přístup k elektronické poště a manipulaci s ní. Jedná se o textový protokol využívající transportní protokol TCP se standardním síťovým portem 143. Protokol je ve své základní verzi provozován bez jakéhokoliv zabezpečení provozu. Existuje ovšem i rozšířená varianta protokolu *Internet Message Access Protocol Secure* (IMAPS), která k zabezpečení komunikace využívá protokoly SSL či TLS. Varianta IMAPS může být provozována v explicitním či implicitním režimu. Při explicitním režimu dochází nejprve k navázání spojení bez zabezpečení, ale následně je v tomto spojení klientem iniciováno zabezpečení pomocí příkazu `STARTTLS`. Explicitní režim pro svoji komunikaci využívá identický síťový port jako nezabezpečená varianta protokolu IMAP, tedy 143. Při implicitním režimu server již při prvotním navázání spojení očekává zabezpečení protokoly SSL/TLS a pro komunikaci standardně využívá síťový port 993.

Po navázání spojení mezi serverem a klientem server uvítá klienta úvodní zprávou. Po této zprávě očekává server autentizaci klienta. Standardním autentizačním mechanismem protokolu IMAP je zadání uživatelského jména a hesla pomocí příkazu `LOGIN`. Existují ale i další autentizační mechanismy přístupné pomocí příkazu `AUTHENTICATE`, jako například anonymní přihlášení či využití protokolu Kerberos. Po úspěšné autentizaci jsou klientovi zpřístupněny jeho e-mailové zprávy. Klient v této fázi může využít další příkazy pro přístup k těmto zprávám, jako například `SELECT` (výběr e-mailové schránky), `LIST` (výpis seznamu e-mailových schránek) či `FETCH` (vyžádání dat e-mailové zprávy). Při přístupu k e-mailovým zprávám jsou klientům odeslány kopie těchto zpráv a původní zprávy jsou na serveru nadále uchovávány. Tímto chováním se protokol IMAP liší od protokolu POP3 (viz sekce 2.4), u kterého jsou e-mailové zprávy ze serveru standardně mazány po jejich vyzvednutí klientem. Tato vlastnost protokolu IMAP umožňuje klientům ke svým e-mailovým zprávám přistupovat z více zařízení zároveň. [21, 30, 53, 54, 55]

Následující text popisuje zranitelnosti protokolu, které byly nalezeny studiem veřejně dostupných informací a dokumentů.

IMAP injection

Pokud se v testovaném prostředí nachází kromě IMAP serveru i jiná aplikace (například webová aplikace), která s IMAP serverem zřejmě interaguje, je vhodné se u této aplikace zaměřit na hledání zranitelnosti typu IMAP injection. [47]

Zranitelná verze

Totožné se zranitelností již popsanou u protokolu FTP v sekci 2.1. [56]

Anonymní přihlášení

Totožné se zranitelností již popsanou u protokolu POP3 v sekci 2.4. [54]

NTLM autentizace

Totožné se zranitelností již popsanou u protokolu SMTP v sekci 2.3. [56]

Slabé přihlašovací údaje

Totožné se zranitelností již popsanou u protokolu FTP v sekci 2.1. [56]

Odposlech nešifrované komunikace

Totožné se zranitelností již popsanou u protokolu FTP v sekci 2.1. [56]

Neaktuální verze protokolů SSL/TLS a jejich podporovaných algoritmů

Totožné se zranitelností již popsanou u protokolu FTP v sekci 2.1. [22, 23]

Obcházení multifaktorové autentizace e-mailových klientů

Totožné se zranitelností již popsanou u protokolu SMTP v sekci 2.3. [46]

3 Vlastní návrh a implementace kontrolních seznamů

V této kapitole je představen vlastní návrh kontrolních seznamů pro penetrační testování aplikačních protokolů uvedených v kapitole 2. Rovněž je zde popsána implementace kontrolních seznamů pro platformu Penterep.

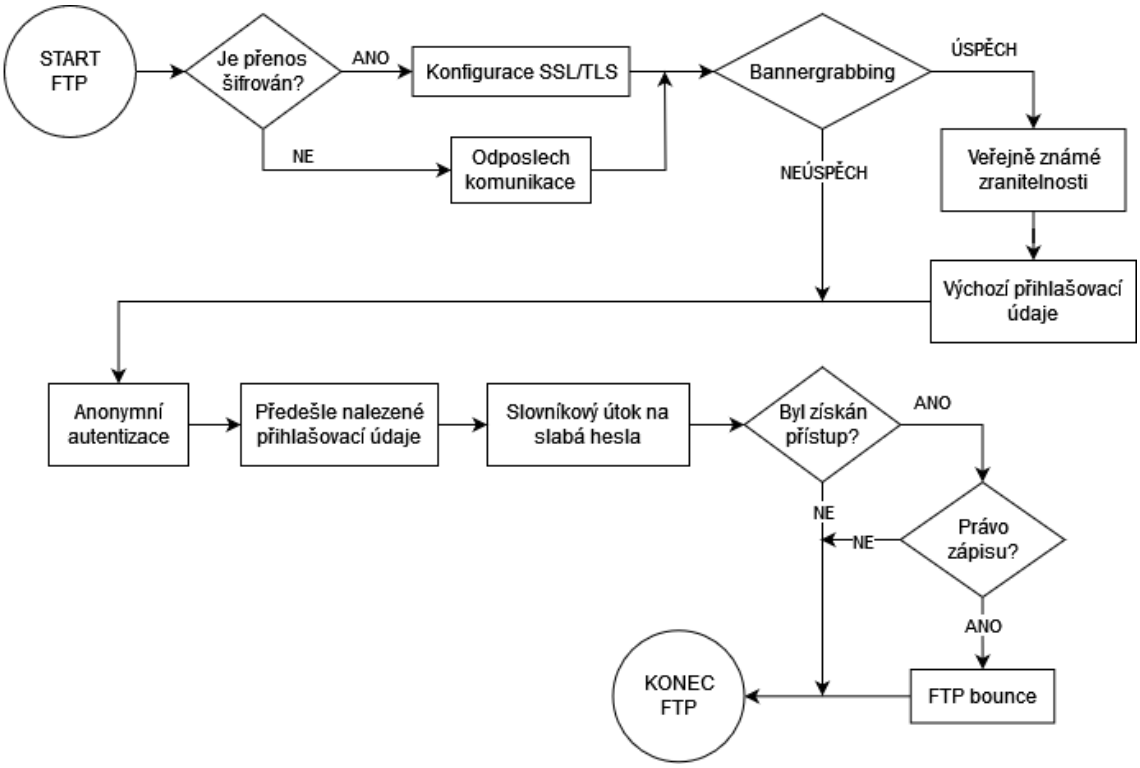
3.1 Návrh kontrolních seznamů

Návrh kontrolních seznamů spočívá ve vytvoření grafických diagramů zobrazujících postup, který by měl být vykonán v rámci penetračního testování daného aplikačního protokolu.

3.1.1 Protokol FTP

Navržený kontrolní seznam pro aplikační protokol FTP (obrázek 3.1) obsahuje celkem 12 kroků a vychází ze zranitelností uvedených v sekci 2.1. Prvním bodem seznamu je kontrola šifrovaného přenosu. Pokud je přenos šifrován, následuje kontrola konfigurace protokolů SSL/TLS. Pokud provoz šifrován není, je následujícím krokem pokus o odposlech síťového provozu serveru. Dalším krokem je pokus o provedení tzv. bannergrabbingu, tedy zjištění softwaru a verze FTP serveru. Při úspěšném zjištění následují navazující kroky, a to vyhledání veřejně známých zranitelností pro danou verzi FTP serveru (například s využitím nástroje ptvulnsearcher[1]) a pokus o nalezení a využití výchozích přihlašovacích údajů k aplikačnímu serveru. Následující 3 kroky se věnují různým způsobům autentizace, kterými jsou anonymní autentizace, využití předešle nalezených přihlašovacích údajů (pokud takové údaje byly nalezeny) a slovníkový útok vůči slabým uživatelským heslům. Pokud byl během penetračního testování získán přístup k FTP serveru a server povoluje zápis souborů, realizuje se poslední krok seznamu, a to využití útoku FTP bounce pro přístup k interním síťovým službám.

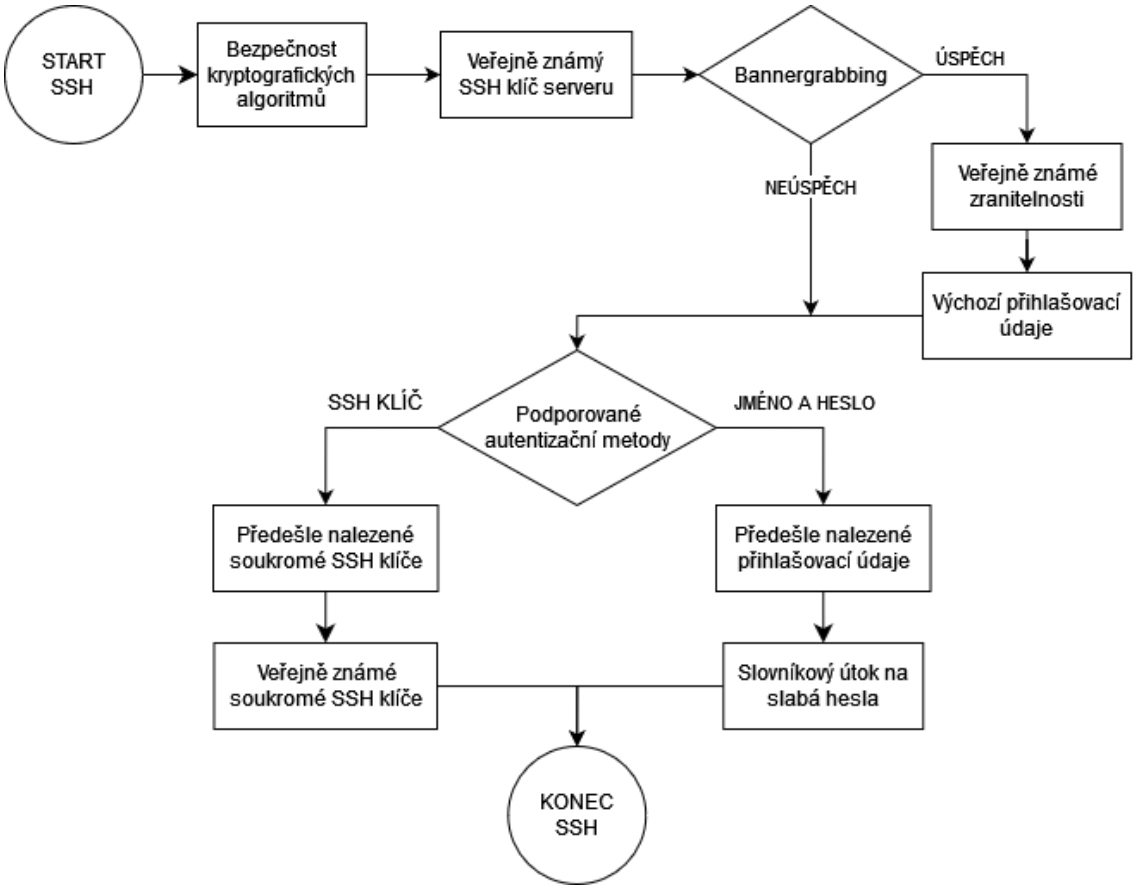
[1] ptvulnsearcher: `https://pypi.org/project/ptvulnsearcher/`



Obr. 3.1: Kontrolní seznam FTP

3.1.2 Protokol SSH

Kontrolní seznam protokolu SSH (obrázek 3.2) má celkem 10 bodů a vychází ze zranitelností uvedených v sekci 2.2. Seznam začíná ověřením aktuálnosti kryptografických algoritmů, které SSH server podporuje. Dalším krokem je kontrola, zda SSH server nevyužívá veřejně známý SSH klíč, který by umožnil provedení útoků typu Man-in-the-middle. Následuje pokus o zjištění softwaru a verze SSH serveru a pokud je tento krok úspěšný, pokračuje se dále hledáním veřejně známých zranitelností pro tuto verzi serveru a také pokusem o nalezení a využití výchozích přihlašovacích údajů. Zbylé kroky seznamu se věnují snaze o úspěšné přihlášení k serveru. Seznam je rozvětven do dvou nezávislých větví v závislosti na autentizačních metodách, které server podporuje. V případě, že server podporuje více než jednu z uvedených autentizačních metod, měl by být vykonán průchod všech odpovídajících větví. V případě autentizace pomocí SSH klíčů je dalším krokem pokus o přihlášení pomocí předešle nalezených soukromých SSH klíčů, pokud tedy takové klíče byly během penetračního testování nalezeny. Dalším a posledním krokem je pak kontrola, zda server umožňuje přihlášení pomocí některého z veřejně známých soukromých SSH klíčů. V případě autentizace pomocí jména a hesla je dalším krokem pokus o přihlášení pomocí předešle nalezených přihlašovacích údajů, pokud takové údaje byly během testování nalezeny. Posledním krokem je pokus o přihlášení pomocí slovníkového útoku.

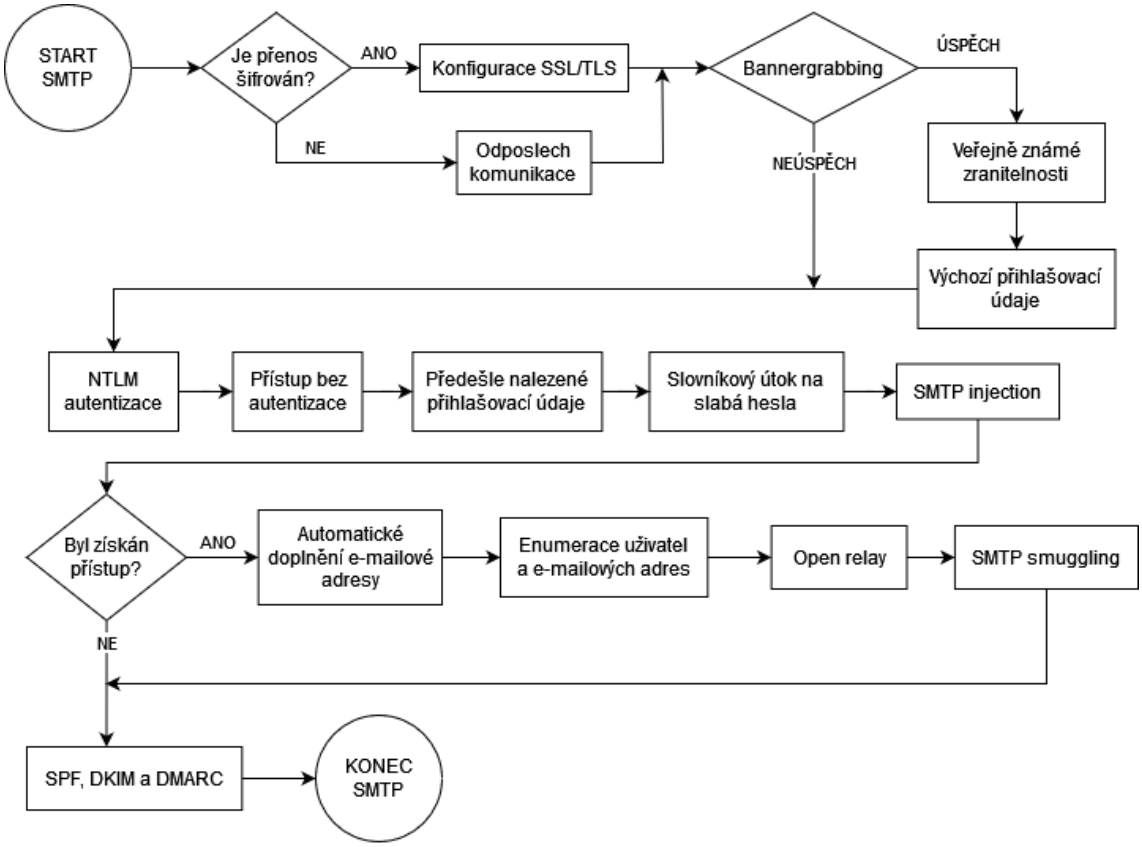


Obr. 3.2: Kontrolní seznam SSH

3.1.3 Protokol SMTP

Navržený kontrolní seznam aplikačního protokolu SMTP (obrázek 3.3) obsahuje celkem 17 kroků a vychází ze zranitelností uvedených v sekci 2.3. Úvodním bodem je kontrola, zda je přenos SMTP protokolu šifrován. Pokud je přenos šifrován, následuje kontrola konfigurace protokolů SSL/TLS. Pokud provoz šifrován není, je následujícím krokem pokus o odposlech síťového provozu serveru. Seznam dále pokračuje pokusem o zjištění softwaru a verze SMTP serveru. Při úspěšném zjištění verze následují 2 navazující kroky, a to vyhledání veřejně známých zranitelností pro danou verzi SMTP serveru a pokus o nalezení výchozích přihlašovacích údajů. Dalším bodem je pak využití NTLM autentizace k odhalení interních informací o SMTP serveru. Následující 3 kroky zkoumají různé způsoby přihlášení k SMTP serveru. Nejprve je provedena kontrola, zda server vyžaduje klientskou autentizaci pro odesílání e-mailových zpráv. Tento krok je následován pokusem o přihlášení s využitím přihlašovacích údajů předešle nalezených během testování a rovněž s využitím slovníkového útoku. Dále následuje kontrola, zda se v testovaném prostředí nachází aplikace komunikující s testovaným SMTP serverem a pokud ano, tak zda se

v této aplikaci nachází zranitelnost SMTP injection. Pokud byl během dosavadního procesu penetračního testování získán přístup k SMTP serveru, jsou dalšími kroky kontrolního seznamu test automatického doplňování e-mailové adresy odesílatele, enumerace existujících e-mailových adres, kontrola tzv. open relay chování a test zranitelnosti SMTP smuggling. Posledním krokem seznamu je pak kontrola výskytu bezpečnostních mechanismů SPF, DKIM a DMARC a kontrola jejich konfigurace.

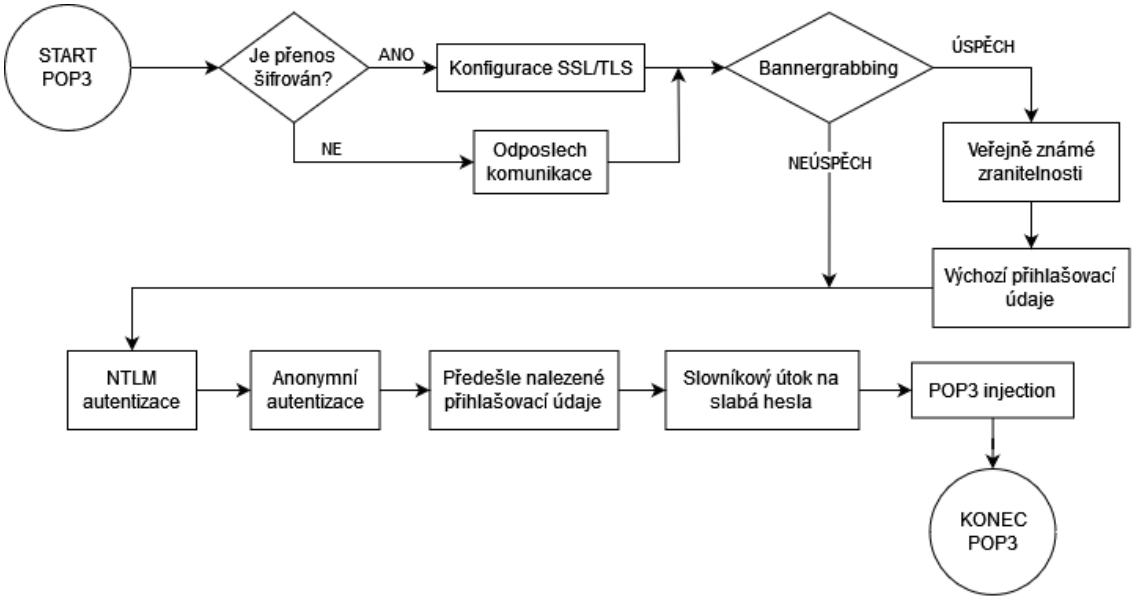


Obr. 3.3: Kontrolní seznam SMTP

3.1.4 Protokol POP3

Kontrolní seznam pro protokol POP3 (obrázek 3.4) se skládá celkem z 11 kroků a byl vytvořen na základě zranitelností představených v sekci 2.4. Diagram začíná kontrolou zabezpečení komunikace protokolu. V případě, že je přenos šifrován, diagram pokračuje kontrolou konfigurace protokolů SSL/TLS. V opačném případě je dalším krokem pokus o odposlech nešifrované síťové komunikace serveru. Diagram dále pokračuje pokusem o zjištění softwaru a verze POP3 serveru. V případě úspěšného zjištění těchto informací následuje vyhledání veřejně známých zranitelností pro danou verzi POP3 serveru a pokus o nalezení platných výchozích přihlašovacích údajů. Dalším krokem sloužícím pro zjištění více informací o serveru je pokus

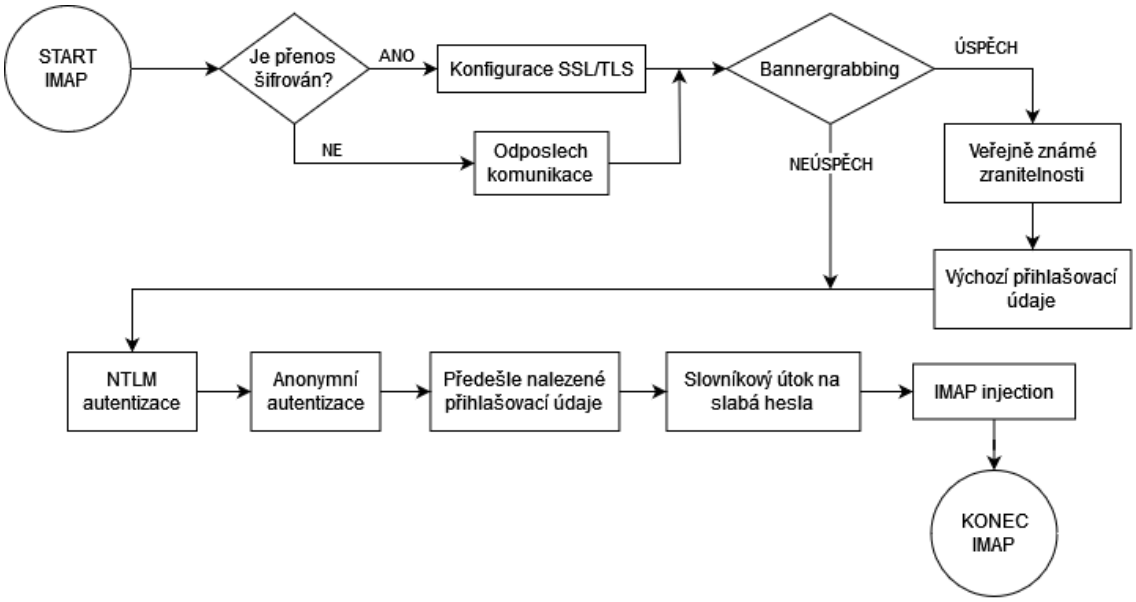
o využití NTLM autentizace. Tento krok je dále následován třemi kroky věnujícími se různým způsobům autentizace, konkrétně přihlášení s využitím anonymní autentizace, předešle nalezených přihlašovacích údajů a slovníkového útoku. Závěrečným krokem diagramu je kontrola, zda se v testovaném prostředí nachází aplikace komunikující s testovaným POP3 serverem a pokud ano, tak zda se v této aplikaci nachází zranitelnost POP3 injection.



Obr. 3.4: Kontrolní seznam POP3

3.1.5 Protokol IMAP

Navržený kontrolní seznam pro aplikační protokol IMAP (obrázek 3.5) obsahuje celkem 11 kroků a vychází ze zranitelností uvedených v sekci 2.5. Návrh kontrolního seznamu pro tento protokol je téměř totožný s návrhem pro protokol POP3, jenž byl předešle popsán v sekci 3.1.4. Jediným rozdílem je poslední krok, ve kterém je hledána zranitelnost IMAP injection namísto zranitelnosti POP3 injection.



Obr. 3.5: Kontrolní seznam IMAP

3.2 Implementace kontrolních seznamů

Každý z kontrolních seznamů navržených v předchozí sekci byl implementován formou textových popisů jednotlivých testů a zranitelností sestrojených ve značkovacím jazyce *Markdown*[2]. Všechny sepsané testy i zranitelnosti se nachází v elektronické příloze této práce, ovšem pro demonstraci obsahu a formátu vytvořených dokumentů je v této sekci uveden výpis jednoho testu a jedné zranitelnosti. Na konci sekce je rovněž uveden snímek obrazovky demonstrující integraci implementovaných kontrolních seznamů do platformy Penterep.

Celkem bylo sepsáno 33 testů, z nichž 4 jsou specifické pro protokol FTP, 5 je specifických pro protokol SSH, 11 pro protokol SMTP, 5 pro protokol POP3 a 5 pro protokol IMAP. Zbylými testy jsou 2 obecné testy aplikovatelné na více aplikačních protokolů a 1 test specifický pro protokoly SMTP, POP3 a IMAP. Každý test se skládá z následujících sekcí:

- Úkol (test) – krátký a výstižný popis testu.
- Popis testu – podrobnější popis testu a bližší specifikace jeho cílů.
- Obtížnost testu – subjektivní hodnocení náročnosti provedení testu.
- Jak provést test – detailní popis testovacího postupu obsahující návodné instrukce a potřebné příkazy.
- Testem lze odhalit tyto zranitelnosti – seznam zranitelností, jejichž výskyt lze odhalit daným testem.

[2]Markdown: `https://www.markdownguide.org/`

- Testem lze zpřístupnit navazující testy – seznam testů, pro jejichž vykonání je nutné pozitivní splnění daného testu.

Pro demonstraci obsahu a formátu testů je v následujícím výpisu uveden test bannergrabbing protokolu FTP.

Výpis 3.1: Test bannergrabbing protokolu FTP

```
Úkol (test)
-----------
Je možné z odpovědí FTP serveru odhalit jeho software, případně i
   konkrétní verzi?

Popis testu
-----------
Úkolem tohoto testu je ověřit, zda FTP server poskytuje informace o
    svém softwaru a jeho konkrétní verzi.

Obtížnost testu
---------------
Snadné

Jak provést test
----------------
Pomocí nástroje ‘netcat‘, ‘telnet‘ (nešifrované FTP) či ‘openssl
   s_client‘ (implicitní či explicitní FTPS) se pokuste navázat
   spojení s testovaným FTP serverem. Protokol FTP je standardně
   dostupný na portech 21 (nešifrováno či explicitní FTPS) či 990 (
   implicitní FTPS).

V případě využití nástroje ‘openssl s_client‘ pro explicitní FTPS
   použijte argument ‘-starttls ftp‘. U implicitního FTPS tento
   argument nepoužívejte.

V případě, že po navázání spojení server zdánlivě nereaguje na žádn
   é příkazy, zkuste spojení navázat znovu, tentokrát s konfigurací
    odesílání CRLF sekvence (‘-C‘ pro příkaz ‘netcat‘, ‘-crlf‘ pro
   příkaz ‘openssl s_client‘).

‘‘‘
/usr/bin/nc [-C] -nv <TARGET_IP_ADDRESS> <PORT>
/usr/bin/openssl s_client [-crlf] [-starttls ftp] -connect <
   TARGET_IP_ADDRESS>:<PORT>
‘‘‘

Pokud server po navázání spojení zobrazil nějakou úvodní zprávu,
   pozorně si ji prohlédněte. Dále se pokuste serveru zaslat ná
```

```
   sledující příkazy a pozorně si prohlédněte odpověď serveru.

‘‘‘
STAT
SYST
‘‘‘

Pokud některá ze zpráv serveru obsahuje textový řetězec podobný ná
   sledujícím příkladům, jedná se o identifikaci softwaru FTP
   serveru.

‘‘‘
220 (vsFTPd 3.0.3)
220-FileZilla Server 1.8.1
220 VisuPro Ftp Server V1.00 Ready.
220 Please visit https://filezilla-project.org/
215 UNIX emulated by FileZilla.
‘‘‘

Testem lze odhalit tyto zranitelnosti
-------------------------------------
[[GENERAL-vuln-bannergrabbing]]

Testem lze zpřístupnit navazující testy
---------------------------------------
[[FTP-test-default_creds]]
```

Dále bylo sepsáno celkem 17 zranitelností, z nichž 2 jsou specifické pro protokol SSH a 7 je specifických pro protokol SMTP. Zbylými zranitelnostmi je 7 obecných zranitelností aplikovatelných na více aplikačních protokolů a 1 zranitelnost specifická pro protokoly SMTP, POP3 a IMAP. Každá zranitelnost se skládá z následujících sekcí:

- Název zranitelnosti.
- Popis zranitelnosti – výstižný a shrnující popis zranitelnosti.
- Příčiny – důvod výskytu zranitelnosti.
- Projevy – chování napovídající či dokazující výskyt zranitelnosti.
- Dopady – bezpečnostní dopad, tedy např. co je schopen útočník pomocí zranitelnosti získat či způsobit.
- Náprava / doporučení – kroky vedoucí k nápravě zranitelnosti.
- Závažnost – objektivní hodnocení zranitelnosti dle hodnoty *Common Vulnerability Scoring System* (CVSS)[3].
- CVSS score – číselná hodnota získaná z kalkulačky CVSS[4].

[3]CVSS: `https://nvd.nist.gov/vuln-metrics/cvss`

[4]Kalkulačka CVSS 3.1: `https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator`

- CVSS string – textový řetězec (tzv. vektor) získaný z kalkulačky CVSS.

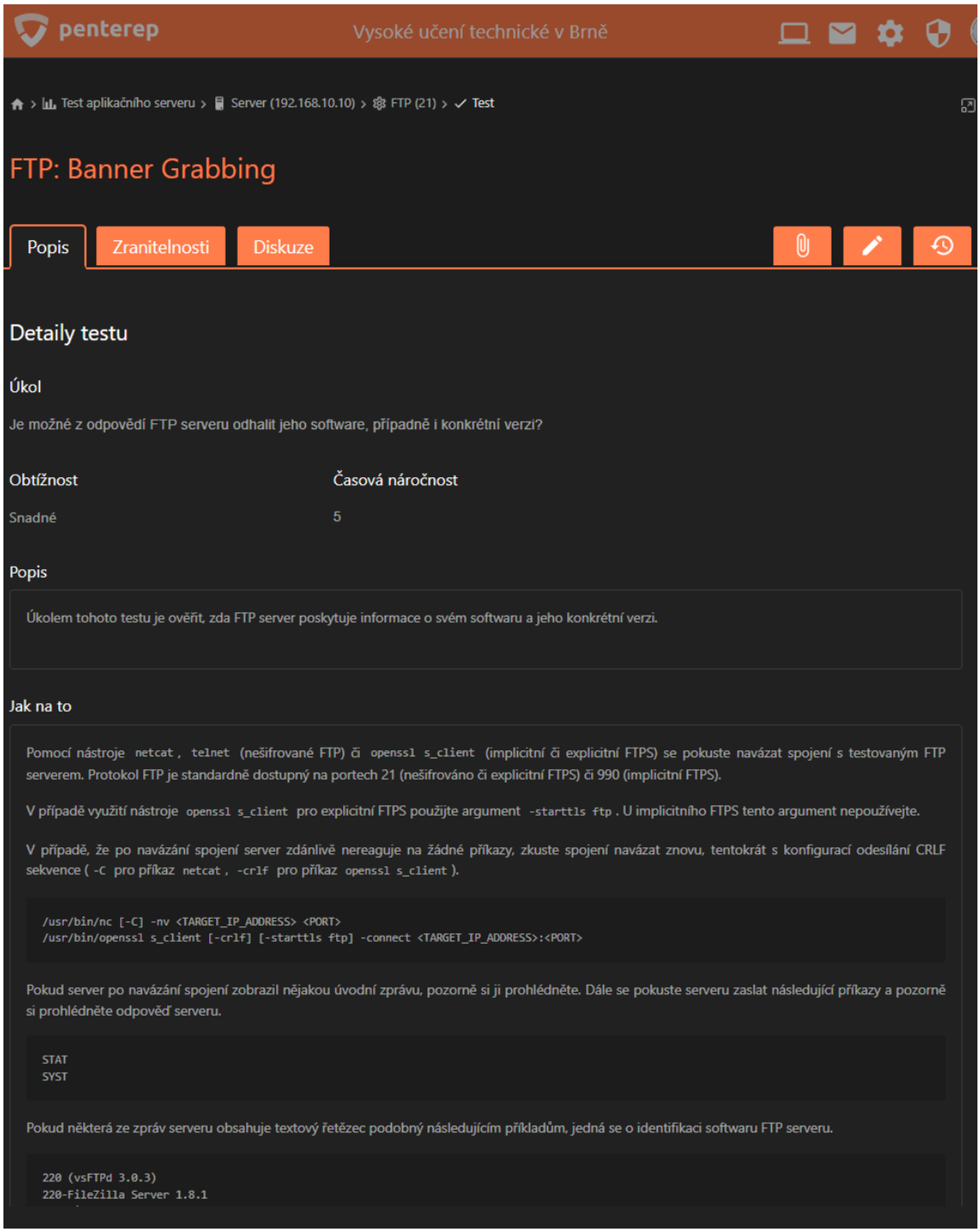
Pro demonstraci obsahu a formátu zranitelností je v následujícím výpisu uvedena obecná zranitelnost bannergrabbing.

Výpis 3.2: Zranitelnost bannergrabbing

```
Název zranitelnosti
-------------------
Odhalení informací o softwaru (a jeho konkrétní verzi)

Popis zranitelnosti
-------------------
Aplikační server prozrazuje informace o svém softwaru, případně i o
    jeho konkrétní verzi.

Příčiny
-------
Aplikační server je nakonfigurován tak, aby umožňoval uživatelům
   zjistit informace o jeho softwaru. Některé aplikační servery
   toto chování mohou mít zabudované a deaktivace může být obtížná.

Projevy
-------
Aplikační server v některé ze svých odpovědí zahrnuje informace o
   svém softwaru, případně i o jeho konkrétní verzi.

Dopady
------
Prozrazení informací o softwaru a jeho konkrétní verzi umožňuje úto
   čníkům efektivněji hledat a identifikovat zranitelnosti aplikačn
   ího serveru.

Náprava / doporučení
--------------------
Doporučujeme aplikační server nakonfigurovat tak, aby informace o
   svém softwaru a jeho konkrétní verzi neposkytoval.

Závažnost
---------
Střední

CVSS score
----------
4.9

CVSS string
```

```
-----------
CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N/E:U/RL:X/RC:C
```

Následující snímek obrazovky demonstruje integraci implementovaných kontrolních seznamů do platformy Penterep.



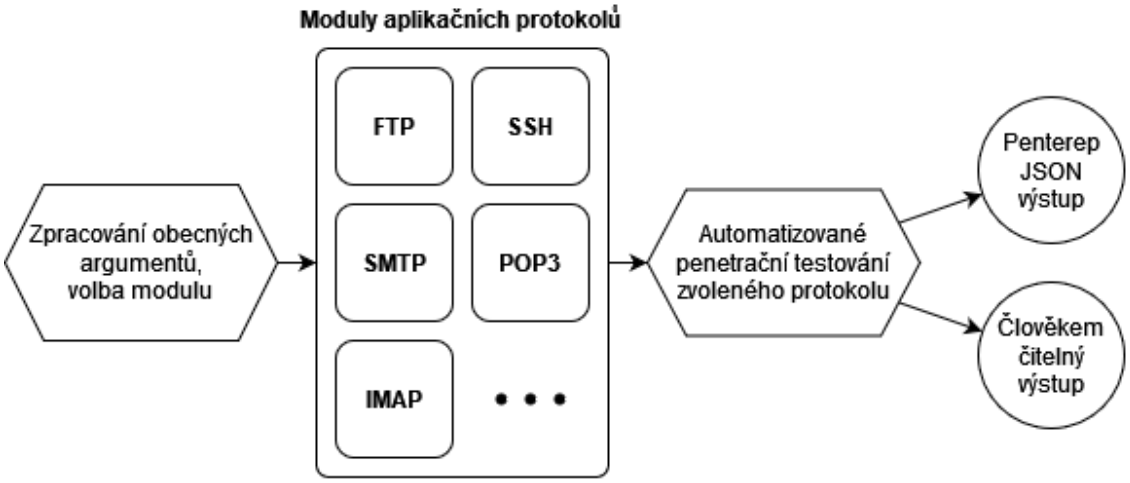
Obr. 3.6: Test bannergrabbing FTP v platformě Penterep

4 Vlastní návrh a implementace nástroje ptapptest

Obsahem této kapitoly je popis návrhu a následné implementace automatizovaného nástroje na penetrační testování aplikačních protokolů s názvem *ptapptest*, který umožňuje automatizaci některých bodů z kontrolních seznamů navržených v sekci 3.1. Na začátku kapitoly je představen návrh nástroje a jeho modulární architektura. Dalším obsahem kapitoly je poté popis tvorby základních částí nástroje a jeho modulů, kde pro každý modul (kromě modulu SMTP) je provedena analýza existujících řešení, návrh modulu a jeho následná implementace. Součástí sekce modulu POP3 je rovněž představení samostatného nástroje *ptntlmauth*, který byl vyvinut v rámci této diplomové práce.

4.1 Návrh nástroje

Nástroj ptapptest byl navržen jako modulární nástroj na automatizované penetrační testování aplikačních protokolů. Pro tvorbu nástroje byla zvolena modulární architektura, kde každý z modulů bude zaměřen na testování konkrétního aplikačního protokolu. V rámci diplomové práce budou implementovány moduly pro testování protokolů FTP, SSH, SMTP, POP3 a IMAP. Modulární návrh nástroje ale dále umožní jeho snadnou rozšiřitelnost i o další moduly pro libovolné jiné aplikační protokoly.



Obr. 4.1: Návrh nástroje ptapptest

Nástroj bude vytvořen v programovacím jazyce Python tak, aby bylo možné jej využít jakožto rozšiřující modul do platformy Penterep. Ovládání nástroje bude

zajištěno skrze textové rozhraní příkazové řádky. Základní část nástroje bude nabízet konfiguraci obecných nastavení, jako například specifikaci formátu výstupu či povolení výpisu ladicích zpráv. Jednotlivé moduly dále budou nabízet podrobnější konfiguraci, jako například specifikaci testovaného cíle (např. IP adresa a číslo portu) a určení testovacích metod, které mají být v rámci běhu nástroje provedeny. Po dokončení běhu modulu bude poskytnut textový výstup upravený do požadovaného výstupního formátu. Podporovanými výstupními formáty budou formát pro čtení člověkem a strojový formát pro další zpracování nástrojem Penterep (specifický *JSON*[1] formát). Pro tento výstupní formát byla společností Penterep Security s.r.o. poskytnuta dokumentace popisující jeho strukturu. Tato struktura je obsažena v následujícím výpisu.

Výpis 4.1: Penterep JSON formát

```
{
    "satid": "",
    "guid": "",
    "status": "",
    "message": "",
    "results": {
        "nodes": [],
        "properties": {},
        "vulnerabilities": []
    }
}
```

Pole `status` určuje, zda byl běh nástroje úspěšný či nikoliv a mělo by obsahovat hodnotu „finished“ nebo „error“. V případě výskytu chyby při běhu nástroje pole `message` obsahuje popis chybové události, která nastala. V případě úspěšného běhu jsou výsledky nástroje uloženy do pole `results`. Pole `properties` slouží pro obecné informace zjištěné při běhu nástroje, jako je například uvítací zpráva serveru. Pole `vulnerabilities` pak slouží pro zranitelnosti, které byly nástrojem odhaleny. Následující výpis obsahuje strukturu, která náleží hodnotám pole `vulnerabilities`.

Výpis 4.2: Formát pole `vulnerabilities`

```
{
    "vulnCode": "",
    "vulnRequest": "",
    "vulnResponse": ""
}
```

Pole `vulnCode` obsahuje název zranitelnosti, který danou zranitelnost jednoznačně identifikuje v rámci platformy Penterep. Pole `vulnRequest` popisuje, jakou

[1] JSON: `https://www.json.org/json-en.html`

žádostí či operací byla daná zranitelnost odhalena a pole `vulnResponse` dále popisuje chování testovaného cíle, které prokázalo výskyt dané zranitelnosti.

Vývoji každého modulu bude předcházet analýza již existujících dostupných řešení, na jejímž základě bude modul navržen a implementován. Během této analýzy budou zkoumána specifika existujících nástrojů, včetně jejich licence, programovacího jazyka a stavu vývoje (aktivní či neaktivní). Pro vývoj nástroje ptapptest bude vyžadováno, aby jím využité externí nástroje byly vydávány pod licencí, která nenaruší možnost komerčního využití platformy Penterep a zároveň nebude vyžadovat zveřejnění celého zdrojového kódu platformy formou open source. Z tohoto důvodu nebudou do analýzy dostupných řešení zahrnuty nástroje *Nmap*[2] (vlastní licence NPSL[3]) a *Hydra*[4] (licence AGPL). Rovněž bude vyžadováno aby jím využité externí nástroje byly napsány v programovacím jazyce Python, a to z toho důvodu, aby nástroj nekladl zbytečné nároky na své produkční prostředí. Z tohoto důvodu nebude do analýzy zahrnut nástroj *Metasploit Framework* [5] (programovací jazyk Ruby). Posledním požadavkem pro vývoj nástroje ptapptest bude limitace využití externích nástrojů za účelem implementace takových funkcionalit, které by bylo možné bez větších obtíží realizovat vlastní implementací. Tento požadavek cílí opět na snížení nároků na produkční prostředí, ale rovněž také na maximalizaci nezávislosti nástroje a kontroly nad jeho chováním. Z tohoto důvodu nebude do analýzy zahrnut nástroj *NetExec* [6] (nástupce nástroje *CrackMapExec* [7]).

4.2 Implementace základních částí nástroje

Nástroj ptapptest je modulárním nástrojem implementovaným v jazyce Python verze 3.12 sloužícím pro penetrační testování aplikačních protokolů. Penetrační testování je realizováno jednotlivými moduly nástroje, kde každý z modulů slouží pro testování právě jednoho aplikačního protokolu. Všechny moduly sdílí společnou strukturu danou bázovou třídou `BaseModule` uvedenou ve výpisu 4.3. Tato třída obsahuje několik abstraktních metod (`module_args`, `__init__` a `run`), jejichž implementace musí být obsaženy v odvozených třídách jednotlivých modulů. Rovněž se zde ale nachází 2 standardní metody `ptdebug` a `ptprint`, které sjednocují implementaci textového výstupu všech modulů. Pro tisk požadovaného textu na standardní výstup nástroje využívají obě metody modul `ptprinthelper` z knihovny *ptlibs*[8],

[2]Nmap: `https://nmap.org/`
[3]Licence NSPL nástroje Nmap: `https://nmap.org/npsl/`
[4]Hydra: `https://github.com/vanhauser-thc/thc-hydra`
[5]Metasploit Framework: `https://github.com/rapid7/metasploit-framework`
[6]NetExec: `https://github.com/Pennyw0rth/NetExec`
[7]CrackMapExec: `https://github.com/byt3bl33d3r/CrackMapExec`
[8]ptlibs: `https://pypi.org/project/ptlibs/`

která je součástí platformy Penterep a obsahuje pomocné funkce sloužící pro vývoj automatizovaných nástrojů platformy.

Výpis 4.3: Bázová třídá BaseModule

```
class BaseModule(ABC):
    @staticmethod
    @abstractmethod
    def module_args() -> BaseArgs:
        return BaseArgs()

    @abstractmethod
    def __init__(self, args: BaseArgs, ptjsonlib: PtJsonLib):
        self.args = args
        self.ptjsonlib = ptjsonlib
        raise NotImplementedError

    @abstractmethod
    def run(self) -> None:
        raise NotImplementedError

    def ptdebug(self, string: str, out: Out = Out.TEXT, title: bool = False, end: str = "\n"):
        if not self.args.debug:
            return

        if title:
            colortext = True
            category = Out.TITLE.value
        else:
            colortext = False
            category = out.value

        ptprinthelper.ptprint(string, category, True, flush=True, colortext=colortext, end=end)

    def ptprint(
        self,
        string: str,
        out: Out = Out.TEXT,
        title: bool = False,
        end: str = "\n",
        json: bool = False,
    ):

        if json and not self.args.json:
            # trying to print JSON in normal mode
            return
```

```
42          elif not json and self.args.json:
43              # trying to print normal text in JSON mode
44              return
45
46          if title:
47              colortext = True
48              category = Out.TITLE.value
49          else:
50              colortext = False
51              category = out.value
52
53          ptprinthelper.ptprint(string, category, True, flush=True,
    colortext=colortext, end=end)
```

Zmíněné metody `ptdebug` a `ptprint` usnadňují kontrolu nad výstupem nástroje v závislosti na jeho konfiguraci. Metoda `ptdebug` slouží pouze pro výpis ladicích zpráv a provádí výpis požadovaného textu pouze v případě, kdy byl nástroj spuštěn s přepínačem `--debug`. Metoda `ptprint` naopak slouží pro výpis stěžejních informací, který provádí v závislosti na přepínači `--json`. Bez tohoto přepínače dochází touto metodou k výpisu výstupních zpráv určených pro čtení člověkem. Při použití přepínače `--json` dochází k potlačení těchto výstupů a k jejich výpisu dojde pouze tehdy, pokud je metoda volána s argumentem `json=True`, čímž je umožněno metodu využít i pro zobrazení závěrečného výsledku ve formátu JSON.

Přepínače `--debug` a `--json` patří společně s přepínačem `--version` k obecným parametrům nástroje. Ke zpracování veškeré konfigurace nástroje, tedy nastavení již zmíněných obecných parametrů, ale i podrobnějších parametrů jednotlivých modulů, dochází v hlavním souboru nástroje `ptapptest.py`. Zkrácený kód tohoto zpracování a následného spuštění zvoleného modulu nástroje se nachází ve výpisu 4.4. Veškerá konfigurace je definována i zpracovávána pomocí standardního modulu `argparse`[9]. Ke zpracování argumentů nástroje dochází na řádcích 30–31. Po konfiguraci nástroje dojde ke spuštění metody `run` (řádky 16–21), která voláním `module.run()` spustí zvolený modul na základě pozičního argumentu `module` a voláním `module.output()` vytiskne jeho výsledek. Možné hodnoty argumentu `module` jsou definovány globální proměnnou `MODULES` typu `dict`, která mapuje názvy jednotlivých modulů na jejich odpovídající třídy. Všechny tyto třídy jsou odvozeny od již zmíněné bázové třídy `BaseModule`.

Výpis 4.4: Zpracování argumentů a spuštění zvoleného modulu

```
1 from ptlibs import ptprinthelper, ptjsonlib
2 ...
3 from .modules._base import BaseArgs
```

[9] argparse: https://docs.python.org/3/library/argparse.html

```
4  from .modules.ftp import FTP
5  from .modules.ssh import SSH
6  from .modules.smtp import SMTP
7  from .modules.pop3 import POP3
8  from .modules.imap import IMAP
9
10 MODULES = {"ftp": FTP, "ssh": SSH, "smtp": SMTP,"pop3": POP3, "imap
       ": IMAP}
11
12 class PtAppTest:
13     def __init__(self, args: BaseArgs) -> None:
14         self.args = args
15
16     def run(self) -> None:
17         ptjson = ptjsonlib.PtJsonLib()
18
19         module = MODULES[self.args.module](self.args, ptjson)
20         module.run()
21         module.output()
22
23 def parse_args() -> BaseArgs:
24     parser = argparse.ArgumentParser(add_help=True)
25 ...
26     subparsers = parser.add_subparsers(required=True, dest="module"
       )
27     for name, module in MODULES.items():
28         module.module_args().add_subparser(name, subparsers)
29 ...
30     args = parser.parse_args(namespace=BaseArgs)
31     args = parser.parse_args(namespace=MODULES[args.module].
       module_args())
32 ...
```

Nezávislé konfigurace každého z modulů bylo dosaženo pomocí tzv. subparsers. Pomocí této funkcionality modulu `argparse` je v každém modulu definována vlastní sada parametrů, k jejichž zpracování dojde právě tehdy, když je nástroj spuštěn s pozičním argumentem odpovídajícím danému modulu. Každý modul definuje vlastní parametry pomocí samostatné třídy parametrů odvozené od bázové třídy `BaseArgs` uvedené ve výpisu 4.5. V rámci této bázové třídy jsou definovány již zmíněné obecné parametry `json`, `debug` a `module` (řádky 2–4), ale také abstraktní metoda `add_subparser` (řádky 8–17), která slouží pro konfiguraci parametrů daného modulu pomocí již zmíněné funkcionality subparsers. Tato metoda je volána na řádcích 26–28 předchozího výpisu 4.4. Volání této metody je umožněno předcházejícím voláním statické metody `module_args()`, která vytvoří a navrátí instanci třídy parametrů daného modulu.

Výpis 4.5: Bázová třídá BaseArgs

```
class BaseArgs(argparse.Namespace):
    json: bool
    debug: bool
    module: str

    @abstractmethod
    def add_subparser(self, name: str, subparsers: argparse._SubParsersAction) -> None:
        """Subparsers example"""
        modname = __name__.split(".")[-1]
        parser = subparsers.add_parser(modname, add_help=True)
...
        parser.add_argument(
            "target", type=valid_target, help="IP[:PORT] ..."
        )
        actions = parser.add_argument_group("actions")
        actions.add_argument("--banner", action="store_true", help="get the service banner")
```

Následující výpis zobrazuje nápovědu nástroje ptapptest. V textu nápovědy lze pozorovat obecné parametry nástroje (sekce „options“) a aktuálně dostupné moduly (sekce „positional arguments“).

Výpis 4.6: Nápověda nástroje ptapptest

```
$ ptapptest -h
usage: ptapptest [-h] [-v] [-j] [--debug] {ftp,ssh,smtp,pop3,imap}
    ...

positional arguments:
  {ftp,ssh,smtp,pop3,imap}

options:
  -h, --help            show this help message and exit
  -v, --version         print version
  -j, --json            use Penterep JSON output format
  --debug               enable debug messages
```

4.3 Vývoj modulu FTP

Na základě kontrolních bodů navržených v sekci 3.1.1 byly pro modul FTP navrženy následující funkcionality:

- kontrola konfigurace protokolů SSL/TLS,
- bannergrabbing,

- anonymní autentizace,
- slovníkový útok,
- kontrola přístupových oprávnění,
- útok FTP bounce.

Na základě navržených funkcionalit byla provedena analýza již existujících programových řešení, s cílem nalézt takové nástroje, které by bylo možné využít pro zajištění některých z požadovaných funkcionalit. Během této analýzy byly nalezeny nástroje *FTP-Security-Scanner* (FSS)[10], *sslscan*[11], *tls-scan*[12] a *sslyze*[13]. Tabulka 4.1 shrnuje funkcionality a další specifika těchto nástrojů.

Tab. 4.1: Specifika nalezených nástrojů protokolu FTP

Specifikace	**FSS**	**sslscan**	**tls-scan**	**sslyze**
Kontrola SSL/TLS	✗	✓	✓	✓
Bannergrabbing	částečná podpora	✗	✗	✗
Anonymní autentizace	✓	✗	✗	✗
Slovníkový útok	✗	✗	✗	✗
Kontrola oprávnění	✓	✗	✗	✗
FTP bounce	✗	✗	✗	✗
Aktivní vývoj	✗	✓	✓	✓
Jazyk	Python	Python	C	Python
Licence	MIT	GPL	BSD	AGPL

Nástroj FTP-Security-Scanner podporuje některé požadované funkcionality a je distribuován s přijatelnou licencí MIT, ovšem již není aktivně vyvíjen, a tedy jeho využití v rámci nástroje ptapptest není žádoucí. Všechny 3 zbývající nástroje umožňují kontrolu konfigurace protokolů SSL/TLS, avšak žádný z těchto nástrojů není vhodným pro využití v rámci nástroje ptapptest, a to z důvodu kolize se stanovenými požadavky pro jeho vývoj. Nástroje sslscan a sslyze jsou distribuovány s nepřijatelnými licencemi GPL a AGPL. Nástroj tls-scan je sice distribuován s přijatelnou licencí BSD, ovšem je napsán v programovacím jazyce C, a tedy neodpovídá požadavku na programovací jazyk Python.

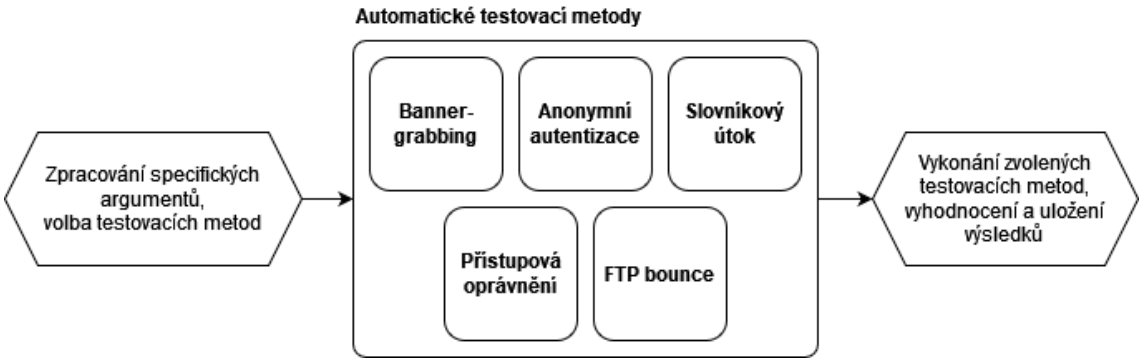
Na základě analýzy dostupných řešení bylo rozhodnuto, že modul FTP bude vyvinut bez využití již existujících externích nástrojů. Rovněž bylo zjištěno, že žádný

[10]FTP-Security-Scanner: `https://github.com/QuentinCG/FTP-Security-Scanner`
[11]sslscan: `https://github.com/rbsec/sslscan`
[12]tls-scan: `https://github.com/prbinu/tls-scan`
[13]sslyze: `https://github.com/nabla-c0d3/sslyze`

z nalezených nástrojů na kontrolu protokolů SSL/TLS nesplňuje požadavky nástroje ptapptest, a tedy nemůže být použit při jeho vývoji. Z tohoto důvodu tyto nástroje již nebudou zahrnovány do analýzy dostupných řešení u dalších modulů (SMTP, POP3 a IMAP). Rovněž bylo rozhodnuto, že časové nároky potřebné pro realizaci vlastní implementace kontroly protokolů SSL/TLS, která by dosahovala požadované úrovně kvality a efektivity, jsou příliš vysoké a přesahují rámec této diplomové práce. Z tohoto důvodu byla kontrola protokolů SSL/TLS vyjmuta ze seznamu navržených funkcionalit pro modul FTP. Rovněž bylo rozhodnuto, že tato funkcionalita nebude zahrnuta ani v dalších modulech nástroje (SMTP, POP3 a IMAP).

4.3.1 Návrh modulu

Na základě analýzy existujících nástrojů byl navržen modul na penetrační testování protokolu FTP (obrázek 4.2). Veškerá implementace modulu bude realizována vlastním řešením s využitím modulů obsažených ve standardní knihovně jazyka Python. Pro realizaci komunikace protokolu FTP bude využit standardní modul `ftplib`[14]



Obr. 4.2: Návrh modulu FTP

4.3.2 Implementace modulu

Implementace modulu FTP se nachází v souboru `modules/ftp.py` a jejími stěžejními částmi jsou hlavní třída modulu `FTP`, třída parametrů `FTPArgs` a datová třída `FTPResults` sloužící pro uložení výsledků běhu modulu. Definice tříd `FTPArgs` a `FTPResults` jsou uvedeny v následujícím výpisu.

Výpis 4.7: Třídy `FTPArgs` a `FTPResults`

```
1 @dataclass
2 class FTPResults:
3     info: InfoResult | None = None
4     access: AccessCheckResult | None = None
```

[14] `ftplib`: `https://docs.python.org/3/library/ftplib.html`

```
    anonymous: bool | None = None
    creds: set[Creds] | None = None
    bounce: BounceResult | None = None

class FTPArgs(ArgsWithBruteforce):
    target: Target
    active: bool
    tls: bool
    starttls: bool
    anonymous: bool
    info: bool
    access: bool
    access_list: bool
    bounce: Target | None
    bounce_file: str | None

    def add_subparser(self, name: str, subparsers) -> None:
...
```

Hlavní metodou modulu je metoda `run`, ve které dochází ke spouštění jednotlivých testovacích metod. Tyto metody jsou spouštěny v závislosti na konfiguraci parametrů definovaných v již zmíněné třídě `FTPArgs` a výsledky těchto metod jsou ukládány do instance rovněž již zmíněné třídy `FTPResults`. Následující výpis obsahuje kód metody `run`.

Výpis 4.8: Hlavní metoda `run`

```
def run(self) -> None:
    self.results = FTPResults()
    self.ftp = self.connect()

    if self.args.anonymous:
        self.results.anonymous = self.anonymous()

    if self.do_brute:
        self.results.creds = simple_bruteforce(
            self._try_login,
            self.args.user,
            self.args.users_file,
            self.args.passw,
            self.args.passw_file,
            self.args.spray,
            self.args.threads,
        )

    if self.args.info:
        self.results.info = self.info()
```

```
21
22      if self.args.access:
23          self.results.access = self.access_check()
24
25          if self.args.bounce:
26              self.results.bounce = self.bounce()
```

Metoda `connect` zajišťuje jednotný způsob vytvoření nového spojení s FTP serverem. Jsou zde zohledněny argumenty modulu specifikující IP adresu a síťový port serveru, ale také argumenty týkající se explicitního či implicitního zabezpečení spojení pomocí protokolů SSL/TLS.

Metoda `info` implementuje funkcionalitu bannergrabbingu zaznamenáním úvodní zprávy obdržené od serveru a následným provedením příkazů `SYST` a `STAT`, u nichž rovněž zaznamenává odpovědi serveru. Výsledkem metody je instance datové třídy `InfoResult` obsahující všechny 3 zaznamenané zprávy.

Metoda `anonymous` provádí anonymní autentizaci při které dochází k pokusu o přihlášení pomocí výchozích přihlašovacích údajů modulu `ftplib`, jimiž jsou jméno „anonymous“ a heslo „anonymous@“. Výsledkem metody je booleovská hodnota vyjadřující úspěch či neúspěch pokusu o přihlášení.

Slovníkový útok je v modulu implementován pomocí funkce `simple_bruteforce` a metody `_try_login`. Funkce `simple_bruteforce` je pomocnou funkcí využívanou i ostatními moduly nástroje, která umožňuje vícevláknové zpracování slovníkového útoku na základě poskytnutých přihlašovacích údajů a specifikované přihlašovací metody. Vícevláknové provedení je realizováno pomocí modulu `ptthreads` z knihovny ptlibs. Návratovou hodnotou této funkce je množina datových struktur `Creds`, které reprezentují nalezené validní kombinace přihlašovacích jmen a hesel.

Metoda `access_check` umožňuje identifikovat přístupová oprávnění, která jsou obdržena při autentizaci pomocí validních přihlašovacích údajů. Průběh metody se skládá z několika kroků, kterými jsou: pokus o výpis obsahu kořenového adresáře, pokus o zápis souboru do některého z dostupných adresářů, pokus o přečtení předešle zapsaného souboru a pokus o smazání předešle zapsaného souboru. Návratovou hodnotou je datová třída `AccessCheckResult`, jejíž obsah reflektuje průběh testovací metody a dosažené výsledky.

Metoda `bounce` je poslední implementovanou testovací metodou a umožňuje provedení útoku FTP bounce. Pro ukázku kódu modulu jsou v následujícím výpisu uvedeny fragmenty této metody. Na začátku metody jsou zvoleny vhodné přihlašovací údaje, pro jejichž získání či ověření je nutné využít předešle představenou metodu `anonymous` či `simple_bruteforce`. Dále je proveden pokus o provedení konfigurace útoku bounce (řádky 7–8), která spočívá v nastavení IP adresy a síťového portu datového spojení protokolu pomocí příkazu `PORT` či `EPRT`. V případě úspěšného provedení

konfigurace metoda pokračuje buď pokusem o provedení textového požadavku na cílovou službu (řádky 11–26) nebo pokusem o pouhé zjištění, zda je cílová síťová služba dostupná (řádky 28–31). V případě pokusu o provedení textového požadavku dochází postupně k nahrání souboru obsahujícího daný textový požadavek na FTP server, obnovení konfigurace útoku bounce a opětovnému stažení daného souboru. Poslední zmíněný krok díky konfiguraci útoku bounce způsobí odeslání vyžádaného souboru na cílovou síťovou službu. Po provedení tohoto kroku je proveden pokus o smazání předešle nahraného souboru z FTP serveru. V případě pouhé identifikace dostupnosti síťové služby je provedena pouze žádost o výpis adresáře FTP serveru, která opět díky konfiguraci útoku bounce vyvolá spojení FTP serveru s cílovou síťovou službou. Výstupem metody `bounce` je instance datové třídy `BounceResult`, jejíž obsah popisuje průběh a výsledek útoku FTP bounce.

Výpis 4.9: Zkrácený obsah metody `bounce`

```
1  def bounce(self) -> BounceResult:
2  ...
3          ftp = self.connect()
4          ftp.login(creds.user, creds.passw)
5
6          # Bounce setup attempt
7          if not self._bounce_setup(ftp, self.args.bounce):
8              return BounceResult(self.args.bounce, creds, False,
       None, None)
9
10         if self.args.bounce_file and write_path is not None:
11             # Full bounced request
12 ...
13                 # Upload request file onto FTP server
14                 with open(self.args.bounce_file, "rb") as f:
15                     p = ftp.storbinary("STOR " + filename, f)
16 ...
17                 # Refresh bounce setup after STOR
18                 self._bounce_setup(ftp, self.args.bounce)
19
20                 # Upload request to bounce target
21                 ftp.sendcmd("RETR " + filename)
22 ...
23                     # Cleanup the uploaded request file
24 ...
25                         ftp.delete(filename)
26 ...
27         else:
28             # Just port scan
29 ...
```

```
                    ftp.sendcmd("LIST")
...
```

Po dokončení běhu hlavní metody `run` je možné pomocí metody `output` vytisknout na standardní výstup nástroje výsledky běhu modulu. Během této metody dochází k inspekci hodnot nacházejících se v instanci datové třídy `FTPResults`, která byla postupně plněna výsledky běhu nástroje v metodě `run`. U každého pole třídy pak dochází k interpretaci jeho hodnot a následnému volání již zmíněné metody `ptprint()`, která dle konfigurace parametru `--json` buď vypíše požadovaný výsledek na standardní výstup, či nikoliv. V metodě `output` je rovněž připravován výstup ve formátu JSON, a to s využitím modulu `ptjsonlib` z knihovny ptlibs. Pro demonstraci užití metody `ptprint` a zaznamenání zranitelnosti pomocí modulu `ptjsonlib` je v následujícím výpisu uveden úryvek kódu vykonávající interpretaci výsledku kontroly anonymní autentizace. Ve výpisu se nachází volání metody `ptprint()` i přidání nalezené zranitelnosti do dříve popsané Penterep JSON struktury.

Výpis 4.10: Úryvek metody `output`

```
if (anon := self.results.anonymous) is not None:
    self.ptprint(f"Anonymous authentication: {anon}", title=True)
    if anon:
...
        self.ptjsonlib.add_vulnerability(
            VULNS.Anonymous.value, "anonymous login", response_str
        )
```

Následující výpisy demonstrují standardní výstup modulu při jeho úspěšném běhu, a to jak ve formátu pro čtení člověkem (výpis 4.11), tak v JSON formátu pro platformu Penterep (výpis 4.12).

Výpis 4.11: Výstup pro čtení člověkem

```
 ____            _                          _____          _
|  _ \ ___ _ __ | |_ ___ _ __ ___ _ __ __   |_   _|__  ___ | |___
| |_) / _ \ '_ \| __/ _ \ '__/ _ \ '_ \    | |/ _ \ / _ \| / __|
|  __/  __/ | | | ||  __/ | |  __/ |_) |   | | (_) | (_) | \__ \
|_|   \___|_| |_|\__\___|_|  \___| .__/    |_|\___/ \___/|_|___/
                                 |_|           ptapptest v1.0.0
                                       https://www.penterep.com


[*] Server information
[i] Banner:
220 (vsFTPd 3.0.3)
[i] SYST command:
None
[i] STAT command:
```

```
211-FTP server status:
     Connected to 192.168.113.1
     Logged in as ftp
     TYPE: ASCII
     No session bandwidth limit
     Session timeout in seconds is 300
     Control connection is plain text
     Data connections will be plain text
     At session startup, client count was 1
     vsFTPd 3.0.3 - secure, fast, stable
211 End of status
[*] Anonymous authentication: True
(Directory listing: True, Write: writez/YKGPQAFFQRBZMUO.txt, Read:
   None, Delete: None)
[*] Directory listing
-rw-r--r--    1 0        0          35 May 05 10:50 http.txt
drwxr-xr-x    2 0        0        4096 May 03 17:57 spaces in name
drwxr-xr-x    2 0        0        4096 May 03 17:55 testdir
drwxr-xr-x    2 1004     0        4096 May 09 11:25 writemaster
drwxrwxrwx    2 65534    65534    4096 May 14 08:07 writez
```

Výpis 4.12: Výstup v Penterep JSON formátu

```
{
    "satid": "",
    "guid": "",
    "status": "finished",
    "message": "",
    "results": {
        "nodes": [],
        "properties": {
            "banner": "220 (vsFTPd 3.0.3)",
            "systCommand": null,
            "statCommand": "211-FTP server status:\n     Connected
   to 192.168.113.1\n...",
            "directoryListing": "-rw-r--r--    1 0        0          3
   5 May 05 10:50 http.txt\ndrwxr-xr-x    2 0        0        4096 May
    03 17:57 spaces in name\n..."
        },
        "vulnerabilities": [
            {
                "vulnCode": "PTV-FTP-ANONYMOUS",
                "vulnRequest": "anonymous login",
                "vulnResponse": "(Directory listing: True, Write:
   writez/YXEHBMXKHJZSRZL.txt, Read: None, Delete: None)"
            }
        ]
```

```
    }
}
```

4.4 Vývoj modulu SSH

Z návrhu kontrolních bodů v sekci 3.1.2 byly pro modul SSH navrženy následující funkcionality:

- kontrola podporovaných kryptografických algoritmů,
- ověření, zda SSH server nepoužívá veřejně známý SSH klíč,
- identifikace softwaru a verze serveru (bannergrabbing),
- identifikace veřejně známých zranitelností,
- zjištění podporovaných autentizačních metod,
- slovníkový útok se jmény a hesly,
- slovníkový útok se soukromými SSH klíči.

Na základě těchto funkcionalit byla provedena analýza existujících nástrojů, během které byly hledány takové nástroje, které by bylo možné využít pro zajištění některých z požadovaných funkcionalit. Během této analýzy byl nalezen pouze nástroj *ssh-audit*[15]. Následující tabulka shrnuje specifika tohoto nástroje.

Tab. 4.2: Specifika nástroje ssh-audit

Specifikace	**ssh-audit**
Kryptografické algoritmy	✓
Veřejně známý SSH klíč	✗
Bannergrabbing	✓
Veřejně známé zranitelnosti	✓
Autentizační metody	✗
Slovníkový útok (jméno, heslo)	✗
Slovníkový útok (SSH klíč)	✗
Aktivní vývoj	✓
Jazyk	Python
Licence	MIT

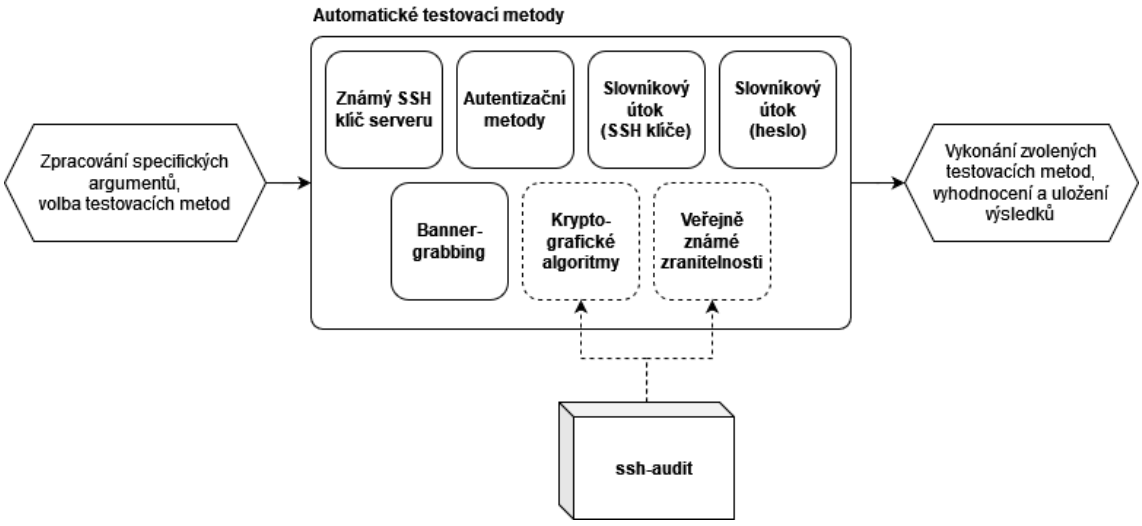
Nástroj ssh-audit umožňuje kontrolu kryptografických algoritmů, bannergrabbing a identifikaci veřejně známých zranitelností softwaru serveru. Nástroj je rovněž aktivně vyvíjen, distribuován s vyhovující licencí MIT a je napsán v jazyce Python. Tyto charakteristiky jej činí vhodným nástrojem pro integraci do nástroje

[15]ssh-audit: `https://github.com/jtesta/ssh-audit`

ptapptest, a proto bude v modulu SSH využit pro zajištění některých ze zmíněných funkcionalit.

4.4.1 Návrh modulu

Na základě analýzy existujících nástrojů byl navržen modul na penetrační testování protokolu SSH (obrázek 4.3). Modul bude využívat existující nástroj ssh-audit pro kontrolu podporovaných kryptografických algoritmů a pro identifikaci veřejně známých zranitelností softwaru serveru. Zbylé mechanismy budou řešeny vlastní implementací s využitím vhodných knihoven. Pro realizaci vlastní komunikace protokolu SSH bude využita primárně knihovna *Paramiko*[16].



Obr. 4.3: Návrh modulu SSH

4.4.2 Implementace modulu

Implementace modulu SSH se nachází v souboru `modules/ssh.py` a obsahuje hlavní třídu modulu `SSH`, třídu parametrů `SSHArgs`, datovou třídu `SSHResults` sloužící pro uložení výsledků běhu modulu a několik dalších datových tříd. Definice tříd `SSHArgs` a `SSHResults` jsou uvedeny v následujícím výpisu.

Výpis 4.13: Třídy `SSHArgs` a `SSHResults`

```
@dataclass
class SSHResults:
    info: InfoResult | None = None
    ssh_audit: SSHAuditResult | None = None
    bad_pubkey: BadPubkeyResult | None = None
    bad_authkeys: list[str] | None = None
```

[16]Paramiko: `https://www.paramiko.org/`

```
    brute: BruteResult | None = None

class SSHArgs(ArgsWithBruteforce):
    target: Target
    info: bool
    auth_methods: bool
    ssh_audit: bool
    bad_pubkeys: str | None
    bad_authkeys: str | None
    privkeys: str | None

    def add_subparser(self, name: str, subparsers) -> None:
...
```

Stejně jako u předchozího modulu dochází v hlavní metodě `run` ke spouštění jednotlivých testovacích metod na základě konfigurace parametrů modulu. Každá z testovacích metod navrací výsledek, který je uložen do instance třídy `SSHResults`. Následující výpis obsahuje kód metody `run`.

Výpis 4.14: Hlavní metoda `run`

```
def run(self) -> None:
    self.results = SSHResults()

    if self.args.info:
        self.results.info = self.info(self.args.auth_methods)

        # Pubkey check requires info (that retrieves the key)
        if self.args.bad_pubkeys:
            self.results.bad_pubkey = self.bad_pubkey(
                self.args.bad_pubkeys, self.results.info.host_key
            )

    if self.args.ssh_audit:
        self.results.ssh_audit = self.run_ssh_audit()

    if self.args.bad_authkeys:
        self.results.bad_authkeys = self.bad_authkeys(self.args.
    bad_authkeys)

    if self.do_brute:
        self.results.brute = self.bruteforce()
```

Metoda `info` zajišťuje funkcionalitu bannergrabbingu připojením k serveru a uložením jeho úvodní zprávy. Následně je zde pomocí knihovny Paramiko zahájena SSH

spojení s SSH serverem, během kterého je zaznamenán jeho veřejný SSH klíč a dostupné autentizační metody. Všechny tyto hodnoty jsou jakožto výsledek metody navráceny formou instance datové třídy `InfoResult`[17].

Metoda `bad_pubkey` zkoumá veřejný SSH klíč serveru získaný metodou `info` a porovnává jej s množinou veřejně známých SSH klíčů ve snaze nalézt shodu s některým z těchto klíčů. Množina klíčů je specifikována konfigurací modulu, během které je zadána cesta ke složce obsahující soubory s porovnávanými SSH klíči. Výsledkem metody je instance datové třídy `BadPubkeyResult`, která obsahuje informace o případné nalezené shodě.

Metoda `bad_authkeys` implementuje pokusy o přihlášení pomocí soukromých SSH klíčů poskytnutých v takovém formátu, který lze pozorovat na GitHub repozitáři ssh-badkeys[18] společnosti Rapid7[19]. Cílem této metody je umožnění přímého využití zmíněného repozitáře pro penetrační testování cílového SSH serveru. Výsledkem metody je seznam soukromých klíčů, pomocí nichž se podařilo úspěšně přihlásit k testovanému serveru.

Metoda `bruteforce` umožňuje vykonávat pokusy o přihlášení s využitím slovníkového útoku, a to s využitím jak kombinací uživatelských jmen a hesel, tak kombinací uživatelských jmen a soukromých SSH klíčů. U varianty s SSH klíči je rovněž možné využít SSH klíče chráněné heslem (při specifikaci správného hesla).

Metoda `run_ssh_audit` je poslední z implementovaných testovacích metod modulu SSH a zajišťuje integraci externího nástroje ssh-audit do tohoto modulu. Pro ukázku kódu modulu je její zkrácená verze uvedena v následujícím výpisu. V metodě je nejprve provedena základní konfigurace nástroje ssh–audit (řádky 2–4) a následně je spuštěna jeho testovací metoda `audit` (řádek 7). Po skončení této metody je pomocí standardního modulu `json`[20] zpracován výstup nástroje ve formátu JSON do datové struktury typu `dict` (řádky 9–10). Z této struktury jsou následně extrahovány všechny úpravy konfigurace kryptografických protokolů, které nástroj doporučuje provést (řádky 12–27) a rovněž všechny veřejně známé zranitelnosti, které nástroj identifikoval (řádky 29–33). Jakožto výsledek metody je navrácena instance datové třídy `SSHAuditResult` obsahující předešle extrahované nálezy nástroje.

Výpis 4.15: Zkrácený obsah metody `run_ssh_audit`

```
1 def run_ssh_audit(self) -> SSHAuditResult:
2     out = ssh_audit.OutputBuffer()
```

[17] Datová třída s názvem `InfoResult` byla uvedena již u modulu FTP, ovšem nejedná se o stejné třídy, pouze o třídy se shodnými jmény. Stejně tak je tomu i u modulů SMTP, POP3 a IMAP, které rovněž implementují vlastní třídy s názvem `InfoResult`.

[18] Repozitář ssh-badkeys: `https://github.com/rapid7/ssh-badkeys/tree/master/authorized`

[19] Rapid7: `https://www.rapid7.com/`

[20] `json`: `https://docs.python.org/3/library/json.html`

```
3       aconf = ssh_audit.AuditConf(self.args.target.ip, self.args.
    target.port)
4       aconf.json = True
5
6       try:
7           status = ssh_audit.audit(out, aconf)
8   ...
9           buf = out.get_buffer()
10          bufj = json.loads(buf)
11  ...
12          findings: list[CryptoFinding] = []
13          recommendations = bufj.get("recommendations", None)
14          if recommendations is not None:
15              # {"critical": {}, "warning": {}, ...}
16              for level, actions in recommendations.items():
17                  # "critical": {"del": {}, "add": {}, ...}
18                  for action, categories in actions.items():
19                      # "del": {"key": [], "enc": [], ...}
20                      for category, details in categories.items():
21                          # "key": [{"name": "", "notes": ""}, {"name
    ": "", "notes": ""}, ...]
22                          for detail in details:
23                              # {"name": "", "notes": ""}
24                              name = detail["name"]
25                              notes = detail["notes"]
26
27                              findings.append(CryptoFinding(level,
    action, category, name, notes))
28
29          cves: list[CVE] = []
30          cves_ = bufj.get("cves", None)
31          if cves_ is not None:
32              for cve in cves_:
33                  cves.append(CVE(cve["name"], cve["description"],
    float(cve["cvssv2"])))
34
35          return SSHAuditResult(None, findings, cves)
36      except SystemExit as e:
37          return SSHAuditResult(e.code, [], [])
```

Po skončení hlavní metody `run` jsou stejně jako u předchozího modulu metodou `output` vytisknuty na standardní výstup nástroje výsledky běhu modulu. Během této metody je opět zajištěn jak výstup pro čtení člověkem, tak výstup ve formátu JSON pro platformu Penterep. Pro demonstraci zaznamenání obecných informací o serveru pomocí modulu `ptjsonlib` je v následujícím výpisu uveden úryvek kódu interpretující výsledek metody `info`.

Výpis 4.16: Úryvek metody `output`

```
properties = self.ptjsonlib.json_object["results"]["properties"]

# Server information
if info := self.results.info:
    self.ptprint("Server information", title=True)

    self.ptprint(f"Banner:", Out.INFO)
    self.ptprint(f"{info.banner}")
    properties["banner"] = info.banner

    self.ptprint(f"Host key:", Out.INFO)
    self.ptprint(info.host_key)
    properties["hostKey"] = info.host_key

    if info.auth_methods is not None:
        self.ptprint(f"Authentication methods:", Out.INFO)
        self.ptprint(", ".join(info.auth_methods))
        properties["authMethods"] = info.auth_methods
```

Následující výpisy obsahují zkrácený standardní výstup modulu při jeho úspěšném běhu, a to jak ve formátu pro čtení člověkem (výpis 4.17), tak v JSON formátu pro platformu Penterep (výpis 4.18).

Výpis 4.17: Výstup pro čtení člověkem

```
 ____            _                        _____          _
|  _ \ ___ _ __ | |_ ___ _ __ ___ _ __    |_   _|__   ___ | |___
| |_) / _ \ '_ \| __/ _ \ '__/ _ \ '_ \     | |/ _ \ / _ \| / __|
|  __/  __/ | | | ||  __/ | |  __/ |_) |    | | (_) | (_) | \__ \
|_|   \___|_| |_|\__\___|_|  \___| .__/     |_|\___/ \___/|_|___/
                                 |_|            ptapptest v1.0.0
                                       https://www.penterep.com


[*] Server information
[i] Banner:
SSH-2.0-OpenSSH_7.9p1 Debian-10+deb10u2
[i] Host key:
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQDPsyXTzbbzzvfLWNTt1rkg...
[i] Authentication methods:
publickey, password
[*] ssh-audit scan results
[i] Identified 4 CVEs
CVE-2021-41617 (7.0): privilege escalation via supplemental groups
CVE-2020-15778 (7.8): command injection via anomalous argument
   transfers
...
```

```
[i] Identified 15 insecure SSH configurations
CRITICAL kex/del: diffie-hellman-group14-sha1
CRITICAL kex/del: ecdh-sha2-nistp256
...
```

Výpis 4.18: Výstup v Penterep JSON formátu


```
{
    "satid": "",
    "guid": "",
    "status": "finished",
    "message": "",
    "results": {
        "nodes": [],
        "properties": {
            "banner": "SSH-2.0-OpenSSH_7.9p1 Debian-10+deb10u2",
            "hostKey": ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABg...",
            "authMethods": [
                "publickey",
                "password"
            ],
            "sshauditStatus": "ok"
        },
        "vulnerabilities": [
            {
                "vulnCode": "PTV-SSH-VULNERABLEVERSION",
                "vulnRequest": "ssh-audit scan",
                "vulnResponse": "CVE-2021-41617 (7.0): privilege
   escalation via supplemental groups\nCVE-2020-15778 (7.8):
   command injection via anomalous argument transfers\n..."
            },
            {
                "vulnCode": "PTV-SSH-INSECURECRYPTOGRAPHY",
                "vulnRequest": "ssh-audit scan",
                "vulnResponse": "CRITICAL kex/del: diffie-hellman-
   group14-sha1\nCRITICAL kex/del: ecdh-sha2-nistp256\n..."
            }
        ]
    }
}
```

4.5 Vývoj modulu SMTP

Modul SMTP byl vyvinut ve spolupráci se společností HACKER Consulting s.r.o.[21], která implementovala většinu funkcionalit obsažených v tomto modulu a jeho rozpracovanou verzi poskytla autorovi této diplomové práce k zapracování do vyvíjeného nástroje ptapptest. Z tohoto důvodu nebyla v rámci vypracování této diplomové práce provedena analýza existujících nástrojů pro testování protokolu SMTP a rovněž nebyl autorem této práce proveden návrh modulu SMTP. V rámci této práce byl obdržený kód modulu SMTP upraven tak, aby odpovídal architektuře ostatních modulů a rovněž byl doplněn o použití pomocných funkcionalit nástroje, a to konkrétně o již zmíněnou metodu `simple_bruteforce` a o využití nástroje ptntlmauth. Nástroj ptntlmauth byl v rámci této diplomové práce vyvinut během vývoje modulu POP3, a proto bude představen až při popisu vývoje tohoto modulu v sekci 4.6.

Konečná implementace modulu SMTP se nachází v souborech `modules/smtp.py` a `modules/utils/blacklist_parser.py` a skládá se z hlavní třídy modulu `SMTP`, třídy parametrů `SMTPArgs`, datové třídy `SMTPResults` a třídy `BlacklistParser`. Definice tříd `SMTPArgs` a `SMTPResults` jsou uvedeny v následujícím výpisu.

Výpis 4.19: Třídy `SMTPArgs` a `SMTPResults`

```
@dataclass
class SMTPResults:
    blacklist: BlacklistResult | None = None
    spf_records: dict[str, list[str]] | None = None
    creds: set[Creds] | None = None
    enum_results: list[EnumResult] | None = None
    info: InfoResult | None = None
    max_connections: MaxConnectionsResult | None = None
    ntlm: NTLMResult | None = None
    open_relay: bool | None = None

class SMTPArgs(ArgsWithBruteforce):
    target: Target
    tls: bool
    starttls: bool
    info: bool
    ntlm: bool
    mail_from: str | None
    rcpt_to: str | None
    wordlist: str | None
    fqdn: str | None
    enumerate: list[str] | str | None
    blacklist_test: bool
```

[21] HACKER Consulting s.r.o.: `https://www.hacker-consulting.cz/`

```
    max_connections: bool
    slow_down: bool
    spf_test: bool
    open_relay: bool
    interactive: bool

    def add_subparser(self, name: str, subparsers) -> None:
...
```

Hlavní metodou modulu je opět metoda `run`, která na základě konfigurace modulu spouští jednotlivé testovací metody. Výsledky všech testovacích metod jsou postupně ukládány do instance třídy `SMTPResults`. Následující výpis obsahuje kód metody `run`.

Výpis 4.20: Hlavní metoda `run`

```
def run(self):
    self.results = SMTPResults()
    smtp = None

    # Indirect scanning
    if self.args.blacklist_test:
        self.results.blacklist = self.test_blacklist(self.target)

    if self.args.spf_test:
        self.results.spf_records = self._get_nameservers(self.
    target)

    # Direct scanning
    # enter only if any of these arguments were explicitly
    specified
    if (
        self.args.info
        or self.args.interactive
        or self.args.ntlm
        or self.args.open_relay
        or self.args.enumerate
        or self.args.max_connections
        or self.do_brute
    ):
        smtp, info = self.initial_info()

        if self.args.info:
            self.results.info = info

        if self.args.interactive and not self.use_json:
            self.start_interactive_mode(smtp)

```

```
31          if self.args.ntlm:
32              self.results.ntlm = self.auth_ntlm(smtp)
33
34          if self.args.open_relay:
35              self.results.open_relay = self.open_relay_test(
36                  smtp, "TEST", self.args.mail_from, self.args.
    rcpt_to
37              )
38
39          if self.args.enumerate is not None:
40              self.results.enum_results = self.enumeration(smtp)
41
42          if self.args.max_connections:
43              self.results.max_connections = self.
    max_connections_test()
44
45          if self.do_brute:
46              self.results.creds = simple_bruteforce(
47                  self._try_login,
48                  self.args.user,
49                  self.args.users_file,
50                  self.args.passw,
51                  self.args.passw_file,
52                  self.args.spray,
53                  self.args.threads,
54              )
```

Metoda `test_blacklist` využívá třídu `BlacklistParser` pro komunikaci se službou Blacklists webové aplikace *MXToolBox*[22], skrze kterou zjišťuje, zda se zkoumaná doména či IP adresa nachází v některém z blacklistů, které aplikace MXToolBox zkoumá. Výsledkem metody je instance datové třídy `BlacklistResult` obsahující informace o tom, zda se zkoumaná doména nachází na některém z blacklistů.

Metoda `initial_info` provádí bannergrabbing zaznamenáním uvítací zprávy serveru a jeho odpovědi na příkaz `EHLO`. Výsledkem metody jsou zaznamenané odpovědi obsažené v instanci třídy `InfoResult`.

Metoda `start_interactive_mode` umožňuje uživateli nástroje navázat interaktivní relaci s cílovým SMTP serverem. Metoda přijímá standardní vstup uživatele a přijatý text posílá SMTP serveru. Po zpracování příkazů SMTP serverem je uživateli zobrazena odpověď serveru. Vzhledem k charakteru metody po jejím skončení není navrácen žádný výsledek.

Metoda `auth_ntlm` využívá již zmíněný nástroj ptntlmauth pro realizaci NTLM autentizace s SMTP serverem. Cílem metody je z odpovědi SMTP serveru pomocí

[22]MXToolBox: `https://mxtoolbox.com/`

nástroje ptntlmauth extrahovat interní informace o serveru, jako je jeho doménové jméno či verze operačního systému.

Metoda `open_relay_test` se pokouší pomocí cílového SMTP serveru odeslat e-mailovou zprávu, a to bez předešlé autentizace. Výsledkem této metody je booleovská hodnota reprezentující zjištění, zda se SMTP server chová jako open relay či nikoliv.

Metoda `enumeration` umožňuje ověřit, zda je SMTP server zranitelný vůči enumeraci uživatelských jmen při použití příkazů `EXPN`, `VRFY` či `RCPT TO`. Metoda vykoná každý z příkazů společně s neexistujícím uživatelským jménem a pokud odpověď serveru naznačuje, že daný uživatel neexistuje, je to známka zranitelnosti daného příkazu. Výsledkem metody je seznam instancí datových tříd `EnumResult`, kde každá z těchto instancí popisuje zranitelnost jednoho z uvedených příkazů.

Metoda `max_connections_test` zkoumá maximální počet souběžných spojení, která SMTP server povoluje navázat. V rámci tohoto testu je rovněž sledováno, zda SMTP server při překročení daného limitu začne blokovat tvorbu nových spojení daného klienta. Pokud server blokování implementuje, je dále zkoumána doba, po kterou blokace platí a po jejímž uplynutí je server opět ochoten přijímat nová spojení od daného klienta.

Stejně jako u modulu FTP je i v tomto modulu pro umožnění pokusu o přihlášení s využitím slovníkového útoku využita pomocná funkce `simple_bruteforce`. Jediným rozdílem od modulu FTP je zde odlišná implementace metody `_try_login`.

Metoda `_get_nameservers` je poslední implementovanou metodou SMTP modulu a slouží pro získání DNS záznamů cílové domény obsahujících konfigurace mechanismu SPF. Za účelem demonstrace kódu modulu SMTP je v následujícím výpisu uveden zkrácený kód této metody. Na začátku metody je pomocí knihovny *dnspython*[23] provedena DNS žádost o poskytnutí všech NS záznamů náležících cílové doméně (řádky 4–6). Z těchto záznamů jsou extrahována doménová jména nalezených DNS serverů a následně jsou tato jména přeložena na jejich odpovídající IP adresy (řádky 9–14). Pomocí získaných IP adres je následně možné kontaktovat tyto DNS servery s žádostí o poskytnutí všech TXT a SPF záznamů cílové domény (řádek 18, metoda `_get_spf_for_ns`). Na konci metody je z obdržených DNS záznamů sestavena a navrácena datová struktura typu `dict`, která realizuje mapování doménových jmen DNS serverů cílové domény na seznamy DNS záznamů, které byly obdrženy z daného DNS serveru.

Výpis 4.21: Zkrácený obsah metody `_get_nameservers`

```
1 def _get_nameservers(self, domain) -> dict[str, list[str]]:
2     resolver = dns.resolver.Resolver()
3 ...
4     try:
```

[23] dnspython: `https://www.dnspython.org/`

```
        ns_query = resolver.resolve(domain, "NS", tcp=True)
        nameserver_list = [str(rdata)[:-1] for rdata in ns_query]
...
    spf_result = {}
    for ns in nameserver_list:
        try:
            ns_ip = socket.gethostbyname(ns)
        except Exception as e:
            self.ptdebug(f"Exception - {e}", Out.ERROR)
            continue
        resolver.nameservers = [ns_ip]
        spf_result.update({ns: []})
...
        spf_result[ns].extend(self._get_spf_for_ns(domain, resolver
    ))
...
    results = {ns: val for ns, val in spf_result.items() if len(val
    ) > 0}
    return results
```

Po dokončení běhu hlavní metody `run` dochází opět metodou `output` k vytištění výsledků běhu modulu na standardní výstup nástroje, a to buď ve formátu pro čtení člověkem, nebo ve formátu JSON pro zpracovávní platformou Penterep. Ukázka obou těchto výstupních formátů je uvedena v následujících výpisech.

Výpis 4.22: Výstup pro čtení člověkem

```
 ____            _                          _____           _
|  _ \ ___ _ __ | |_ ___ _ __ ___ _ __    |_   _|__   ___ | |___
| |_) / _ \ '_ \| __/ _ \ '__/ _ \ '_ \     | |/ _ \ / _ \| / __|
|  __/  __/ | | | ||  __/ | |  __/ |_) |    | | (_) | (_) | \__ \
|_|   \___|_| |_|\__\___|_|  \___| .__/     |_|\___/ \___/|_|___/
                                 |_|            ptapptest v1.0.0
                                       https://www.penterep.com


[*] Blacklist information: listed
[i] Listed on the following blacklists:
UCEPROTECTL3: "67.215.240.194 was listed" (TTL=2100)
[*] Server information
[i] Banner:
ecrater.com ESMTP
[i] EHLO:
ecrater.com
PIPELINING
8BITMIME
[*] NTLM information failed
[*] Open relay: False
```

```
[*] User enumeration methods
[i] Method "expn" not vulnerable
[i] Method "vrfy" not vulnerable
[i] Method "rcpt" vulnerable
```

Výpis 4.23: Výstup v Penterep JSON formátu


```
{
    "satid": "",
    "guid": "",
    "status": "finished",
    "message": "",
    "results": {
        "nodes": [],
        "properties": {
            "banner": "ecrater.com ESMTP",
            "ehloCommand": "ecrater.com\nPIPELINING\n8BITMIME",
            "ntlmInfoStatus": "failed"
        },
        "vulnerabilities": [
            {
                "vulnCode": "PTV-SMTP-BLACKLIST",
                "vulnRequest": "blacklists containing target 67.215
   .240.194",
                "vulnResponse": "UCEPROTECTL3: \"67.215.240.194 was
    listed\" (TTL=2100)"
            },
            {
                "vulnCode": "PTV-SMTP-USERENUMERATION",
                "vulnRequest": "enumeration methods: ['VRFY', 'EXPN
   ', 'RCPT']",
                "vulnResponse": "Method \"rcpt\" vulnerable"
            }
        ]
    }
}
```


4.6 Vývoj modulu POP3

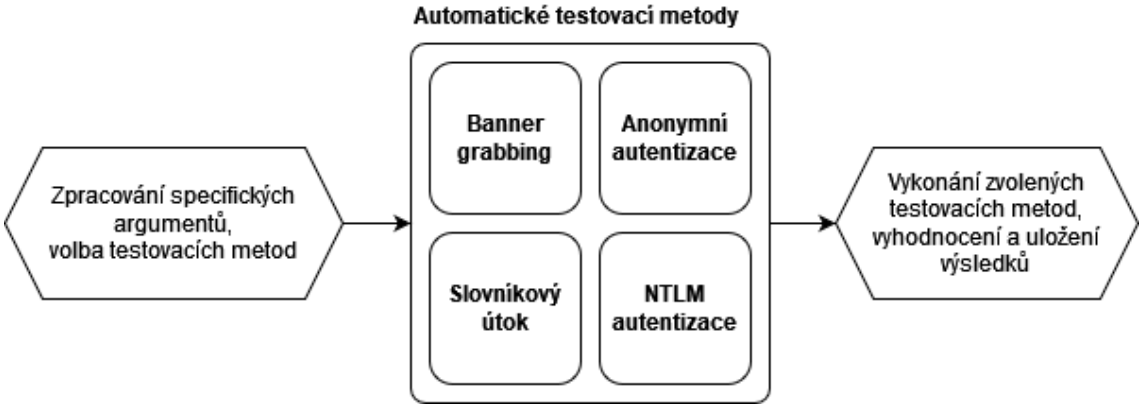
Z návrhu kontrolních bodů v sekci 3.1.4 byly pro modul POP3 navrženy následující funkcionality:

- identifikace softwaru a verze serveru (bannergrabbing),
- zjištění interních informací o serveru pomocí NTLM autentizace,
- anonymní autentizace,
- slovníkový útok.

Na základě požadovaných funkcionalit byla provedena analýza existujících nástrojů, během které ovšem nebyly nalezeny žádné nástroje, které by implementovaly jakoukoliv z požadovaných funkcionalit. Na základě tohoto zjištění tedy bylo rozhodnuto, že veškeré funkcionality modulu budou realizovány vlastní implementací.

4.6.1 Návrh modulu

Po provedení analýzy existujících nástrojů byl navržen modul na penetrační testování protokolu POP3 (obrázek 4.4). Modul nebude využívat žádné existující externí nástroje a veškeré jeho funkcionality budou implementovány pouze s využitím vhodných modulů obsažených ve standardní knihovně jazyka Python. Pro realizaci komunikace protokolu POP3 bude využit standardní modul `poplib`[24].



Obr. 4.4: Návrh modulu POP3

4.6.2 Implementace modulu

Implementace modulu POP3 se nachází v souboru `modules/pop3.py` a zahrnuje hlavní třídu modulu `POP3`, třídu parametrů `POP3Args`, datovou třídu `POP3Results` a několik dalších datových tříd. Definice tříd `POP3Args` a `POP3Results` jsou uvedeny v následujícím výpisu.

Výpis 4.24: Třídy `POP3Args` a `POP3Results`

```
@dataclass
class POP3Results:
    info: InfoResult | None = None
    anonymous: bool | None = None
    ntlm: NTLMResult | None = None
    creds: set[Creds] | None = None

```

[24] poplib: https://docs.python.org/3/library/poplib.html

```
class POP3Args(ArgsWithBruteforce):
    target: Target
    tls: bool
    starttls: bool
    info: bool
    ntlm: bool
    anonymous: bool

    def add_subparser(self, name: str, subparsers) -> None:
...
```

Totožně jako u předchozích modulů se i v tomto modulu veškeré testovací metody spouští v rámci hlavní metody `run` dle konfigurace parametrů modulu. Výsledky testovacích metod jsou v rámci běhu modulu ukládány do instance třídy `POP3Results`. Následující výpis obsahuje kód metody `run`.

Výpis 4.25: Hlavní metoda `run`

```
def run(self) -> None:
    self.results = POP3Results()
    self.pop3 = self.connect()

    if self.args.info:
        self.results.info = self.info()

    if self.args.ntlm:
        self.results.ntlm = self.auth_ntlm()

    if self.args.anonymous:
        self.results.anonymous = self.auth_anonymous()

    if self.do_brute:
        self.results.creds = simple_bruteforce(
            self._try_login,
            self.args.user,
            self.args.users_file,
            self.args.passw,
            self.args.passw_file,
            self.args.spray,
            self.args.threads,
            )
```

Metoda `connect` obdobně jako u modulu FTP poskytuje možnost vytvoření nového spojení s POP3 serverem, jehož parametry jsou dány konfigurací modulu (IP adresa, síťový port, způsob zabezpečení).

Metoda `info` slouží pro zaznamenání úvodní uvítací zprávy serveru a následnému provedení příkazu `CAPA`, který cílí na zjištění dalších informací o POP3 serveru.

Výsledkem metody je instance datové třídy `InfoResult` obsahující zaznamenané informace.

Metoda `auth_anonymous` implementuje proces anonymní autentizace, během kterého dochází k pokusu o přihlášení pomocí příkazu `AUTH ANONYMOUS`. Výsledkem této metody je booleovská hodnota reprezentující úspěch či neúspěch tohoto procesu.

Slovníkový útok je v tomto modulu obdobně jako u předchozích modulů zajištěn využitím pomocné funkce `simple_bruteforce` a implementací vlastní metody `_try_login`, která je pomocnou funkcí volána.

Metoda `auth_ntlm` je poslední testovací metodou modulu a stejně jako u modulu SMTP automatizuje extrakci interních informací o POP3 serveru na základě odpovědí serveru přijatých během procesu NTLM autentizace. Během vývoje této metody bylo rozhodnuto, že implementace mechanismů pro zpracování NTLM autentizace je natolik obecná a nezávislá (není vázána na konkrétní aplikační protokol), že by bylo vhodné ji realizovat formou samostatného nástroje, který by mohl být následně integrován do libovolných nástrojů platformy Penterep, anebo sloužit uživatelům jako samostatný pomocný nástroj. Z tohoto důvodu byl kód zpracování NTLM autentizace extrahován do samostatného nástroje s názvem ptntlmauth. Stěžejními částmi tohoto nástroje jsou datová třída `NTLMInfo` a funkce `get_NegotiateMessage_data` a `decode_ChallengeMessage_blob`. Třída `NTLMInfo` je uvedena v následujícím výpisu a slouží pro uchování informací extrahovaných z procesu NTLM autentizace.

Výpis 4.26: Třída `NLTLMInfo`

```
class NTLMInfo(NamedTuple):
    target_name: str | None
    netbios_domain: str | None
    netbios_computer: str | None
    dns_domain: str | None
    dns_computer: str | None
    dns_tree: str | None
    os_version: str | None
```

Funkce `get_NegotiateMessage_data` slouží pro sestrojení úvodní zprávy NTLM protokolu a pro tuto operaci využívá modul `ntlm` z knihovny *ntlm-auth*[25], konkrétně jeho třídu `NegotiateMessage`. Výsledkem této metody je bajtová reprezentace sestrojené zprávy. Funkce `decode_ChallengeMessage_blob` pak zajišťuje samotný proces extrakce informací a vytvoření instance třídy `NTLMInfo` z bajtové reprezentace NTLM odpovědi serveru. Pro zpracování NTLM odpovědi je opět využit modul `ntlm`, konkrétně jeho třída `ChallengeMessage`. Pro demonstraci využití nástroje ptntlmauth nástrojem ptapptest je v následujícím výpisu uveden obsah

[25] ntlm-auth: `https://pypi.org/project/ntlm-auth/`

metody `auth_ntlm`. Proces NTLM autentizace je zahájen příkazem `AUTH NTLM` (řádek 6), po kterém server očekává zaslání úvodní NTLM zprávy `NegotiateMessage`. Tato zpráva je vytvořena pomocí nástroje ptntlmauth a její Base64 zakódovaná reprezentace je odeslána serveru (řádky 8–9). Následně je od serveru obdržena Base64 zakódovaná odpověď, která je po jejím dekódování předána ke zpracování nástroji ptntlmauth (řádek 14), který z této zprávy extrahuje informace o serveru formou instance třídy `NTLMInfo`.

Výpis 4.27: Metoda `auth_ntlm`

```
1  def auth_ntlm(self) -> NTLMResult:
2      try:
3          # Separate connection not to corrupt the main socket
4          pop3 = self.connect()
5
6          res: bytes = pop3._shortcmd("AUTH NTLM")
7          if res.strip().startswith(b"+"):
8              b64_ntlm_negotiation = b64encode(
    get_NegotiateMessage_data()).decode()
9              res = pop3._shortcmd(b64_ntlm_negotiation).strip()
10
11             # res = b'+ base64containing+signs'
12             b64_ntlm_challenge = b"+".join(res.split(b"+")[1:])
13
14             ntlminfo = decode_ChallengeMessage_blob(b64decode(
    b64_ntlm_challenge))
15             return NTLMResult(True, ntlminfo)
16         else:
17             return NTLMResult(False, None)
18     except:
19         return NTLMResult(False, None)
```

Po dokončení hlavní metody `run` stejně je jako u předchozích modulů metodou `output` realizován standardní výstup nástroje obsahující výsledky běhu modulu. Následující výpisy obsahují zkrácený standardní výstup modulu při jeho úspěšném běhu, a to jak ve formátu pro čtení člověkem (výpis 4.28), tak v JSON formátu pro platformu Penterep (výpis 4.29).

Výpis 4.28: Výstup pro čtení člověkem

```
 ____            _                          _____           _
|  _ \ ___ _ __ | |_ ___ _ __ ___ _ __     |_   _|__   ___ | |___
| |_) / _ \ '_ \| __/ _ \ '__/ _ \ '_ \      | |/ _ \ / _ \| / __|
|  __/  __/ | | | ||  __/ | |  __/ |_) |     | | (_) | (_) | \__ \
|_|   \___|_| |_|\__\___|_|  \___| .__/      |_|\___/ \___/|_|___/
                                 |_|            ptapptest v1.0.0
                                          https://www.penterep.com
```

```
[*] Server information
[i] Banner:
+OK POP3 server ready <f92f2ba2-1cbb-44c7-b8b5-5b3d10bfaf99@ws275.
   win.arvixe.com>
[i] CAPA command:
TOP UIDL USER STLS
SASL: NTLM PLAIN
IMPLEMENTATION: Smartertools_SmarterMail
[*] Anonymous authentication: False
[*] NTLM information
[i] Target name: WS275
[i] NetBios domain name: WIN.ARVIXE.COM
[i] NetBios computer name: WS275
[i] DNS domain name: win.arvixe.com
[i] DNS computer name: WS275.win.arvixe.com
[i] DNS tree: win.arvixe.com
[i] OS version: 10.0.14393
```

Výpis 4.29: Zkrácený výstup v Penterep JSON formátu


```
{
    "satid": "",
    "guid": "",
    "status": "finished",
    "message": "",
    "results": {
        "nodes": [],
        "properties": {
            "banner": "+OK POP3 server ready <b7b4304d-90d0-4b6d-b6
   23-c7b639873a30@ws275.win.arvixe.com>",
            "capability": "TOP UIDL USER STLS\nSASL: NTLM PLAIN\
   nIMPLEMENTATION: Smartertools_SmarterMail",
            "ntlmInfoStatus": "ok"
        },
        "vulnerabilities": [
            {
                "vulnCode": "PTV-POP3-NTLMINFORMATION",
                "vulnRequest": "ntlm authentication",
                "vulnResponse": "Target name: W\u0000S\u00002\u0000
   7\u00005\u0000\nNetBios domain name: W\u0000I\u0000N\u0000.\u000
   0A\u..."
            }
        ]
    }
}
```

4.7 Vývoj modulu IMAP

Z bodů kontrolního seznamu protokolu IMAP navrženého v sekci 3.1.5 byly pro modul IMAP navrženy následující funkcionality:

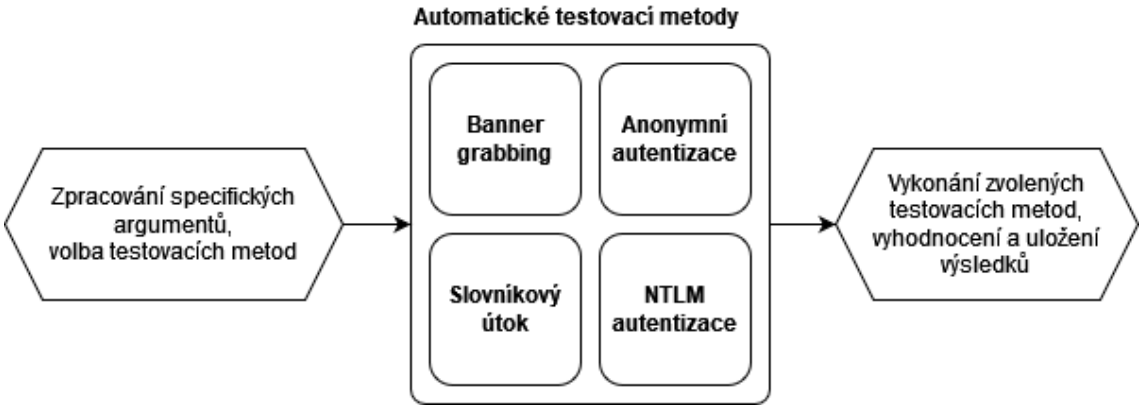
- identifikace softwaru a verze serveru (bannergrabbing),
- odhalení interních informací pomocí NTLM autentizace,
- anonymní autentizace,
- slovníkový útok.

Na základě požadovaných funkcionalit byla provedena analýza existujících nástrojů, ovšem stejně jako u modulu POP3 nebyly během analýzy nalezeny žádné nástroje, které by umožňovaly zajistit některé z požadovaných funkcionalit. Jediným nástrojem, který lze v rámci modulu IMAP využít, je vlastní nástroj ptntlmauth představený v předchozí sekci. Na základě výsledku analýzy tedy bylo rozhodnuto, že implementace veškerých funkcionalit modulu bude realizována vlastním řešením s tím, že pro zajištění komponenty NTLM autentizace bude využit nástroj ptntlmauth.

4.7.1 Návrh modulu

Na základě analýzy existujících nástrojů byl proveden návrh modulu IMAP (obrázek 4.5), který je vzhledem k podobnosti protokolů a shodnosti jejich požadovaných funkcionalit totožný s návrhem předchozího modulu POP3. Implementace požadovaných funkcionalit bude realizována vlastním řešením, integrací vlastního nástroje ptntlmauth a využitím vhodných modulů obsažených ve standardní knihovně jazyka Python. Pro realizaci komunikace protokolu IMAP bude využit standardní modul `imaplib`[26].

[26] `imaplib`: `https://docs.python.org/3/library/imaplib.html`



Obr. 4.5: Návrh modulu IMAP

4.7.2 Implementace modulu

Implementace modulu IMAP se nachází v souboru `modules/imap.py` a obsahuje hlavní třídu modulu `IMAP`, třídu parametrů `IMAPArgs`, datovou třídu `IMAPResults` a další datové třídy. Definice tříd `IMAPArgs` a `IMAPResults` jsou uvedeny v následujícím výpisu. Z atributů těchto tříd lze pozorovat již zmíněnou podobnost modulu IMAP s modulem POP3.

Výpis 4.30: Třídy `IMAPArgs` a `IMAPResults`

```
@dataclass
class IMAPResults:
    info: InfoResult | None = None
    anonymous: bool | None = None
    ntlm: NTLMResult | None = None
    creds: set[Creds] | None = None

class IMAPArgs(ArgsWithBruteforce):
    target: Target
    tls: bool
    starttls: bool
    info: bool
    anonymous: bool
    ntlm: bool

    def add_subparser(self, name: str, subparsers) -> None:
```

V hlavní metodě `run` lze opět pozorovat shodnost modulu s modulem POP3. Struktura této metody je zcela totožná, jediným rozdílem je využití datové třídy `IMAPResults` pro uchovávání průběžných výsledků jednotlivých testovacích metod. Následující výpis obsahuje kód metody `run`.

Výpis 4.31: Hlavní metoda `run`

```
1  def run(self) -> None:
2      self.results = IMAPResults()
3      self.imap = self.connect()
4
5      if self.args.info:
6          self.results.info = self.info()
7
8      if self.args.ntlm:
9          self.results.ntlm = self.auth_ntlm()
10
11     if self.args.anonymous:
12         self.results.anonymous = self.auth_anonymous()
13
14     if self.do_brute:
15         self.results.creds = simple_bruteforce(
16             self._try_login,
17             self.args.user,
18             self.args.users_file,
19             self.args.passw,
20             self.args.passw_file,
21             self.args.spray,
22             self.args.threads,
23         )
```

Metoda `connect` opět jako u předchozích modulů umožňuje vytvářet nová spojení s IMAP serverem zohledňující konfiguraci cílové adresy a způsoby zabezpečení komunikace.

Metoda `info` pozoruje uvítací zprávu serveru společně s odpověďmi na příkazy `ID` a `CAPABILITY`. Výsledkem metody je instance třídy `InfoResult` reprezentující pozorované informace.

Metoda `auth_ntlm` zajišťuje proces NTLM autentizace a extrakce interních informací o serveru s využitím nástroje ptntlmauth. Tento proces se skládá z příkazu `a1 AUTHENTICATE NTLM` následovaného využitím funkcí nástroje ptntlmauth pro získání instance třídy `NTLMInfo`. Výsledkem metody je instance třídy `NTLMResult` popisující průběh a výsledek této metody.

Funkcionalita slovníkového útoku je v tomto modulu opět zajištěna pomocnou funkcí `simple_bruteforce` společně s vlastní implementací metody `_try_login`.

Poslední metodou modulu je metoda `auth_anonymous`, která zajišťuje funkcionalitu anonymní autentizace. Pro ukázku kódu modulu je tato metoda uvedena v následujícím výpisu. Nejprve je zde definována vnořená funkce `authobject` (řádky 2–5), která je následně předána metodě `authenticate` modulu `imaplib` (řádek 8) jakožto tzv. callback funkce pro využití během procesu anonymní autentizace. V průběhu

autentizace modul `imaplib` tuto callback funkci volá, aby mu poskytla zprávu, která má být zaslána serveru. V tomto případě bude poskytnutou zprávou náhodný řetězec o délce 5 až 10 znaků. Po provedení procesu autentizace je prozkoumán stavový kód odpovědi serveru uložený v proměnné `typ` (řádek 9) a na základě hodnoty tohoto kódu je metodou navrácena booleovská hodnota reprezentující úspěch či neúspěch procesu anonymní autentizace.

Výpis 4.32: Metoda `auth_anonymous`

```
1  def auth_anonymous(self) -> bool:
2      def authobject(b: bytes):
3          return b"".join(
4              random.choice(ascii_letters).encode() for _ in range(
   random.randint(5, 10))
5          )
6
7      try:
8          typ, _ = self.imap.authenticate("ANONYMOUS", authobject)
9          return True if typ == "OK" else False
10     except:
11         return False
```

Po provedení hlavní metody `run` je opět metodou `output` uskutečněn standardní výstup nástroje prezentující výsledky běhu modulu. Následující výpisy obsahují zkrácený standardní výstup modulu při jeho úspěšném běhu, a to jak ve formátu pro čtení člověkem (výpis 4.33), tak v JSON formátu pro platformu Penterep (výpis 4.34).

Výpis 4.33: Výstup pro čtení člověkem

```
 ____             _                          _____           _
|  _ \ ___ _ __ | |_ ___ _ __ ___ _ __ __   |_   _|__   ___ | |___
| |_) / _ \ '_ \| __/ _ \ '__/ _ \ '_ \     | |/ _ \ / _ \| / __|
|  __/  __/ | | | ||  __/ | |  __/ |_) |    | | (_) | (_) | \__ \
|_|   \___|_| |_|\__\___|_|  \___| .__/     |_|\___/ \___/|_|___/
                                 |_|           ptapptest v1.0.0
                                       https://www.penterep.com


[*] Server information
[i] Banner:
* OK [CAPABILITY IMAP4rev1 LITERAL+ SASL-IR LOGIN-REFERRALS ID
   ENABLE IDLE STARTTLS AUTH=ANONYMOUS AUTH=PLAIN AUTH=DIGEST-MD5
   AUTH=CRAM-MD5 AUTH=LOGIN] Dovecot ready.
[i] ID command:
NIL
[i] CAPABILITY command:
```

```
IMAP4REV1, LITERAL+, SASL-IR, LOGIN-REFERRALS, ID, ENABLE, IDLE,
    STARTTLS, AUTH=ANONYMOUS, AUTH=PLAIN, AUTH=DIGEST-MD5, AUTH=CRAM
    -MD5, AUTH=LOGIN
[*] Anonymous authentication: True
[*] NTLM information failed
```

Výpis 4.34: Zkrácený výstup v Penterep JSON formátu


```
{
    "satid": "",
    "guid": "",
    "status": "finished",
    "message": "",
    "results": {
        "nodes": [],
        "properties": {
            "banner": "* OK [CAPABILITY IMAP4rev1 LITERAL+ SASL-IR
    LOGIN-REFERRALS ID ENABLE IDLE STARTTLS AUTH=ANONYMOUS AUTH=
    PLAIN AUTH=DIGEST-MD5 AUTH=CRAM-MD5 AUTH=LOGIN] Dovecot ready.",
            "idCommand": "NIL",
            "capabilityCommand": "IMAP4REV1, LITERAL+, SASL-IR,
    LOGIN-REFERRALS, ID, ENABLE, IDLE, STARTTLS, AUTH=ANONYMOUS,
    AUTH=PLAIN, AUTH=DIGEST-MD5, AUTH=CRAM-MD5, AUTH=LOGIN",
            "ntlmInfoStatus": "failed"
        },
        "vulnerabilities": [
            {
                "vulnCode": "PTV-IMAP-ANONYMOUS",
                "vulnRequest": "anonymous authentication"
            }
        ]
    }
}
```

5 Testování nástroje ptapptest

Tato kapitola je závěrečnou kapitolou této práce a věnuje se testování nástroje ptapptest. Během testování nástroje byl sestrojen testovací skript, s jehož pomocí bylo vygenerováno 5 CSV souborů zaznamenávajících výsledky testování jednotlivých modulů nástroje. Nástroj byl testován celkem na 1000 aplikačních serverech, z nichž každý modul nástroje byl testován na 200 aplikačních serverech z této množiny. Testovací skript i CSV soubory s výsledky testů jsou součástí elektronické přílohy této práce.

5.1 Testovací skript

Testování nástroje probíhalo s pomocí vyhledávače *Shodan*[1], který umožňuje vyhledávat síťová zařízení dostupná skrze síť internet na základě definovaných filtrů. Zařízení lze filtrovat například na základě otevřených síťových portů, což umožňuje efektivně vyhledávat taková zařízení, která jsou aplikačními servery hledaných aplikačních protokolů. Vyhledávač Shodan také nabízí webové API rozhraní, pro jehož využití existuje oficiální knihovna *shodan*[2] pro programovací jazyk Python. [57]

Za účelem testování nástroje ptapptest byl sestrojen testovací skript `test.py`, jenž využívá modul `client` knihovny shodan pro vyhledávání skrze Shodan API. Následující výpis obsahuje hlavní funkci `main` tohoto skriptu. Pro všechny testované aplikační protokoly jsou pomocí funkce `search_shodan` vyhledány odpovídající aplikační servery (řádek 19). Následně je pro všechna nalezená zařízení pomocí modulu `concurrent.futures` uskutečněno vícevláknové provedení testovací metody daného protokolu (řádky 23–27). Výsledky těchto metod jsou agregovány do proměnné `results` (řádky 29–30), jejíž obsah je po ukončení testování každého protokolu uložen do souboru ve formátu CSV (řádky 33-36).

Výpis 5.1: Funkce `main`

```
1 SHODAN_KEY = "..."
2 PROTOCOLS = {"ftp": 21, "ssh": 22, "smtp": 25, "pop3": 110, "imap":
      143}
3 SERVERS = {"ftp": [], "ssh": [], "smtp": [], "pop3": [], "imap":
     []}
4
5 def main():
6     functions = {
7         "ftp": test_ftp,
8         "ssh": test_ssh,
```

[1] Vyhledávač Shodan: `https://www.shodan.io/dashboard`
[2] Knihovna shodan: `https://github.com/achillean/shodan-python`

```
9           "smtp": test_smtp,
10          "pop3": test_pop3,
11          "imap": test_imap,
12      }
13
14      # Number of scanned targets per protocol
15      n_targets = 200
16
17      for proto in PROTOCOLS:
18          # Obtain IP addresses that have the protocol port open
19          search_shodan(proto, f"port:{PROTOCOLS[proto]}", n_targets)
20
21          # Run appropriate test function for these targets
22          results: list[dict[str, bool]] = []
23          with concurrent.futures.ThreadPoolExecutor(30) as executor:
24              results_ = [
25                  executor.submit(functions[proto], Target(ip,
    PROTOCOLS[proto]))
26                  for ip in SERVERS[proto]
27              ]
28
29              for future in concurrent.futures.as_completed(results_)
    :
30                  results.append(future.result())
31
32          # Write scan results as CSV
33          with open(f"{proto}.csv", "w", newline="") as f:
34              writer = csv.DictWriter(f, results[0].keys())
35              writer.writeheader()
36              writer.writerows(results)
```

Moduly nástroje byly během testování spouštěny přímo skrze jejich programové rozhraní jazyka Python. Pro demonstraci konfigurace argumentů, spuštění modulu a interpretace jeho výsledků mimo prostředí příkazové řádky je v následujícím výpisu uveden zkrácený kód testovací metody `test_ftp`. Do testovacího skriptu jsou importovány všechny moduly nástroje (řádek 1), které jsou následně využívány pro přístup k potřebným třídám a jejich metodám. Struktura každé testovací funkce se skládá z vytvoření a vyplnění třídy parametrů modulu (řádky 5–11), předání těchto parametrů hlavní třídě testovaného modulu (řádek 14) a následného volání jeho hlavní metody `run()` (řádek 15). Po dokončení běhu modulu jsou z instance jeho hlavní datové třídy extrahovány zkoumané výsledky (řádky 16–24), které jsou následně formou datového typu `dict` navráceny hlavní testovací metodě `main` (řádky 26–40).

Výpis 5.2: Funkce `test_ftp`

```
from ptapptest.ptapptest.modules import ftp, ssh, smtp, pop3, imap

def test_ftp(target: Target) -> dict[str, str | bool | None]:
...
    args = ftp.FTPArgs()
    args.json = False
    args.tls = False
...
    args.target = target
    args.anonymous = True
    args.info = True
...
    try:
        module = ftp.FTP(args, PtJsonLib())
        module.run()
        r = module.results

        banner, syst, stat = False, False, False
        if i := r.info:
            banner = bool(i.banner)
            syst = bool(i.syst)
            stat = bool(i.stat)

        anonymous = bool(r.anonymous)
...
        return {
            "ip": target.ip,
            "accessible": True,
            "banner": banner,
            "syst": syst,
            "stat": stat,
            "anonymous": anonymous,
            "access_error": access_error,
            "dirlist": dirlist,
            "write": write,
            "read": read,
            "delete": delete,
            "bounce_valid": bounce_valid,
            "bounce_open": bounce_open,
        }
...
```

5.2 Výsledky testování

Pro každý modul nástroje byla zvolena určitá konfigurace argumentů, s níž bylo následně provedeno testování vůči 200 aplikačním serverům nalezeným pomocí vyhledávače Shodan. Konfigurace modulů byly voleny tak, aby testování pokrylo maximální možné množství funkcionalit nástroje. Do konfigurace nebyly zahrnovány taková nastavení, která by byla příliš invazivní vůči testovaným aplikačním serverům (např. slovníkové útoky). Následující text obsahuje stručný popis výsledků testování každého modulu, a to včetně konfigurace, se kterou byl daný modul testován.

Testování modulu FTP proběhlo vůči 200 aplikačním serverům, z nichž se podařilo úspěšně spojit se 162 z těchto serverů. Výsledky testování těchto serverů jsou uvedeny v tabulce 5.1. Testování modulu probíhalo s následující konfigurací:

- bannergrabbing a příkazy `SYST` a `STAT`,
- anonymní autentizace,
- kontrola přístupu,
- útok FTP bounce pro skenování dostupnosti služby nacházející se v cloudovém prostředí platformy *DigitalOcean*[3].

Tab. 5.1: Výsledky testování modulu FTP

Test	**Počet úspěšných nálezů**
Bannergrabbing	162
Příkaz `SYST`	29
Příkaz `STAT`	15
Anonymní autentizace	8
Výskyt chyby při kontrole přístupu	7
Výpis adresáře	6
Právo zápisu	1
Právo čtení	1
Právo mazání	1
Úspěšná konfigurace bounce	1
Bounce služba přístupná	0

Testování modulu SSH proběhlo vůči 200 aplikačním serverům, z nichž se podařilo úspěšně spojit se 168 z těchto serverů. Výsledky testování těchto serverů jsou uvedeny v tabulce 5.2. Testování modulu probíhalo s následující konfigurací:

- bannergrabbing,
- zjištění autentizačních metod,

[3]DigitalOcean: `https://www.digitalocean.com/`

- sken nástrojem ssh-audit,
- porovnání veřejného klíče SSH serveru s repozitářem ssh-badkeys,
- pokus o přihlášení se soukromými klíči z repozitáře ssh-badkeys.

Tab. 5.2: Výsledky testování modulu SSH

Test	**Počet úspěšných nálezů**
Bannergrabbing	168
Autentizační metody	154
Úspěšný běh nástroje ssh-audit	165
Nálezy kryptografických protokolů	158
Nálezy veřejně známých zranitelností	94
Známý veřejný SSH klíč serveru	8
Výskyt známých soukromých SSH klíčů	0

Testování modulu SMTP proběhlo vůči 200 aplikačním serverům, z nichž se podařilo úspěšně spojit se 148 z těchto serverů. Výsledky testování těchto serverů jsou uvedeny v tabulce 5.3. Testování modulu probíhalo s následující konfigurací:

- bannergrabbing a příkaz `EHLO`,
- NTLM autentizace,
- kontrola enumeračních příkazů `EXPN`, `VRFY` a `RCPT TO`,
- kontrola blacklistů,
- kontrola open relay.

Tab. 5.3: Výsledky testování modulu SMTP

Test	**Počet úspěšných nálezů**
Bannergrabbing	148
Příkaz `EHLO`	148
NTLM autentizace	8
Blacklisting	39
Open relay	0
Enumerace uživatel příkazem `EXPN`	26
Enumerace uživatel příkazem `VRFY`	21
Enumerace uživatel příkazem `RCPT TO`	49

Testování modulu POP3 proběhlo vůči 200 aplikačním serverům, z nichž se podařilo úspěšně spojit se 151 z těchto serverů. Výsledky testování těchto serverů jsou uvedeny v tabulce 5.4. Testování modulu probíhalo s následující konfigurací:

- bannergrabbing a příkaz `CAPA`,

- anonymní autentizace,
- NTLM autentizace.

Tab. 5.4: Výsledky testování modulu POP3

Test	**Počet úspěšných nálezů**
Bannergrabbing	151
Příkaz `CAPA`	137
Anonymní autentizace	1
NTLM autentizace	8

Testování modulu IMAP proběhlo vůči 200 aplikačním serverům, z nichž se podařilo úspěšně spojit se 162 z těchto serverů. Výsledky testování těchto serverů jsou uvedeny v tabulce 5.5. Testování modulu probíhalo s následující konfigurací:

- bannergrabbing a příkazy `ID` a `CAPABILITY`,
- anonymní autentizace,
- NTLM autentizace.

Tab. 5.5: Výsledky testování modulu IMAP

Test	**Počet úspěšných nálezů**
Bannergrabbing	162
Příkaz `ID`	148
Příkaz `CAPABILITY`	162
Anonymní autentizace	1
NTLM autentizace	3

Závěr

V rámci diplomové práce byly analyzovány aplikační protokoly FTP, SSH, SMTP, POP3 a IMAP a jejich zranitelnosti. Na základě nalezených zranitelností byl pro každý z protokolů navržen diagram kontrolního seznamu sloužící pro provedení penetračního testera procesem penetračního testování daného aplikačního protokolu. Veškeré kroky uvedené v navržených diagramech byly následně implementovány formou textových dokumentů obsahujících návodné instrukce pro provedení penetračního testera daným testem. Rovněž byly obdobnou formou implementovány veškeré zranitelnosti, které lze identifikovat při provedení předešle zmíněných testů. U zranitelností byla popsána jejich specifika, bezpečnostní dopady a hodnocení jejich závažností dle metriky CVSS. Na základě navržených kontrolních seznamů byl dále navržen a v jazyce Python implementován vlastní automatizovaný nástroj pro penetrační testování aplikačních protokolů s názvem ptapptest. Nástroj ptapptest byl vyvinut jako modulární nástroj, kde pro každý ze zmíněných aplikačních protokolů byl navržen a implementován samostatný testovací modul. Návrhu každého z modulů předcházela analýza existujících dostupných nástrojů, z jejíchž výsledků návrh vycházel. Během implementace nástroje ptapptest byl navíc sestrojen další samostatný nástroj ptntlmauth, který byl do nástroje ptapptest integrován. Po dokončení implementace těchto nástrojů následovalo závěrečné testování nástroje ptapptest vůči dohromady 1000 reálným aplikačním serverům (200 pro každý modul nástroje) nalezeným s pomocí vyhledávače Shodan. Všechny stanovené cíle této diplomové práce byly splněny.

Literatura

[1] NÁRODNÍ ÚŘAD PRO KYBERNETICKOU A INFORMAČNÍ BEZPEČNOST *Vláda schválila Zprávu o stavu kybernetické bezpečnosti ČR za rok 2022.* [online]. [cit. 2023-12-13]. Dostupné z: <https://www.nukib.cz/cs/infoservis/aktuality/1981-vlada-schvalila-zpravu-o-stavu-kyberneticke-bezpecnosti-cr-za-rok-2022/>

[2] NÁRODNÍ ÚŘAD PRO KYBERNETICKOU A INFORMAČNÍ BEZPEČNOST *PENETRAČNÍ TESTOVÁNÍ – ÚVOD DO PROBLEMATIKY.* [online]. [cit. 2023-12-13]. Dostupné z: <https://nukib.cz/download/publikace/podpurne_materialy/2022-03-07_Penetracni-testovani_v1.1.pdf>

[3] MITNICK SECURITY *Understanding the main types of penetration testing.* [online]. [cit. 2023-12-13]. Dostupné z: <https://www.mitnicksecurity.com/blog/understanding-the-main-types-of-penetration-testing>

[4] YASAR, K. *pen testing (penetration testing).* [online]. [cit. 2023-12-13]. Dostupné z: <https://www.techtarget.com/searchsecurity/definition/penetration-testing>

[5] THE OWASP FOUNDATION a komunita přispěvatelů *The OWASP Testing Project.* [online]. [cit. 2023-12-13]. Dostupné z: <https://owasp.org/www-project-web-security-testing-guide/stable/2-Introduction/README#penetration-testing>

[6] SECURELAYER7 *Automated Vs Manual Pen-Testing – What's The Difference?* [online]. [cit. 2023-12-13]. Dostupné z: <https://blog.securelayer7.net/automated-vs-manual-pen-testing/>

[7] EC-COUNCIL *Understanding the Five Phases of the Penetration Testing Process.* [online]. [cit. 2023-12-13]. Dostupné z: <https://www.eccouncil.org/cybersecurity-exchange/penetration-testing/penetration-testing-phases/>

[8] OLNEY, M. *What Are the 5 Stages of Penetration Testing?* [online]. [cit. 2023-12-13]. Dostupné z: <https://insights.integrity360.com/what-are-the-5-stages-of-penetration-testing>

[9] JEŘÁBEK, J. *Komunikační technologie.* [online]. [cit. 2023-12-13]. Dostupné z: <https://www.vut.cz/www_base/priloha_fs.php?dpid=194797&skupina=dokument_priloha>

[10] ORACLE CORPORATION AND/OR ITS AFFILIATES *TCP/IP Protocol Architecture Model*. [online]. [cit. 2023-12-13]. Dostupné z: <https://docs.oracle.com/cd/E19683-01/806-4075/ipov-10/index.html>

[11] PENTEREP SECURITY S.R.O. [online]. [cit. 2024-04-24]. Dostupné z: <https://www.penterep.com/static/fbaa38dc6f93380e4fb1e1aa523d2ae3/d1b6d/platform_screenshots_07d5bc1b69.webp>

[12] LAZAROV, W.; MARTINÁSEK, Z. Web Platform for Comprehensive Penetration Testing. In *Proceedings II of the 28th Conference STUDENT EEICT 2022*. Brno University of Technology, Faculty of Electrical Engineering and Communication. april 2022. s. 88–91. doi:10.13164/eeict.2022.88. [online]. [cit. 2023-12-13].

[13] PENTEREP SECURITY S.R.O. [online]. [cit. 2024-04-24]. Dostupné z: <https://www.penterep.com/cs/penterep/vlastnosti/siroka-nabidka-kontrolnich-seznamu/>

[14] KÜMMEL, R.; LAZAROV, W.; MARTINÁSEK, Z. Penterep – inovativní platforma pro podporu manuálního penetračního testování. In *Sborník 25. konference ISSS*. Hradec Králové. 2023. s. 37–41. ISBN 978-80-907164-5-2. [online]. [cit. 2024-04-24]. Dostupné z: <https://isss.cz/wp-content/uploads/2023/05/isss2023-sbornik.pdf>

[15] POSTEL, J.; REYNOLDS, J. *File Transfer Protocol*. RFC 959. Říjen 1985. doi:10.17487/RFC0959. [online]. [cit. 2024-04-01]. Dostupné z: <https://www.rfc-editor.org/info/rfc959>

[16] FORD-HUTCHINSON, P. *Securing FTP with TLS*. RFC 4217. Říjen 2005. doi:10.17487/RFC4217. [online]. [cit. 2024-04-01]. Dostupné z: <https://www.rfc-editor.org/info/rfc4217>

[17] IBM *Explicit Versus Implicit FTP - SSL*. [online]. [cit. 2024-04-07]. Dostupné z: <https://www.ibm.com/support/pages/explicit-versus-implicit-ftp-ssl>

[18] HACKTRICKS a komunita přispěvatelů *21 - Pentesting FTP*. [online]. [cit. 2024-04-01]. Dostupné z: <https://book.hacktricks.xyz/network-services-pentesting/pentesting-ftp>

[19] BROMBERG, C. a komunita přispěvatelů *FTP*. [online]. [cit. 2024-04-07]. Dostupné z: <https://www.thehacker.recipes/infra/protocols/ftp>

[20] DEUTSCH, P.; EMTAGE, A.; MARINE, A. *How to Use Anonymous FTP*. RFC 1635. Květen 1994. doi:10.17487/RFC1635. [online]. [cit. 2024-04-01]. Dostupné z: <https://www.rfc-editor.org/info/rfc1635>

[21] SCHAIK, B. V. *TLS/SSL VULNERABILITIES*. [online]. [cit. 2024-04-07]. Dostupné z: <https://shadowtrackr.com/what-is-starttls>

[22] AKIMBO CORE *TLS/SSL VULNERABILITIES*. [online]. [cit. 2024-04-07]. Dostupné z: <https://akimbocore.com/article/tls-ssl-vulnerabilities/>

[23] PRODROMOU, A. *TLS Security 6: Examples of TLS Vulnerabilities and Attacks*. [online]. [cit. 2024-04-07]. Dostupné z: <https://www.acunetix.com/blog/articles/tls-vulnerabilities-attacks-final-part/>

[24] LONVICK, C. M.; YLONEN, T. *The Secure Shell (SSH) Transport Layer Protocol*. RFC 4253. Leden 2006. doi:10.17487/RFC4253. [online]. [cit. 2023-12-13]. Dostupné z: <https://www.rfc-editor.org/info/rfc4253>

[25] CLOUDFLARE, INC *What is SSH? | Secure Shell (SSH) protocol*. [online]. [cit. 2023-12-13]. Dostupné z: <https://www.cloudflare.com/learning/access-management/what-is-ssh/>

[26] YLONEN, T. *SSH Protocol – Secure Remote Login and File Transfer*. [online]. [cit. 2023-12-13]. Dostupné z: <https://www.ssh.com/academy/ssh/protocol>

[27] YLONEN, T. *Basic overview of SSH Keys*. [online]. [cit. 2023-12-13]. Dostupné z: <https://www.ssh.com/academy/ssh-keys>

[28] HACKTRICKS a komunita přispěvatelů *22 - Pentesting SSH/SFTP*. [online]. [cit. 2023-12-13]. Dostupné z: <https://book.hacktricks.xyz/network-services-pentesting/pentesting-ssh>

[29] RAPID7 a komunita přispěvatelů *SSH Bad Keys*. [online]. [cit. 2023-12-13]. Dostupné z: <https://github.com/rapid7/ssh-badkeys/tree/master>

[30] MOORE, K.; NEWMAN, C. *Cleartext Considered Obsolete: Use of Transport Layer Security (TLS) for Email Submission and Access*. RFC 8314. Leden 2018. doi:10.17487/RFC8314. [online]. [cit. 2024-04-19]. Dostupné z: <https://www.rfc-editor.org/info/rfc8314>

[31] KLENSIN, J. C. *Simple Mail Transfer Protocol*. RFC 5321. Říjen 2008. doi:10.17487/RFC5321. [online]. [cit. 2024-04-08]. Dostupné z: <https://www.rfc-editor.org/info/rfc5321>

[32] SIEMBORSKI, R.; MELNIKOV, A. *SMTP Service Extension for Authentication*. RFC 4954. Červenec 2007. doi:10.17487/RFC4954. [online]. [cit. 2024-04-08]. Dostupné z: <https://www.rfc-editor.org/info/rfc4954>

[33] FREED, N.; BORENSTEIN, N. S. *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*. RFC 2045. Listopad 1996. doi:10.17487/RFC2045. [online]. [cit. 2024-04-08]. Dostupné z: <https://www.rfc-editor.org/info/rfc2045>

[34] CLOUDFLARE, INC *What is the Simple Mail Transfer Protocol (SMTP)?* [online]. [cit. 2024-04-08]. Dostupné z: <https://www.cloudflare.com/learning/email-security/what-is-smtp/>

[35] CLOUDFLARE, INC *What SMTP port should be used? Port 25 or 587?* [online]. [cit. 2024-04-08]. Dostupné z: <https://www.cloudflare.com/learning/email-security/smtp-port-25-587/>

[36] MALEK, P. *Everything You Need to Know About SMTP Security*. [online]. [cit. 2024-04-09]. Dostupné z: <https://mailtrap.io/blog/smtp-security/>

[37] KITTERMAN, S. *Sender Policy Framework (SPF) for Authorizing Use of Domains in Email, Version 1*. RFC 7208. Duben 2014. doi:10.17487/RFC7208. [online]. [cit. 2024-04-08]. Dostupné z: <https://www.rfc-editor.org/info/rfc7208>

[38] CLOUDFLARE, INC *What is a DNS SPF record?* [online]. [cit. 2024-04-08]. Dostupné z: <https://www.cloudflare.com/learning/dns/dns-records/dns-spf-record/>

[39] KUCHERAWY, M.; CROCKER, D.; HANSEN, T. *DomainKeys Identified Mail (DKIM) Signatures*. RFC 6376. Září 2011. doi:10.17487/RFC6376. [online]. [cit. 2024-04-08]. Dostupné z: <https://www.rfc-editor.org/info/rfc6376>

[40] CLOUDFLARE, INC *What is a DNS DKIM record?* [online]. [cit. 2024-04-08]. Dostupné z: <https://www.cloudflare.com/learning/dns/dns-records/dns-dkim-record/>

[41] KUCHERAWY, M.; ZWICKY, E. *Domain-based Message Authentication, Reporting, and Conformance (DMARC)*. RFC 7489. Březen 2015. doi:10.17487/RFC7489. [online]. [cit. 2024-04-08]. Dostupné z: <https://www.rfc-editor.org/info/rfc7489>

[42] CLOUDFLARE, INC *What is a DNS DMARC record?* [online]. [cit. 2024-04-08]. Dostupné z: <https://www.cloudflare.com/learning/dns/dns-records/dns-dmarc-record/>

[43] HACKTRICKS a komunita přispěvatelů *25,465,587 - Pentesting SMTP/s*. [online]. [cit. 2024-04-08]. Dostupné z: <https://book.hacktricks.xyz/network-services-pentesting/pentesting-smtp>

[44] OZTURK, F. *What is SMTP Open Mail Relay Vulnerability?* [online]. [cit. 2024-04-23]. Dostupné z: <https://threatmon.io/blog/what-is-smtp-open-mail-relay-vulnerability/>

[45] LONGIN, T. *SMTP Smuggling - Spoofing E-Mails Worldwide*. [online]. [cit. 2024-04-20]. Dostupné z: <https://sec-consult.com/blog/detail/smtp-smuggling-spoofing-e-mails-worldwide/>

[46] ZORZ, Z. *Attackers are exploiting IMAP to bypass MFA on Office 365, G Suite accounts*. [online]. [cit. 2024-04-08]. Dostupné z: <https://www.helpnetsecurity.com/2019/03/20/imap-based-password-spraying/>

[47] THE OWASP FOUNDATION a komunita přispěvatelů *Testing for IMAP SMTP Injection*. [online]. [cit. 2024-04-08]. Dostupné z: <https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/10-Testing_for_IMAP_SMTP_Injection>

[48] ROSE, M. T.; MYERS, J. G. *Post Office Protocol - Version 3*. RFC 1939. Květen 1996. doi:10.17487/RFC1939. [online]. [cit. 2024-04-19]. Dostupné z: <https://www.rfc-editor.org/info/rfc1939>

[49] NEWMAN, C. *Using TLS with IMAP, POP3 and ACAP*. RFC 2595. Červen 1999. doi:10.17487/RFC2595. [online]. [cit. 2024-04-19]. Dostupné z: <https://www.rfc-editor.org/info/rfc2595>

[50] SIEMBORSKI, R.; MENON-SEN, A. *The Post Office Protocol (POP3) Simple Authentication and Security Layer (SASL) Authentication Mechanism*. RFC 5034. Červenec 2007. doi:10.17487/RFC5034. [online]. [cit. 2024-04-19]. Dostupné z: <https://www.rfc-editor.org/info/rfc5034>

[51] HACKTRICKS a komunita přispěvatelů *110,995 - Pentesting POP*. [online]. [cit. 2024-04-19]. Dostupné z: <https://book.hacktricks.xyz/network-services-pentesting/pentesting-pop>

[52] OPEN TEXT *Mail Command Injection: POP3*. [online]. [cit. 2024-04-20]. Dostupné z: <https://vulncat.fortify.com/en/detail?id=desc.dataflow.java.mail_command_injection_pop3>

[53] MELNIKOV, A.; LEIBA, B. *Internet Message Access Protocol (IMAP) - Version 4rev2*. RFC 9051. Srpen 2021. doi:10.17487/RFC9051. [online]. [cit. 2023-12-13]. Dostupné z: <https://www.rfc-editor.org/info/rfc9051>

[54] ZEILENGA, K. *Anonymous Simple Authentication and Security Layer (SASL) Mechanism*. RFC 4505. Červen 2006. doi:10.17487/RFC4505. [online]. [cit. 2023-12-13]. Dostupné z: <https://www.rfc-editor.org/info/rfc4505>

[55] MYERS, J. G. *IMAP4 Authentication Mechanisms*. RFC 1731. Prosinec 1994. doi:10.17487/RFC1731. [online]. [cit. 2023-12-13]. Dostupné z: <https://www.rfc-editor.org/info/rfc1731>

[56] HACKTRICKS a komunita přispěvatelů *143,993 - Pentesting IMAP*. [online]. [cit. 2023-12-13]. Dostupné z: <https://book.hacktricks.xyz/network-services-pentesting/pentesting-imap>

[57] SHODAN *Shodan Search Engine*. [online]. [cit. 2024-05-15]. Dostupné z: <https://www.shodan.io/dashboard>

Seznam symbolů a zkratek

CR Carriage Return

CVE Common Vulnerabilities and Exposures

CVSS Common Vulnerability Scoring System

DKIM DomainKeys Identified Mail

DMARC Domain-based Message Authentication, Reporting, and Conformance

DNS Doman Name System

ESMTP Extended Simple Mail Transfer Protocol

FTP File Transfer Protocol

FTPS File Transfer Protocol Secure

IMAP Internet Message Access Protocol

IMAPS Internet Message Access Protocol Secure

IP Internet Protocol

LF Line Feed

NVD National Vulnerability Database

OSINT Open Source Intelligence

POP3 Post Office Protocol 3

POP3S Post Office Protocol 3 Secure

PyPI Python Package Index

SMTP Simple Mail Transfer Protocol

SMTPS Simple Mail Transfer Protocol Secure

SPF Sender Policy Framework

SSH Secure Shell

SSL Secure Sockets Layer

TCP Transmission Control Protocol

TLS Transport Layer Security

UDP User Datagram Protocol

A Obsah elektronické přílohy

Elektronická příloha obsahuje vytvořené dokumenty testů a zranitelností (implementace kontrolních seznamů), zdrojové kódy vyvinutých nástrojů ptapptest a ptntlmauth, kopii diplomové práce, skript použitý při testování nástroje společně s tabulkami výsledků testování a textový soubor s komentářem k instalaci nástrojů.

```
/ ..... Kořenový adresář přiloženého archivu
├── penterep_texty/ ..... Kořenový adresář dokumentů testů a zranitelností
├── ptapptest/ ..... Kořenový adresář nástroje ptapptest
│   ├── ptapptest/ ..... Zdrojový kód nástroje
│   │   ├── modules/ ..... Zdrojový kód modulů nástroje
│   │   │   ├── utils/
│   │   │   ├── __init__.py
│   │   │   ├── _base.py
│   │   │   ├── ftp.py
│   │   │   ├── imap.py
│   │   │   ├── pop3.py
│   │   │   ├── smtp.py
│   │   │   └── ssh.py
│   │   ├── __init__.py
│   │   ├── _version.py
│   │   └── ptapptest.py
│   ├── LICENSE
│   ├── pyproject.toml
│   ├── README.md
│   ├── requirements.txt
│   └── setup.py
├── ptntlmauth/ ..... Kořenový adresář nástroje ptntlmauth
│   ├── ptntlmauth/ ..... Zdrojový kód nástroje
│   │   ├── __init__.py
│   │   ├── _version.py
│   │   └── ptntlmauth.py
│   ├── LICENSE
│   ├── pyproject.toml
│   ├── README.md
│   ├── requirements.txt
│   └── setup.py
├── DP_Vasicek.pdf ..... Diplomová práce (PDF)
├── ftp.csv ..... Tabulka výsledků testování modulu FTP
├── imap.csv ..... Tabulka výsledků testování modulu IMAP
├── pop3.csv ..... Tabulka výsledků testování modulu POP3
├── readme.md ..... Komentář k instalaci nástrojů ptapptest a ptntlmauth
├── smtp.csv ..... Tabulka výsledků testování modulu SMTP
├── ssh.csv ..... Tabulka výsledků testování modulu SSH
└── test.py ..... Skript použitý pro testování nástroje
```