VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

DETEKCE BOTNETŮ ZALOŽENÝCH NA DGA

DETECTION OF DGA-BASED BOTNETS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MATEJ KEZNIKL**
AUTHOR

VEDOUCÍ PRÁCE **Ing. RADEK HRANICKÝ, Ph.D.**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

155758

Ústav: Ústav informačních systémů (UIFS)
Student: **Kezniki Matej**
Program: Informační technologie
Název: **Detekce botnetů založených na DGA**
Kategorie: Bezpečnost
Akademický rok: 2023/24

Zadání:

1. Nastudujte problematiku Domain Generation Algorithms (DGA) a existující výzkum v oblasti jejich detekce.
2. Vytvořte datovou sadu domén pro účely tvorby a testování mechanismů pro detekci DGA.
3. Po konzultaci s vedoucím navrhněte nástroj, který bude schopen na základě síťové komunikace detekovat komunikaci s botnetovými sítěmi na bázi DGA.
4. Navržený nástroj implementujte.
5. Na vytvořené datové sadě experimentálně ověřte použitelnost implementovaného řešení.
6. Zhodnoťte dosažené výsledky a navrhněte možná rozšíření.

Literatura:

- Sivaguru, Raaghavi, Chhaya Choudhary, Bin Yu, Vadym Tymchenko, Anderson Nascimento, and Martine De Cock. "An evaluation of DGA classifiers." In *2018 IEEE international conference on Big Data, s.* 5058-5067. IEEE, 2018.
- Drichel, Arthur, Ulrike Meyer, Samuel Schüppen, and Dominik Teubert. "Analyzing the real-world applicability of DGA classifiers." In *Proceedings of the 15th International Conference on Availability, Reliability and Security*, s. 1-11. 2020.
- Drichel, Arthur, Nils Faerber, and Ulrike Meyer. "First step towards EXPLAINable DGA multiclass classification." In *The 16th International Conference on Availability, Reliability and Security*, s. 1-13. 2021.
- Drichel, Arthur, Ulrike Meyer, Samuel Schüppen, and Dominik Teubert. "Making use of NXt to nothing: the effect of class imbalances on DGA detection classifiers." In *Proceedings of the 15th International Conference on Availability, Reliability and Security*, s. 1-9. 2020.
- Setinský Jiří: Detekce škodlivých doménových jmen. *Bakalářská práce.* FIT VUT v Brně. 2021.
- Polišenský Jan: Vyhodnocování rizik internetových domén. *Bakalářská práce.* FIT VUT v Brně. 2022.
- Bučko Filip: Klasifikace doménových jmen generovaných algoritmy DGA. *Bakalářská práce.* FIT VUT v Brně. 2023.

Při obhajobě semestrální části projektu je požadováno:
Body 1 až 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Hranický Radek, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1.11.2023
Termín pro odevzdání: 9.5.2024
Datum schválení: 30.10.2023

Abstrakt

Botnety predstavujú vážne hrozby pre kybernetickú bezpečnosť, pričom sú schopné spôsobiť rozsiahle škody na počítačových sieťach a systémoch. Významným spôsobom, akým botnety skrývajú svoju existenciu a komunikujú so servermi, je používanie algoritmov typu DGA na generovanie domén, ktoré umožňujú dynamické vytváranie doménových mien pre kontrolu botnetu. Práca sa zameriava na návrh a vývoj vysoko efektívneho a distribuovaného detekčného systému na analýzu vzorov komunikácie a správania botnetov v rámci sieťovej prevádzky systému DNS a identifikáciu doménových mien vytvorených na báze DGA. S využitím lexikálnych vlastností doménového mena, charakteristických pre DGA, boli porovnané viaceré techniky strojového učenia, s cieľom určiť klasifikátor, ktorý vykazuje najlepšie výsledky. Klasifikátor LightGBM, dosahujúci ROC-AUC skóre v hodnote až 99,18%, bol následne integrovaný do detekčného systému. Pre zabezpečenie funkčnosti a spoľahlivosti celého systému bolo vykonané jednotkové testovanie jednotlivých blokov a taktiež integračné testovanie s cieľom zaistenia vzájomnej kompatibility jednotlivých blokov. Výsledná implementácia detekčného systému dosahuje vysokú úspešnosť pri binárnej klasifikácii doménových mien vytvorených na báze DGA, čím zabezpečuje jeho pripravenosť na efektívne nasadenie v reálnych pracovných prostrediach.

Abstract

Botnets represent significant cybersecurity threats due to their potential to cause extensive damage to computer networks and systems. One primary method by which botnets conceal their existence and communicate with servers is through the use of Domain Generation Algorithms (DGA), which enable the dynamic creation of domain names for controlling the botnet. This thesis focuses on the design and development of a highly efficient and distributed detection system for analyzing communication patterns and behaviors of botnets within DNS network traffic and identifying domain names created based on DGA. Several machine learning techniques were compared, utilizing lexical features of domain names characteristic of DGA, to determine the classifier exhibiting the best results. The LightGBM classifier, achieving a ROC-AUC score of up to 99.18%, was subsequently integrated into the detection system. Unit testing of individual blocks and integration testing were performed to ensure the functionality and reliability of the entire system and the mutual compatibility of its components. The resulting implementation of the detection system achieves high accuracy in the binary classification of domain names created based on DGA, ensuring its readiness for effective deployment in real-world operational environments.

Kľúčové slová

DGA, botnet, bezpečnosť siete, strojové učenie, klasifikácia

Keywords

DGA, botnet, network security, machine learning, classification

Citácia

KEZNIKL, Matej. *Detekce botnetů založených na DGA*. Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Hranický, Ph.D.

Detekce botnetů založených na DGA

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením Ing. Radka Hranického, Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Matej Keznikl
9. mája 2024

Poďakovanie

Chcel by som sa predovšetkým poďakovať svojmu vedúcemu Ing. Radkovi Hranického Ph.D. za jeho ústretovosť, ochotu a čas, ktorý mi venoval pri vypracovávaní bakalárskej práce. Ďalej by som sa chcel poďakovať združeniu CESNET za poskytnutie dát potrebných pre testovanie. Taktiež by som sa chcel poďakovať mojej priateľke, rodičom, starým rodičom a kamarátom za všetku podporu počas celého štúdia.

Obsah

Zoznam obrázkov

Kapitola 1

Úvod

V dnešnej dobe je internet pre väčšinu ľudskej populácie kľúčovou súčasťou ich každodenného života. Jedným zo základných stavebných kameňov tejto globálnej siete je systém DNS (Domain Name System), ktorý vykonáva preklad ľahko zapamätateľných doménových mien na IP adresy, umožňujúc tak efektívnu komunikáciu medzi počítačmi a inými zariadeniami na internete.

Táto kritická infraštruktúra ale nie je imúnna voči hrozbám, ktoré ohrozujú bezpečnosť a stabilitu internetu. S jeho rýchlym rozvojom a nepretržitým rastom používateľskej základne vznikajú aj nové hrozby a bezpečnostné výzvy. Kybernetickí zločinci neustále hľadajú nové spôsoby, ako využiť túto neustále sa rozvíjajúcu platformu na dosiahnutie svojich nekalých zámerov. Jednou z týchto hrozieb je použitie botnetov založených na algoritme generovania domén (DGA - Domain Generation Algorithm).

DGA je sofistikovaným nástrojom, ktorý umožňuje kybernetickým útočníkom dynamicky vytvárať množstvo doménových mien na základe určitých parametrov. Týmto spôsobom sa útočníci snažia udržať komunikáciu so svojimi infikovanými zariadeniami (tzv. botmi) skrytú pred bezpečnostnými systémami a analýzou prevádzky.

Komunikačné kanály (C&C - Command and Control) týchto botnetov, ktoré využívajú DGA, sú často závislé na predvídavom generovaní domén, čo robí ich identifikáciu ešte náročnejšou. Úspešná detekcia týchto botnetov je však nesmierne dôležitá, pretože môže predstavovať ochranu pred rôznymi formami kybernetických hrozieb, vrátane distribuovaných útokov, šírenia malvéru, krádeže dát a mnohých ďalších.

Presne z toho dôvodu je cieľom tejto práce preskúmať, analyzovať a implementovať systém pre detekciu a obranu proti botnetom využívajúcim DGA. Systém bude navrhnutý na báze strojového učenia s využitím zoznamov blacklist a whitelist na predspracovanie známych malígnych a benígnych domén. Detekcia bude diferencovaná medzi prevádzkou s NXDomain odpoveďami a ostatnou prevádzkou systému DNS, pričom vetva s NXDomain odpoveďami bude spracovávaná prioritne.

Implementácia tohto systému na báze strojového učenia zabezpečí presnú a rýchlu identifikáciu botnetov využívajúcich DGA v rámci sieťového prostredia, čo umožní užívateľom mitigáciu následkov útokov, prípadne i zastavenie pokusu o útok botnetu. Využitie zoznamov blacklist a whitelist zabezpečí redukciu falošne pozitívnych i falošne negatívnych výsledkov detekcie. S cieľom dosiahnutia najlepšieho možného výkonu klasifikátorov bude vykonané porovnanie rôznych typov klasifikátorov strojového učenia, pričom bude vybraný ten, ktorý preukáže najlepšie schopnosti pri detekcii botnetov a taktiež z hľadiska počtu falošne pozitívnych, či negatívnych výsledkov klasifikácie. Prioritizácia prevádzky s NXDo-

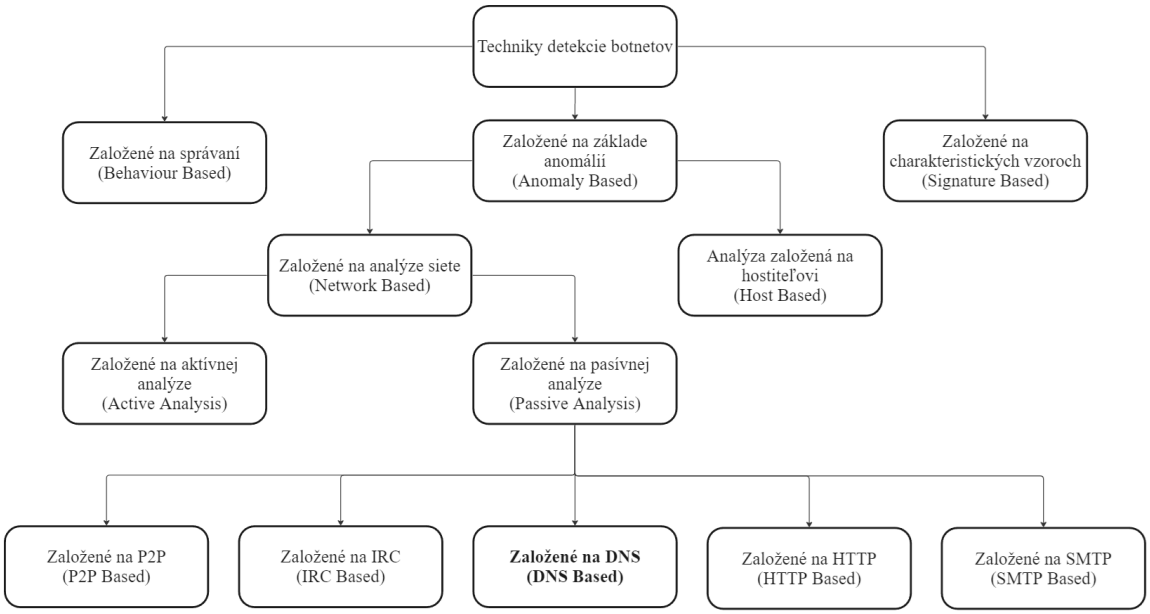
main odpoveďami zaistí dôslednú kontrolu podozrivej prevádzky systému DNS, ktorá by mohla naznačovať pokusy o skrytú komunikáciu s jednotlivými botmi.

Spočiatku budú v kapitole 2 predstavené existujúce relevantné štúdie v oblasti identifikácie botnetov. Následne bude popísaný systém DNS a jeho súvislosť s problematikou v kapitole 3. Popis botnetu a využitie DGA v rámci botnetov budú popísané v kapitole 4. Na tejto teoretickej báze bude potom vypracovaný návrh v kapitole 5 a jeho konečná implementácia bude popísaná v kapitole 6. V kapitole 7 bude vykonané testovanie viacerých klasifikátorov a nakoniec bude vybratý klasifikátor s najlepším výkonom na základe kritérií, ktoré budú bližšie špecifikované v kapitole 7. Bude vykonané testovanie navrhnutého klasifikátoru a taktiež celého detekčného systému v reálnych podmienkach, tzn. v reálnej sieťovej prevádzke v kapitole 8. Na záver budú v kapitole 9 zhodnotené výsledky práce a budú navrhnuté prípadné vylepšenia. V prílohách je popísaný postup inštalácie potrebný pre úspešné spustenie detekčného nástroja a taktiež modulov pre trénovanie klasifikačných modelov.

Kapitola 2

Existujúce relevantné štúdie

Pre detekciu botnetov existuje v súčasnosti veľké množstvo štúdií a techník, pričom ich rozdelenie je možné vidieť na obrázku 2.1.



Obr. 2.1: Existujúce techniky detekcie botnetov

2.1 Techniky založené na správaní

Medzi techniky založené na správaní patrí použitie Honey-netov, teda lákadiel pre útočníkov. Ich použitie spočíva v simulovaní prostredia, ktoré má byť napadnuté, s cieľom zistiť charakteristické správanie napadnutého stroja, binárne súbory bota, IP adresu alebo doménové meno C&C servera. Použitie týchto techník je limitované na sledovanie iba niektorých nekalých činností a nedokážu odhaliť botov, ktorí nepoužívajú mechanizmy šírenia ako je spam a pod. Ich ďalšou limitáciou je možnosť sledovania iba strojov, ktoré sú pripravené ako pasca na útočníka, takže je možné sledovať alebo analyzovať stroj iba v prípade, že bol systém v sieti infikovaný botom [10, 36].

2.2 Techniky založené na charakteristických vzoroch

Ďalšími využívanými technikami sú techniky založené na charakteristických vzoroch, ktoré využívajú databázu vzorov na kontrolu známych botnetov. Tieto techniky detekcie sú veľmi rýchle, dobre fungujú v prípade známych botnetov a majú tendenciu vykazovať veľmi nízky počet falošne pozitívnych výsledkov, avšak majú obmedzenú schopnosť odhaliť neznáme botnety. Botmaster často aktualizuje binárne súbory botov, aby sa vyhol detekcii na základe charakteristických vzorov [68]. Tento druh techník sa napríklad využíva na detekciu DDoS útokov (Distributed Denial of Service) založených na PGA (Packet Packet Generation Algorithms) [72].

2.3 Techniky založené na základe anomálií

Najviac existujúcich štúdií sa zaoberá technikami detekcie na základe anomálií, tzn. hľadajú vzory v správaní hostiteľa alebo siete, ktoré sú neobvyklé alebo odchyľujúce sa od očakávaného normálu. Patrí medzi ne napríklad zvýšená latencia, neobvyklé správanie systému apod. Tieto techniky dokážu odhaliť nové alebo neidentifikované botnety. Existujú dve podskupiny techník detekcie na základe anomálií, ako je možné vidieť na obrázku 2.1.

Analýza založená na hostiteľovi funguje na princípe monitorovania hostiteľa s cieľom zistiť akékoľvek podozrivé aktivity, ako napríklad v systémových volaniach, volaniach socketov, registroch, atď. Nástroj BotSwat [71] slúži na odhaľovanie všeobecných botnetov bez ohľadu na štruktúru botnetu a nasadeného komunikačného protokolu C&C. Kompletne monitoruje vykonávanie ľubovoľného binárneho súboru s cieľom identifikovať potencionálne nakazenie botom [71].

Techniky založené na analýze siete spočívajú v hľadaní vzorov v prevádzke, ktoré sú odlíšiteľné od normálneho toku. Delia sa na dve hlavné kategórie, ako naznačuje zobrazenie v obrázku 2.1.

Aktívna analýza je postavená na princípe vkladania testovacích paketov do sledovanej siete. Jej cieľom je vyvolať reakciu siete, čo však môže vytvárať dodatočné prenosy v sieti [73]. Na tomto princípe funguje napríklad systém BotProbe [24]. Aktívna analýza môže zahŕňať aj metódu známu ako active DNS probing. Táto technika spočíva v aktívnom dotazovaní DNS serverov s cieľom získať informácie o doménach a ich priradených IP adresách. V rámci tejto analýzy sa generujú špecifické DNS dotazy, ktoré môžu odhaliť potenciálne nebezpečné alebo neštandardné správanie v sieti. Pri active DNS probing sú vytvárané úmyselné DNS dotazy na sledované domény, čím sa vyvolávajú odpovede z DNS serverov. Analyzátor môže následne vyhodnotiť tieto odpovede a identifikovať potenciálne anomálie alebo neobvyklé vzory správania, ktoré môžu naznačovať aktivitu botnetov, malware alebo iných bezpečnostných hrozieb. Tento princíp sa využíva napríklad v systéme BotGRABBER [57] a ďalších [58, 79].

Pasívna analýza v kontexte detekcie botnetov sa zameriava na sledovanie, zbieranie a analýzu sieťovej prevádzky bez vkladania testovacích paketov alebo vyvolávania zámernej reakcie v sieti. Cieľom je identifikovať neobvyklé vzory a anomálie, ktoré môžu naznačovať prítomnosť botnetov alebo iných škodlivých aktivít. Pasívne techniky detekcie možno kategorizovať podľa používaných sieťových protokolov botnetov, ako sú P2P, IRC, DNS, HTTP, SMTP, atď [35].

Techniky založené na P2P sa zameriavajú na analýzu komunikácie medzi botmi a ich C&C servermi počas šírenia, čo môže pomôcť k odhaleniu ďalších peerov [4]. Metódy založené na protokole IRC identifikujú špecifické vzory v sieťovej prevádzke [64]. V prípade

HTTP využívajú techniky analýzy konfiguračných súborov, vrátených C&C servermi cez HTTP GET požiadavky, s využitím N-gramov a vzdialenosti medzi bytovými vektormi [75]. Detekcia pomocou SMTP protokolu pracuje na analýze obsahu SMTP paketov a pozorovaní obsahu TCP/IP prevádzky [62].

Pasívna DNS analýza je jedným z kľúčových nástrojov v boji proti botnetom. Systém DNS uchováva a poskytuje informácie o dotazovaných doménových menách, čo využívajú botnety na lokalizáciu svojich C&C serverov, prípadne serverov na aktualizáciu binárnych súborov bota [33]. Analýza týchto dotazov je kľúčová pre odhalenie anomálnych aktivít svedčiacich o prevádzke generovanej botnetom vrátane detekcie použitia techník DGA, prípadne fast-flux na vyhnutie sa detekcii [9, 60]. Analýza dotazov NxDomain, ktoré signalizujú nedostupnosť domény, môžu zodpovedať pokusom botov o dosiahnutie serverov C&C, ktoré boli zrušené alebo migrované [76, 81].

Existuje mnoho postupov, kde je pasívna DNS analýza účinným nástrojom na detekciu botnetov, využívajúc široké spektrum prístupov vrátane štatistických metód (napr. [39]), analýzy grafov (napr. [32]), zhlukovania (napr. [54]), entropie (napr. [31, 81]) a techník založených na umelej inteligencii [5].

Techniky založené na umelej inteligencii sa v tomto kontexte delia do dvoch hlavných kategórií podľa ich prístupu k funkcionalite [82]:

- Učenie pod dohľadom s prístupom založeným na vlastnostiach (Supervised Learning with Feature-Based Approach).
- Učenie bez dohľadu s prístupom bez vlastností (Unsupervised Learning with Featureless Approach).

Pri učení pod dohľadom s prístupom založeným na vlastnostiach sa často na binárnu klasifikáciu využívajú metódy, ako napríklad SVM[1] (napr. [27]), DT[2] (napr. [2]), RF[3] (napr. [80]), XGBoost (napr. [14]), K-NN[4] (napr. [37]) apod.

Pre učenie bez dohľadu s prístupom bez vlastností a s dôrazom na binárnu klasifikáciu sa často využívajú metódy, ako RNN[5] (napr. [66]), CNN[6] (napr. [85]) a ďalšie.

Po dôkladnej analýze existujúcich štúdií v oblasti detekcie botnetov bolo identifikovaných viacero perspektívnych prístupov. Návrh tejto práce kombinuje techniky aktívnej a pasívnej analýzy siete so zameraním na protokol DNS. Okrem toho budú využité aj techniky založené na umelej inteligencii na účely binárnej klasifikácie, konkrétne učenie pod dohľadom s prístupom založeným na vlastnostiach. Tento prístup je zvolený na základe vynikajúcich alebo porovnateľných výsledkov v porovnaní s technikami učenia bez dohľadu s prístupom bez vlastností. Bakalárska práca je motivovaná zámerom vytvoriť plne funkčný detekčný systém zameraný na odhaľovanie botnetov využívajúcich DGA, čo umožní výrazné zlepšenie ochrany súkromia a bezpečnosti online prostredia.

[1] Support Vector Machine
[2] Decision Tree
[3] Random Forrest
[4] K-Nearest Neighbors
[5] Recurrent neural network
[6] Convolutional neural network

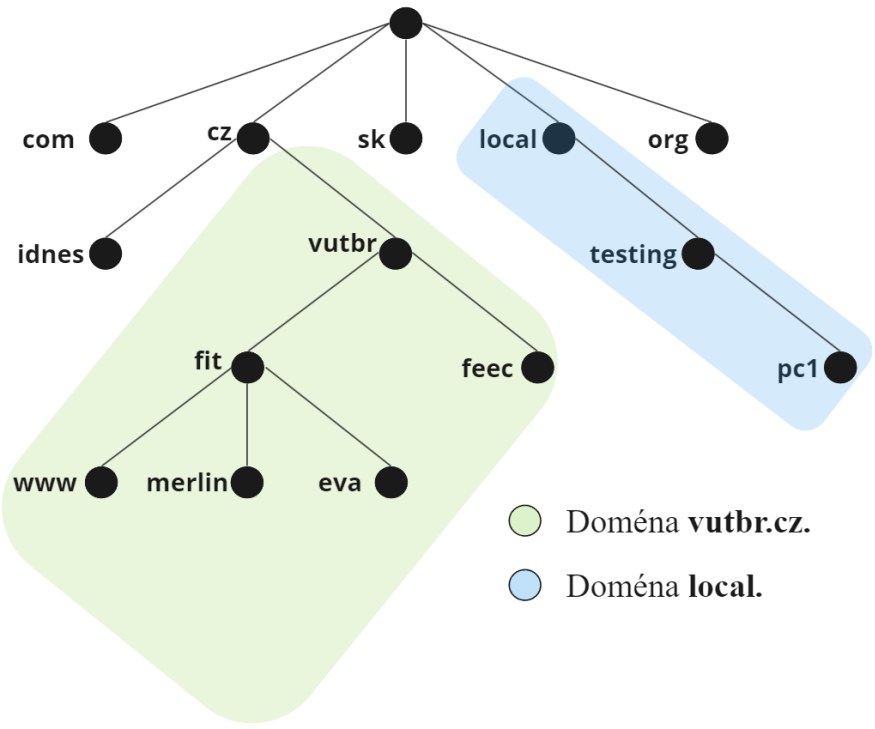
Kapitola 3

Systém DNS

V tejto kapitole bude predstavený systém doménových mien (Domain Name System - DNS [45]) s cieľom lepšie porozumieť, akým spôsobom útočníci zneužívajú DNS na nekalé účely. Základnou úlohou DNS je mapovanie doménových adries na IP adresy. Identifikácia zariadení pomocou IP adries je nepraktická pre užívateľov, preto sa využíva textový formát doménových mien, ale pre zariadenie je 32-bitové (alebo 128-bitové u IP verzie 6) číslo ideálnym formátom pre porovnávanie a prefixové vyhľadávanie [40].

3.1 Priestor doménových mien

V systéme DNS je logický priestor doménových mien organizovaný do hierarchického stromu, nazývaného koreňový strom doménových mien, ktorého príklad je možné vidieť na obrázku 3.1.



Obr. 3.1: Priestor doménových mien systému DNS

Táto štruktúra je podobná súborovému systému v Unixe a pripomína acyklický graf s uzlami a hranami [40]. Koreň stromu sa nazýva the root a jednotlivé uzly sú pomenované textovými reťazcami dĺžky max. 63 znakov. Doména je podstrom v tomto grafe a doménové meno je cesta medzi uzlom (vrcholom domény) a koreňom stromu DNS. Domény sú hierarchicky usporiadané podľa vzdialenosti od koreňa, pričom doména prvej úrovne (TLD[1]) je najbližšia koreňu, nasleduje doména druhej úrovne (SLD[2]) a počet N subdomén, pričom N môže mať hodnotu maximálne 127. Plne kvalifikované doménové meno (FQDN[3]) je sekvenciou mien uzlov až ku koreňu stromu DNS, oddelená bodkami. Na rozdiel od FQDN je Čiastočne kvalifikované doménové meno (PQDN[4]) sekvenciou mien uzlov do konkrétnej domény. Rozdiel medzi FQDN a PQDN je možné vidieť v tabuľke 3.1.

Príklady subdomén (doména vutbr.cz.)	**Príklady FQDN**	**Príklady PQDN** (doména vutbr.cz.)
`fit.vutbr.cz.`	`www.fit.vutbr.cz.`	`eva.fit`
`feec.vutbr.cz.`	`eva.fit.vutbr.cz.`	`www.fit`

Tabuľka 3.1: Príklad subdomén, FQDN a PQDN

DNS databáza obsahuje jednotlivé doménové mená usporiadané do podstromov. Správa domén, vrátane pridávania, rušenia a vytvárania subdomén, je decentralizovaná a deleguje sa na rôzne organizácie [40].

Usporiadanie priestoru doménových mien

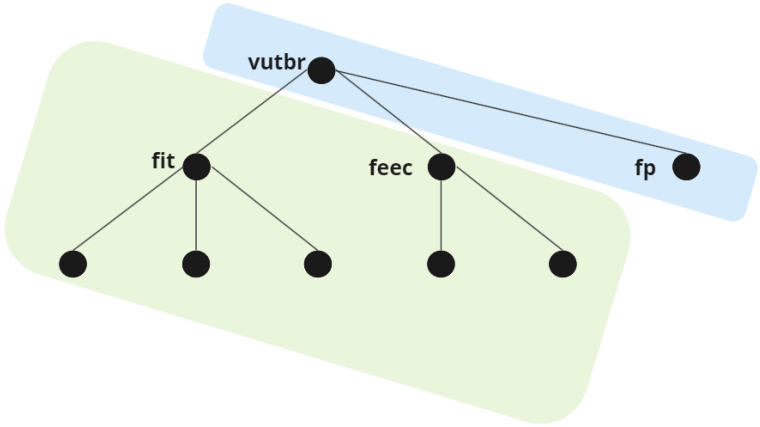
Stromová štruktúra DNS nie je uložená na jednom mieste, ale je rozptýlená medzi lokálnymi DNS servermi, ktoré dohromady tvoria systém DNS. Každá časť tohto stromu je fyzicky uložená na týchto serveroch, ktoré obsahujú informácie o doménach a záznamoch DNS, vrátane informácií o primárnych a sekundárnych DNS serveroch, správcoch domén, poštových serveroch a ďalších [40].

Samotná doména je časťou priestoru adries s jednotným koncovým označením (napríklad doména vutbr.cz), zatiaľ čo zóna je fyzickou časťou tohto priestoru uloženou na konkrétnom serveri. Zóna môže zahŕňať celú doménu alebo len jej časť, tzn. nemusí byť totožná s doménou. Napríklad poskytovateľ internetového pripojenia môže spravovať viacero domén, ktoré tvoria jednu zónu, alebo veľká doména (napríklad edu) môže byť rozdelená a umiestnená na viacerých DNS serveroch. Zóna môže tiež obsahovať subdomény pod jednou správou, ako napríklad subdomény fit.vutbr.cz a feec.vutbr.cz [40]. Existujú aj špeciálne typy zón [40]:

- Zóna typu `stub` obsahuje len informácie o serveroch, ktoré obsluhujú subdomény, ale samotná neobsahuje žiadne údaje.
- Zóna typu `hint` obsahuje zoznam koreňových serverov DNS.

Napríklad doména vutbr.cz môže byť rozdelená na viacero zón, ktoré pokrývajú rôzne časti doménového priestoru a ktoré sú spravované rôznymi subjektami. Príklad tohto členenia je možné vidieť na obrázku 3.2, pričom subdomény fit a feec predstavujú jednu zónu a doména vutbr a subdoména fp ďalšiu subdoménu.

[1] Top Level Domain
[2] Second Level Domain
[3] Fully Qualified Domain Name
[4] Partially Qualified Domain Name



Obr. 3.2: Príklad usporiadania priestoru doménových mien do zón

3.2 Protokol DNS

Princíp protokolu DNS bol prvýkrát navrhnutý v roku 1983 v rámci RFC 882 a 883 [43, 44], pričom štandardom pre protokol DNS a pre komunikáciu v systéme DNS sa stalo v roku 1987 RFC 1035 [45]. Pre posielanie dotazov používa protokol DNS transportný protokol UDP[5] na porte číslo 53, avšak na väčšie prenosy ako napr. prenos zón sa využíva protokol TCP[6] s využitím portu 53. Protokol DNS rozlišuje dva kľúčové prvky systému:

- **Server (nameserver)** – Uchováva údaje z priestoru doménových mien. Tieto údaje sú organizované do zón a umiestnené na jednotlivých serveroch DNS. Základným účelom servera DNS je odpovedanie na dotazy smerujúce na databázu DNS, pričom informácie, ktoré spravuje, sú označované ako autoritatívne. Údaje sú uložené vo forme záznamov DNS (resource records), ktoré môžu byť uložené v lokálnom súbore alebo načítané z iného servera DNS pomocou prenosu zón. Existujú dva základné typy serverov DNS: primárne a sekundárne.

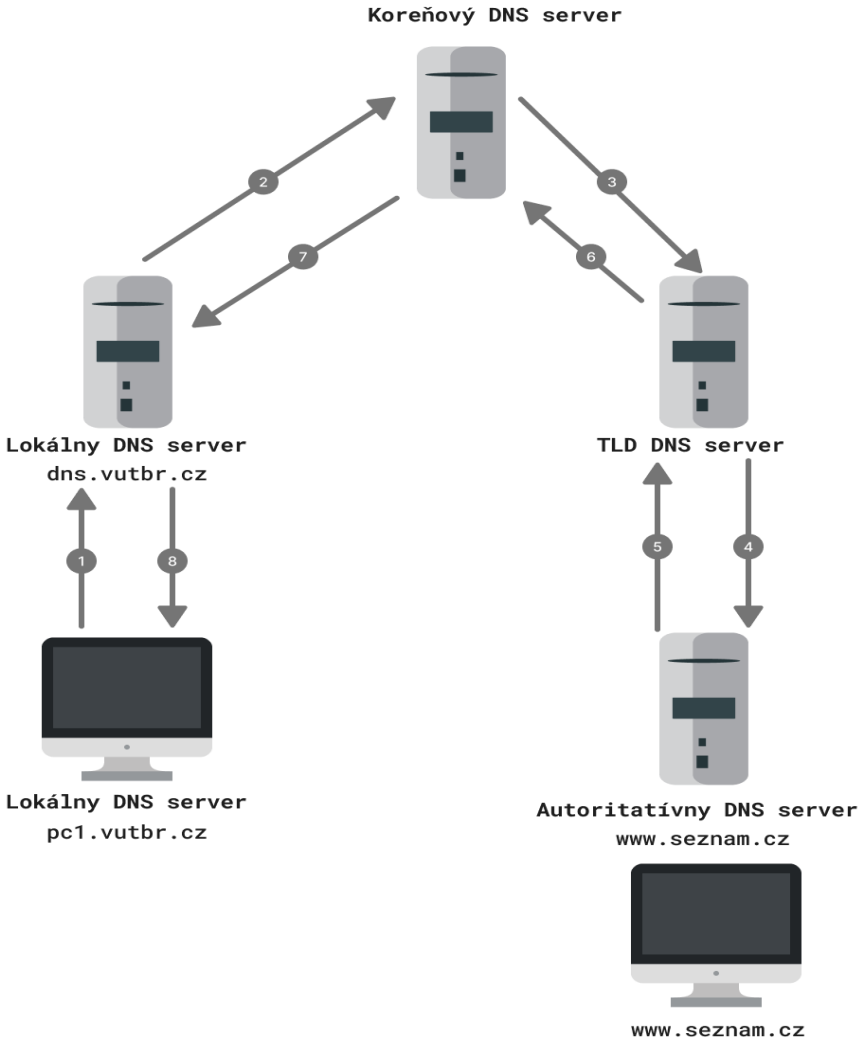
- **Resolver** – Klientský program, ktorý slúži na dotazovanie údajov uložených v systéme DNS. Užívateľské programy, ktoré potrebujú informácie z DNS, komunikujú s týmito údajmi prostredníctvom resolvera. Resolver má za úlohu posielať dotazy na servery DNS, interpretovať odpovede od serverov a poskytovať informácie užívateľským programom. Resolver musí byť schopný pristupovať aspoň k jednému serveru DNS a môže získať odpoveď priamo od tohto serveru, alebo získať odkaz na ďalší server, kde je hľadaná informácia uložená.

V rámci protokolu DNS rozlišujú servery dva typy dotazov:

- **Rekurzívny dotaz** – Dotaz je podobný rekurzii, ako je známa napríklad z programovania. Resolver posiela dotaz o určitý údaj v strome DNS konkrétnemu serveru DNS. Server DNS musí odpovedať na dotaz buď požadovanými údajmi, alebo chybovým hlásením, napríklad, ak nevie odpoveď. Ak server nie je autoritatívny pre hľadané údaje, musí sa spýtať ďalších serverov a nájsť autoritatívnu odpoveď. Môže posielať rekurzívny dotaz na niektorý z autoritatívnych serverov a čakať na odpoveď. Taktiež

[5] User Datagram Protocol
[6] Transmission Control Protocol

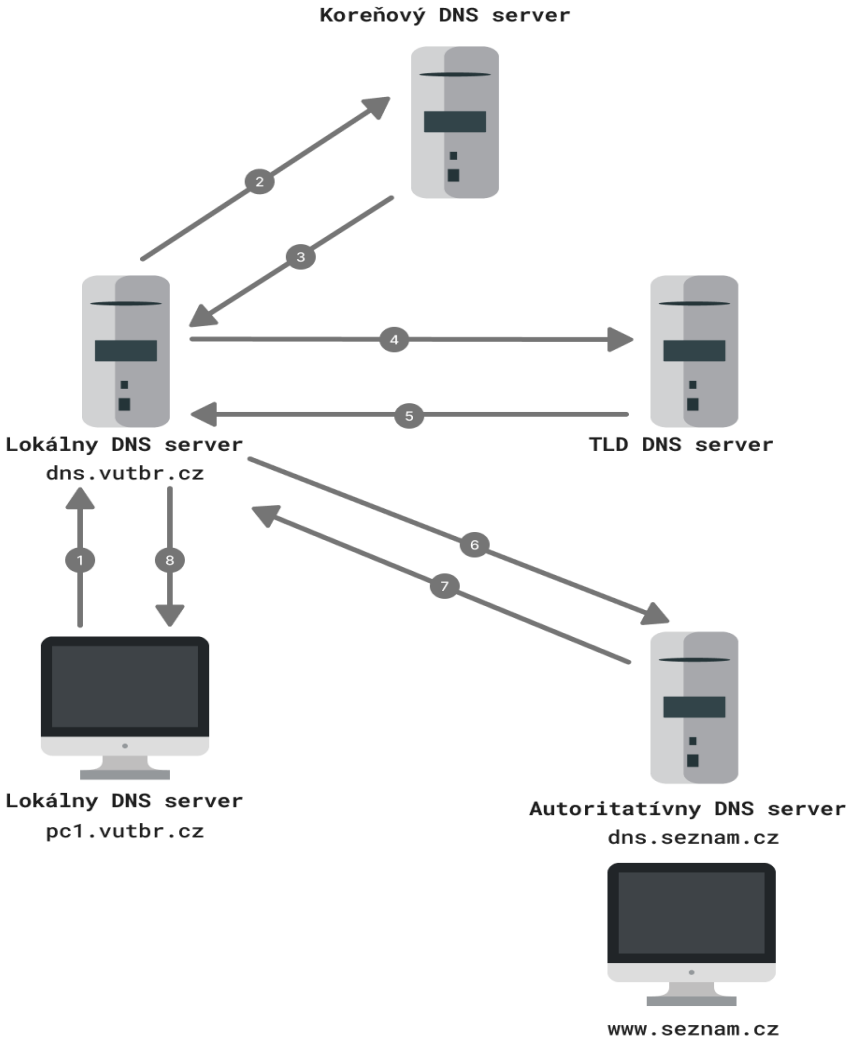
môže posielať iteratívny dotaz a získať odkaz na iný server, ktorý pozná odpoveď. Či server podporuje rekurzívne alebo iteratívne dotazovanie, závisí od jeho konfigurácie. Server DNS, ktorý obdržal rekurzívny dotaz, vždy preposiela rovnaký dotaz na ďalšie servery. Nikdy sa nespýta napríklad na záznamy NS pre hľadanú doménu. Väčšina programov (napríklad nslookup, dig) posiela rekurzívne dotazy [40]. Príklad rekurzívneho dotazovania je možné vidieť na obrázku 3.3.



Obr. 3.3: Rekurzívne dotazovanie v rámci systému DNS

- **Iteratívny dotaz** – Dotaz, ktorý šetrí prácu na strane servera DNS. Pri tomto dotazovaní vráti server resolveru najlepšiu odpoveď, ktorú môže dať. Dotazovaný server DNS sa pozrie do svojej lokálnej databázy a v prípade, ak nenájde odpoveď, vráti adresy serverov, ktoré sú najbližšie hľadanej adrese [40]. Príklad iteratívneho dotazovania je možné vidieť na obrázku 3.4.

Na zníženie počtu dotazov na systém DNS využívajú servery DNS lokálnu vyrovnávaciu pamäť (cache), kde ukladajú odpovede od serverov pre budúce vyhľadávanie. Niektoré verzie serverov DNS implementujú aj negatívnu cache pre ukladanie informácií o neexistujúcich záznamoch v danej doméne [40].



Obr. 3.4: Iteratívne dotazovanie v rámci systému DNS

Pakety systému DNS obsahujú veľké množstvo dôležitých informácií v hlavičke, ktorá je kritickou súčasťou každého dotazu. Obsahuje informácie nevyhnutné pre riadenie a interpretáciu dotazu, napr. pole RCODE indikuje úspešnosť jednotlivých dotazov (napr. No error, NxDomain atď.), RA súvisí s dostupnosťou rekurzívneho dotazovania na serveri a význam ďalších polí je možné nájsť v RFC 1035 [45]. Obrázok hlavičky DNS správy je možné vidieť v obrázku 3.5.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ID															
QR	**Opcode**				**AA**	**TC**	**RD**	**RA**	**Z**			**RCODE**			
QDCOUNT															
ANCOUNT															
NSCOUNT															
ARCOUNT															

Obr. 3.5: Formát hlavičky paketu DNS

Súčasťou dotazu a odpovede sú záznamy, ktorých typ je dôležitou súčasťou štruktúry systému DNS, pretože určujú typ informácií uložených v záznamoch. Medzi najbežnejšie typy záznamov DNS patria záznamy [40]:

- **A**: Typ záznamu, ktorý mapuje doménové meno na IPv4 adresu. Slúži na prevod doménových mien na konkrétne IP adresy.
- **AAAA**: Podobne ako A záznam, ale mapuje doménové meno na IPv6 adresu. Slúži na prevod doménových mien na konkrétne IPv6 adresy.
- **CNAME**: Slúži na vytvorenie aliasu pre existujúce doménové meno. Používa sa na presmerovanie jedného doménového mena na iné.
- **MX**: Určuje mailový server pre konkrétnu doménu, indikuje, kam majú byť doručené e-maily.
- **TXT**: Umožňuje pridanie textovej informácie k doménovému menu. Používa sa pre rôzne účely, ako je napríklad overovanie domény pre SPF (Sender Policy Framework).
- **PTR**: Slúži na mapovanie IP adresy na doménové meno. Používa sa v reverzných DNS záznamoch.
- **NS**: Určuje autoritatívne DNS servery pre konkrétnu doménu. Definuje, ktoré servery sú zodpovedné za spravovanie DNS pre túto doménu.
- **SRV**: Poskytuje informácie o službe dostupnej v určitej doméne. Obsahuje informácie ako doménové meno služby, port, prioritu a váhu.

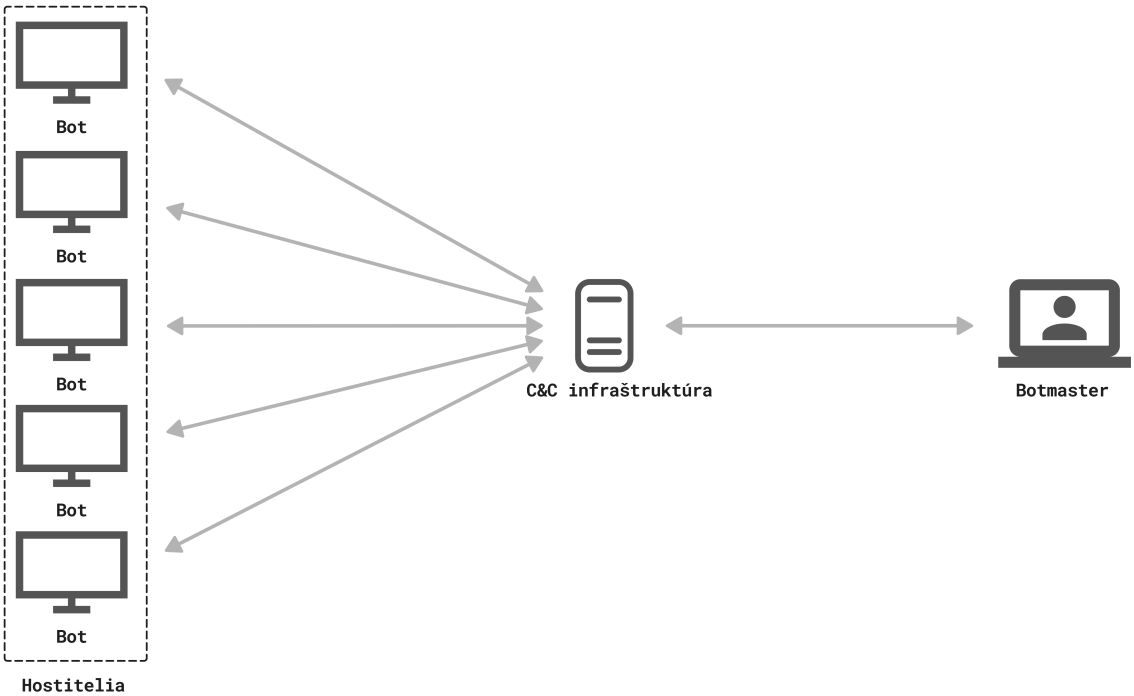
Kapitola 4

Botnet a využitie DGA

V tejto kapitole je predstavený koncept botnetu s podrobným vysvetlením infikácie jednotlivých hostiteľov, procesu vytvárania botnetu a používaných architektúr. Taktiež sú podrobne analyzované metódy, ktoré útočníci využívajú na zakrytie a ochranu svojich sietí s cieľom uniknúť odhaleniu, medzi ktoré patria metódy Fast Flux, prípadne Domain Fluxing, s ňou spojené využitie DGA a ďalšie.

4.1 Botnet

Z dôvodu lepšieho porozumenia budú najskôr predstavené jednotlivé komponenty botnetu ako sú zobrazené na obrázku 4.1 a následne bude uvedený pojem botnet.



Obr. 4.1: Prvky botnetu

Komponenty botnetu

- **Bot** – Software inštalovaný na hostiteľovi, ktorý je schopný vykonávať sériu akcií, zvyčajne škodlivých. Tento druh softvéru môže byť nainštalovaný na počítačoch obetí rôznymi spôsobmi, vrátane vírusových mechanizmov alebo prístupom k infikovaným stránkam. Botom môže taktiež byť nazývaný infikovaný hostiteľ. Z dôvodu čo najväčšej dostupnosti sú typicky nastavení tak, aby sa inicializovali pri každom spustení počítača obete [67].

- **Botmaster** – Osoba, ktorá má kontrolu nad botnetom a vydáva príkazy botom na vykonávanie nelegálnych aktivít. Môže taktiež získavať finančné výhody tým, že prenajme svoju sieť iným používateľom, napr. na zasielanie nevyžiadanej pošty, ako napríklad spam [67].

- **Hostitelia** – Zariadenia pripojené k sieti internet, ktoré boli infikované softvérom šíreným botmasterom cez určitý mechanizmus šírenia. Po infikovaní sa z týchto strojov stávajú boti (prípadne zombie) a môžu byť použité ako útočné platformy proti ďalším hostiteľom [67].

- **Infraštruktúra C&C** – Kritická súčasť botnetu, pozostávajúca z botov a riadiacej entity, ktorá môže byť buď centralizovaná alebo decentralizovaná. Jeden alebo viac komunikačných protokolov sa používa na vydávanie príkazov infikovaným zariadeniam a koordináciu ich akcií. Architektúra infraštruktúry C&C určuje robustnosť, stabilitu a reakčný čas botnetu [67].

Botnet predstavuje sieť tvorenú mnohými hostiteľmi (botmi, prípadne zombie), ktoré sú infikované a ovládané útočníkom (botmasterom) s cieľom vykonávať škodlivé aktivity [77]. Botmaster využíva C&C[1] server na riadenie jednotlivých botov a vykonávanie rôznych kybernetických útokov ako napríklad distribuované odopretie služby (DDoS[2]), spam, phishing, podvodné prekliky (click fraud), krádeže informácií atď. [77].

4.1.1 Životný cyklus botnetu

Samotný životný cyklus botnetu je tvorený niekoľkými fázami, ktoré zahŕňajú infikáciu hostiteľov, ich transformáciu na boty a ich následné využitie na vykonávanie útokov, či nelegálnych aktivít [67]. Tento cyklus je možné vidieť na obrázku 4.2 a môže pozostávať z nasledujúcich etáp:

1. **Počiatočná infekcia** – Hostiteľ je infikovaný škodlivým softvérom a stáva sa potenciálnym botom [67].

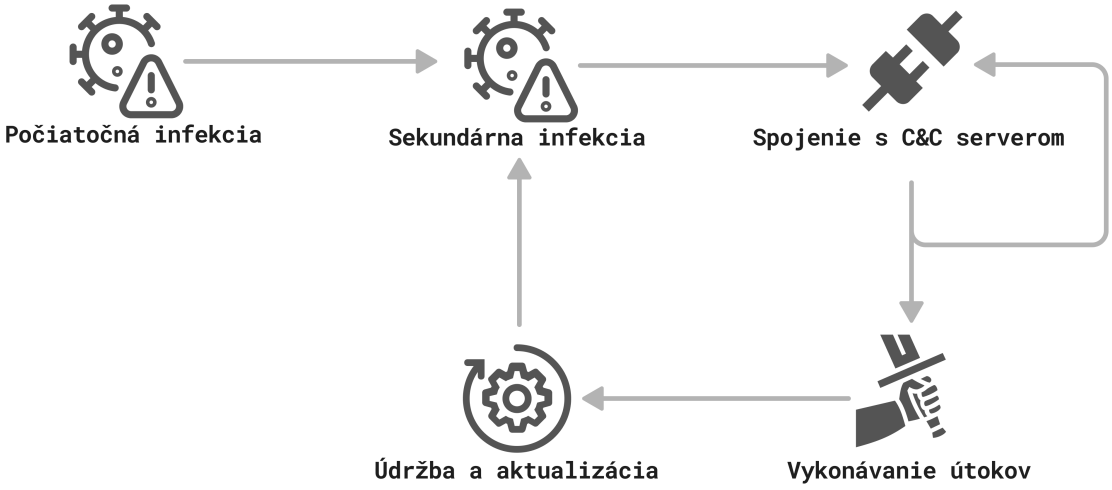
2. **Sekundárna infekcia** – Infikovaný hostiteľ spustí program, ktorý vyhľadáva malvérové binárne súbory v určitom databázovom systéme. Po stiahnutí a spustení týchto binárnych súborov sa hostiteľ správa ako skutočný bot [67].

3. **Spojenie s C&C serverom** – Bot nadväzuje spojenie s Command and Control serverom, ktorý poskytuje pokyny a prípadné aktualizácie. Toto spojenie je nevyhnutné pre vykonávanie škodlivých aktivít [26, 67].

[1] Command and Control
[2] Distributed Denial of Service

4. **Vykonávanie útokov** – Bot je pripravený vykonávať príkazy z C&C servera, čo môže zahŕňať rôzne formy útokov ako DDoS, phishing, krádeže informácií a ďalšie [26, 67].

5. **Údržba a aktualizácia** – Botmaster pravidelne aktualizuje softvér v botnete, aby unikol detekcii a pridával nové funkcie. Toto môže spôsobiť zmenu v správaní botov, čo môže byť pozorované napríklad v DNS dotazoch alebo zdieľaní súborov [26, 67].



Obr. 4.2: Životný cyklus botnetu

4.1.2 Používané architektúry

Existujú rôzne architektúry botnetov, ktoré ovplyvňujú spôsob, akým komunikuje C&C server s infikovanými zariadeniami (botmi). Tieto architektúry sa obvykle klasifikujú ako centralizované, decentralizované a hybridné [30, 77], prípadne i náhodné [67]. Každá z týchto architektúr má svoje vlastné charakteristiky a spôsoby fungovania.

V nasledujúcich sekciách budú detailne rozobrané tieto architektúry a ich špecifikácie, ako aj ich vplyv na prevádzku a správu botnetu.

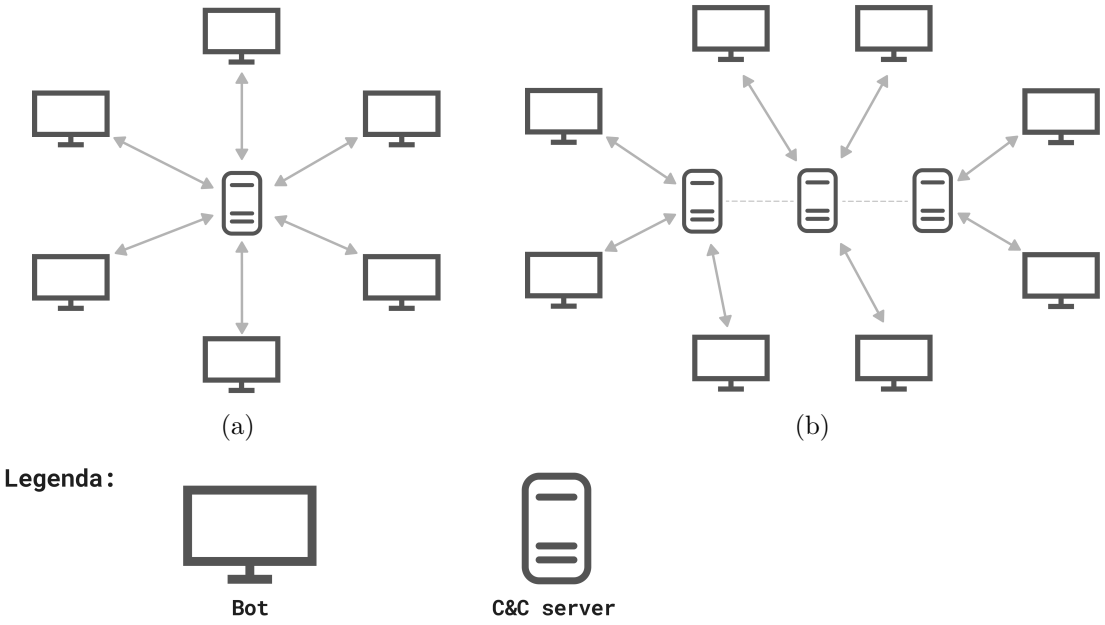
Centralizovaná architektúra

Najjednoduchšou architektúrou komunikácie botnetu je centralizovaná topológia. Táto architektúra využíva centrálny dedikovaný C&C server, ku ktorému sa každý bot priamo pripája, viď obrázok 4.3. Táto topológia je ľahko nastaviteľná, má nízku odozvu a vysokú škálovateľnosť. Je ľahko nastaviteľná, pretože nevyžaduje špeciálne požiadavky na protokol, preto môžu byť použité jednoduché protokoly, ako napríklad Internet Relay Chat (IRC) alebo HTTP. Nízka odozva a vysoká škálovateľnosť sú spôsobené jednoduchou štruktúrou siete, v ktorej sú príkazy prenášané priamo z C&C servera na každého bota v botnete [77].

Centralizovaná architektúra prináša výhody na úkor nízkej odolnosti, keďže C&C server predstavuje jediný bod zlyhania. Na znefunkčnenie celej botnetovej siete postačuje vypnúť daný server, prípadne znemožniť komunikáciu servera s botnetom. Tento nedostatok je

možné čiastočne riešiť prostredníctvom viacerých replikovaných C&C serverov namiesto jedného [77]. Na obrázku 4.3 je možné vidieť príklad centralizovanej architektúry s jedným serverom (viď obr. 4.3a) i s viacerými replikovanými servermi (viď obr. 4.3b).

Problémom C&C serverov je, že adresy musia byť pevne zakódované do botnetu. Šifrovanie alebo zatemňovanie kódu programu má iba dočasnú účinnosť, pretože kód môže byť po nejakom čase dešifrovaný. Okrem toho môže byť C&C server detegovaný pozorovaním sieťovej prevádzky bota, čo je možné zmierňovať napríklad použitím DGA [67].



Obr. 4.3: Centralizovaná architektúra siete Botnet

Decentralizovaná architektúra

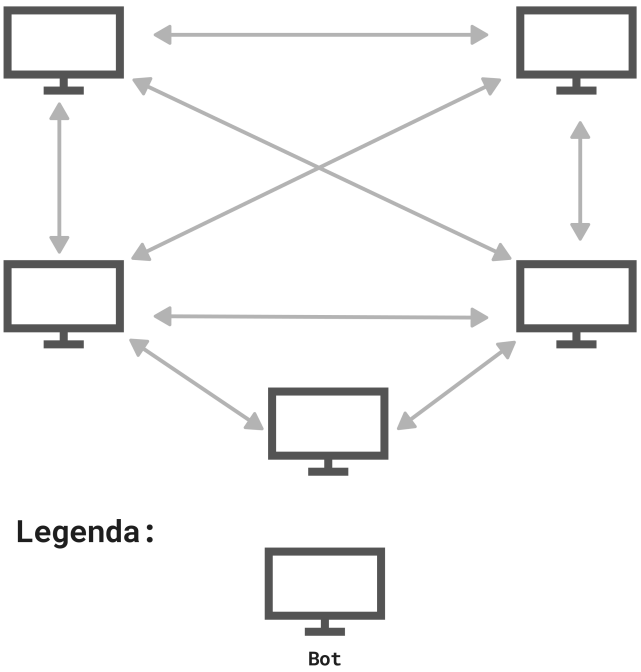
Na zvýšenie odolnosti voči možnému zlyhaniu, krádeži, alebo úplnému zničeniu siete sa využíva decentralizovaná architektúra typu Peer-to-Peer (P2P). Táto architektúra sa skladá výlučne z botov, pričom každý z nich môže fungovať ako potenciálny C&C server [30]. Príkazy od botmastera môžu dosiahnuť celú botnetovú sieť iba vtedy, ak je každý bot spojený s minimálne jedným ďalším botom [77], ktorým môže preposielať príkazy, viď obrázok 4.4.

V plne prepojenej sieti je každý bot spojený so všetkými ostatnými botmi. To zabezpečuje nízku latenciu komunikácie, avšak s narastajúcim počtom botov môže byť takýto prístup neškálovateľný. Väčšina P2P botnetov preto nie je úplne prepojená, čo znižuje viditeľnosť siete a minimalizuje počet komunikačných spojení [77]. Dôvodom je zvyšovanie odolnosti voči detekcii danej siete, avšak za cenu zvýšenej latencie komunikácie v rámci botnetu.

Implementácia P2P[3] topológií je náročná z dôvodu vyhľadávania počiatočných peerov a spoľahlivého distribuovania príkazov. Jeden spôsob je vložiť známe pevné adresy peerov priamo do binárnych súborov bota, prípadne je možné využiť cache servery existujúcich P2P aplikácií. Avšak tieto metódy sú ľahko detekovateľné a môžu ohroziť bezpečnosť siete. Pre

[3]Peer-to-Peer

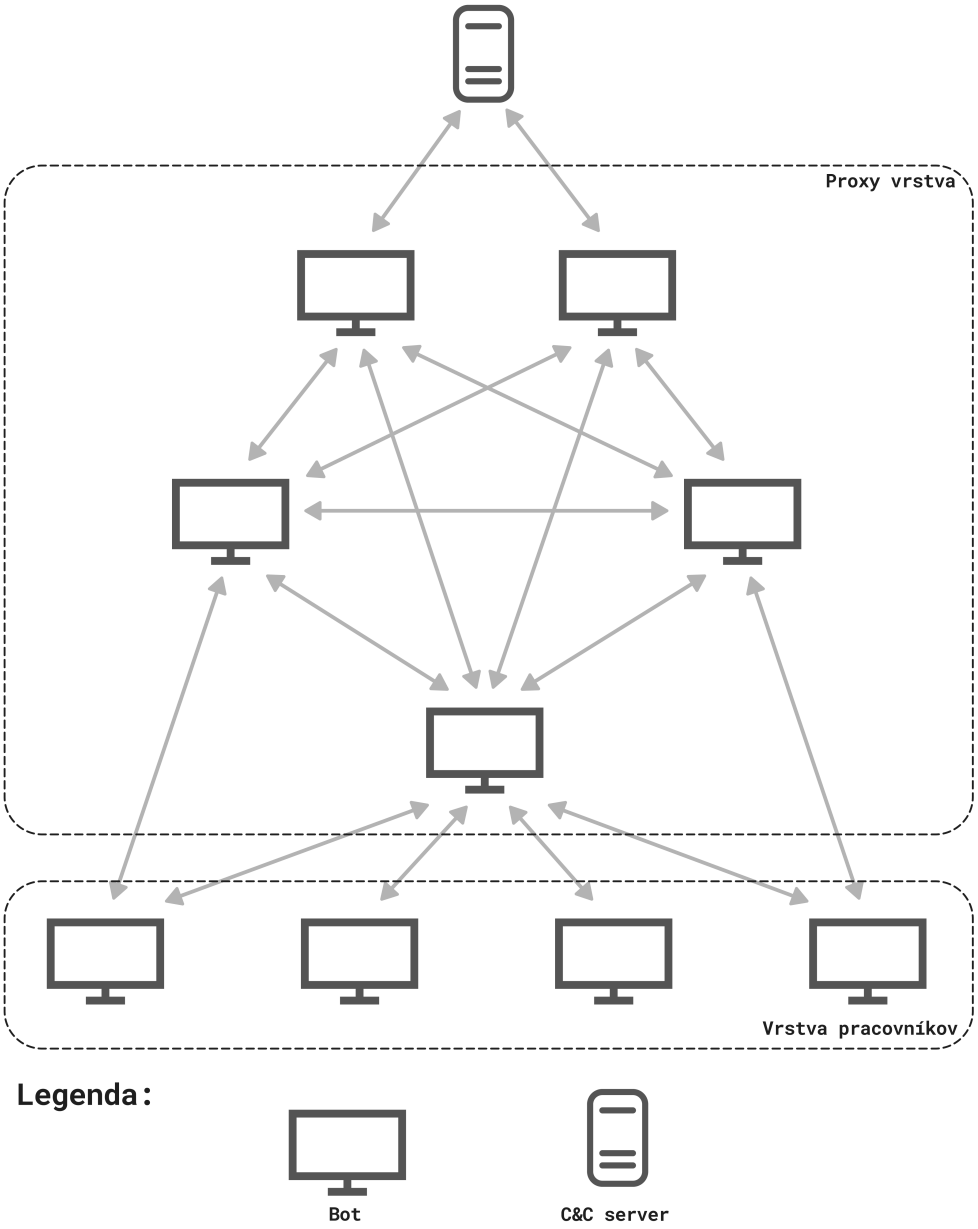
minimalizáciu rizika sú niektoré boty schopné náhodne skenovať internet a hľadať nových peerov [67, 77].



Obr. 4.4: Decentralizovaná architektúra siete Botnet

Hybridná architektúra

Hybridná architektúra botnetov kombinuje výhody centralizovaných a decentralizovaných topológií [67]. Tento prístup umožňuje prekonať nedostatky oboch architektúr, čím sa znižuje riziko v prípade výpadku časti siete a zabezpečuje spoľahlivé doručovanie príkazov. Pridaním proxy vrstvy, i vrstvy pre vykonávanie príkazov (vrstva pracovníkov) sa znižuje viditeľnosť proxy vrstvy a poskytuje ďalšia úroveň ochrany C&C serverov voči odhaleniu. Taktiež sa uplatňujú rozličné využitia centralizovaných a decentralizovaných topológií v rôznych častiach hybridnej architektúry, čo prispieva k flexibilite a robustnosti systému [77]. Príklad hybridnej architektúry botnetov je zobrazený na obrázku 4.5



Obr. 4.5: Hybridná architektúra siete Botnet

4.1.3 Metódy vyhýbania sa detekcii

Botnety sa neustále vyvíjajú a využívajú sofistikované metódy na vyhnutie sa odhaleniu. Ich cieľom je stať sa ťažko identifikovateľnými a znemožniť ich odhalenie [84]. Metódy vyhýbania sa detekcii je možné rozdeliť na:

- **Techniky založené na rýchlych zmenách (Fluxing)**
  - Fast Flux (FF) – Technika využívajúca rýchle mapovanie medzi viacerými IP adresami a jedným doménovým menom [84].

  - Domain Flux (DF) – Využíva DGA[4] na generovanie doménových mien, čím zabezpečuje, že botov a C&C servery nie je možné jednoducho identifikovať alebo zablokovať na základe pevne stanovej doménovej adresy. Tento proces neustále mení doménové mená, čo robí detekciu a blokovanie komunikácie s takýmito servermi pre bezpečnostné systémy omnoho náročnejšie. Využitie DGA v rámci botnetov bude bližšie popísané v sekcii 4.2 [67, 84].

- **Techniky založené na tunelovaní a manipulácii s protokolmi** – Využívajú sa rôzne protokoly (HTTP, ICMP, VoIP[5]), tunelovanie IPv6, i šifrovaná prevádzka na skrytie komunikácie botnetu a zťaženie detekcie [67].

- **Techniky založené na dynamických vzoroch** – Používa sa náhodnosť komunikačných vzorov, výber portov a priradenie úloh medzi robotmi na vyhnutie sa detekcii [67].

- **Techniky založené na zneužití legitímnych aplikácií** – Niektoré botnety zneužívajú legitímne aplikácie ako Skype[6] na vytvorenie parazitárnych sietí [67].

4.2 DGA

Algoritmus generovania domén (Domain Generation Algorithm - DGA) je metóda využívaná na dynamické generovanie veľkého počtu zdanlivo náhodných doménových mien. Využívajú sa rôzne spôsoby generovania domén, často s použitím rôznych parametrov alebo seedov, ako sú numerické konštanty, aktuálny čas, dátum [69] alebo trendy z verejných platforiem ako je Facebook, Twitter, prípadne podľa ceny určitej komodity na burze [56, 77].

Táto technika je často využívaná botnetmi, čím umožňuje útočníkom neustále meniť domény, cez ktoré komunikujú s infikovanými zariadeniami (botmi). C&C servery a nakazené zariadenia musia medzi sebou zdieľať rovnaký seed, tzn. rovnakú počiatočnú podmienku z dôvodu, aby mohlo medzi nimi vzniknúť spojenie, takzvaný rendezvous point [84]. Dynamická povaha generovaných domén robí detekciu a blokovanie botnetovej komunikácie pre obranné mechanizmy zložitou úlohou z dôvodu ľahkého obchádzania statických zoznamov blokovaných doménových mien. DGA naviac umožňuje botnetom vyhnúť sa zablokovaniu doménovými reputačnými službami. Toho je možné dosiahnuť tým, že si registrujú krátkodobo existujúce domény tesne pred ich použitím, čím minimalizujú riziko, že budú označené ako podozrivé alebo nebezpečné [56].

4.2.1 Rozdelenie algoritmov generovania domén

Algoritmy generovania domén (DGA) môžu byť kategorizované na základe niekoľkých kľúčových vlastností nasledovne [56]:

1. Rozdelenie podľa zdroja počiatočnej podmienky (seedu):

   - **Časová závislosť** – Zahŕňa časový zdroj (napr. systémový čas kompromitovaného hostiteľa alebo dátumové pole v HTTP odpovedi). V dôsledku toho majú vygenerované domény iba obmedzenú platnosť, tzn. počas obdobia, kedy sú tieto domény vyhľadávané kompromitovaným systémom.

[4] Domain Generation Algorithm
[5] Voice over Internet Protocol
[6] Skype: www.skype.com

- **Determinizmus** – Uvádza úroveň pozorovateľnosti a dostupnosti parametrov, ktoré môžu byť deterministické alebo nedeterministické (napr. trendy z verejných platforiem ako je Facebook, Twitter, prípadne podľa ceny určitej komodity na burze).

2. Rozdelenie podľa schémy generovania:

   - **DGA založené na aritmetike** – Vypočítavajú postupnosť hodnôt, ktoré buď majú priamu ASCII reprezentáciu použiteľnú pre doménové meno, alebo označujú offset v jednom či viacerých natvrdo definovaných poliach, ktoré tvoria abecedu DGA.
   - **DGA založené na hašovacej funkcii** – Využívajú hašovacie funkcie (napr. MD5, SHA256) na generovanie domén.
   - **DGA založené na slovníkoch** – Kombinujú slová zo zoznamov na vytvorenie menej náhodných a náročnejšie odhaliteľných domén.
   - **DGA založené na permutácii** – Odvodzujú nové domény permutáciou pôvodného doménového mena.

Kombináciou zdrojov počiatočnej podmienky a schém generovania môžu vzniknúť rôzne typy DGA, ktoré je možné rozdeliť na [56]:

- **TID-A** – Časovo nezávislé, deterministické, aritmetické.
- **TDD-A** – Časovo závislé, deterministické, aritmetické.
- **TDD-W** – Časovo závislé, deterministické, založené na slovníku.
- **TDD-H** – Časovo závislé, deterministické, založené na hashi.
- **TDN-A** – Časovo závislé, nedeterministické, aritmetické.
- **TID-P** – Časovo nezávislé, deterministické, permutačné.

Toto systematické rozdelenie umožňuje štruktúrované a efektívne zaradenie DGA na základe vlastností ich seedu, schém generovania a výsledných typov, s cieľom lepšie porozumieť, ako funguje generovanie domén pomocou algoritmov generovania doménových mien (DGA).

4.2.2 Rodiny DGA

Rodina DGA sa odkazuje na skupinu malvérov alebo botnetov, ktoré využívajú rovnaký typ alebo variantu algoritmu generovania domén (DGA). Každá rodina je typická svojou špecifickou sadou charakteristík a správania, ktoré môžu byť špecifické pre danú hrozbu alebo útočníka [56]. Nižšie uvedená tabuľka obsahuje typy DGA a ich priradenie k jednotlivým rodinám.

Typ	Meno rodiny DGA
TDD-A	Bamital, Corebot, CryptoLocker, Geodo, Gameover DGA, Gameover P2P, Gootkit, Gozi, Matsnu, Mewsei, Murofet 1, Murofet 2, Necurs, Nymaim, Pushdo, Pykspa 1, Pykspa 2, QakBot, Ranbyus, Szribi, Torpig, UrlZone, Virut
TID-A	Banjori, DirCrypt, Feodo, Fobber, Hesperbot, Kraken, Ramdo, Ramnit, Redyms, Rovnix, Shifu, Simda, TinyBanker, Tempedreve
TDD-H	Dyre
TID-P	VolatileCedar

Tabuľka 4.1: Príslušnosť jednotlivých rodín k jednotlivým typom DGA [56]

Príklad doménových mien vytvorenými jednotlivými rodinami DGA je možné vidieť v tabuľke 4.2.

Meno rodiny DGA	Príklady domén vytvorených pomocou DGA
Bamital	xhbqxeanj.biz, naksrcyam.com
Corebot	m4gpqxct10wpuvc.sg, g6axa8qrsd3jc2wdkp1.ddns.net
Dyre	k82b4974eba4851617bfbc462ce1ae5320.hk
VolatileCedar	getadobeflashplayer.net, etadobeflasghplayer.net

Tabuľka 4.2: Príklady domén pre jednotlivé rodiny DGA

Kapitola 5

Návrh detekčného systému

Táto kapitola je venovaná predstaveniu návrhu detekčného nástroja botnetov založených na DGA. V prvej časti budú definované požiadavky systému v sekcii 5.1. Následne budú uvedené používané dátové sady a jednotlivé bloky systému, pričom bude vysvetlený ich jednotlivý účel.

5.1 Požiadavky na systém

Pri návrhu detekčného systému boli stanovené kľúčové požiadavky, ktoré zohľadňujú jeho účel a funkcionalitu. Tieto požiadavky slúžia na usmernenie pre vývoj a implementáciu systému.

1. **Presnosť** – Detekčný systém by mal byť schopný poskytovať presné výsledky pri identifikácii DGA domén a benígnych domén.
2. **Efektivita a Rýchlosť** – Systém by mal byť navrhnutý tak, aby dosiahol vysokú efektivitu a rýchlosť pri analýze DNS paketov a klasifikácii domén. Rýchle spracovanie a odpoveď je kritická z dôvodu reakcie systému v reálnom čase na nové domény.
3. **Škálovateľnosť** – Systém by mal byť škálovateľný a schopný zvládať veľké objemy DNS dát. To znamená, že by mal byť schopný spracovávať stále rastúce množstvo dát bez straty výkonu a presnosti.
4. **Flexibilita** – Systém by mal byť navrhnutý s ohľadom na flexibilitu a možnosť jednoduchej aktualizácie a rozšírenia. To môže zahŕňať pridávanie nových metód detekcie a prispôsobenie sa novým trendom v oblasti kybernetickej bezpečnosti.
5. **Prenositeľnosť** – Systém by mal byť navrhnutý s ohľadom na prenositeľnosť, aby bol schopný efektívne fungovať na rôznych platformách a operačných systémoch. Táto vlastnosť umožní systému nasadenie v rôznych prostrediach a zabezpečí jeho použiteľnosť vo viacerých kontextoch bez väčších úprav.
6. **Integrácia s inými systémami** – Systém by mal byť schopný integrovať sa s inými bezpečnostnými systémami a nástrojmi, aby poskytoval celkovú ochranu siete.
7. **Užívateľské rozhranie** – Implementovaný systém by mal mať užívateľsky príveti rozhranie, ktoré umožní správcom siete a bezpečnostným analytikom jednoduché monitorovanie a správu identifikovaných domén.

5.2 Dátové sady

Kvalitné dátové sady sú základom pre presné a spoľahlivé fungovanie detekčných systémov a klasifikátorov. Tieto sady poskytujú nevyhnutné informácie, ktoré umožňujú týmto systémom lepšie rozlíšiť medzi nežiaducimi a povolenými entitami.

Kritickým faktorom je kvalita a reprezentatívnosť dátových sád. Kvalitné dáta zabezpečujú presnosť a spoľahlivosť v identifikácii, zatiaľ čo reprezentatívne dáta umožňujú systémom generalizovať a správne aplikovať naučené vzory na nové, doposiaľ nevidené údaje. Z toho dôvodu boli vytvorené dátové sady obsahujúce:

- Viac ako 123 miliónov DGA domén a 93 rodín získaných z DGA archívu Fraunhofer Institute for Communication, Information Processing and Ergonomics FKIE.
- Viac ako 95 miliónov DGA domén, 108 rodín získaných z HYDRA dátovej sady [15].
- 226 tisíc benígnych domén získaných v rámci výskumnej skupiny NES@FIT.
- 7 miliónov benígnych domén získaných od inštitútu CESNET

Výsledné spracované dátové sady obsahujú až 196 miliónov DGA domén, 115 rodín a benígne domény, pričom budú využité na filtráciu známych malígnych a benígnych domén, ale i na trénovanie a testovanie klasifikátorov strojového učenia. Kolekcia dát obsahujúca vzorky rodín DGA tvorí 115 súborov vo formáte Parquet[1]. Každý z týchto súborov reprezentuje jednu rodinu DGA, pričom rozloženie jednotlivých rodín v tejto zbierke je veľmi nevyvážené.

5.3 Ukladanie dát

Pre efektívne ukladanie dát bude využitá kombinácia dokumentovo orientovaných databázových systémov a relačných databázových systémov. Tento prístup umožní flexibilné riešenie pre rôzne typy dát a potreby detekčného nástroja, tzn. týmto spôsobom sa súčasne distribuuje záťaž medzi rôzne systémy, čím sa zvýši celková výkonnosť a efektivita spracovania dát v systéme.

Dokumentovo orientované databázové systémy

Dokumentovo orientované databázové systémy sú vhodné pre ukladanie neštruktúrovaných alebo rozsiahlych dát. Tieto systémy používajú formát ako napríklad BSON (binárne zakódovaný JSON[2]) na ukladanie dát. Z toho dôvodu boli vybrané pre ukladanie výsledkov detekcie a taktiež zoznamov Blacklist a Whitelist, ktoré budú popísané v sekcii 5.5.2.

Relačné databázové systémy

Relačné databázové systémy sú využívané pre štruktúrované dáta a vynikajú vo vysokých rýchlostiach, flexibilite a spoľahlivosti pri spracovaní dát. Tieto systémy podporujú SQL[3], čo zjednodušuje manipuláciu s dátami. Relačné databázové systémy budú využívané vo webovej aplikácii, ktorej návrh je popísaný v sekcii 5.7. Výber bol realizovaný na základe

[1] Parquet: https://parquet.apache.org/
[2] JSON: https://www.json.org/
[3] Structured Query Language

minimálnej podpory dokumentovo orientovaných databázových systémov technológiami slúžiacimi na vytváranie aplikačných programovacích rozhraní (API) v kontexte správy užívateľov.

Objektovo-relačný mapovací framework

Na prácu s relačnými databázovými systémami bude využitý objektovo-relačný mapovací framework (ORM framework), ktorý umožňuje manipuláciu s databázou pomocou objektov v programovacích jazykoch. Tieto frameworky zjednodušujú vývoj a udržiavanie aplikácií, poskytujú podporu pre rôzne operácie s databázou a slúžia na efektívne využívanie dát.

5.4 Komunikácia medzi blokmi

Pre efektívnu komunikáciu medzi jednotlivými komponentami detekčného nástroja, ktoré sú podrobne opísané v nasledujúcich sekciách, bol zvolený sprostredkovateľ správ fungujúci na princípe fronty. Tento systém umožňuje asynchrónnu komunikáciu medzi komponentmi, čo znamená, že odosielateľ správy nemusí čakať na prijatie správy prijímateľom, aby mohol pokračovať vo svojej činnosti. Týmto spôsobom je zabezpečená vysoká efektivita a škálovateľnosť celého nástroja. Vlastnosti, ktoré by mal systém fronty správ pre tento účel spĺňať, zahŕňajú:

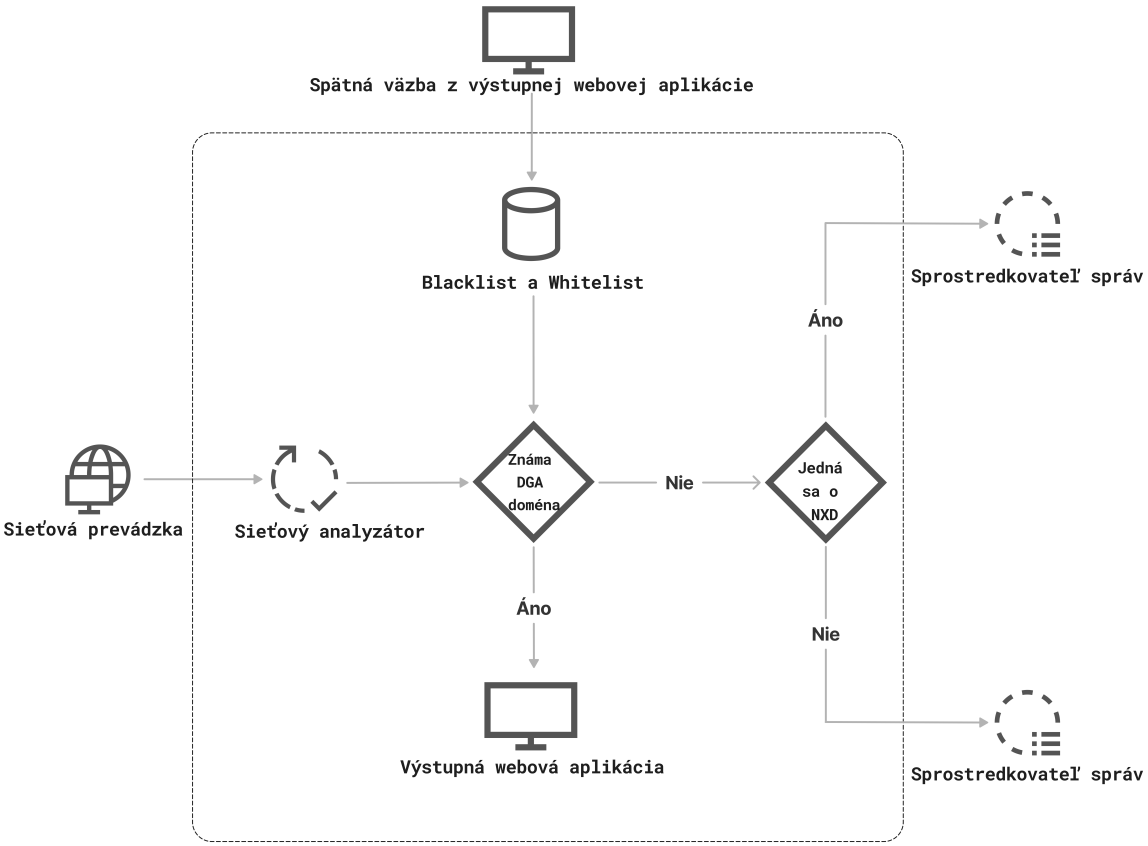
1. **Vysoká dostupnosť a odolnosť** – Systém by mal byť navrhnutý tak, aby zabezpečil kontinuálnu operáciu aj v prípade čiastočných výpadkov, minimalizujúc tak riziko prerušenia celého procesu detekcie.
2. **Zabezpečenie doručenia správ** – Je dôležité, aby systém zahŕňal mechanizmy, ktoré zaručujú, že každá správa bude doručená iba raz a v správnom poradí, aby sa zachovala integrita dát.
3. **Škálovateľnosť** – Systém musí byť schopný zvládať rastúce objemy dát a počet komponentov bez potreby radikálnych zmien v jeho základnej architektúre.
4. **Flexibilita v spracovaní správ** – Podpora rôznych formátov správ a možnosti ich transformácie alebo filtrovania v priebehu prenosu sú kľúčové pre adaptabilitu systému na meniace sa požiadavky.
5. **Monitorovanie a logovanie** – Systém by mal obsahovať nástroje na sledovanie jeho aktivity a logovanie udalostí, slúžiacich na efektívnu diagnostiku a optimalizáciu procesov.

Výber systému fronty správ, ktorý spĺňa tieto kritériá, je kľúčový pre zabezpečenie spoľahlivej a efektívnej komunikácie medzi komponentmi detekčného nástroja. Takýto systém umožňuje efektívne rozdelenie úloh a spracovanie dát v reálnom čase, čím výrazne prispieva k celkovej efektivite a schopnosti detekčného systému reagovať na dynamické podmienky v prostredí, v ktorom je nasadený.

5.5 Zber a kategorizácia neznámych DNS domén

Prvý blok detekčného nástroja, pomenovaný Detektor, sa zaoberá zbieraním i kategorizáciou neznámych domén systému DNS. Skladá sa z menších komponentov, ktoré vykonávajú

jednotlivé činnosti nezávisle od seba, pričom bol kladený dôraz na efektivitu a plnenie definovaných požiadaviek systému, ako je bližšie popísané v sekcii 5.1. Tento proces je vizualizovaný na obrázku 5.1, kde je detailne znázornený postup od získavania neznámych DNS domén po ich kategorizáciu. Jednotlivé časti návrhu prvého bloku budú vysvetlené ďalej v texte.



Obr. 5.1: Konceptuálny návrh prvého bloku detekčného nástroja

5.5.1 Sieťový analyzátor

Sieťový analyzátor má za úlohu sledovať, analyzovať a zachytávať pakety v sieti, ktoré sú generované systémom DNS. Jeho implementácia by mala byť prenositeľná a taktiež čo najefektívnejšia a najrýchlejšia, aby spĺňala požiadavky na systém definované v sekcii 5.1.

Pre implementáciu je možné využiť rôzne knižnice alebo nástroje, ktoré sú k dispozícii v rôznych programovacích jazykoch a platformách. Medzi tieto nástroje patria napríklad knižnice na zachytávanie a spracovanie sieťových paketov alebo nízkoúrovňové programovacie rozhranie na prácu so sieťovými soketmi.

5.5.2 Blacklist a Whitelist

V tejto časti návrhu je predstavený koncept implementácie Blacklistu a Whitelistu pre domény založené na DGA a benígne domény. Na ukladanie relevantných údajov sa využíva dokumentovo orientovaný databázový systém, ktorý bol predstavený v sekcii 5.3.

Blacklist

Blacklist bude obsahovať známe DGA domény, ktoré sú explicitne považované za nebezpečné alebo nežiaduce. Kolekcia `blacklist` v databázovom systéme bude obsahovať dokumenty s informáciami o zakázaných doménach. Názov domény bude zachytený v poli `domain_name` a ďalšie metadáta, ako `date_added`, budú poskytovať informácie o dátume pridania do Blacklistu.

Whitelist

Whitelist bude zahrňovať benígne, známe domény považované za bezpečné. Kolekcia `whitelist` v databázovom systéme bude obsahovať dokumenty s informáciami o povolených doménach. Podobne ako pri Blackliste, pole `domain_name` a prípadne ďalšie metadáta budú slúžiť na identifikáciu povolených domén.

Integrácia s výstupnou webovou aplikáciou

Blacklist a Whitelist sú súčasťou výstupnej webovej aplikácie predstavenej v sekcii 5.7. Zoznamy obsahujú preddefinované hodnoty a užívateľ webovej aplikácie má možnosť manuálne pridávať nové záznamy do zoznamov DGA a benígnych domén, avšak automatické pridávanie nových záznamov nebude implementované z dôvodu možného zavedenia chyby v prípade falošných pozitív.

Pre efektívne ukladanie dát v databázovom systéme budú vytvorené vhodné indexy, napríklad na polia `domain_name`, aby sa zabezpečila rýchla i efektívna schopnosť vyhľadávania. Dáta v kolekciách budú štruktúrované tak, aby umožnili jednoduché a presné filtrovanie DGA domén.

Filtrácia domén zo sieťového analyzátora

Pri identifikácii domén zo sieťového analyzátora systém nasleduje tieto kroky a vykonáva príslušné opatrenia:

1. **Detekcia v Blackliste** – Systém okamžite kontroluje, či sa identifikovaná doména nachádza v Blackliste. V prípade potvrdenia prítomnosti v Blackliste je táto informácia zaznamenaná a následne zobrazená vo výstupnej webovej aplikácii. Týmto spôsobom užívateľ získava varovanie o prítomnosti potenciálne nebezpečnej domény.

2. **Overovanie vo Whiteliste** – V prípade, ak systém nájde zhodu vo Whiteliste, automaticky danú doménu ignoruje, pretože bola preddefinovaná ako bezpečná. Tento krok je dôležitý na elimináciu falošných poplachov, čo výrazne zvyšuje efektivitu celkovej analýzy.

3. **Žiadna zhoda** – Ak systém nenašiel zhodu ani v jednom zo zoznamov, pokračuje v ďalšej analýze popísanej v sekcii 5.5.3.

Systematická a spoľahlivá filtrácia domén prispieva k bezpečnosti i presnosti sieťovej analýzy a taktiež k rýchlosti spracovávania z dôvodu predfiltrovania známych benígnych a DGA domén.

5.5.3 Filtrácia domén pomocou príznaku NXDomain

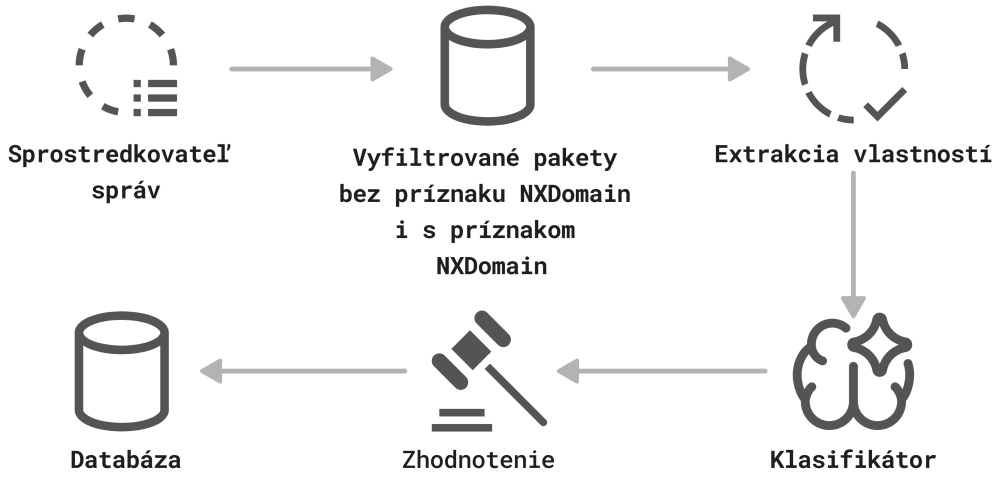
Príznak NXDomain [45] v DNS odpovediach predstavuje dôležitý ukazovateľ, signalizujúci neexistenciu konkrétnej domény. Identifikácia tohto príznaku poskytuje cenné informácie o úspešnosti DNS dotazov a umožňuje implementáciu efektívnych bezpečnostných opatrení a riadenia siete.

Filtrácia domén na základe príznaku NXDomain je realizovaná prostredníctvom podrobnej analýzy DNS hlavičky. Tento proces zahŕňa vyhodnocovanie hodnoty Response Code (RCODE) v DNS odpovedi. Ak sa hodnota RCODE rovná 3, signalizuje to prítomnosť NXDomain príznaku v danom DNS pakete. Táto metóda poskytuje spoľahlivú identifikáciu neexistujúcich domén a slúži ako základ pre ďalšie bezpečnostné a administratívne opatrenia v sieti.

Výstupom tejto filtračnej techniky sú dva typy paketov: pakety s príznakom NXDomain i tie, ktoré NXDomain príznak neobsahujú. Tieto výsledky slúžia ako jednotlivé vstupy do bloku č. 2, ktorý je podrobne opísaný v sekcii 5.6, pričom ich komunikáciu zabezpečuje sprostredkovateľ správ, ktorý bol predstavený v sekcii 5.4.

5.6 Extrahovanie vlastností a klasifikácia domén

Druhý blok detekčného nástroja, pomenovaný Processor, sa venuje extrakcii vlastností a získavaniu nových vlastností z paketov systému DNS, ich následnou binárnou klasifikáciou pomocou techník strojového učenia a následným uložením do databázového systému. Tento proces je vizualizovaný na obrázku 5.2, kde je detailne vyobrazený postup od extrakcie vlastností až po uloženie do databázy. Jednotlivé časti návrhu druhého bloku budú vysvetlené ďalej v texte.



Obr. 5.2: Konceptuálny návrh druhého bloku detekčného nástroja

Vstupom do druhého bloku detekčného nástroja sú rozdelené DNS pakety podľa príznaku NXDomain, tzn. podľa poľa RCODE v DNS pakete. Extrakcia vlastností z týchto paketov je popísaná v sekcii 5.6.1.

5.6.1 Návrh extrakcie vlastností

V tejto sekcii bude detailne predstavený návrh extrakcie vlastností, ktoré majú kľúčový význam pre implementáciu klasifikátorov a binárnu klasifikáciu doménových mien založených na DGA.

Definícia vlastností

Predmetom extrakcie bude doménové meno, tzn. cieľom extrakcie budú predovšetkým lexikálne vlastnosti doménového mena. Vlastnosti boli zvolené na základe dôkladného preskúmania odbornej literatúry a existujúcich prác v oblasti analýzy doménových mien na detekciu DGA [3, 61, 38, 82, 83]. Medzi vybrané vlastnosti patria:

1. **Dĺžka reťazca** – jedná sa o počet znakov reťazca.

   Platí pre:

   - FQDN[4]
   - TLD[5]
   - SLD[6]

2. **Počet numerických znakov v rámci reťazca**
3. **Pomer numerických znakov reťazca** – súčet numerických znakov reťazca vydelený celkovým počtom znakov.
4. **Pomer nealfanumerických znakov reťazca** – súčet nealfanumerických znakov reťazca vydelený celkovým počtom znakov.
5. **Počet spoluhlások reťazca**
6. **Pomer spoluhlások reťazca** – súčet spoluhlások v rámci reťazca vydelený celkovým počtom znakov.
7. **Počet samohlások reťazca**
8. **Pomer samohlások reťazca** – súčet samohlások v rámci reťazca vydelený celkovým počtom znakov.
9. **Počet hexadecimálnych znakov reťazca**
10. **Pomer hexadecimálnych znakov reťazca** – súčet hexadecimálnych znakov v rámci reťazca vydelený celkovým počtom znakov.
11. **Počet nealfanumerických znakov reťazca**
12. **Shannonova entropia reťazca** – je dôležitou metrikou v informačnej teórii. Určuje neusporiadanosť alebo náhodnosť reťazca [3]. Vzorec pre Shannonovu entropiu je možné vidieť v rovnici 5.1.

$$H(X) = -\sum_{i=1}^{n} P(x_i) \cdot \log_2(P(x_i)) \tag{5.1}$$

[4] Fully Qualified Domain Name
[5] Top Level Domain
[6] Second Level Domain

Kde:

- $H(X)$ je entropia náhodnej premennej X.
- $P(x_i)$ je pravdepodobnosť výskytu i-teho znaku zo všetkých možných znakov v texte.
- n je počet všetkých možných znakov v texte.

Vlastnosti č. 2 až 5 platia pre:

- SLD[6]
- Všetky subdomény

13. **Celkový počet kľúčových slov súvisiacich s phishingom**

    Platí pre:

    - FQDN[7]
    - SLD[6]

14. **Príznak numerického znaku v rámci reťazca** – reťazec obsahuje aspoň jeden numerický znak
15. **Príznak reťazca začínajúceho numerickým znakom**
16. **Najdlhšia sekvencia po sebe idúcich znakov v rámci reťazca**

    Vlastnosti č. 7 až 10 platia pre:

    - FQDN[8]

17. **Skóre zneužitia TLD podľa ScoutDNS** [1]
18. **Počet jedinečných znakov v kombinácii TLD[9] a SLD[6]**
19. **Počet subdomén**
20. **Najdlhšia sekvencia po sebe idúcich spoluhlások**
21. **Počet zhôd s DGA N-gramami** – N-gramy sú sekvencie zložené z n jednotiek, ktoré môžu mať podobu znakov, slov alebo častí slov tvorených n jednotkami. Počítanie n-gramov sa zvyčajne vykonáva posúvaním sa o jedno slovo vpred. Napríklad vo vete "Toto je veta", ak sa extrahuje n-gram s n=1, stane sa nasledovné sekvencie: ["toto", "je", "veta"], s n=2 by sa stali nasledujúce sekvencie: [toto je, je veta], a s n=3 by sa stali nasledujúce sekvencie: [toto je veta]. N-gramy s číslom n=1 sú označovanané ako unigramy, pre n=2 ako bigramy a pre n=3 ako trigramy [29]. Počet zhôd bude zahŕňať počítanie výskytov n-gramov domény v rámci zoznamu známych n-gramov DGA domén.

[7] Fully Qualified Domain Name
[8] Fully Qualified Domain Name
[9] Top Level Domain

22. **Jaccardov index** – využíva sa na určenie podobnosti medzi dvoma množinami [23]. Vzorec pre výpočet Jaccardovho indexu je možné vidieť v rovnici 5.2.

$$J(X,Y) = \frac{|X \cap Y|}{|X \cup Y|} \tag{5.2}$$

kde

- $J(X,Y)$ predstavuje podobnosť dvoch množín X a Y.
- X predstavuje prvú z porovnávaných množín
- Y predstavuje druhú z porovnávaných množín

Jaccardov index môže byť využitý na detekciu DGA doménových mien následovne:

1. Vytvoria sa dve množiny n-gramov. Jedna z nich bude obsahovať n-gramy vytvorené z benígnych doménových mien a druhá n-gramy pochádzajúce z DGA doménových mien.
2. Pre každé klasifikované doménové meno sa vytvorí množina n-gramov.
3. Vypočíta sa hodnota podobnosti s využitím Jaccardovho indexu medzi množinou n-gramov doménového mena určeného na klasifikáciu a množinou n-gramov vytvorených z benígnych doménových mien, respektíve n-gramov pochádzajúcich z DGA doménových mien.

Pre reálne využitie Jaccardovho indexu na detekciu DGA domén v reálnom čase je však klasický výpočet veľmi neefektívny. Z toho dôvodu je nutné využiť optimalizovanú verziu Jaccardovho indexu [63], ktorá sa zameriava na zvýšenie výpočtovej efektivity, pričom je definovaná v rovnici 5.3 a porovnanie jednotlivých verzií je možné vidieť v tabuľke 5.1.

$$J(X,Y) = \frac{|X \cap Y|}{|X|} \tag{5.3}$$

Verzia	**Počet domén**	**Rýchlosť spracovania [doménové meno/s]**
Klasická	34591	143.08
Optimalizovaná	34591	26608.46

Tabuľka 5.1: Porovnanie výkonnosti klasickej a optimalizovanej verzie Jaccardového indexu

Extrakcia bude vykonávaná oboma modulmi na extrakciu dát. Na základe vlastností získaných z domén budú navrhnuté vhodné klasifikátory na vykonávanie binárnej klasifikácie v sekcii 5.6.2.

5.6.2 Návrh klasifikátorov

Na binárnu klasifikáciu doménových mien založených na DGA je navrhnuté použitie niekoľkých klasifikačných metód. Budú využité metódy Logistická regresia, SVM[10], XGBoost, Random Forest, Decision Tree a taktiež Neurónová sieť. Tieto metódy majú rôzne charakteristiky a výhody, ktoré je dôležité zvážiť pri výbere najlepšej možnosti pre detekciu DGA domén.

- **Logistická regresia** – Štatistický model používaný hlavne na riešenie problémov binárnej klasifikácie, kde je cieľom predikovať pravdepodobnosť príslušnosti pozorovaní k jednej zo dvoch kategórií. Model pracuje na princípe transformácie lineárnej kombinácie vstupných premenných pomocou logistickej funkcie, čo zabezpečuje, že výstupné hodnoty sú v rozsahu medzi 0 a 1 a vyjadrujú pravdepodobnosti. Vďaka svojej flexibilite a interpretovateľnosti je logistická regresia často uprednostňovaná v oblastiach ako medicína, sociálne vedy alebo marketing. Hoci je výkonná pri modelovaní lineárnych vzťahov, môže sa potýkať s obmedzeniami pri komplexnejších vzťahoch alebo vysoko dimenzionálnych dátach bez vhodnej regularizácie [86].

- **SVM**[10] – Klasifikačný algoritmus, ktorý vytvára hyperrovinu alebo skupinu hyperrovín v priestore s cieľom oddeliť body do rôznych tried. Jeho schopnosť efektívne pracovať s veľkými priestormi vlastností a vysoká presnosť robia SVM vhodným pre problémy klasifikácie. Môže mať ale obmedzenú efektivitu pri veľkých dátových sadách a vyžaduje vhodný výber jadra a parametrov [18].

- **Rozhodovacie stromy** – Sú jednoduchými a ľahko interpretovateľnými klasifikátormi, ktoré rozdeľujú dáta do rôznych skupín podľa hierarchie príznakov. Ich schopnosť efektívne pracovať s kategorickými aj numerickými dátami ich robí vhodnými pre tento účel. Avšak, môžu mať problémy s overením generalizovateľnosti na nové dáta, najmä pri jednoduchých modeloch [28].

- **Random Forest** – Klasifikátor založený na rozhodovacích stromoch, ktorý kombinuje viacero stromov a využíva techniku agregácie pre dosiahnutie vysokej presnosti. Je odolný voči pretrénovaniu, môže efektívne pracovať s veľkým počtom príznakov a poskytuje informácie o dôležitosti príznakov. Môže mať ale tendenciu k vyššej výpočtovej náročnosti pri veľkých dátových sadách [12].

- **AdaBoost** – Metóda strojového učenia, ktorá vylepšuje slabšie klasifikátory ich kombináciou s cieľom vytvoriť silnejší klasifikátor. Algoritmus dynamicky upravuje váhy tréningových vzoriek, čím sa zameriava na ťažšie prípady a zvyšuje celkovú presnosť klasifikácie. Vytvorený model potom funguje na princípe váženého hlasovania viacerých klasifikátorov, čo vedie k robustnejšiemu a presnejšiemu rozhodovaniu [78].

- **XGBoost** – Extreme Gradient Boosting je výkonný algoritmus pre učenie sady rozhodovacích stromov, ktorý sa špecializuje na vysokú presnosť a efektívnosť. Jeho schopnosť manipulovať s rôznymi typmi dát a automatické ladenie hyperparametrov robí XGBoost veľmi výhodným pre tento účel, ale môže mať tendenciu k pretrénovaniu na trénovacích dátach, čo si vyžaduje primeranú reguláciu [16].

- **LightGBM** – Efektívna implementácia algoritmu gradientne zosilnených stromov, ktorá bola navrhnutá na spracovanie veľkého počtu dát a vlastností. Využíva techniky ako Gradient-based One-Side Sampling (GOSS) a Exclusive Feature Bundling (EFB) na zvýšenie efektívnosti a presnosti. GOSS zlepšuje výpočty tým, že udržiava dáta s veľkými gradientmi a náhodne zahadzuje tie s malými. EFB znižuje počet funkcií zoskupením tých, ktoré sú vzájomne exkluzívne. Vďaka týmto inováciám dokáže LightGBM výrazne urýchliť tréningový proces a zachovať presnosť modelu [34].

[10] Support Vector Machine

- **Neurónová sieť** – Jedná sa o modely inšpirované fungovaním ľudského mozgu, používané na riešenie problémov, ktoré vyžadujú schopnosť učenia sa a rozpoznávania vzorcov. Skladajú sa z vrstiev umelých neurónov, kde každý neurón prijíma vstupy, spracuje ich pomocou funkcie a generuje výstup. Vzťahy medzi neurónmi sú definované váhami, ktoré sa upravujú počas učenia siete. Proces učenia obvykle zahŕňa nastavenie týchto váh na základe trénovacích dát, aby sieť mohla presne predpovedať alebo klasifikovať nové dáta. Neurónové siete sa používajú v mnohých oblastiach, vrátane rozpoznávania obrazu a spracovania prirodzeného jazyka [25].

Pri výbere najlepšieho klasifikátora pre detekciu DGA doménových mien je dôležité zvážiť ich schopnosť efektívne rozlíšiť medzi DGA a benígnymi doménami, rýchlosť trénovania a predikcie, ako aj výpočtovú náročnosť. Experimentovanie s týmito metódami na dátových sadách obsahujúcich DGA aj benígne domény (ako je popísané v sekcii 5.2) umožní vyhodnotiť ich výkonnosť a vybrať najlepšiu možnosť pre použitie v detekčnom systéme.

5.7 Webová aplikácia

V tejto sekcii bude detailne popísaný návrh webovej aplikácie slúžiacej na zobrazenie výsledných informácií o klasifikovaných doménach a čase ich klasifikácie. Bude predstavená jej architektúra, účel a taktiež hlavné funkcionality.

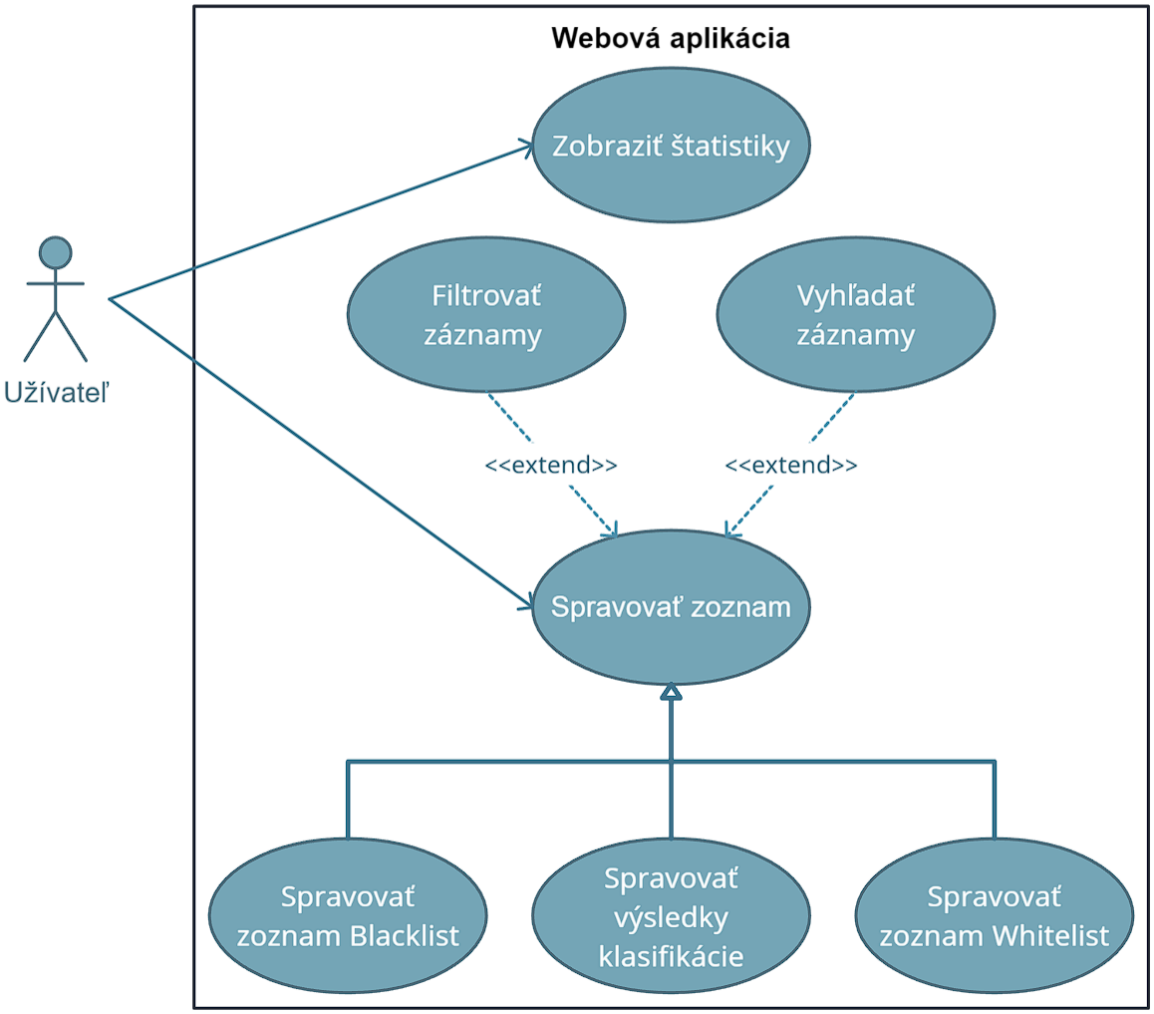
5.7.1 Účel a funkcionality

Účelom webovej aplikácie je poskytnúť užívateľom prehľadný a efektívny spôsob, ako zobraziť a spravovať informácie o klasifikovaných doménach a čase ich klasifikácie. Aplikácia umožňuje nielen pasívne prezeranie dát, ale aj aktívnu interakciu s nimi prostredníctvom možnosti pridávania, upravovania alebo vymazávania z výsledkov detekcie, Whitelistu alebo Blacklistu. To užívateľom poskytuje nástroje pre preddefinovanie domén, ktoré majú byť výslovne povolené alebo zakázané, čo zabezpečuje lepšiu kontrolu nad bezpečnosťou a integritou webového obsahu. Kľúčové funkcionality aplikácie zahŕňajú:

- **Zobrazenie výsledkov klasifikácie** – Poskytuje prehľadné rozhranie, kde užívatelia môžu vidieť aktuálne výsledky klasifikácie domén, vrátane času ich detekcie. Toto umožňuje rýchle identifikovanie potenciálne škodlivých alebo podozrivých domén.
- **Správa Whitelistu, Blacklistu a výsledkov klasifikácie** – Poskytuje komplexnú funkcionalitu pre efektívne riadenie prístupu k internetovým zdrojom, umožňujúc užívateľom nielen pridávať, odstraňovať a upravovať záznamy v zoznamoch povolených (Whitelist) a zakázaných (Blacklist) domén, ale tiež spravovať výsledky klasifikácie s rovnakou úrovňou flexibility. Táto rozšírená správa umožňuje užívateľom efektívne preddefinovať, ktoré domény sú bezpečné, ktoré by mali byť blokované, a zároveň ponúka možnosť reagovať na výsledky klasifikácie, tzn. môžu záznamy v prípade potreby pridávať do Whitelistu alebo Blacklistu, odstraňovať ich alebo upravovať.
- **Filtrácia a vyhľadávanie** – Poskytuje možnosti filtrácie a vyhľadávania v rámci výsledkov detekcie, ako aj v Whiteliste a Blackliste. Umožňuje to užívateľom efektívne nájsť a spravovať konkrétne domény podľa názvu alebo času ich pridania/zachytenia.
- **Zobrazenie štatistík** – Užívateľ si bude môcť zobraziť štatistiky, ako je počet skontrolovaných domén z dnešného dňa, celkový počet pozitívnych výsledkov z dnešného

dňa, počet odfiltrovaných domén s využitím zoznamu Blacklist a celkový počet položiek v zoznamoch Blacklist a Whitelist.

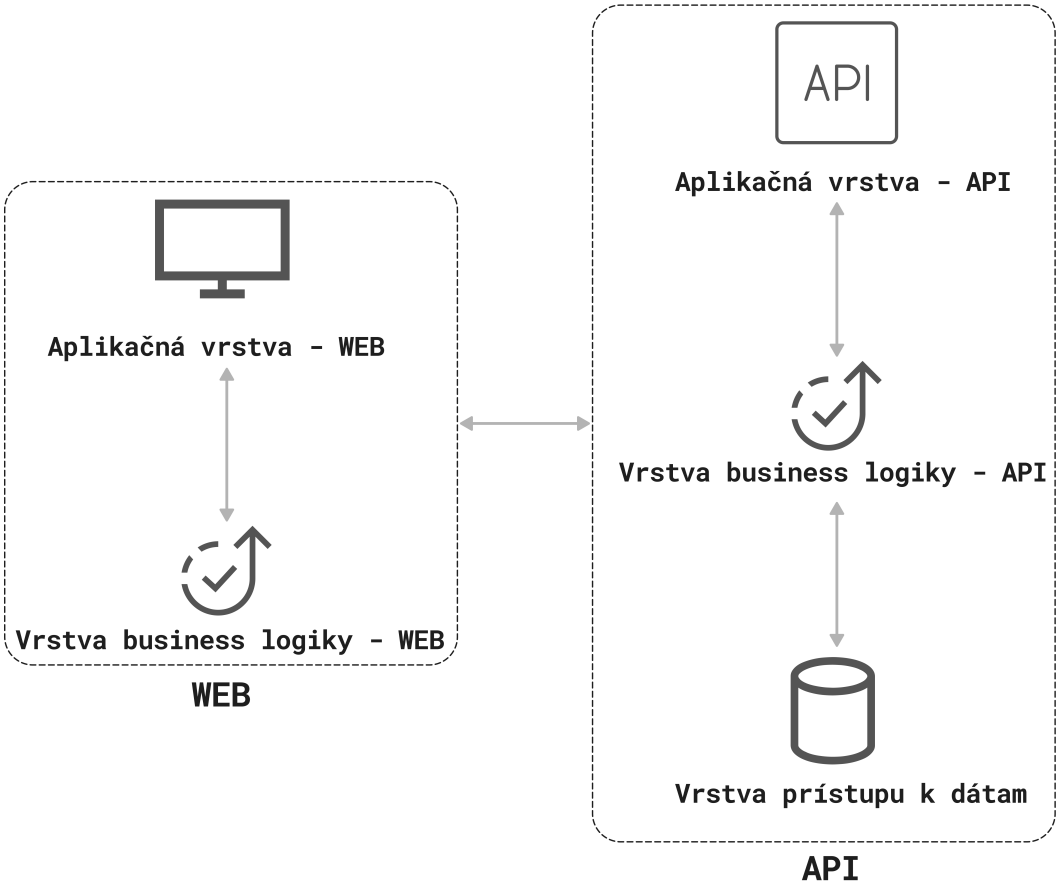
Pre lepšie pochopenie interakcie medzi užívateľmi a webovou aplikáciou bol vytvorený diagram prípadov použitia, viď obrázok 5.3. Tento diagram ilustruje, ako sú jednotlivé funkcionality aplikácie prístupné užívateľom a ako medzi sebou súvisia.



Obr. 5.3: Diagram prípadov použitia webovej aplikácie

5.7.2 Architektúra

Architektúra webovej aplikácie, znázornená na obrázku 5.4, sa skladá z dvoch hlavných častí: webového rozhrania (WEB) a aplikačného programovacieho rozhrania (API).



Obr. 5.4: Architektúra webovej aplikácie

Aplikačné programovacie rozhranie

Aplikačné programovacie rozhranie (API) predstavuje serverovú časť aplikácie, ktorá spracováva požiadavky od klientov a komunikuje s databázou. Táto časť je zodpovedná za logiku aplikácie a správu dát. Bude sa skladať z viacerých vrstiev, medzi ktoré patria:

- **Vrstva prístupu k dátam (DAL - Data Access Layer)** – Je zodpovedná za interakciu so zdrojmi dát, tzn. databázovými servermi. Umožňuje aplikácii čítať a zapisovať dáta bez potreby priamej interakcie s konkrétnymi databázovými technológiami. Vďaka tomu je možné ľahko meniť zdroje dát alebo technológie použité pre ukladanie dát bez vplyvu na zvyšok aplikácie.
- **Vrstva business logiky (BL - Business Logic)** – Obsahuje všetku podnikovú logiku aplikácie. Zodpovedá za spracovanie dát prijatých z webového rozhrania, vykonávanie potrebných operácií s týmito dátami (tzn. validácia a transformácia dát) a následné poskytnutie výsledkov späť webovému rozhraniu alebo uloženie výsledkov do zdroja dát.
- **Aplikačná vrstva (APP)** – Poskytuje definované rozhranie pre komunikáciu medzi webovým rozhraním a aplikačnou logikou. Táto vrstva využíva REST[11] architektúru,

[11] Representational State Transfer

ktorá je štandardom pre vytváranie webových služieb. REST využíva metódy protokolu HTTP, ako sú GET, POST, PUT, DELETE apod. na manipuláciu s reprezentáciami zdrojov, čo poskytuje jednoduché a intuitívne rozhranie pre vykonávanie operácií, ako je získavanie zoznamu dát, aktualizácia údajov, alebo vytvorenie nových záznamov. Kľúčovým rozširujúcim aspektom API je jeho dizajn podľa OpenAPI[12] špecifikácie, ktorá predstavuje priemyselný štandard pre definovanie rozhraní. OpenAPI slúži na podrobný a presný popis celej štruktúry API, vrátane dostupných operácií, parametrov, odpovedí a možných chýb, čím značne uľahčuje automatizáciu testovania, generovanie dokumentácie a integráciu s inými službami. Tento prístup zabezpečuje vysoký stupeň interoperability, zjednodušuje vývoj a integráciu a zvyšuje celkovú efektívnosť a bezpečnosť API.

Na základe definovaných vrstiev aplikačného programovacieho rozhrania je potrebné špecifikovať jednotlivé kontroléry, kde každý kontrolér spracováva prichádzajúce HTTP požiadavky na konkrétnych adresách URL, tzn. na konkrétnych koncových bodoch a riadi tok dát medzi užívateľským rozhraním a vrstvou business logiky. Kontroléry zodpovedajú za prijímanie požiadaviek, volanie príslušných služieb alebo metód vrstvy business logiky a následné odoslanie odpovedí klientovi. Jednotlivé kontroléry, ich koncové body a taktiež popis ich funkcionality je predstavený nižšie.

Result Kontrolér

Kontrolér `Result` sa zaoberá spracovaním dát týkajúcich sa výsledkov detekcie doménových mien. Poskytuje koncové body špecifikované v tabuľke 5.2.

Metóda	Koncový bod	Popis
GET	/Result	Získa zoznam všetkých výsledkov detekcie.
POST	/Result	Zaznamená nový výsledok detekcie.
PATCH	/Result	Aktualizuje výsledok detekcie.
GET	/Result/Id	Získa konkrétny výsledok detekcie podľa ID.
DELETE	/Result/Id	Odstráni konkrétny výsledok detekcie podľa ID.
GET	/Result/Max/Page/Filter	Získa stránkovaný zoznam výsledkov detekcie filtrovaný podľa dotazu.
GET	/Result/Count	Spočíta všetky výsledky detekcie.
GET	/Result/FilteredByBlacklist	Spočíta položky filtrované čiernou listinou.
GET	/Result/NumberOfDomainsToday	Spočíta domény spracované dnes.
GET	/Result/PositiveResultsToday	Spočíta pozitívne výsledky detekcie získané z dnešného dňa.

Tabuľka 5.2: Špecifikácie koncových bodov pre kontrolér Result

[12]OpenAPI: https://www.openapis.org/

Blacklist Kontrolér

Kontrolér `Blacklist` slúži na spracovanie dát týkajúcich sa zoznamu Blacklist. Definuje koncové body špecifikované v tabuľke 5.3.

Metóda	Koncový bod	Popis
GET	/Blacklist	Získa zoznam položiek na čiernej listine.
POST	/Blacklist	Pridá položku na čiernu listinu.
PATCH	/Blacklist	Aktualizuje položku na čiernej listine.
GET	/Blacklist/Id	Získa konkrétnu položku na čiernej listine podľa ID.
DELETE	/Blacklist/Id	Odstráni konkrétnu položku z čiernej listiny podľa ID.
POST	/Blacklist/MoveResultToBlacklist	Presunie výsledok detekcie na čiernu listinu.
GET	/Blacklist/Max/Page/Filter	Získa stránkovaný zoznam položiek na čiernej listine filtrovaný podľa dotazu.
GET	/Blacklist/Count	Spočíta všetky položky na čiernej listine.

Tabuľka 5.3: Špecifikácie koncových bodov pre kontrolér Blacklist

Whitelist Kontrolér

Kontrolér `Whitelist` sa zaoberá spracovaním dát týkajúcich sa zoznamu Whitelist. Poskytuje koncové body špecifikované v tabuľke 5.4.

Metóda	Koncový bod	Popis
GET	/Whitelist	Získa zoznam položiek na bielom zozname.
POST	/Whitelist	Pridá položku na biely zoznam.
PATCH	/Whitelist	Aktualizuje položku na bielom zozname.
GET	/Whitelist/Id	Získa konkrétnu položku na bielom zozname podľa ID.
DELETE	/Whitelist/Id	Odstráni konkrétnu položku z bieleho zoznamu podľa ID.
POST	/Whitelist/MoveResultToWhitelist	Presunie výsledok detekcie na biely zoznam.
GET	/Whitelist/Max/Page/Filter	Získa stránkovaný zoznam položiek na bielom zozname filtrovaný podľa dotazu.
GET	/Whitelist/Count	Spočíta všetky položky na zozname povolení.

Tabuľka 5.4: Špecifikácie koncových bodov pre kontrolér Whitelist

Definícia modelov

Pre správnosť implementácie aplikačného programovacieho rozhrania je nevyhnutné definovať štruktúru dátových modelov, ktoré reprezentujú informácie manipulované v jednotlivých častiach systému. Tieto modely poskytujú abstraktnú reprezentáciu dát a sú základným kameňom pre spracovanie a uchovávanie informácií v databáze. V nasledujúcich tabuľkách sú definované modely pre záznamy na čiernom a bielom zozname, ako aj modely pre výsledky detekcie.

Tabuľka 5.5 predstavuje definíciu modelu `Blacklist`. Obsahuje všetky potrebné údaje, ktoré je nutné vedieť o doméne, ktorá by mala byť prednostne klasifikovaná ako nebezpečná, tzn. doménové meno, čas pridania a jedinečný identifikátor.

Pole	**Popis**
DomainName	Meno domény
Added	Dátum pridania do zoznamu Blacklist
Id	Jedinečný identifikátor záznamu

Tabuľka 5.5: Definícia modelu `Blacklist`.

V tabuľke 5.6 je zobrazená definícia modelu `Result`, predstavujúca výsledok detekcie. Model obsahuje doménové meno detegovanej domény, čas detekcie, príznak detekcie danej domény s využitím zoznamu Blacklist a taktiež percentuálnu hodnotu nebezpečnosti, indikátor nebezpečnosti domény i jedinečný identifikátor daného záznamu.

Pole	**Popis**
DomainName	Meno domény
Detected	Dátum a čas detekcie domény
DidBlacklistHit	Indikátor detekcie na čiernom zozname
DangerousProbabilityValue	Číselná hodnota pravdepodobnosti nebezpečenstva
DangerousBoolValue	Indikátor nebezpečenstva domény
Id	Jedinečný identifikátor výsledku

Tabuľka 5.6: Definícia modelu `Result`.

Tabuľka 5.5 predstavuje definíciu modelu `Whitelist`. Obsahuje údaje, ktoré je nutné vedieť o doméne, ktorá by mala byť prednostne klasifikovaná ako bezpečná, tzn. doménové meno, čas pridania a jedinečný identifikátor

Pole	**Popis**
DomainName	Meno domény
Added	Dátum pridania do zoznamu Whitelist
Id	Jedinečný identifikátor záznamu

Tabuľka 5.7: Definícia modelu `WhitelistModel`.

Tieto modely slúžia pre spracovanie a uchovávanie dát a zabezpečujú integritu a dostupnosť informácií pre aplikačné programovacie rozhranie.

Webové rozhranie

Webové rozhranie (WEB) predstavuje časť aplikácie, ktorá je zodpovedná za interakciu s užívateľmi. Bude využívať štandardné webové technológie a na poskytnutie dynamických a reaktívnych užívateľských rozhraní bude využitá vhodná knižnica, ktorá podporuje vytváranie komponentov a umožňuje efektívnu aktualizáciu a správu stavu aplikácie.

Dizajn užívateľského rozhrania bude využívať moderné typografické a vizuálne štandardy, ktoré budú zdôrazňovať čistotu a prehľadnosť. Použitie konzistentných farieb, písiem a rozloženia naprieč celou aplikáciou minimalizuje kognitívnu námahu používateľa a zvýši celkový estetický dojem z aplikácie. Vďaka tejto koherentnosti používatelia rýchlo nájdu požadované funkcie a budú sa môcť intuitívne navigovať aplikáciou.

Rozhranie je navrhnuté tak, aby bolo responzívne, tzn. prispôsobí sa rôznym veľkostiam obrazoviek.

Interaktívne prvky, ako sú tlačidlá, formuláre a navigačné komponenty, sú navrhnuté tak, aby boli intuitívne a ľahko prístupné. Na zvýšenie používateľskej interakcie aplikácia využíva vizuálne efekty, ako sú animácie a dynamické zmeny vizuálneho stavu komponentov pri interakcii. Tieto vizuálne spätné väzby zabezpečujú, že používatelia dostanú okamžité potvrdenie o svojich akciách. Takéto prvky zlepšujú celkový zážitok z používania aplikácie a prispievajú k jej intuitívnosti a plynulosti používania.

Navrhnuté užívateľské rozhranie aplikácie, ktorého vzhľad je postavený na šablóne Cyber –Ez - Cyber Security [13], sa skladá zo štyroch hlavných webstránok, ktorých funkcionalita je popísaná nižšie.

Domovská stránka

Na domovskej stránke zobrazenej na obrázku 5.5 je poskytnutý rýchly prehľad o najdôležitejších metrikách a štatistikách.

Obr. 5.5: Návrh domovskej stránky webovej aplikácie

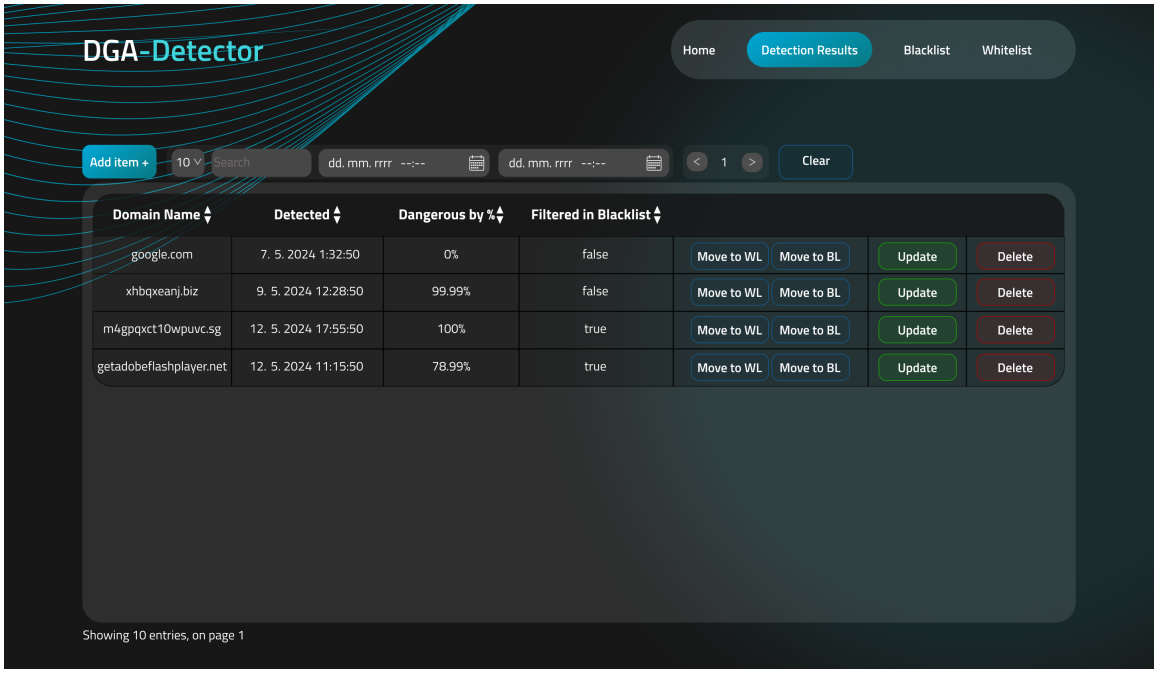
Užívateľom je zobrazený počet domén, ktoré boli klasifikované v daný deň, počet pozitívnych výsledkov v daný deň, počet doménových mien zachytených zoznamom Blacklist a celkový počet doménových mien v rámci zoznamov Blacklist a Whitelist. Domovská obrazovka taktiež slúži ako centrálny bod pre prístup k ostatným funkcionalitám aplikácie, ku ktorým je možné sa dostať po kliknutí na navigačný panel, nachádzajúci sa v pravom hornom rohu.

Výsledky detekcie

Stránka výsledkov detekcie, viď 5.6, poskytuje komplexný nástroj umožňujúci užívateľom prehľadávať, analyzovať a riadiť informácie o doménach, ktoré boli klasifikované. Táto stránka zobrazuje podrobný zoznam detegovaných domén, ktorý zahŕňa dátum a čas ich detekcie, špecifikáciu dôvodu ich klasifikácie a percentuálny ukazovateľ rizika alebo nebezpečnosti každej domény. Informácie sú prezentované v tabuľke, ktorá umožňuje užívateľom zoradiť údaje podľa rôznych kritérií, vrátane doménového mena, času detekcie, dôvodu klasifikácie a stupňa nebezpečnosti.

Na zvýšenie užívateľskej efektivity poskytuje stránka možnosti pre hlbšie filtrovanie výsledkov. Užívatelia môžu uplatniť filtre na vyhľadávanie konkrétnych domén alebo obmedziť výsledky podľa dátumu detekcie, čo umožňuje efektívnejšie spravovanie výsledkov detekcie. Každá položka v zozname ponúka možnosti pre rýchle akcie, vrátane pridania do zoznamu Whitelist pre bezpečné domény alebo Blacklist pre blokované domény, ako aj možnosti pre aktualizáciu alebo odstránenie položiek z evidencie.

Stránka tiež podporuje prispôsobenie zobrazenia, pričom užívatelia môžu vybrať počet zobrazených položiek na stránku z prednastavených možností 10, 20, 50 alebo až 100 záznamov na stránku.



Obr. 5.6: Návrh stránky určenej pre vizualizáciu výsledkov detekcie

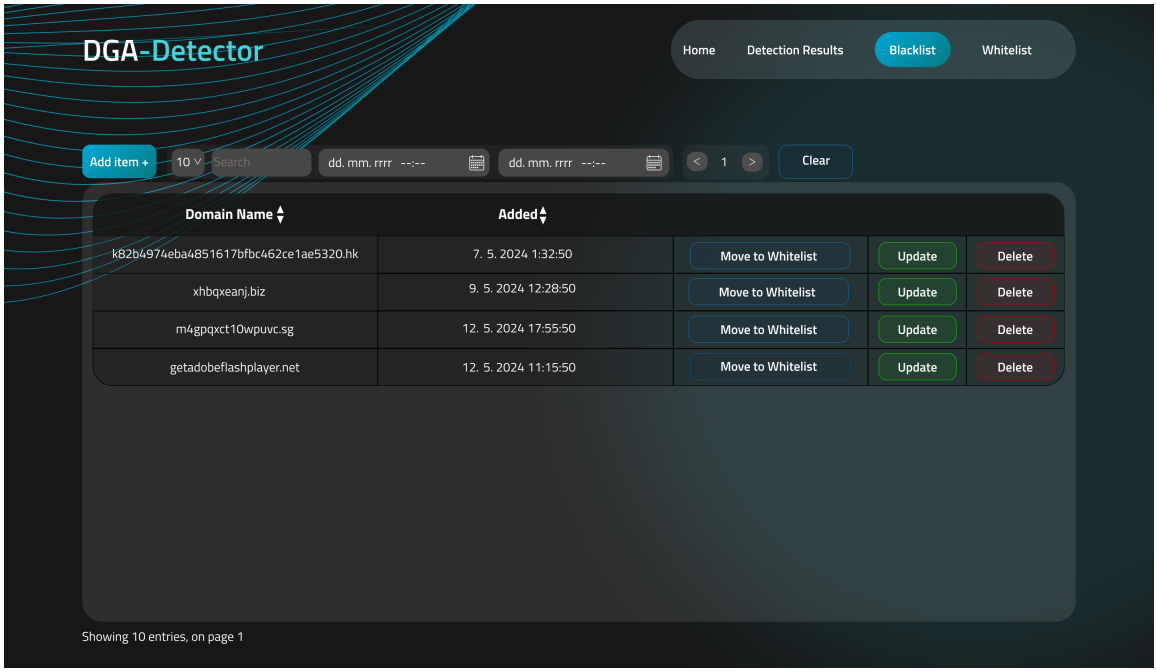
Zoznamy Blacklist a Whitelist

Stránka, zobrazená na obrázku 5.7, slúži na správu zoznamov Blacklist a Whitelist. Blacklist obsahuje domény, ktoré majú byť blokované, zatiaľ čo Whitelist obsahuje domény, ktoré sú povolené v rámci aplikácie. Na obrázku je zobrazená iba stránka určená pre zoznam Blacklist, ale rovnaká stránka slúži taktiež na správu zoznamu Whitelist a z toho dôvodu tu budú uvedené spoločne.

Užívatelia majú možnosť efektívne vyhľadávať a filtrovať záznamy v týchto zoznamoch. Sú k dispozícii filtre pre vyhľadávanie konkrétnych domén alebo obmedzenie zobrazených záznamov podľa dátumu pridania, čo umožňuje rýchlu a presnú správu zoznamov.

Každý záznam v zozname ponúka možnosti pre rýchle akcie, ako je aktualizácia, odstránenie záznamu alebo presunutie medzi jednotlivými zoznamami Blacklist a Whitelist podľa potreby.

Zobrazenie zoznamov Blacklist a Whitelist môže byť prispôsobené podľa preferencií užívateľov, tzn. môžu si nastaviť počet zobrazených záznamov na stránku z prednastavených možností na 10, 20, 50 alebo až 100 záznamov na stránku.



Obr. 5.7: Návrh stránky určenej pre vizualizáciu zoznamov Blacklist a Whitelist

Kapitola 6

Implementácia

Pre úspešnú implementáciu detekčného systému bolo nevyhnutné vykonať dôkladný návrh z hľadiska rozšíriteľnosti i efektívnosti. Tento proces bol detailne popísaný a realizovaný v kapitole 5. Využitím týchto aspektov bolo zabezpečené, že výsledný systém nielenže spĺňa požiadavky na funkčnosť a efektivitu, ale je tiež pripravený na budúce rozšírenia a aktualizácie.

Na základe kvalitného návrhu bude v tejto kapitole popísaná implementácia celého detekčného systému. V sekcii 6.1 budú popísané vhodné programovacie jazyky s ohľadom na požiadavky systému, ktoré boli definované v sekcii 5.1. Následne bude popísaný výber efektívnych technológií pre účely ukladania dát v sekcii 6.2 a komunikácie medzi blokmi v sekcii 6.3.

V ďalšom priebehu kapitoly je pozornosť sústredená na jednotlivé komponenty systému. Sekcia 6.4 sa zaoberá implementáciou prvého bloku detekčného nástroja, nazývaného Detektor, sekcia 6.5 sa venuje implementácii druhého bloku detekčného nástroja, nazývaného Processor a sekcia 6.6 sa zameriava na implementáciu webovej aplikácie.

6.1 Implementačné programovacie jazyky

V rámci implementácie bolo zvažovaných viacero programovacích jazykov medzi ktoré patrili jazyky ako C, C++, Rust, Go a Zig, ktoré sú známe pre svoju vysokú výkonnosť, C# preferovaný pre multiplatformný vývoj, a vďaka ASP.NET[1] je obzvlášť vhodný pre vytváranie robustných API. Nachádza široké využitie aj v oblastiach umelej inteligencie a strojovom učení, kde ale výrazne prevažuje jazyk Python napriek tomu, že sa nejedná o najrýchlejší jazyk z dôvodu jeho interpretovanej povahy.

Pre kritickú časť detekčného systému bola zvolená implementácia v jazyku C++, ktorý je známy svojou vysokou výkonnosťou a schopnosťou efektívne pracovať s nízkoúrovňovými operáciami, čo je nevyhnutné pri vysoko náročných úlohách a čo taktiež umožňuje splnenie definovaných požiadaviek systému, ktoré boli definované v sekcii 5.1. C++ poskytuje presnú kontrolu pamäte a optimalizáciu kódu pomocou rôznych techník, ako sú inline optimalizácie, použitie optimalizačných úrovní ako `-O2` alebo `-O3` pri kompilácii, čo výrazne zlepšuje výkonnosť a efektivitu implementácie. Tieto optimalizačné techniky zabezpečujú dosiahnutie rýchlych a spoľahlivých výsledkov pri spracovávaní vysokých objemov dát v reálnom čase.

[1] ASP.NET: https://dotnet.microsoft.com/en-us/apps/aspnet

Python sa stal de facto jazykom pre oblasť umelej inteligencie a strojového učenia a z toho dôvodu bol zvolený pre implementáciu časti týkajúcej sa týchto oblastí. Jeho jednoduchosť a bohatý ekosystém knižníc ako TensorFlow[2], PyTorch[3], a Scikit-learn[4] umožňujú vývojárom rýchlo a efektívne vytvárať a testovať modely. I keď nepatrí medzi jazyky s vysokým výkonom a je pomalší v porovnaní s niektorými inými ako C++ či C#, jeho flexibilita, čitateľnosť kódu a schopnosť rýchlo prototypovať nové nápady ho robia ideálnou voľbou pre projekty zamerané na umelej inteligencii, kde rýchlosť vývoja a experimentovanie sú kľúčové.

Pre implementáciu API, ktoré je špecificky navrhnuté pre potreby webovej aplikácie opísanej v sekcii 5.7, bol zvolený programovací jazyk C#. Tento výber je motivovaný popularitou C# v oblasti vývoja robustných webových služieb a API, hlavne vďaka jeho úzkej integrácii s ASP.NET. Platforma ASP.NET je známa svojimi rozsiahlymi možnosťami pre vývoj webových aplikácií, ponúkajúc vývojárom súbor nástrojov na efektívne spracovanie požiadaviek a zabezpečenú komunikáciu medzi serverom a klientom. S kombináciou C# a ASP.NET môžeme vytvoriť silné a bezpečné API, ktoré poskytne pevný základ pre funkčnosť webovej aplikácie, zjednoduší interakciu s dátami a zabezpečí spoľahlivé služby pre užívateľov.

Webová aplikácia, ktorá využíva toto API, je podrobne opísaná v sekcii 5.7 a bola pre ňu zvolená technológia React[5]. React je populárna knižnica v jazyku Javascript pre tvorbu dynamických a interaktívnych užívateľských rozhraní. Jeho využitie je široké vďaka schopnosti poskytnúť rýchlu odozvu a vysoký výkon pre používateľské rozhranie. React je obzvlášť vhodný pre projekty, kde je kladený dôraz na efektívne spravovanie dát a stavov aplikácie, čo vedie k plynulej a príjemnej skúsenosti pre užívateľov. Integrácia Reactu s technológiami C# a ASP.NET API zabezpečuje, že celý systém funguje koherentne a efektívne pri poskytovaní služieb užívateľom.

6.2 Ukladanie dát

Pre efektívne ukladanie dát bol zvolený dokumentovo orientovaný databázový systém MongoDB[6] a taktiež relačné databázové systémy MySQL[7] alebo SQLite[8].

MongoDB

MongoDB je NoSQL databáza, ktorá využíva formát BSON (binárne zakódovaný JSON[9]) pre ukladanie dát. Vyniká svojou schopnosťou pracovať s neštruktúrovanými a rozsiahlymi dátami, čo je kľúčové pre úspešné ukladanie a získavanie informácií [46, 47].

Na komunikáciu s MongoDB slúžia oficiálne ovládače pre konkrétne jazyky a platformy, ktoré umožňujú jednoduché pripojenie k MongoDB serveru, manipuláciu s databázami, kolekciami a vykonávanie dotazov. Jedná sa konkrétne o:

- **PyMongo**[10] – MongoDB ovládač pre jazyk Python.

[2] TensorFlow: https://www.tensorflow.org/
[3] PyTorch: https://pytorch.org/
[4] Scikit-learn: https://scikit-learn.org/
[5] React: https://react.dev/
[6] MongoDB: https://www.mongodb.com/
[7] MySQL: https://www.mysql.com/
[8] SQLite: https://www.sqlite.org/
[9] JSON: https://www.json.org/
[10] PyMongo: https://pymongo.readthedocs.io/

- **MongoDB C++ Driver**[11] – MongoDB ovládač pre jazyk C++.
- **MongoDB C# Driver**[12] – MongoDB ovládač pre jazyk C#.

MySQL

MySQL je široko používaný open-source relačný databázový systém. Jeho hlavné vlastnosti zahŕňajú vysokú rýchlosť, flexibilitu a spoľahlivosť pri spracovaní veľkých objemov dát. MySQL podporuje širokú škálu aplikácií, od webových stránok až po databázové aplikácie pre veľké podniky. Umožňuje efektívnu prácu s dátami pomocou SQL[13], čo zjednodušuje tvorbu, manipuláciu a získavanie dát [52].

SQLite

SQLite je kompaktná, samostatne fungujúca databázová knižnica, ktorá poskytuje kompletný relačný databázový systém bez nutnosti externého servera. Táto knižnica sa vyznačuje jednoduchosťou použitia, vysokou portabilitou, minimálnymi požiadavkami na systémové zdroje a efektívnym výkonom [70]. Vďaka týmto vlastnostiam je SQLite ideálnym riešením pre aplikácie vyžadujúce spoľahlivú databázovú funkčnosť s obmedzenými systémovými požiadavkami, ako sú mobilné aplikácie, menšie až stredne veľké webové projekty a zabudované systémy.

Entity Framework Core

Pre prácu s relačnými databázovými systémami MySQL a SQLite bude využitý Entity Framework Core (EF Core). EF Core je moderný objektovo-relačný mapovací framework (ORM) pre .NET, ktorý poskytuje vývojárom možnosť pracovať s databázou pomocou .NET objektov, čím sa znižuje potreba písať komplikovaný SQL kód. Tento framework poskytuje podporu pre LINQ[14] dotazy, sledovanie zmien, migrácie a seedovanie databázy, čo uľahčuje a zefektívňuje vývoj databázových aplikácií. Využitím EF Core s MySQL a SQLite sa zjednodušuje integrácia a manipulácia s dátami v aplikáciách, čo prispieva k vyššej produktivite vývoja a udržateľnosti kódu [42].

6.3 Komunikácia medzi blokmi

Pre zabezpečenie efektívnej komunikácie medzi prvým a druhým blokom detekčného nástroja, ktorých návrh bol popísaný v sekciách 5.5 a 5.6, bol zvolený sprostredkovateľ správ RabbitMQ[15]. Voľba bola motivovaná splnením všetkých kritických požiadavkov špecifikovaných v sekcii 5.4, pričom ich splnenie bolo potrebné pre spoľahlivé fungovanie celého detekčného nástroja.

RabbitMQ

RabbitMQ je jednou z najúspešnejších implementácií sprostredkovateľa správ (message broker), ktorý umožňuje efektívnu distribúciu a riadenie správ a úloh medzi rôznymi kom-

[11] MongoDB C++ Driver: https://mongocxx.org/
[12] MongoDB C# Driver: https://www.mongodb.com/docs/drivers/csharp/current/
[13] Structured Query Language
[14] Language Integrated Query
[15] RabbitMQ: https://www.rabbitmq.com/

ponentmi v distribuovaných systémoch. RabbitMQ implementuje protokol AMQP[16] a je navrhnutý s dôrazom na vysokú dostupnosť, odolnosť a škálovateľnosť, poskytuje detailnú konfiguráciu a ponúka pokročilé funkcie ako smerovanie správ, pracovné fronty, publikovanie/predplatné a trvalosť správ. Taktiež poskytuje nástroje na monitorovanie a správu servera, čo uľahčuje správu a diagnostiku v produkčných prostrediach [74].

6.4 Implementácia prvého bloku detekčného nástroja

Prvý blok detekčného nástroja, popísaný v sekcii 5.5, sa zaoberá zbieraním a kategorizáciou neznámych domén systému DNS. V tejto sekcii bude bližšie popísaná jeho implementácia. Jazykom implementácie pre tento blok sa stal jazyk C++, čo už bolo spomenuté v sekcii 6.1.

6.4.1 Využité knižnice

V rámci implementácie prvého bloku bolo využitých mnoho knižníc, ktoré boli nainštalované a spravované pomocou nástroja vcpkg[17].

Vcpkg je správca balíčkov pre projekty v jazykoch C a C++, ktorý zabezpečuje jednoduché pridávanie a spravovanie knižníc v projektoch. Umožňuje užívateľom inštalovať, aktualizovať a odstraňovať knižnice na jednom mieste, čím zjednodušuje správu závislostí v projekte. Podporuje množstvo platforiem vrátane Windows, Linux a macOS.

Knižnice boli vybrané na základe ich špecifických funkcií a prínosov pre projekt, čím zabezpečujú efektívnu implementáciu požadovaných funkčností a zvyšujú celkovú kvalitu a výkon aplikácie. Použitie jednotlivých knižníc bude špecifikované v sekcii 6.4.3, pričom sa jedná o nasledujúce knižnice:

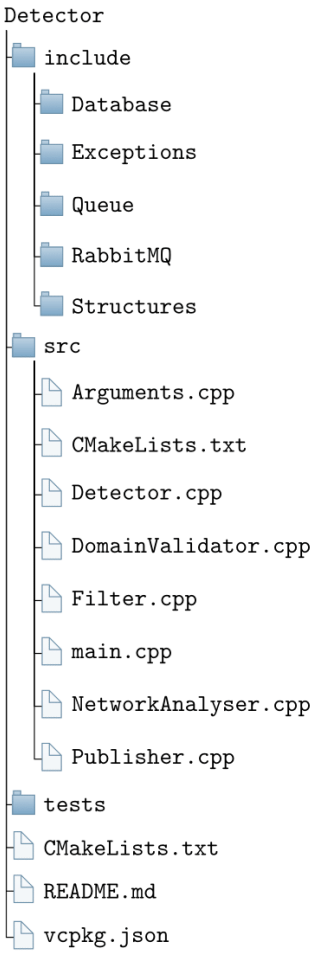
- **LibPcap** – Efektívna a univerzálne použiteľná knižnica v jazyku C pre zachytávanie sieťových paketov, ktorá je kompatibilná s rôznymi operačnými systémami, vrátane Unix, Linux, BSD a Windows. Táto knižnica zjednodušuje proces zachytávania paketov tým, že poskytuje jednotné programovacie rozhranie pre rôzne platformy, čím eliminuje potrebu zaoberať sa špecifikami nižšej úrovne sieťovej komunikácie [21].
- **PcapPlusPlus**[18] – Efektívna knižnica v jazyku C++, ktorá poskytuje jednotné rozhranie pre zachytávanie, spracovanie, analýzu a tvorbu paketov, čo zjednodušuje komplexnosť týchto procesov. Podporuje znovuusporiadanie paketov (packet reassembly) a filtráciu s vlastným jednoduchým a štruktúrovaným mechanizmom na vytváranie filtrov [53].
- **MongoDB C++ Driver**[19] – MongoDB ovládač pre jazyk C++.
- **JSON for Modern C++ (nlohmann-json)** – Poskytuje jednoduchý a intuitívny spôsob práce s formátom JSON v jazyku C++ [51].
- **MPMCQueue** – Fronta umožňujúca viacvláknový prístup k dátam, tzn. povoľuje viacero producentov a konzumentov. Je napísaná v jazyku C++, štandard C++11 a jedná sa o tzv. bez-zámkovú frontu (lock-free queue). Zabezpečuje efektívne vkladanie a vyberanie prvkov s podporou pre blokujúce aj neblokujúce operácie a využíva sa napr. v herných enginoch a v oblastiach s nízkou latenciou [59].

[16] Advanced Message Queuing Protocol
[17] vcpkg: https://vcpkg.io/en/
[18] PcapPlusPlus: https://pcapplusplus.github.io/
[19] MongoDB C++ Driver: https://mongocxx.org/

- **Librabbitmq: RabbitMQ C client** – Multiplatformná knižnica v jazyku C určená na komunikáciu s RabbitMQ, ktorá umožňuje vývojárom integrovať podporu pre protokol AMQP do ich aplikácií [7].
- **POCO** – známe ako C++ Portable Components, je knižnica poskytujúca množstvo funkcií pre prácu s vláknami, sieťovým programovaním, šifrovaním, logovaním atď. [8].

6.4.2 Zostavenie a štruktúra

Prvý blok detekčného nástroja je postavený s použitím vývojového prostredia a nástrojov, ktoré sú štandardom pre jazyk C++. Využíva CMake[20] ako nástroj na automatizáciu zostavovania softvéru, ktorý umožňuje definovať a spravovať zložité projekty s viacerými závislosťami. CMake konfiguračné súbory (`CMakeLists.txt`) sú prítomné v koreňovom adresári projektu a v priečinkoch so zdrojovými súbormi, čo poskytuje možnosť flexibilného a platformovo nezávislého zostavovania projektu.



Obr. 6.1: Ukážka adresárovej štruktúry prvého bloku detekčného nástroja

Využíva hierarchickú štruktúru priečinkov typickú pre CMake projekty, ktorá rozdeľuje zdrojové kódy, hlavičkové súbory, testy, dokumentáciu a ďalšie súčasti projektu do logických

[20] CMake: https://cmake.org/

celkov. Táto štruktúra zabezpečuje prehľadnosť a usporiadanosť projektu, čo uľahčuje jeho správu a rozvoj. Adresárovú štruktúru prvého bloku detekčného nástroja je možné vidieť na obrázku 6.1, pričom každý zdrojový kód `.cpp` má v zložke include rovnako pomenovaný hlavičkový `.hpp` súbor, avšak pre potreby tohto textu bola táto informácia zjednodušená.

V rámci adresárovej štruktúry predstavuje každý `.cpp` súbor (okrem súborov `main.cpp` a `Detector.cpp`, ktoré slúžia ako vstup do prvého bloku a `Arguments.cpp` určený na spracovanie argumentov príkazového riadka) jeden samostatne pracujúci blok ako bolo špecifikované v sekcii 5.5.

6.4.3 Architektúra

Architektúra prvého bloku detekčného nástroja je zameraná na efektívne spracovanie veľkého objemu dát, ktoré prúdia zo systému DNS.

Paralelné a asynchrónne spracovanie

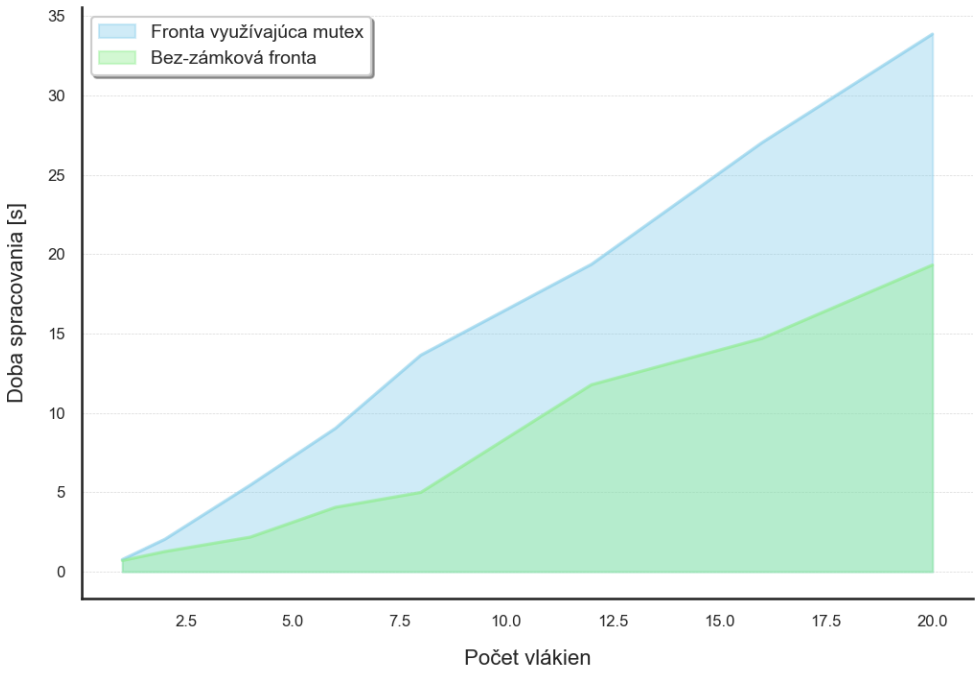
Vzhľadom na náročnosť a objem spracovávaných dát je kľúčové optimalizovať architektúru pre paralelizmus a vysokú dostupnosť. Implementácia v jazyku C++ umožňuje efektívne využitie systémových zdrojov, ako sú procesor a operačná pamäť a z toho dôvodu bolo po dôkladnej analýze a zvážení ostatných prístupov využité viacvláknové programovanie (multithreading) pre implementáciu prvého bloku detekčného nástroja. Táto voľba bola motivovaná nasledujúcimi faktormi:

- **Portabilita** – Multithreading v jazyku C++ je podporovaný na väčšine platformách a operačných systémoch, čo zaisťuje vysokú úroveň portability celého detekčného systému.
- **Efektivita** – Viacvláknové programovanie zabezpečuje efektívne využitie viacjadrových procesorov bez potreby zložitého správania medziprocesovej komunikácie, ktoré je typické pre multiprocessing.
- **Flexibilita** – Multithreading poskytuje veľkú mieru kontroly nad spôsobom, akým sú úlohy spracované a distribuované medzi jednotlivé vlákna, čo poskytuje možnosť optimalizácie výkonu podľa špecifických potrieb detekčného nástroja.

Využitie bez-zámkovej fronty

Bez-zámková fronta je dátová štruktúra navrhnutá na efektívnu prácu v prostredí s viacerými vláknami bez potreby použitia zámkov. Tento dizajn zabezpečuje súbežné pridávanie a odstraňovanie prvkov viacerými producentmi a konzumentmi bez vzájomného blokovania. Kľúčovým aspektom bez-zámkovej fronty je eliminácia potreby čakania a závislosti na tradičných mechanizmoch synchronizácie, ako sú mutexy a semafory, čo vedie k značnému zlepšeniu výkonu [20]. Na obrázku 6.2 je možné vidieť porovnanie implementácie fronty využívajúcej mutex a bez-zámkovej fronty.

Porovnanie výkonnosti implementácií fronty využívajúcej mutex a a bez-zámkovej fronty bolo vykonané na systéme špecifikovanom v tabuľke 6.1. Pre porovnanie bol použitý štandard C++20 s kompilátorom GCC verzie 11.4. Počet vlákien sa pohyboval od 2 do 20, pričom jedno vlákno vykonávalo operáciu zápisu a ostatné vykonávali operáciu čítania na dátovej štruktúre predstavujúcej paket systému DNS. Jednalo sa celkovo o 100 miliónov operácií zápisu. Zaznamenaným metrickým ukazovateľom bola celková doba spracovania,



Obr. 6.2: Porovnanie implementácie fronty využívajúcej mutex a bez-zámkovej fronty

ktorú všetky vlákna potrebovali na dokončenie svojej pracovnej záťaže, meraná pomocou knižnice `std::chrono` v jazyku C++. Každá konfigurácia bola opakovaná päťkrát za účelom vyrovnanosti prípadných variácií a zabezpečenia spoľahlivých údajov.

Operačný systém	Windows 11 Pro
Procesor	AMD Ryzen 7 5850u
Operačná pamäť	16 GB DDR4
Disk	512GB

Tabuľka 6.1: Hardwarová špecifikácia systému použitého na porovnanie implementácií front

V rámci implementácie prvého bloku detekčného nástroja bude využitá implementácia bez-zámkovej fronty MPMCQueue od Erika Rigtorpa [59]. Hlavným dôvodom vybratia tejto implementácia bola možnosť využitia operácie `emplace()`.

Operácia `emplace()` v kontexte dátových štruktúr ako je bez-zámková fronta, umožňuje efektívne pridávanie nových prvkov do fronty priamo na mieste, bez nutnosti predtým vytvárať objekt a potom ho vkladať. Tento prístup minimalizuje počet alokácií pamäte a kopírovacích operácií, čo je obzvlášť užitočné v prostredí s viacerými vláknami, kde každá úspora výkonu môže mať výrazný dopad na celkovú efektivitu systému.

Protokolovanie problémov

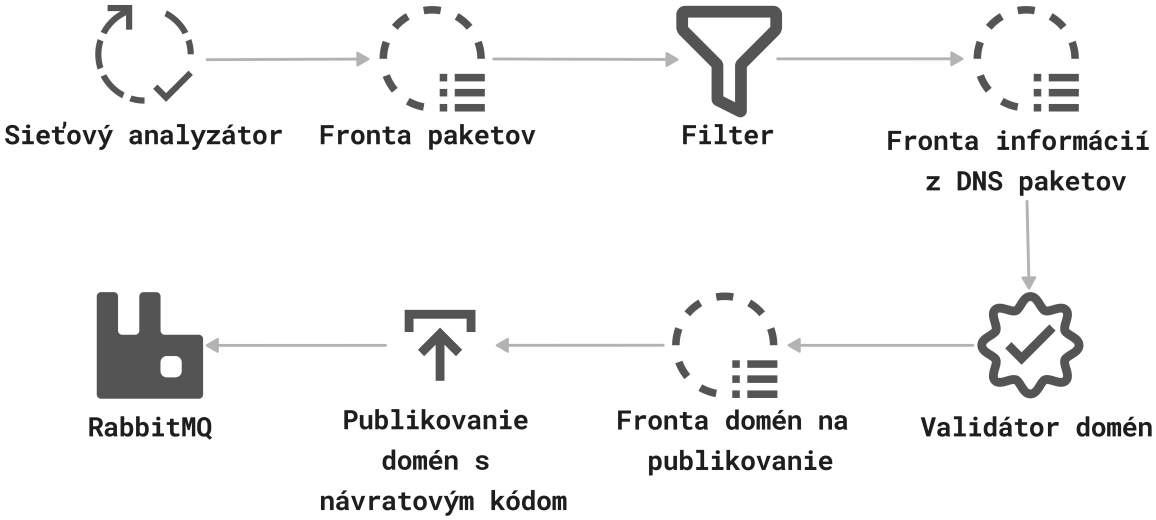
V rámci prvého bloku detekčného nástroja je pre protokolovanie problémov použité logovanie. Na tento účel je využitá knižnica POCO, predstavená v sekcii 6.4.1. Logovanie je

implementované prenositeľne, tzn. informácie sú zapisované buď do systémového denníka udalostí na platforme Windows alebo do systému Syslog na platforme Linux.

Funkčné bloky

Prvý blok detekčného systému sa skladá z viacerých samostatne pracujúcich blokov, ako už bolo špecifikované v kapitole 5 i v sekcii 6.4.2, kde bolo možné vidieť `.cpp` súbory jednotlivých blokov v rámci ukážky adresárovej štruktúry.

Každý jeden blok predstavuje minimálne jedno vlákno programu, tzn. jednotlivé bloky systému môžu byť nasadené vo viacerých inštanciách pracujúcich paralelne voči ostatným blokom a pre komunikáciu medzi jednotlivými blokmi je využitá bez-zámková fronta kvôli svojej efektivite. Na obrázku 6.3 je možné vidieť jednotlivé funkčné bloky prvého bloku detekčného systému.



Obr. 6.3: Funkčné bloky prvého bloku detekčného systému

Sieťový analyzátor

Sieťový analyzátor má za úlohu sledovať, analyzovať a zachytávať pakety, ktoré generuje systém DNS ako bolo špecifikované v sekcii 5.5.1. V rámci implementácie v jazyku C++ je možné využiť širokú škálu knižníc, medzi ktoré patria LibPcap a PcapPlusPlus popísané v sekcii 6.4.1, alebo je taktiež možné využiť implementáciu s využitím raw soketov.

Raw sokety v jazyku C umožňujú priamu manipuláciu so sieťovými paketmi na najnižšej úrovni. Tento prístup poskytuje absolútnu kontrolu nad paketmi a ich obsahom, čo je vhodné pre špecifické scenáre, ako je implementácia vlastného sieťového analyzátora s presnými požiadavkami na spracovanie paketov. Použitie raw soketov si vyžaduje manipuláciu s hlavičkami paketov, vytváranie vlastných správ a riadenie toku dát. Tento prístup poskytuje maximálnu flexibilitu a kontrolu nad sieťovou komunikáciou, avšak je potrebné zvážiť zvýšenú náročnosť a zodpovednosť spojenú s priamym prístupom k sieťovému rozhraniu.

Pre implementáciu sieťového analyzátora bola zvolená knižnica LibPcap z dôvodu jej veľmi efektívneho spracovania, schopnosti pracovať na rôznych platformách a taktiež prenositeľnosti. V tabuľke 6.2 je porovnanie výkonnosti knižníc LibPcap, PcapPlusPlus a imple-

mentácie s využitím raw soketov vzhľadom na počet spracovaných paketov systému DNS za sekundu.

Metóda	**Celkový čas [s]**	**Rýchlosť spracovania [paketov/s]**
LibPcap	0.087	5 747 126
Použitie Raw soketov	0.096	5 208 333
PcapPlusPlus	0.259	1 930 502

Tabuľka 6.2: Porovnanie výkonnosti medzi knižnicami jazyka C++ a použitím Raw soketov

Porovnanie výkonnosti bolo vykonané na hardvérovej konfigurácii špecifikovanej v tabuľke 6.3. Každý test bol vykonaný pri spracovaní 500,000 paketov systému DNS, pričom každá konfigurácia bola opakovaná päťkrát za účelom vyrovnanosti prípadných variácií a zabezpečenia spoľahlivých údajov.

Operačný systém	Windows 11 Pro
Procesor	AMD Ryzen 7 5850u
Operačná pamäť	16 GB DDR4
Disk	512GB

Tabuľka 6.3: Hardwarová špecifikácia systému použitého na porovnanie prístupov

Filter

Implementácia bloku Filter slúži na filtráciu domén pomocou príznaku NXDomain [45] ako bolo špecifikované v sekcii 5.5.3.

Filtrácia domén na základe príznaku NXDomain je realizovaná prostredníctvom podrobnej analýzy DNS hlavičky. Tento proces zahŕňa vyhodnocovanie hodnoty Response Code (RCODE) v DNS odpovedi. Ak sa hodnota RCODE rovná 3, signalizuje to prítomnosť príznaku NXDomain v danom DNS pakete. Výstupom tohto bloku sú dva typy paketov: pakety s príznakom NXDomain i tie, ktoré NXDomain príznak neobsahujú. Pre účely detailnej analýzy DNS hlavičiek bola využitá knižnica PcapPlusPlus popísaná vyššie.

Validátor domén

Validátor domén má za úlohu validáciu domén v rámci zoznamov Blacklist a Whitelist ako bolo špecifikované v rámci návrhu, konkrétne v sekcii 5.5.2.

Validácia jednotlivých domén prebieha po dávkach, tzn. načíta sa maximálne možné množstvo až po dosiahnutie maximálnej veľkosti dávky alebo maximálneho počtu cyklov, čím sa zaisťuje efektívne spracovanie bez zbytočného meškania. Táto metóda dávkového spracovania je zvlášť výhodná v situáciách, kde systém čelí prívalom DNS požiadaviek, umožňuje flexibilnú adaptáciu na meniace sa objemy dát a optimalizuje využitie zdrojov.

Po zhromaždení dávky domén prebieha ich validácia v rámci oboch zoznamov Blacklist a Whitelist s využitím knižnice MongoDB C++ Driver špecifikovanej vyššie. Táto knižnica je využívaná na dotazovanie sa na známe benígne, či DGA doménové mená, čo je kľúčové pre ich identifikáciu a odfiltrovanie známych škodlivých alebo nežiaducich domén.

Pre databázu je tento prístup k validácii domén veľmi výhodný, keďže redukuje počet dotazov tým, že vykonáva hromadné kontroly, namiesto individuálnych dotazov pre každú doménu zvlášť. To môže výrazne znižovať záťaž na databázový server a zlepšovať celkovú

odozvu systému. Okrem toho, efektívne využitie cache pre často vyhľadávané domény môže ešte viac znížiť potrebu opakovaných dotazov na databázu. Vďaka optimalizácii dotazov a inteligentnému manažmentu dát sa tak zvyšuje škálovateľnosť a efektivita databázových operácií, čo je kľúčové pre systémy, ktoré musia spracovávať veľké objemy informácií v reálnom čase.

Publikovanie domén s návratovým kódom

Implementácia bloku zodpovedného za publikovanie domén s návratovým kódom vykonáva spracovanie dvojíc `doménové meno : návratová hodnota` do formátu JSON[21]. Pre toto spracovanie je využívaná knižnica JSON for Modern C++ (nlohmann-json) popísaná vyššie.

Po dokončení spracovania je výsledný zoznam domén s návratovými hodnotami odoslaný pomocou sprostredkovateľa správ RabbitMQ s využitím knižnice Librabbitmq: RabbitMQ C client do druhého bloku detekčného nástroja, ktorý je popísaný v sekcii 6.5.

6.5 Implementácia druhého bloku detekčného nástroja

Druhý blok detekčného nástroja, ktorého návrh bol popísaný v sekcii 5.6, sa zaoberá extrakciou vlastností získaných z paketov systému DNS a ich následnou klasifikáciou. V tejto sekcii bude bližšie popísaná jeho implementácia. Jazykom implementácie pre tento blok sa stal jazyk Python, čo už bolo spomenuté v sekcii 6.1.

6.5.1 Využité knižnice

V rámci implementácie druhého bloku detekčného nástroja boli využité mnohé knižnice jazyka Python, ktoré boli nainštalované a spravované pomocou systému na správu balíčkov, ktorý sa nazýva Pip[22].

Pip je štandardný správca balíčkov pre jazyk Python, ktorý umožňuje užívateľom inštalovať, aktualizovať a odstraňovať knižnice jazyka Python a balíčky z PyPI[23] alebo iných repozitárov. To zjednodušuje správu závislostí v projekte a podporuje rôzne operačné systémy vrátane Windows, Linux a macOS.

Knižnice boli vybrané na základe ich funkcionality a prínosov pre projekt s cieľom zabezpečiť efektívnu implementáciu požadovaných funkcií a zvýšiť celkovú kvalitu a výkon aplikácie. Použitie jednotlivých knižníc bude špecifikované v sekcii 6.5.3, pričom medzi kľúčové využité knižnice patria:

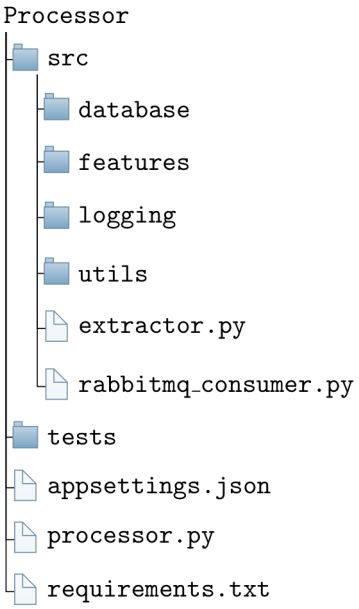
- **Pika** – Knižnica pre jazyk Python určená na prácu s protokolom AMQP 0-9-1, vrátane rozšírenia RabbitMQ[24], ktorá je implementovaná v jazyku Python [55].
- **PyMongo**[25] – MongoDB ovládač pre jazyk Python.
- **NumPy** – Knižnica v jazyku Python určená pre matematické a vedecké výpočty. Táto knižnica je nevyhnutná pre efektívnu prácu s veľkými dátovými súbormi, umožňuje vykonávanie komplexných matematických a štatistických operácií na poliach a viacrozmerných maticiach [19].

[21] JSON: https://www.json.org/
[22] Pip: https://pypi.org/project/pip/
[23] Python Package Index - PyPI: https://pypi.org/
[24] RabbitMQ: https://www.rabbitmq.com/
[25] PyMongo: https://pymongo.readthedocs.io/

- **Pandas** – Knižnica v jazyku Python, ktorá je využívaná predovšetkým v oblasti analýzy dát. Zabezpečuje manipuláciu s dátami a ich analýzu deklaratívnym spôsobom, čo zjednodušuje prácu s veľkými objemami informácií. Dáta v Pandas sú organizované do dvojrozmerných štruktúr nazývaných DataFrame, kde každý stĺpec môže obsahovať typ dát nezávislý na ostatných stĺpcoch. Táto funkcionalita poskytuje vysokú flexibilitu pri práci s heterogénnymi dátami. DataFrame podporuje rôzne operácie, ako sú výber, pridávanie alebo odstraňovanie dát, čo z neho robí mocný nástroj pre predspracovanie a transformáciu dát pred ich analýzou alebo vizualizáciou [17].
- **Pyahocorasick** – Knižnica navrhnutá na rýchle a pamäťovo efektívne vyhľadávanie viacerých vzorov v textových reťazcoch. Výhodou je možnosť predčasného zostavenia a uloženia indexu, ktorý je možné neskôr znovu načítať a použiť. Algoritmus Aho-Corasick, ktorý je implementovaný v tejto knižnici, slúži na efektívne vyhľadávanie viacerých kľúčových slov naraz v jedinom prechode cez text [49].
- **Joblib** – Knižnica optimalizovaná pre ukladanie a načítanie objektov jazyka Python, ktoré využívajú dátové štruktúry NumPy, pričom sa často využíva kvôli jej efektívnosti pri ukladaní a načítaní modelov strojového učenia [65].

6.5.2 Štruktúra

Druhý blok detekčného nástroja je navrhnutý s využitím vývojového prostredia a nástrojov, ktoré sú štandardom pre jazyk Python. Využíva systém pip[26] pre správu balíčkov a virtualenv pre izoláciu závislostí, čo umožňuje jednoduchú a efektívnu prácu s balíčkami potrebnými pre projekt. Konfiguračné súbory requirements.txt sú umiestnené v koreňovom adresári projektu a obsahujú zoznam všetkých externých závislostí potrebných pre jeho spustenie.



Obr. 6.4: Ukážka adresárovej štruktúry druhého bloku detekčného nástroja

[26] Pip: https://pypi.org/project/pip/

Využíva štruktúrovanú adresárovú hierarchiu typickú pre projekty v jazyku Python, ktorá umožňuje rozdelenie zdrojových kódov, testov, dokumentácie a ďalších súčastí do logicky oddelených celkov. Táto štruktúra zabezpečuje ľahkú orientáciu v projekte a podporuje jeho prehľadnosť a usporiadanosť, čo napomáha jeho správe a ďalšiemu rozvoju. Adresárovú štruktúru druhého bloku detekčného nástroja je možné vidieť na obrázku 6.4.

V rámci adresárovej štruktúry predstavuje každý `.py` súbor jeden samostatne pracujúci blok.

6.5.3 Architektúra

Architektúra druhého bloku detekčného nástroja je zameraná na efektívne spracovanie dát prichádzajúcich z prvého bloku detekčného nástroja.

Paralelné a asynchrónne spracovanie

V rámci optimalizácie a efektívnosti spracovania veľkých objemov dát je nevyhnutné zameranie sa na paralelizmus, prípadne asynchrónne spracovanie pre zabezpečenie vysokej dostupnosti druhého bloku detekčného nástroja. Po dôkladnej analýze a zvážení všetkých dostupných prístupov bola pre implementáciu druhého bloku detekčného nástroja zvolená stratégia viacvláknového programovania (threading) aj napriek oboznámeniu sa s problémom Global Interpreter Lock (GIL) v rámci jazyka Python. Využitie viacerých procesov bude slúžiť výhradne na extrakciu vlastností, ktorá bola popísaná v sekcii 5.6.1. Toto rozhodnutie bolo motivované nasledujúcimi faktormi:

- **Kompatibilita s knižnicami** – Pri vývoji bolo zistené, že využitie viacerých procesov (multiprocessing) spôsobuje problémy s kompatibilitou viacerých používaných knižníc, ktoré sú kľúčové pre funkčnosť detekčného nástroja. V prípade využitia výlučne na extrakciu vlastností sa nevyskytovali žiadne problémy.
- **Simplifikácia dizajnu** – Viacvláknové programovanie zabezpečuje jednoduchšie a priamejšie rozhranie pre správu vlákien v porovnaní s viacprocesovým programovaním, čo zjednodušuje dizajn a implementáciu detekčného nástroja bez nutnosti zaoberať sa zložitosťami medziprocesovej komunikácie.

Protokolovanie problémov

Protokolovanie problémov je zrealizované v druhom bloku detekčného nástroja prostredníctvom logovania. Na tieto účely je využitý štandardný logovací mechanizmus jazyka Python, a to prostredníctvom štandardnej knižnice `logging`. Implementácia logovania je prenositeľná, čo znamená, že informácie sú zaznamenávané buď do systémového denníka udalostí na operačnom systéme Windows alebo do logovacieho systému Syslog na operačnom systéme Linux.

Funkčné bloky

Druhý blok detekčného systému sa skladá z viacerých samostatne pracujúcich blokov, ako už bolo špecifikované v sekcii 6.5.2, kde boli zobrazené `.py` súbory jednotlivých blokov v rámci ukážky adresárovej štruktúry.

Každý jeden blok predstavuje minimálne jedno vlákno programu, tzn. jednotlivé bloky systému môžu byť nasadené vo viacerých inštanciách pracujúcich kvázi-paralelne voči ostatným blokom a pre komunikáciu medzi jednotlivými blokmi je využitá fronta zo štandardnej

knižnice jazyka Python. Na obrázku 6.5 sú zobrazené jednotlivé funkčné bloky druhého bloku detekčného systému.

Obr. 6.5: Funkčné bloky druhého bloku detekčného systému

Odberateľ RabbitMQ správ

Implementácia odberateľa RabbitMQ[27] správ je zodpovedná za prijímanie a spracovanie správ odoslaných prostredníctvom sprostredkovateľa správ RabbitMQ, pričom na tento účel využíva knižnicu Pika špecifikovanú v sekcii 6.5.1. Tento blok slúži ako most medzi publikujúcim blokom, ktorý odosiela zoznam domén s návratovými hodnotami a ďalšími komponentmi detekčného nástroja, ktoré vykonávajú hlbšiu analýzu na základe prijatých dát. Ako kľúčová súčasť systému, odberateľ správ RabbitMQ umožňuje efektívne spracovanie prichádzajúcich správ, čím znižuje latenciu a zlepšuje reakčnú schopnosť celého systému.

Správy sú prijímané cez špecifický kanál RabbitMQ a po ich obdržaní sú ihneď presunuté do vnútornej fronty, viď. 6.5, pre ďalšie spracovanie. Každá správa je spracovaná s maximálnou prioritou, aby boli dáta čo najskôr k dispozícii pre analýzu. Implementácia odberateľa správ RabbitMQ je odolná voči chybám a zabezpečuje neustále pripojenie k serveru RabbitMQ. V prípade odpojenia alebo iných neočakávaných chýb, systém automaticky pokračuje v opätovnom pripojení, čím minimalizuje prerušenie v spracovaní správ. Každý pokus o opätovné pripojenie je logovaný, čo umožňuje lepšiu diagnostiku a údržbu systému.

Implementácia odberateľa správ RabbitMQ predstavuje kľúčovú časť infraštruktúry detekčného nástroja, pričom zaručuje, že správy sú prijímané, spracované a presmerované do interných front s vysokou spoľahlivosťou a minimálnou stratou údajov.

Extrakcia a klasifikácia

Implementácia bloku pre extrakciu a klasifikáciu je zásadnou súčasťou druhého bloku detekčného systému. Tento blok je zodpovedný za spracovanie správ JSON[28] v tvare `doménové meno : návratová hodnota`. Po prijatí správy tento blok vykonáva dva hlavné kroky: extrakciu vlastností z doménových mien a následnú klasifikáciu každého doménového mena.

Extrakcia vlastností prebieha s využitím knižníc NumPy a Pandas. Zahŕňa analýzu každého doménového mena s cieľom identifikovať špecifické charakteristiky, ktoré môžu naznačovať, či ide o DGA alebo o benígne doménové meno. Všetky využité vlastnosti boli popísané v rámci návrhu, konkrétne v sekcii 5.6.1. Pre implementáciu extrakcie vlastnosti, ktorá sa zaoberá hľadaním počtu zhôd s DGA N-gramami bol využitý algoritmus Aho-Corasick, konkrétne jeho implementácia v jazyku Python, Pyahocorasick, ktorý extrakciu tejto vlastnosti výrazne zefektívňuje, viď tabuľka 6.4.

[27] RabbitMQ: https://www.rabbitmq.com/
[28] JSON: https://www.json.org/

Verzia	Počet domén	Rýchlosť spracovania [doménové meno/s]
Klasická	22959	237.552
Aho-Corasick	22959	101588.5

Tabuľka 6.4: Porovnanie rýchlosti implementácií hľadania zhôd s DGA N-gramami

Porovnanie rýchlosti implementácií bolo vykonané na systéme špecifikovanom v tabuľke 6.5. Prebiehalo testovaním na vzorke 22959 doménových mien, pochádzajúcej z dátových sád špecifikovaných v sekcii 5.2. Z dôvodu zabezpečenia spoľahlivých údajov a vyrovnanosti prípadných variácií bola každá konfigurácia opakovaná päť krát.

Operačný systém	Windows 11 Pro
Procesor	AMD Ryzen 7 5850u
Operačná pamäť	16 GB DDR4
Disk	512GB

Tabuľka 6.5: Hardwarová špecifikácia systému použitého na porovnanie implementácií extrakcie vlastností

Klasifikácia predstavuje fázu, kde je každé doménové meno priradené do jednej z preddefinovaných kategórií, tzn. či ide o DGA alebo o benígne doménové meno. Pred začiatkom klasifikácie je potrebné najprv načítať modely. K tomuto účelu sa využíva knižnica joblib, ktorá umožňuje efektívne načítanie predom natrénovaných modelov strojového učenia a hlbokého učenia. Pre účely klasifikácie doménových mien je využitý najlepší z predom natrénovaných modelov strojového učenia a hlbokého učenia, konkrétne taký, ktorý má najväčšiu presnosť a efektivitu. Konkrétne využité modely sú špecifikované v kapitole 7.

Po úspešnej klasifikácii doménových mien systém následne uloží výsledky do databázy MongoDB s využitím knižnice PyMongo. Tento krok zahŕňa zápis informácií o každom doménovom mene, jeho kategorizáciu (DGA alebo benígne), ako aj ďalších relevantných metadát, ktoré môžu byť využité pre ďalšiu analýzu alebo monitorovanie.

6.6 Implementácia webovej aplikácie

Tretí blok detekčného nástroja, ktorého návrh bol popísaný v sekcii 5.7 sa zaoberá zobrazovaním informácií o klasifikovaných doménach, čase ich klasifikácie a ďalších informácií s tým spojenými. Jazykom implementácie aplikačného rozhrania (API) sa stal C#, konkrétne bude využitý framework ASP.NET a pre implementáciu webového rozhrania, ktoré využíva toto API, je využitá knižnica React[29] v jazyku Javascript, pričom tieto informácie už boli špecifikované v rámci sekcie 6.1.

6.6.1 Využité knižnice

V tejto sekcii budú predstavené využité knižnice aplikačného programovacieho, resp. webového rozhrania.

[29]React: https://react.dev/

Aplikačné programovacie rozhranie

Pri implementácii aplikačného programovacieho rozhrania v prostredí C# ASP.NET boli využité mnohé knižnice, ktoré boli nainštalované a spravované pomocou systému na správu balíčkov NuGet[30].

NuGet je štandardný správca balíčkov pre platformu .NET, ktorý umožňuje užívateľom inštalovať, aktualizovať a odstraňovať knižnice a balíčky z úložiska NuGet alebo iných zdrojov. Tento nástroj zjednodušuje správu závislostí v projekte a je podporovaný na rôznych operačných systémoch vrátane Windows, Linux a macOS.

Knižnice boli vybrané na základe ich funkcionality a prínosov pre projekt, s cieľom zabezpečiť efektívnu implementáciu požadovaných funkcií, zvýšiť celkovú kvalitu a výkon aplikácie. Medzi kľúčové knižnice v rámci aplikačného programovacieho rozhrania, ktorých využitie je popísané v sekcii 6.6.3, môžeme zahrnúť:

- **AutoMapper** – Nástroj pre automatizované mapovanie objektov v aplikáciách, uľahčujúci prenos dát medzi rôznymi vrstvami. Používa sa na jednoduché a efektívne transformácie komplexných objektov na jednoduchšie, čím zjednodušuje serializáciu a komunikáciu [11]. Jeho flexibilita a široká škála funkcií, vrátane validácie a vkladania závislostí, robia z AutoMapper silný nástroj pre vývojárov.
- **Swashbuckle.AspNetCore** – Nástroj pre ASP.NET Core aplikácie, ktorý slúži na generovanie dokumentácie API. Poskytuje prehľadnú a interaktívnu dokumentáciu, ktorá poskytuje informácie o rôznych API volaniach, ich parametroch a očakávaných odpovediach, avšak taktiež umožňuje vývojárom testovať tieto volania API priamo v prehliadači [48].
- **Entity Framework Core** – Moderný objektovo-relačný mapovací framework pre .NET, ktorý umožňuje vývojárom pracovať s databázou pomocou .NET objektov, čím sa znižuje potreba písať komplikovaný kód SQL. Tento framework poskytuje podporu pre LINQ[31] dotazy, sledovanie zmien, migrácie a seedovanie databázy, čo uľahčuje a zefektívňuje vývoj databázových aplikácií. Využitím EF Core s MySQL a SQLite sa zjednodušuje integrácia a manipulácia s dátami v aplikáciách, čo prispieva k vyššej produktivite vývoja a udržateľnosti kódu [42].
- **Serilog** – Logovací nástroj pre .NET, ktorý podporuje štruktúrované logovanie a poskytuje možnosť ľahkého smerovania logov do rôznych destinácií [50].
- **Scrutor** – Knižnica pre .NET, ktorá zabezpečuje vkladanie závislostí automatickým skenovaním a registráciou tried v rámci aplikácií. Táto knižnica je kľúčom k efektívnejšiemu a pružnejšiemu kódu, pretože umožňuje vývojárom jednoduchšie spravovať závislosti a implementovať rozšíriteľné systémy [41].
- **ASP.NET Identity** – API poskytujúce možnosť registrácie, prihlásenia a správy užívateľov v ASP.NET aplikáciách. Je určené pre webové aplikácie, kde pridáva užívateľské rozhranie pre autentifikáciu a autorizáciu [6].

[30] NuGet: https://www.nuget.org/
[31] Language Integrated Query

Webové rozhranie

Pri implementácii webového rozhrania v prostredí React a jazyku Javascript boli využité mnohé knižnice, ktoré boli nainštalované a spravované pomocou systému na správu balíčkov NPM[32].

NPM (Node Package Manager) je populárny nástroj pre správu balíčkov pre jazyk JavaScript. Poskytuje vývojárom možnosť jednoducho inštalovať a aktualizovať balíčky, ktoré potrebujú pre svoje projekty. S NPM je možné ľahko integrovať a používať rôzne knižnice a nástroje, čo zrýchľuje vývoj aplikácií. Je to kľúčový nástroj pre vývojárov pracujúcich s Node.js a jazykom JavaScript.

Boli vybrané iba typické knižnice pre knižnicu React, ktoré sú nevyhnutné pre správne fungovanie aplikácie a umožňujú ľahkú integráciu potrebných funkcií. Medzi ne patria knižnice na správu stavu aplikácie, na smerovanie a komponenty na prácu s formulármi.

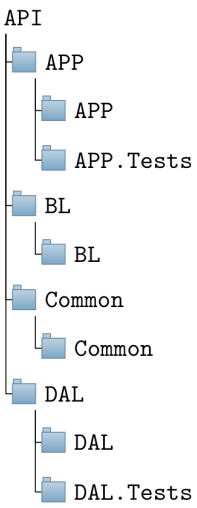
6.6.2 Štruktúra

V tejto sekcii bude popísaná štruktúra tretieho bloku detekčného nástroja, pričom bude najprv predstavená štruktúra aplikačného programovacieho rozhrania a následne webového rozhrania.

Aplikačné programovacie rozhranie

Aplikačné programovacie rozhranie je navrhnuté s využitím vývojového prostredia a nástrojov, ktoré sú štandardom pre jazyk C#. Pre správu závislostí využíva NuGet správcu balíčkov, ktorý bol špecifikovaný v sekcii 6.6.1.

Riešenie (Solution) je štruktúrované so štandardmi a osvedčenými postupmi pre ASP–.NET aplikácie, čo zahŕňa rozdelenie riešenia do viacerých projektov alebo knižníc v rámci jedného riešenia pre lepšiu modularitu a udržateľnosť. Každá časť návrhu aplikačného programovacieho rozhrania predstavuje jednu časť architektúry špecifikovanej v sekcii 5.7.2, pričom projekt Common slúži pre uloženie spoločných častí architektúry. Štruktúru implementácie aplikačného rozhrania je možné vidieť na obrázku 6.6.



Obr. 6.6: Ukážka adresárovej štruktúry aplikačného programovacieho rozhrania

[32]NPM: https://www.npmjs.com/

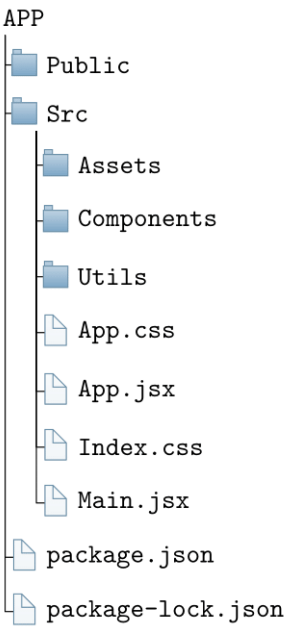
Webové rozhranie

Webové rozhranie je vyvinuté pomocou štandardných nástrojov pre vývoj webových aplikácií v jazyku JavaScript. Na správu balíčkov sa využíva npm popísaný v sekcii 6.6.1. Konfiguračné súbory `package.json` a `package-lock.json` sú umiestnené v koreňovom adresári a sú kľúčové pre správu závislostí aplikácie.

Súbor `package.json` obsahuje metadáta, ktoré umožňujú identifikovať projekt a jeho konfiguračné parametre. Zahŕňa informácie ako názov projektu, verzia, autor, licencia a zoznam závislostí , ktoré definujú iné balíky npm, potrebné pre fungovanie aplikácie.

Súbor `package-lock.json` je automaticky generovaný súbor, ktorý presne špecifikuje strom závislostí použitý pri poslednej inštalácii. Tento súbor zabezpečuje, že inštalácie sú konzistentné a reprodukovateľné, aj keď sa vykonávajú v rôznych prostrediach, čím zaisťuje kompatibilitu a funkčnosť aplikácie bez ohľadu na lokálne rozdiely v balíčkoch npm.

Využíva štruktúrovanú adresárovú hierarchiu typickú pre projekty v rámci knižnice React[33], ktorá umožňuje rozdelenie zdrojových kódov, komponentov a ďalších súčastí do logicky oddelených celkov. Táto štruktúra zabezpečuje ľahkú orientáciu v projekte a podporuje jeho prehľadnosť a usporiadanosť, čo napomáha jeho správe a ďalšiemu rozvoju. Štruktúru implementácie webového rozhrania je možné vidieť na obrázku 6.7.



Obr. 6.7: Ukážka adresárovej štruktúry webového rozhrania

6.6.3 Architektúra aplikačného programovacieho rozhrania

V tejto sekcii bude predstavená architektúra prvej časti tretieho bloku detekčného nástroja, konkrétne sa jedná o architektúru aplikačného programovacieho rozhrania.

[33]React: https://react.dev/

Asynchrónne spracovanie

Pri zameraní sa na optimalizáciu a efektívnosť spracovania veľkých objemov dát je nevyhnutné využívanie asynchrónneho spracovania, najmä pri zabezpečovaní vysokej dostupnosti kritických komponentov systému. Po dôkladnej analýze a zvážení všetkých dostupných prístupov pre implementáciu kritických častí systému, konkrétne pre aplikačné programovacie rozhranie, bola pre C# zvolená stratégia asynchrónneho programovania s využitím konštruktov `async` a `await`.

Tento prístup je v C# ideálny pre operácie, kde je potrebné dosiahnuť vysokú efektivitu bez blokovania vykonávania hlavného vlákna aplikácie, čo je zásadné pre udržanie responzivity a dostupnosti aplikácie. Asynchrónne programovanie umožňuje aplikácii efektívne spracovávať veľké množstvo operácií, napríklad pri komunikácii so vzdialenými servermi, čítaní veľkých súborov alebo vykonávaní náročných databázových dotazov, bez nutnosti čakať na dokončenie každej operácie a tým blokovať ďalšie procesy.

Vkladanie závislostí

Vkladanie závislostí (Dependency Injection, DI) prispieva k väčšej modularite a testovateľnosti aplikácií. V kontexte architektúry aplikačného programovacieho rozhrania je využívanie DI nielen žiaduce, ale často nevyhnutné pre dosiahnutie čistej separácie zodpovedností medzi rôznymi časťami systému.

Pri implementácii DI v prostredí jazyka C#, najmä v rámci asynchrónne navrhnutej architektúry, sa využívajú špecializované kontajnery pre DI, ktoré zjednodušujú správu životného cyklu objektov, ich vytváranie, zdieľanie a zničenie. Tieto kontajnery umožňujú definovať závislosti medzi rôznymi časťami aplikácie deklaratívne, čo vedie k dynamickejšej a flexibilnejšej štruktúre kódu.

Základným princípom návrhového vzoru DI je inverzia závislostí, čo znamená, že miesto priameho vytvárania závislých objektov v rámci jednotlivých komponentov, sú tieto objekty injektované zvonka, často prostredníctvom konštruktora alebo vlastností (properties). Týmto spôsobom sa dosahuje oddelenie vytvárania objektov od ich používania, čo vedie k väčšej flexibilite a uľahčuje testovanie jednotlivých komponentov aplikácie.

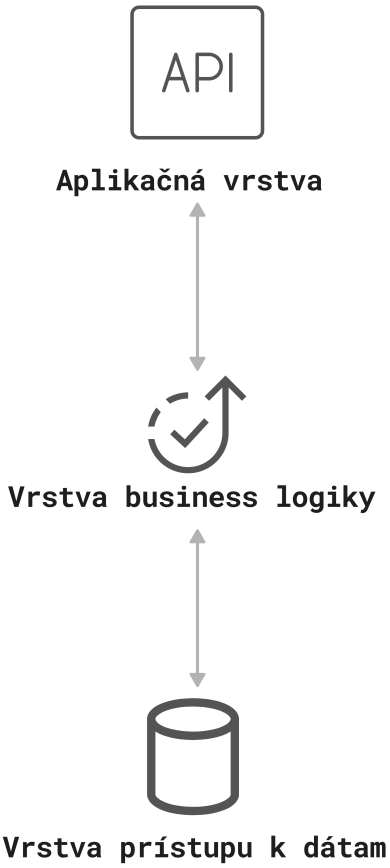
Knižnica Scrutor, špecifikovaná v sekcii 6.6.1, bola vybraná pre tieto účely z dôvodu, že umožňuje rozšírenie služieb `IServiceCollection` ponúkaných frameworkom ASP.NET Core, čo značne uľahčuje implementáciu vzorov DI pri zachovaní vysokej konfigurovateľnosti a efektívnosti. Scrutor poskytuje pokročilé možnosti na skenovanie a registráciu závislostí, čím sa znižuje množstvo manuálneho kódu potrebného pre konfiguráciu DI kontajnera, a tým je možné ľahko implementovať čistejšiu a modulárnejšiu štruktúra aplikačného programovacieho rozhrania.

Protokolovanie problémov

V rámci webovej aplikácie je pre protokolovanie problémov použité logovanie. Na tento účel je využitá knižnica Serilog, predstavená v sekcii 6.6.1. Logovanie je implementované prenositeľne, tzn. informácie sú zapisované buď do systémového denníka udalostí na platforme Windows alebo do systému Syslog na platforme Linux.

Funkčné bloky

Aplikačné programovacie rozhranie sa skladá z viacerých blokov, konkrétne z vrstvy prístupu k dátam, vrstvy business logiky a aplikačnej vrstvy, ako už bolo špecifikované v kapitole 5 i v sekcii 6.6.2, pričom ich je taktiež možné vidieť na obrázku 6.8.



Obr. 6.8: Funkčné bloky aplikačného programovacieho rozhrania

Vrstva prístupu k dátam

Vrstva prístupu k dátam (Data Access Layer, DAL) predstavuje komponent aplikačného programovacieho rozhrania, ktorý pracuje s databázou. Jej hlavnou úlohou je abstrahovať prístup k databáze tak, aby aplikácia mohla pracovať s dátami bez potreby zaoberať sa špecifikami konkrétnej databázy alebo dotazovacím jazykom. Táto vrstva teda poskytuje jednotné API (aplikačné programovacie rozhranie) pre vyššie vrstvy aplikácie a zároveň izoluje logiku prístupu k dátam od zvyšku systému. Skladá sa z viacerých komponentov, medzi ktoré patria:

- **Entity** – Predstavujú dátové modely alebo objekty, ktoré reprezentujú štruktúru a charakteristiky dát uložených v databáze. Ich primárnou funkciou je mapovanie databázových tabuliek a záznamov v prípade MongoDB na objekty, ktoré sú použité v aplikačnom programovacom rozhraní, čo umožňuje prácu s dátami s vyššou úrovňou abstrakcie.

- **Repozitáre** – Predstavujú implementáciu návrhového vzoru Repository, ktorý abstrahuje logiku prístupu k databáze s využitím knižnice Entity Framework Core, špecifikovanej v sekcii 6.6.1, pre rôzne entity aplikácie. Každý repozitár spravuje operácie pre konkrétnu entitu, čím zjednodušuje prácu s dátami a prispieva k lepšej testovateľnosti a oddeleniu zodpovedností v aplikácii.

- **Migrácie** – Sú dôležitým mechanizmom pre správu zmien v schéme databázy. Umožňujú definíciu zmien databázy v programovateľnej forme, čo zabezpečuje konzistenciu databázového schématu naprieč rôznymi prostrediami.

- **Inštalátory (Installers)** – Sú súčasťou mechanizmu na konfiguráciu a registráciu komponentov vrstvy prístupu k dátam v rámci DI kontajnera. Využíva konfiguračné nastavenia databázy na dynamické nastavenie DbContextu pre MySQL alebo SQLite databázy, čím umožňuje flexibilitu pri výbere databázového systému. Taktiež automaticky hľadá a registruje všetky repozitáre, ktoré implementujú rozhranie `IRepository<>`, a zaisťuje ich dostupnosť cez DI kontajner ako služby s obmedzeným životným cyklom.

- **ApiDbContext** – Predstavuje komponent vrstvy prístupu k dátam, prostredníctvom ktorej aplikácia komunikuje s MongoDB, poskytujúc tak prístup k databázovým operáciám, ako sú čítanie, zápis, aktualizácia alebo mazanie dát. V rámci architektúry aplikačného programovacieho rozhrania zabezpečuje ApiDbContext izoláciu a abstrakciu databázových operácií špecifických pre MongoDB, čím umožňuje vývojárom sústrediť sa na biznis logiku bez potreby hlbokého zaoberania sa detailmi implementácie databázových volaní.

- **UserDbContext** – Predstavuje komponent vrstvy prístupu k dátam, ktorý rozširuje funkcionalitu IdentityDbContext z balíčka Microsoft.AspNetCore.Identity.EntityFrameworkCore a umožňuje efektívne spravovať užívateľské údaje v databáze. Pracuje s entitou User, ktorá je špecificky prispôsobená potrebám aplikácie. Vďaka integrácii s Entity Framework Core, UserDbContext poskytuje flexibilitu v použití s rôznymi databázovými systémami, ako sú MySQL alebo SQLite, čím umožňuje výber optimálnej databázy podľa špecifických potrieb a prostredia aplikácie. UserDbContext tak slúži ako kritický most medzi aplikačným programovacím rozhraním a jej databázovým úložiskom, zabezpečujúc efektívne vykonávanie operácií ako sú autentifikácia, autorizácia, a správa užívateľských profilov.

Vrstva business logiky

Vrstva business logiky (Business Layer, BL) je časťou aplikačného programovacieho rozhrania, ktorá zabezpečuje spracovanie a riadenie dát, operácií a pravidiel aplikácie. Táto vrstva sa nachádza medzi vrstvou prístupu k dátam (DAL) a aplikačnou vrstvou, kde slúži na prenos požiadaviek medzi užívateľským rozhraním a databázou. Cieľom BL je abstrahovať a spracovať špecifické požiadavky aplikácie, čím uľahčuje vývoj, údržbu a testovanie aplikácie.

V kontexte popisovaného aplikačného programovacieho rozhrania môžeme identifikovať niekoľko kľúčových komponentov vrstvy business logiky:

- **Modely** – Definujú štruktúru dát, ktoré sú používané v rámci business logiky pre spracovanie a riadenie obchodných pravidiel aplikácie. Modely predstavujú abstrakciu

entít a konceptov špecifických pre vrstvu prístupu k dátam aplikácie a slúžia ako prenosové objekty medzi vrstvami aplikácie.

- **Profily pre mapovanie (Mapper profiles)** – Obsahujú konfigurácie pre automatizované mapovanie dát medzi modelmi a entitami v rámci aplikačného programovacieho rozhrania. Využitie knižnice AutoMapper uľahčuje transformáciu dát pri ich prenose medzi vrstvami aplikácie a znižujú potrebu manuálneho mapovania a konverzie.

- **Fasády** – Slúžia v rámci vrstvy business logiky ako rozhrania zjednodušujúce interakciu s vrstvou DAL. Zefektívňujú komunikáciu medzi aplikačnou vrstvou a samotnou business logikou tým, že konsolidujú a predstavujú súbor vybraných operácií v prístupnejšej a intuitívnejšej forme. Tým umožňujú vyšším vrstvám aplikácie realizovať komplexné business procesy prostredníctvom jednoduchých volaní, čím znižujú komplexitu integrácie a podporujú čistejšiu architektúru a lepšiu udržateľnosť kódu.

- **Inštalátory (Installers)** – Sú súčasťou mechanizmu na konfiguráciu a registráciu komponentov vrstvy business logiky v rámci DI kontajnera. Zabezpečujú dynamickú konfiguráciu služieb a závislostí potrebných pre správne fungovanie BL, čím zjednodušujú správu a rozšírenie aplikácie.

Aplikačná vrstva

Aplikačná vrstva, predstavujúca aplikačné programovacie rozhranie (API), poskytuje rozhranie, prostredníctvom ktorého môže webové rozhranie alebo užívatelia pristupovať k funkcionalitám a dátam poskytovaným webovou aplikáciou. Táto vrstva teda zjednodušuje a štandardizuje spôsob, akým je aplikačné programovacie rozhranie využívané, a zároveň izoluje internú logiku systému od jeho užívateľov. Pre dokumentáciu aplikačného programovacieho rozhrania bola využitá knižnica Swashbuckle.AspNetCore poskytujúce rozhranie Swagger v OpenAPI[34] špecifikácii. Medzi základné komponenty aplikačnej vrstvy patria:

- **Kontroléry (Controllers)** – Sú základnými stavebnými blokmi v rámci API, kde každý kontrolér spracováva prichádzajúce HTTP požiadavky na konkrétnych adresách URL, tzn. na konkrétnych koncových bodoch špecifikovaných v sekcii 5.7.2 a riadia tok dát medzi užívateľským rozhraním a vrstvou business logiky. Kontroléry zodpovedajú za prijímanie požiadaviek, volanie príslušných služieb alebo metód vrstvy business logiky a následné odoslanie odpovedí klientovi.

- **Konfiguračné súbory** – Sú zásadné pre správne nastavenie a konfiguráciu aplikácie, vrátane nastavení databázy, middleware, routovania a iných služieb. Slúžia na centralizované a jednoduché spravovanie konfiguračných nastavení aplikácie, čo zjednodušuje údržbu a nasadzovanie aplikácie v rôznych prostrediach. V rámci implementácie sa konfigurácia nachádza v súbore `appsettings.json` a `Program.cs`.

V rámci rozšírenia aplikačnej vrstvy bola implementovaná aj funkčnosť prihlásenia a registrácie, pričom na tieto účely bola využitá knižnica ASP.NET Identity. Rozšírenie funkcionality si vyžiadalo zmeny v existujúcich koncových bodoch navrhnutých v sekcii 5.7.2 a pridanie nových. Tieto zmeny umožňujú bezpečnejší a efektívnejší prístup k funkcionalitám aplikácie a zároveň poskytujú lepšiu kontrolu nad spracovaním užívateľských dát. Nižšie

[34] OpenAPI: https://www.openapis.org/

je uvedený prehľad aktualizovaných a nových koncových bodov patriacich pod jednotlivé kontroléry.

Auth Kontrolér

Kontrolér `Auth` je implementovaný s použitím knižnice ASP.NET Identity a zaoberá sa spracovaním autentifikačných a autorizačných požiadaviek v aplikácii. V tabuľke 6.6 sú uvedené špecifikované koncové body, ktoré sú využívané na správu užívateľských účtov a prístupových práv. Nie všetky koncové body sú avšak využívané, ale sú zahrnuté pre možnosť rozšírenia funkcionality. Medzi operácie, ktoré kontrolér Auth poskytuje a sú zároveň využité v implementácii, patrí registrácia nových užívateľov, prihlásenie a odhlásenie.

Metóda	**Endpoint**	**Popis**
POST	/Register	Registrácia nového užívateľa.
POST	/Login	Autentifikácia užívateľa a vrátenie prístupového tokenu.
POST	/Refresh	Obnovenie vypršaného prístupového tokenu pomocou obnovovacieho tokenu.
GET	/ConfirmEmail	Potvrdenie e-mailovej adresy užívateľa pomocou kódu.
POST	/ResendConfirmationEmail	Znovu odošle potvrdzovací e-mail.
POST	/ForgotPassword	Iniciuje proces obnovenia hesla pre užívateľa.
POST	/ResetPassword	Obnovenie hesla užívateľa pomocou kódu.
POST	/Manage/2fa	Spravuje nastavenia dvojfaktorovej autentifikácie.
GET	/Manage/info	Získava informácie o účte užívateľa.
POST	/Manage/info	Aktualizuje informácie o účte užívateľa.
POST	/Logout	Odhlási užívateľa a ukončí jeho reláciu.
GET	/Pingauth	Testuje službu autentifikácie na overenie prístupu.

Tabuľka 6.6: Špecifikácie koncových bodov pre kontrolér Auth

Public Kontrolér

Kontrolér `Public` slúži na spracovanie dát pre zobrazovanie štatistík webovým rozhraním a taktiež pre obnovenie hesla v prípade jeho zabudnutia, pričom pre získanie štatistík alebo obnovy hesla užívateľ nemusí byť prihlásený. Z toho dôvodu tu aj boli z kontrolérov Result, Blacklist a Whitelist presunuté koncové body týkajúce sa štatistík. Kontrolér Public špecifikuje koncové body zobrazené v tabuľke 6.7.

Metóda	Koncový bod	Popis
GET	/Public/Blacklist/Count	Vráti počet všetkých domén aktuálne na čiernej listine.
GET	/Public/Whitelist/Count	Vráti počet všetkých domén aktuálne na bielej listine.
GET	/Public/Results/Count	Vráti počet všetkých výsledkov detekcie.
GET	/Public/FilteredByBlacklist	Vráti počet položiek filtrovaných čiernou listinou.
GET	/Public/NumberOfDomainsToday	Vráti počet domén spracovaných dnes.
GET	/Public/PositiveResultsToday	Vráti počet pozitívnych výsledkov detekcie získaných dnes.
POST	/Public/Reset-password	Resetuje heslo užívateľa na základe zodpovedanej bezpečnostnej otázky).
GET	/Public/SecurityQuestion/email	Získa bezpečnostnú otázku prepojenú s daným emailom.

Tabuľka 6.7: Špecifikácie koncových bodov pre kontrolér Public

User Kontrolér

Kontrolér `User` sa zaoberá spracovaním dát týkajúcich sa daného užívateľa. Model predstavujúci užívateľa patrí do knižnice ASP.NET Identity, pričom bola nad týmto modelom implementovaná funkcionalita na správu jednotlivých užívateľov. Kontrolér User špecifikuje koncové body špecifikované v tabuľke 6.8.

Metóda	Koncový bod	Popis
GET	/User/Get	Získa detaily používateľa.
PUT	/User/Update	Aktualizuje detaily používateľa.
POST	/User/Delete/UserId	Odstráni používateľa podľa ID.
POST	/User/Change-password	Zmení heslo používateľa.

Tabuľka 6.8: Špecifikácie koncových bodov pre kontrolér User

6.6.4 Architektúra webového rozhrania

V tejto sekcii bude predstavená architektúra druhej časti tretieho bloku detekčného nástroja, konkrétne sa jedná o architektúru webového rozhrania.

Architektúra webového rozhrania sa skladá z reaktívnych komponentov, ktoré sú dynamicky integrované do tzv. jednostránkovej aplikácie (Single-page application - SPA). Tento prístup zabezpečuje rýchlu odozvu a minimalizuje načítanie stránky, čo je ideálne pre interaktívne prostredie, kde užívatelia vykonávajú časté operácie, ako sú pridávanie, upravovanie alebo mazanie záznamov v reálnom čase.

Štruktúra zložená z komponentov

Komponenty webového rozhrania sú navrhnuté tak, aby boli modulárne a ľahko integrovateľné s rôznymi časťami aplikácie. Každý komponent je zodpovedný za svoju špecifickú

funkcionalitu a spolupracuje s ostatnými komponentmi prostredníctvom predávania vlastností (properties) a využívania kontextov alebo globálneho stavu pre zdieľanie dát. Nižšie budú popísané využité komponenty v rámci aplikácie a taktiež bude zobrazené ich konkrétne umiestnenie v aplikácii.

Základné komponenty

- **App** – Jedná sa o komponent, ktorý zastrešuje celkovú logiku smerovania a integruje všetky podkomponenty do jednotného užívateľského rozhrania, tzn. zabezpečuje, aby boli všetky časti aplikácie správne spojené a fungovali ako koherentný celok.
- **Main** – Inicializuje aplikáciu, nastavuje poskytovateľov stavov a závislostí a renderuje aplikáciu do DOM[35].

Zdieľané komponenty

1. **Button** – Predstavuje komponent, ktorý je využívaný ako tlačidlo. Je možné ho prispôsobiť rôznymi štýlmi a veľkosťami. Jeho použitie je flexibilné a je implementovaný tak, aby vyhovoval rôznym požiadavkám užívateľského rozhrania.
2. **HomePage** – Komponent slúžiaci ako úvodná stránka aplikácie. Je navrhnutý tak, aby poskytoval používateľovi prvú interakciu s aplikáciou a základné informácie o jej funkcionalitách. Na tejto stránke sú zobrazené štatistiky. Okrem zobrazenia dynamického obsahu je na tejto stránke implementované aj animovanie prvkov pomocou knižnice GSAP, čo zvyšuje vizuálnu príťažlivosť a používateľskú interaktivitu.
3. **Login** – Predstavuje komponent určený pre prihlásenie používateľov do aplikácie. Je navrhnutý tak, aby umožňoval jednoduché a bezpečné prihlasovanie prostredníctvom emailu a hesla. Vstupné polia sú validované s použitím knižnice Formik, čo zaručuje, že všetky údaje sú správne zadané pred ich odoslaním na server. Pri úspešnom prihlásení je používateľ presmerovaný na hlavnú stránku, zatiaľ čo pri neúspešnom pokuse je prostredníctvom notifikácie z knižnice react-toastify zobrazená chybová správa. Okrem toho sú v rámci komponentu poskytované odkazy na stránky pre registráciu nových účtov a obnovu zabudnutých hesiel.
4. **Register** – Predstavuje komponent určený pre registráciu nových používateľov do aplikácie. V tomto komponente je možné zadať email, heslo, ako aj odpoveď na bezpečnostnú otázku. Všetky vstupné polia sú dôkladne validované s využitím knižnice Formik, aby bolo zabezpečené, že všetky zadané údaje sú správne a kompletne vyplnené. Po úspešnej registrácii je užívateľ presmerovaný na domovskú stránku. V prípade detekcie chyby pri registrácii je používateľovi zobrazená chybová správa prostredníctvom komponentu react-toastify.
5. **ForgotPassword** – Tento komponent slúži pre obnovu zabudnutých hesiel používateľov. Po zadaní emailovej adresy je automaticky odoslaný požiadavok na server pre získanie bezpečnostnej otázky, ktorá bola nastavená pri registrácii užívateľa. Po správnej odpovedi na bezpečnostnú otázku je používateľovi umožnené zadať a potvrdiť nové heslo. Všetky vstupné polia sú dôkladne validované pomocou knižnice Formik, čím je zabezpečené, že nové heslo spĺňa všetky bezpečnostné požiadavky. Pri

[35] Document Object Model

úspešnom nastavení nového hesla je užívateľovi prostredníctvom komponentu react-toastify zobrazená notifikácia o úspešnej zmene hesla. V prípade detekcie chýb pri obnove hesla je užívateľovi zobrazená chybová správa.

6. **NavBar** – Tento komponent je navrhnutý na zobrazovanie navigačnej lišty aplikácie, ktorá je zásadnou súčasťou užívateľského rozhrania. V závislosti od stavu autentifikácie používateľa je tento komponent dynamicky zobrazovaný, tzn. neautentifikovaným používateľom je skrytý, zatiaľ čo autentifikovaní používatelia majú prístup k svojmu profilu, výsledkom detekcie a obsahom zoznamov Blacklist a Whitelist. Animácie navigačnej lišty sú implementované s využitím knižnice GSAP, ktorá zaisťuje plynulé a atraktívne vizuálne efekty. Okrem toho je v NavBar integrované tlačidlo na odhlásenie, ktoré je zobrazené iba autentifikovaným používateľom a umožňuje bezpečné a rýchle odhlásenie z aplikácie.

7. **DetectionResults** – Tento komponent je navrhnutý na zobrazovanie výsledkov detekcie v tabuľkovom formáte. Umožňuje pridanie nových výsledkov, aktualizáciu existujúcich, alebo vymazanie podľa potreby užívateľa. Filtrácia výsledkov je podporovaná na základe rôznych kritérií, čím sa zvyšuje efektivita spracovania dát. Stránkovanie je implementované pre lepšiu navigáciu cez veľké množstvá dát, čo používateľom umožňuje prehľadnejšie a efektívnejšie prezeranie výsledkov.

8. **Add** – Tento komponent je navrhnutý na pridanie nových údajov prostredníctvom formulára. Používateľom je umožnené vkladať údaje, ktoré sú následne spracované a ukladané do databázy. Implementácia využíva knižnicu Formik, čo zabezpečuje efektívne spravovanie stavu formulára a jeho validáciu. Pridávanie údajov je zabezpečené tak, aby bolo intuitívne a užívateľsky prívetivé. Chyby vo formulári sú zobrazované v reálnom čase, čím je používateľom umožnené ich okamžité opravenie. Každý úspešný príspevok dát je potvrdený prostredníctvom notifikácie, ktorá je generovaná knižnicou react-toastify, zatiaľ čo pri neúspešnom pokuse o odoslanie dát je používateľ upozornený na chybu.

9. **Update** – Tento komponent je navrhnutý na aktualizáciu existujúcich údajov v aplikácii. Umožňuje používateľom upravovať údaje prostredníctvom formulára, ktorý je dynamicky prispôsobený podľa typu dát, ktoré sú aktualizované. Aktualizácia údajov je spravovaná pomocou knižnice Formik, čo zabezpečuje, že každý údaj je validovaný pred jeho odoslaním na server. V prípade úspešnej aktualizácie dát je používateľovi zobrazená potvrdzujúca notifikácia od knižnice react-toastify, zatiaľ čo pri detekcii chyby pri aktualizácii je zobrazená chybová správa.

10. **Table** – Tento komponent je navrhnutý na zobrazenie dát v tabuľkovej forme, pričom je kladený dôraz na flexibilitu a užívateľskú prívetivosť. Dáta sú zobrazované na základe poskytnutých vstupov a môžu byť zoradené podľa rôznych stĺpcov v stúpajúcom alebo klesajúcom poradí. Interakcie s tabuľkou, ako sú aktualizácia, vymazanie alebo presunutie riadkov, sú implementované tak, aby vyhovovali potrebám užívateľov. Celkové rozhranie komponentu je navrhnuté tak, aby bolo čisté, moderné a zároveň efektívne z hľadiska načítania a spracovania dát, čo zabezpečuje vynikajúce používateľské skúsenosti pri práci s veľkými množstvami informácií.

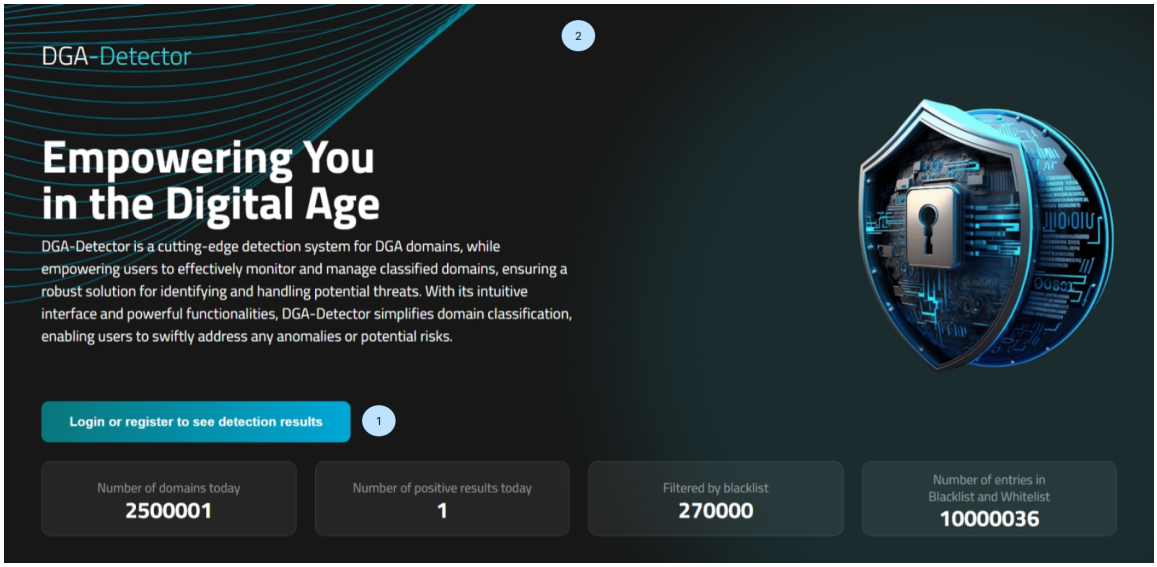
11. **Blacklist** – Tento komponent slúži na správu zoznamu Blacklist v aplikácii. Umožňuje pridávanie, aktualizáciu a taktiež odstraňovanie položiek. Funkcionalita tohto

komponentu je implementovaná tak, aby zabezpečovala efektívnu interakciu s databázou prostredníctvom volaní aplikačného programovacieho rozhrania. Rozhranie užívateľovi poskytuje možnosť filtrovania záznamov, čo uľahčuje nájdenie konkrétnych položiek. Interakcie sú zjednodušené pomocou intuitívneho užívateľského rozhrania, ktoré zahrňuje vizuálne prvky a animácie implementované s využitím knižnice GSAP. Tento komponent tiež zahŕňa stránkovanie, ktoré umožňuje zobrazenie obmedzeného počtu položiek na stránku, čím sa znižuje preťaženie užívateľského rozhrania pri zobrazení veľkého množstva údajov. Notifikácie o pridávaní alebo odstraňovaní položiek sú generované prostredníctvom knižnice react-toastify, čo zlepšuje komunikáciu s užívateľom pri správe čiernej listiny.

12. **Profile** – Tento komponent je navrhnutý na zobrazenie a aktualizáciu informácií o užívateľovi aplikácie. Informácie, ako sú meno, email, telefónne číslo a profilová fotografia, sú zobrazené v užívateľsky prívetivom rozhraní. Je poskytnutý formulár na aktualizáciu údajov, pričom je využívaná knižnica Formik na validáciu a správu stavu formulára. Notifikácie o úspešnej zmene údajov, alebo o detekcii chýb pri aktualizácii sú zobrazené prostredníctvom komponentu react-toastify.

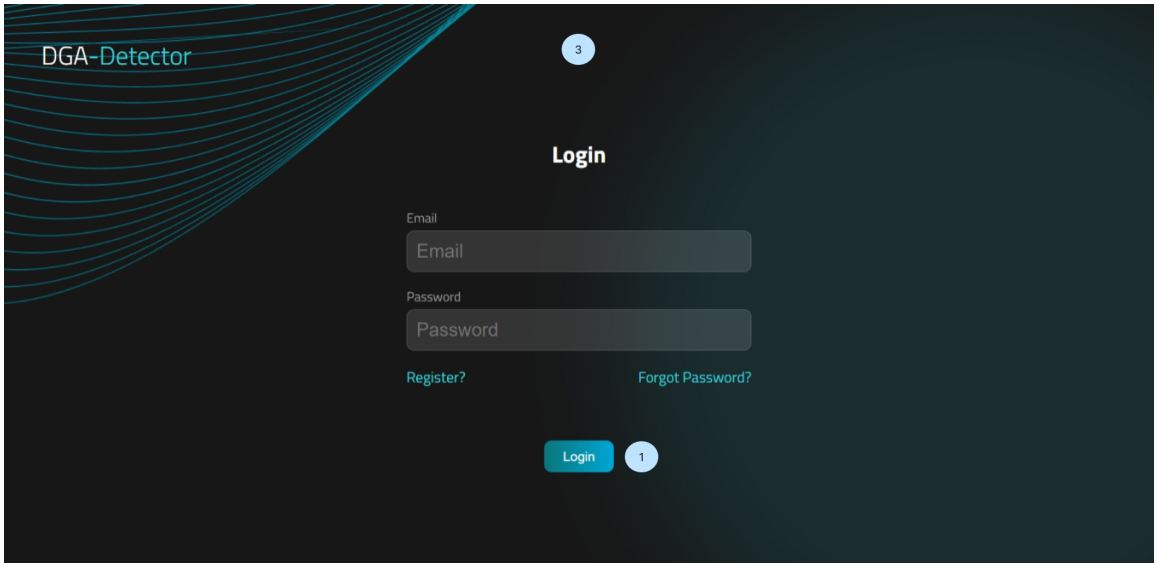
Využitie komponentov v aplikácii

Obrázok 6.9 predstavuje domovskú stránku aplikácie z pohľadu neprihláseného užívateľa, konkrétne sa jedná v rámci implementácie o komponent `HomePage`, označený číslom 2. Na obrazovke je taktiež zobrazený komponent `Button` označený číslom 1, ktorý umožňuje odnavigovanie sa na funkcionalitu prihlásenia.



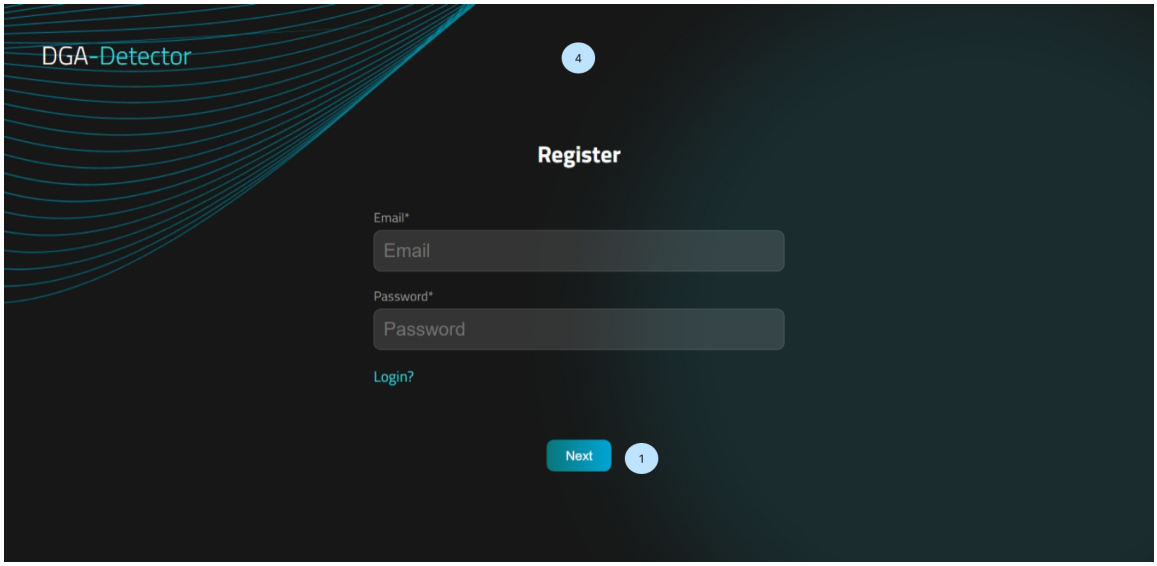
Obr. 6.9: Domovská stránka z pohľadu neprihláseného užívateľa

Funkcionalita prihlásenia, zobrazená na obrázku 6.10, predstavuje komponent Login, označený číslom 3. Obsahuje rovnako ako predchádzajúca domovská stránka komponent `Button` označený číslom 1, slúžiaci na potvrdenie údajov prihlásenia a taktiež odkazy na registráciu a obnovu zabudnutého hesla.



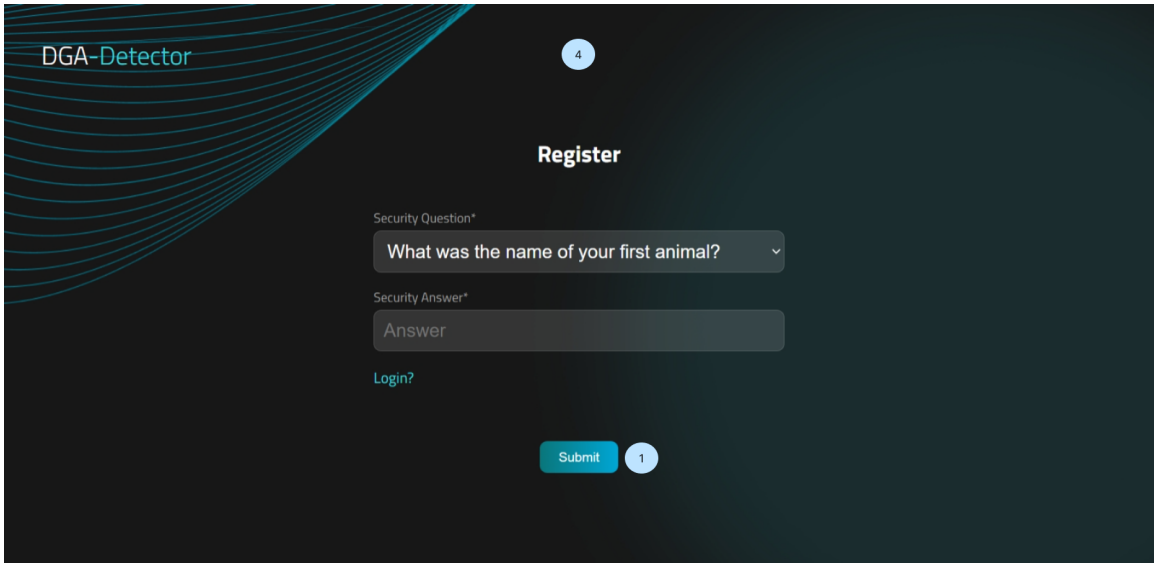
Obr. 6.10: Pohľad na stránku prihlasovania

Obrázok 6.11 predstavuje stránku zodpovednú za registráciu užívateľa, konkrétne sa jedná o komponent `Register` s číslom 4. Pre pokračovanie registrácie je nutné zadať údaje a následne potvrdiť využitím komponentu `Button` s číslom 1, ktorý užívateľa presmeruje na druhý krok registrácie.



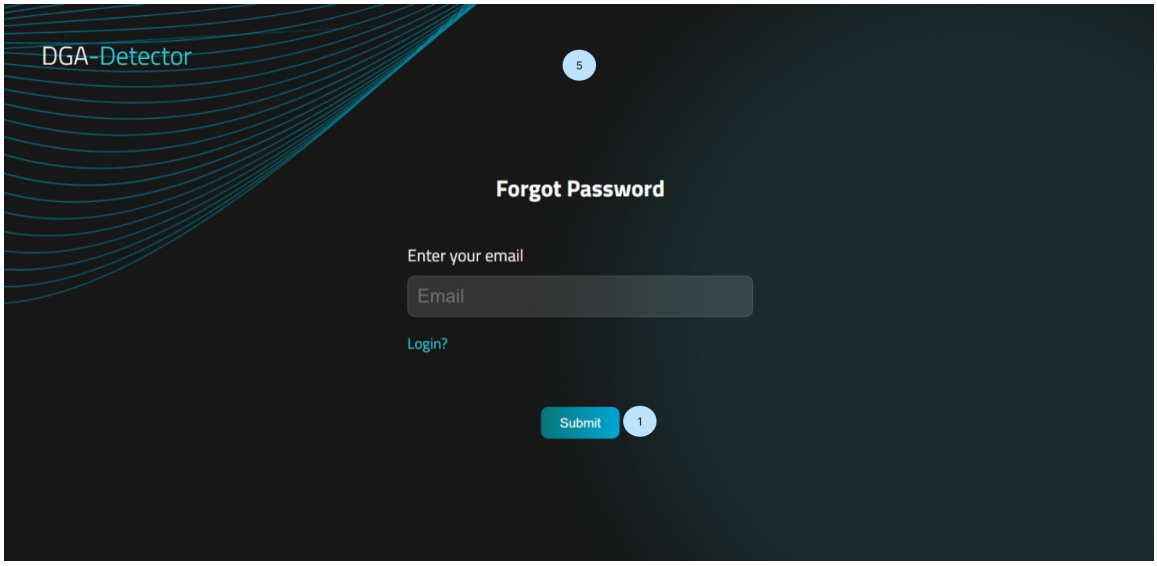
Obr. 6.11: Pohľad na stránku registrácie - zadávanie údajov

Pohľad na stránku registrácie predstavujúcu vyplňovanie bezpečnostnej otázky je zobrazený na obrázku 6.12. Jedná sa rovnako ako v predchádzajúcom príklade o komponent `Register` s číslom 4. Po vyplnení bezpečnostnej otázky je užívateľ úspešne registrovaný a presmerovaný na domovskú stránku, avšak už ako prihlásený užívateľ.



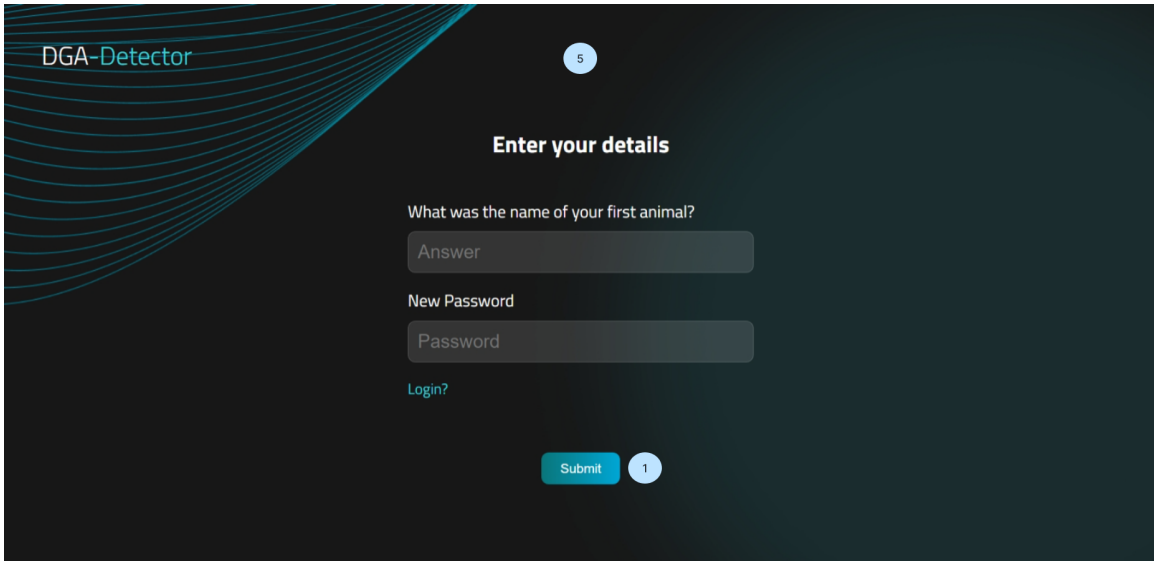
Obr. 6.12: Pohľad na stránku registrácie - vyplňovanie bezpečnostnej otázky

Obrázok 6.13 predstavuje pohľad na stránku poskytujúcu obnovu hesla. Jedná sa o komponent `ForgotPassword` s číslom 5. Na obrazovke je taktiež zobrazený komponent `Button` označený číslom 1, ktorý umožňuje pokračovanie na druhý krok obnovy hesla po zadaní údajov, tzn. zodpovedanie bezpečnostnej otázky a nastavenie nového hesla.



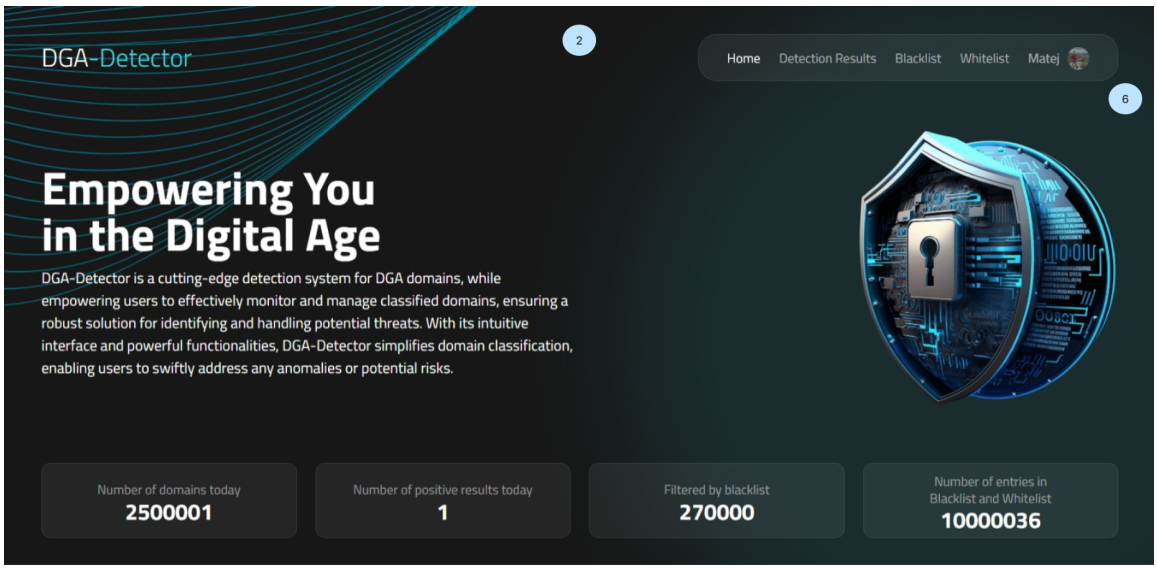
Obr. 6.13: Pohľad na stránku obnovy hesla – vyplňovanie údajov

Druhý krok obnovy hesla, zobrazený na obrázku 6.14, umožňuje zodpovedanie bezpečnostnej otázky a nastavenie nového hesla. Tento proces je súčasťou komponentu s číslom 5, `ForgotPassword`, ktorý bol predstavený vyššie. Zobrazený komponent `Button` označený číslom 1 poskytuje možnosť potvrdenia obnovy hesla a následné presmerovanie na stránku prihlásenia.



Obr. 6.14: Pohľad na stránku obnovy hesla – zodpovedanie bezpečnostnej otázky

Obrázok 6.15 predstavuje domovskú stránku aplikácie z pohľadu prihláseného užívateľa, konkrétne sa jedná v rámci implementácie o komponent `HomePage`, označený číslom 2. Na obrazovke je taktiež zobrazený komponent `NavBar` s číslom 6, ktorý zabezpečuje navigáciu prihláseným užívateľom.

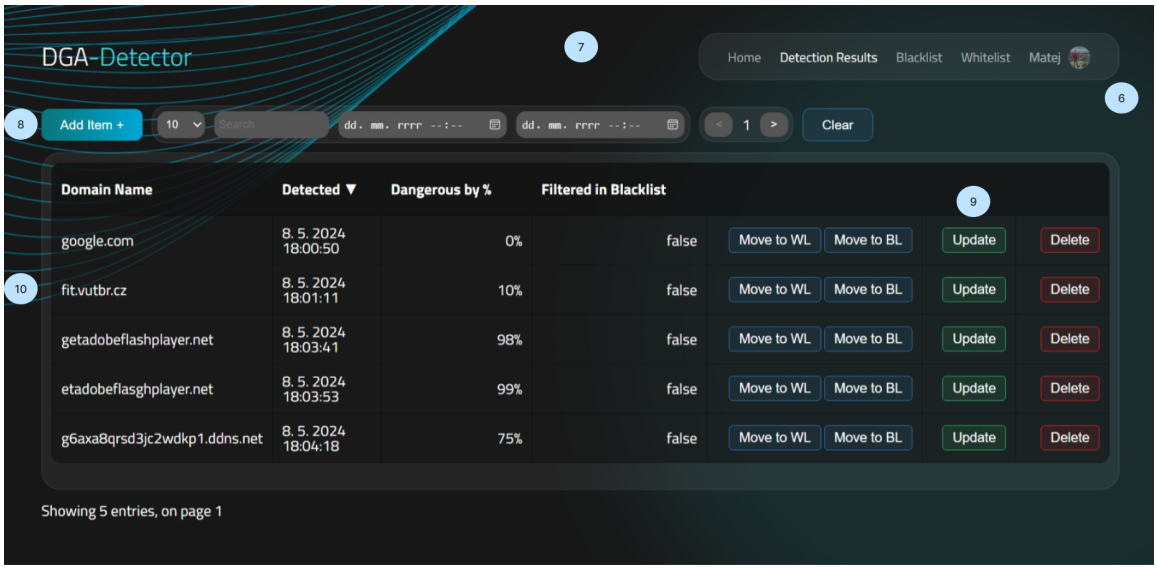


Obr. 6.15: Pohľad na domovskú stránku ako prihlásený užívateľ

Pohľad na stránku predstavujúcu zoznam výsledkov detekcie je zobrazený na obrázku 6.16. Jedná sa o komponent číslo 7 s názvom `DetectionResults`. Obsahuje taktiež nasledovné komponenty:

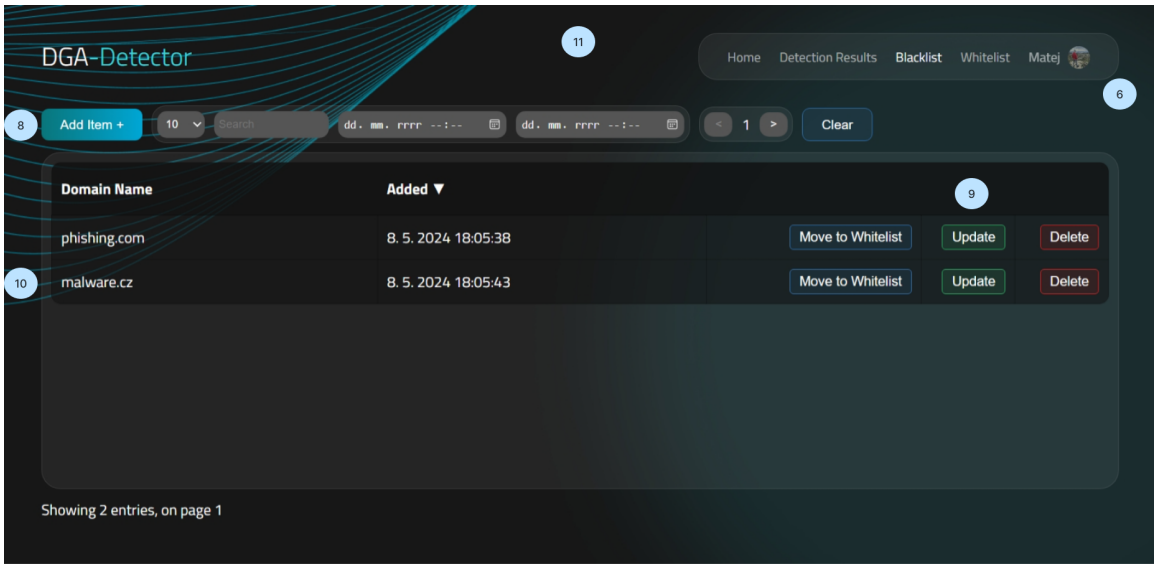
- **NavBar** – Zabezpečuje navigáciu prihláseným užívateľom, je označený číslom 6.
- **Add** – Umožňuje bezpečné manuálne pridanie nových záznamov, na obrázku je možné ho vyhľadať pod číslom 8.

- **Update** – Poskytuje možnosť bezpečnej úpravy aktuálnych záznamov v rámci výsledkov detekcie. Komponent je označený číslom 9.
- **Table** – Predstavuje zobrazenie jednotlivých výsledkov detekcie. Je možné ho vyhľadať pod číslom 10.



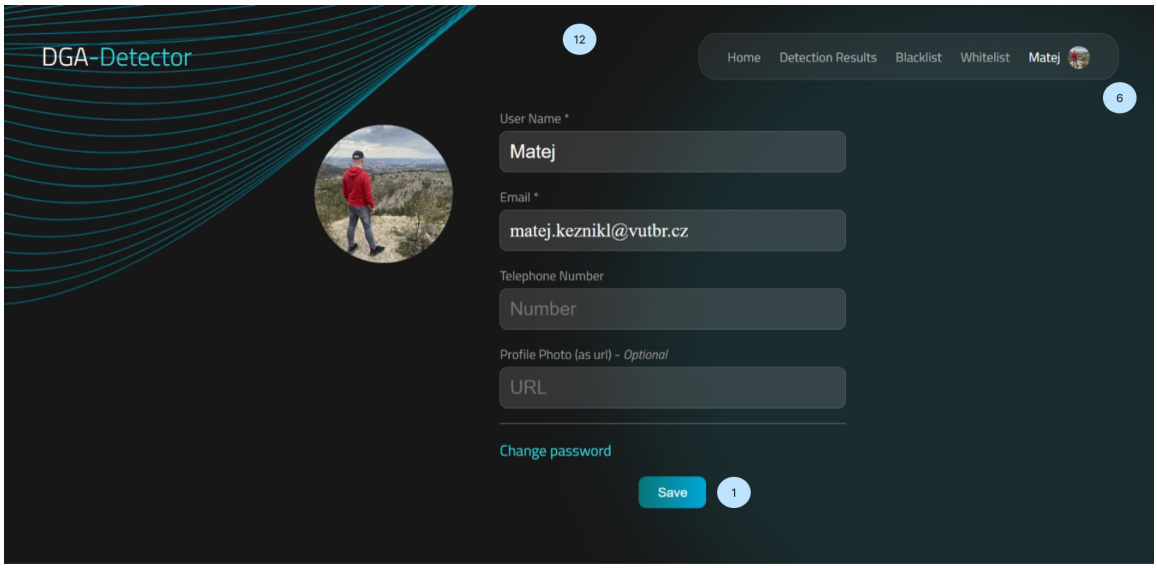
Obr. 6.16: Pohľad na stránku zobrazujúcu výsledky detekcie

Obrázok 6.17 predstavuje stránku zodpovednú za zobrazovanie zoznamu Blacklist, označeného číslom 11. Jedná sa o komponent Blacklist a obsahuje rovnaké komponenty ako komponent DetectionResults, avšak s jediným rozdielom v podobe menšieho počtu stĺpcov v tabuľke. Zoznam Whitelist predstavuje identickú funkcionalitu ako zoznam Blacklist a z toho dôvodu nie je predvedený i komponent Whitelist



Obr. 6.17: Pohľad na stránku predstavujúcu zoznam Blacklist

Obrázok 6.18 predstavuje užívateľský profil, konkrétne sa jedná v rámci implementácie o komponent `Profile`, označený číslom 12. Na obrazovke je zobrazený komponent `NavBar` s číslom 6, ktorý zabezpečuje navigáciu prihláseným užívateľom a taktiež komponent `Button` s číslom 1, slúžiaci na potvrdenie zmeny údajov užívateľa.



Obr. 6.18: Pohľad na stránku predstavujúcu užívateľský profil

Kapitola 7

Hodnotenie a selekcia klasifikátorov

Predmetom tejto kapitoly je ohodnotenie viacerých klasifikačných metód špecifikovaných v sekcii 5.6.2. V rámci tejto kapitoly budú porovnané z hľadiska schopnosti efektívne rozoznať medzi DGA a benígnymi doménami, rýchlosti trénovania a predikcie, ako aj podľa výpočtovej náročnosti.

7.1 Príprava dát

V rámci prípravy dát pre účely hodnotenia klasifikátorov bolo nutné pripraviť dáta do vhodného formátu, tzn. aby zodpovedali požadovaným špecifikáciám pre spracovanie. V prvom kroku boli zo súborov Parquet[1] obsahujúcich už extrahované vlastnosti (špecifikované v sekcii 5.6.1), odstránené nepotrebné stĺpce, ako sú napr. časové značky. Tieto operácie boli vykonané pomocou knižnice Pandas[2], ktorá umožňuje efektívne manipulovať s dátami uloženými v dátovej štruktúre `dataframe`.

Následne bolo nevyhnutné vyriešiť problém s chýbajúcimi hodnotami. Chýbajúce hodnoty (NaN[3]) boli v dátovom rámci identifikované a nahradené hodnotou -1, čo pomáha zabezpečiť konzistentnosť dátových sád pre analytické účely.

Po úprave a očistení dát bol nasledujúci krok rozdelenie dátového súboru na trénovaciu a testovaciu množinu. Tento krok je nevyhnutný pre validáciu efektívnosti klasifikátorov v simulovaných reálnych podmienkach. Dáta boli rozdelené v pomere 70% ku 30%, kde 70% dát bolo vyhradených pre trénovaciu množinu a zvyšných 30% pre testovaciu množinu. Toto rozdelenie zabezpečuje objektívne hodnotenie efektívnosti modelov a umožňuje overiť, či sú modely schopné generalizácie na nové, doposiaľ nevidené dáta.

Toto rozdelenie sa vykonalo pomocou funkcií knižnice Scikit-learn[4], konkrétne metódou `train_test_split`. Táto metóda umožňuje náhodné rozdelenie dát tak, aby obidve množiny boli reprezentatívne a obsahovali rozmanité prípady z celého dátového súboru. Udržiavanie náhodnosti v rozdelení je kľúčové pre zabezpečenie, že trénovacie algoritmy nebudú zaujaté alebo pretrénované na špecifické vzory, ktoré by mohli byť dominantné v nevhodne rozdelených dátach.

[1] Parquet: https://parquet.apache.org/
[2] Pandas: https://pandas.pydata.org/
[3] Not a Number
[4] Scikit-learn: https://scikit-learn.org/

Po rozdelení boli trénovacie a testovacie dáta starostlivo skontrolované, aby sa potvrdila ich integrita a príslušnosť k stanoveným kritériám.

7.2 Metriky využité na porovnanie klasifikátorov

Pri hodnotení efektivity klasifikátorov budú využívané rôzne metriky, ktoré sú zásadné pre meranie presnosti a účinnosti modelov. Je nevyhnutné zahrnúť viaceré metriky, aby bolo možné komplexne posúdiť, ako modely reagujú na rozličné situácie. Každá metrika poskytuje unikátny pohľad na rôzne aspekty výkonu modelu, čím umožňuje odhaliť potenciálne slabiny, ktoré by mohli byť pri použití len jedinej metriky prehliadnuté. Správny výber metrík je kritický pre zabezpečenie, že modely dokážu fungovať efektívne a sú schopné správne predpovedať výsledky.

Predtým ako budú predstavené metriky na hodnotenie klasifikátorov, je potrebné definovať základné pojmy, ktoré sú používané pri výpočte týchto metrík. Tieto pojmy zahŕňajú:

- **TP (z anglického True Positive)**: Počet správne klasifikovaných pozitívnych príkladov, tj. DGA domény správne identifikované ako DGA.
- **TN (z anglického True Negative)**: Počet správne klasifikovaných negatívnych príkladov, tj. benígne domény správne identifikované ako benígne.
- **FP (z anglického False Positive)**: Počet nesprávne klasifikovaných negatívnych príkladov, tj. benígne domény nesprávne identifikované ako DGA.
- **FN (z anglického False Negative)**: Počet nesprávne klasifikovaných pozitívnych príkladov, tj. DGA domény nesprávne identifikované ako benígne.

Metriky, vybrané pre túto analýzu, boli odpozorované v rámci odbornej literatúry v [22, 28], pričom majú za cieľ poskytnúť nestranný a realistický pohľad na schopnosti modelov a zároveň identifikovať ich prednosti a nedostatky. Jedná sa konkrétne o metriky:

- **Accuracy** – Meria pomer správne klasifikovaných prípadov (tj. DGA a benígne domény) k celkovému počtu prípadov. Hodnotu metriky je možné vypočítať podľa vzorca 7.1.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{7.1}$$

- **Precision** – Meria podiel správne identifikovaných pozitívnych prípadov (DGA domény) z celkového počtu prípadov, ktoré boli označené ako pozitívne. Je užitočná pre posúdenie spoľahlivosti modelu pri klasifikácii domén DGA. Výpočet metriky je možné vidieť na rovnici 7.2.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{7.2}$$

- **Recall** – Táto metrika zohľadňuje schopnosť modelu identifikovať všetky pozitívne prípady (DGA domény) z celkového množstva pozitívnych prípadov. Hodnotu metriky je možné získať z vzorca 7.3.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{7.3}$$

- **FPR (z anglického False Positive Ratio)** – Meria pomer nesprávne identifikovaných negatívnych prípadov (benígne domény) k celkovému počtu negatívnych prípadov v testovacej množine. Vysoká hodnota FPR môže naznačovať, že model je príliš agresívny v identifikácii pozitívnych príkladov, čo vedie k vysokému počtu falošných pozitív. Táto metrika je dôležitá pre vyváženie presnosti a spoľahlivosti modelu. Hodnotu FPR je možné vypočítať podľa vzorca 7.4.

$$\text{FPR} = \frac{FP}{FP + TN} \tag{7.4}$$

- **F1 Score** – Predstavuje harmonický priemer metrík Precision a Recall. Využíva sa v situáciách, kedy je potrebná vyváženosť medzi metrikami Precision a Recall, ako napr. pri klasifikácii DGA domén, kde prehliadnutie aktívneho DGA môže mať vážne bezpečnostné dôsledky. F1 Score poskytuje robustnejšie meranie celkového výkonu klasifikátora, obzvlášť v nevyvážených dátových sadách, kde môže byť zavádzajúce používať samotné metriky Precision alebo Recall. Hodnotu metriky je možné vypočítať podľa vzorca 7.5.

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{7.5}$$

- **ROC-AUC (Receiver Operating Characteristic - Area Under Curve)** – Grafické znázornenie výkonu klasifikátora pri rôznych prahových hodnotách. AUC hodnota predstavuje mieru, ako dobre model rozlišuje medzi pozitívnymi (DGA domény) a negatívnymi (benígne domény) prípadmi. Čím vyššia je hodnota AUC, tým lepšie model rozlišuje medzi danými dvoma skupinami. Výpočet metriky je možné vidieť na rovnici 7.6.

$$\text{AUC} = \int_0^1 F_0(F_1^{-1}(v))dv \tag{7.6}$$

  Kde:

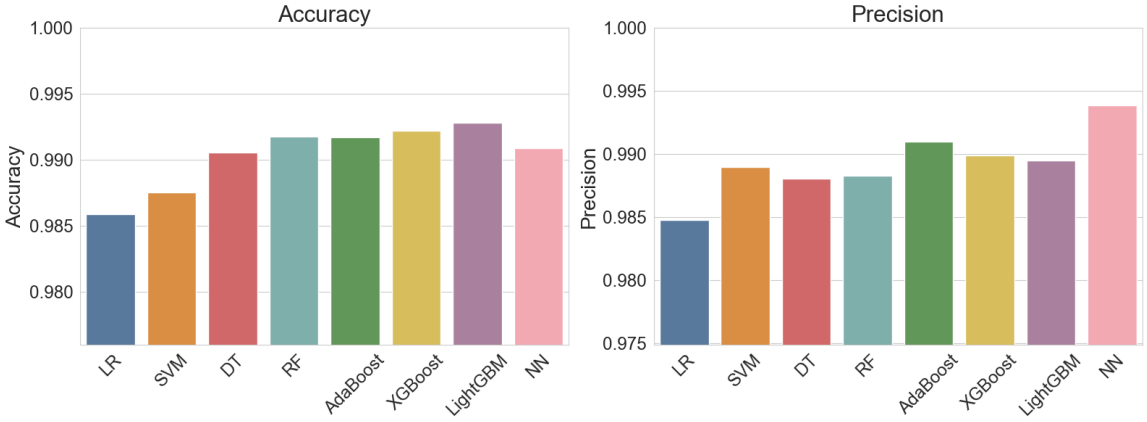
  - $F_0(t)$ – Miera pravdivých pozitív (Recall) pri prahovej hodnote t.
  - $F_1(t)$ – Miera falošných pozitív (False Positive Ratio) pri prahovej hodnote t.

- **Čas trénovania (Training Time)** – Meria celkový čas potrebný na vybudovanie modelu klasifikátora. Zahŕňa čas potrebný na učenie modelu a optimalizáciu parametrov. Táto metrika je dôležitá pre porovnanie rôznych klasifikátorov z hľadiska ich efektivity pri trénovaní, čo môže mať významný vplyv na výber modelu v praxi, najmä v prípadoch, kde je potrebné často aktualizovať model na základe nových dát.

- **Čas predikcie (Prediction Time)** – Udáva čas potrebný na vyhodnotenie nových vstupov pomocou natrénovaného modelu. Tento čas je kritický v aplikáciách, kde je dôležitá rýchla odozva, ako sú systémy, ktoré vyžadujú okamžité rozhodovanie založené na predikciách. Čas predikcie závisí od komplexity modelu a od efektivity algoritmu použitého pri predikcii.

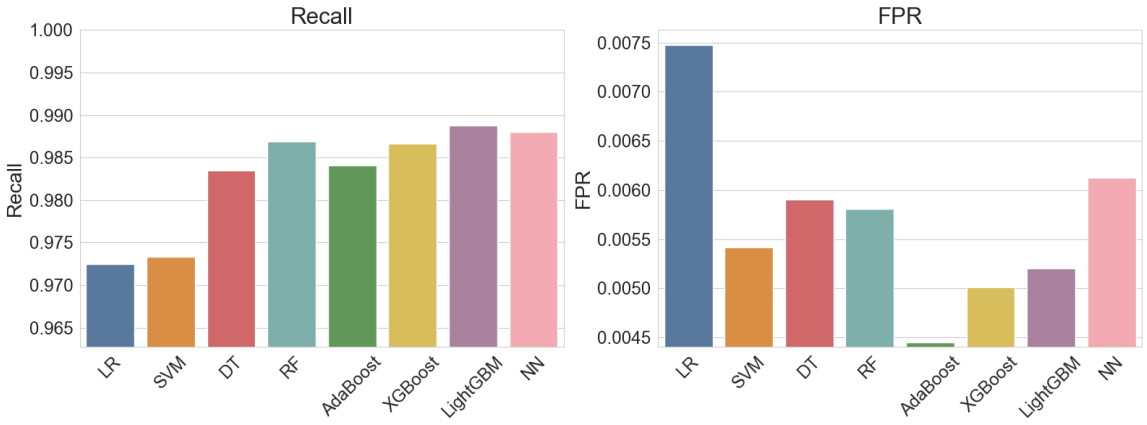
7.3 Porovnanie klasifikátorov

Predmetom tejto sekcie je porovnanie a analýza klasifikačných metód špecifikovaných v sekcii 5.6.2. Cieľom porovnania je identifikovať klasifikátor alebo klasifikátory, ktoré preukážu najlepší výkon na základe vopred definovaných metrík uvedených v sekcii 7.2. Porovnanie bude vykonávané v rámci 5 behov, kde každý beh bude mať nastavenú inú počiatočnú hodnotu (seed). Po vykonaní všetkých 5 behov budú jednotlivé hodnoty spriemerované a následne zobrazené v grafoch jednotlivých metrík.



Obr. 7.1: Metriky Accuracy a Precision jednotlivých klasifikátorov

Na základe predloženého grafu zobrazeného na obrázku 7.1 je možné konštatovať, že všetky porovnávané klasifikátory dosahujú vysokú úroveň metrík Accuracy a Precision. Pokiaľ sa jedná o metriku Accuracy, všetky klasifikátory okrem Logistickej regresie a SVM dosiahli približne rovnaké výsledky. Avšak, v rámci metriky Precision výraznejšie exceluje klasifikátor Neurónová sieť, tzn. najsprávnejšie identifikuje DGA doménové mená zo všetkých klasifikátorov v pomere k celkovému počtu pozitívnych prípadov.

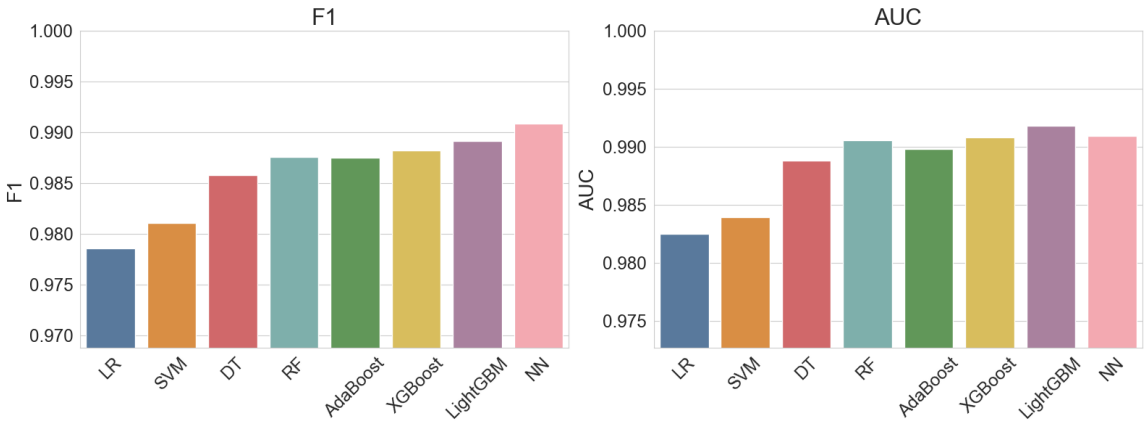


Obr. 7.2: Metriky Recall a FPR jednotlivých klasifikátorov

Z grafov zobrazených na obrázku 7.2 je možno konštatovať, že všetky porovnávané klasifikátory, okrem Logistickej regresie a SVM dosahujú vysoké hodnoty pre metriku Recall. Znamená to, že skoro všetky vybrané klasifikátory sú schopné správne identifikovať pozi-

tívne prípady. Najlepšie výsledky v tejto metrike dosahuje klasifikátor LightGBM, avšak rozdiel oproti klasifikátoru Neurónová sieť je minimálny.

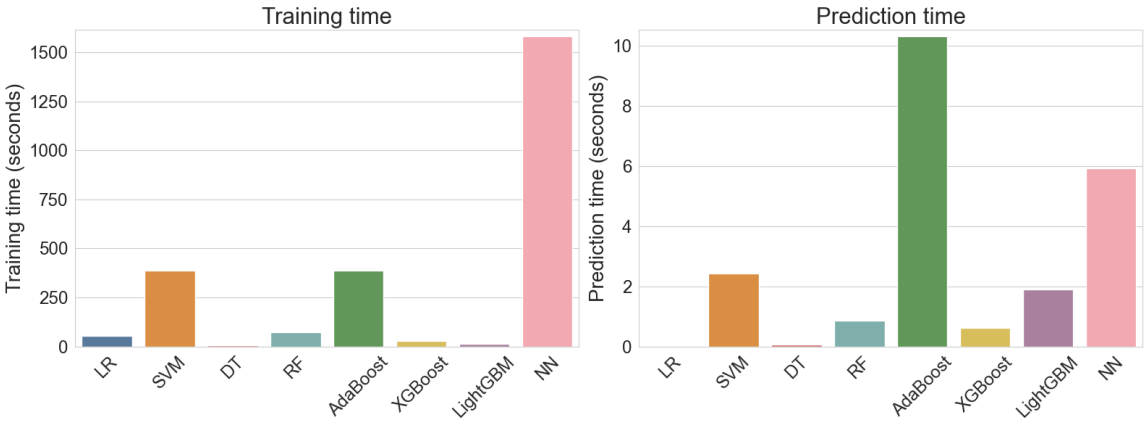
V prípade metriky FPR možno vidieť, že klasifikátory majú tendenciu vykazovať nízke hodnoty FPR. To značí, že klasifikátory majú nízku pravdepodobnosť klamlivého označenia negatívneho prípadu za pozitívny. Najnižšiu hodnotu FPR preukazuje klasifikátor AdaBoost, nasledovaný XGBoost a LightGBM, čo z nich robí klasifikátory s najlepším výkonom v tejto metrike v porovnaní s ostatnými.



Obr. 7.3: Metriky F1 a AUC jednotlivých klasifikátorov

Klasifikátory sú taktiež hodnotené na základe metrík F1 a AUC. Z grafu 7.3 je zjavné, že všetky klasifikátory, okrem Logistickej regresie a SVM preukázali vysoké hodnoty v oboch metrikách, čo signalizuje ich efektívnosť v klasifikácii. F1 skóre kombinuje metriky Accuracy a Recall, čím poskytuje vyvážený pohľad na výkonnosť klasifikátora. AUC skóre poskytuje celkovú mieru schopnosti klasifikátora rozlišovať medzi jednotlivými triedami.

Neurónové siete a LightGBM sa vyznačujú najvyšším F1 skóre, čo naznačuje ich kompetentnosť v správnej klasifikácii pozitívnych prípadov v porovnaní s celkovým počtom predpovedaných pozitívnych prípadov. Tieto modely sú taktiež na popredných miestach z hľadiska AUC, čo dokazuje ich konzistentnosť v predikcii pravdepodobnosti príslušnosti k pozitívnej triede.



Obr. 7.4: Metriky čas trénovania a čas predikcie jednotlivých klasifikátorov

Na obrázku 7.4 je zobrazený graf, ktorý ukazuje dve kľúčové metriky pre hodnotenie výkonu klasifikátorov: čas potrebný na tréning modelu a čas potrebný na predikciu. Z grafu je vidieť, že medzi rôznymi typmi klasifikátorov existujú významné rozdiely v čase potrebnom na tréning i predikciu.

Pri pohľade na metriku času trénovania, Neurónová sieť (NN) vyžaduje oveľa viac času na tréning v porovnaní s ostatnými klasifikátormi, pričom hodnota prekračuje 1500 sekúnd, t.j. 25 minút.

V rámci metriky času predikcie je znázornený obrázok, kde Neurónová sieť a AdaBoost zaberajú výrazne viac času na vykonanie predikcie v porovnaní s ostatnými modelmi. Najdlhší čas predikcie má klasifikátor AdaBoost, čo možno považovať za nevýhodu v real-time aplikáciách alebo systémoch vyžadujúcich rýchle reakcie ako je aj napr. detekčný systém, ktorým sa táto práca zaoberá. Na druhej strane, klasifikátory ako Logistic Regression, Decision Tree a Random Forrest sú schopné vykonať predikcie veľmi rýchlo, čo ich robí vhodnými pre aplikácie, ktoré vyžadujú okamžité výsledky.

V predchádzajúcej analýze bolo preukázané, že klasifikátory Neurónová sieť a LightGBM vykazujú najlepší výkon v porovnaní s ostatnými modelmi na základe viacerých metrík. Na základe dosiahnutých výsledkov boli tieto dva modely identifikované ako najefektívnejšie v klasifikácii DGA doménových mien.

V nasledujúcej sekcii budú parametre týchto klasifikátorov ladené s cieľom ďalšieho zvýšenia ich výkonnosti. Ladenie parametrov bude realizované s dôrazom na metriky Precision, Recall, AUC a F1. Táto fáza bude zahrňovať detailné nastavenie hyperparametrov a experimentovanie s rôznymi konfiguráciami, aby sa dosiahli optimálne výsledky pre oba modely.

7.4 Ladenie parametrov najlepších klasifikátorov

Ladenie parametrov klasifikátorov predstavuje kritickú fázu v procese vývoja klasifikačných modelov, pričom cieľom je optimalizácia výkonnosti modelu na základe zvolených metrík. Proces ladenia zahŕňa systematické vyhľadávanie, testovanie a výber najlepších hodnôt parametrov, ktoré ovplyvňujú schopnosť modelu naučiť sa a generalizovať na základe dostupných dát. Efektívne nastavené parametre môžu výrazne zlepšiť hodnoty metrík, ktoré sú dôležité pre klasifikáciu doménových mien založených na DGA. Táto sekcia popisuje ladenie kľúčových parametrov vybraných klasifikačných modelov a metódy ich optimalizácie, aby boli dosiahnuté najlepšie možné výsledky.

7.4.1 LightGBM

V tejto sekcii bude predstavené ladenie parametrov klasifikátora strojového učenia LightGBM, pričom bude uskutočňované prostredníctvom nastavenia nasledovných kľúčových parametrov:

- **objective** – Tento parameter špecifikuje cieľovú funkciu, ktorá je optimalizovaná počas trénovania.
- **boosting_type** – Typ boosting algoritmu použitý na vytváranie modelu.
- **min_child_samples** – Minimálny počet vzoriek (alebo pozorovaní) vyžadovaný v listoch stromu. Tento parameter sa používa na kontrolu pretrénovania tým, že zabráni vytvoreniu príliš špecifických pravidiel.

- **colsample_bytree** – Tento parameter určuje podiel príznakov vybraných na trénovanie každého stromu. Hodnota 1 znamená, že sa používajú všetky príznaky.
- **reg_lambda** – Predstavuje regularizačný člen pre L2 regularizáciu. Pomáha predchádzať pretrénovaniu tým, že penalizuje príliš veľké váhy a podporuje hladšie modely.
- **subsample** – Podiel trénovacích dát použitých na trénovanie každého stromu. Subsampling pomáha zvýšiť rýchlosť tréningu a znižuje riziko pretrénovania.
- **subsample_freq** – Frekvencia vykonávania subsample. Napríklad hodnota 1 znamená, že subsample sa vykoná pri každej iterácii.
- **subsample_for_bin** – Počet dátových bodov, ktoré sa majú vzorkovať pre konštrukciu histogramov. Tento parameter môže ovplyvniť rýchlosť trénovania a presnosť.
- **min_split_gain** – Minimálny zisk, ktorý musí byť dosiahnutý, aby sa vykonalo ďalšie rozdelenie v rámci stromu. Tento parameter pomáha kontrolovať vytváranie nových uzlov v strome.
- **n_estimators** – Počet stromov, ktoré majú byť vybudované v modeli. Viac stromov môže viesť k lepšej presnosti, ale taktiež môže viesť k dlhšiemu času trénovania.
- **max_depth** – Maximálna hĺbka každého stromu. Ovplyvňuje komplexnosť modelu a schopnosť zachytiť viac informácií z dát, ale môže viesť k pretrénovaniu.
- **num_leaves** – Maximálny počet listov v strome. Tento parameter ovplyvňuje komplexnosť modelu a môže viesť k jeho väčšej presnosti ale i pretrénovaniu.
- **learning_rate** – Tento parameter kontroluje, ako rýchlo sa daný model učí. Nižšie hodnoty môžu spomaliť trénovanie, avšak môžu zlepšiť konečnú presnosť modelu tým, že mu umožnia prispôsobovať sa viac dátam.
- **scale_pos_weight** – Predstavuje váhy pre pozitívne prípady v rámci binárnej klasifikácie. Tento parameter je užitočný v nevyvážených dátových sadách.

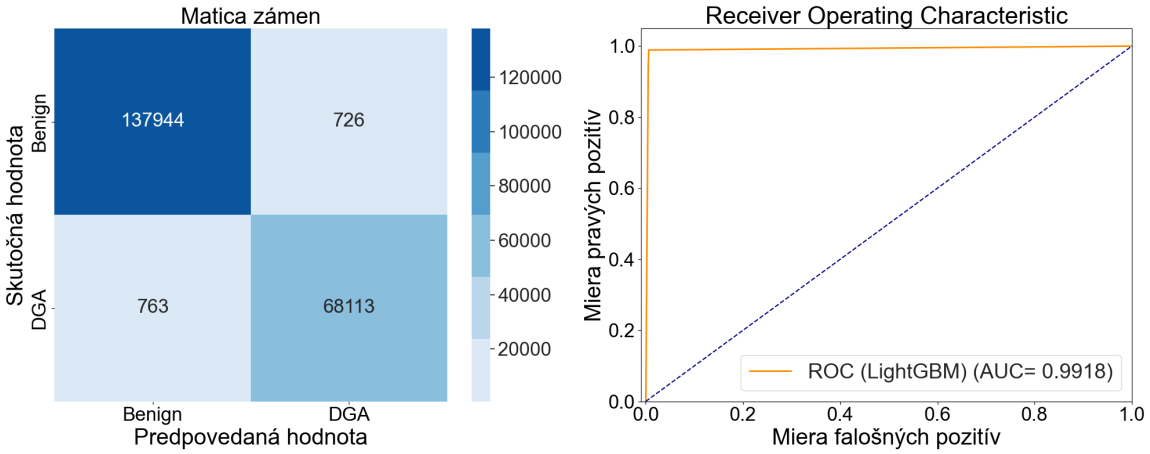
Pri výbere základných parametrov pre model LightGBM bolo dôležité zvážiť niekoľko kritérií. Základné parametre boli vybrané s ohľadom na všeobecné odporúčania pre nastavenie LightGBM, ako aj na špecifické charakteristiky dátového súboru, ako sú rozsah dát, distribúcia tried a počet príznakov. Základné parametre sa zvyčajne volia tak, aby poskytovali dobrý kompromis medzi rýchlosťou tréningu a prediktívnou schopnosťou, pričom sú nastavené na hodnoty, ktoré sú zvyčajne odporúčané v dokumentácii a literatúre. Vybrané základné parametre sú zobrazené v tabuľke 7.1.

Po nastavení základných parametrov modelu LightGBM bola využitá metóda mriežkového vyhľadávania (anglicky grid-search) na systematické prehľadávanie širokého spektra kombinácií parametrov, spolu s využitím krížovej validácie (anglicky cross-validation) na overenie výkonnosti modelu na nezávislej dátovej sade. Cieľom mriežkového vyhľadávania bola identifikácia najlepšej kombinácie parametrov, pričom základné a výsledné parametre je možné vidieť v tabuľke 7.2.

Parameter	Základná hodnota
objective	binary
boosting_type	gbdt
min_child_samples	25
colsample_bytree	1
reg_lambda	0.1
subsample	0.85
subsample_freq	1
subsample_for_bin	200000
min_split_gain	0.01
n_estimators	850
max_depth	10
num_leaves	25
learning_rate	0.01
scale_pos_weight	1.5

Tabuľka 7.1: Základné parametre modelu LightGBM

Po získaní výsledných parametrov modelu LightGBM pomocou metódy mriežkového vyhľadávania boli tieto parametre využité na trénovanie konečného modelu. Tento model bol ďalej vyhodnotený na testovacej sade a dosiahol vynikajúce výsledky, ktoré sú zobrazené na obrázku 7.5.



Obr. 7.5: Matica zámen a ROC-AUC krivka výsledného modelu LightGBM

7.4.2 Neurónová sieť

V tejto sekcii bude predstavené ladenie parametrov klasifikátora hlbokého učenia nazývaného Neurónová sieť, pričom bude uskutočňované prostredníctvom nastavenia nasledovných kľúčových parametrov:

- **Architektúra siete** – Počet vrstiev a neurónov určuje schopnosť modelu naučiť sa komplexné vzory. Správna kombinácia je nevyhnutná pre dosiahnutie dobrej generalizácie.

Parameter	Základná hodnota	Výsledné parametre
objective	binary	binary
boosting_type	gbdt	gbdt
min_child_samples	25	25
colsample_bytree	1	1
reg_lambda	0.1	0.45
subsample	0.85	0.85
subsample_freq	1	1
subsample_for_bin	200000	200000
min_split_gain	0.01	0.01
n_estimators	850	1050
max_depth	10	12
num_leaves	25	30
learning_rate	0.01	0.075
scale_pos_weight	1.5	1.5

Tabuľka 7.2: Porovnanie základných a výsledných parametrov modelu LightGBM

- **Aktivačná funkcia** – Voľba aktivačnej funkcie ovplyvňuje, ako sieť transformuje vstupy na výstupy, čo má priamy vplyv na učenie a výkonnosť siete.
- **Miera učenia** – Kľúčový parameter pre nastavenie tempa, akým sieť aktualizuje svoje váhy. Jeho optimalizácia je rozhodujúca pre efektívne učenie.
- **Veľkosť dávky** – Ovplyvňuje rýchlosť trénovania a stabilitu konvergencie, kde menšie dávky môžu viesť k lepšej generalizácii.
- **Optimalizátor** – Algoritmus, ktorý sa používa na optimalizáciu váh siete počas trénovania.
- **Trvanie trénovania** – Udáva počet epoch, teda úplných prechodov trénovacími dátami, počas ktorých sa sieť trénuje.

Pri výbere základných parametrov pre model neurónovej siete bolo dôležité zvážiť niekoľko kritérií. Základné parametre boli vybrané s ohľadom na všeobecné odporúčania pre nastavenie neurónových sietí, ako aj na špecifické charakteristiky dátového súboru, ako sú rozsah dát, distribúcia tried a počet príznakov. Z toho dôvodu bol zvolený počet vrstiev a neurónov tak, aby sieť mala dostatočnú kapacitu na naučenie sa komplexných vzorov, ale zároveň sa predišlo pretrénovaniu. Aktivačné funkcie boli vybraté s ohľadom na typ úlohy a charakteristiky dát. Miera učenia a veľkosť dávky boli nastavené tak, aby bolo dosiahuté stabilné učenie siete a zabránilo sa divergencii alebo konvergencii k neoptimálnym bodom. Vybrané základné parametre sú zobrazené v tabuľke 7.3.

Experimenty s architektúrou siete zahŕňali testovanie rôznych počtov vrstiev a neurónov. Rôzne testovacie konfigurácie zahŕňali siete s menším počtom vrstiev a vyšším počtom neurónov a taktiež naopak, siete s väčším počtom vrstiev, ale menším počtom neurónov v každej vrstve. Tieto testovacie konfigurácie boli zamerané na zistenie optimálneho rozloženia zdrojov, ktoré by umožnilo modelu efektívne sa učiť bez rizika pretrénovania, ktoré môže nastať v prípade príliš komplexnej architekúry neurónovej siete.

Následne boli vykonané experimenty s mierou učenia, kde sa skúmali rôzne hodnoty od veľmi nízkych po vyššie, aby sa zistil ich vplyv na stabilitu a rýchlosť učenia. Veľkosť dávky

Parameter	Hodnota
Architektúra	4 skryté vrstvy, každá po 64 neurónov
Aktivačná funkcia	ReLU (Rectified Linear Unit)
Miera učenia	0.001
Veľkosť dávky	512
Optimalizátor	Adam
Trvanie trénovania	300 epoch

Tabuľka 7.3: Základné parametre neurónovej siete

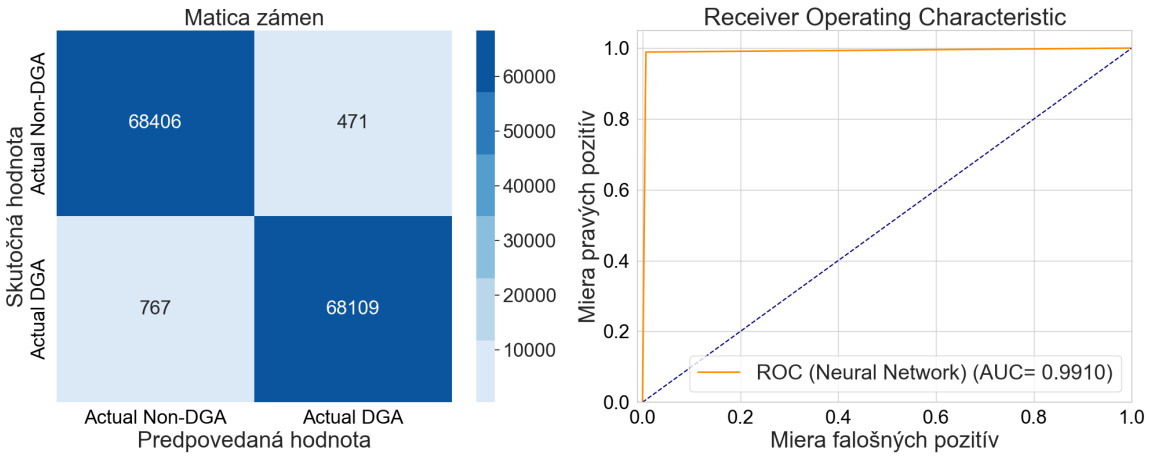
bola upravovaná od malých po veľké hodnoty, aby sa pozoroval vplyv na tréningový proces, najmä na rýchlosť a stabilitu konvergencie.

Taktiež boli testované rôzne aktivačné funkcie vrátane sigmoid a tanh, aby sa zistilo, ako tieto funkcie ovplyvňujú výkon modelu na danej dátovej sade.

Parameter	Hodnota
Architektúra	3 skryté vrstvy po 64 neurónov, 1 skrytá vrstva po 32 neurónov
Aktivačná funkcia	ReLU (Rectified Linear Unit)
Miera učenia	0.001
Veľkosť dávky	127
Optimalizátor	Adam
Trvanie trénovania	290 epoch

Tabuľka 7.4: Výsledné parametre neurónovej siete

Výsledné parametre boli určené na základe experimentov, kde boli testované rôzne kombinácie hodnôt parametrov. Zvolené parametre boli vyhodnotené ako optimálne pre dosiahnutie požadovanej výkonnosti modelu neurónovej siete a sú zobrazené v tabuľke 7.4. Následne boli výsledné parametre využité na trénovanie konečného modelu. Tento model bol ďalej vyhodnotený na testovacej sade a dosiahol vynikajúce výsledky, ktoré sú zobrazené na obrázku 7.6.



Obr. 7.6: Matica zámen a ROC-AUC krivka výsledného modelu neurónovej siete

7.4.3 Vyhodnotenie

Výsledným klasifikátorom využitým v rámci detekčného nástroja sa nakoniec stal klasifikátor LightGBM. Hlavným dôvodom pre tento výber boli lepšie výsledky v rámci oblasti falošných pozitív, kde klasifikátor LightGBM dosahoval skoro polovičné výsledky oproti klasifikátoru Neurónová sieť. Klasifikátor LightGBM avšak vykazoval lepšie výsledky i v metrike ROC-AUC a napr. pre detekčný nástroj kritickej metrike Prediction time, kde klasifikátor Neurónová sieť vykazoval výsledky až troj násobne vyššie ako klasifikátor LightGBM.

Kapitola 8

Testovanie

Táto kapitola sa zaoberá detailným popisom procesu testovania výsledného detekčného nástroja, pričom kľúčovým aspektom je zabezpečiť, že nástroj je spoľahlivý, efektívny a pripravený na praktické použitie. V rámci kapitoly sú podrobne rozobrané rôzne prístupy a metódy testovania, ktoré boli aplikované s cieľom overiť funkčnosť a výkonnosť vyvinutého softvéru. Najskôr sú predstavené jednotkové testy, ktoré sú nevyhnutné pre overenie správnosti jednotlivých komponentov systému. Následne je pozornosť venovaná testovaniu celého detekčného systému, kde sú integrované všetky komponenty a testovanie je vykonané v koherentnom a realistickom prostredí.

8.1 Jednotkové testy

Jednotkové testy boli vykonané pre každý funkčný blok detekčného nástroja. Tieto testy sú nevyhnutné pre overenie, že každý samostatný komponent systému funguje podľa očakávaní a špecifikácií, bez ohľadu na ostatné časti systému.

V rámci jednotkových testov sa zameriava na to, aby boli testy realizované automaticky a repetitívne, čo je kľúčové pre integrovanie testov do procesov CI/CD (Continuous Integration/Continuous Delivery). Automatizácia jednotkových testov umožňuje, aby boli testy spúšťané automaticky pri každej zmene kódu v rámci Git repozitára. Tento prístup zabezpečuje, že všetky zmeny sú dôkladne testované ešte predtým, než budú integrované do hlavnej vetvy vývoja, čo výrazne znižuje riziko zavedenia chýb do produkčného prostredia.

Tento proces nielenže zvyšuje dôveru vo funkčnosť vyvinutého softvéru, ale taktiež prispieva k rýchlejšej a efektívnejšej iterácii vývoja, pretože umožňuje vývojárom rýchlo identifikovať a opraviť chyby ešte predtým, ako sa dostanú do neskorších fáz vývoja alebo do produkčného prostredia.

8.2 Integračné testovanie detekčného systému

Integračné testovanie je kľúčovou súčasťou procesu overovania detekčného nástroja, ktoré sleduje, ako efektívne a spoľahlivo pracujú jednotlivé komponenty systému spolu v reálnom prostredí. Cieľom tohto testovania je zistiť, či sú komponenty navzájom kompatibilné a či je celý systém ako jeden funkčný celok schopný plniť svoje predpokladané úlohy.

V rámci integračného testovania boli vykonané testy bez využitia zoznamov Blacklist a Whitelist, čím bola overená integrácia detekčného systému a navrhnutých klasifikátorov z kapitoly 7 bez externých vplyvov. Následne boli do testovacieho procesu pridané zoznamy

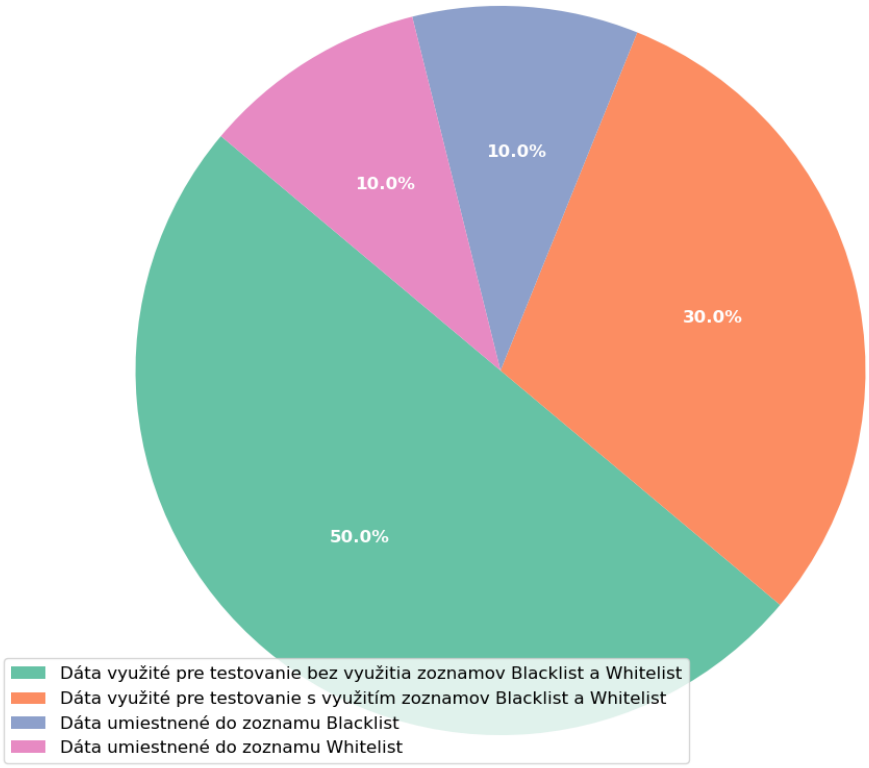
Blacklist a Whitelist, ktoré umožnili sledovať ako sa systém správa v prítomnosti explicitne definovaných bezpečných alebo rizikových entít. Testovanie umožnilo overiť schopnosť systému identifikovať a reagovať na skutočné hrozby v kontrolovanom, ale realistickom prostredí.

Príprava dát pre účely testovania

V rámci prípravy dát pre účely integračného testovania bolo nutné pripraviť dáta do vhodného formátu, tzn. aby zodpovedali požadovaným špecifikáciám pre spracovanie.

Pripravené dáta museli zodpovedať nasledujúcim špecifikám:

- **Exkluzivita dát** – Žiadne dáta použité v testovacej sade nesmú byť súčasťou trénovacej sady modelov. Toto zabezpečí, že výsledky testovania budú objektívne a neovplyvnené predchádzajúcim kontaktom modelu s týmito dátami.
- **Proporčné rozdelenie dát** – Testovacie dáta musia byť rozdelené na dve rovnaké časti, kde jedna polovica bude použitá v testoch bez využitia zoznamov Blacklist a Whitelist. Druhá polovica dát bude využitá pri testovaní so zoznamami Blacklist a Whitelist, čo umožní zistiť, ako dobre systém zvláda identifikáciu a spracovanie entít uvedených v týchto zoznamoch a ich integráciu s vybranými klasifikátormi. Proporčné rozdelenie dát je zobrazené na obrázku 8.1.



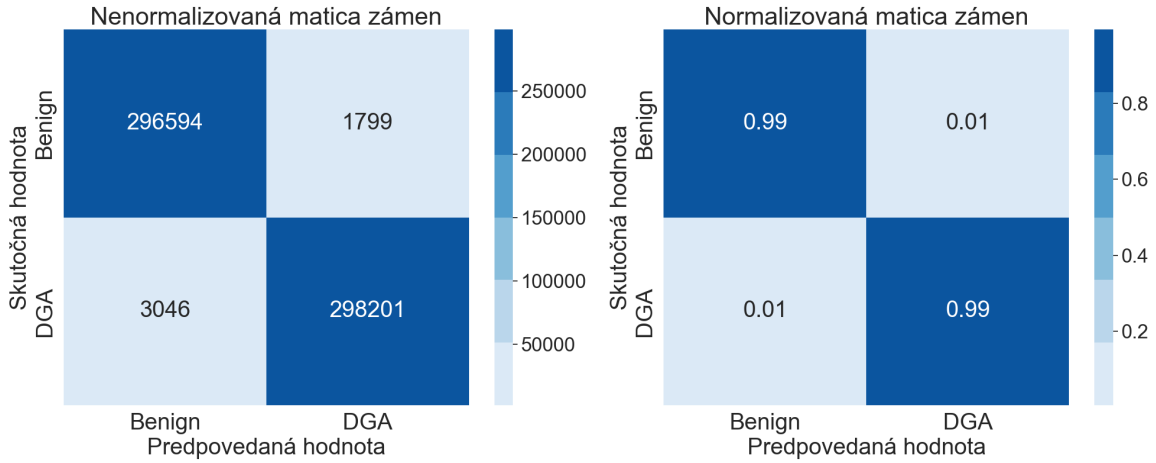
Obr. 8.1: Ukážka proporčného rozdelenia dát určených pre testovanie

Táto metóda prípravy a rozdelenia dát zabezpečila, že každá fáza testovania mala k dispozícii adekvátne a relevantné informácie, ktoré zabezpečili presné hodnotenie výkonu detekčného systému v rámci rôznych podmienok nasadenia.

Testovanie bez využitia Blacklistu a Whitelistu

V tejto fáze testovania bol detekčný systém testovaný bez použitia zoznamov Blacklist a Whitelist. Hlavným zámerom bolo overiť, ako efektívne dokážu samotné klasifikátory identifikovať a spracovať podozrivé doménové mená bez akýchkoľvek externých vplyvov. Tento test bol zameraný na zistenie základnej účinnosti a spoľahlivosti detekčného nástroja.

Ako už bolo špecifikované v sekcii 8.2, testované dáta pre túto fázu boli vybrané tak, aby neobsahovali žiadne doménové mená z trénovacej sady ani z preddefinovaných zoznamov. Tým bol zabezpečený objektívny a neovplyvnený pohľad na schopnosti klasifikátorov spracovať neznáme dáta.



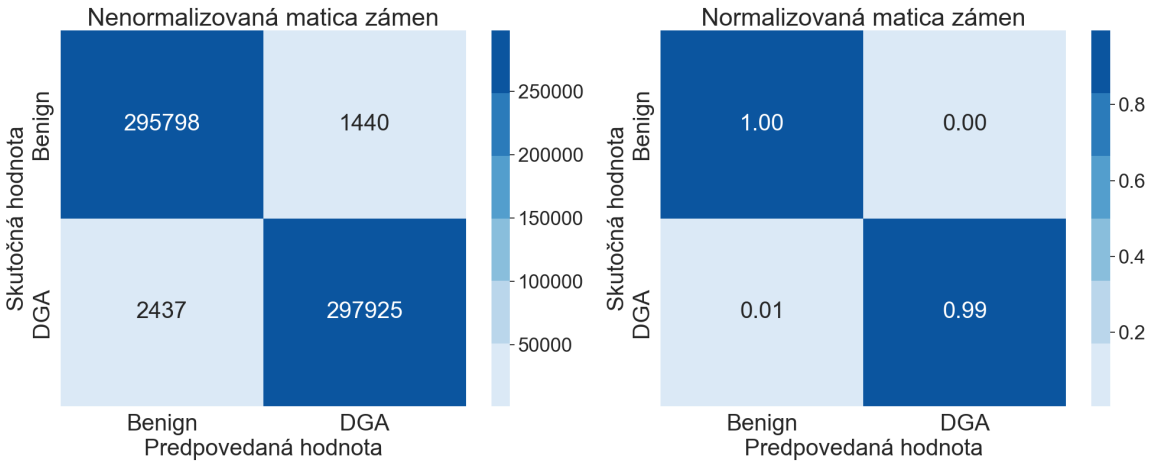
Obr. 8.2: Výsledky testovania bez využitia zoznamov Blacklist a Whitelist

Výsledky zobrazené na obrázku 8.2 vo forme matice zámen preukazujú, že klasifikátory sú schopné efektívne identifikovať a spracovávať neznáme doménové mená, čo zaisťuje vysokú úroveň spoľahlivosti pri detekcii.

Testovanie s využitím Blacklistu a Whitelistu

Následne bol systém testovaný s použitím zoznamov Blacklist a Whitelist. Cieľom týchto testov bolo zistiť, ako prítomnosť explicitne definovaných bezpečných alebo rizikových entít ovplyvní celkovú funkčnosť systému. Tieto testy zahŕňali analýzu správania sa systému pri identifikácii a spracovaní doménových mien zo zoznamov Blacklist a Whitelist, pričom bol zvlášť sledovaný vplyv týchto zoznamov na výkon klasifikátorov.

Z výsledkov viditeľných na obrázku 8.3 vo forme matice zámen je zrejmé, že integrácia Blacklistu a Whitelistu zlepšila celkovú presnosť a úspešnosť identifikácie DGA a benígnych doménových mien, pričom sa len minimálne znížila rýchlosť spracovania dát. Výsledky poukazujú na to, že systém je schopný efektívne integrovať externé zoznamy do procesu detekcie, čo zvyšuje jeho praktickú uplatniteľnosť a spoľahlivosť. Výsledok potvrdzuje, že detekčný nástroj je pripravený na praktické použitie a môže poskytnúť spoľahlivú ochranu pred hrozbami zahrnutými v sieťovej prevádzke.



Obr. 8.3: Výsledky testovania s využitím zoznamov Blacklist a Whitelist

Kapitola 9

Záver

Cieľom práce bolo vytvoriť systém pre detekciu a obranu proti botnetom využívajúcim algoritmy generovania domén (DGA), ktoré môžu byť použité na rôzne typy kybernetických hrozieb, vrátane distribuovaných útokov, šírenia malvéru, krádeže dát a mnohých ďalších.

V práci bola predstavená problematika botnetov a spôsob, akým využívajú DGA v kontexte systému doménových mien (DNS). Na základe daných poznatkov bolo možné stanoviť jasné požiadavky a kritériá pre návrh a implementáciu detekčného nástroja s cieľom dosiahnutia efektívnosti a presnosti pri detekcii botnetov využívajúcich DGA s minimalizáciou výskytu falošných pozitívnych aj negatívnych výsledkov.

Na základe týchto požiadaviek bol postavený návrh komplexného detekčného systému, ktorý integruje pokročilé techniky strojového učenia pre analýzu a klasifikáciu sieťovej prevádzky systému DNS. Vývoj detekčného systému bol rozdelený do niekoľkých blokov, pričom každý z nich sa zameriava na špecifický aspekt detekcie a obrany proti botnetom založeným na DGA.

Pre potrebu čo najefektívnejšieho spracovania veľkých objemov dát sieťovej prevádzky systému DNS bol prvý blok detekčného nástroja implementovaný v nízko úrovňovom jazyku C++, s dôrazom na využitie viacvláknového programovania pre efektívne paralelné spracovanie dát. Kľúčovou technológiou pre optimalizáciu rýchlosti spracovania boli bez-zámkové fronty, ktoré umožňujú pridávať a odstraňovať prvky efektívne a bez vzájomného blokovania, čím účinne minimalizujú oneskorenie. S cieľom ďalšieho zvýšenia efektivity boli využité i zoznamy Blacklist a Whitelist, obsahujúce známe DGA a taktiež benígne doménové mená.

Druhý blok detekčného nástroja je zameraný na dôkladnú extrakciu a analýzu lexikálnych vlastností doménových mien, ktoré sú charakteristické pre DGA a ich následnú klasifikáciu s využitím techník strojového učenia. Proces extrakcie vlastností zahŕňa komplexnú analýzu 48 rôznych vlastností, napr. Shannonovej entropie reťazca, Jaccardového indexu, počtu zhôd s DGA N-gramami a iných lexikálnych vlastností. Pre zefektívnenie extrakcie vlastností spojených s N-gramami bol využitý algortimus Aho-Corasick, ktorý umožňuje rýchle vyhľadávanie a identifikáciu viacerých vzorov súčasne v danom textovom reťazci a je zvlášť efektívny v prostredí s veľkým objemom dát. Táto optimalizácia bola taktiež využitá pre zefektívnenie extrakcie vlastností v rámci riešenia pod výskumným projektom FETA.

S úsilím dosiahnuť najlepšie výsledky klasifikácie bolo nevyhnutné porovnať rôzne klasifikátory strojového učenia a nakoniec vybrať tie s najlepšími výsledkami. Boli vybrané dva klasifikátory, LightGBM a Neurónová sieť na základe ich výkonnosti a schopnosti efektívne rozlišovať medzi DGA a benígnymi doménovými menami. Porovnanie bolo vykonané na širokej škále modelov strojového učenia, kde každý z nich bol hodnotený podľa komplexných metrík s cieľom podrobného posúdenia, akým spôsobom modely reagujú na rozličné situácie.

Klasifikátor LightGBM, ktorý dosiahol ROC-AUC skóre v hodnote až 99,18% a vynikajúce skóre 98.88% v metrike Recall a celkovo lepšie výsledky oproti klasifikátoru Neurónová sieť, bol následne implementovaný a integrovaný do detekčného systému, čím sa zvýšila celková efektívnosť a spoľahlivosť riešenia.

Implementácia webovej aplikácie sa sústreďuje na efektívne zobrazovanie informácií o klasifikovaných doménach a pridružených údajoch. Vývoj sa zameriava na funkčnosť a užívateľskú prívetivosť, pričom prioritou je intuitívnosť a rýchlosť používateľského rozhrania. Architektúra aplikácie je navrhnutá pre rozšíriteľnosť a udržateľnosť, čo zabezpečuje jej spoľahlivú integráciu s existujúcimi komponentami a pripravenosť na praktické nasadenie.

Pre zabezpečenie spoľahlivosti, efektivity a pripravenosti na praktické použitie bolo vykonané testovanie detekčného nástroja, pričom boli využité rôzne metódy testovania. Iniciálne boli realizované jednotkové testy na overenie správnosti komponentov systému, ktoré sú súčasťou procesu CI (Continuous Integration). Nasledovalo integračné testovanie, ktoré preverilo koherentnosť a funkčnosť celého systému v realistických podmienkach. Výsledky testovania potvrdili vysokú úroveň funkčnosti a pripravenosť nástroja na efektívne rozpoznávanie a reagovanie na hrozby v praxi.

Kybernetické hrozby v podobe botnetov sa neustále vyvíjajú a ich detekcia je stále náročnejšia vďaka rastúcej zložitosti útokov a ich sofistikovanosti. Z toho dôvodu bude ďalší výskum smerovaný na nové technológie ako DNS over HTTPS (DoH), ktoré sa stávajú obľúbenými nástrojmi pre kybernetických útočníkov, pretože umožňujú skryť ich činnosť a obísť tradičné bezpečnostné mechanizmy. DoH umožňuje šifrovanie dotazov systému DNS priamo cez HTTPS, čím sa zmenšuje možnosť ich odhalenia a blokovania. Jedným z prístupov, ktorým môže výskum pokračovať, môže byť analýza šifrovanej prevádzky systému DNS a hľadanie charakteristických vzorov alebo správania, ktoré sú typické pre botnety. Ďalšou možnosťou môže byť kombinácia behaviorálnej analýzy s použitím strojového učenia na identifikáciu nezvyčajného alebo podozrivého správania sa zariadení v sieti. V rámci implementácie by bolo vhodné zvážiť implementáciu pokročilých analýznych techník pre identifikáciu anomálií v sieťovej prevádzke, ktoré by mohli naznačovať prítomnosť nových typov hrozieb, ako sú napr. ransomware, phishing apod.

Literatúra

[1] Adams, T. *ScoutDNS Most Abused Top Level Domains List – October 2020* [online]. 2020 [cit. 2024-04-18]. Dostupné z: https://www.scoutdns.com/most-abused-top-level-domains-list-october-scoutdns/.

[2] Ahluwalia, A., Traore, I., Ganame, K. et al. Detecting Broad Length Algorithmically Generated Domains. In: Traore, I., Woungang, I. a Awad, A., ed. *Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments*. Cham: Springer International Publishing, 2017, s. 19–34. ISBN 978-3-319-69155-8.

[3] Al Mashhadi, S., Anbar, M., Hasbullah, I. et al. Hybrid rule-based botnet detection approach using machine learning for analysing DNS traffic. *PEERJ COMPUTER SCIENCE*. 341-345 OLD ST, THIRD FLR, LONDON, EC1V 9LL, ENGLAND: PEERJ INC. AUG 13 2021, zv. 7. DOI: 10.7717/peerj-cs.640. ISSN 2376-5992.

[4] Alauthman, M., Aslam, N., Al Kasassbeh, M. et al. An efficient reinforcement learning-based Botnet detection approach. *Journal of Network and Computer Applications*. 2020, zv. 150, s. 102479. DOI: 10.1016/j.jnca.2019.102479. ISSN 1084-8045. Dostupné z: https://www.sciencedirect.com/science/article/pii/S108480451930339X.

[5] Alieyan, K., ALmomani, A., Manasrah, A. et al. A survey of botnet detection based on DNS. *Neural Computing and Applications*. Júl 2017, zv. 28, č. 7, s. 1541–1558. DOI: 10.1007/s00521-015-2128-0. ISSN 1433-3058. Dostupné z: https://doi.org/10.1007/s00521-015-2128-0.

[6] Anderson, R. *Introduction to Identity on ASP.NET Core* [online]. 25. augusta 2023 [cit. 2024-04-09]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-8.0&tabs=visual-studio.

[7] Antonuk, A. *Rabbitmq-c: RabbitMQ C client* [online]. 2024 [cit. 2024-04-03]. Dostupné z: https://github.com/alanxz/rabbitmq-c.

[8] Applied Informatics Software Engineering GmbH. *POCO C++ LIBRARIES INFORMATION* [online]. 2024 [cit. 2024-04-03]. Dostupné z: https://pocoproject.org/about.html#about.

[9] Ayo, F. E., Awotunde, J. B., Folorunso, S. O. et al. A genomic rule-based KNN model for fast flux botnet detection. *Egyptian Informatics Journal*. 2023, zv. 24, č. 2, s. 313–315. DOI: 10.1016/j.eij.2023.05.002. ISSN 1110-8665.

[10] Bajtoš, T. *Analýza botnetov pomocou honeypotov*. Košice, SK, 2017. 22 s. Diplomová práca. Univerzita Pavla Jozefa Šafárika v Košiciach, Prírodovedecká fakulta. Dostupné z: https://opac.crzp.sk/?fn=detailBiblioForm&sid=EE347D3FB9176FA23D68667BB7F1.

[11] Bogard, J. *AutoMapper* [online]. 2024 [cit. 2024-04-09]. Dostupné z: https://automapper.org.

[12] Breiman, L. Random Forests. *Machine Learning*. October 2001, zv. 45, č. 1, s. 5–32. DOI: 10.1023/A:1010933404324. ISSN 1573-0565. Dostupné z: https://doi.org/10.1023/A:1010933404324.

[13] Budogol. *CyberEz - Cyber Security* [online]. 2023 [cit. 2023-12-17]. Licencované pod licenciou CC BY 4.0. Dostupné z: https://www.figma.com/community/file/1302801971340508755.

[14] Bučko, F. *Klasifikácia doménových mien generovaných algoritmami DGA*. Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně. Fakulta informačních technologií. Dostupné z: http://hdl.handle.net/11012/212736.

[15] Casino, F., Lykousas, N., Homoliak, I. et al. *HYDRA dataset*. Zenodo, júl 2020. DOI: 10.5281/zenodo.3965397. Dostupné z: https://doi.org/10.5281/zenodo.3965397.

[16] Chen, T. a Guestrin, C. XGBoost: A Scalable Tree Boosting System. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, August 2016. KDD '16. DOI: 10.1145/2939672.2939785. Dostupné z: http://dx.doi.org/10.1145/2939672.2939785.

[17] Chugh, V. *Python pandas tutorial: The ultimate guide for beginners* [DataCamp Blog]. May 2023 [cit. 2024-04-18]. Dostupné z: https://www.datacamp.com/tutorial/pandas.

[18] Cortes, C. a Vapnik, V. Support-vector networks. *Machine Learning*. Sep 1995, zv. 20, č. 3, s. 273–297. DOI: 10.1007/BF00994018. ISSN 1573-0565. Dostupné z: https://doi.org/10.1007/BF00994018.

[19] DataScientest. *NumPy: The Most Used Python Library in Data Science* [online]. 13. marca 2023 [cit. 2024-04-08]. Dostupné z: https://www.datascientest.com/numpy-the-most-used-python-library-in-data-science.

[20] Desrochers, C. *Detailed Design of a Lock-Free Queue* [online]. 6. november 2014 [cit. 2024-04-06]. Dostupné z: https://moodycamel.com/blog/2014/detailed-design-of-a-lock-free-queue.

[21] Dhanjani, N. a Clarke, J. *Network Security Tools*. O'Reilly, 2005. 244–246 s. O'Reilly Software Series. ISBN 9780596007942. Dostupné z: https://books.google.sk/books?id=iV8DRekYvg0C.

[22] Erickson, B. J. a Kitamura, F. Magician's Corner: 9. Performance Metrics for Machine Learning Models. *Radiology: Artificial Intelligence*. 2021, zv. 3, č. 3, s. 1–7. DOI: 10.1148/ryai.2021200126. Dostupné z: https://doi.org/10.1148/ryai.2021200126.

[23] Glen, S. *Jaccard Index / Similarity Coefficient* [online]. 2024 [cit. 2024-04-18]. Dostupné z: https://www.statisticshowto.com/jaccard-index/.

[24] Gu, G., Yegneswaran, V., Porras, P. et al. Active Botnet Probing to Identify Obscure Command and Control Channels. In: *2009 Annual Computer Security Applications Conference*. Los Alamitos, CA, USA: IEEE Computer Society, Dec 2009, s. 241. DOI: 10.1109/ACSAC.2009.30. ISBN 978-0-7695-3919-5.

[25] Gurney, K. *An Introduction to Neural Networks*. 1. vyd. London: CRC Press, 1997. 10–13 s. ISBN 9781315273570. Dostupné z: https://doi.org/10.1201/9781315273570.

[26] Hachem, N., Ben Mustapha, Y., Granadillo, G. G. et al. Botnets: Lifecycle and Taxonomy. In: *2011 Conference on Network and Information Systems Security*. 2011, s. 1–8. DOI: 10.1109/SAR-SSI.2011.5931395. ISBN 978-1-4577-0735-3.

[27] Han, C. a Zhang, Y. CODDULM: An approach for detecting C&C domains of DGA on passive DNS traffic. In: *2017 6th International Conference on Computer Science and Network Technology (ICCSNT)*. 2017, s. 385–388. DOI: 10.1109/ICCSNT.2017.8343724. ISBN 978-1-5386-0494-6.

[28] Han, J., Kamber, M. a Pei, J. 8 - Classification: Basic Concepts. In: Han, J., Kamber, M. a Pei, J., ed. *Data Mining*. 3. vyd. Boston: Morgan Kaufmann, 2012, s. 364–377. The Morgan Kaufmann Series in Data Management Systems. DOI: https://doi.org/10.1016/B978-0-12-381479-1.00008-3. ISBN 978-0-12-381479-1. Dostupné z: https://www.sciencedirect.com/science/article/pii/B9780123814791000083.

[29] Hariaji, A. S. a Girsang, A. S. Algorithmic Detection of Malicious Domains using N-Gram Embedding and Attention-Based Bidirectional Gated Recurrent Unit. *Journal of Theoretical and Applied Information Technology*. September 2023, zv. 101, č. 18. ISSN 1992-8645.

[30] Hemanth, J., Xing, Y., Shu, H. et al. Survey on Botnet Detection Techniques: Classification, Methods, and Evaluation. *Mathematical Problems in Engineering*. Hindawi. 2021, zv. 2021, s. 3. DOI: 10.1155/2021/6640499. ISSN 1024-123X. Dostupné z: https://doi.org/10.1155/2021/6640499.

[31] Huang, S.-Y., Mao, C.-H. a Lee, H.-M. Fast-Flux Service Network Detection Based on Spatial Snapshot Mechanism for Delay-Free Detection. In: *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*. New York, NY, USA: Association for Computing Machinery, 2010, s. 101–111. ASIACCS '10. DOI: 10.1145/1755688.1755702. ISBN 9781605589367. Dostupné z: https://doi.org/10.1145/1755688.1755702.

[32] Jiang, N., Cao, J., Jin, Y. et al. Identifying suspicious activities through DNS failure graph analysis. In: IEEE. *The 18th IEEE International Conference on Network Protocols*. November 2010, s. 144–153. DOI: 10.1109/ICNP.2010.5762763. ISBN 978-1-4244-8644-1.

[33] Karim, A., Salleh, R. B., Shiraz, M. et al. Botnet detection techniques: review, future trends, and issues. *Journal of Zhejiang University SCIENCE C*. Nov 2014, zv. 15, č. 11, s. 948. DOI: 10.1631/jzus.C1300242. ISSN 1869-196X.

[34] Ke, G., Meng, Q., Finley, T. et al. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In: Guyon, I., Luxburg, U. V., Bengio, S. et al., ed. *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2017, sv. 30. Dostupné z: https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

[35] Khehra, G. a Sofat, S. Botnet Detection Techniques: A Review. In: *2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)*. 2018, s. 1319–1326. DOI: 10.1109/ICCONS.2018.8663082. ISBN 978-1-5386-2843-0.

[36] Li, Z., Goyal, A. a Chen, Y. Honeynet-based botnet scan traffic analysis. In: Lee, W., Wang, C. a Dagon, D., ed. *Botnet Detection: Countering the Largest Security Threat*. Springer New York, NY, 2008, s. 25–27. DOI: 10.1007/978-0-387-68768-1. ISBN 978-0-387-68766-7.

[37] Mahdavifar, S., Maleki, N., Lashkari, A. H. et al. Classifying Malicious Domains using DNS Traffic Analysis. In: *2021 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*. Los Alamitos, CA, USA: IEEE Computer Society, Október 2021, s. 60–67. DOI: 10.1109/DASC-PICom-CBDCom-CyberSciTech52372.2021.00024. ISBN 978-1-6654-2175-1.

[38] Manasrah, A. M., Khdour, T. a Freehat, R. DGA-based botnets detection using DNS traffic mining. *Journal of King Saud University - Computer and Information Sciences*. 2022, zv. 34, č. 5, s. 2045–2061. DOI: https://doi.org/10.1016/j.jksuci.2022.03.001. ISSN 1319-1578. Dostupné z: https://www.sciencedirect.com/science/article/pii/S1319157822000726.

[39] Marko, P. a Vilhan, P. Efficient Detection of Malicious Nodes Based on DNS and Statistical Methods. In: IEEE. *2012 IEEE 10th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*. Herl'any, Slovakia: [b.n.], 2012, s. 227–230. DOI: 10.1109/SAMI.2012.6208963. ISBN 978-1-4577-0197-9.

[40] Matoušek, P. *Síťové služby a jejich architektura*. Publishing house of Brno University of Technology VUTIUM, 2014. 107–161 s. ISBN 978-80-214-3766-1.

[41] Maze, C. *Introduction to Scrutor Library in .NET* [online]. 19. júna 2023 [cit. 2024-04-09]. Dostupné z: https://code-maze.com/dotnet-dependency-injection-with-scrutor/.

[42] Microsoft. *Entity Framework Core - Overview* [online]. 2024 [cit. 2024-03-26]. Dostupné z: https://docs.microsoft.com/en-us/ef/core/.

[43] Mockapetris, P. *Domain names: Concepts and facilities* [RFC 882]. Request for Comments (RFC) 882. Internet Engineering Task Force (IETF), november 1983. Obsoleted by RFCs 1034, 1035, updated by RFC 973. Dostupné z: http://www.ietf.org/rfc/rfc882.txt.

[44] Mockapetris, P. *Domain names: Implementation specification* [RFC 883]. Request for Comments (RFC) 883. Internet Engineering Task Force (IETF), november 1983.

Obsoleted by RFCs 1034, 1035, updated by RFC 973. Dostupné z: http://www.ietf.org/rfc/rfc883.txt.

[45] MOCKAPETRIS, P. *Domain names - implementation and specification* [RFC 1035 (INTERNET STANDARD)]. Request for Comments (RFC) 1035. Internet Engineering Task Force (IETF), november 1987. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2673, 2845, 3425, 3658, 4033, 4034, 4035, 4343, 5936, 5966, 6604. Dostupné z: http://www.ietf.org/rfc/rfc1035.txt.

[46] MONGODB, INC.. *MongoDB Documentation: Databases and Collections* [online]. 2023 [cit. 2023-11-29]. Dostupné z: https://www.mongodb.com/docs/manual/core/databases-and-collections/.

[47] MONGODB INC.. *MongoDB Documentation: Introduction to MongoDB* [online]. 2023 [cit. 2023-11-29]. Dostupné z: https://www.mongodb.com/docs/manual/core/databases-and-collections/.

[48] MORRIS, R. *Swashbuckle.AspNetCore* [online]. 2024 [cit. 2024-04-09]. Dostupné z: https://github.com/domaindrivendev/Swashbuckle.AspNetCore.

[49] MUŁA, W., OMBREDANNE, P. et al. *Pyahocorasick Documentation* [online]. 2024 [cit. 2024-04-18]. Dostupné z: https://pyahocorasick.readthedocs.io/.

[50] MWATSON. *Serilog Tutorial for .NET Logging: 16 Best Practices and Tips* [online]. 27. februára 2024 [cit. 2024-04-09]. Dostupné z: https://stackify.com/serilog-tutorial-net-logging.

[51] NLOHMANN. *Nlohmann/json: JSON for Modern C++* [online]. 2024 [cit. 2024-04-03]. Dostupné z: https://github.com/nlohmann/json.

[52] ORACLE CORPORATION. *What is MySQL?* [online]. 2024 [cit. 2024-03-26]. Dostupné z: https://www.oracle.com/mysql/what-is-mysql/.

[53] PCAPPLUSPLUS CONTRIBUTORS. *PcapPlusPlus: Feature Overview* [online]. 2024 [cit. 2024-03-29]. Dostupné z: https://pcapplusplus.github.io/v1912/docs/features.

[54] PERDISCI, R., CORONA, I., DAGON, D. et al. Detecting Malicious Flux Service Networks through Passive Analysis of Recursive DNS Traces. In: *2009 Annual Computer Security Applications Conference*. 2009, s. 311–320. DOI: 10.1109/ACSAC.2009.36. ISBN 978-1-4244-5327-6.

[55] PIKA. *Pika RabbitMQ Client Library for Python* [online]. 2024 [cit. 2024-04-08]. Dostupné z: https://github.com/pika/pika.

[56] PLOHMANN, D., YAKDAN, K., KLATT, M. et al. A Comprehensive Measurement Study of Domain Generating Malware. In: *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, August 2016, s. 263–278. ISBN 978-1-931971-32-4. Dostupné z: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/plohmann.

[57] Pomorova, O., Savenko, O., Lysenko, S. et al. Anti-evasion Technique for the Botnets Detection Based on the Passive DNS Monitoring and Active DNS Probing. In: Gaj, P., Kwiecień, A. a Stera, P., ed. *Computer Networks*. Cham: Springer International Publishing, 2016, s. 83–95. DOI: 10.1007/978-3-319-39207-3_8. ISBN 978-3-319-39207-3.

[58] Prieto, I., Magaña, E., Morató, D. et al. Botnet Detection based on DNS Records and Active Probing. In: INSTICC. *Proceedings of the International Conference on Security and Cryptography - Volume 1: SECRYPT, (ICETE 2011)*. SciTePress, 2011, s. 307–316. DOI: 10.5220/0003522903070316. ISBN 978-989-8425-71-3.

[59] Rigtorp, E. *MPMCQueue: A bounded multi-producer multi-consumer concurrent queue written in C++11* [online]. 2024 [cit. 2024-04-03]. Dostupné z: https://github.com/rigtorp/MPMCQueue.

[60] Ruts, D. *Improved DGA-based botnet detection through context-related feature selection based on packet flow information*. London, UK, 2023. 1–7 s. Diplomová práca. Open University, faculty of Science. Dostupné z: https://research.ou.nl/ws/portalfiles/portal/65157531/Ruts_D_IM990C_Scriptie_Pure.pdf.

[61] Schüppen, S., Teubert, D., Herrmann, P. et al. FANCI : Feature-based Automated NXDomain Classification and Intelligence. In: *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, August 2018, s. 1165–1181. ISBN 978-1-939133-04-5. Dostupné z: https://www.usenix.org/conference/usenixsecurity18/presentation/schuppe.

[62] Seewald, A. K. a Gansterer, W. N. On the detection and identification of botnets. *Computers & Security*. 2010, zv. 29, č. 1, s. 45–58. DOI: https://doi.org/10.1016/j.cose.2009.07.007. ISSN 0167-4048. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0167404809000820.

[63] Setinský, J. *Detekce škodlivých doménových jmen*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně. Fakulta informačních technologií. Dostupné z: https://www.vut.cz/www_base/zav_prace_soubor_verejne.php?file_id=230467.

[64] Shafee, A. Botnets and their detection techniques. In: *2020 International Symposium on Networks, Computers and Communications (ISNCC)*. 2020, s. 1–6. DOI: 10.1109/ISNCC49221.2020.9297307. ISBN 978-1-7281-5629-3.

[65] Shah, A. *How to Save a Classifier to Disk in Scikit-learn* [online]. 16. August 2023 [cit. 2024-04-18]. Dostupné z: https://wandb.ai/a-sh0ts/publications/reports/How-to-Save-a-Classifier-to-Disk-in-Scikit-learn--Vmlldzo0NDc1ODI0.

[66] Shahzad, H., Sattar, A. R. a Skandaraniyam, J. DGA Domain Detection using Deep Learning. In: *2021 IEEE 5th International Conference on Cryptography, Security and Privacy (CSP)*. 2021, s. 139–143. DOI: 10.1109/CSP51677.2021.9357591. ISBN 978-1-7281-8622-1.

[67] Silva, S. S., Silva, R. M., Pinto, R. C. et al. Botnets: A survey. *Computer Networks*. 2013, zv. 57, č. 2, s. 378–403. DOI: 10.1016/j.comnet.2012.07.021. ISSN

1389-1286. Dostupné z: https://www.sciencedirect.com/science/article/pii/S1389128612003568.

[68] Singh, M., Singh, M. a Kaur, S. Issues and challenges in DNS based botnet detection: A survey. *Computers & Security*. Elsevier. 2019, zv. 86, č. 13, s. 34. DOI: 10.1016/j.cose.2019.05.019. ISSN 1872-6208.

[69] Sood, A. K. a Zeadally, S. A Taxonomy of Domain-Generation Algorithms. *IEEE Security & Privacy*. 2016, zv. 14, č. 4, s. 46–53. DOI: 10.1109/MSP.2016.76. ISSN 1540-7993.

[70] SQLite. *About SQLite* [online]. 2024 [cit. 2024-03-26]. Dostupné z: https://www.sqlite.org/about.html.

[71] Stinson, E. a Mitchell, J. C. Characterizing Bots' Remote Control Behavior. In: M. Hämmerli, B. a Sommer, R., ed. *Detection of Intrusions and Malware, and Vulnerability Assessment*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, s. 89–91. DOI: 10.1007/978-3-540-73614-1_6. ISBN 978-3-540-73614-1.

[72] Szynkiewicz, P. Signature-Based Detection of Botnet DDoS Attacks. In: Kołodziej, J., Repetto, M. a Duzha, A., ed. *Cybersecurity of Digital Service Chains: Challenges, Methodologies, and Tools*. Cham: Springer International Publishing, 2022, s. 120–135. DOI: 10.1007/978-3-031-04036-8_6. ISBN 978-3-031-04036-8.

[73] Tesfahun, A. a Bhaskari, D. L. Botnet detection and countermeasures-a survey. *International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)*. Citeseer. Júl 2013, zv. 2, č. 4, s. 311. ISSN 2278-6856.

[74] Tišňovský, P. *RabbitMQ: jedna z nejúspěšnějších implementací brokera* [online]. 20. decembra 2018 [cit. 2024-03-30]. Dostupné z: https://www.root.cz/clanky/rabbitmq-jedna-z-nejuspesnejsich-implementaci-brokera/#k02.

[75] Tyagi, R., Paul, T., Manoj, B. S. et al. A novel HTTP botnet traffic detection method. In: *2015 Annual IEEE India Conference (INDICON)*. 2015, s. 1–6. DOI: 10.1109/INDICON.2015.7443675. ISBN 978-1-4673-7399-9.

[76] Villamarin, R. a Brustoloni, J. Identifying Botnets Using Anomaly Detection Techniques Applied to DNS Traffic. In: IEEE. *2008 5th IEEE Consumer Communications and Networking Conference*. Február 2008, s. 476. DOI: 10.1109/ccnc08.2007.112. ISBN 978-1-4244-1456-7.

[77] Vormayr, G., Zseby, T. a Fabini, J. Botnet Communication Patterns. *IEEE Communications Surveys & Tutorials*. IEEE. 2017, zv. 19, č. 4, s. 2768–2772. DOI: 10.1109/COMST.2017.2749442. ISSN 1553-877X.

[78] Wu, P. a Zhao, H. Some Analysis and Research of the AdaBoost Algorithm. In: Chen, R., ed. *Intelligent Computing and Information Science*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, s. 1–5. ISBN 978-3-642-18129-0.

[79] Xiaobo, M., Junjie, Z., Zhenhua, L. et al. Accurate DNS query characteristics estimation via active probing. *Journal of Network and Computer Applications*. 2015,

zv. 47, s. 72–84. DOI: 10.1016/j.jnca.2014.09.016. ISSN 1084-8045. Dostupné z: https://www.sciencedirect.com/science/article/pii/S1084804514002239.

[80] Xu, S., Li, S., Meng, K. et al. An Adaptive Malicious Domain Detection Mechanism with DNS Traffic. In: *Proceedings of the 2017 VI International Conference on Network, Communication and Computing*. New York, NY, USA: Association for Computing Machinery, 2017, s. 86–91. ICNCC '17. DOI: 10.1145/3171592.3171595. ISBN 9781450353663.

[81] Yadav, S. a Reddy, A. L. N. Winning with DNS Failures: Strategies for Faster Botnet Detection. In: Rajarajan, M., Piper, F., Wang, H. a Kesidis, G., ed. *Security and Privacy in Communication Networks*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, s. 446–459. ISBN 978-3-642-31909-9.

[82] Zago, M., Gil Pérez, M. a Martínez Pérez, G. Scalable detection of botnets based on DGA. *Soft Computing*. April 1 2020, zv. 24, č. 8, s. 5517–5537. DOI: 10.1007/s00500-018-03703-8. ISSN 1433-7479.

[83] Zhang, H., Gharaibeh, M., Thanasoulas, S. et al. BotDigger: Detecting DGA Bots in a Single Network. In: *Proceedings of the IEEE International Conference on Traffic Monitoring and Analysis*. Louvain La Neuve, Belgium: IEEE, Apríl 2016, s. 16–21. DOI: http://dx.doi.org/10.1109/ICIMP.2010.11. Dostupné z: http://www.cs.colostate.edu/~hanzhang/papers/BotDigger-TMA16.pdf.

[84] Zhang, L., Yu, S., Wu, D. et al. A Survey on Latest Botnet Attack and Defense. In: *2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*. 2011, s. 53–60. DOI: 10.1109/TrustCom.2011.11. ISBN 978-1-4577-2135-9.

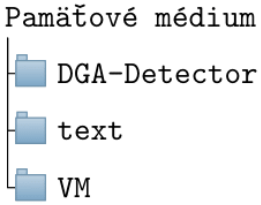
[85] Zhou, S., Lin, L., Yuan, J. et al. CNN-based DGA Detection with High Coverage. In: *2019 IEEE International Conference on Intelligence and Security Informatics (ISI)*. 2019, s. 62–67. DOI: 10.1109/ISI.2019.8823200. ISBN 978-1-7281-2505-3.

[86] Zou, X., Hu, Y., Tian, Z. et al. Logistic Regression Model Optimization and Case Analysis. In: *2019 IEEE 7th International Conference on Computer Science and Network Technology (ICCSNT)*. 2019, s. 135–139. DOI: 10.1109/ICCSNT47585.2019.8962457. ISBN 978-1-7281-3300-3.

Príloha A

Obsah přiloženého paměťového média

Na pamäťovom médiu sa nachádzajú adresáre zobrazené v nasledujúcom obrázku:



Obr. A.1: Štruktúra pamäťového média

kde:

- **DGA-Detector** – Predstavuje zložku so zdrojovými súbormi detekčného nástroja
- **Text** – Obsahuje text práce vo formáte `.pdf` a zdrojové súbory nutné pre jeho vytvorenie
- **VM** – Predstavuje zložku s virtuálnym strojom, na ktorom je detekčný nástroj nasadený

Príloha B

Manuál

V tejto prílohe je popísané, akým spôsobom je možné nainštalovať jednotlivé bloky detekčného nástroja a ich závislosti. Následne je taktiež predvedené spustenie jednotlivých blokov.

B.1 Prvý blok detekčného nástroja

V tejto sekcii bude popísaná inštalácia a spustenie prvého bloku detekčného nástroja nazývaného Detector.

Požiadavky

Pre inštaláciu je potrebné mať nainštalované nasledujúce prerekvizity:

- Kompilátor jazyka C++ a nástroj na jeho zostavenie nazývaný CMake
- Správca balíkov vcpkg
- RabbitMQ
- MongoDB

Inštalácia

Pre inštaláciu a následné spustenie prvého bloku detekčného nástroja je nutné postupovať podľa nasledujúcich krokov:

1. Navigácia do adresára prvého bloku detekčného nástroja:

   ```
   $ cd DGA-Detector/src/Detector
   ```

2. Inštalácia potrebných knižníc jazyka C++ pomocou nástroja vcpkg a kompilácia použitím nasledovného príkazu:

   ```
   $ cmake -B build -S . -DCMAKE_TOOLCHAIN_FILE=<cesta do vcpkg.cmake>
   ```

3. Zadanie argumentov príkazového riadka prvého bloku detekčného nástroja na konfiguráciu MongoDB, RabbitMQ a ďalších potrebných informácií, ktoré je možné získať použitím parametra `-h` alebo jeho dlhou verziou `-help`.

4. Inštancia MongoDB musí obsahovať nasledujúce kolekcie: Blacklist, Whitelist a Result pre správne fungovanie.

5. Spustenie prvý blok detekčného nástroja:

   ```
   $ sudo ./Detector <arguments>
   ```

Prvý blok detekčného nástroja je možné spustiť aj bez argumentov príkazového riadka, tzn. ich špecifikáciou v súbore `appsettings.json`.

B.2 Druhý blok detekčného nástroja

V tejto sekcii bude popísaná inštalácia a spustenie druhého bloku detekčného nástroja nazývaného Processor.

Požiadavky

Pre inštaláciu je potrebné mať nainštalované nasledujúce prerekvizity:

- Python 3.11.7+
- RabbitMQ
- MongoDB

Inštalácia

Pre inštáláciu a následné spustenie druhého bloku detekčného nástroja je nutné postupovať podľa nasledujúcich krokov:

1. Navigácia do adresára druhého bloku detekčného nástroja:

   ```
   $ cd DGA-Detector/src/Processor/
   ```

2. Vytvorenie a aktivácia virtuálneho prostredia:

   ```
   python -m venv venv
   source venv/bin/activate (Pre Windows: venv\Scripts\activate)
   ```

3. Inštalácia závislostí Pythonu:

   ```
   pip install -r requirements.txt
   ```

4. Zadanie argumentov príkazového riadka druhého bloku detekčného nástroja na konfiguráciu MongoDB, RabbitMQ a ďalších potrebných informácií, ktoré je možné získať použitím parametra `-h` alebo jeho dlhou verziou `-help`.

5. Inštancia MongoDB musí obsahovať nasledujúce kolekcie: Blacklist, Whitelist a Result pre správne fungovanie.

6. Spustenie druhého bloku detekčného nástroja:

   ```
   $ python processor.py
   ```

Rovnako ako prvý blok detekčného nástroja je možné aj tento spustiť bez argumentov príkazového riadka, tzn. ich špecifikáciou v súbore `appsettings.json`.

B.3 Webová aplikácia

V tejto sekcii bude predstavená inštalácia a spustenie webovej aplikácie. Ako prvé bude predstavené spustenie a inštalácia aplikačného programovacieho rozhrania a následne webového rozhrania.

B.3.1 Aplikačné programovacie rozhranie

V tejto sekcii bude popísané spustenie a inštalácia aplikačného programovacieho rozhrania webovej aplikácie.

Požiadavky

Pre inštaláciu je potrebné mať nainštalované nasledujúce prerekvizity:

- Nainštalovaný .NET Core 8.0+ SDK
- Spustená inštancia MongoDB
- Spustená inštancia MySQL (voliteľné)

Inštalácia

Pre inštaláciu a následné spustenie aplikačného programovacieho rozhrania je nutné postupovať podľa nasledujúcich krokov:

1. Navigácia do adresára webovej aplikácie:

   ```
   $ cd DGA-Detector/src/Web application/
   ```

2. Inštalácia závislostí a zostavenie projektu:

   ```
   $ dotnet build
   ```

3. Konfigurácia súboru appsettings.json s detailmi pripojenia k MongoDB a MySQL (voliteľné).

4. Spustenie aplikačného programovacieho rozhrania:

```
$ dotnet run
```

B.3.2 Webové rozhranie

V tejto sekcii bude popísané spustenie a inštalácia webového rozhrania webovej aplikácie.

Požiadavky

Pre inštaláciu je potrebné mať nainštalované nasledujúce prerekvizity:

- Nainštalovaný Node.js
- Spustená inštancia MongoDB
- Spustená inštancia MySQL (voliteľné)

Inštalácia

Pre inštaláciu a následné spustenie webového rozhrania je nutné postupovať podľa nasledujúcich krokov:

1. Navigácia do adresára webovej aplikácie:

```
$ cd DGA-Detector/src/Web\ application/APP
```

2. Inštalácia závislostí:

```
$ npm install
```

3. Spustenie webového rozhrania:

```
$ npm install
```