FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Systém pro detekci a monitoring verzí webových aplikací v interních síťových prostředích
Student:	Petr Morávek
Vedoucí:	Ing. Jan Matoušek
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2024/2025

Pokyny pro vypracování

V rozsáhlých sítích není jednoduché sledovat, jaké webové aplikace jsou v provozu na jednotlivých zařízeních. Výstupem této práce je nástroj sledující verze webových aplikacích nasazených na zařízeních v počítačové síti a upozorňující administrátora na zastaralé verze a dostupné aktualizace.

Pokyny k vypracování:
1) Analyzujte problematiku údržby webových aplikací v počítačové síti.
2) Analyzujte požadavky na nástroj, který by mohl administrátorovi pomoct se správou a údržbou většího množství předem známých i neznámých webových aplikací nasazených v síti, proveďte rešerši dostupných nástrojů, pokud existují.
3) Na základě analýzy navrhněte nástroj řešící zjištěné požadavky.
4) Implementujte funkční prototyp navrženého nástroje.
5) Prototyp zdokumentujte a podrobte vhodným testům.
6) Získané zkušenosti shrňte a navrhněte možná rozšíření.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 29. ledna 2024 v Praze.

Bakalářská práce

SYSTÉM PRO DETEKCI A MONITORING VERZÍ WEBOVÝCH APLIKACÍ V INTERNÍCH SÍŤOVÝCH PROSTŘEDÍCH

Petr Morávek

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Jan Matoušek
16. května 2024

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chtěl bych poděkovat především svému vedoucímu, panu Ing. Janu Matouškovi, za vedení této bakalářské práce, skvělou komunikaci a poskytnutí konzultací. Dále bych chtěl poděkovat Václavu Podlipnému, Martinu Hladíkovi, Vojtěchu Svobodovi a firmě Quanti s. r. o. za inspiraci a příležitost pracovat na tomto tématu. V neposlední řadě bych chtěl také poděkovat mé rodině, která mě podporovala od samého začátku bakalářského studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 16. května 2024

Abstrakt

Tato bakalářská práce se zaobírá problematikou detekce a monitoringu webových aplikací nasazených v interních síťových prostředích s operačním systémem Debian. V analytické části byly zkoumány nástroje, které umožňují sken síťového rozsahu a automatickou detekci webových aplikací. Na základě výsledků analýzy byl vytvořen návrh aplikace splňující požadavky zadavatele, jímž je firma Quanti s. r. o. Aplikace je schopna prozkoumat zadaný síťový rozsah a detekovat v něm nasazené běžící webové aplikace. Pro objevené webové aplikace je nástroj schopen detekovat jejich aktuální verzi, pokud je konkrétní webová aplikace součástí podporovaného seznamu. Uživatel je schopen seznam rozšířit a zadefinovat tak další pravidla, použitá při další detekci webových aplikací. Nástroj je dále schopen srovnávat aktuální verzi s nejnovějšími dostupnými verzemi a informovat tak uživatele o vhodné aktualizaci. Aplikace prozkoumává zařízení pomocí technologie SSH a data z webových adres získává za pomoci web scrapingu, avšak její zdrojový kód je navržený tak, aby mohla být v budoucnu snadno rozšířena o jiné postupy. Aplikace je implementována v jazyce Python a s uživatelem komunikuje pomocí API vytvořeného pomocí frameworku Flask. Aplikace byla řádně otestovaná v izolovaném prostředí, které nasimulovalo síťový prostor s běžícími webovými aplikacemi.

Klíčová slova detekce a monitoring, webové aplikace, verze webových aplikací, sken sítě, Debian, síťový rozsah, Python, Docker

Abstract

This bachelor's thesis deals with the problematic of detection and monitoring web applications deployed on internal network environments with Debian operation system. In the analysis part, tools that enable network range scanning and automatic detection of web applications were researched. Based on the results of the analysis, a design of the application was created to meet the requirements of the client, which is the company Quanti s. r. o. The application is able to explore the specified network range and detect web applications running in it. For discovered web applications, the tool is able to detect their current version if the particular web application is part of the supported list. The user is able to extend the list and define additional rules to be used for further detection of web applications. The tool is also able to compare the current version with the latest available versions and thus inform the user about the appropriate update. The application explores the device using SSH technology and retrieves data from web addresses using web scraping, but its source code is designed to be easily extended by other methods in the future. The application is implemented in Python and communicates with the user using an API created using the Flask framework. The application was properly tested in an isolated environment that simulated a network space with running web applications.

Keywords detection and monitoring, web applications, version of web applications, network scan, Debian, network range, Python, Docker

Seznam zkratek

API	Application Programming Interface
APM	Application Monitoring Software
CDN	Content Delivery Network
CI	Continuous Integration
CSS	Cascading Style Sheet
DNS	Domain Name System
DOM	Document Object Model
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IP	Internet Protocol
JSON	JavaScript Object Notation
NAT	Network Address Translation
OSI	Open Systems Interconnection
SSH	Secure Shell
SSL	Secure Sockets Layer
URL	Uniform Resource Locator
YAML	YAML Ain't Markup Language

Úvod

Používání webových aplikací se stalo součástí běžného života většiny z nás. Výhody, které přináší, však s sebou nesou vynaložení značného úsilí ze strany vývojářů a administrátorů takový software vymyslet a v neposlední řadě také udržet ve stabilním a bezpečném stavu. Tento proces je často velmi komplexní a vyžaduje spolupráci mnoha lidí. Lidé nejsou bezchybní, a tak se snaží přijít s vytvořením systémů, které jim ušetří práci a daný proces zautomatizují.

Cílem této práce je najít způsob, jak co nejlépe zmapovat zadaný síťový rozsah a odhalit v něm kromě samotných běžících webových aplikací hostovaných v této síti i jejich verze, které poskytují administrátorovi sítě informaci o aktuálnosti. Jsou zde zkoumány nástroje, které mají blízko požadavkům zadavatele, kterým je samotná společnost Quanti s. r. o. a především správci její sítě, kteří jsou zodpovědní za bezpečný a spolehlivý provoz všech jejich služeb. V případě, že není známo žádné řešení splňující kladené nároky, pak je za cíl považován návrh a implementace vlastního řešení pro výše popsaný problém.

Výstupem této práce je nástroj sledující názvy a verze webových aplikací nasazených na zařízeních ve zkoumané počítačové síti. Nástroj je schopný nalezené verze porovnávat s aktuálně dostupnými verzemi. Hlavní myšlenkou je administrátorovi sítě poskytovat toto srovnání, které může sloužit jako přehled o nutných dostupných aktualizacích a předejít tak bezpečnostním problémům, které jsou spjaty s provozováním neaktuálního softwaru. V neposlední řadě také administrátorovi umožní v síti nalézt webové aplikace, o kterých dosud neměl povědomí. Příslušné informace jsou podávány v přehledné a praktické podobě, kterou může administrátor sítě použít pro své účely.

Výsledek této práce je prospěšný právě pro zmiňované administrátory, kteří mají na starost správu a chod webových aplikací. Jejich úkolem je zajistit, aby dané aplikace splňovaly předem určený cíl a nedocházelo k případným výpadkům provozu nebo incidentům vedoucím k ohrožení kyberbezpečnosti uživatelů.

Dalším přínosem je také poskytnutí přehledu o zkoumané internetové síti, kterou administrátoři spravují. V takové síti není jednoduché sledovat, jaké webové aplikace jsou v provozu na jednotlivých zařízeních (serverech).

Tato práce vznikla ve spolupráci s firmou Quanti s. r. o., jejíž administrátorský tým se podílel na tvorbě zadání práce. Řešení se zaměřuje na webové aplikace nasazené na serverech s operačním systémem Debian.

Práce je strukturována do sedmi kapitol. První kapitola se věnuje teoretické části nutné k pochopení dalších fází. Jádrem kapitoly je moderní vývoj a provoz webových aplikací.

Druhá kapitola se zaměřuje na konkrétní požadavky pro řešení, průzkum dostupných nástrojů a hluboký průnik do problému včetně analýzy technických záležitostí, které jsou nezbytné pro další fáze této práce.

Třetí kapitola je věnována návrhu požadovaného nástroje. Čtenářovi je vysvětlen základní princip a architektura nástroje, která splňuje kladené požadavky. Jsou zde definovány entity

a rozhraní, pro které je navržena realizace pomocí konkrétních tříd. Kapitola je obohacena o diagramy, které pomohou k vizualizaci návrhu.

Čtvrtá kapitola se věnuje implementaci daného návrhu. Jsou představeny konkrétní postupy a technologie, které vedou k fungujícímu prototypu aplikace. V kapitole jsou popsány objevené překážky a jejich řešení. Text je doplňován ukázkami kódu.

Pátá kapitola je zaměřena na testování zhotovené aplikace. Kromě ukázek konkrétních testů je zde popsáno vytvoření izolovaného prostředí sloužícího k nasimulování reálných serverů.

V šesté kapitole jsou zmíněna některá možná rozšíření implementované aplikace a sedmá kapitola slouží jako závěrečné shrnutí této bakalářské práce.

Kapitola 1
Teoretická část

V této kapitole jsou vysvětleny a popsány základní technologie a principy spojeny s webovými aplikacemi a jejich provozem. Porozumění této kapitoly je kritické pro pochopení navazujících kapitol.

1.1 Webové aplikace

Webová aplikace je počítačový program, který využívá webové prohlížeče a různé webové technologie k provádění úkolů v prostředí internetu. [1]

Webové aplikace využívají kombinaci technik, které se starají o ukládání a načítání informací na straně serveru a prezentování informací uživatelům. To umožňuje uživatelům komunikovat se společností pomocí online formulářů, systémů správy obsahu, nákupních košíků atd. Kromě toho aplikace umožňují uživatelům vytvářet dokumenty, spolupracovat na projektech a sdílet informace bez ohledu na umístění nebo zařízení. [1]

Jak fungují webové aplikace

Takto vypadá typický průběh komunikace mezi uživatelem a aplikací:

1. Uživatel vytvoří požadavek na webový server přes internet např. prostřednictvím webového prohlížeče.
2. Webový server předá tento požadavek příslušnému aplikačnímu serveru webové aplikace (Tento proces je detailněji popsán v pozdější fázi textu viz 1.2).
3. Aplikační server provede potřebnou úlohu - například dotaz do databáze nebo zpracování dat - a poté vygeneruje požadovaný výsledek.
4. Aplikační server odešle webovému serveru tyto výsledky.
5. Webový server pošle tato data zpět klientovi, který si je za pomocí prohlížeče zobrazí v přijatelné uživatelské podobě. [1]

Klíčové výhody

Multiplatformnost: Webové aplikace fungují na různých platformách bez ohledu na operační systém nebo typ zařízení.

Náročnost: Nejsou nainstalovány na pevném disku, čímž se eliminují prostorová omezení.

Kompatibilita: Všichni uživatelé mají přístup ke stejné verzi, čímž se eliminují jakékoli problémy s kompatibilitou. [1]

Příklady

- **E-commerce platformy** (např. Amazon, eBay)
- **Informační portály** (např. zpravodajské weby)
- **Sociální sítě** (např. Facebook, X)
- **Video platformy** (např. YouTube, Netflix)
- **Online nástroje a služby** (např. Google Docs, GitLab)

1.1.1 HTTP protokol

Ke komunikaci mezi klientem (uživatelem) a serverovou částí webové aplikace (aplikací běžící na vzdáleném zařízení posílající klientovi data, která si zobrazuje v prohlížeči) se běžně používá HTTP, respektive HTTPS protokol. Klienti a servery komunikují výměnou jednotlivých zpráv[1]. Zprávy odesílané klientem, obvykle webovým prohlížečem, se nazývají požadavky a zprávy odesílané serverem odpovědi. Ve skutečnosti se ale může mezi klientem a serverem vyskytovat více zařízení: směrovače, modemy, load balancery atd. [2]

HTTP je protokol sedmé a poslední síťové vrstvy OSI, a proto jsou prostředníci mezi klienty a servery skryti v nižších vrstvách. [3] Dá se říci, že je to tak správně, protože z hlediska samotného protokolu a aplikační vrstvy je pro uživatele podstatná primárně cílová adresa, kterou může použít k odeslání požadavku, stejně tak jako pro server adresa klienta, kterou potřebuje znát k odeslání odpovědi.

1.1.2 Obsah komunikace

Klient a server si mezi sebou posílají data. Jedním z nejzákladnějších formátů pro tato data je jazyk *HTML*. Jeho historie sahá do 90. let minulého století a slouží k vytváření stránek a publikaci dokumentů na internetu. HTML data jsou rozdělena do částí, které se nazývají elementy. Celkově pak tento obsah tvoří jakýsi strom textových informací nazývaný DOM (Document Object Model). [4]

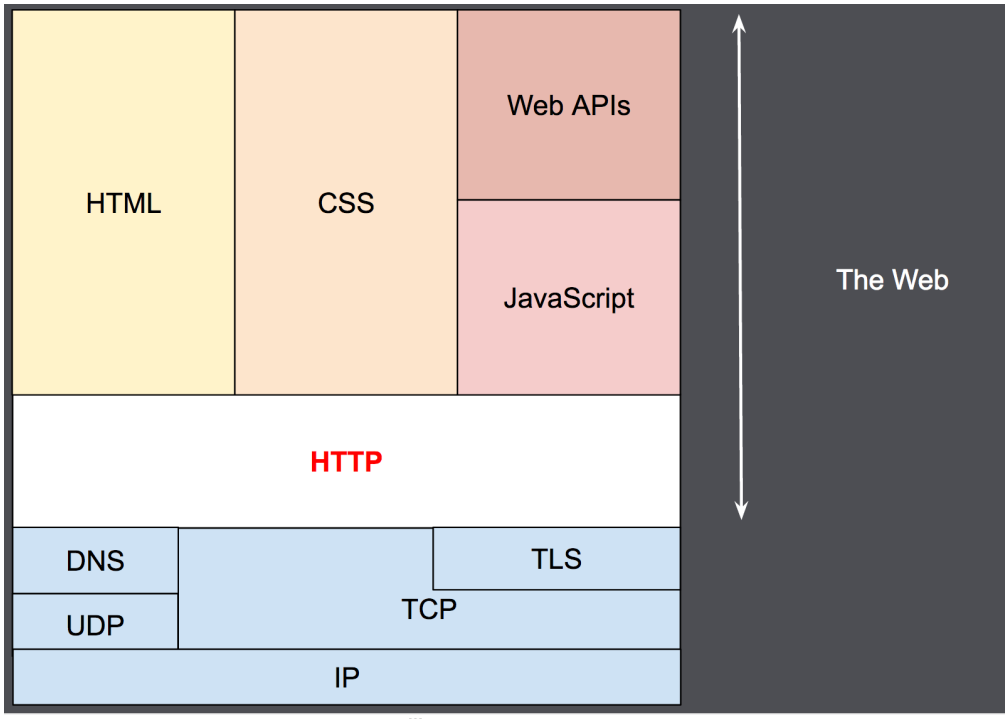
Zatímco jazyk HTML se používá k definování struktury a sémantiky obsahu, jazyk CSS slouží k jeho stylování a rozvržení. Pomocí CSS lze například měnit písmo, barvu, velikost a rozestupy obsahu, rozdělit jej do více sloupců nebo přidat animace a další ozdobné prvky. [5]

Většina dnešních webových stránek a aplikací by se neobešla ani bez technologie JavaScript. JavaScript je interpretovaný (nebo just-in-time kompilovaný) programovací jazyk. [6]

Počítačovému programu, který interpretuje kód JavaScriptu, se říká JavaScript engine. Tento engine je zodpovědný za provádění kódu. Každý moderní prohlížeč má JavaScript engine, který je schopný zmiňovaný kód provádět a vyhodnocovat. [7]

Tomuto principu, kdy klient k plnohodnotnému zobrazení stránky nejprve potřebuje na pozadí vyhodnotit určitý kus kódu, se říká *Client-side rendering*. Naopak pokud k veškeré logice vyhodnocení požadavku dojde na straně serveru, který již pošle úplný HTML obsah, jedná se o *Server-side rendering*. [8] Často se lze setkat s jakýmsi kompromisem, kdy část informací je poslána ve formě finálního HTML obsahu a část musí klient ještě zpracovat ve svém prohlížeči.

[1] Nejedná se o oboustranný proud dat. Za tímto účelem se zpravidla využívá jiného komunikačního protokolu jako je např. WebSockets.



Obrázek 1.1 Obsah a vrstvy webu [2]

1.1.3 Bezpečnostní zranitelnosti

Webové aplikace mohou čelit celé řadě útoků v závislosti na cílech útočníka, druhu cílové stránky a konkrétních bezpečnostních nedostatcích aplikace. Mezi běžné typy útoků patří např:

- **Cross site scripting (XSS):** XSS je typ útoku, který umožňuje útočníkovi vkládat skripty na stranu klienta, které se v daný okamžik spustí, a útočník tak získá přímý přístup k důležitým informacím, vydává se za uživatele nebo ho přiměje k odhalení důležitých informací.
- **SQL injection (SQi):** SQi je metoda, při kterém útočník zneužívá zranitelnosti ve způsobu, jakým databáze propojená s webovou aplikací provádí vyhledávací dotazy. Útočníci pomocí SQi získávají přístup k neoprávněným informacím, upravují nebo vytvářejí nová uživatelská oprávnění nebo jinak manipulují s citlivými daty či je ničí.
- **Útoky typu DoS (Denial-of-service) a DDoS (Distributed Denial-of-service):** Útočníci jsou schopni přetížit cílový server. Server přestane být schopen efektivně zpracovávat příchozí požadavky a nakonec dojde k přerušení obsluhy pro legitimní uživatele. [9]

1.1.4 Sémantické verzování

Software v průběhu svého životního cyklu prochází změnami, vylepšeními a opravami. Je tedy označován verzemi. Jedno z řešení, které se nabízí, je takzvané *Sémantické verzování*, které přichází s jednoduchou sadou pravidel a požadavků, které určují, jak se čísla verzí přidělují a zvyšují.

Verzování je formátu **X.Y.Z (Major.Minor.Patch)**, kde je každá z těchto tří částí reprezentována nějakým číslem. [10]

- **X** je **hlavní (major) verze**. Zvyšuje se inkrementálně o jedna a zvýšení vynuluje jak patch, tak minor verzi. Pokud je aktuální verze 1.2.3, pak zvýšení major verze povede k 2.0.0. Hodnota X se zvyšuje při změně stávajícího rozhraní, které s předchozím již nebude kompatibilní.
- **Y** reprezentuje **minor verzi**. Používá se pro nasazení nové funkce do systému. Zvýšení vynuluje patch verzi. Pokud je aktuální verze 1.2.3, pak zvýšení povede na 1.3.0. Hodnota Y se zvyšuje při implementaci nových funkcionalit, které jsou zpětně kompatibilní v rámci stejné major verze.
- **Z** reprezentuje **patch verzi**, která se používá k opravě chyb. Při aktualizacích patch verzí nedochází ke změnám funkčnosti. Pokud je aktuální verze 1.2.3, tak aktualizace povede k 1.2.4. [10]

1.1.5 Způsoby nasazení aplikací

Nasazení aplikací lze rozdělit zpravidla na dva způsoby.

1.1.5.1 Self-hosting

Self-managed hosting nebo *self-hosting* je pojem používaný k pojmenování softwaru, který je nainstalován a spravován na soukromém serveru namísto používání služby mimo vlastní kontrolu. [11, 12] Tento způsob přináší větší kontrolu a flexibilitu, ale také větší odpovědnost za správu a údržbu. [13] Například k vytvoření self-hosted webové stránky může být potřeba:

- Zakoupit název domény u registrátora domén.
- Získat hostingový účet od vhodného poskytovatele.
- Nakonfigurovat záznamy DNS.[2]
- Spravovat certifikáty SSL.[3]
- Pravidelně kontrolovat nové aktualizace nebo zranitelnosti. [13]

1.1.5.2 Managed-hosting

Jedná se o typ hostingu, u kterého se stará o většinu úkolů správy a údržby serveru (instalace, aktualizace, zabezpečení, zálohování, monitorování a podpora) externí poskytovatel. Zákazník se musí soustředit pouze na své aplikace a jejich obsah. Poskytovatel hostingu také poskytuje další funkce a služby, jako je optimalizace výkonu, škálovatelnost, vyrovnávání zátěže, ukládání do mezipaměti atd. Zákazník má menší kontrolu nad serverem a jeho konfigurací, ale také méně starostí a rizik. [13]

Výhody self-hostingu

Kontrola: Pokud si software hostujete sami, máte nad svými daty plnou kontrolu. Kritická data jsou uchovávána přímo u vás a žádná jiná strana k nim nemá přístup.

Offline přístup: K přístupu k datům a softwaru na lokálně spravovaných zařízeních není potřeba internetové připojení.

[2] Domain Name System je hierarchický a distribuovaný systém pojmenování počítačů, služeb a dalších zdrojů na internetu nebo jiných sítích s internetovým protokolem (IP). Především převádí snadno zapamatovatelná doménová jména na číselné IP adresy potřebné pro vyhledávání a identifikaci počítačových služeb a zařízení pomocí základních síťových protokolů. [14]

[3] SSL/TLS používá certifikáty k vytvoření šifrovaného spojení mezi serverem a klientem. To umožňuje bezpečný přenos citlivých informací, jako jsou například údaje o kreditní kartě, přes internet. [15]

Nezávislost: Svěřit kritická podniková data dodavateli softwaru není snadné rozhodnutí. Pokud vznikne závislost na nějakém konkrétním poskytovateli, který posléze přestane službu poskytovat nebo ji poskytuje ve špatné kvalitě, je velmi obtížné a složité migrovat data do jiného prostředí. [16]

Nevýhody self-hostingu

Technické dovednosti: Samostatně spravovaný server vyžaduje technické dovednosti a znalosti pro správu a údržbu serveru. Je potřeba se vypořádat se složitými úkoly, jako je instalace, monitorování a řešení problémů. Administrátor musí také znát základní operační systém a software, který se na serveru používá.

Údržba: Self-hosting hosting vyžaduje více času a úsilí na udržení hladkého a efektivního chodu serveru.

Podpora: K vyřešení všech problémů nebo výzev se lze spoléhat pouze na své vlastní schopnosti a zdroje. [13]

1.1.6 Údržba

Zadavatelé softwarových systémů se často domnívají, že jakmile je jejich softwarový produkt vydán, není už co řešit. Ale tak jednoduché to není. Webové aplikace neustále rostou a vyvíjejí se a je třeba je udržovat. Udržování aplikací bezpečných, stabilních, užitečných a aktuálních vyžaduje čas. [17]

1.1.6.1 Zálohování

Zálohy lze použít k obnově dat po jejich ztrátě v důsledku vymazání nebo poškození dat nebo k obnově dat z dřívější doby. Zálohování představuje jednoduchou formu obnovy po havárii. [18][19]

1.1.6.2 Aktualizace

Aktualizace slouží k záměrným změnám v softwaru. Odstraňují chyby a mění funkcionality či uživatelská rozhraní. Některé změny jsou vítané, dokonce očekávané, a někdy jsou změny subjektivně nežádoucí, což vede k tomu, že se uživatelé aktualizacím vyhýbají. Pokud uživatelé aktualizace odkládají, může to mít pro jejich zařízení závažné bezpečnostní důsledky. Aktualizace jsou jedním z hlavních mechanismů pro opravu odhalených zranitelností. [20] Hlavní rizika spojená s neaktuálním softwarem:

Bezpečnost: Útočníci nutí vydavatele softwaru být vždy ve střehu a zveřejňovat aktualizace, jakmile objeví nějakou zranitelnost, nejlépe dříve, než někdo zlomyslný najde způsob, jak ji zneužít a způsobit škodu.

Bugy: Vývojáři softwaru nejsou neomylní a dělají chyby. Navíc oprava chyb může v některých případech přinést nové chyby. Co je však důležitější, bez ohledu na to, jak dobře je software testován, některé z těchto chyb nebudou odhaleny přímo ve vývoji a nakonec se dostanou do oficiálního vydání softwaru. Většina těchto chyb ovlivní uživatele pouze za velmi specifických a vzácných okolností. Proto původně zůstávají nepovšimnuty. Existuje však riziko, že jedna konkrétní akce, kterou je třeba provést, bude mít kvůli softwarové chybě neočekávané výsledky.

Aktuálnost: Aktualizace softwaru nemusí řešit pouze chyby a bezpečnostní zranitelnosti, ale někdy pouze přidává nové nebo vylepšuje stávající funkce, například v reakci na nové právní požadavky nebo oborové směrnice. Pokud se rozhodnete neinstalovat aktualizace softwaru, vystavujete se riziku, že budete používat zastaralý software, který nemůže využívat nejnovější vylepšení nebo změny.

Nekompatibilita: Zastaralý software nemusí také očekávaně a spolehlivě pracovat s jiným (řádně aktualizovaným) softwarem. Například někteří uživatelé používající aktualizovaný software mohou mít problémy s kompatibilitou při výměně dat s jinými uživateli, kteří používají starší software. [21]

1.1.6.3 Monitorování

Monitorování aplikací je proces, který poskytuje informace o výkonu aplikace v reálném čase a umožňuje vývojářům rychle reagovat na technické problémy. S pomocí nástrojů pro monitorování aplikací mohou IT odborníci a vývojáři snadno sledovat její dostupnost, využití zdrojů, chyby a další důležité aspekty, které mají vliv na zážitek koncových uživatelů. K monitorování softwaru je využíván APM (Application Monitoring Software). Mezi jeho funkce patří například měření výkonu aplikace, detekce chyb nebo sledování využití zdrojů zařízení. [22]

1.2 Webové servery

Termín webový server může označovat hardwarový nebo softwarový prvek, případně obojí dohromady. [23]

Pokud jde o hardware, webový server je zařízení - počítač, který uchovává software webového serveru a soubory, které jsou součástí webových stránek (například dokumenty HTML, obrázky, soubory CSS a soubory JavaScript). Webový server je připojený k internetu a poskytuje fyzickou výměnu dat s jinými zařízeními připojenými k webu. [23]

Pokud se bavíme o webovém serveru jakožto softwarovém prvku, tak ten se skládá z několika částí, které řídí odesílání souborů klientovi. Jedna z částí je HTTP server. Server HTTP je software, který rozumí adresám URL (webovým adresám) a protokolu HTTP. Je přístupný prostřednictvím URL webových stránek (doménových jmen) a doručuje obsah těchto hostovaných webových stránek do zařízení koncového uživatele. [23]

- **Statický webový server** odesílá své hostované soubory do prohlížeče v nezměněné podobě.
- **Dynamický webový server** se skládá ze statického webového serveru a dalšího softwaru, nejčastěji aplikačního serveru a databáze. Říká se mu „dynamický“, protože aplikační server aktualizuje hostované soubory před odesláním obsahu klientovi. [23]

Například pro vytvoření finálních webových stránek, které lze vidět v prohlížeči, může aplikační server naplnit šablonu HTML obsahem z databáze. Stránky jako např. Wikipedia obsahují tisíce webových stránek. Tyto typy webů se obvykle skládají pouze z několika šablon HTML a obrovské databáze, nikoli z tisíců statických dokumentů HTML. Tento princip usnadňuje údržbu a poskytování obsahu. [23]

1.2.1 Reverzní proxy

Webový server se často používá jako tzv. reverzní proxy. Jedná se o server, který se nachází před několika dalšími webovými servery a přeposílá požadavky klienta na ně. Reverzní proxy servery jsou obvykle implementovány za účelem zvýšení bezpečnosti, výkonu a spolehlivosti. [24]

Často se lze taky potkat s pojmem *forward proxy*. Zjednodušeně by se dalo říci, že forward proxy server stojí před klientem a zajišťuje, že žádný koncový server nikdy nekomunikuje přímo

s tímto klientem. Naopak reverzní proxy server stojí před původním serverem a zajišťuje, že žádný klient nikdy nekomunikuje přímo s ním. [24]

Některé výhody použití reverzní proxy:

Vyrovnávání zátěže Populární webové stránky, které denně navštíví miliony uživatelů, nemusí být schopny zvládnout veškerý příchozí provoz na jediném serveru. Místo toho může být web rozdělen mezi skupinu různých serverů, které všechny zpracovávají požadavky na stejný web. V takovém případě může reverzní proxy server rozloží zátěž a v případě, že některý server zcela selže, ostatní servery se mohou o provoz postarat. [24]

Ochrana před útoky Díky reverznímu proxy serveru nemusí web nebo služba nikdy odhalit IP adresu svého původního serveru. To útočníkům výrazně ztěžuje možnost využít proti nim cílený útok. Místo toho se útočníci budou moci zaměřit pouze na reverzní proxy server. [24]

Ukládání do mezipaměti (caching) Reverzní proxy server může také ukládat obsah do mezipaměti, což vede k vyššímu výkonu. Odpovědi lze cashovat na reverzním proxy serveru, a tím pádem samotná aplikace nemusí být dotazována. [24] S tím dále souvisí i další bod.

CDN Úloha reverzních proxy serverů se stává ještě významnější, když se vezme v úvahu Content Delivery Network (CDN). V nastavení CDN jsou reverzní proxy servery prvky, které spravují příchozí a odchozí provoz. Tento proces zvyšuje výkon a spolehlivost a maximalizuje efektivitu sítě CDN. Reverzní proxy servery zde kromě jiného usnadňují rychlé doručení obsahu. Toho dosahují tím, že ukládají verze webových stránek do mezipaměti serverů na geograficky rozptýlených místech. To znamená, že když klient zadá požadavek, očekávaný obsah je již poblíž. [25]

Reverzní proxy server může sloužit jako komunikační článek, jenž přesměrovává zprávy od klienta ze svého HTTP portu na nějaký další port dané aplikace, která zprávu zpracuje a odpověď opět pošle zpět serveru, který ji následně přesměruje na klienta.

1.2.2 Virtual hosting

Termín *virtual host* se vztahuje k provozování více než 1 webu (např. *company1.example.com* a *company2.example.com*) na jednom počítači resp. jednom webovém serveru. [26]

Virtual hosting může být **IP-based**, což znamená, že každý web má jinou IP adresu, nebo **name-based**, což znamená, že na jedné IP adrese je hostováno více doménových jmen. Skutečnost, že běží na stejném fyzickém serveru, není pro koncového uživatele přímo patrná. [26]

U name-based virtual hostingu se server spoléhá na to, že klient nahlásí doménovou adresu jako součást HTTP hlavičky. Pomocí této techniky může stejnou IP adresu sdílet mnoho různých hostitelů. [27]

Kapitola 2
Analýza

V této kapitole byl proveden podrobný sběr požadavků pro požadovaný nástroj. Byl zanalyzován aktuální stav a provedena analýza již existujících nástrojů, které by mohly plně či částečně realizovat všechny potřebné požadavky. Konec kapitoly je věnován shrnutí těchto získaných informací. Jsou zde také probrány možné postupy a technologie. Výsledky těchto poznatků jsou použity k návrhu vlastního nástroje v pozdější kapitole.

2.1 Katalog požadavků

Vývoj aplikace pro automatické zjišťování verzí softwaru v síti, který je předmětem této bakalářské práce, byl realizován ve spolupráci s firmou Quanti s. r. o. Tato spolupráce poskytla jedinečnou příležitost k aplikaci teoretických poznatků získaných během studia v praktickém a profesionálním prostředí.

Sběr a specifikace požadavků na vyvíjenou aplikaci probíhal v úzké spolupráci s administrátorským týmem firmy, který má na starosti správu serverů a služeb na nich běžících. Mezi tyto služby patří i *self-hosted* webové aplikace. Diskuze s týmem probíhaly formou neformálních konverzací, během kterých byly identifikovány klíčové potřeby a očekávání spojené s monitorováním a správou softwarových verzí v síti.

Typickým uživatelem aplikace je tedy administrátor sítě, který spravuje a monitoruje infrastrukturu s nasazenými self-hosted webovými aplikacemi.

Z těchto interakcí byly čerpány jak veškeré funkční, tak i nefunkční požadavky, což zajišťovalo, že navrhované řešení bude co nejlépe odpovídat potřebám a očekáváním administrátorského týmu.

Interakce umožnila hlubší porozumění praktickým aspektům správy IT infrastruktury a přinesla cenné náhledy, které byly zásadní pro definování cílů a funkcionalit vyvíjeného řešení. Tímto způsobem byl zajištěn vysoký stupeň relevance a užitečnosti aplikace pro potřeby firmy Quanti s. r. o. a jejího administrátorského týmu, který bude dále v textu zmiňován jako *zadavatel*.

Během sběru požadavků a pozdějšího návrhu bylo však přihlédnuto k tomu, aby řešení nebylo závislé pouze na konkrétním prostředí zadavatele, ale využitelné také v jiných prostředí se stejnou problematikou. Hledaný nástroj by měl splňovat všechny nadcházející funkční i nefunkční požadavky nehledě na konkrétních principech a technických záležitostech zadavatele jako jsou např. konkrétní serverové konfigurace.

Požadavky jsou rozděleny na kritickou, vysokou, střední a nízkou prioritu.

2.1.1 Funkční požadavky

2.1.1.1 F1 Monitorování verzí známých webových aplikací

Aplikace musí být schopna získávat informace o nejnovějších dostupných verzích vybraných webových aplikací. Velkou výhodou by také byla snadná možnost rozšíření, která by uživateli pomohla k zadefinování pravidel pro monitorování dalších webových aplikací, které nejsou součástí základního balíčku (viz 2.1.1.6).

Priorita:
Jedná se o **vysokou** prioritu. Aplikace by se dala používat maximálně za napůl použitelnou, pokud by tuto funkcionalitu nesplňovala, jelikož by nebyla schopná nalezený software srovnat s jeho oficiální nejnovější verzí.

Složitost
Implementace bude vyžadovat unikátní přístup pro každou podporovanou aplikaci. Některé služby nebudou poskytovat informaci o verzi v sofistikované podobě.

2.1.1.2 F2 Sken sítě

Aplikace musí umožňovat základní sken privátní sítě, který získá seznam adres, kde pravděpodobně běží nějaká webová aplikace. Nestačí zjistit pouze dané IP adresy, protože jedna IP adresa může hostovat více různých doménových jmen (viz *virtual hosting* 1.2.2). Výsledkem tohoto požadavku bude tedy seznam webových stránek, které hostuje webový server běžící na zařízení v daném síťovém rozsahu.

Priorita:
Aplikace by teoreticky mohla splňovat ostatní funkcionality jako hledání služeb a verzí čistě pro zadaná doménová jména a obejít se bez tohoto skenu. Nedošlo by ale k odhalení vynechaných adres a nástroj by ztratil prvek automatizace. Proto priorita tohoto požadavku zůstává **vysoká**.

Složitost:
Ke splnění požadavku bude využit některý z volně dostupných existujících nástrojů. Tento nástroj bude potřeba pouze zakomponovat do aplikace. Nemělo by se tedy jednat o příliš složitý požadavek.

2.1.1.3 F3 Určení konkrétní webové aplikace

Aplikace se musí pokusit o detekci názvu konkrétní webové aplikace, která se na dané adrese v síti vyskytuje, pokud je nativně podporována.

Priorita:
Tento požadavek je **kritický** pro fungování aplikace. Nástroj se bez něj neobejde.

Složitost:
Při navštívení webové aplikace je pro lidské oko ve většině případů celkem snadné detekovat o jakou konkrétní aplikaci se jedná. Náročnost se tedy bude odvíjet od toho, jak obtížné bude tento proces programově zautomatizovat. Odhadem se jedná o požadavek se střední složitostí.

2.1.1.4 F4 Určení verze webové aplikace

Aplikace musí přijít s řešením, které umožní získání verze dané webové aplikace, pokud zná její adresu v síti a konkrétní specifikaci, o jakou službu se jedná.

Priorita:
Je nutné, aby aplikace dokázala splnit tento požadavek pro většinu webových aplikací, které bude nativně podporovat. Priorita tohoto požadavku je **vysoká**.

Složitost:
Jedná se o poměrně složitý požadavek. Kromě toho, že ne vždy se informace nachází na stejném místě, tak některé služby mohou před uživateli verzi skrývat a bude potřeba projít nějakým autorizačním mechanismem, díky kterému přejdeme přes vrstvu běžného uživatele. Požadavek je tedy odhadován jako velmi složitý.

2.1.1.5 F5 Vystavení souhrnu informací formou API

Výsledkem **F1** a **F4** by mělo být srovnání rozdílu těchto dvou informací. Tento souhrn by měla aplikace poskytovat formou API vystaveného v dané privátní síti.

Priorita:
Bez vystaveného souhrnu postrádá aplikace smysl. Jedná se tedy o **kritickou** prioritu.

Složitost:
Lze konstatovat, že díky vývoji a dostupnosti moderních vývojových frameworků a nástrojů se stává tento požadavek ne příliš komplikovaným. Tyto frameworky poskytují mnoho vestavěných funkcí. Náročnost se tedy bude odvíjet pouze od osvojení si takového nástroje.

2.1.1.6 F6 Možnost doplnění vlastní aplikace

Uživatel (administrátor sítě) by měl mít možnost zadefinovat novou aplikaci k monitorování. Po zadání příslušných informací do seznamu ostatních podporovaných aplikací bude sken sítě v rámci **F3** a **F4** pátrat také po webové aplikaci s touto definicí. Velkou výhodou by také bylo bylo zadefinování podpory pro vyhledání nejnovějších verzí, tedy možnost rozšíření seznamu pro **F1**.

Priorita:
Aplikace bude bez tohoto požadavku použitelná pro daný předpřipravený seznam, avšak neumožní rozšířitelnost, která povede využitelnosti aplikace v jiných prostředích či v prostředí stávajícím, kde může v budoucnu docházet ke změnám. Požadavek má tedy **střední** prioritu.

Složitost:
Řešení bude vyžadovat kvalitní návrh, který umožní potřebné definice pronést do samotné aplikace. Zadání informací by mělo probíhat uživatelsky přívětivou formou, která nebude vyžadovat zásahy do kódu. Definice by měla vést k validnímu způsobu získání názvu a verze dané nové aplikace. Požadavek se jeví jako velmi složitý.

Seznam podporovaných webových aplikací

Zadavatel práce přišel se seznamem konkrétních webových aplikací, které požaduje, aby nástroj přímo podporoval. Předchozí funkční požadavky **F1**, **F3** a **F4** by tedy měly být splněny pro následující webové aplikace:

- Atlasian Jira
- Atlasian Confluence
- JFrog Artifactory
- Bareos

- GitLab
- Prometheus
- Grafana
- Zabbix
- Greylog
- SnipeIT
- TestRail
- TeamCity
- Keycloak
- Minio

Bez seznamu, které aplikace budou podporované, nebude nástroj moci být bez dalšího zásahu označen za funkční. Zadavatel dodal tento konkrétní seznam, který není krátký, ale zároveň zde existuje jakási tolerance. Nemělo by dojít k značnému narušení nefunkčních požadavků na úkor podpory všech služeb z tohoto seznamu.

2.1.2 Nefunkční požadavky

2.1.2.1 N1 Technologie

Aplikace bude nasazena na operační systém Debian. Tento operační systém používají taktéž zařízení, na nichž jsou nasazeny samotné zkoumané webové aplikace. Toto je zásadní informace, pokud by bylo třeba na serverech automatizovaně vykonávat některé úkony. Operační systém Debian může být v mnoha ohledech odlišný od jiných Linuxových distribucí nemluvě o operačních systémech používající jiné jádro jako macOS či Windows.

Webové aplikace používají k distribuování obsahu webové servery *Nginx* a *Apache*.

2.1.2.2 N2 Rozšiřitelnost

Aplikace by měla být navržena tak, aby umožňovala snadné přidávání dalších metod, které povedou k zajištění validních výsledků a spolehlivému chodu aplikace.

Návrh aplikace by měl být také uzpůsobený případnému přidání dalších druhů webových serverů k detekci.

2.1.2.3 N3 Jednoduchost rozhraní

Rozhraní ve formě API by mělo být intuitivní. Struktura informací v rámci daných odpovědí by měla obsahovat pouze a právě důležitá data bez zbytečných detailů, které by mohly snížit srozumitelnost výsledných informací.

2.1.2.4 N4 Dokumentace

Dokumentace by měla být úplná; pokrývat všechny aspekty API včetně požadavků, odpovědí a statusových kódů. Dokumentace by také měla být psána jasně a srozumitelně, s příklady použití, které pomáhají rychle pochopit, jak API funguje.

2.1.2.5 N5 Cena

Implementace zadání by neměla obsahovat žádné komerční prvky, za které by uživatel aplikace musel přímo zaplatit.

2.1.2.6 N6 Automatizace

Řešení by nemělo využívat příliš vysoký vstup od uživatele. V ideálním případě by sken sítě měl proběhnout zcela bez nutnosti zadání dalších vstupních informací např. o zařízeních či konfiguracích webových aplikací. Systém by měl fungovat téměř zcela samostatně. Toleruje se nějaká forma autentizace.

2.1.2.7 N7 Bezpečnost

Aplikace by neměla požadovat administrátorská oprávnění ke zkoumaným serverům.

2.2 Aktuální stav

V tuto chvíli není využíván žádný nástroj, který by dokázal souhrnně a exaktně informovat o adresách a verzích self-hosted webových aplikací nasazených na serverech v privátní síti Quanti s. r. o.

K základnímu přehledu o nasazených službách v firemní síti slouží nástroj CMDB. Jedná se o vlastní implementaci tzv. konfigurační databáze.

Configuration Management Database (CMDB) je databáze, která obsahuje všechny relevantní informace o hardwarových a softwarových komponentách používaných v IT službách organizace. CMDB uchovává informace, které zajišťují organizovaný pohled na konfigurační data a prostředky pro zkoumání těchto dat. [28]

Aktuální stav této databáze nesplňuje požadavky požadovaného nástroje (viz 2.1.1). Databáze obsahuje seznam doménových jmen a jejich IP adres všech služeb nasazených ve firemní síti. Postrádá informaci o tom, zda se na adrese vyskytuje webová aplikace (na adrese se může vyskytovat jiný typ služby jako např. databázový cluster fungující nad jinou technologií než HTTP protokol, který je běžně používán na procházení webových aplikací). Kromě toho by se také mohlo stát, že CMDB není aktuální a nějaká služba ji uniká.

Ve firmě tedy neexistuje žádný sofistikovaný a automatický způsob, který by byl schopný odhalit webové aplikace a jejich názvy. Seznam webových aplikací je shromážděn v podobě textového seznamu v interním dokumentačním systému, který musí být manuálně udržován. Počet služeb není nikterak vysoký, a proto nebylo prozatím nutné přijít s nějakým sofistikovaným řešením. Administrátoři si také sami pamatují seznam aplikací a služeb, což se dá pochopitelně považovat za velmi špatný způsob udržování informací.

Pro zjištění rozdílu mezi aktuálně nasazenou a oficiálně nejnovější verzí webové aplikace musí uživatel manuálně verzi zkontrolovat jak na dané adrese v síti, tak v oficiální dokumentaci služby. Toto řešení vyžaduje časovou investici ze strany uživatele, navíc může na nějakou z aplikací zapomenout a v neposlední řadě toto řešení také negarantuje získání informace včas, což může být problém, protože daná aktualizace může mít kritický vliv na její bezpečnostní chování.

2.3 Dostupné nástroje

Tato část je zaměřena na služby, které jsou analyzovány z hlediska funkcionality a případného splnění funkčních a nefunkčních požadavků.

Nástroj *Nessus* byl doporučen k analýze po komunikaci s členy z administrátorského týmu Quanti s. r. o. Údajně se mělo jednat o nástroj, který se nejvíc podobá zadání aplikace.

Ostatní nástroje byly vybrány po průchodu internetem, při němž byla vyhledávána klíčová slova: „web application scanner“, „discover versions of web applications“, „web server discover“,

„get version of self hosted web application“ apod. V první vlně analýzy byly z dlouhého seznamu nástrojů vyloučeny ty, které buďto neobsahovaly směrodatnou dokumentaci, nebo se požadavkům blížily pouze zdánlivě. Ačkoliv jedním z požadavků je také cena, níže jsou zmíněny i některé komerční produkty.

2.3.1 Nmap

Nmap je flexibilní open source nástroj pro hledání počítačů a služeb v počítačové síti, čili k vytvoření jakési „mapy“ sítě. Aby Nmap mohl vykonávat tyto úkoly, odesílá na cílové počítače speciálně upravené pakety a poté analyzuje odpovědi. Má možnost skenovat otevřené porty a identifikovat běžící služby, včetně webových serverů. [29]. Klíčové aspekty použití Nmapu pro účel této práce zahrnují:

1. **Skenování Portů:** Nmap umožňuje identifikovat otevřené porty na cílových zařízeních. [29] Toto je první krok, který může pomoct s **F2** (2.1.1.2) protože webové servery obvykle naslouchají na specifických portech (80 - HTTP, 443 - HTTPS).

2. **Použití Skriptů NSE (Nmap Scripting Engine)**: Nmap má vestavěný skriptovací engine (NSE), který umožňuje spouštění různých skriptů pro rozšířené skenování a sběr informací. Pro zjištění verzí aplikací existují specifické NSE skripty, které se zaměřují na detekci služeb a verzí softwaru. [30]

   - **http-enum:** Za určitých podmínek dokáže vypsat souborovou strukturu webové aplikace (pro komplexní webové aplikace běžící za reverzním proxy serverem je toto většinou nepoužitelné). [31]
   - **banner:** Jednoduchý banner grabber, který se připojí k otevřenému portu TCP a vypíše vše, co naslouchající služba odešle do pěti sekund. [32] Po otestování vyšlo najevo, že pro porty, kde běží webový server, nástroj našel pouze název webového serveru (např. Nginx). Služba dokázala získat informaci o některých webových aplikacích, které na zařízení běžely na jiných portech (webový server na některé z nich odkazoval jako reverzní proxy), ale nešlo o nic moc víc než informace, které by mohly být zjištěny např. pomocí příkazu *curl*[1].

3. **Verze Detekce Služby (Service Version Detection)**: Pomocí přepínače *-sV* v příkazové řádce Nmap, lze provést detekci verze služby. Tento přepínač instruuje Nmap, aby zkoumal identifikované otevřené porty a snažil se zjistit informace o službách, které na nich běží, včetně jejich verzí. [33] Po další analýze bylo zjištěno, že v určitých momentech je Nmap schopný zjistit druh a verzi webového serveru (Nginx, Apache atd.), nikoliv verzi samotné webové aplikace.

4. **Vlastní NSE Skripty:** Pokud standardní skripty NSE nezjistí potřebné informace, je možnost napsat nebo upravit vlastní skripty v Lua, programovacím jazyce, který Nmap používá pro NSE. [30]

Souhrn: Nmap má vícero způsobů, jak prozkoumat aplikace běžící na specifických portech. Pro potřebu splnění **F3** (2.1.1.3) a **F4** (2.1.1.4) je to pravděpodobně nedostatečné, jelikož Nmap pravděpodobně nepracuje na tak vysoké komunikační vrstvě. Teoreticky by možná ještě stálo za to prozkoumat variantu vlastních NSE skriptů. Nmap není na potřebné účely přizpůsoben a vynaložené úsilí by bylo větší než při vlastní implementací řešení. Nmap ale částečně splňuje **F2** (2.1.1.2), jelikož dokáže poměrně snadno a přehledně projít daný rozsah sítě a zjistit otevřené porty. Mohl by tedy kontrolovat HTTP(S) porty, a tím pádem pravděpodobně detekovat běžící webové servery. Nmap bohužel nedokáže odhalit všechna hostovaná doménová jména daného webového serveru.

[1] cURL je softwarový projekt poskytující jednoduchý command-line nástroj (curl) pro přenos dat pomocí různých síťových protokolů.

2.3.2 Nikto

Nikto je open source skener webových serverů, který provádí komplexní testy na více než 6700 potenciálně nebezpečných souborů/programů, kontroluje zastaralé verze a problémy s konkrétními verzemi. Kontroluje také položky konfigurace serveru, jako je přítomnost více indexových souborů a pokouší se identifikovat nainstalované webové servery a software. Jednotlivé kontrolované položky a moduly jsou často aktualizovány. [34]

Při testování nástroje v síti zadavatele vyšlo najevo, že je schopný odhalit webovou aplikaci Jira (už ale ne její verzi) a to díky *favicon* [2]. Nikto si uchovává databázi těchto ikon a některé z nich je tedy schopný rozpoznat. Databáze je ale poměrně malá, a tudíž nedostačující.

2.3.3 WhatWeb

WhatWeb identifikuje webové stránky. Rozpoznává webové technologie včetně systémů pro správu obsahu (CMS), platforem pro blogování, statistických / analytických balíčků, knihoven JavaScript, webových serverů a vestavěných zařízení. WhatWeb má více než 1800 modulů, z nichž každý rozpoznává něco jiného. Jedná se o open source projekt. [36]

Pro dané adresy místy dokáže odhalit danou webovou aplikaci. V odhalování verzí je ale velmi omezený. Navzdory tomu, že projekt tuto funkcionalitu zmiňuje, nepodařilo se v dokumentaci najít postup k získání verzí aplikací a při běžném používání (i se silnými přepínači) většinou verzi nenašel. Zde je příklad použití:

```
root@e46997465591:/app# ./whatweb -a 3 https://jira-preprod.XXX.cz

https://jira-preprod.XXX.cz/ [200 OK] Atlassian-JIRA,
Cookies[JSESSIONID,atlassian.xsrf.token], Country[RESERVED][ZZ],
HTML5, HTTPServer[nginx], HttpOnly[JSESSIONID], IP[192.168.68.140],
Java, OpenSearch[/osd.jsp], Script[context,module,
resource,text/javascript], Strict-Transport-Security[max-age=31536000,
max-age=15768000], Title[Loading...], X-Frame-Options[SAMEORIGIN], nginx
```

2.3.4 Wappalyzer

Wappalyzer je open-source nástroj pro detekci technologií používaných na webových stránkách, který umožňuje uživatelům zjistit, jaké webové technologie (například webové servery, CMS systémy, frameworky, knihovny JavaScriptu, analytické nástroje atd.) a jejich verze jsou použity na konkrétních webových stránkách. Tato aplikace může být použita přímo v prohlížeči jako rozšíření nebo jako samostatná aplikace. Wappalyzer pomáhá vývojářům, marketérům, a bezpečnostním analytikům lépe pochopit použité technologie a může být využit k analýze trhu, konkurence, nebo k identifikaci potenciálních bezpečnostních zranitelností. Díky své schopnosti rozpoznávat stovky technologií, je Wappalyzer cenným nástrojem pro každého, kdo se zabývá webovou analýzou nebo bezpečnostním auditováním. [37]

Aplikace je poskytována uživatelům ve formě rozšíření do webového prohlížeče Google Chrome. Jelikož se jedná o open source projekt, je možné dohledat oficiální Git repozitář. Projekt umožňuje uživatelům použít nástroj pomocí API, tudíž by teoreticky bylo možné si jej osvojit v rámci této práce.

Nástroj má předdefinovanou sadu pravidel pro každý software, který podporuje. [38] Pravidla určují způsob, jakým Wappalyzer získává informace o dané službě na konkrétním URL. Po prostudování kódu bylo zjištěno, že informaci o verzi softwaru získává z HTML elementů na

[2] Favicon, známá také jako ikona zástupce, ikona webové stránky, ikona karty, ikona URL nebo ikona záložky, je soubor obsahující jednu nebo více malých ikon spojených s určitou webovou stránkou. [35]

dané stránce. Často se tato informace vyskytuje v metadatech HTML dokumentu. Databáze čítá velké množství podporovaných služeb a mezi nimi se nachází také webové aplikace.

Po průzkumu bylo zjištěno, že Wappalyzer obsahuje podporu pro většinu požadovaných webových aplikací ze seznamu zadavatele (viz 2.1.1.6). Po otestování vyšlo ale najevo, že tato databáze je pravděpodobně neaktuální, jelikož pro aplikaci Jira, Artifactory, TeamCity a Zabbix běžících ve zkoumané firemní síti nebyl nástroj schopen zjistit verzi, ačkoliv je podporuje a verze se na stránce vyskytuje. Nicméně u všech zmíněných kromě Bareos a Keycloak se mu alespoň podařilo detekovat název služby. Službě se povedlo úspěšně detekovat verzi pouze u webové aplikace Confluence.

U některých webových aplikací jako GitLab došlo k pokusu použití nástroje na hlavní stránce, kde uživatel zadává své přístupové údaje. Po prozkoumání HTML dokumentu došlo k zjištění, že žádnou informaci o verzi neobsahuje, a tudíž je nemožné ji jen čistě z tohoto zdroje získat. V některých případech je tedy nutné nejprve proniknout za tuto bezpečností vrstvu. Navíc je možné, že se informace v HTML obsahu nebude vyskytovat ani po přihlášení uživatele. Tato skutečnost je dále analyzována v sekci 2.4.2 - *Detekce názvu a verze.*

2.3.5 Nessus

Nessus od firmy Tenable, Inc. je proprietární bezpečnostní nástroj, který slouží k hledání a vyhodnocování zranitelností.

Projekt vznikl v roce 1998 jako free remote bezpečnostní skener. [39] 5. října 2005, s vydáním Nessus 3, projekt přešel z GNU Public License na proprietární licenci. [40]

Kromě celkového skenu zařízeních připojených do veřejného internetu a zabezpečení cloudové infrastruktury také bezpečně skenuje webové aplikace a identifikuje zranitelnosti v kódu vlastních aplikací a komponentách třetích stran. Výsledky pak dokáže zobrazovat v rámci reportů dostupných z jeho webové aplikace. [41]

Nástroj je dostupný ve třech variantách. Nessus Expert a Nessus Professional jsou placené verze tohoto softwaru nabízející většinu funkcionalit. [41] Existuje však také bezplatná verze Nessus Essentials. Ta umožňuje skenovat prostředí (až 16 IP adres) se stejnou rychlostí, hodnocením do hloubky a pohodlným skenováním bez použití agenta, jaké využívají placené verze. [42]

Nástroj mohl být v bezplatné verzi Essentials vyzkoušen přímo v interní síti, kde jsou nasazeny webové aplikace ze seznamu aplikací požadovaných k podpoře (viz 2.1.1.6). Následující poznatky o funkčnosti aplikace nebudou pocházet z oficiálních zdrojů, nýbrž ze zkušeností během tohoto testování.

Webová aplikace nástroje nabízí opravdu komplexní možnosti, které uživatel může využít k otestování bezpečnostních zranitelností celých částí sítě či konkrétní adresy. Nessus nabízí šablony, které může uživatel využít k vytvoření specifického skenu. Pro účely této práce byla nejzajímavější „Web Application Tests“ šablona. Pomocí ní proběhl test na webové aplikaci Jira a Artifactory nasazené na známé adrese v síti.

Po průchodu výsledků vyplynulo, že Nessus vezme zadanou IP adresu či URL a pokouší se na ni přistoupit kromě klasického HTTP a HTTPS portu skrz jiné další porty. Uživatel může vybrat hloubku skenu a tím se zvýší počet otestovaných portů. Nessus pak zkouší přistupovat a zanořovat se na nejrůznější podsložky adresy a vytěžit z nich maximum informací.

Výsledky, které byly seřazeny na základě závažnosti testů, poukázaly na spoustu možných zranitelností webového serveru či samotné aplikace. Jednou z nízko závažných zranitelností bylo také zjištění názvu a verze aplikace v případě testu pro Jira. U Artifactory byl sken schopný najít pouze název této služby. Nicméně nástroj také nabízí možnost použití autentizace. Před startem skenu tedy uživatel vyplní příslušné uživatelské jméno a heslo, které nástroji umožní proniknout za hranici uvítací obrazovky. Tento princip vedl v případě Artifactory ke zjištění aktuální verze. [3]

[3] Ne zcela aktuální - sémantická verze se lišila o 1 číslo v *patch* části.

Nessus kromě jiného obsahuje databázi odhalených konkrétních zranitelností, které vyžadují aktualizaci aplikace, pokud chce poskytovatel předejít možným útokům. [43] Tato databáze se pravděpodobně pravidelně aktualizuje, avšak nedařilo se dohledat, jak často a kým. Nicméně pokud je Nessus schopný aktuální verzi služby najít a srovnat s nějakým záznamem ze zmíněné databáze, tak výsledný sken může mimo jiné obsahovat informaci i o tomto problému.[4] Tento proces byl ověřen v rámci testování pro jednu ze zmíněných webových aplikací.

V rámci testování nástroje byl také proveden hluboký sken sítě, který Nessus Essenials nabízí. Vstupem skenu byl rozsah sítě. Byla využita přímo interní síť zadavatele. Výsledný sken fungoval poměrně omezeně, jelikož nedokázal odhalit všechna zařízení. Pro zařízení na nichž se nacházela běžící webová aplikace za webovým serverem nedokázal nástroj ve většině případů spolehlivě odhalit název ani verzi těchto aplikací.

Souhrn

Co se týče **F3** (2.1.1.3) a **F4** (2.1.1.4), nebyla nalezena žádná dostupná a bezplatná služba, která by byla schopná spolehlivě splňovat tyto požadavky. Většina volných nebo komerčních nástrojů má spíše za cíl provádět bezpečnostní skeny na typické bezpečností zranitelnosti. Nástroj, který by alespoň do určité míry dokázal poskytovat informace o názvech a verzích známých webových aplikací, buďto neexistuje, nebo nedošlo k jeho nalezení. Některé nástroje umožnily získat název webové aplikace (např. Nikto nebo WhatWeb). Pouze ale ve výjimečných případech dokáží najít také její verzi. O něco úspěšnější byl Nessus, který dokázal detekovat verze mnohem častěji. Bohužel je ale jeho bezplatná verze omezená, co se do počtu adres týče. Navíc je jeho výstup příliš komplexní na to, aby se dal použít jako část nástroje. Použít Nessus jako takový pro splnění celého zadání také nelze. Kromě zmíněného problému s bezplatnou verzí nedokáže ve většině případů spolehlivě informovat o zastaralosti softwaru, jelikož tuto informaci vydává jen někdy, pravděpodobně jen pokud se jedná o bezpečnostní upozornění z jeho databáze zranitelností.

Nástroj Nmap se jeví jako velmi vhodný k částečné realizaci **F2** (2.1.1.2). Aplikace je poměrně jednoduchá na použití. Její práce ale končí v bodě, kdy odhalí IP adresy zařízení s otevřenými webovými porty.

Dále byly objeveny některé další nástroje jako (*Pulseway Lansweeper*, *ManageEngine* či *AssetExplorer*), které podle prvních dojmů ukazovaly potenciál ke splnění požadavků. Jedná se o robustní služby, které jsou schopny monitorovat firemní software. To, jestli jsou schopné mimo jiné monitorovat i verzi webových aplikací na interních serverech se nepodařilo dohledat. Služby jsou navíc až moc komplexní na to, aby mohly být použity k poměrně jednoduchým zkoumaným požadavkům. A to, že jsou placené, byl také další důvod k zavržení.

Při hledání byly dále nalezeny služby, které jsou schopny pravidelně monitorovat vybraný element na webové stránce a v případě změny informovat uživatele. Pokud by webová aplikace neimplementovala žádný jiný pohodlnější způsob (API, RSS feed) k získání její aktuální verze, teoreticky by se dalo tento nástroj využít. Ale vzhledem k tomu, že je to jen velmi částečná pomoc, bude mnohem jistější a jednodušší **F1** (2.1.1.1) naimplementovat kompletně vlastním způsobem např. pomocí *web scrapingu*[5].

2.4 Možné postupy

Tato sekce je věnována analýze principů a způsobů, jejichž zjištění a pochopení je důležité k provedení ideálního návrhu systému, který povede ke splnění zadání.

[4] Závažnost výsledku je definována dle závažnosti záznamu v databázi.

[5] Web scraping, web harvesting, nebo také web data extraction je způsob získávání dat z webových stránek. Data jsou často extrahována z HTML elementů samotného obsahu stránky. [44]

2.4.1 Hledání webových aplikací

Jeden z prvních kroků ke splnění zadání je najít všechny webové aplikace v dané síti. Lze uvažovat, že ke skenu síťového rozsahu a zjištění otevřených HTTP a HTTPS portů bude využit volně dostupný nástroj Nmap, jelikož implementace vlastního nástroje by byla zbytečná a příliš rozsáhlá.

Sken nástroje Nmap dokákže poskytnout seznam IP adres, na nichž se nachází otevřené porty dle výběru. [45] Při zadání HTTP(S) portu je tedy obdržen seznam IP adres ve zkoumané síti, kde pravděpodobně běží nějaký webový server. Tato informace ale sama o sobě nemusí být dostačující hned z několika důvodů:

1. **Na HTTP(S) portu se nemusí nutně vyskytovat webová aplikace.** Ačkoliv jsou TCP/IP port nižší než 1024 speciální v tom, že normální uživatelé na nich nemohou spouštět servery [46], může se stát, že se administrátor zařízení rozhodne použít tyto porty pro jiné účely než běh webového serveru.

2. **Tato informace neříká, že na adrese běží nějaká webová aplikace.** Běžící webový server může být využíván k přesměrování provozu na jiné zařízení s webovou aplikací, která se nenachází ve zkoumaném síťovém rozsahu.

3. **Nejsou známa konkrétní doménová jména webových aplikací, které webový server poskytuje.** Stále nejsou známa všechna doménová jména, která webový server odbavuje. Jsou známy pouze IP adresy, ale na nich může díky technice virtual hosting docházet k vyřizování požadavků pro několik webových aplikací.

4. **To, že na zařízení neběží webový server neznamená, že se na něm nevyskytuje webová aplikace.** Pokud se považuje vyhledávání otevřených webových portů v síti jako primární krok, výsledkům mohou uniknout webové aplikace, které nejsou tímto způsobem poskytovány. Webová aplikace ve formě statických souborů se může na zařízení vyskytovat a být na ní odkazováno v rámci jiného webového serveru mimo rozsah sítě. Stejně tak složitější webová aplikace může naslouchat na nějakém svém vlastním specifickém portu a externí webový server na něj může odkazovat.

Lze tedy uvažovat, že v počátečním skenu sítě je získán seznam IP adres, na nichž je otevřený HTTP(S) port. Ačkoliv by bylo teoreticky možné používat tyto porty pro jiné služby, jak je zmíněno v prvním bodě, tak je velmi pravděpodobné, že se skutečnou naslouchající službou je webový server. I kdyby nebyl, tak to nebrání k další fázi průzkumu, která by případně v pozdější fázi selhala. Tato okolnost by při nejhorším zpomalila celkový sken, avšak nevynechala by adresy se skutečně legitimním webovým serverem.

Pokud by bylo možné nalezené zařízení s webovým serverem vzdáleně navštívit s dostatečnými oprávněními [6], tak lze prozkoumat konfigurační soubory jeho případných webových serverů. Jak Apache tak Nginx si musí nutně uchovávat informace o tom, jaké webové stránky hostují. Tím by byl získán seznam všech virtuálních hostů a druhý zmiňovaný bod je tedy pokrytý. Nemusí se ale nutně jednat o webové aplikace běžící v daném síťovém rozsahu, viz druhý bod. Pokud ale webový server přeřazuje požadavky pro dané doménové jméno na jinou adresu v síti, tak se tato informace bude opět muset vyskytovat někde v konfiguracích serveru.

Ke zjištění, zda se daná adresa vyskytuje ve zkoumané síti nebo ne, poslouží **reverzní DNS vyhledávání**. Jedná se o techniku, která je schopna pro dané doménové jméno zjistit jeho IP adresu. [49] Pokud bude výsledná adresa patřit do rozsahu zkoumané sítě, tak je jasné, že příslušné doménové jméno náleží nějaké webové aplikaci běžící ve zkoumané síti. Záleží ale, z jaké sítě se vyhledání provádí. Díky metodě NAT se IP adresy z lokální sítě mohou mapovat na jiné

[6] Jedním z oblíbených nástrojů pro vzdálený přístup k Linuxovým zařízením je SSH (Secure Shell). Jedná se o šifrovaný síťový protokol a také program umožňující bezpečnou komunikaci mezi dvěma počítači pomocí command line příkazů. [47, 48]

IP adresy, když přes routovací zařízení komunikují s vnější sítí. [50] Kromě toho, že celkový sken musí tedy být prováděn ze stejné privátní sítě, která je zkoumána, tak např. v případě techniky vyvažování zátěže může reverzní DNS vyhledání z různých zařízení vrátit jinou IP adresu.

Bod č. 4 je poměrně komplexní problém, na který by dosavadní postup nestačil. Teoreticky by bylo možné v rámci skenu sítě hledat i jiné otevřené porty, což by mohlo dopomoct k nalezení adres, u kterých by se mohlo zkoumat, zda jsou na nich nasazeny webové aplikace. Po dohodě se zadavatelem bylo toto označeno za krajní případ, který není požadován k pokrytí.

2.4.2 Detekce názvu a verze

Tato část je zaměřena na způsoby, které mohou vést k odhalení názvů a verzí webových aplikací běžících na operačním systému Debian. Předpokládá se, že podobně by se dalo postupovat i na jiných Linuxových operačních systémech, což ale není momentálně požadavkem zadavatele. Vstupem tohoto problému bude tedy doménové jméno a výstupem případný název a verze webové aplikace nacházejí se na této adrese.

Způsoby jsou analyzovány z hlediska 3 kritérií:

Věrohodnost: Míra toho, jak moc se dá spoléhat na výslednou informaci.

Prerekvizity: Nutné podmínky, které je potřeba zaručit, aby způsob vracel validní výstup.

Proveditelnost: Pokud jsou prerekvizity vysoké, je možné, že program nebude moci způsob využít, jelikož by potřeboval velké množství vstupů od uživatele.

2.4.2.1 Obsah stránky

Touto technikou se myslí extrahování vybraného HTML elementu při navštívení URL aplikace a načtení celého obsahu stránky. Získaný obsah (nebo alespoň jeho část, kde se verze nachází) je ve formě HTML. Hodnota elementu nebo její část přímo obsahuje danou informaci o aplikaci. Pokud je známa daná adresa s otevřeným přístupem, je velmi jednoduché požadovaný obsah získat.

V tabulce 2.1 lze vidět, že některé webové aplikace tuto informaci na své hlavní vstupní stránce bez autentizace neukazují. Dalo by se říci, že takto veřejná informace dává příležitost k bezpečnostnímu ohrožení. Útočník může totiž využít této informace k tomu, aby verzi porovnal s aktuální verzí služby a v rozdílu verzí našel možné bezpečnostní zranitelnosti, kterých by mohl zneužít ve svůj prospěch. Pokud verze není veřejná, útočník nemusí vědět o jakou verzi jde, a tudíž v tomto ohledu nemá tak velkou motivaci k útoku.

Věrohodnost
Informace uvedená přímo na stránce se dá považovat za zcela věrohodnou. Výjimkou by mohla být pouze chyba na straně aplikace či podvržení jiné verze útočníkem, což se jeví jako velmi nepravděpodobné.

Prerekvizity
Přístup na URL hlavní stránky aplikace

Proveditelnost
Tento způsob je velmi proveditelný, jelikož je zapotřebí pouze přístup na webovou adresu aplikace.

2.4.2.2 Obsah stránky (s autentizací)

Věrohodnost
Platí to stejné jako v předchozím případě.

Prerekvizity
Přístup na URL hlavní stránky aplikace a přístupové údaje pro autentizaci, která povede k proniknutí za hlavní veřejnou stranu aplikace.

Proveditelnost
Jedná se o něco složitější způsob, který bude vyžadovat získání a zadání údajů k autentizaci.

Tabulka konkrétních webových aplikací z požadovaného seznamu zadavatele (viz 2.1.1.6). Tabulka obsahuje informaci, zda se její aktuální verze vyskytuje na hlavní stránce, příp. na úvodní stránce po autorizaci.

Název	Hlavní strana	Hlavní strana po autentizaci
Atlasian Jira	✓	✓
Atlasian Confluence	✓	✓
JFrog Artifactory	✓	✓
Bareos	✓	✓
GitLab	✗	✓
Prometheus	✓[7]	✓
Grafana	✓	✓
Zabbix	✗	✓
Graylog	✗	✓
SnipeIT	✗	✓
TestRail	✓	✓[8]
TeamCity	✓	✓
Keycloak	✗	✓
Minio	✗[9]	✗

Tabulka 2.1 Výskyt aktuální verze na úvodní straně

2.4.2.3 Balíčky

Operační systém Debian a jeho deriváty nabízejí mechanismus balíčkování (anglicky „packages“). Balíčky Debianu jsou standardní unixové archivy, které obsahují dva archivy tar. Jeden archiv obsahuje řídicí informace a druhý instalační data. [51] Nástroje *dpkg* a *APT* umožňují uživateli jednoduchou instalaci těchto balíčků a tedy možnost, jak nainstalovat software na tento operační systém. [52] Mezi takový software můžou patřit i webové aplikace.

Věrohodnost
Na zařízení se může vyskytovat nainstalovaných několik verzí dané aplikace. Pomocí získání informací o balíčcích nelze zjistit, jaká verze je aktuálně nasazena. Dokonce nemusí být nasazena ani jedna z nich - nainstalovaný balíček neimplikuje běžící aplikaci. Jedná se tedy o velmi nevěrohodný způsob.

Prerekvizity
Přístup na server a povolení k výpisu seznamu nainstalovaných balíčků.

Proveditelnost
Pokud se předpokládá validní přístup na server, dá se tato metoda požadovat za velmi proveditelnou.

[7] Nenachází se přímo na hlavní straně.
[8] Nenachází se přímo na hlavní straně po přihlášení.
[9] Nepodařilo se dohledat žádný důkaz o tom, že by Minio podporovalo jakýkoliv druh verzování.

2.4.2.4 Konkrétní umístění aplikace na serveru

Pro každou self-hosted webovou aplikaci zpravidla musí existovat umístění, kde se nachází její konfigurační soubory a především spustitelná binární aplikace. Prohledání tohoto umístění, přečtení určitých souborů a případně spuštění binárních souborů s různými přepínači by mohlo vést k detekci verze.

Věrohodnost
Umístění aplikace se může měnit a tudíž způsobovat problémy s detekcí. Teoreticky by šlo se ke konkrétnímu umístění dopátrat, pokud by se začala zkoumat konfigurace webových serverů. Tento způsob je ale považován za velmi nespolehlivý, komplexní a náchylný na chyby.

Prerekvizity
Přístup na server a informace o konkrétním umístění aplikace.

Proveditelnost
Umístění složky s aplikací je zcela na rozhodnutí administrátora serveru. Poskytování této informace pro každou aplikaci by nevedlo k příliš dobré uživatelské přívětivosti. Dále je třeba vzít v potaz to, že toto umístění se může často měnit a tento vstupní požadavek by uživatele nutil mít stále aktuální pojem o těchto informacích. V neposlední řadě je potřeba také zmínit, že způsob zjištění samotné verze, i v případě splnění prerekvizit, bude nejspíš mezi všemi aplikacemi velmi odlišný. Jedná se tedy pravděpodobně o způsob s nejmenší proveditelností.

2.4.3 Monitorování nejnovějších verzí

Splnění **F1** bude vyžadovat monitorování aktuálně vydaných a podporovaných verzí vybraných aplikací ze seznamu zadavatele. Bylo zkoumáno, zda kromě získávání dat z oficiálních stránek či oficiálních Gitových repozitářů aplikací existuje nějaký další způsob. Například, zda neexistuje nástroj, který by poskytoval centralizované informace o více aplikacích.

2.4.3.1 Endoflife.date

Endoflife.date je webová stránka, která dokumentuje data o konci životnosti (EOL - End-Of-Life) a podpoře až pro 295 různých produktů. Endoflife.date shromažďuje data z různých zdrojů a prezentuje je srozumitelným a stručným způsobem. Data také zpřístupňuje pomocí snadno přístupného rozhraní API. [53] Pro získání požadovaných dat je třeba do URL požadavku vyplnit název softwaru (viz dokumentace).

Název	Podpora
Atlasian Jira	✓
Atlasian Confluence	✓
JFrog Artifactory	✓
Bareos	✗
GitLab	✓
Prometheus	✓
Grafana	✓
Zabbix	✓
Graylog	✓
SnipeIT	✗
TestRail	✗
TeamCity	✗
Keycloak	✓
Minio	✗

Tabulka 2.2 Podporované aplikace endoflife.date k 11. 3. 2024

2.4.3.2 GitHub RestAPI

GitHub volně poskytuje RestAPI, které lze použít k získání informací o dostupných verzí („releases“) pro dané repozitáře. Pro získání požadovaných hodnot je třeba vyplnit autora a název repozitáře. [54] Oproti *Endoflife.date* ale nemusí nutně obsahovat informace o podpoře verzí.

Souhrn

Webová stránka Endoflife.date poskytuje mnoho potřebných informacích, které se dají poměrně prakticky získat pomocí API.

U ostatních aplikací byla provedena následující analýza:

Bareos: Informace o releasech lze dohledat v oficiálním GitHub repozitáři. Jednotlivé verze neobsahují časovou informaci o konci podpory. [55]

TestRail: Na oficálních stránkách TestRailu se vyskytuje tabulka s verzemi a platností podpory. [56]

SnipeIT: Informace o releasech lze dohledat v oficiálním GitHub repozitáři. Jednotlivé verze neobsahují časovou informaci o konci podpory. [57]

TeamCity: Na oficiálních stránkách TeamCity lze dohledat seznam verzí. Zastaralé verze, které obsahují bezpečnostní zranitelnosti jsou zde označeny. [58]

Minio: Nepodařilo se dohledat žádný důkaz o tom, že by Minio podporovalo nějakou směrodatnou formu verzování. Jejich releasy v oficiálním GitHub repozitáři nesou název podle vydaného data. [59]

Kapitola 3
Návrh

Tato kapitola je zaměřena na návrh aplikace, která bude splňovat funkční a nefunkční požadavky zadavatele. K návrhu budou brány v potaz poznatky a zjištění z předchozí kapitoly, stejně tak jako zanalyzované existující technologie, které mohou ulehčit práci při vývoji. Tato kapitola obsahuje návrh rozhraní a předpis tříd dodržující správné principy objektového programování. Tyto návrhové prvky jsou vizualizovány ve formě diagramů. V neposlední řadě jsou zde zvoleny technologie použity pro následný vývoj a nasazení aplikace.

Z předchozí kapitoly o analýze vychází najevo, že neexistuje nástroj, který by spolehlivě splňoval všechny potřebné požadavky. Proto došlo k rozhodnutí o vytvoření návrhu a implementace nové aplikace, nebo alespoň prototypu, jehož cílem bude pokusit se co nejvíce přiblížit splnění požadavků.

Návrh byl proveden bez spolupráce s jiným týmem či osobou. Výjimkou bylo pouze navržení API rozhraní (viz 2.1.1.5), jenž bude v budoucnu využíváno administrátorským týmem Quanti s. r. o., pro monitorování stavu jejich serverů.

3.1 Volba technologií

Při volbě technologií bylo přihlédnuto k autorovým dosavadním zkušenostem s tvorbou aplikací. Již během návrhu bylo jasné, že aplikace bude vykonávat spoustu různých činností, od skenování sítě, procházení serverů po komunikaci s různými HTTP API a scrapování webového obsahu. Proto budou pro vývoj zvoleny následující technologie:

Python bude zvolen jako programovací jazyk. Jedná se o dynamicky typovaný jazyk využívající garbage collector.[1] Podporuje více programovacích paradigmat jako procedurální, objektové nebo funkcionální programování. Díky rozsáhlé standardní knihovně je často označován jako „batteries included“ jazyk. [61, 62] Jeho package manager PyPi hostuje tisíce modulů třetích stran. [63] Podle zkušeností autora je vhodný k řešení různorodých úkolů.

Flask bude zvolen jako technologie k tvorbě API. Flask je mikroframework napsaný v jazyce Python. Je klasifikován jako mikroframework, protože nevyžaduje zvláštní nástroje nebo knihovny. [64]

JSON bude zvolen jako formát pro výsledná data odesílané uživateli přes API. JSON je „lightweight“ formát pro výměnu dat. Lidé jej mohou snadno číst i zapisovat v něm. Stroje

[1] Garbage collection ulehčuje programátorovi práci se správou paměti. V případě ruční správy paměti totiž programátor určuje, které objekty mají být dealokovány a vráceny do paměťového systému a kdy se tak má stát. [60]

jej snadno analyzují a generují. Je založen na podmnožině standardu programovacího jazyka JavaScript ECMA-262 3rd Edition - prosinec 1999. [65]

JSON je textový formát, který je zcela nezávislý na programovacím jazyku, ale používá konvence, které jsou známé programátorům jazyků rodiny C, včetně jazyků C, C++, C#, Java, JavaScript, Perl, Python a mnoha dalších. Díky těmto vlastnostem je JSON ideálním jazykem pro výměnu dat. [65]

Docker je platforma určená pro vývoj, nasazení a běh programů. Umožňuje spuštění aplikace v izolovaném prostředí - kontejneru, který obsahuje veškerou digitální infrastrukturu potřebnou pro danou aplikaci. Kontejnery jsou „lightweight“ a obsahují vše potřebné ke spuštění aplikace. Při instalaci se tedy nemusí spoléhat na to, co je nainstalováno na hostitelském zařízení. Kontejnery můžou být při práci sdíleny s jistotou, že každý, s kým jsou sdíleny, dostane stejný kontejner, který funguje stejným způsobem. [66]

V kontextu aplikace to tedy znamená, že při instalaci nebude potřeba řešit existenci daných verzí Pythonu a knihoven, ale postačí samotná instalace Dockeru a přístup k rozběhnutí kontejneru. To se může velmi hodit, jelikož navrhovaná aplikace bude pravděpodobně potřebovat spoustu knihoven třetích stran a dalších závislostí.

MongoDB je volně dostupná dokumentová databáze, která ukládá data ve flexibilních JSON formátech. Model dokumentu se mapuje na objekty v kódu aplikace, což usnadňuje práci s daty. [67]

Co se týká práce s daty, hlavním účelem aplikace bude pouze ukládání a načítání výsledků průzkumu sítě a jeho odesílání uživateli v JSON formátu. Technologie MongoDB byla vybrána jako vhodná databáze pro perzistentní uchování těchto dat.

3.2 Principy návrhu

Cílem návrhu je kromě splnění požadavků také vytvoření srozumitelné a stabilní architektury a rozvržení tříd, které budou zaručovat rozšiřitelnost a testovatelnost budoucího kódu.

Již při návrhu byl brán ohled a kladen důraz na **SOLID** - 5 základních principů objektově orientovaného programování, které bude jakožto programovací paradigma využito během většiny implementace.

Single-Responsibility Principle: Každá třída by měla vykonávat pouze jednu úlohu.

Open-Closed Principle: Objekty nebo entity by měly být otevřené k rozšíření, ale uzavřené k úpravám.

Liskov Substitution Principle: Každá podtřída nebo odvozená třída by měla být nahraditelná svou základní nebo rodičovskou třídou.

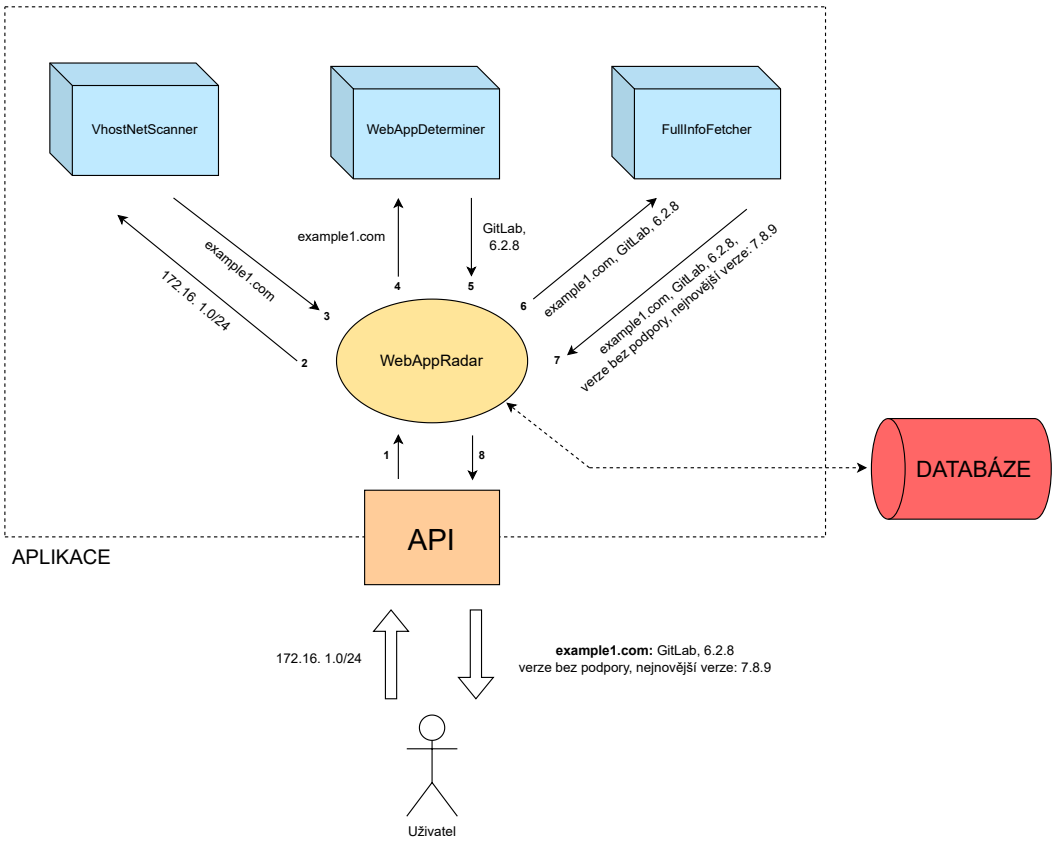
Interface Segregation Principle: Žádná třída by nikdy neměla být nucena implementovat rozhraní, které nepoužívá, nebo záviset na metodách, které nepoužívá.

Dependency Inversion Principle: Entity musí záviset na abstrakcích, nikoli na konkrétních implementacích. [68]

3.3 Obecná architektura

Aplikaci je rozdělena do několika modulů - logických celků či tříd - které budou přijímat požadavky na sken sítě a poskytovat uživateli výsledné informace o nalezených webových aplikacích srovnaných s nejnovějšími verzemi. Náhled na architekturu aplikace je zachycena do diagramu

3.1). V diagramu lze díky pořadí šipek také pochopit pořadí, ve kterém se využívají jednotlivé logické celky aplikace během skenu sítě. Celý proces průzkumu sítě, detekce webových aplikací a jejich srovnání s aktuálně dostupnými verzemi je pak detailně znázorněn v sekvenčním diagramu 3.12.



Obrázek 3.1 Rozvržení architektury aplikace do několika modulů

3.4 API

Uživatel bude interagovat s aplikací pomocí HTTP API, které bude sloužit jako jasný prostředník mezi ním a aplikací. Uživatel bude oddělen od jakékoliv vnitřní logiky aplikace a ke komunikaci a získání požadovaných výstupů bude používat pouze zdokumentované entity. Tento způsob interakce s uživatelem s sebou nese několik výhod:

Nezávislost: Vysílat a příjmat HTTP komunikaci je velmi jednoduché nehledě na operačním systému či samotném zařízení.

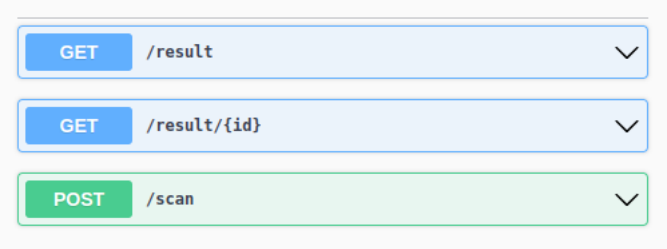
Široké možnosti na straně klienta: Pokud by se kdokoliv v budoucnu rozhodl vytvořit sofistikovanou formu klientské aplikace, např. webové rozhraní, desktopovou aplikaci, mobilní aplikaci atp., serverová logika může být zachována pro všechny případy.

Volná interpretace výsledků: Uživateli není řečeno, co s výsledky dělat, ale pouze jsou poskytnuty a uživatel se o dalším zpracování rozhodne sám.[2]

[2]Toto s sebou ovšem nese další nutné zpracování dat na straně uživatele.

Škálovatelnost: V případě bezestavového API, tedy rozhraní, které si nebude pamatovat žádné informace o klientovi, s kterým aktuálně komunikuje nebo komunikovalo, je tak získána možnost aplikaci horizontálně škálovat pro zvýšení výkonu.

API bude obsahovat 3 endpointy, které postačí pro veškerou komunikaci. k jejich zdokumentování poslouží Swagger (OpenAPI) UI [69] (viz 3.2). Kompletní dokumentace API včetně detailních popisů endpointů a metod s příklady je součástí přílohy této práce.



Obrázek 3.2 Základní přehled API endpointů v dokumentaci OpenAPI

3.4.1 Zahájení skenu (/scan)

K poslání požadavku bude zapotřebí použití HTTP metody, která umožňuje vyplnění dat do těla požadavku. Toto umožňuje například metoda POST. Uživatel pošle seznam síťových rozsahů či IP adres (nebo kombinaci obou), které si přeje prozkoumat.

- V případě odeslání nevalidního vstupu uživatel obdrží chybovou odpověď s kódem 400 (Bad Request).
- Při návrhu se musí brát také v potaz pravděpodobná výkonnostní náročnost samotného skenu, obzvlášť v případě průzkumu velkého síťového rozsahu. Nehledě na to by také nemělo dojít k tomu, aby se aplikace stala terčem velmi frekventovaných požadavků v krátkém čase, které by mohly mít za následek selhání aplikace. Proto došlo k dohodě se zadavatelem, že aplikace nebude povolovat běh více skenů najednou. Toto dodatečné omezení by ostatně nemělo zhoršit kvalitu aplikace, jelikož se nepředpokládá potřeba paralelních skenů. Pokud uživatel vyšle zmíněný požadavek v čase, kdy probíhá jiný sken, obdrží chybovou odpověď s kódem 409 (Conflict).
- V kladném případě uživatel obdrží odpověď s přiděleným unikátním ID pro daný sken. Toto ID může uživatel využít v jiných endpointech.
- Sken bude probíhat asynchronně, tedy uživatel ihned obdrží odpověď o zahájení, ale samotný sken bude probíhat na jiném vlákně.

3.4.2 Získání souhrnu všech skenů (/result)

Uživatel dostane možnost vyžádat si všechny historické skeny, seřazené podle času od nejnovějšího po nejstarší. Požadavek bude prováděn metodou GET a odpověď nebude obsahovat kompletní výsledky jednotlivých skenů, protože by byla příliš obsáhlá. Odpověď bude v těle obsahovat seznam entit s následující strukturou:

- **ID**: unikátní ID skenu
- **Čas dokončení skenu**

- **Status**: jednoslovný výsledek signalizující úspěšnost provedeného skenu, např. „success“ nebo „fail“

Tento endpoint uživateli například umožní získat ID posledního skenu, který dále využije v dalším endpointu.

3.4.3 Získání výsledku skenu (/result/{id})

Uživatel pomocí metody GET zašle společně s parametrem „id“ požadavek na získání kompletních výsledků skenu. V odpovědi budou zahrnuty veškeré zjištěné informace o webových aplikacích v síti.

Odpověď bude začínat obecnými informacemi o skenu:

- **ID**: unikátní ID skenu
- **Čas dokončení skenu**
- **Status**: jednoslovný výsledek signalizující úspěšnost provedeného skenu, např. „success“ nebo „fail“
- **Seznam cílů**: seznam síťových rozsahů nebo IP adres, které byly prozkoumávány v rámci skenu

Dále bude odpověď obsahovat seznam entit. Každá entita bude zastupovat veškeré informace zjištěné o jednom doménovém jménu. Pokud sken nenajde v rámci síťových rozsahů žádná zařízení, na kterých neběží webová aplikace, pak bude tento seznam prázdný. Každá entita bude nést následující strukturu:

- **Doménové jméno („hostname“)**: objevená webová stránka, např.: *git.example.io*
- **Název**: název webové aplikace, např.: *GitLab*
- **Verze**: aktuální verze, v které je webová aplikace nasazena, např.: *7.4.8*
- **Nejnovější verze**: nejnovější dostupná verze, která existuje pro danou webovou aplikaci, např.: *8.0.3*
- **Nejnovější patch**: nejnovější verze v rámci stejného cyklu. Předchozí informace o kompletně nejnovější verzi softwaru uživatele často nemusí zajímat, protože aktualizace na ní může obnášet změnu rozhraní či funkcionalit nebo dokonce pořízení nové licence. Často tedy uživatele zajímá pouze verze, na kterou by měl software aktualizovat v rámci stejného cyklu bez změn funkcionalit. Důvod pro takové aktualizace jsou často opravy chyb nebo bezpečnostních zranitelností v aplikaci. Příklad: *7.4.11*
- **Konec podpory**: boolovská hodnota informující uživatele, že jeho aktuální verze již není oficiálně podporována
- **Datum konce podpory**: datum, kdy dojde ke skončení platnosti podpory pro aktuálně nasazenou verzi webové aplikace

Ne vždy se aplikaci podaří zjistit kompletní informace o dané adrese. Proto budou **všechny položky kromě samotného doménového jména nepovinné**.

3.5 VhostNetScanner

První modul aplikace byl pro účely návrhu pojmenován jako *VhostNetScanner*. Účelem tohoto logického celku bude průzkum síťového rozsahu, z kterého vzejde seznam doménových jmen všech webových aplikací, které jsou nasazeny na zařízeních v této síti. Chování modulu bude definovat rozhraní *IVhostNetScanner*.

3.5.1 WebServerVhostNetScanner

Jedná se o implementaci *IVhostNetScanner*, která bude zjišťovat doménová jména pomocí průzkumu webových serverů. Třída bude přijímat následující dvě rozhraní:

IWebServerScanner: Pro daný síťový rozsah bude schopné zjistit všechny IP adresy, kde běží nějaký webový server.

IVhostDiscoverer: Rozhraní sloužící k získání všech doménových jmen, které hostuje webový server na dané adrese.

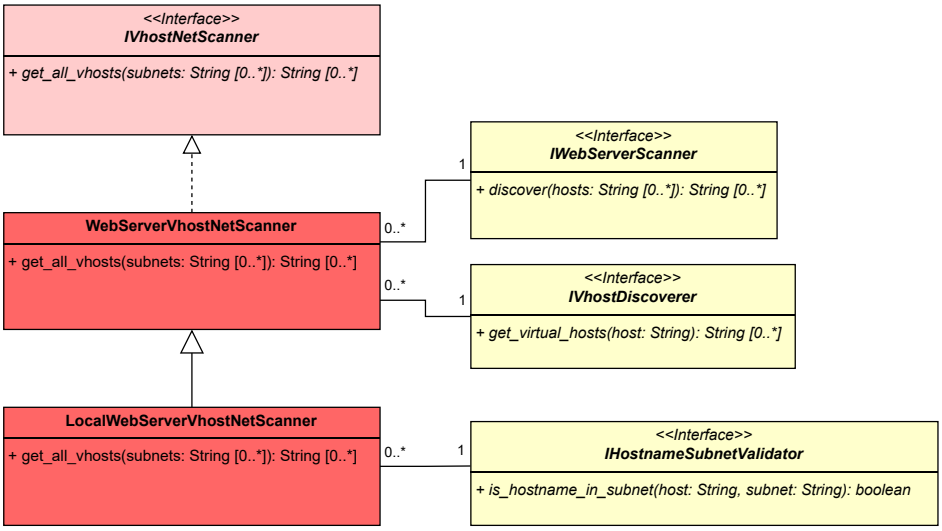
Pomocí těchto dvou rozhraní třída umožní proskenovat síť, zjistit na kterých adresách běží webový server a pro každou tuto adresu získá všechny doménová jména, která webový server hostuje.

3.5.2 LocalWebServerVhostNetScanner

Tato třída bude rozšiřovat *WebServerVhostNetScanner* o kontrolu, zda všechny nalezená doménová jména skutečně vedou na webové aplikace nasazené na adresách v dané zkoumané síti (a nejsou např. hostované na daných webových serverech pouze v rámci load balacingu). Třída bude tedy navíc potřebovat následující rozhraní:

IHostnameSubnetValidator: Umožní zkontrolovat, zda se doménové jméno skutečně vyskytuje v rámci zadaného síťového rozsahu. Jinými slovy, toto rozhraní je určeno k validaci, zda webová aplikace přístupná pod daným doménovým jménem ukazuje v rámci DNS na IP adresu, která patří do zadaného rozsahu.

Třída obohacená o možnost využití tohoto rozhraní má vše potřebné k realizaci **F2** (2.1.1.2).



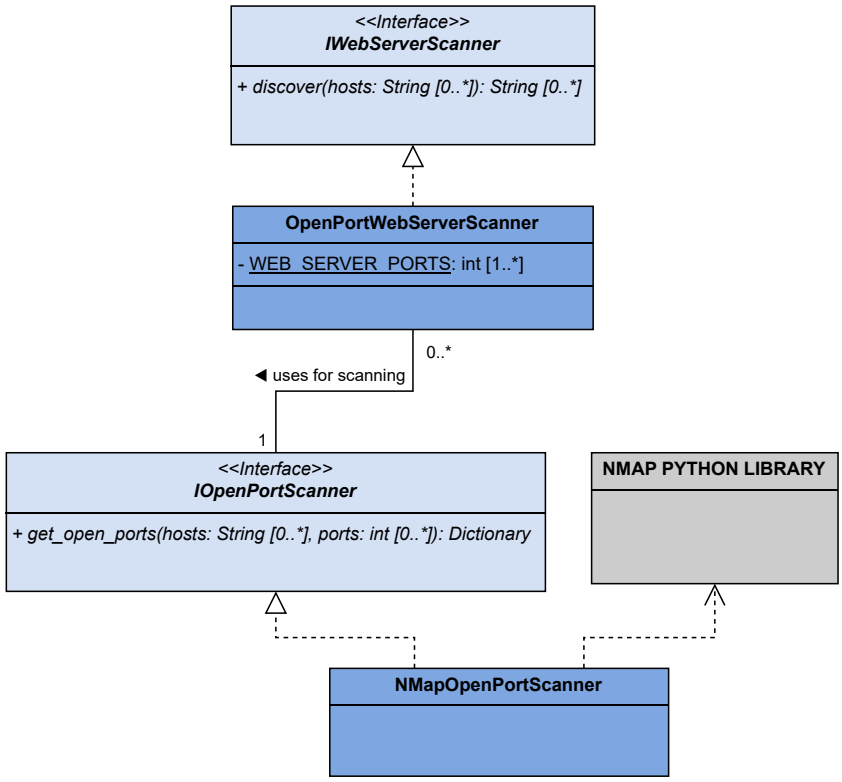
Obrázek 3.3 Třída LocalWebServerVhostNetScanner umožňující nalezení všech doménových jmen webových aplikací nasazených v rámci daného síťového rozsahu

3.5.2.1 OpenPortWebServerScanner

Jedná se o třídu, která implementuje rozhraní *IWebServerScanner* (viz 3.5.1). Webové servery bude vyhledávat na základě otevřených portů 80 (HTTP) a 443 (HTTPS).

K získání požadovaného výsledku bude potřebovat třídu, která bude schopná samotný sken vykonat, tedy pro daný síťový rozsah a seznam portů přijít se seznamem adres v tomto rozsahu s jejich otevřenými porty ze vstupu. K tomu poslouží rozhraní *IOpenPortScanner*, resp. jeho implementace *NMapOpenPortScanner*, která ke skenování využije technologii Nmap.

Třída OpenPortWebServerScanner vezme výsledky skenu IOpenPortScanner a pokud pro danou adresu bude alespoň jeden z portů 80 či 443 otevřený, pak bude adresa označena za adresu s aktivním webovým serverem. Možné nevýhody tohoto přístupu jsou rozebrány v kapitole o analýze (viz 2.4.1).



Obrázek 3.4 IWebServerScanner a jeho navržená implementace pomocí NMap

3.5.2.2 SshAgentVhostDiscoverer

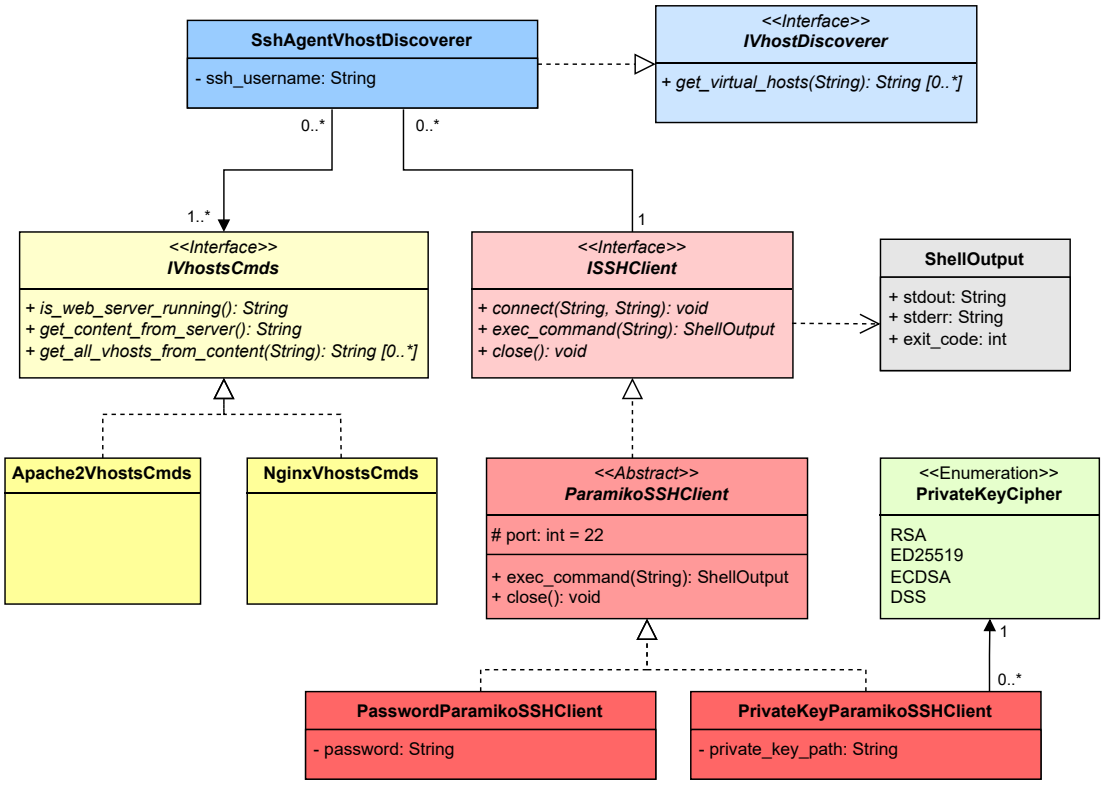
Tato třída bude implementovat rozhraní *IVhostDiscoverer* (viz 3.5.1). K získání všech doménových jmen hostovaných na dané adrese použije vyčítání těchto informací z konfiguračních souborů webových serverů, které navštíví přes SSH protokol. K tomu bude potřebovat následující:

- SSH klienta, přes kterého bude schopná navštívit server a zadávat a číst jednotlivé shell příkazy. Pro implementaci bude využita Python knihovna *Paramiko*. Bude tedy vytvořena třída používající tuto knihovnu. Třída bude abstraktní a na potomkovi bude ponechávat implementaci přihlášení na server. Navrženy budou 2 implementace umožňující jednak přihlášení pomocí hesla a jednak zašifrovaného klíče,
- uživatele, pomocí kterého bude na server skrz SSH přistupovat,
- seznam podporovaných webových serverů a instrukce, které bude SSH klient používat k vyčtení jednotlivých konfigurací.

  Každý webový server pravděpodobně ukládá své konfigurační soubory s potřebnými daty na jiná místa. Kromě toho také používá jinou syntaxi k zadefinování virtuálních hostů. Programátor tedy musí přijít s jasnou definicí těchto informací pro každý jeden webový server (Nginx, Apache atd.). Tyto definice budou mít jedno společné rozhraní, které bude moci třída *SshAgentVhostDiscoverer* polymorfně využívat.

  Pro tyto účely tedy vznikne rozhraní *IVhostsCmds*, jehož metoda *is_web_server_running* vrátí ve formě řetězce příkaz, který po vykonání na daném Debian serveru zjistí, zda jeho určitý webový server běží (SSH klient by měl umět zkontrolovat návratový kód příkazů). Dále bude také obsahovat metodu *get_content_from_server*, která vrátí příkaz k získání

všech nutných konfigurací v jednom řetězci. Tento výstup může být dále zpracován metodou *get_all_vhosts_from_content*, jenž z něj vyextrahuje jednotlivá doménová jména.



Obrázek 3.5 Realizace IVhostDiscoverer pomocí průzkumu serverových konfigurací po SSH

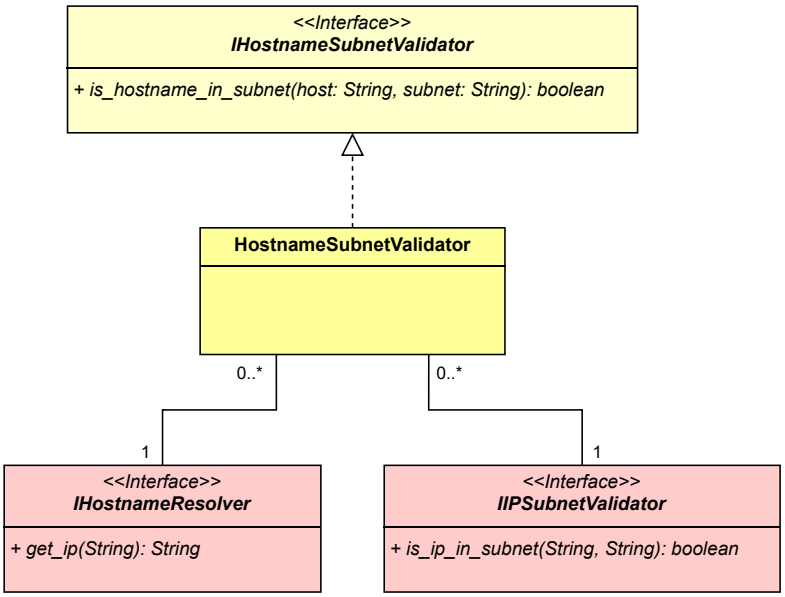
3.5.2.3 HostnameSubnetValidator

Tato třída bude implementovat rozhraní *HostnameSubnetValidator* (viz 3.5.2). K validaci, zda se dané doménové jméno přijímané v její metodě zpětně přeloží na IP adresu patřící do síťového rozsahu z dalšího argumentu metody, využije:

IHostnameResolver: Rozhraní, které bude umožňovat překlad doménového jména na IP adresu.

IIPSubnetValidator: Rozhraní, které bude schopné zvalidovat, zda určitá IP adresa patří do určitého síťového rozsahu.

V implementaci metody *is_hostname_in_subnet* této třídy půjde pouze o zkombinování metod těchto dvou rozhraní.



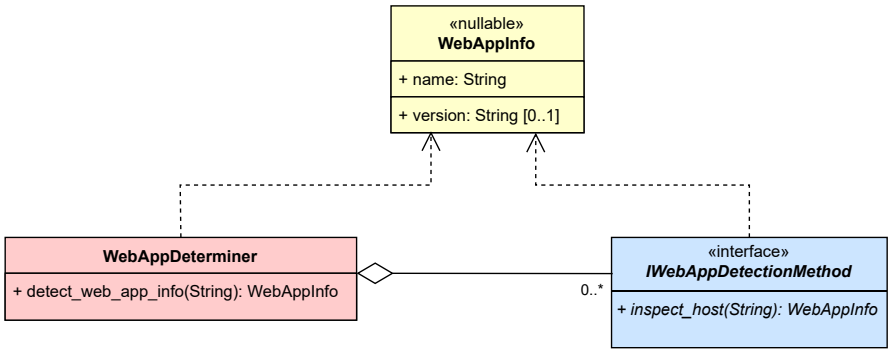
Obrázek 3.6 HostnameSubnetValidator kombinující překlad doménového jména na IP adresu s validací síťového rozsahu

3.6 WebAppDeterminer

Dalším modulem, který bude tvořit jádro aplikace, je třída *WebAppDeterminer*. Ta bude obsahovat důležitou metodu, která:

- Jako vstup bude přijímat doménové jméno.
- Návratovou hodnotou bude instance třídy WebAppInfo, která bude obsahovat informaci o názvu a verzi webové aplikace (verze bude nepovinná) nasazené pod daným doménovým jménem. Návratová hodnota bude moci být nulová v případě, že na dané adrese pravděpodobně neběží žádná webová aplikace.

Z analýzy (viz 2.4.2) vyšlo najevo, že nejvhodnějším způsobem k detekci informací o webové aplikaci na dané adrese, bude hledaní určitých HTML elementů po navštívení adresy. Při návrhu bylo bráno v potaz, aby aplikace mohla být v budoucnu rozšířena i o jiné způsoby a navíc byla schopna je kombinovat. Proto třída *WebAppDeterminer* v konstruktoru obdrží kolekci rozhraní *IWebAppDetectionMethod*. Ty budou obsahovat metodu, která se pokusí zjistit požadované informace o webové aplikaci pro dané doménové jméno. V případě, že daný způsob zjištění neuspěje, metoda vrátí nulový výstup.



Obrázek 3.7 Vztah mezi třídou WebAppDeterminer a rozhraním IWebAppDetectionMethod

3.6.1 HtmlContentParsingMethod

Jednou z implementací rozhraní *IWebAppDetectionMethod* bude *HtmlContentParsingMethod*. Tato třída se bude pokoušet získávat data o dopředu neznámé aplikaci pomocí hledání určitých elementů v HTML obsahu stránky.

3.6.1.1 WebAppRule

Tato třída bude určená k specifikování toho, jak se z obsahu webové stránky pozná, jaké konkrétní webové aplikaci náleží. Každé webové aplikaci bude náležet právě jedna instance této třídy. Metodou *HtmlContentParsingMethod* budou moci být detekovány pouze takové webové aplikace, pro které bude existovat instance třídy *WebAppRule*.

Třída bude obsahovat:

- Unikátní identifikátor (řetězec ve formě regulárního výrazu), který bude v obsahu stránky vyhledán. V případě, že bude nalezen, tak dané doménové jméno s tímto obsahem bude označeno za webovou aplikaci s názvem z nadcházejícího bodu. V tomto úspěšném případě bude průzkum webového obsahu pokračovat ke zjištění verze webové aplikace.
- Název webové aplikace
- Řetězec ve formě regulárního výrazu s parametrem, který bude sloužit k extrahování verze z obsahu webové stránky.
- Relativní URL cestu, kde se nachází informace o verzi. Někdy se verze totiž nenachází přímo na hlavní stránce. Tento nepovinný atribut přichází s možností specifikace speciální cesty, která bude přidána k samotnému doménovému jménu pro vyhledání verze.
- Instanci třídy *Auth* (viz 3.6.1.2), která umožní automatické vyplnění přihlašovacích údajů a proniknutí za přihlašovací bránu skrývající informace o aktuální verzi.

3.6.1.2 Auth

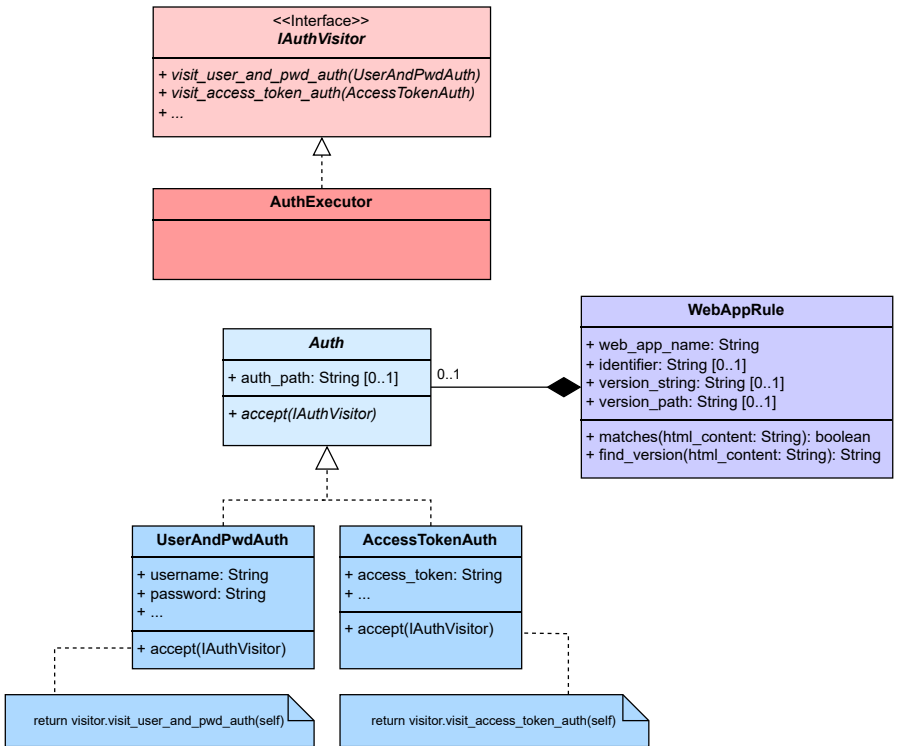
Tato třída bude umožňovat průnik za přihlašovací bránu. Tento průnik je u některých webových aplikací nutný k nalezení verze, která je bez přihlášení skrytá.

Jedná se pouze o abstraktní třídu. Způsobů přihlašování může existovat několik (například autentizační token, jméno a heslo atd.). Jednotlivé způsoby budou tedy potomky této třídy.

V každém případě, stejně jako u verze, bude autentizace umožněna na jiném URL než čistě na hlavní straně.

3.6.1.3 IAuthVisitor

Konkrétní implementaci autentizace a získávání dat z webové stránky bude věnován prostor v kapitole Implementace (viz 4.3.1). Již při návrhu ale bylo jasné, že různé autentizační metody mohou probíhat dosti různě. Z hlediska správného objektového návrhu by bylo vhodné ponechat logiku přihlášení na jiné samostatné třídě, kterou bude samotný potomek třídy Auth schopný volat bez ohledu na jeho specifikaci. Až tato třída bude obsahovat všechny nutné informace a objekty, které povedou k pokusu o autentizaci. Tomuto požadavku náramně vyhovuje návrhový vzor *Visitor pattern*. [3]



Obrázek 3.8 Autentizace ve formě návrhového vzoru Visitor pattern

3.6.1.4 UserAndPwdAuth

Ačkoliv je v předchozím diagramu (3.8) naznačena implementace dvou tříd abstraktní třídy Auth, skutečně implementována bude jen *UserAndPwdAuth*, neboť všechny aplikace ze seznamu zadavatele, které vyžadují autentizaci, podporují právě přihlášení uživatelským jménem a heslem.

Aby přihlášení bylo možné, bude potřeba zadefinovat, kde se přihlašovací boxy (pro uživatelské jméno a heslo) v obsahu stránky nachází. Třída UserAndPwdAuth obsahuje dva seznamy instancí třídy *HTMLElementParam*, která slouží k lokalizaci HTML elementu v DOM. Tímto způsobem bude moci být například řečeno, že vstupní box (input box) pro zadání uživatelského jména obsahuje atribut „key“ s hodnotou „value“ (ukázový vstupní box 3.1 by mohl být např. lokalizován pomocí *key* - „id“ a *value* „username“ či pomocí *key* - „name“ a *value* „os_username“).

[3] Visitor pattern poskytuje snadný a udržitelný způsob, jak reprezentovat operaci, která má být provedena na prvcích struktury objektu. Umožňuje snadno zadefinovat novou operaci, aniž by bylo potřeba měnit třídy prvků, se kterými samotný objekt pracuje. Aby toho bylo dosaženo, Visitor pattern definuje operaci v samostatné třídě nazývané *Visitor*. Tím se operace oddělí od samotného objektu (či kolekcí objektů), se kterou pracuje. Pro každou novou operaci, která má být definována, se vytvoří nová třída Visitor. [70]

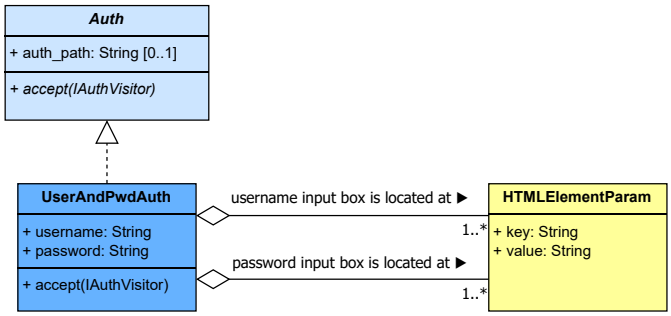
Jak pro heslo, tak uživatelské jméno lze zadat těchto parametrů několik, pokud by například dva vstupní boxy obsahovaly stejný atribut, ale lišily se v jiném.

Třída také obsahuje atribut se samotným uživatelským jménem a heslem.

Jedná se o jednofaktorové přihlášení. Tento způsob autentizace nebude funkční v případě, že by webové aplikace vyžadovala dvoufaktorové ověření, což by pravděpodobně vyžadovalo vytvoření potomka této třídy.

```
<input class="example" id="username" name="os_username" type="text">
```

Listing 3.1 Příkladný vstupní box (input box) v DOM webové stránky



Obrázek 3.9 Třída UserAndPwdAuth obsahující nutné informace k automatickému přihlášení

3.6.1.5 HTMLContentParsingFromFileMethod

Ke splnění funkčního požadavku o rozšiřitelnosti (viz 2.1.1.6) bude navržena třída *HTMLContentParsingFromFileMethod*, která bude rozšiřovat třídu *HtmlContentParsingMethod* o načítání instancí *WebAppRule* ze souboru. Tímto způsobem bude zadefinovaná podpora pro extrahování informací z obsahu stránek pro všechny webové aplikace na jednom místě. Pokud tvůrci webové aplikace nějakým způsobem změní rozložení informací v obsahu stránky (např. informace o verzi se bude nacházet na jiném místě) nebo uživatel aplikace bude chtít databázi podporovaných webových aplikací rozšířit, bude stačit pouze upravit soubor dle zadané specifikace v uživatelském manuálu. Třída bude navržena tak, aby mohla podporovat různé formáty souboru.

3.7 FullInfoFetcher

Třetí modul bude přijímat doménové jméno (např. *example.org*), název webové aplikace (např. *GitLab*) a případně její verzi (např. *6.8.11*). Na základě tohoto a informací stažených z internetu bude poskytovat porovnání a veškeré možné informace ohledně aktuálnosti právě nasazené verze.

Modul by měl být schopný získávat z internetu taková data, která budou moci poukázat na to, že daná verze softwaru již není aktuální nebo dokonce podporována. Výsledné shrnutí bude v kódu zastupovat třída *FullWebAppInfo* zkombinovaná se samotným doménovým jménem.

Výsledek tohoto modulu je tedy struktura informací, s kterou se do kontaktu dostává uživatel celé aplikace. Ne vždy bude možné pro danou webovou aplikaci tyto informace dohledat, proto jsou všechny atributy *FullWebAppInfo* nepovinné.

FullInfoFetcher bude implementován samostatnou třídou, jenž bude pro své fungování potřebovat další dva logické prvky:

- Třídu, která bude schopná pro daný název webové aplikace stáhnout seznam všech jejich dostupných verzí. Kromě seznamu samotných verzí by se měla třída také pokoušet o zjištění podpory jednotlivých verzí. Tato třída bude nést název *ReleaseFetcher*.

- Seznam komparátorů, které budou schopný vzít předchozí seznam vydaných verzí a aktuální informace o zkoumané webové aplikaci, tyto dvě věci porovnat a vrátit požadovaný výsledek.

  Každá webová aplikace může používat jiný způsob verzování, a tak porovnání nebude vždy stejné. Existuje například sémantické verzování, při kterém se porovnává nejprve nejvyšší část verze (major), poté prostřední (minor) a nakonec třetí (patch) část, všechny oddělené tečkou (viz 1.1.4 - sémantické verzování). Ne všechny aplikace ale tento způsob používají. Proto je pro každou podporovanou aplikaci nutné přijít s komparátorem - porovnávacími pravidly - na základě kterých se budou porovnávat dostupné vydané verze s aktuální verzí.

  Dále by bylo vhodné počítat s nějakým výchozím komparátorem, který bude použit, pokud pro danou webovou aplikaci není určený nějaký jiný způsob porovnání.

  Každý komparátor bude implementovat rozhraní *IVersionComparator*.

3.7.1 VersionCycleInfo

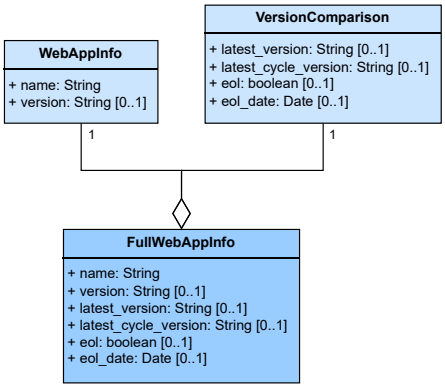
Při stahování informací o dostupných verzích a platnosti jejich podpory často není nutné, a v některých případech ani možné, je získávat pro každou jednu konkrétní verzi. Často budou tyto informace dostupné pro nějakou nadskupinu verzí. Pro tuto nadskupinu vznikne třída *VersionCycleInfo*. Ta bude v sobě nést informace o označení dané nadskupiny (v případě sémantického verzování například „8.2“, tedy *major.minor* část verze), platnosti podpory (boolean), případně data vypršení platnosti podpory a celkovou nejnovější dostupnou verzi dané nadskupiny (v případě sémantického verzování například „8.2.11“, tedy *major.minor.patch* část verze).

Tento mechanismus se dá využít i v případě, že verze nebude rozdělitelná na nadskupiny a podskupiny (např. pokud by se verze skládala jen z jednoho samostatného čísla). V tom případě bude instance *VersionCycleInfo* zastupovat celou verzi.

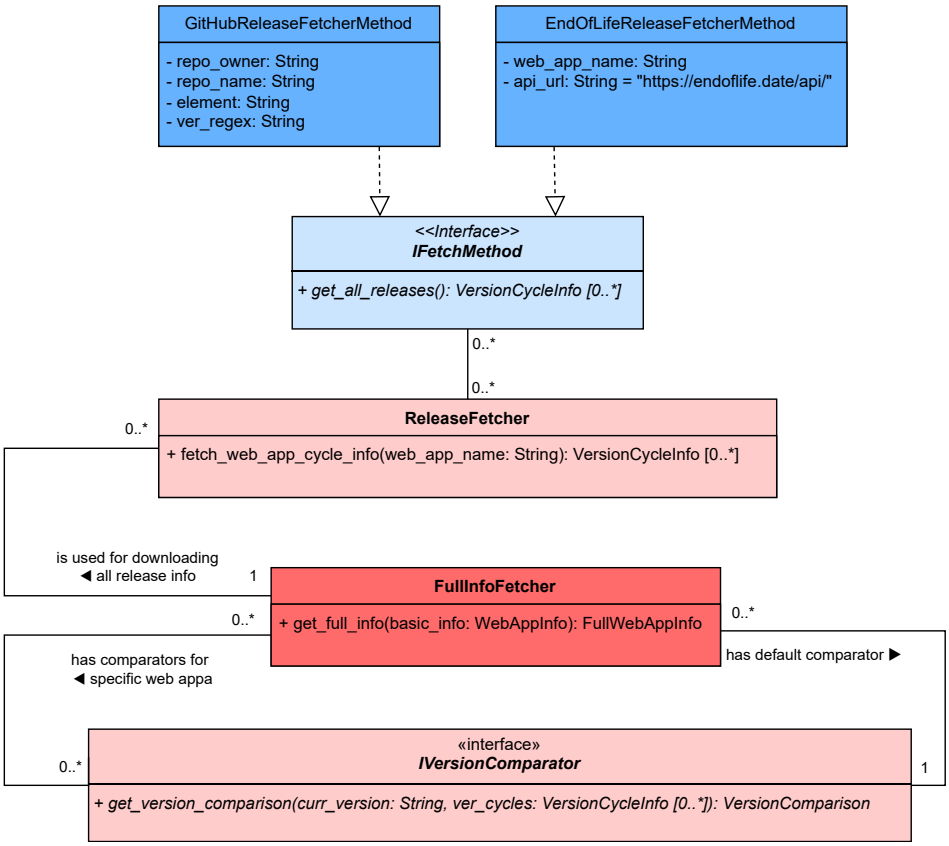
3.7.2 VersionComparison

Výsledkem porovnání všech dostupných vydaných verzí (instancí *VersionCycleInfo*) a aktuálně nasazené verze (instance *WebAppInfo*) bude instance třídy *VersionComparison*. Ta bude nést informaci o nejnovější verzi v rámci stejné nadskupiny nasazené verze, celkově nejnovější verzi, která pro aplikaci existuje, a informaci o platnosti podpory pro nadskupinu nasazené verze.

Výsledný výstup celého modulu *FullInfoFetcher* bude, jak již bylo zmíněno dříve, instance třídy *FullWebAppInfo*, jenž bude kombinovat aktuální informace o nasazené webové aplikaci (*WebAppInfo*) se srovnáním s dostupnými verzemi (*VersionComparison*) viz diagram 3.10.



Obrázek 3.10 Struktura FullWebAppInfo složená z WebAppInfo a VersionComparison



Obrázek 3.11 FullInfoFetcher a jeho pomocné třídy nutné k získávání kompletních informací o dané webové aplikaci

3.7.3 ReleaseFetcher

Jak již bylo naznačeno dříve, tato třída bude sloužit k získání informací o všech vydaných verzích. Tyto informace bude poskytovat ve formě kolekce instancí *VersionCycleInfo*.

Třída bude v konstruktoru přijímat datovou kolekci *dictionary*, jejíž klíčem bude název webové aplikace a hodnotou rozhraní *IFetchMethod*. Pomocí těchto dat bude schopná poskytovat požadované informace.

3.7.4 IFetchMethod

Implementace tohoto rozhraní budou schopny používat různé druhy mechanismů (např. volání API, web scraping aj.) k získání informací o všech dostupných verzích. Rozhraní bude obsahovat jedinou metodu, *get_all_releases*, která bude vracet kolekci instancí *VersionCycleInfo*.

Z analýzy vyšlo najevo, že pro většinu aplikací ze seznamu zadavatele postačí použít Endoflife.date API a GitHub API. Budou tedy navrženy dvě třídy reprezentující tyto dva způsoby. V případě aplikací, jejíž informace nejsou součástí webové stránky Endoflife.date a neposkytují volně dostupný GitHub repozitář, bude v budoucnu pro podporu srovnání s aktuálními verzemi potřeba přijít s dalšími třídami, implementující rozhraní *IFetchMethod*.

3.8 EndOfLifeReleaseFetcherMethod

Tato třída implementující rozhraní IFetchMethod bude k získání informací používat *Endoflife.date* API. Pro odesílání požadavků bude potřebovat název dané webové aplikace ve formě řetězce podporovaného zmíněnou webovou stránkou.

3.9 GitHubReleaseFetcherMethod

Další třída implementující rozhraní IFetchMethod bude k získání informací používat GitHub API. Kromě základních informací (viz 2.4.3.2) o daném repozitáři projektu bude z těla přijaté odpovědi ve formě JSON pole potřeba také vybrat potřebné informace o verzi (vzhledem k časté absenci informací o podpoře daných verzí budou touto metodou získávány pouze samotné verze). K tomu bude tudíž potřeba zadefinovat element (klíč JSON objektu), určující, kde se informace o verzi pro daný release vyskytuje, a regulární výraz s parametrem pro extrakci dané verze.

3.10 WebAppRadar

K propojení modulů *VhostNetScanner*, *WebAppDeterminer* a *FullInfoFetcher* poslouží třída *WebAppRadar*. Ta bude kromě instancí těchto tří logických celků dále v konstruktoru vyžadovat rozhraní *IScanRepository* pro komunikování s databází.

3.10.1 IScanRepository

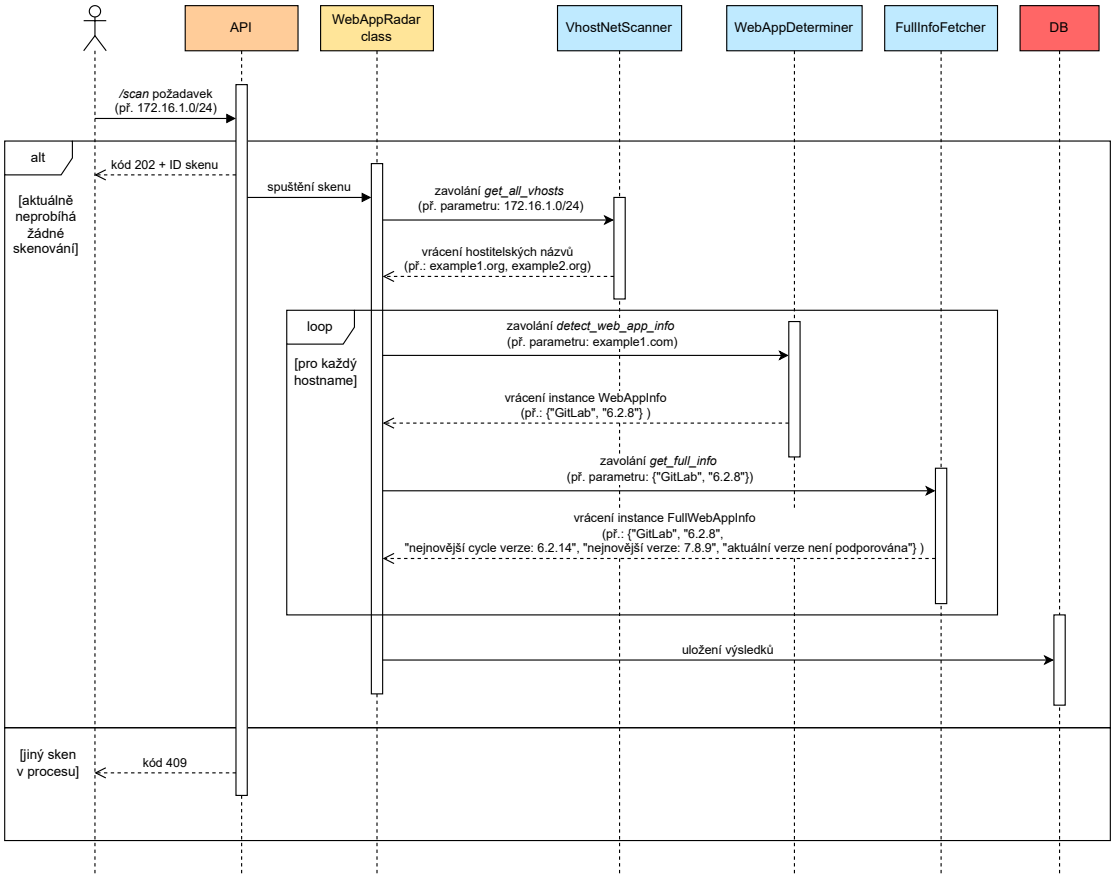
Jedná se o rozhraní perzistentní vrstvy aplikace. Tato vrstva slouží k práci s daty, která mohou být uchovávána v rámci různých běhů aplikace. Rozhraní bude definovat tři metody, díky nimž bude možné:

1. uložit výsledek skenu sítě,
2. načíst shrnutí všech posledních skenů,
3. načíst detaily konkrétního skenu.

Rozhraní bude obecné a nezávislé na konkrétním způsobu ukládání dat či technologii. Pro komunikaci s MongoDB databází bude vytvořena speciální třída, která bude implementovat toto rozhraní.

3.10.2 Proces skenování sítě

Celý proces skenování sítě byl hrubě nastíněn v jednom z předchozích diagramů (3.1), jehož účelem bylo čtenářovi vysvětlit rozdělení aplikace na několik modulů. Detailní proces zahájení a průběhu skenu požadovaného síťového rozsahu ke zjištění všech dostupných informacích o webových aplikacích v této síti je zachycen na sekvenčním diagramu (3.12). Na něm je zachycena příkladová ukázka, díky které může čtenář lépe pochopit propojení a využití jednotlivých modulů, databáze a API.



Obrázek 3.12 Vyžádání skenu pro konkrétní síťový rozsah a interakce jednotlivých částí aplikace

1. Uživatel zašle na API endpoint seznam podsítí a IP adres k prozkoumání. V případě, že jiný sken ještě nebyl dokončen, je vrácena chybová odpověď.

2. Díky modulu *VhostNetScanner* bude získán seznam všech doménových jmen, které vedou na nějakou webovou aplikaci nasazenou na zařízeních na zadaných adresách.

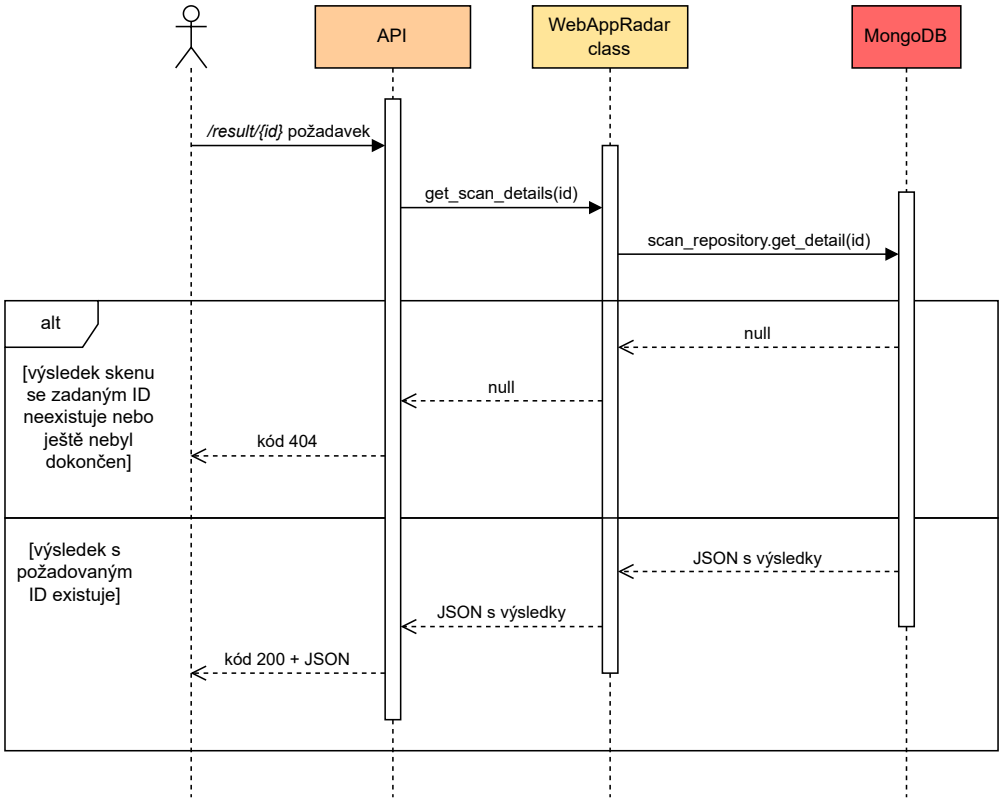
3. Pro každé doménové jméno se modul *WebAppDeterminer* snaží zjistit, jaká konkrétní webová aplikace na této adrese běží a v jaké verzi je nasazena.

4. Na základě těchto dat se poté modul *FullInfoFetcher* snaží z internetových zdrojů zjistit dostupné aktualizace a informace o platnosti podpory pro aktuální verzi. Po tomto srovnání vrátí souhrn informací o každé jedné webové aplikaci.

5. Výsledek bude uložen do databáze, aby si ji uživatel mohl přes API kdykoliv vyžádat.

3.10.3 Proces vyžádání výsledku skenu sítě

Uživatele pochopitelně zajímají výsledky skenu sítě. Ty si bude schopný vyžádat pomocí API a přiděleného unikátního ID skenu, který získal během požadavku na sken nebo pomocí jiného endpointu (viz 3.4.2). Celý proces získání výsledku skenu je zachycen na následujícím sekvenčním diagramu 3.13.



Obrázek 3.13 Proces získání požadovaného výsledku skenu sítě

Kapitola 4
Implementace

V této kapitole jsou zmíněny konkrétní postupy, které byly využity k realizaci návrhu z předchozí kapitoly, vedoucí k fungující aplikaci. Jsou zde uvedeny kusy kódu, překážky, ke kterým došlo v průběhu implementace a popis jejich řešení. V neposlední řadě kapitola obsahuje také soupis dokumentace, strukturu repozitáře projektu a splnění požadavků zadavatele.

4.1 Cíle implementace

Cílem implementace bylo realizovat návrh aplikace, která korektně splňuje funkční i nefunkční požadavky (viz 2.1). Během vývoje bylo dbáno na to, aby byl kód napsán *srozumitelně* a především dostatečně *modulárně* tak, aby mohl být *znovu využíván* a *dobře otestován*.

Dalším cílem bylo komentování tříd, rozhraní a metod, stejně tak jako vytvoření krátkých komentářů složitějších kusů kódu. Nejenom že tato praktika pomůže případným dalším vývojářům, kteří budou chtít projekt rozšířit nebo změnit, ale také může sloužit jako potřebný stavební kámen k vygenerování dokumentace kódu.

Výsledkem implementace je tedy kód dodržující kromě již zmíněných zásad následující správné principy programování:

KISS: *Keep it simple, stupid* je návrhový princip, který říká, že jednoduché řešení je lepší než složité. Zásada KISS upřednostňuje srozumitelnost před zbytečnou komplexitou. Řešení může vypadat „hloupě“, ale bude srozumitelné a jednoduché. Nemá cenu, aby řešení bylo „chytré“, když by stačilo jednodušší řešení. [71]

DRY: *Don't Repeat Yourself* je princip softwarového vývoje, jehož cílem je omezit opakování vzorů a duplikaci kódu ve prospěch abstrakcí a vyhýbání se redundancí. [72]

YAGNI: *You Aren't Gonna Need It* je princip, který říká, že programátor by neměl přidávat funkce, dokud to nepovažuje za nezbytné. [73]

Princip fungování a dědičnost tříd, které budou v této sekci popsány, byl již představen. V nadcházejícím textu je uveden konkrétní kód. Popíšu řešení překážek, které se v menší míře nečekaně vyskytly a jaké nedostatky může daný implementovaný kód obsahovat.

4.2 Implementace modulu VhostNetScanner

Tato sekce navazuje na návrh logického celku *VhostNetScanner* (viz 3.5), resp. rozhraní *IVhostNetScanner*, nastínění jeho realizace a všech jeho nutných komponent.

4.2.1 NMapOpenPortScanner

Třída slouží k proskenování zadaných IP adres či síťových rozsahů na zadané otevřené porty, k čemuž využívá Nmap. Nejprve bylo potřeba se seznámit se samotným nástrojem, který nabízí command line rozhraní. Nástroj byl otestován a následně došlo k úvaze, jak nástroj zaobalit do požadované třídy. Výsledky skenů byly vypisovány na standardní výstup zachovávající určitou strukturu.

Přemýšlení o zaobalení aplikace v Python kódu doputovalo k myšlence, že zcela jistě někdo v minulosti již podobný problém řešil a pravděpodobně existuje nějaká knihovna s požadovanou funkcionalitou. V PyPi databázi balíčků byl nalezen balíček *python3-nmap* a *python-nmap*. Oba dva měly v době psaní práce veřejnou licenci. Poslední vydaná verze prvního zmiňovaného balíčku sahala do roku 2022, zatímco verze druhého do roku 2021, proto byl nejprve otestován balíček *python3-nmap*. Po prostudování dokumentace i kódu se bohužel nepovedlo zjistit, zda balíček podporuje kýženou funkcionalitu - proskenování síťových rozsahů pouze pro konkrétní porty. Došlo tedy k rozhodnutí přejít k druhému balíčku *python-nmap*, který požadavek splňoval. Pomocí jeho metod byla proskenována lokální síť a výsledky srovnány s klasickým command line nástrojem. Výsledky se shodovaly, a proto nezbylo nic jiného, než balíček využít v třídě *NMapOpenPortScanner*. Za důležitou zmínku určitě stojí to, že balíček pro své fungování potřebuje na zařízení, kde se kód používá, nainstalovaný samotný nástroj Nmap.

V následujícím kódu (4.1) je kromě samotné implementace poukázáno na určité prvky, které byly využívány na různých místech v rámci celého projektu.

- Lze si povšimnout, že metoda *get_open_ports* by měla přijímat *Iterable*, což je jakýkoliv objekt, který umožňuje iterování přes jeho prvky. [74] V jiných částech implementace se také lze setkat s objektem *Collection*, jenž oproti *Iterable* umožňuje také rychlé získání informace o počtu prvků v objektu. Tento předpis návratových a vstupních typů není kontrolovaný samotným enginem Pythonu, jelikož je to dynamicky typovaný jazyk. [61] Tento předpis slouží pouze jako „good practice“ pomůcka pro programátory, kteří díky němu mohou očekávat určité chování.

  Tento princip definování obecných nebo abstraktních vstupních či výstupních typů používaný v celé aplikaci umožňuje zvolit si konkrétní datový typ kolekce (např. *list*, *set* atd.) až během implementace samotné metody, během které nebude programátor dopředu omezen. Při návrhu rozhraní či definici vstupních a výstupních typů se pouze určí, jaké chování a vlastnosti jsou požadovány.

- Python nepodporuje privátní metody ani atributy, proto se metody, které slouží pouze k účelům uvnitř dané třídy prefixují podtržítkem.

- *FatalError* je vlastní výjimka využívaná napříč celým projektem. Pokud je někde v kódu zachycena kterákoliv výjimka, která znamená kritickou chybu znemožňující pokračování průběhu celého skenu, tak dojde k vyhození výjimky FatalError, která se zachytává až v nejvyšší vrstvě celé aplikace. Do instance výjimky je možné vložit dva řetězce - jeden, který obecně popisuje příčinu chyby, a druhý sloužící k detailnějšímu popisu. Při zachycování výjimky je první řetězec vypisován do error logu, zatímco druhý do debug logu aplikace.

- Metoda *get_open_ports* vrací *dictionary*, jehož klíčem je řetězec (doménové jméno - hostname) a hodnotou další *dictionary*, jehož klíčem je řetězec (port) a hodnotou boolean signalizující, zda je daný port otevřený.

Pro vstup *„["192.168.0.0", "192.168.0.1"], [5432, 80]“* tedy lze očekávat např. návratovou hodnotu na jednom z ukázkových kódů níže (4.2).

```python
class NMapOpenPortScanner(IOpenPortScanner):
    def __init__(self):
        super().__init__()
        self.nmap = nmap.PortScanner()

    @staticmethod
    def _process_host_input_for_nmap(hosts: Iterable[str]) -> str:
        ...

    @staticmethod
    def _process_port_input_for_nmap(ports: Iterable[int]) -> str:
        ...

    def get_open_ports(self, hosts: Iterable[str], ports: Iterable[int])
    -> Dict[str, Dict[str, bool]]:

        nmap_hosts = NMapOpenPortScanner._process_host_input_for_nmap(hosts)
        nmap_ports = NMapOpenPortScanner._process_port_input_for_nmap(ports)
        scan_results = {}
        try:
            # Launch the scan on the defined subnets and ports
            self.nmap.scan(hosts=nmap_hosts, arguments=f'-p {nmap_ports}')
        except PortScannerError:
            raise FatalError("...", "...")

        # Loop through all the hosts found
        for host in self.nmap.all_hosts():
            # Check if the host status is up
            if self.nmap[host].state() == "up":
                # Initialize the host entry in the results dictionary
                scan_results[host] = {}

                # Loop through all protocols (tcp, udp) found for the host
                for proto in self.nmap[host].all_protocols():
                    # Get all scanned ports for the current protocol
                    lport = self.nmap[host][proto].keys()

                    for port in sorted(lport):
                        # Add port and its state (True open, False otherwise)
                        scan_results[host][port] = \
                        self.nmap[host][proto][port]['state'] == 'open'
        return scan_results
```

Listing 4.1 Metoda get_open_ports třídy NMapOpenPortScanner

```
{
    "192.168.0.0": {
        "5432": true,
        "80": false,
    },
    "192.168.0.1": {
        "5432": false,
        "80": false,
    }
}
```

Listing 4.2 Příklad výstupu metody get_open_ports třídy NMapOpenPortScanner

4.2.2 SshAgentVhostDiscoverer

Tato třída využívá rozhraní *ISSHClient* k tomu, aby se pokusila o přihlášení na zadanou adresu přes SSH a provedla zde shellové příkazy pro detekci všech doménových jmen, které hostuje webový server běžící na dané adrese. Všechny příkazy implementují společné rozhraní *IVhostsCmds*. Na následujícím kusu kódu lze vidět konstruktor třídy, který přijímá všechny potřebné objekty.

```
class SshAgentVhostDiscoverer(IVhostDiscoverer):
def __init__(self, ssh_client: ISSHClient,
                   web_server_cmds: Iterable[IVhostsCmds],
                   ssh_username: str):
    self.ssh_client = ssh_client
    self.web_server_cmds = web_server_cmds
    self.ssh_username = ssh_username
```

Listing 4.3 Konstruktor třídy SshAgentVhostDiscoverer

V metodě *get_virtual_hosts* (viz 4.4) lze pak vidět proces detekce všech doménových jmen. SSH klient se snaží připojit na danou adresu. Způsob přihlášení ponechává na samotném klientovi. V případě úspěchu je zahájen cyklus, v němž se iteruje přes všechny instance *IVhostsCmds*, neboli přes všechny podporované druhy webových serverů.

V cyklu se nejprve přes SSH klienta, který od instance *IVhostsCmds* obdrží příkaz ve formě řetězce, snaží zjistit, zda daný webový server běží. Kontroluje se zde návratový kód - k úspěchu je potřeba nula.

Stejným způsobem se snaží získat výpis všech konfiguračních souborů daného webového serveru, které obsahují seznam virtuálních hostů (hledaných doménových jmen).
Metodou *get_all_vhosts_from_content* se poté z tohoto obsahu vyextrahují jednotlivé adresy.

Na řádku 18-20 si lze povšimnout, že pokud se v konfiguračních souborech objeví „localhost“ či „127.0.0.1“, přidá se do výsledného seznamu samotná adresa, která je zkoumána.

V případě, že je kdykoliv v procesu zachycena výjimka, je vrácen dosavadní seznam, který je reprezentován kolekcí *set*, jenž uchovává pouze unikátní prvky. [75]

```
def get_virtual_hosts(self, host: str) -> Iterable[str]:
    try:
        self.ssh_client.connect(host, self.ssh_username)
    except ConnectionError as e:
        logger.error(e)
        return []
    res = set()
    for cmd in self.web_server_cmds:
        try:
            res1 = self.ssh_client.exec_command(cmd.is_web_server_running())
            if res1.exit_code == 0:
                logger.debug("...")
                content_cmd = cmd.get_content_from_server()
                server_vhosts_content = \
                    self.ssh_client.exec_command(content_cmd).stdout
                vhosts = \
                    cmd.get_all_vhosts_from_content(server_vhosts_content)
                vhosts_w_localhost = \
                    {host if (vhost == "localhost" or vhost == "127.0.0.1")
                        else vhost for vhost in vhosts}
                res = res.union(vhosts_w_localhost)
            else:
                logger.debug("...")
        except Exception as e:
            logger.error(f"Error during SSH {host} inspection. {e}")
            return res
    return res
```

Listing 4.4 Metoda get_virtual_hosts třídy SshAgentVhostDiscoverer

4.2.3 Implementace IVhostsCmds

Každá implementace tohoto rozhraní přichází s podporou jiného typu konfigurace webového serveru. Kvůli požadavkům zadavatele bylo rozhodnuto implementovat podporu pro webový server Apache a Nginx.

Jak již bylo naznačeno dříve, rozhraní definuje tři metody:

- Metodu, která vrací příkaz ve formě řetězce, který po vykonání v příkazové řádce na Debian serveru vrátí kód 0, pokud je daný webový server aktivní.
- Metodu, která opět vrací příkaz ve formě řetězce, který po vykonání v příkazové řádce na Debian serveru vrátí výpis všech konfigurací v jednom řetězci obsahující všechna doménová jména hostovaná tímto webovým serverem.
- Metodu, která je schopná výpis přeměnit pouze na seznam unikátních doménových jmen.

4.2.3.1 Nginx

Debian servery pro běh aplikací často používá systemd managera.[1]

[1] Systemd je správce systému a služeb pro Linux. Od Debianu 8 („jessie“) je výchozím init systémem Debianu. [76]

Některé starší systémy mohou stále používat příkaz *service*, který spouští System V init skripty. [77]

Bylo rozhodnuto počítat s oběma variantami.

```
class NginxVhostsCmds(IVhostsCmds):
    def is_web_server_running(self) -> str:
        return ("systemctl -q is-active nginx"
                "|| (service nginx status "
                    "&& service nginx status | grep -q -e running -e active)")
```

Listing 4.5 Metoda is_web_server_running třídy NginxVhostsCmds

Shell příkaz funguje následujícím způsobem: Nejprve se přes *systemctl* zkusí ověřit, zda je služba nginx aktivní. Pokud ano, příkaz uspěje, nepokračuje dál a vrátí nulový návratový kód. Pokud ne, dojde k pokusu o zjištění statusu služby nginx přes příkaz *service*. Pokud tento příkaz neselže (tato služba existuje), spustí stejný příkaz s vyhledáním klíčového slova *active* a *running* pomocí nástroje grep[2]. Pokud je ve výpisu alespoň jedno z těchto slov obsaženo, může to signalizovat, že je služba aktivní.

Tento způsob má následující nedostatky:

- Pokud se služba jmenuje jinak než „nginx“, tento způsob nebude fungovat.
- Zároveň nebude fungovat, pokud webový server neběží jako služba.
- Pokud se na zařízení používá *System V init skriptování*, tak nalezení klíčového slova *active* nebo *running* ve výpisu statusu služby nemusí implikovat běžící službu a naopak.

Ačkoliv to není podloženo žádným věrohodným zdrojem, všechny tři případy se z dosavadních zkušeností jeví jako velmi nepravděpodobné. Zároveň nedošlo k vymyšlení žádného spolehlivějšího řešení. Řešení bylo otestováno v třech nezávislých prostředích; na lokálně nastaveném webovém serveru, v Docker kontejneru oficiálního Nginx image a především v konkrétním prostředí zadavatele. V obou případech příkaz vracel nulový návratový kód, pokud byl webový server Nginx aktivní, a nenulový, pokud nebyl aktivní, což je očekávaný výstup.

Dalším nedostatkem by mohla být skutečnost, že nalezení příslušných virtuálních hostů v konfiguračních souborech nemusí nutně znamenat fungující a dosažitelnou webovou stránku pod tímto doménovým jménem. Webové aplikaci či stránce můžou chybět nutné soubory, kvůli kterým nebude funkční. Dále nemusí být nastavená DNS konfigurace, takže stránka nebude dosažitelná. Žádná z těchto či jiných podobných okolností nebyly považovány za zásadní důvod k nezahrnutí těchto adres do výsledků.

V případě získání všech konfiguračních souborů obsahujících seznam virtuálních hostů, by se mohl použít příkaz „nginx -T“. [79] Po otestování vyšlo najevo, že tento příkaz požaduje *sudo* privilegia. Jedním z cílů implementace bylo se vyhnout tomu, aby uživatel, používaný na procházení serverů, vyžadoval administrátorská privilegia (viz. 2.1.2.7). Došlo k rozhodnutí, že z webového serveru budou vypisovány pouze konfigurace z výchozí složky „/etc/nginx/sites-enabled/“. Ačkoliv ukládání těchto konfiguračních souborů může být teoreticky nastaveno do jiné lokace, byla zvolena tato možnost, aby byla zachovaná zásada nejnižších privilegií.

```
def get_content_from_server(self) -> str:
    return "cat /etc/nginx/sites-enabled/*"
```

Listing 4.6 Metoda get_content_from_server třídy NginxVhostsCmds

[2]Nástroj grep prohledává vstupní soubory a vybírá řádky odpovídající jednomu nebo více vzorům. Vzory můžou být zadány parametrem *-e* [78]

4.2.3.2 Apache

Při zkoumání webového serveru Apache, došlo k zjištění, které zapříčinilo nepoužití naprosto stejného způsobu jako u Nginx. Pokud je totiž Apache nasazen pomocí oficiálního Docker image *httpd* [3], tak v jeho kontejneru jednak server neběží jako služba, a jednak se konfigurační soubory nenachází v tradiční lokaci „/etc/apache2“, nýbrž „/usr/local/apache2/“.

Metoda *is_web_server_running* byla tedy obohacena o pokus detekce běžícího webového serveru ve formě procesu *httpd*.[4] K tomuto pokusu dojde, pokud předchozí zmiňované dva neuspějí. Příkaz „ps ax | grep httpd“ vypíše všechny procesy, které obsahují řetězec „httpd“. Filtr „| grep -v grep“ z nich následně odfiltruje samotný proces, který je spuštěn pro vyhledání pomocí nástroje grep. Pokud nějaký takový proces existuje, návratový kód příkazu bude 0.

```
class Apache2VhostsCmds(IVhostsCmds):
    def is_web_server_running(self) -> str:
        return ("systemctl -q is-active apache2"
                "|| (service apache2 status "
                    "&& service apache2 status | grep -q -e running -e active) "
                "|| ps ax | grep httpd | grep -v grep)"
```

Listing 4.7 Metoda is_web_server_running třídy Apache2VhostsCmds

```
def get_content_from_server(self) -> str:
    return "cat /etc/apache2/sites-enabled/* /usr/local/apache2/sites-enabled/*"
```

Listing 4.8 Metoda get_content_from_server třídy Apache2VhostsCmds

4.3 Implementace modulu WebAppDeterminer

Implementace metody *detect_web_app_info* třídy *WebAppDeterminer* je jednoduchá a přímočará. V cyklu se proiterují všechny dostupné metody detekce a pokud některá z nich vrátí validní výsledek, je vrácen jako návratová hodnota.

```
class WebAppDeterminer:
    def __init__(self, detection_methods: Iterable[IWebAppDetectionMethod]):
        self.detection_methods = detection_methods

    def detect_web_app_info(self, host: str) -> Optional[WebAppInfo]:
        for detection_method in self.detection_methods:
            info = detection_method.inspect_host(host)
            if info is not None:
                return info
        return None
```

Listing 4.9 Implementace třídy WebAppDeterminer

[3] Konkrétně jde o případ image *httpd* s hash kódem *sha256:1a3d41a99f66b29d48caf1e57e9f5ed8d539ba12cafb9062488bee377f5c86ab* pro manifest digest.

[4] Httpd je jiný název pro webový server Apache. [80]

4.3.1 HtmlContentParsingMethod

Tato třída je prozatím jedinou implementací rozhraní *IWebAppDetectionMethod*. K detekci informací o daném doménovém jménu použije navštívení hlavního stránky přes prohlížeč, načtení obsahu stránky a vyextrahování potřebných informací.

4.3.1.1 Načtení webové stránky

Nejprve se musel vymyslet způsob, kterým se zautomatizuje proces navštívení a načtení obsahu stránky. Bylo jasné, že nebude stačit použít běžnou knihovnu, která zachytí obsah tak, jak je běžně zaslán do klientského prohlížeče, ale bude potřeba použít komplexnější nástroj, který je schopný zpracovat data (např. JavaScript kód) na straně klienta. Nikdy totiž nemusí být zaručeno, že například verze stránky bude obsažena přímo ve statickém HTML obsahu. Může se stát, že informace s verzí je načtena až po zpracování kódu na straně klienta. Byla tedy nalezena knihovna, která je schopná daný problém vyřešit.

Selenium je open source projekt zastřešující řadu nástrojů a knihoven zaměřených na podporu automatizace webového prohlížeče. [81] Selenium podporuje automatizaci všech hlavních prohlížečů na trhu pomocí nástroje *WebDriver*. WebDriver je rozhraní API a protokol, který definuje jazykově neutrální rozhraní pro ovládání webových prohlížečů. Pro každý prohlížeč existuje podpora specifické implementace WebDriveru, která se nazývá driver. Driver je komponenta odpovědná za delegování instrukcí na prohlížeč a zajišťuje komunikaci mezi Seleniem a prohlížečem. [82]

Pro Firefox a Chromium prohlížeče Selenium podporuje „headless“ režim, který umožňuje, že okno prohlížeče není viditelné. [83] Díky tomuto režimu může server s běžící aplikací ušetřit výkon.

Došlo tedy k rozhodnutí implementovat třídu s názvem *SeleniumRenderer*, která je schopná načíst webovou stránku a vrátit její webový obsah ve formě řetězce. Řetězec je načten celý v kuse, což by u obrovské webové stránky mohlo vést k přetečení bufferu. U webových stránek, které byly testovány k tomu nikdy nedošlo, ačkoliv řetězec mnohdy obsahoval tisíce řádků. Došlo tedy k rozhodnutí tento potenciální problém ignorovat.

```python
class SeleniumRenderer(IClientSideRenderer, ABC):
    @abstractmethod
    def _get_driver(self) -> selenium.webdriver.remote.webdriver.WebDriver:
        pass

    def __init__(self, page_load_timeout: int = 10,
                 explicit_waiting: float = 3.5):
        self.explicit_waiting = explicit_waiting
        self.driver = self._get_driver()
        self.driver.set_page_load_timeout(page_load_timeout)

    def get_page_content(self, host: str) -> str:
        try:
            self.driver.get(host)
        except TimeoutException:
            raise ConnectionError("Timeout while waiting for page to load")
        except selenium.common.exceptions.WebDriverException:
            raise ConnectionError(f"Unable to resolve: '{host}'")
        time.sleep(self.explicit_waiting)
        return self.driver.page_source
```

Listing 4.10 Třída SeleniumRenderer

- Metoda *get_page_content* použije driver, resp. prohlížeč k navštívení stránky. Poté dojde k uspání vlákna na určitou dobu, během které daný prohlížeč může načítat kód na straně klienta. Každé stránce může trvat načítání jinak dlouhou dobu. Proto bylo potřeba přijít s konkrétním časovým intervalem, který zajistí, že důležité informace budou vždy načteny, ale zároveň se nebude čekat moc dlouho, což by zpomalilo běh celé aplikace.

  Selenium podporuje sofistikovanější metody, například čekání na načtení konkrétního elementu. [84] Bohužel tato technika nemohla být využita, protože chtěné informace nebudou obsaženy vždy ve stejném elementu. Je pravděpodobné, že u každé webové aplikace se bude lokace s požadovanou informací lišit.

- Metoda, která určuje jaký konkrétní webový prohlížeč bude na pozadí využit k propojení s driverem, se nazývá *__get_driver* a její implementace, je určena na potomkovi třídy. Pro účely aplikace došlo k rozhodnutí vytvořit třídu *SeleniumChromeRenderer*, která dědí ze SeleniumRenderer, a jak napovídá název, jako prohlížeč používá Google Chrome.

Co se týče samotné třídy HtmlContentParsingMethod a její metody *__get_full_page_content*, již musí být zřejmé, že k načítání webového obsahu bude využívat právě zmíněný SeleniumRanderer, který obdrží v konstruktoru.

4.3.1.2 Načtení seznamu instancí WebAppRule

Třída potřebuje k svému fungování seznam instancí WebAppRule (viz 3.6.1.1). Co jedna instance, to jedna webová aplikace, která může být z obsahu stránky detekována. Získání tohoto seznamu je provedeno pomocí abstraktní metody, jejíž implementace je ponechána na potomkovi třídy.

```
@abstractmethod
def _get_web_app_rules(self) -> Iterable[WebAppRule]:
    pass
```

Listing 4.11 Načítání instancí WebAppRule v třídě HtmlContentParsingMethod ponecháno jako abstraktní metoda

Aby mohl být seznam dynamicky rozšiřován a upravován osobou bez znalosti kódu a konkrétního programovacího jazyka, došlo k rozhodnutí vytvořit třídu *HTMLContentParsingFromFileMethod* (viz její návrh - 3.6.1.5), která dědí ze třídy HtmlContentParsingMethod, a seznam pravidel načítá ze souboru. Jako formát souboru byl zvolen JSON. Třída přijímá souborový parser v konstruktoru, tudíž je připravena na případné rozšíření o jiný formát.

Součástí aplikace bude tedy tento seznam v JSON souboru, který jednak bude moci být samozřejmě udržován v rámci verzování samotného repozitáře, a jednak rozšiřován a udržován v rámci konkrétního nasazení administrátory aplikace.

```
[
  {
    "name": "Grafana",
    "identifier": "<title>Grafana</title>",
    "version": "rel=\"version\" id=\"version\">v(\\d+\\.\\d+\\.\\d+)"
  },
  {
    "name": "GitLab",
    "identifier": "<meta content=\"GitLab\" property=\"og:site_name\">",
    "auth":{
      "method": "username_and_password",
      "user_box_params": [
        {"key": "name", "value": "username"}
      ],
      "pwd_box_params": [
        {"key": "name", "value": "password"}
      ],
      "username": "user123",
      "password": "password123"
    },
    "version": "gon.version=\"(\\d+\\.\\d+\\.\\d+)\";"
  },
```

Listing 4.12 Příklad databáze pravidel ve formátu JSON pro detekci webových aplikací z obsahu stránky

Lze si povšimnout, že v případě, že k získání verze webové aplikace je potřeba autentizace ve formě přihlášení uživatelským jménem a heslem, je součástí souboru kromě lokace vstupních boxů pro tyto údaje (viz 4.3.1.4) také samotné jméno a heslo. Pro implementaci prvního prototypu aplikace se tato volba zdála býti dostačující. V budoucnu lze uvažovat o přesunu těchto citlivých údajů do samostatného souboru.

4.3.1.3 Extrahování informací z webové stránky

Pokus o získání informací o případné webové aplikaci na dané adrese probíhá v metodě *inspect_host* (viz útržek kódu 4.13) následujícím způsobem:

1. Řádek 2-3: Je získán seznam pravidel (instancí WebAppRule) a načtena webová stránka (pro navštívení přímo je zadáno doménové jméno obdržené jako parametr metody).
2. Řádek 6: Pokud bylo navštívení a načtení stránky úspěšné, dojde ke spuštění cyklu. V něm se iteruje přes seznam načtených pravidel.
3. Řádek 7-8: Metoda *match* třídy WebAppRule hledá v obsahu stránky regulární výraz *identifier* (atribut třídy WebAppRule, viz 4.14). Pokud ho najde, je rozhodnuto, že dané doménové jméno náleží konkrétní webové aplikaci.
4. Řádek 9-13: Pokud je k detekci verze vyžadována autentizace (viz 3.6.1.2), tak se použije rozhraní IAuthVisitor (jenž je získáno v konstruktoru třídy) v kombinaci se samotným pravidlem. Tento návrhový vzor je popsán v předchozí kapitole (viz 3.6.1.3) a jeho implementace níže (viz 4.3.1.4). Pokud autentizace nebyla úspěšná, dojde k vypsání chyby a verze tedy nemůže být nalezena.
5. Řádek 14-19: Pokud se element s verzí vyskytuje na jiném URL než je hlavní strana či hlavní strana po autentizaci, je třeba tuto relativní cestu přidat k aktuálnímu URL a danou stránku načíst.
6. Řádek 22: Dále dojde k pokusu o nalezení samotné verze přes regulární výraz s parametrem (viz přiložený kód 4.14).
7. Řádek 24: Návratovou hodnotou je instance WebAppInfo (viz 3.7). Pokud je úspěšně nalezena verze webové aplikace, tato instance ji bude obsahovat.
8. Pokud se žádné z dostupných pravidel neshoduje s webovou aplikací na daném doménovém jménu, vrátí se nulová hodnota.

4.3.1.4 Autentizace

Jak již bylo naznačeno v návrhu autentizace přes návrhový vzor *Visitor pattern* (viz 3.6.1.3), proniknutí přes autentizační bránu webové stránky proběhne za pomocí rozhraní IAuthVisitor (konkrétně jeho implementace AuthExecutor, kterou popisuji níže) a třídy Auth, kterou obsahuje třída WebAppRule jako nepovinný atribut. Pokud je tento atribut nenulový, je jasné, že k získání verze webové aplikace je nutná autentizace. Níže budu popisovat implementaci autentizace pomocí uživatelského jména a hesla. V kapitole o návrhu je zmíněno, proč a jakým způsobem je aplikace připravena na rozšíření o další druh autentizace (viz 3.6.1.4 a 3.8).

Logika provedení přihlášení je implementována ve třídě AuthExecutor, jenž implementuje rozhraní IAuthVisitor. Toto rozhraní je přijímáno v konstruktoru samotné třídy *HtmlContentParsingMethod*.

Selenium umožňuje zadávání textového obsahu do vstupních boxů, proto bylo rozhodnuto jít touto cestou. Je však potřeba mít k dispozici stejný driver, resp. stejnou instanci SeleniumRenderer, která je používána k předchozímu načítání obsahu webové stránky, aby se neztratil kontext aktuálně navštívené stránky. Tato instance je tedy přijímána v konstruktoru AuthExecutor. Proces přihlášení pomocí uživatelského jména a hesla v metodě *visit_user_and_pwd_auth* (viz přiložený kód 4.15) probíhá následujícím způsobem:

1. Řádek 15-17: Pokud se přihlašovací okno nevyskytuje přímo na hlavní straně, je nejprve načtena příslušná strana skládající se z aktuální stránky a relativní cesty.

```
def inspect_host(self, host: str) -> Optional[WebAppInfo]:
    web_app_rules = self._get_web_app_rules()
    page_content = self._get_full_page_content(host)
    if not page_content:
        return None
    for rule in web_app_rules:
        if rule.matches(page_content):
            name = rule.web_app_name
            if rule.auth:
                page_content = rule.auth.accept(self.auth_executor)
                if page_content is None:
                    logger.error("...")
                    return WebAppInfo(name, None)
            if rule.version_path:
                ful_ver_path = host + rule.version_path
                page_content = self._get_full_page_content(ful_ver_path)
                if page_content is None:
                    logger.error("...")
                    return WebAppInfo(name, None)
            ver = None
            if page_content:
                ver = rule.find_version(page_content)
            logger.info("...")
            return WebAppInfo(name, ver)
    logger.info(f"{host} was not matched with any known web application")
    return None
```

Listing 4.13 Extrahování informací o webové aplikaci z obsahu stránky pomocí metody *inspect_host* třídy HtmlContentParsingMethod

2. Řádek 18-19: Vytvoří se řetězec podle kterého se posléze na řádku 20-22 lokalizuje vstupní box pro zadání uživatelského jména
3. Řádek 25-26: Obsah boxu se smaže a vloží se do něj uživatelské jméno
4. Řádek 28-36: To samé proběhne i pro uživatelské heslo
5. Řádek 38: Během toho, co je vybrán vstupní box pro heslo, je zadána klávesa ENTER pro potvrzení.[5]
6. Řádek 39: Proběhne uspání vlákna, což driveru poskytne čas k načtení stránky po přihlášení, jejíž obsah je z metody na řádku 40 navrácen.

4.3.1.5 Reflexe

Srovnání tohoto způsobu zjišťování informací o webových aplikacích s jinými možnými způsoby je věnována část v analytické kapitole (viz 2.4.2). Bylo by avšak vhodné zmínit, že tato metoda bude

[5] Ačkoliv to není případ ani jedné webové aplikace ze seznamu zadavatele (2.1.1.6), tak tato použitá technika nemusí být naprosto spolehlivá, jelikož k potvrzení zadání údajů nemusí stisknutí klávesy ENTER vždy postačit. Při implementaci byla brána v úvahu varianta existence dvou možností potvrzení. Kromě potvrzení pomocí klávesy ENTER by uživatel (osoba spravující seznam s pravidly webových aplikací) mohl zadefinovat lokaci potvrzovacího tlačítka stejně tak jako zadefinuje lokaci vstupního boxu pro jméno a heslo. Došlo k rozhodnutí nejít touto cestou, protože by výsledný soubor s pravidly mohl být příliš nepřehledný.

spolehlivá, pouze pokud se používaný seznam pravidel bude udržovat. Konkrétní lokace elementů v HTML obsahu stránky obsahující názvy a verze se pravděpodobně v průběhu vývoje budou měnit. Autor či správce aplikace je zodpovědný za údržbu tohoto seznamu, aby byla udržena nastavená míra spolehlivosti, se kterou je aplikace schopná detekovat potřebné informace.

V případě, že je pro jednu webovou aplikaci potřeba definovat více takových pravidel (např. jiné verze obsahují požadované informace na jiných místech), lze jednoduše každé pravidlo uvést zvlášť (se stejným názvem pro webovou aplikaci).

4.4 Implementace modulu FullInfoFetcher

Kvalitní návrh třídy *FullInfoFetcher* (viz 3.7) umožnil snadnou realizaci získávání dostupných informací o webových aplikacích a jejich srovnávání. Její kód lze vidět v útržku 4.16. Jako výchozí komparátor - implementace rozhraní *IVersionComparator* - byla zvolena třída *SemanticVersionComparator*. Tato třída je také jedinou implementací tohoto rozhraní, jelikož všechny webové aplikace ze seznamu zadavatele používající verzování používají právě systém sémantického verzování.

4.4.1 SemanticVersionComparator

Pro implementaci této třídy byla použita Python knihovna *packaging*, konkrétně její modul *version*. Tento modul umožňuje snadné porovnávání verzí. [85] Pro ukázku je uveden kus kódu (viz 4.17), na kterém je zachycen způsob určení nejvyšší verze ze zadané kolekce instancí *VersionCycleInfo*. Používá se zde funkce *max*. Lambda funkce, jejíž výstupem je nejvyšší verze dané nadskupiny verzí (cyklu) je použita při samotném porovnání, jelikož právě její návratová hodnota je porovnávána. Využívá se zde třída *Version* knihovny *packaging*, do jejíhož konstruktoru je vložen zmiňovaný řetězec s nejvyšší verzí daného cyklu.

Zde lze opět vidět výhodu Pythonu, díky které se nemuselo přijít s implementací vlastního komparátoru, který by nejprve řetězec s verzí musel rozdělit na jednotlivé číselné části (major, minor atd.) a poté je porovnávat dle správného pořadí. Vše řeší knihovna *packaging*.

```
@staticmethod
def get_ltst_ver(ver_cycles: Iterable[VersionCycleInfo]) -> Optional[str]:
    try:
        ver_cycle_w_latest_ver = (
            max(ver_cycles, key=lambda v: Version(v.latest), default=None)
        )
        return ver_cycle_w_latest_ver.latest
    except Exception as e:
        logger.error("...")
        return None
```

Listing 4.17 Získání nejvyšší verze ve třídě SemanticVersionComparator

4.4.2 EndOfLifeReleaseFetcherMethod a GitHubReleaseFetcherMethod

Atributy těchto tříd (popsané v kapitole o návrhu viz 3.11) postačí k volání HTTP požadavků na API webových stránek *Endoflife.date* a *GitHub.com*. K samotnému volání byla využita jednoduchá knihovna *requests*.

Pro zamezení duplicity kódu byla vytvořena jednoduchá třída *JsonFetcher* s metodou *get_json_from_api*, která knihovnu *requests* používá k získání JSON souboru z URL API zadaného v parametru. Třídy *EndOfLifeReleaseFetcherMethod* a *GitHubReleaseFetcherMethod* pak dědí z této třídy a používají tuto její metodu.

4.5 Implementace API

Pro implementaci HTTP API bylo nutné prostudovat dokumentaci frameworku Flask. Velkou výhodou by bylo automatické vygenerování dokumentace Swagger UI. K tomu bylo zapotřebí použití další knihovny *flask_restx*. V nadcházejícím kusu kódu (4.18) je uvedena implementace endpointu sloužícímu k zahájení skenu pomocí frameworku Flask a knihovny *flask_restx*.

```
@api.route('/scan')
class Scan(Resource):
    @api.doc('start_scan', description="...")
    @api.expect(subnet_model)
    @api.response(400, 'Invalid input. ...')
    @api.response(409, 'Another scan already in progress')
    @api.response(202, 'Scan started')
    def post(self):
        data = request.json
        subnets = data.get('subnets', '')
        if not subnets or not validate_subnets(subnets):
            api.abort(400, 'Invalid or empty subnet input')
        if not scan_lock.acquire(blocking=False):
            api.abort(409, 'Scan already in progress')
        scan_id = str(uuid.uuid4())
        thread = threading.Thread(target=run_scan,
                                  args=(subnets, scan_id,
                                        scan_lock, web_app_radar))
        thread.start()
        return {'message': 'Scan started', 'scan_id': scan_id}, 202
```

Listing 4.18 Implementace API endpointu pro zahájení skenu

Pomocí anotace *api.route* je určen samotný URL endpoint. Ostatní antace slouží k dokumentaci endpointu. Po validaci vstupu přijde na řadu ověření, zda aktuálně neprobíhá jiný sken sítě. K tomu je použit mezivláknový zámek z Python knihovny *threading* (konkrétně instance třídy *Lock*.[6]) Pokud není zámek zrovna uzamčen, je vygenerováno unikátní ID, které je zasláno zpět klientovi. ID by mohlo být vygenerováno automaticky například po přidání záznamu do databáze. Implementovaná varianta se zdála být nejjednodušší, protože na ID z databáze by se bylo potřeba doptávat po několika vrstvách aplikace.

Lze si povšimnout, že pro zahájení skenu dojde k zavolání jakési funkce *run_scan* ve zvláštním vlákně. Tato funkce pouze zaobaluje zahájení samotného skenu pomocí třídy *WebAppRadar* a následného uvolnění zámku.

[6] Zámek je synchronizační primitivum, které v okamžiku uzamčení není vlastněno konkrétním vláknem. Zámek se nachází v jednom ze dvou stavů: „zamčeno“ nebo „odemčeno“. Vytváří se v odemčeném stavu. Metoda *acquire(blocking=False)* okamžitě vrátí „False“, pokud je zámek zamčený. Pokud je otevřený, zamkne se a pokračuje se dál v kódu. [86]

4.6 Nasazení aplikace

Pro snadné nasazení aplikace lze využít *docker-compose* konfiguračního souboru, který může být spuštěn jednoduchým příkazem. [7]

Tento soubor obsahuje dvě definice služeb: samotnou aplikaci a databázi. V případě budoucího přesunu databáze na jiné zařízení se pouze ze souboru odmaže sekce s databází a aplikaci se podají jiné údaje k připojení k databázi.

Kontejner obsahující aplikaci je potřeba spustit bez síťové izolace. Standardně jsou totiž Docker kontejnery zaobaleny do speciální Docker sítě s názvem *bridge*. [88, 89] Ta by mohla způsobit nepožadované chování během skenování sítě, jelikož zde dochází k izolaci a překladu adres. [90] Tomu se dá zamezit volbou přepínače *host*. *Host* odstraní síťovou izolaci mezi kontejnerem a hostitelským zařízením. Kontejner použije přímo síť hostitele. [89]

Celý soubor *docker-compose.yml* je k vidění v elektronické příloze.

4.7 Dokumentace

K dispozici jsou tři, resp. čtyři dokumentační soubory, které postačí k seznámení se s aplikací na administrátorské a uživatelské úrovni. Aplikace je v dokumentaci označená pod názvem „WebAppRadar“.

1. Swagger UI dokumentace detailně dokumentuje API endpointy. Uvádí strukturu a příklady vstupních a výstupních entit. Rovněž také uvádí možné návratové HTTP kódy. Dokumentace je po spuštění aplikace dostupná přímo na jejím URL. Offline verze dokumentace je dostupná v elektronické příloze.

2. Instalační manuál v anglickém jazyce, který slouží k vysvětlení nasazení, konfigurace a správy aplikace. Je zde vysvětlena konfigurace SSH připojení, rozšiřování seznamu podporovaných aplikací či systém logování. Tento manuál je součástí přílohy A.

3. Uživatelský manuál v anglickém jazyce, jak aplikaci používat a k čemu slouží. Čtenář se dozví, jakým způsobem by měl používat API aplikace a jaká data očekávat či neočekávat a proč. Tento manuál je součástí přílohy B.

4. „README“ soubor v anglickém jazyce, sloužící k navigaci a zaštítění předchozích dvou souborů. (Soubor slouží spíše k přehlednosti celkového repozitáře.) Také jsou zde uvedeny příkazy ke spuštění automatických testů a dema aplikace. Tento soubor je součástí repozitáře přiloženého v elektronické příloze.

[7] Docker Compose je nástroj pro definování a spouštění aplikací s více Docker kontejnery. Compose zjednodušuje kontrolu nad seznamem aplikací a usnadňuje správu v jediném srozumitelném konfiguračním souboru YAML. Jediným příkazem pak lze vytvořit a spustit všechny služby z konfiguračního souboru. [87]

4.8 Struktura projektu

```
app
├── src ..... zdrojové kódy implementace
├── tests ..... potřebné soubory k automatickému testování
│   └── network_tests ..... konfigurační soubory ke spuštění testů v izolovaných prostředích
│       ├── env1
│       ├── env2
│       └── env3
├── doc
│   ├── INSTALL_MANUAL.md ..... instalační manuál v anglickém jazyce
│   └── USER_MANUAL.md ..... uživatelský manuál v anglickém jazyce
├── app.py ..... hlavní zdrojový kód aplikace s definicí endpointů
├── init.py ..... zdrojový kód určený k vytvoření všech potřebných objektů
├── demo.py ..... zdrojový kód ke spuštění dema
├── docker-compose.yml ..... hlavní Docker Compose konfigurační soubor
├── docker-compose-demo.yml ..... Docker Compose konfigurační soubor ke spuštění dema
├── Dockerfile ..... definice Docker image aplikace
├── README.md
├── requirements.txt ..... seznam Python knihoven
├── config.ini ..... hlavní konfigurační soubor
└── web-apps.json ..... databáze pravidel pro detekci webových aplikací
```

Listing 4.19 Struktura repozitáře aplikace

4.9 Splnění požadavků

Tato sekce se odkazuje na funkční a nefunkční požadavky zadavatele z kapitoly o návrhu (2.1).

Všechny funkční požadavky byly splněny. Za napůl splněné by se daly označit **F1**, resp. **F6**. Možnost zadefinování vlastních webových aplikací je implementována pouze pro detekci názvu a verze. Aktuální řešení neumožňuje rozšíření či změnu definic pro získávání nejnovějších dostupných verzí. Aplikace je však schopna dané informace získávat pro seznam webových aplikací z požadavku zadavatele (viz 2.1.1.6).

N1 byl dodržen. **N2** je ze subjektivního hlediska splněn. Návrh byl proveden co nejvíce modulárně a při jakémkoliv náznaku možného rozšíření byla snaha o zachování správných objektových principů. **N3**, **N4** a **N5** se taktéž považují za splněné.

N6 se nedá považovat za zcela splněný, jelikož je pro kompletní proces aplikace potřeba vytvoření a konfigurace SSH uživatele na všech zkoumaných zařízeních. V případě, že uživatel vyžaduje podporu pro webovou aplikaci, jejíž verze se vyskytuje za autentizační bránou, může být navíc potřeba vytvoření a zadání speciálních uživatelských údajů. Tento fakt je bohužel daň za vybrané metody a technologie, které byly i tak v kapitole o analýze označeny za nejvhodnější možný způsob. V případě jiného řešení by nedostatky aplikace byly pravděpodobně mnohem větší.

N7 je splněn. SSH uživatel používaný k přístupu na zkoumané zařízení, stejně tak jako samotná aplikace, nevyžaduje administrátorská oprávnění.

```python
class WebAppRule:
    def __init__(self, web_app_name: str,
                 identifier: str,
                 version_string: Optional[str],
                 version_path: Optional[str] = None,
                 auth: Optional[Auth] = None):
        self.web_app_name = web_app_name
        self.identifier = identifier
        self.version_string = version_string
        self.version_path = version_path
        self.auth = auth

        # Compile the regular expression pattern for better performance
        self.id_pattern = (
            WebAppRule._compile_regex_pattern(self.identifier))
        if self.version_string is not None:
            self.ver_pattern = (
                WebAppRule._compile_regex_pattern(self.version_string))

    @staticmethod
    def _compile_regex_pattern(pattern: str) -> re.Pattern[str]:
        try:
            return re.compile(pattern)
        except re.error:
            raise ValueError(f"Invalid regex pattern: {pattern}")

    def matches(self, html_content: str) -> bool:
        return bool(self.id_pattern.search(html_content))

    def find_version(self, html_content: str) -> Optional[str]:
        if self.version_string is None:
            return None
        match = self.ver_pattern.search(html_content)
        if match:
            return match.group(1)
        else:
            return None
```

Listing 4.14 Určení webové aplikace a její verze z obsahu stránky pomocí regulárních výrazů

```python
@staticmethod
def _get_xpath_params(params: Iterable[HTMLElementParam]) -> str:
    return ''.join(f"[@{param.key}='{param.value}']" for param in params)

def _create_absolute_auth_path(self, relative_auth_path: str):
    # both current URL of the driver and relative path
    # might contain '/' -> remove one
    if (self.renderer.driver.current_url.endswith('/')
            and relative_auth_path.startswith('/')):
        return self.renderer.driver.current_url[:-1] + relative_auth_path
    return self.renderer.driver.current_url + relative_auth_path

def visit_user_and_pwd_auth(self, auth: UserAndPwdAuth) -> Optional[str]:
    try:
        if auth.auth_path:
            abs_auth_path = self._create_absolute_auth_path(auth.auth_path)
            self.renderer.get_page_content(abs_auth_path)
        username_input_xpath = \
            f"//input{AuthExecutor._get_xpath_params(auth.user_box_params)}"
        username_input = (
            self.renderer.driver.find_element(By.XPATH,
                                              username_input_xpath))
        if not username_input:
            return None
        username_input.clear()
        username_input.send_keys(auth.username)

        password_input_xpath = \
            f"//input{AuthExecutor._get_xpath_params(auth.pwd_box_params)}"
        password_input = (
            self.renderer.driver.find_element(By.XPATH,
                                              password_input_xpath))
        if not password_input:
            return None
        password_input.clear()
        password_input.send_keys(auth.password)

        password_input.send_keys(Keys.ENTER)
        time.sleep(self.renderer.explicit_waiting)
        return self.renderer.driver.page_source
    except Exception as e:
        logger.info(f"Exception during authentication: {e}")
        return None
```

Listing 4.15 Provedení autentizace ve třídě AuthExecutor

```python
class FullInfoFetcher:
    def __init__(self, release_fetcher: ReleaseFetcher,
                 ver_cmps: dict[str, IVersionComparator],
                 default_ver_cmp: IVersionComparator
                    = SemanticVersionComparator()):
        self.release_fetcher = release_fetcher
        self.ver_cmps = ver_cmps
        self.default_ver_cmp = default_ver_cmp

    def get_full_info(self, basic_info: WebAppInfo) -> FullWebAppInfo:
        cycles = self.release_fetcher.fetch_web_app_cycle_info(basic_info.name)
        if not cycles:
            return FullWebAppInfo(basic_info.name, basic_info.version)
        if basic_info.name not in self.ver_cmps:
            comparison = (
                self.default_ver_cmp
                .get_version_comparison(basic_info.version, cycles))
        else:
            comparison = (self.ver_cmps[basic_info.name]
                          .get_version_comparison(basic_info.version, cycles))
        return FullWebAppInfo(basic_info.name,
                              basic_info.version,
                              comparison.latest_version,
                              comparison.latest_cycle_version,
                              comparison.eol,
                              comparison.eol_date)
```

Listing 4.16 Implementace třídy FullInfoFetcher

Kapitola 5
Testování

Tato kapitola je věnována základnímu rozdělení testů v rámci softwarového inženýrství. Jsou zde uvedeny příklady testů, které ověřují správnost konkrétních částí aplikace. Na testech jsou ukázány mechanismy a technologie, které vedly k nutnému ověření funkčnosti aplikace.

Jednou z výhod důsledného návrhu aplikace je dobrá testovatelnost. Ta umožnila včasné odhalení chyb a otestování okrajových případů. Spolehlivost testování však nemusí být stoprocentní, jelikož testovací sady jsou omezené svým počtem a tím, kdo je vytváří, a jak omezeně či neomezeně ke kódu přistupuje. Komplexnější povaha aplikace (skenování sítě a procházení serverů) si navíc vyžádala pokročilejší testovací mechanismy.

Existují různé druhy testů. Každý z nich přistupuje ke kódu jinak do hloubky a do šířky.

5.1 Technika mock testování

Často lze narazit na problém, při kterém je potřeba otestovat určitou logickou část kódu a některé závislosti nasimulovat. Typickým příkladem je připojení k databázi. V rámci testování často nemusí být vhodné zahajovat samotné připojení do databáze, jelikož se nejedná a prvek kódu, který by bylo potřeba testovat a vyžadovat na tomto objektu závislost. Více se hodí samotnou databázi nasimulovat a daný simulační prvek zastupující databázi pouze instruovat k tomu, aby vracel nějaká předdefinovaná data. Objekt se bude ovšem jako databáze tvářit.

Mock testování umožňuje izolovat a testovat kód bez závislostí a dalších proměnných, které jsou například spojeny se síťovým připojením a proudy dat. Zjednodušeně řečeno, při mock testování se závislé objekty nahrazují mock objekty. Tyto objekty simulují chování skutečných objektů a vykazují přesné vlastnosti těch původních. Hlavním cílem mock testování je soustředit pozornost na testování, aniž by se testování zabývalo závislostmi. [91]

S mock testováním se lze typicky setkat v rámci unit testování. Tato technika bude avšak znázorněna v příkladu integračního testu (viz 5.3.1).

5.2 Unit testy

Jednotkové testování (unit testing) je proces, při kterém se testuje nejmenší funkční jednotka kódu. Je to osvědčený postup při vývoji softwaru, kdy se software píše v malých funkčních jednotkách a pro každou jednotku kódu se napíše jednotkový test. Tímto způsobem se může v případě selhání testu rychle izolovat oblast kódu, ve které je chyba. Jednotkové testování zesiluje modulární myšlení a zlepšuje pokrytí a kvalitu testů. [92]

Jednotkový test je blok kódu, který ověřuje správnost menšího, izolovaného bloku kódu aplikace, obvykle funkce nebo metody. Jednotkový test je navržen tak, aby ověřil, že blok kódu běží podle očekávání, v souladu s teoretickou logikou vývojáře, který za ním stojí. Jednotkový test je schopen komunikovat s blokem kódu pouze prostřednictvím vstupů a zachyceného výstupu. [92]

Pokud blok kódu vyžaduje ke svému běhu další části systému, nelze použít jednotkový test. Jednotkový test musí běžet izolovaně. Pro funkčnost kódu mohou být nutná jiná systémová data, například databáze, objekty nebo síťová komunikace. V takovém případě by se měla data nasimulovat (viz mock testování). [92]

5.2.1 Příklad z aplikace

Příkladem unit testu v rámci implementované aplikace může být třída *TestApache2VhostsCmds*. Třídy, které dědí z třídy *TestCase* Python knihovny *unittest*, mohou být snadno použity pro automatické testování. Jejich metody mohou obsahovat ověřovací „assert“ funkce. Při zavolání určitých command line příkazů jsou všechny unit testy spuštěny a vyhodnoceny. Uživatel může vidět jednoduché statistiky.

Podle názvu lze usoudit, že třída TestApache2VhostsCmds bude testovat třídu *Apache2VhostsCmds*. Vzhledem k tomu, že metody *is_web_server_running* a *get_content_from_server* pouze vrací staticky zadefinovaný řetězec, připadá v úvahu testovat pouze metodu *get_all_vhosts_from_content*, která ze zadaných konfiguračních souborů ve formě jednoho řetězce vyextrahuje všechna hostovaná doménová jména, v tomto případě pro webový server Apache. Na následujících úryvcích kódu bude ukázán samotný proces testování.

Nejprve se vytvoří instance testovaného objektu, která je využívána ve všech nadcházejících testech (viz 5.1).

```
class TestApache2VhostsCmds(unittest.TestCase):
    def setUp(self):
        self.cmds = Apache2VhostsCmds()
```

Listing 5.1 Příprava testu pro Apache2VhostsCmds

V nadcházejícím kusu kódu (5.2) je uveden test pro jedenu konkrétní konfiguraci. Ta je na řádcích 2-18 zadefinována jako testovací vstup. Na řádku 19 je vytvořen očekávaný výstup - dvě doménová jména. Na řádku 20 dojde k zavolání testované metody s daným testovacím vstupem. Na řádku 21 je proveden test, zda kolekce navrácená z metody odpovídá očekávanému výsledku. Porovnání očekává jejich rovnost, co se do počtu a výskytu jednotlivých prvků týče.

```
def test_example1(self):
    content = """
    # Ensure that Apache listens on port 80
    Listen 80
    <VirtualHost *:80>
        DocumentRoot "/www/example1"
        ServerName www.example.com

        # Other directives here
    </VirtualHost>

    <VirtualHost *:80>
        DocumentRoot "/www/example2"
        ServerName www.example.org

        # Other directives here
    </VirtualHost>
    """
    expected_vhosts = {'www.example.com', 'www.example.org'}
    result_vhosts = self.cmds.get_all_vhosts_from_content(content)
    self.assertSetEqual(expected_vhosts, result_vhosts)
```

Listing 5.2 Test metody get_all_vhosts_from_content třídy Apache2VhostsCmds

5.3 Integrační testy

Integrační testování je typ testování softwaru, při kterém jsou postupně integrovány součásti softwaru a následně testovány jako jednotná skupina. Obvykle tyto komponenty fungují dobře již samostatně, ale při integraci s jinými komponentami může dojít k neočekávaným výsledkům. Při integračním testování chtějí testeři najít chyby, které se objeví v důsledku konfliktů kódu mezi softwarovými moduly, když jsou vzájemně integrovány. Obvykle probíhá po jednotkovém testování a před systémovým testováním. [93]

5.3.1 Příklad z aplikace

V testu *TestHTMLContentParsingFromFileMethod* se testuje integrace samotné třídy *HTMLContentParsingFromFileMethod* (viz 3.6.1.5) s třídou *JsonWebAppRulesDeserializer*, která se stará o deserializaci instancí WebAppRules z JSON souboru. Aktuálně je toto výchozí způsob načítání pravidel pro detekci webových aplikací, proto bylo vhodné ho důkladně otestovat.

Cílem testování je načtení JSON souboru s pravidly a využití těchto pravidel (instancí WebAppRule) k detekci webové aplikace na daném doménovém jménu.

Za normálních okolností třída navštíví a načte dané doménové jméno pomocí instance *SeleniumRenderer*, případně proběhne také autentizace díky *IAuthVisitor*. Účelem testování nemá být čtení reálných webových stránek, ale pouze ověření, zda je třída schopná správně z určitého HTML obsahu stránky zjistit správné informace (název a verzi). Pro tyto třídy *SeleniumRenderer* a *IAuthVisitor* bude tedy použit mock. Při inicializaci testu se předdefinovaný seznam doménových jmen napáruje na HTML obsahy stránek pevně uložených v repozitáři v jednotlivých souborech. Metoda *_get_full_page_content* třídy *HTMLContentParsingFromFileMethod* se „namockuje“, tedy její chování se upraví tak, že vždy když je zavolána s daným doménovým jménem, vrátí příslušný obsah HTML souboru.

Testovanou metodou bude *inspect_host*, která pro zadané doménové jméno vrátí instanci *WebAppInfo* či nulovou hodnotu, pokud doménové jméno nedokáže identifikovat.

```
@patch.object(HTMLContentParsingFromFileMethod, '_get_full_page_content')
def test_inspect_host(self, mock_get_full_page_content):
    json_deserializer = JsonWebAppRulesDeserializer()
    method = (
        HTMLContentParsingFromFileMethod(MagicMock(spec=SeleniumRenderer),
                                         MagicMock(spec=IAuthVisitor),
                                         os.path.join(os.path.dirname(__file__),
                                                      "assets",
                                                      "web-apps.json"),
                                         json_deserializer)
    )

    # Use the preloaded HTML content for the mock side effect
    mock_get_full_page_content.side_effect = \
        lambda host: self.html_contents.get(host)

    result_jira = method.inspect_host('jira.example.org')
    result_bareos = method.inspect_host('bareos.example.org')
    result_unknown_webapp = method.inspect_host('unknown.example.org')

    self.assertEqual(WebAppInfo("Atlassian Jira", "9.4.18"), result_jira)
    self.assertEqual(WebAppInfo("Bareos", "17.2.4"), result_bareos)
    self.assertEqual(None, result_unknown_webapp)
```

Listing 5.3 Test metody inspect_host třídy HTMLContentParsingFromFileMethod

V útržku kódu 5.3 lze vidět samotnou implementaci testu. Ve skutečnosti je testováno mnohem více doménových jmen, kód byl však pro přehlednost uveden ve zkrácené podobě.

Na začátku testu se vytvoří instance JsonWebAppRulesDeserializer. Ta je použita v testovaném objektu třídy HTMLContentParsingFromFileMethod. Třída přijímá mock objekty pro SeleniumRenderer a IAuthVisitor. V tomto případě se jedná o tzv. „white box“ testování, tedy je známo, že tyto dva objekty se využívají v metodě *__get_full_page_content*, která bude simulována, proto není potřeba vkládat reálné instance těchto tříd. Pokud by testy tvořil někdo, kdo kód nezná, pravděpodobně by musel řešit simulované chování i těchto objektů či doplnit reálné objekty.

Instance dále přijímá cestu k definicím pravidel pro webové aplikace ve formě JSON souboru.

Po inicializaci testovaného objektu se definuje chování „mockované“ metody.
V *self.html_contents* se vyskytuje *dictionary*, jehož klíčem je řetězec ve formě doménového jména a hodnotou příslušný předpřipravený HTML obsah stránky.

Na konci testuje proběhne samotné ověření, zda detekované informace z obsahu stránky odpovídají tomu, co se na stránce skutečně nachází či nenachází.

5.4 Systémové testy

Systémové testování je proces, při kterém tým zajišťující kvalitu softwaru vyhodnocuje, jak spolu jednotlivé součásti aplikace spolupracují v rámci celého integrovaného systému nebo aplikace. Systémové testování ověřuje, zda aplikace plní úkoly tak, jak bylo navrženo. Jedná se o „black

box“ testování, které se zaměřuje spíše na funkčnost aplikace jako celku než na vnitřní fungování systému. [94]

5.4.1 Testovací prostředí

K otestování větších logických částí aplikace, které vyžadují skutečnou podsíť se zařízeními a nasazenými webovými aplikacemi bylo nutné přijít s vytvořením jakéhosi testovacího izolovaného prostředí, které by právě tyto prvky simulovalo.

K tomu posloužilo vytvoření speciální Docker sítě s kontejnery. Každý kontejner s vlastní s vlastní IP adresou simuluje reálné zařízení (server). Kontejnery obsahují korektně nakonfigurované webové servery. Tyto webové servery hostují vybraná doménová jména. Pod každým takovým doménovým jménem je dostupná nějaká webová aplikace. Nejedná se o skutečně nasazené webové aplikace, nýbrž servírování statického obsahu. Webový server je nakonfigurován tak, aby při návštěvě daného doménového jména či nějakých jeho subdomén vrátil obsah, jako by se jednalo o skutečnou webovou aplikaci. Tento způsob umožňuje nasimulovat konkrétní webové aplikace (všechny ze seznamu zadavatele) bez toho, aniž by musely být v testovací síti skutečně nasazeny, což by obnášelo složitou konfiguraci a v některých případech také koupi licence.

Tento způsob navíc přináší výhodu přenositelnosti, tedy síť se zařízeními může být deterministicky vytvořena a spuštěna automaticky dle konfiguračního souboru na kterémkoliv zařízení např. v rámci CI (průběžné integrace).

Repozitář aplikace obsahuje konfigurace pro tři testovací prostředí. Každé z nich simuluje webové servery do určité hloubky. Například první z nich obsahuje jen prázdné webové servery a testuje pouze třídy *NMapOpenPortScanner* a *OpenPortWebServerScanner*. Třetí z nich již simuluje skutečné webové aplikace, jak bylo popsáno v předchozích odstavcích.

5.4.1.1 Definice sítě a kontejnerů

Primárním konfiguračním souborem pro prostředí je *docker-compose.yml*. Ten obsahuje definici sítě a seznam zařízení (kontejnerů). Tento soubor také umožní snadnou inicializaci prostředí a spuštění příslušných testů pomocí jednoduchého příkazu.

V souboru je uvedena definice izolované sítě s daným rozsahem.

```
networks:
  custom_network:
    driver: bridge
    ipam:
      driver: default
      config:
        - subnet: 192.0.0.16/29
```

Listing 5.4 Definice izolované sítě v docker-compese.yml testovacího Docker prostředí

Každý kontejner - simulované zařízení s webovým a příp. SSH serverem - má přidělenou statickou IP adresu. Konfigurace samotného kontejneru je definována v příslušném souboru Dockerfile.

```
nginx-1:
  build:
    context: .
    dockerfile: dockerfiles/nginx-1.Dockerfile
  networks:
    custom_network:
      ipv4_address: 192.0.0.18
```

Listing 5.5 Definice zařízení v docker-compese.yml testovacího Docker prostředí simulujícího webový server

5.4.1.2 Definice webových serverů

Jak již bylo naznačeno dříve, samotná definice kontejneru (operační systém, nainstalovaný software, konfigurace a obsažené soubory) jsou součástí souboru Dockerfile. Kompletní soubory jsou součástí elektronické přílohy. Níže je uveden pouze stručný obsah konfigurace.

1. Konfigurační soubor začíná definicí základního image. Ten používá oficiální image pro Apache2 či Nginx založený na operačním systému Debian.

2. Jsou staženy Debian balíčky pro SSH komunikaci, kterou aplikace používá jako mechanismus k vyčítání konfigurací webových serverů a detekci hostovaných doménových jmen.

3. Jsou nastaveny SSH konfigurace nutné k připojení (např. vytvoření uživatele a hesla).

4. Dojde ke korektní konfiguraci webového serveru tak, aby hostoval daná doménová jména.

5. Nakopírování HTML souborů simulující webové aplikace, které bude webový server zasílat klientům.

Během tvorby těchto konfigurací se vyskytly různé problémy. Například docházelo k tomu, že pokud se uživatel připojil do kontejneru pomocí SSH, neměl potřebná oprávnění a správně nastavené PATH proměnné k tomu, aby mohl vykonat příkazy definované v *Apache2VhostsCmds* a *NginxVhostsCmds* (viz 4.2.3). Konfigurace souboru Dockerfile byla tedy obohacena o toto nastavení. Avšak k podobným problémům může dojít také v reálné aplikaci, kdy zkoumané servery nebudou pro SSH uživatele poskytovat dostatečná práva a nastavení. Jedná se bohužel o možný nedostatek, které se snaží podchytit uvedení těchto informací v instalačním manuálu aplikace. Případné chyby v připojení lze také vyčíst ze samotných logů aplikace.

5.4.1.3 Spuštění testů

Účelem vytvoření popisovaného prostředí je otestování aplikace. Součástí prostředí je tedy i kontejner obsahující kód aplikace a spouštějící konkrétní testy, které na rozdíl od předchozích testů mohou využít faktu, že jsou spouštěny v prostředí simulující síť s nasazenými webovými aplikacemi.

Součástí definice zmíněného kontejneru v *docker-compose.yml* je také položka *extra_hosts*, která přidá mapování doménových jmen do konfigurace síťového rozhraní kontejneru (*/etc/hosts*). [95] Toto simuluje reálnou DNS konfiguraci nutnou k přístupu na doménová jména stránek na jiných zařízeních.

```
app:
  build:
    context: ../../../.
    dockerfile: ./Dockerfile
  # run all tests in the specific directory
  entrypoint: ["python3", "-m", "unittest", "discover",
               "tests/network_tests/env3"]
  depends_on:
    - nginx-1
    - apache-1
    ...
  networks:
    custom_network:
      ipv4_address: 192.0.0.22
  extra_hosts:
    - "jira.webappradar-example.io:192.0.0.18"
    - "bareos.webappradar-example.io:192.0.0.21"
    - ...
```

Listing 5.6 Definice kontejneru provádějící testy v izolovaném Docker prostředí

5.4.2 Příklad z aplikace

Popsané testovací prostředí je využito v rámci různých integračních a systémových testů. Následná ukázka testu třídy *WebAppDeterminer* je právě na pomezí integračního a systémového testu. Testuje celý logický celek sloužící k detekci všech informací o webové aplikaci pod daným doménovým jménem (viz modul WebAppDeterminer 3.6).

Pro testování se vytvoří objekty použité ve výchozí inicializaci aplikace. K detekci informací se používá vyčítání obsahu webových stránek, k samotnému načítání obsahu webových stránek je použita instance *SeleniumChromeRenderer* a seznam pravidel webových aplikací je načten z JSON souboru. V *self.det* je uchován testovaný objekt.

```
web_apps_json_path = os.path.join(os.path.dirname(__file__), 'web-apps.json')
renderer = SeleniumChromeRenderer()
det_methods = [HTMLContentParsingFromFileMethod(renderer,
                                                AuthExecutor(renderer),
                                                web_apps_json_path,
                                                JsonWebAppRulesDeserializer())]
self.det = WebAppDeterminer(det_methods)
```

Listing 5.7 Inicializace objektů pro integrační, resp. systémové testování třídy WebAppDeterminer

Na zkráceném útržku kódu (5.8) lze vidět samotný proces testování třídy WebAppDeterminer, resp. její metody *detect_web_app_info*.

```
def test_discover_valid_hosts(self):
    res1 = self.det.detect_web_app_info("http://jira.webappradar-example.io")
    res3 = self.det.detect_web_app_info("bareos.webappradar-example.io")
    res9 = self.det.detect_web_app_info("www.google.com")

    expected1 = WebAppInfo("Atlassian Jira", "9.4.18")
    expected3 = WebAppInfo("Bareos", "17.2.4")
    expected9 = None  # Google not on list of known web apps

    self.assertEqual(res1, expected1)
    self.assertEqual(res3, expected3)
    self.assertEqual(res9, expected9)

def test_discover_invalid_hosts(self):
    res1 = self.det.detect_web_app_info("http://420.webappradar-example.io")
    res2 = self.det.detect_web_app_info("64sd5f6sd4f65d4sf")

    expected1 = None
    expected2 = None

    self.assertEqual(res1, expected1)
    self.assertEqual(res2, expected2)
```

Listing 5.8 Testování třídy WebAppDeterminer v izolovaném Docker prostředí

Kapitola 6

Možná rozšíření

V této kapitole jsou představena možná rozšíření aplikace, která by v případě implementace mohla vést ke zkvalitnění uživatelského zážitku a zvýšení přínosu aplikace.

Následné sekce jsou věnovány jednotlivým možným rozšířením.

6.1 Paralelní načítání obsahu stránek

Získání informací o webových aplikacích probíhá sekvenčně, tedy v cyklu, kdy se nejprve zkoumá jedno doménové jméno a po dokončení průzkumu a extrakci informací se přejde na další. V případě jediné implementované metody průzkumu, třídy HtmlContentParsingMethod, může samotný proces pro velký síťový rozsah s velkým počtem nasazených webových aplikací zabrat dlouhou dobu.

Ke zkrácení času by mohlo dojít, pokud by bylo v několika výpočetních vláknech načítáno a zkoumáno více webových stránek najednou. V konfiguračním souboru by mohl být zadefinován maximální počet těchto vláken. V případě HtmlContentParsingMethod by každému vláknu náležela jedna instance SeleniumRenderer. Tento přístup s více instancemi a vlákny by urychlil běh aplikace, ale zároveň kladl vyšší požadavky na výkon serveru.

6.2 Získání dostupných verzí v konfiguračním souboru

Seznam podporovaných webových aplikací s pravidly pro jejich detekci aktuálně neobsahuje definici pro získání jejich nejnovějších dostupných verzí. Jinými slovy **F1** (2.1.1.1) je splněn, avšak uživatel nemůže definice měnit pomocí žádného konfiguračního souboru stejně tak, jako u detekce názvu a verze. Způsob, jakým aplikace získává tato data o konkrétních webových aplikacích, je zadefinován v kódu.

Třídy EndOfLifeReleaseFetcherMethod a GitHubReleaseFetcherMethod jsou navrženy tak, aby po dosazení správných parametrů mohly být využity i pro jiné webové aplikace. Díky tomuto základu by bylo možné doplnit konfigurační JSON soubor o způsoby získávání nejnovějších dostupných verzí. Konfigurační soubor by mohl obsahovat název metody (např. „endoflife“) a nutné parametry (v případě *Endoflife.date* tedy pouze název aplikace).

6.3 IP adresa součást výsledku

Doposud výsledky skenu poukazují pouze na samotná doménová jména. Výsledná data by mohla být obohacena i o IP adresu. Kromě získání vyšší informativní hodnoty by zde mohl být přínos také v případě, kdy sken sice zjistí, že na dané IP adrese pravděpodobně běží nějaká aplikace, ale z nějakého důvodu už se mu nepodaří dohledat hostovaná doménová jména.

6.4 Podpora dalších operačních systémů

Aplikace aktuálně podporuje detekci webových aplikací nasazených pouze na operačním systému Debian a jeho derivátech. Aplikace by mohla být rozšířena o podporu jiných Linuxových distribucí (CentOS, Fedora, atd.). V rámci aktuálního návrhu aplikace by to pouze znamenalo další implementace rozhraní *IVhostsCmds*, které by obsahovaly definice příkazů pro specifickou distribuci Linuxu.

Kapitola 7
Závěr

Tato bakalářská práce se zaobírá problematikou detekce a monitoringu webových aplikací nasazených v interních síťových prostředích s operačním systémem Debian.

Cílem práce bylo přijít s řešením umožňující automatickou detekci webových aplikací a jejich srovnání s dostupnými verzemi, na základě kterých lze získat podrobný souhrn informací o aktuálním stavu zkoumané sítě.

Práce byla rozdělena na kapitoly, které by se daly sloučit do dvou základních sekcí - teoretické a praktické.

V teoretické sekci byla analyzována problematika údržby webových aplikací. Dále byly představeny důležité základní pojmy a technologie nutné k porozumění nadcházející analytické kapitoly. V té byly zkoumány existující nástroje, které umožňují sken síťového rozsahu a automatickou detekci webových aplikací. Byly shromážděny požadavky zadavatele, jímž je firma Quanti s. r. o., na vytvoření nástroje pro automatický průzkum síťového rozsahu, detekci webových aplikací a monitorování jejich aktualizací. Dále byla provedena analýza možných postupů pro návrh vlastního nástroje.

V praktické sekci byl na základě předchozích poznatků vytvořen návrh aplikace splňující zmíněné požadavky. Při návrhu bylo přihlédnuto k tomu, aby byla aplikace rozšiřitelná a navržená dle správných principů objektového programování. Díky důslednému návrhu a implementaci došlo ke splnění drtivé většiny požadavků. Aplikace je schopna prozkoumat zadaný síťový rozsah a detekovat v něm běžící webové aplikace nasazené na zařízeních právě v tomto rozsahu. Pro objevené webové aplikace je nástroj schopen detekovat jejich aktuální verzi, pokud je konkrétní webová aplikace součástí podporovaného seznamu. Uživatel je schopen seznam rozšířit a zadefinovat tak další pravidla, použitá při další detekci webových aplikací. Nástroj je dále schopen srovnávat aktuální verzi s dostupnými verzemi a informovat tak uživatele o vhodné aktualizaci. Uživatel s aplikací komunikuje pomocí API, které vystavuje data v obecné formě a uživatel tak sám rozhoduje o dalším zpracování informací. V kapitole zabývající se implementací je možné nalézt konkrétní principy a technologie využívané pro realizaci návrhu. Nasazení aplikace do Docker kontejneru umožňuje přenositelnost mezi různými prostředími. Aplikace byla řádně otestovaná v izolovaném prostředí, které nasimulovalo síťový prostor s běžícími webovými aplikacemi. Testovací prostředí bylo navrženo a implementováno tak, aby nevyžadovalo existenci skutečných zařízení a testy byly přenositelné, případně zautomatizované v rámci pipeline. Celá aplikace byla zdokumentována tak, aby ji uživatel mohl snadno nastavit a používat.

Aplikace může být velkým přínosem pro administrátory rozsáhlých sítí, ve kterých není snadné sledovat, jaké konkrétní webové aplikace jsou na nich nasazeny. V neposlední řadě může velmi snadno a rychle administrátora upozornit na nutnou aktualizaci, která zamezí případnému bezpečnostnímu riziku.

Příloha A

Na následujících stranách je přiložena instalační příručka implementované aplikace v anglickém jazyce.

Installation manual

This is a guide for configuring and running the WebAppRadar application. The application runs in isolated Docker environment along with MongoDB database. Application provides HTTP API endpoints for integrating with users.

Prerequisites

Mandatory requirements of the application:

- internet connection
- Docker installed
- all servers you want to scan have to have SSH connection enabled (see SSH section)
- the SSH user that is to be used during SSH connection does not require sudo privilege (it only needs READ permissions to */etc/nginx* and */etc/apache2* configuration files and permission to *systemctl/service status* command)

Configuration

SSH

In order the app to be able to remotely browse servers (to read virtual host configuration), the SSH authentication has to be configured correctly. The app tries to connect to all found devices in given network by SSH. If the server does not accept SSH connection, no info will be detected from the server.

To configure the correct connection on the application side, you have to fill in correct information in the `config.ini` file.

There are two **authentication options** supported:

Private key

Set `private_key` as SSH method and fill in correct private key cipher.

The path to the private key **should not be changed** unless not running the app using docker-compose (standard way), because the private key is rather mounted to the Docker container. That means the file path defined in the `config.ini` should correspond to the container's key path defined in the docker-compose file. In other words, if you don't change it there, you should not change it in the `config.ini`.

The mentioned private key will be used in all connections to all discovered servers.

```
[SSH]
username = user123

method = private_key
path_to_private_key_file = /app/key
private_key_cipher = RSA
```

Password

Set `password` as SSH auth method and fill in correct password which will be used in all connections to all discovered servers.

```
[SSH]
username = user123

method = password
password = pwd123
```

In both scenarios, you have to provide correct username to be used to log in to the server.

If any of the necessary value is not defined correctly, the app will log the error message and terminate at the start-up.

Docker

The application runs in isolated Docker environment. Before running the application, you can set following configuration:

Change API port

To change the port that is used for HTTP API connections on the local machine, please add following lines to `docker-compose.yml`:

```
command: ["-p", "<port_number>"]
```

so the complete file can look something like this:

```
services:
  mongodb:
    image: mongo:latest
    ports:
      - "27017:27017"

  app:
    build:
      context: .
      dockerfile: Dockerfile
    network_mode: host
    environment:
      - MONGO_URI=mongodb://localhost:27017/
    volumes:
      - ${PWD}/config.ini:/app/config.ini
      - /home/john/.ssh/id_rsa:/app/key
    depends_on:
      - mongodb
    command: ["-p", "8080"]
```

Change SSH private key path

If you chose to use private key method ass SSH authentication, you have to adjust to correct file path to the key on the local machine in `docker-compose.yml`. Set the correct path instead of `/home/john/.ssh/id_rsa`.

Extending the list of supported web applications

You can change the list (`web-apps.json`) the app is using for determining the current name and version of the web application.

In order to determine these pieces of information, the app visits all the found hostnames via headless HTTP browser. It reads the whole page content and looks for specific HTML element parts. If the element part is found using regular expressions, the app matches the given hostname with the name of the application.

Similar principle is used for searching the version of the web app. Regular expression with parameter is used for searching the current version of the web app from the HTML page content.

Here is an example for `Grafana` web application defined in `web-apps.json`:

```
{
    "name": "Grafana",
    "identifier": "<title>Grafana</title>",
    "version": "a href=\"\\S+\" target=\"_blank\" rel=\"noopener noreferrer\" id=\"version\">v(\\d+\\.\\d+\\.\\d+)"
}
```

During web app determination phase, WebAppRadar visits a hostname using headless browser, renders client side JavaScript and gets the whole content page. Then it looks for all `identifier` values and tries to match them with the page content. If `<title>Grafana</title>` is found on the page, now we know *Grafana* is running under the hostname.

WebAppRadar then tries to extract a version from the page by using `version` as parametrized regex. If the parameter is matched, now we also found the current version of Grafana.

Sometimes versions are not present on the main page and they are hidden behind login. User can provide `username` and `password` that will be used to get behind the auth wall. However, the WebAppRadar needs to know where the username and password input boxes are located on the page in order to successfully submit the values.

```
{
  "name": "GitLab",
  "identifier": "<meta content=\"GitLab\" property=\"og:site_name\">",
  "auth": {
    "method": "username_and_password",
    "user_box_params": [
      {
        "key": "name",
        "value": "username"
      }
    ],
    "pwd_box_params": [
      {
        "key": "name",
        "value": "password"
      }
    ],
    "username": "user123",
    "password": "password123"
  },
  "version": "gon.version=\"(\\d+\\.\\d+\\.\\d+)\";"
}
```

Here we can see that apart from `identifier` and `version` values, WebAppRadar needs to know the user HTML input box element has HTML attribute `name` with value `username` and the password input box has `name` attribute with `password` value. The `username` and `password` are then filled in to those input boxes and submitted using ENTER. **Only after successful authentication a version is to be read** unlike the case where the authentication part is not specified and the version is read directly from the main page.

Identifier is always being matched from the main page.

Regardless of whether the authentication method is used or not, sometimes the version does not appear immediately on the page.

You can define a relative path to the hostname that is to be used for searching the `version`.

```
{
    "name": "Prometheus",
    "identifier": "<title>Prometheus</title>",
    "version_path": "/status",
    "version": "<th scope=\"row\">Version</th>\\s*<td>(\\d+\\.\\d+\\.\\d+)</td>"
}
```

Web application information discovery process if the web app is `Prometheus`:

1. Hostname *example.com* main page is completely loaded using headless browser.
2. `<title>Prometheus</title>` is found on the page.
3. `example.com/status` is loaded.
4. `<th scope=\"row\">Version</th>\\s*<td>(\\d+\\.\\d+\\.\\d+)</td>` is found on the page and the version is extracted using the regex parameter.

Another example that combines version path with authentication (also on separate path):

```
{
    "name": "Keycloak",
    "identifier": "<title>Welcome to Keycloak</title>",
    "auth":{
      "auth_path": "/admin/master/console/",
      "method": "username_and_password",
      "user_box_params": [
        {"key": "id", "value": "username"}
      ],
      "pwd_box_params": [
        {"key": "id", "value": "password"}
      ],
      "username": "user123",
      "password": "password123"
    },
    "version_path": "/status",
    "version": "<div class=\"version\">(\\d+\\.\\d+\\.\\d+)</div>"
  }
```

Web application information discovery process if the web app is `Keycloak`:

1. Hostname *example2.com* main page is completely loaded using headless browser.
2. `<title>Welcome to Keycloak</title>` is found on the page.
3. `example.com/admin/master/console/` is loaded.
4. Input box with HTML attribute `id=username` is found
5. Input box with HTML attribute `id=password` is found
6. `user123` and `password` are filled in those input boxes and submitted. The page after successful authentication is loaded.
7. `/status` is appended to current URL and the page is loaded.
8. `<div class=\"version\">(\\d+\\.\\d+\\.\\d+)</div>` is found on the page and the version is extracted using the regex parameter.

If any step fails, only the previously found facts are returned.

You can adjust the `web-apps.json` or add other web app rules. Restart the WebAppRadar app to propagate the changes.

NOTE: There is currently no support for defining additional web app releases. WebAppRadar currently only compares the current web app info with latest releases of predefined list of web apps, that cannot be changed in a form of configuration file.

Running the app

After successful configuration you can run the app using `docker compose up --build` or `docker compose up --build -d` in detached mode.

Logging

The app uses logging mechanism that implicitly logs all DEBUG, INFO, WARNING and ERROR logs to `app.log` inside the Docker container and INFO, WARNING and ERROR logs to the console. This can be adjusted in `app.py`.

Příloha B

Na následujících stranách je přiložena uživatelská příručka implementované aplikace v anglickém jazyce.

User manual

WebAppRadar provides HTTP API for user interaction.

You can access the Swagger UI documentation with examples on `<url>:<port>` of WebAppRadar (see installation manual for configuring the port) using your web browser.

Running the scan

WebAppRadar tries to discover all web applications running in network range given by a user as input. The scan compares the found information with latest web app releases.

To run the scan you should use POST method with `/scan` endpoint and provide the subnets or IP addresses (or combination of both) that will be scanned in form of request body:

```
{
  "subnets": "192.168.0.0/24,192.168.68.68"
}
```

Example: Sending POST request to `<web_app_url>/scan` with the body above will trigger scan of whole `192.168.0.0/24` subnet and `192.168.68.68` IP address. Only the web applications deployed and running on these addresses will be included in the result.

The response will include a unique ID assigned to the scan which can be later used for getting the scan result.

IMPORTANT NOTE: **Only one scan be running at a time.** If another scan in the process, a response with 409 status code is returned.

Getting the results

You can list all finished scans using GET request on `/result` URL.

These results will NOT include the main information about the web application, but rather just the complete list of scan IDs, times of scan competition and the scan statuses. Example:

```
[
  {
    "id": "471edecd-6f4a-4bf1-bc50-7aeb1a6af79a",
    "completed_at": "2024-04-13T19:40:36.323496",
    "status": "success"
  },
  {
    "id": "5432daad-6aaa-4bc4-1234-abc8646af666",
    "completed_at": "2024-03-25T19:35:02.123456",
    "status": "fail"
  }
]
```

To see the cause of `status: fail` you have to read the app logs.

To list the complete result of a specific scan, user can use GET request on `/result/<id>` URL where id is the ID of the scan. Example of the response:

```
{
  "id": "35c92f44-c4a7-47f1-9a19-fc2f9d81b5ba",
  "completed_at": "2024-04-15T12:13:42.434006",
  "status": "success",
  "subnets": "192.168.0.0/24,192.168.68.68",
  "web_apps": [
    {
      "hostname": "jira.example.org",
      "name": "Jira",
      "version": "5.1.1",
      "latest_version": "7.2.3",
      "latest_cycle_version": "5.1.9",
      "eol": false,
      "eol_date": "2024-04-15"
    },
    {
      "hostname": "docu.example.org",
      "name": "Confluence",
      "version": "8.2.5",
      "latest_version": "8.2.5",
      "latest_cycle_version": "8.2.5",
      "eol": null,
      "eol_date": null
    },
    {
      "hostname": "rail-page.io",
      "name": "TestRail",
      "version": "6.0.0",
      "latest_version": "7.0.0",
      "latest_cycle_version": null,
      "eol": null,
      "eol_date": null
    },
    {
      "hostname": "git.example.org",
      "name": "GitLab",
      "version": "14.5.19",
      "latest_version": null,
      "latest_cycle_version": null,
      "eol": null,
      "eol_date": null
    },
    {
      "hostname": "nms.example.com",
      "name": "Zabbix",
      "version": null,
      "latest_version": null,
      "latest_cycle_version": null,
      "eol": null,
      "eol_date": null
    },
    {
      "hostname": "super-web-page.example.org",
      "name": null,
      "version": null,
      "latest_version": null,
      "latest_cycle_version": null,
      "eol": null,
      "eol_date": null
    }
  ]
}
```


- `hostname` : Hostname of the scanned device

- `name` : Name of the web application (nullable)
- `version` : Current version of the web application (nullable)
- `latest_version` : Latest available version of the web application (nullable)
- `latest_cycle version` : Latest version of the web application regarding the current cycle version (MAJOR.MINOR) (nullable)
- `eol` : Flag indicating whether the current version reached EOL support (nullable)
- `eol_date` : End of Life date of the current version (nullable)

For more information please open the Swagger UI documentation running on `<url>:<port>` of the WebAppRadar app.

Bibliografie

1. EDGE ACADEMY. *What is a Web Application? | StackPath* [online]. [B.r.]. [cit. 2024-03-16]. Dostupné z: https://www.stackpath.com/edge-academy/what-is-a-web-application/.
2. *An overview of HTTP - HTTP | MDN* [online]. 2023. [cit. 2024-03-06]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview.
3. *ISO - 35.100 - Open systems interconnection (OSI)* [online]. [B.r.]. [cit. 2024-03-06]. Dostupné z: https://www.iso.org/ics/35.100/x/.
4. *HTML Standard* [online]. 2024. [cit. 2024-03-11]. Dostupné z: https://html.spec.whatwg.org/.
5. MDN CONTRIBUTORS. *Learn to style HTML using CSS - Learn web development | MDN — developer.mozilla.org* [online]. 2024. [cit. 2024-03-11]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/CSS.
6. MDN CONTRIBUTORS. *JavaScript | MDN — developer.mozilla.org* [online]. 2024. [cit. 2024-03-11]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript.
7. CHRISTOPHER, Esther. *How Does JavaScript Work Behind the Scenes? JS Engine and Runtime Explained — freecodecamp.org* [online]. [B.r.]. [cit. 2024-03-11]. Dostupné z: https://www.freecodecamp.org/news/how-javascript-works-behind-the-scenes/.
8. OSMANI, Addy; MILLER, Jason. *Rendering on the Web | Articles | web.dev — web.dev* [online]. 2019. [cit. 2024-03-11]. Dostupné z: https://web.dev/articles/rendering-on-the-web.
9. *What is web application security? | Web security | Cloudflare* [online]. [B.r.]. [cit. 2024-03-16]. Dostupné z: https://www.cloudflare.com/learning/security/what-is-web-application-security/.
10. PRESTON-WERNER, Tom. *Semantic Versioning 2.0.0 | Semantic Versioning* [online]. [B.r.]. [cit. 2024-03-11]. Dostupné z: https://semver.org/.
11. FITZPATRICK, Jason. *Here's Why Self-Hosting a Server Is Worth the Effort — howtogeek.com* [online]. 2022. [cit. 2024-03-11]. Dostupné z: https://www.howtogeek.com/846979/heres-why-self-hosting-a-server-is-worth-the-effort/.
12. DEVINE, Richard. *How I fell into the self-hosting rabbit hole in 2021 — windowscentral.com* [online]. 2021. [cit. 2024-03-11]. Dostupné z: https://www.windowscentral.com/self-hosting-2021.
13. *Self-Managed or Managed Hosting: Which One is Right for You?* [online]. [B.r.]. [cit. 2024-04-09]. Dostupné z: https://runcloud.io/blog/self-managed-or-managed-hosting-which-one-is-right-for-you.

14. WU, Hao; DANG, Xianglei; WANG, Lidong; HE, Longtao. Information fusion-based method for distributed domain name system cache poisoning attack detection and identification. *IET Information Security*. 2016, roč. 10, č. 1, s. 37–44. ISSN 1751-8717. Dostupné z DOI: 10.1049/iet-ifs.2014.0386.
15. *Co je SSL /TLS: Podrobný průvodce – SSL.com* [online]. 2023. [cit. 2024-04-09]. Dostupné z: https://www.ssl.com/cs/article/what-is-ssl-tls-an-in-depth-guide/.
16. BERNSEN, Rebecca. *Why to self-host your software? Should you go for on-premises or cloud?* [online]. 2021. [cit. 2024-05-14]. Dostupné z: https://www.openproject.org/blog/why-self-hosting-software/.
17. *Web Application Maintenance — From Bugs to Scaling* [online]. [B.r.]. [cit. 2024-04-09]. Dostupné z: https://rubygarage.org/blog/web-application-maintenance.
18. COUGIAS, Dorian; HEIBERGER, E. L.; KOOP, Karsten. *The Backup Book: Disaster Recovery from Desktop to Data Center*. Chapter 1: What's a Disaster Without a Recovery? Network Frontiers, 2003. ISBN 0-9729039-0-9.
19. NELSON, Steven. *Introduction to Backup and Recovery*. Apress, 2011. ISBN 978-1-4302-2663-5. Retrieved 8 May 2018.
20. VANIEA, Kami; RASHIDI, Yasmeen. Tales of Software Updates: The process of updating software. In: *Proceedings of the Conference on Human Factors in Computing Systems*. San Jose, CA, USA: University of Waterloo; Indiana University Bloomington, 2016. Dostupné z DOI: 10.1145/2858036.2858303.
21. *The risks of not updating software* [online]. [B.r.]. [cit. 2024-04-09]. Dostupné z: https://www.adremsoft.com/blog/view/blog/17317071759651/the-risks-of-not-updating-software.
22. *Why is Monitoring Your Application Important in 2024?* [online]. [B.r.]. [cit. 2024-04-09]. Dostupné z: https://www.netadmintools.com/why-is-monitoring-your-application-important/.
23. CONTRIBUTORS, MDN. *What is a web server? - Learn web development | MDN — developer.mozilla.org* [online]. 2023. [cit. 2024-03-15]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Web_mechanics/What_is_a_web_server.
24. *What is a reverse proxy? | Proxy servers explained | Cloudflare* [online]. [B.r.]. [cit. 2024-03-15]. Dostupné z: https://www.cloudflare.com/learning/cdn/glossary/reverse-proxy/.
25. CACHEFLY TEAM. *An In-depth Exploration of Reverse Proxy and Custom Origin Servers* [online]. 2023. [cit. 2024-05-14]. Dostupné z: https://www.cachefly.com/news/an-in-depth-exploration-of-reverse-proxy-and-custom-origin-servers/.
26. *Apache Virtual Host documentation - Apache HTTP Server Version 2.4* [online]. 2023. [cit. 2024-03-16]. Dostupné z: https://httpd.apache.org/docs/2.4/vhosts/.
27. *Name-based Virtual Host Support - Apache HTTP Server Version 2.4* [online]. 2023. [cit. 2024-03-16]. Dostupné z: https://httpd.apache.org/docs/2.4/vhosts/name-based.html.
28. KIRVAN, Paul. *What Is a Configuration Management Database (CMDB)? | Definition from TechTarget* [online]. 2024. [cit. 2024-03-11]. Dostupné z: https://www.techtarget.com/searchdatacenter/definition/configuration-management-database.
29. LYON, Gordon "Fyodor". *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure.com LLC, 2009. ISBN 0979958717. Dostupné také z: https://nmap.org/book/. Dostupné online.

30. *Nmap Scripting Engine (NSE) | Nmap Network Scanning — nmap.org* [online]. [B.r.]. [cit. 2024-03-11]. Dostupné z: https://nmap.org/book/man-nse.html.

31. *http-enum NSE script — Nmap Scripting Engine documentation* [online]. [B.r.]. [cit. 2024-04-09]. Dostupné z: https://nmap.org/nsedoc/scripts/http-enum.html.

32. *banner NSE script — Nmap Scripting Engine documentation* [online]. [B.r.]. [cit. 2024-03-11]. Dostupné z: https://nmap.org/nsedoc/scripts/banner.html.

33. *Service and Version Detection | Nmap Network Scanning — nmap.org* [online]. [B.r.]. [cit. 2024-03-11]. Dostupné z: https://nmap.org/book/man-version-detection.html.

34. *Nikto - Overview & Description — github.com* [online]. 2018. [cit. 2024-03-11]. Dostupné z: https://github.com/sullo/nikto/wiki/Overview-&-Description.

35. LANE, Dave. *Creating a Multi-Resolution Favicon Including Transparency with the GIMP* [online]. 2008. [cit. 2024-05-13]. Dostupné z: http://egressive.com/tutorial/creating-a-multi-resolution-favicon-including-transparency-with-the-gimp. Archived from the original on December 25, 2010. Archive URL: https://web.archive.org/web/20101225130900/http://egressive.com/tutorial/creating-a-multi-resolution-favicon-including-transparency-with-the-gimp.

36. BCOLES. *WhatWeb - Overview & Description* [online]. 2020. [cit. 2024-03-11]. Dostupné z: https://github.com/urbanadventurer/WhatWeb/wiki/.

37. *Find out what websites are built with - Wappalyzer — wappalyzer.com* [online]. [B.r.]. [cit. 2024-03-11]. Dostupné z: https://www.wappalyzer.com/.

38. *wappalyzer/src/technologies at master · tunetheweb/wappalyzer — github.com* [online]. [B.r.]. [cit. 2024-03-11]. Dostupné z: https://github.com/tunetheweb/wappalyzer/tree/master/src/technologies.

39. ROGERS, Russ. *Nessus Network Auditing*. O'Reilly Media, Inc., 2008. ISBN 978-1-59749-208-9.

40. OLENICK, Doug. SC 30th Anniversary Awards. *SC Media*. 2019. Dostupné také z: https://www.scmedia.com.

41. TENABLE. *Nessus Vulnerability Scanner: Network Security Solution | Tenable®* [online]. [B.r.]. [cit. 2024-03-02]. Dostupné z: https://www.tenable.com/products/nessus.

42. TENABLE. *Tenable Nessus Essentials Vulnerability Scanner | Tenable®* [online]. [B.r.]. [cit. 2024-03-02]. Dostupné z: https://www.tenable.com/products/nessus/nessus-essentials.

43. TENABLE. *CGI abuses Plugins — tenable.com* [online]. 2024. [cit. 2024-03-03]. Dostupné z: https://www.tenable.com/plugins/nessus/families.

44. THAPELO, Tsaone Swaabow; NAMOSHE, Molaletsa; MATSEBE, Oduetse; MOTSHEGWA, Tshiamo; BOPAPE, Mary-Jane Morongwa. SASSCAL WebSAPI: A Web Scraping Application Programming Interface to Support Access to SASSCAL's Weather Data. *Data Science Journal*. 2021, roč. 20, s. 24. ISSN 1683-1470. Dostupné z DOI: 10.5334/dsj-2021-024.

45. *Host Discovery | Nmap Network Scanning* [online]. [B.r.]. [cit. 2024-03-10]. Dostupné z: https://nmap.org/book/man-host-discovery.html.

46. *Privileged Ports* [online]. 1995. [cit. 2024-04-28]. Dostupné z: https://www.w3.org/Daemon/User/Installation/PrivilegedPorts.html.

47. YLONEN, T.; LONVICK, C. *The Secure Shell (SSH) Protocol Architecture* [IETF Trust]. 2006. Dostupné z DOI: 10.17487/RFC4251. RFC 4251.

48. YLONEN, T.; LONVICK, C. *The Secure Shell (SSH) Authentication Protocol* [IETF Trust]. 2006. Dostupné z DOI: 10.17487/RFC4252. RFC 4252.

49. *What is reverse DNS? | Cloudflare* [online]. [B.r.]. [cit. 2024-03-10]. Dostupné z: https://www.cloudflare.com/learning/dns/glossary/reverse-dns/.
50. *Network Protocols Handbook*. 2. vyd. Javvin Technologies Inc., 2005. ISBN 9780974094526.
51. HERTZOG, Raphaël. *How to create Debian packages with alternative compression methods* [online]. 2010. [cit. 2024-03-10]. Dostupné z: https://raphaelhertzog.com/2010/09/17/how-to-create-debian-packages-with-alternative-compression-methods/.
52. *Courses/MaintainingPackages/Packages/Management - Debian Wiki* [online]. 2010. [cit. 2024-03-10]. Dostupné z: https://wiki.debian.org/Courses/MaintainingPackages/Packages/Management.
53. ENDOFLIFE.DATE. *Home | endoflife.date* [online]. [B.r.]. [cit. 2024-03-11]. Dostupné z: https://endoflife.date/.
54. *REST API endpoints for releases - GitHub Docs* [online]. [B.r.]. [cit. 2024-03-18]. Dostupné z: https://docs.github.com/en/rest/releases/releases?apiVersion=2022-11-28.
55. *Releases · bareos/bareos* [online]. [B.r.]. [cit. 2024-03-18]. Dostupné z: https://github.com/bareos/bareos/releases.
56. *TestRail Server supported versions – TestRail Support Center* [online]. [B.r.]. [cit. 2024-03-18]. Dostupné z: https://support.testrail.com/hc/en-us/articles/14334287387796-TestRail-Server-supported-versions.
57. *Releases · snipe/snipe-it* [online]. [B.r.]. [cit. 2024-03-18]. Dostupné z: https://github.com/snipe/snipe-it/releases.
58. *Previous Releases Downloads | TeamCity On-Premises Documentation* [online]. [B.r.]. [cit. 2024-03-18]. Dostupné z: https://www.jetbrains.com/help/teamcity/previous-releases-downloads.html.
59. *Releases · minio/minio* [online]. [B.r.]. [cit. 2024-03-18]. Dostupné z: https://github.com/minio/minio/releases.
60. SHELDON, Robert. *What is garbage collection (GC) in programming?* [online]. 2022. [cit. 2024-04-16]. Dostupné z: https://www.techtarget.com/searchstorage/definition/garbage-collection.
61. *About Python* [online]. Python Software Foundation, 2012 [cit. 2012-04-24]. Dostupné z: https://www.python.org/about/. Archived from the original on 20 April 2012. Archive URL: https://web.archive.org/web/20120420010049/http://www.python.org/about/.
62. PYTHON SOFTWARE FOUNDATION. *PEP 206 – Python Advanced Library | peps.python.org* [online]. [B.r.]. [cit. 2024-04-16]. Dostupné z: https://peps.python.org/pep-0206/.
63. PYTHON SOFTWARE FOUNDATION. *About Python™ | Python.org* [online]. [B.r.]. [cit. 2024-04-16]. Dostupné z: https://www.python.org/about/.
64. *Flask Foreword* [Archived from the original on 2017-11-17]. 2017. [cit. 2024-04-16]. Dostupné z: https://web.archive.org/web/20171117015927/http://flask.pocoo.org/docs/0.10/foreword.
65. *JSON* [online]. [B.r.]. [cit. 2024-04-26]. Dostupné z: https://www.json.org/json-en.html.
66. DOCKER INC. *Docker overview | Docker Docs* [online]. [B.r.]. [cit. 2024-04-16]. Dostupné z: https://docs.docker.com/get-started/overview/.
67. *What Is MongoDB? | MongoDB* [online]. [B.r.]. [cit. 2024-04-26]. Dostupné z: https://www.mongodb.com/company/what-is-mongodb.

68. OLORUNTOBA, Samuel. *SOLID: The First 5 Principles of Object Oriented Design | DigitalOcean* [online]. 2024. [cit. 2024-04-29]. Dostupné z: https://www.digitalocean.com/community/conceptual-articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design.
69. SMARTBEAR SOFTWARE. *API Documentation & Design Tools for Teams | Swagger* [online]. [B.r.]. [cit. 2024-04-15]. Dostupné z: https://swagger.io/.
70. *Visitor Pattern (represent operation on object structure)* [online]. [B.r.]. [cit. 2024-04-22]. Dostupné z: https://www.gofpattern.com/behavioral/patterns/visitor-pattern.php.
71. CRAVEN, Adam; REHN, Christian. *Principle - Keep It Simple Stupid (KISS)* [online]. [B.r.]. [cit. 2024-04-29]. Dostupné z: https://principles.dev/p/keep-it-simple-stupid-kiss/.
72. MULDROW, Lyn. *What is DRY Development? | DigitalOcean* [online]. 2020. [cit. 2024-04-29]. Dostupné z: https://www.digitalocean.com/community/tutorials/what-is-dry-development.
73. ZANNIER, Carmen; ERDOGMUS, Hakan; LINDSTROM, Lowell. Extreme Programming and Agile Methods – XP/Agile Universe 2004: 4th Conference on Extreme Programming and Agile Methods. In: Berlin: Springer, 2004, s. 121. ISBN 3-540-22839-X.
74. *Iterables — Python Like You Mean It* [online]. [B.r.]. [cit. 2024-04-16]. Dostupné z: https://www.pythonlikeyoumeanit.com/Module2_EssentialsOfPython/Iterables.html.
75. PYTHON SOFTWARE FOUNDATION. *5. Data Structures — Python 3.10.13 documentation* [online]. 2023. [cit. 2024-04-30]. Dostupné z: https://docs.python.org/3.10/tutorial/datastructures.html.
76. *systemd - Debian Wiki* [online]. [B.r.]. [cit. 2024-04-30]. Dostupné z: https://wiki.debian.org/systemd.
77. *service(8): run System V init script - Linux man page* [online]. [B.r.]. [cit. 2024-04-30]. Dostupné z: https://linux.die.net/man/8/service.
78. *grep(1p) - Linux manual page* [online]. [B.r.]. [cit. 2024-04-30]. Dostupné z: https://man7.org/linux/man-pages/man1/grep.1p.html.
79. *Command-line parameters* [online]. [B.r.]. [cit. 2024-04-30]. Dostupné z: https://nginx.org/en/docs/switches.html.
80. *httpd - Official Image | Docker Hub* [online]. [B.r.]. [cit. 2024-04-30]. Dostupné z: https://hub.docker.com/%5C_/httpd.
81. *The Selenium Browser Automation Project* [online]. [B.r.]. [cit. 2024-05-01]. Dostupné z: https://www.selenium.dev/documentation/.
82. *Getting started | Selenium* [online]. [B.r.]. [cit. 2024-05-01]. Dostupné z: https://www.selenium.dev/documentation/webdriver/getting_started/.
83. MOLINA, Diego. *Headless is Going Away!* [online]. 2023. [cit. 2024-05-08]. Dostupné z: https://www.selenium.dev/blog/2023/headless-is-going-away/.
84. *Waiting Strategies | Selenium* [online]. [B.r.]. [cit. 2024-05-01]. Dostupné z: https://www.selenium.dev/documentation/webdriver/waits/.
85. *Version specifiers - Python Packaging User Guide* [online]. [B.r.]. [cit. 2024-05-05]. Dostupné z: https://packaging.python.org/en/latest/specifications/version-specifiers/.
86. *threading — Thread-based parallelism — Python 3.10.13 documentation* [online]. [B.r.]. [cit. 2024-05-05]. Dostupné z: https://docs.python.org/3.10/library/threading.html%5C#lock-objects.

87. *Docker Compose overview | Docker Docs* [online]. [B.r.]. [cit. 2024-05-05]. Dostupné z: https://docs.docker.com/compose/.

88. *Networking overview | Docker Docs* [online]. [B.r.]. [cit. 2024-05-05]. Dostupné z: https://docs.docker.com/network/.

89. *Network drivers overview | Docker Docs* [online]. [B.r.]. [cit. 2024-05-05]. Dostupné z: https://docs.docker.com/network/drivers/.

90. *Bridge network driver | Docker Docs* [online]. [B.r.]. [cit. 2024-05-05]. Dostupné z: https://docs.docker.com/network/drivers/bridge/.

91. *What is Mock Testing? - Radview* [online]. [B.r.]. [cit. 2024-05-09]. Dostupné z: https://www.radview.com/glossary/what-is-mock-testing/.

92. AMAZON WEB SERVICES. *What is Unit Testing? - Unit Testing Explained - AWS* [online]. [B.r.]. [cit. 2024-05-08]. Dostupné z: https://aws.amazon.com/what-is/unit-testing/.

93. KATALON. *What is Integration Testing? Definition, Examples, How-to* [online]. [B.r.]. [cit. 2024-05-08]. Dostupné z: https://katalon.com/resources-center/blog/integration-testing.

94. YASAR, Kinza. *What is system testing? – TechTarget Definition* [online]. 2023. [cit. 2024-05-08]. Dostupné z: https://www.techtarget.com/searchsoftwarequality/definition/system-testing.

95. DOCKER INC. *Services top-level elements | Docker Docs* [online]. [B.r.]. [cit. 2024-05-10]. Dostupné z: https://docs.docker.com/compose/compose-file/05-services/%5C#extra_hosts.

Obsah příloh

- `readme.txt` ..... stručný popis obsahu média
- `src`
  - `app` ..... repozitář se zdrojovými kódy, soubory ke spuštění a dokumentací
  - `swagger-ui.html` ..... offline verze Swagger UI dokumentace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text`
  - `thesis.pdf` ..... text práce ve formátu PDF