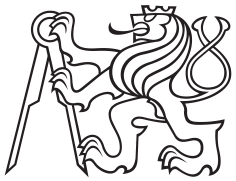
Diplomová práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra řídicí techniky

Detekce anomalit na základě doménových reputací podle klasifikace stažených souborů

Bc. Jiří Bauer
Kybernetika a robotika
bauerji3@fel.cvut.cz

Červen 2018
Vedoucí práce: Ing. Jan Kubr

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Bauer** Jméno: **Jiří** Osobní číslo: **420109**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra řídicí techniky**

Studijní program: **Kybernetika a robotika**

Studijní obor: **Kybernetika a robotika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Detekce anomalit na základě doménových reputací podle klasifikace stažených souborů

Název diplomové práce anglicky:

Anomalies detection using domain reputations based on classifications of downloaded files

Pokyny pro vypracování:

1. Analyzujte možnosti vytváření reputací podle stahování souborů z určitých domén.
2. Zaměřte se na vytvoření doménové reputace URL adres na základě již vytvořených detekcí, oklasifikovaných souborů a cleansetu (množiny neškodných souborů).
3. Následně použijte reputaci pro odhalování false positive detekcí a vytváření detekcí na nových URL adresách se špatnou reputací.
4. Navrhněte a implementujte systém umožňující vypočítat reputace a využít tyto reputace pro detekci anomálií.
5. Součástí implementace bude frontend pro zobrazení výsledků. Řešení otestujte.

Seznam doporučené literatury:

[1] Terry Nelms, Roberto Perdisci, Manos Antonakakis, and Mustaque Ahamad. Webwitness: Investigating, categorizing, and mitigating malware download paths. In Proceedings of the 24th USENIX Conference on Security Symposium, SEC'15, pages 1025-1040, Berkeley, CA, USA, 2015. USENIX Association.
[2] C. Lever, R. Walls, Y. Nadji, D. Dagon, P. McDaniel, and M. Antonakakis. Domain-z: 28 registrations later measuring the exploitation of residual trust in domains. In 2016 IEEE Symposium on Security and Privacy (SP), pages 691-706, May 2016.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Jan Kubr, katedra počítačové grafiky a interakce FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **16.01.2018** Termín odevzdání diplomové práce: **25.05.2018**

Platnost zadání diplomové práce: **30.09.2019**

Ing. Jan Kubr podpis vedoucí(ho) práce	prof. Ing. Michael Šebek, DrSc. podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. Datum převzetí zadání

Podpis studenta

Poděkování / Prohlášení

V první řadě bych rád poděkoval vedoucímu mé diplomové práci Ing. Janu Kubrovi za cenné odborné rady a připomínky k tvorbě práce. Dále bych chtěl poděkovat rodině a přátelům za podporu během celého studia.

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

Praha, 17. 5. 2018

. . . . . .

Abstrakt / Abstract

Cílem této práce bylo vytvořit systém využívající doménovou reputaci a na jejím základě detekovat anomality ve stahovaní souborů na základě znalosti jejich zdrojové URL.

Doménová reputace je založená na databázi, která uchovává historii stažených souborů spolu s jejich klasifikací a URL, ze kterých byly staženy. Implementovaný systém v síťovém provozu detekuje situace, při kterých dochází ke stažení souborů, které jsou detekovány antivirem, z domén s dobrou reputací. Dále hledá domény, ze kterých se stahuje velké množství škodlivého softwaru.

Vytvořený systém během tří měsíců provozu nalezl 87 škodlivých domén a 234 podezřelých PE souborů označil jako čisté. Výsledný systém je v současné době součástí antivirového programu Avast, čímž pomáhá chránit miliony uživatelů po celém světě.

Klíčová slova: Marlowe, doménová reputace, malware, bezpečnost, detekce anomalit, URL, PE

The goal of this thesis is to create a system using domain reputation based on which we shall detect anomalies in file downloads knowing their source URLs.

The domain reputation is based on a database containing history of downloaded files together with their classification and source URL. The implemented system detects downloading of files detected by AV angines from domain with a good reputation and also finds domains serving a large portion of malware.

During the first three months the resulting system found 87 malicious domains and classified 234 suspicious PE files as clean. Nowadays the system is a part of Avast antivirus helping to protect milions of users around the world.

Keywords: Marlowe, domain reputation, malware, security, anomalies detection, URL, PE

Title translation: Anomalies detection using domain reputations based on download files classification

Obsah

Obrázky / Obrázky

Tabulky /

Kapitola 1
Úvod

Žijeme v době, kdy je internet součástí téměř každé domácnosti a lidé jsou zvyklí ho využívat ke spoustě věcí. Od komunikace po sociálních sítích a sledování televize přes nakupování v on-line obchodech a využívání internetového bankovnictví až po vzdálené ovládání IoT[1] zařízení po celém domě i mimo něj.

S touto rostoucí popularitou je stále důležitější a také složitější zajistit bezpečnost pro všechny uživatele, protože ne všichni chtějí internet využívat k legálním účelům. Ne každý si uvědomuje, kolik nepříjemností jim může způsobit nepozornost při práci s počítačem, jako je třeba nečtení dialogů při instalaci programů aj.

Útočníci hledají nové cesty, kudy by se dostali k uživatelům do jejich počítačů, aby se zmocnili jeho kontroly, kradli osobní údaje nebo například zašifrovali celý disk a po jeho vlastníkovi vymáhali výkupné pro jeho zpětné dešifrování. To si kladou za cíl tvůrci tzv. ransomwaru. Největší vlnou těchto útoků byl *WannaCry* ransomware, který během května 2017 zasáhl více než 100 milionů uživatelů po celém světě [6]. Útočníci využili bezpečnostní mezery u uživatelů MS Windows, kteří nenainstalovali aktualizaci systému vydanou o 2 měsíce dříve. Aktualizace řešila právě zranitelnost SMB protokolu, kvůli které se *WannaCry* mohl šířit bez uživatelské interakce.

Ne vždy stačí mít nainstalované nejnovější aktualizace softwaru pro udržení zařízení v bezpečí. O další zabezpečení se starají anti-virové programy. Jedním z těchto antivirových produktů je Avast chránící svých více než 435 milionů[2] uživatelů po celém světě před 3,5 miliardami útoků každý měsíc. Díky obrovské uživatelské základně je Avast schopen monitorovat a analyzovat síťový provoz na všech kontinentech.

Pro některé útoky ani není nutné, aby útočníci infikovali počítač oběti. S vzestupem kryptoměny se stali velice populárními její těžiče, které poskytovatelé mnohých služeb na internetu zabudovávají do svých stránek pomocí JavaScriptu. Skript (Crypto Miner) běží na počítači oběti, využívá její procesor a těží kryptoměnu pro majitele stránek. Tento postup je v dnešní době alternativou pro umisťování reklam, proti kterému se již uživatelé brání využíváním *AdBlocků*[3]. Právě detekce bránící neautorizovanému využívání hardwaru u uživatelů je v současné době jednou z nejvíce aktivních. Například 3. prosince 2017 ochránil Avast svoje uživatele před téměř 35 miliony pokusů o těžení kryptoměny Monero. Útočníkům se pravděpodobně podařilo modifikovat jedno z populárních rozšíření internetového prohlížeče Google Chrome, aby spouštělo JavaScript, který na uživatelově procesoru nepovoleně Monero těžil [5].

V této práci budu spolupracovat s antivirovou bezpečností společností *Avast Software, s.r.o.*, bude využito dat a prostředků této společnosti a výsledky práce budou použity v antivirovém produktu Avast.

[1] IoT (Internet of Things = „internet věcí“) označuje propojená elektronická zařízení jako jsou elektrospotřebiče, automobily, televize aj., která spolu komunikují a spolupracují po síti.

[2] https://cdn2.hubspot.net/hubfs/2706737/media-materials/corporate-factsheet/Avast_corporate_factsheet_A4_en.pdf

[3] *AdBlock* je rozšíření webových prohlížečů využívající blacklist stránek poskytujících reklamní obsah k jejich blokování.

1.1 Cíle práce

Hlavním cílem této práce je vytvořit systém, který bude schopen nalézt anomálie mezi stahovanými soubory. Za anomálii je považován soubor detekovaný Avastem stažený z domény, která v minulosti hostovala převážně neškodné soubory. Na druhou stranu je také anomálií nedetekovaná doména, ze které se stahuje velké množství malwaru.

Abych byl vůbec schopen vytvořit takový systém, je nutné vytvořit databázi, která bude shromažďovat data o souborech a jejich URL zdrojích, ze které budu schopen přiřadit každé doméně její reputaci. Tato reputace bude vyjadřovat míru škodlivosti domény, která je vyjádřena počtem stažených škodlivých a neškodných souborů.

V této práci se omezím pouze na spustitelné soubory (dále v tomto textu budou značeny jako PE[1] soubory). Toto omezení zavádím vzhledem k faktu, že i kompromitovaná doména, která obsahuje nebezpečný obsah, bude s největší pravděpodobností obsahovat i čistý obsah, typicky html, php nebo css soubory, které tvoří vzhled stránek pro uživatele. Při výpočtu reputace domény by potom tyto soubory způsobovaly zkreslení.

Za účelem odhalení anomalit v proudu stažených PE souborů vytvořím systém *Marlowe*[2], který bude schopen na základě vytvořené databáze detekovat a ohlásit podezřelé incidenty jako například podezřelou detekci na souboru staženého z domény, která v minulosti sloužila k šíření neškodného obsahu. *Marlowe* by také měl najít domény, které hostují větší množství škodlivých PE souborů.

Dále vytvořím grafické rozhraní pro zobrazení výsledků s vizualizací dalších dat, která umožní manuální analýzu.

1.2 Struktura práce

Kapitola 2 se zabývá popisem a testem existujících volně dostupných služeb umožňující vyhledat reputace pro doménová jména. Dále jsou zde rozebrány práce jiných autorů, kteří se zabývali různými způsoby jak klasifikovat domény či jim přiřadit určité reputace. V poslední části se seznámíme se způsoby, jak se aktuálně zpracovávají domény v Avastu.

V kapitole 3 bude zavedena doménová reputaci a popsáno, jak bude používána. Představím, jaká vstupní data budu při vývoji využívat a jak budou zpracovávána. Budu se zde zabývat návrhem databáze *Cobra*, která bude uchovávat informace pro výpočet doménové reputace. Implementace *Cobry* není součástí této práce, proto budu popisovat právě jenom její návrh a použití. Dále kapitola obsahuje návrh obou stěžejních částí práce, a to je klasifikace domén a detekci anomálních detekcí.

Kapitola 4 popisuje softwarové technologie, které budou využity při implementaci systému. Zaměřím se zejména na frontové systémy a krátce proběhne seznámení i s programovacími jazyky Python a Scala.

V kapitole 5 budu popisovat implementaci jednotlivých aplikací a skriptů. Nejprve se budu věnovat aplikacím pro zpracování vstupních dat a jejich vložení do *Cobry* a poté jejich zpracování a analýze.

Kapitola 6 se zabývá testováním aplikací pro zpracování vstupních dat z hlediska jejich datové propustnosti. Dále obsahuje analýzu výsledků, kterých *Marlowe* dosáhl po uvedení do provozu.

[1] PE (portable executable) formát souborů je formát pro spustitelné soubory, dynamické knihovny, objektový kód atd. použitý v operačním systému Windows. [10]

[2] Název *Marlowe* je převzat z knih autora Raymonda Chandlera, jejichž ústřední postavou je detektiv Philip Marlowe.

V závěru (7) zhodnotím dosažené výsledky práce a shrnu možné pokračování ve vývoji.

Kapitola 2
State of the art

V této kapitole se budu zabývat již implementovanými veřejnými službami, které klasifikují domény na základě jejich reputace. Tyto služby otestuji na sadě domén a porovnám jejich výsledky. Dále budu diskutovat práce jiných autorů týkající se této oblasti a způsoby jakým se přistupuje ke klasifikaci domén v Avastu v dnešní době.

2.1 Existující služby pro klasifikaci domén na základě jejich reputací

Na internetu lze nalézt několik produktů od různých poskytovatelů, kteří tvrdí, že jejich produkt na základě doménové reputace dokáže například třídit e-maily. Z těchto produktů jsem pro test vybral *Email & Web Traffic Reputation Center* (WTRC) od skupiny Talos bezpečnostní společnosti Cisco a *WebAdvisor* (WA) od McAfee. Další produkty buď nemají dostupné veřejné API nebo pro žádné dotazované domény nevracejí žádné výsledky[1]. WTRC rozděluje domény podle jejich reputace do tří skupin *dobrá, neutrální, slabá*. WA používá trochu jinou terminologii. Domény také rozděluje do tří kategorií, ale místo reputace domény je v názvu kategorií míra risku s doménou spojená. Místo *dobrá, neutrální* a *slabá* reputace WA používá *minimální risk, střední risk* a *vysoký risk*. Pro vyšší přehlednost budu používat terminologii zavedenou podle WTRC.

Pro testování jsem sestavil dvě množiny domén. První z nich obsahuje domény zablokované Avastem během posledních 12 hodin a druhá domény, které byly Avastem zablokovány během stejné doby před týdnem. Všechny tyto domény byly ověřeny pomocí služby *Google Safe Browsing*[2], abych potvrdil jejich skutečnou škodlivost. Každou z těchto 200 domén jsem otestoval oběma systémy, abychom zjistil, kolik z nich bude správně označeno jako domény se *slabou* reputací.

Na obrázku 2.1 je vidět porovnání výsledku testu. Rozdíly mezi klasifikací obou skupin domén (zablokovaných během posledních 12 hodin a před týdnem) jsou velice nepatrné, proto mezi nimi v grafu nerozlišuji. Produkt McAfee má znatelně vyšší úspěšnost v počtu domén označených jako *slabá*, a to 70 % oproti 19 % u Cisca. Naproti tomu Cisco neoznačilo žádnou z domén jako *dobrá*, detekovalo tedy alespoň nějakou míru nebezpečí u všech domén, o kterých mělo dostatek informací. Je ale potřeba zmínit fakt, že o 30 % všech testovaných doménách neměla služba Cisca dostatek dat, a proto jim nebyla přiřazená žádná reputace. Získané výsledky napovídají, že Cisco má reputační systém přesnější, ale pravděpodobně nemají dostatek dat pro poskytnutí výsledků pro

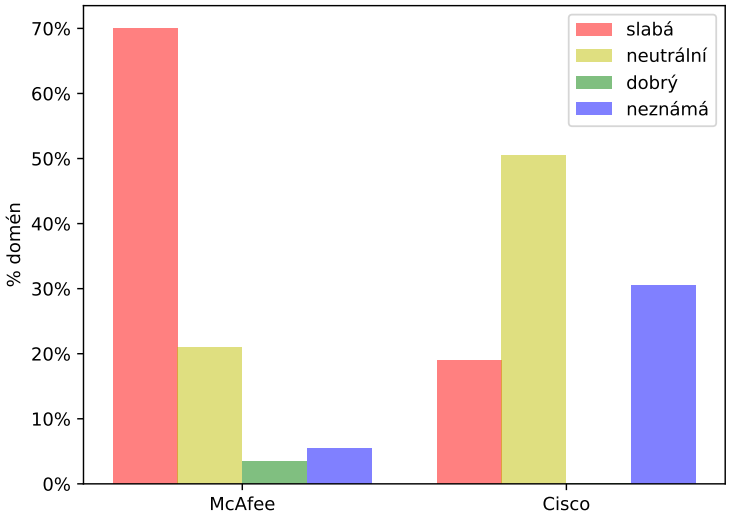
[1] Například *Barracuda Central* http://www.barracudacentral.org/lookups/lookup-reputation pro použité domény vypisuje, že je nemá ve své databázi, *WatchGuard Reputation Authority* www.reputationauthority.org/index.php vrací pouze neutrální výsledky a *Trend MICRO Reputation Services* https://www.ers.trendmicro.com/ funguje jenom pro IP adresy.

[2] API Safe Browsing umožňuje uživatelům dotazovat neustále doplňovanou databázi Googlu škodlivých webových zdrojů jako jsou například phishingové nebo jinak podvodné stránky. https://transparencyreport.google.com/safe-browsing/overview

všechny domény. I když měl produkt McAfee vyšší procento správně klasifikovaných domén, tedy 70 %, celých 5 % domén označil jako domény s *dobrou* reputací.

Testoval jsem také skupinu 100 domén ze seznamu nejnavštěvovanějších domén podle Alexa[1]. Obě služby shodně označily reputaci domény jako *dobrou*.

Vzhledem k tomu, že při testovaní antivirových programů se ty nejlepší dokáží dostat nad hranici 99 % správně oklasifikovaných škodlivých souborů, není možné reputační službu se 70% úspěšností označit jako dostatečnou[2].



Obrázek 2.1. Porovnání doménových reputačních služeb Cisca a McAfee

2.2 Související práce

V době psaní této práce neexistoval žádný veřejný nástroj pro klasifikaci domén na základě stažených souborů, který by měl dostatečné množství dat pro správnou funkčnost. Jeden z pokusů je popsán v [8], kde autoři vytvořili systém *WebWitness*, který analyzoval akademickou síť. Pro vytvoření správně fungujícího systému je ale potřeba monitorovat co nejvyšší možný počet zařízení a mít přístup ke staženým souborům, aby bylo možné vytvořit databázi a podle ní počítat reputaci domén. Právě toho lze dosáhnout u takového antivirového giganta jako je Avast, pro který fungují jeho uživatelé jako senzory sbírající data o souborech a jejich zdrojích po celém světě.

Drobnější projekty nemají k takovým informacím přístup a proto se většina podobných prací zaměřuje na DNS[3] komunikaci, WHOIS[4] databáze nebo na monitorování lokálních sítí.

[1] Alexa Internet je americká společnost analyzující internetový provoz https://www.alexa.com/topsites

[2] Je nutné brát v potaz, že klasifikace domén je mnohem méně exaktním problémem v porovnání s klasifikací souborů. Škodlivost souboru se s časem nemění, pokud se soubor změní, jde už o jiný soubor. Na druhou stranu doména může během svojí životnosti změnit majitele i celý svůj obsah. Nelze proto očekávat, že úspěšnost klasifikování domén bude stejně vysoká jako u souborů.

[3] DNS (systém doménových jmen) je protokol, pomocí kterého si DNS servery vyměňují informace. Jde o hierarchickou strukturu pro vzájemné převody mezi doménovými jmény a IP adresami uzlů v síti.

[4] WHOIS je databáze údajů o registraci doménových jmen a jejich majitelích. Obsahuje informace jako datum registrace a expirace, email, jméno a organizaci majitele atd.

Felegyhazi a spol. ve své práci [4] navrhují jak proaktivně vytvářet blacklist domén za použití DNS a WHOIS databáze. Na základě sady známých škodlivých doménových jmen predikují skupiny domén, které budou v budoucnu pravděpodobně také sloužit ke škodlivým účelům. V dalším kroku používají dvě skupiny extrahovaných vlastností. Registrační informace (z WHOIS) a informace o autoritativním jmenném serveru (NS) pro inferenci dalších sad doménových jmen s podezřením ke škodlivému chování. Z práce vyplývá, že NS škodlivých domén často splňují následující kritéria:

1. NS doména byla nedávno registrována;
2. NS se překládají samy na sebe.

Hlavní výhodou tohoto projektu je právě proaktivita blokování doménových jmen, tzn. že domény mohou být zablokovány ještě před tím, než začnou sloužit svému účelu.

V [1] je popsán systém *Notos*, což je dynamický reputační systém podle DNS. Autoři vytvořili síť doménových a zónových vlastností, které charakterizují použití a chování domén. Mezi tyto vlastnosti patří například počet škodlivých souborů, které se pokusí komunikovat s danou doménou, počet IP adres, na které se doménové jméno přeloží, vyskytujících se na veřejných blacklistech. Dále byly využity také textové vlastnosti doménových jmen. Na základě těchto vlastností je *Notos* schopen se naučit modely chování škodlivých domén a vypočítat reputační skóre pro nové domény.

C. Lever a spol. ve své práci [7] poukazují na možnost zneužití zbytkové důvěry/nedůvěry, která se přenáší i přes změny změny vlastníka doménového jména. Tento přenos je problémem v obou směrech. Pokud doména, která byla v minulosti využívána ke zločinným účelům, změní svého vlastníka a začne být provozována k běžným legálním účelům, může stále zůstat na seznamech blokovaných domén antivirových společností a jiných institucí. Stejně tak může útočník v pravý čas (po uplynutí předplacené doby) přeregistrovat doménu s dobrou reputací na své jméno, stát se jejím vlastníkem a využít ji například k šíření malwaru. V tomto případě naopak může doména zůstat bez povšimnutí právě kvůli zbytkové důvěře z minulosti. Autoři ve svém článku rozebírají některé příklady takového zneužití z minulosti a jako obranu představují *Alembic*: Algoritmus, který na základě behaviorální analýzy dokáže předpovědět možné změny ve vlastnictví domén bez nutnosti mít přímý přístup k WHOIS databázím.

V [3] se Antonakais a spol. snaží najít způsob, jak spolehlivě detekovat škodlivý software používající DGA[1]. Vytvořili *Pleiades*, který zpracovává proud neúspěšně přeložených doménových jmen z DNS, tzv. nx-domén (neexistujících domén), a shlukuje je do klastrů podle kritérií založených na textové podobnosti. *Pleiades* je poté schopen najít infikované počítače, které se pokoušely na nx-domény dotazovat, stejně tak jako C&C servery, které mají podobné doménové jméno jako ostatní nx-domény patřící do stejného klastru.

Ve článku [8] je popisován systém se jménem *WebWitness*, který zaznámenává všechny HTTP transakce. Každá odpověď této transakce je zkontrolována, zda neobsahuje spustitelný soubor. Pokud ano, je poté soubor zkontrolován pomocí *VirusTotalu*[2]. Pro škodlivé soubory je *WebWitness* schopen zpětně zrekonstruovat cestu webových

[1] DGA (algoritmus pro generování doménových jmen) je často využívaný způsob pro šíření malwaru. Infikovaný počítač pomocí tohoto algoritmu generuje doménová jména a snaží se s nimi komunikovat. Jenom některé z vygenerovaných domén se ale pomocí DNS dokáží přeložit na IP adresu. Tato adresa patří řídícímu serveru (C&C), který potom infikovanému počítači posílá další instrukce nebo si od něj například může nechat posílat důvěrné informace.

[2] VirusTotal je webová služba shlukující přes 70 antivirových scannerů a URL/doménových blacklistů umožnující uživatelům nahrát soubory a zjistit, zda nejsou nebezpečné.

stránek, po kterých k němu uživatel došel. Systém je také schopen každou HTTP transakci přiřadit do jedné ze skupin:

1. *driven-by*: stažený soubor je transparentně doručen k uživateli pomocí napadení webového prohlížeče;
2. *social-engineering*: stažení souboru je iniciováno výslovnou interakcí od uživatele (např. kliknutím myši);
3. *update/drop*: stažení která nespadají do žadné z prvních skupin.

Rozdělení do těchto skupin umožňuje přiřadit jednotlivým hrozbám úroveň jejich nebezpečnosti. Hlavní výhodou tohoto projektu je schopnost najít kromě domén, které jsou přímým zdrojem staženého souboru, i ty domény, které stojí na samém začátku řetězce HTTP transakcí.

Posledním doménovým reputačním systémem, který budeme zmiňovat je *Kopis* ([2]). *Kopis* je založen na DNS vrstvě, ale na rozdíl od *Notos* se nespoléhá na monitorování lokálního rekurzivního DNS. Nelms a spol. se snažili využít globálního pohledu, který získali monitorováním vyšší hierarchie DNS. To jim umožňuje odhalit a zablokovat spojení směřující ke škodlivým doménám ještě než dorazí k lokálním sítím. Další výhodou oproti předchozím je také nezávislost na reputacích IP adres, což bude hrát ještě mnohem větší roli v budoucnu, až se plně rozvine IPv6, čímž se prostor všech adres mnohonásobně zvýší a monitorování jejich reputací se stane ještě složitějším problémem.

2.3 Klasifikace domén v Avastu

Nejdůležitějším nástrojem pro klasifikaci domén je v současné době *RevIP*. Jedná se o MySQL databázi, která uchovává vztahy doména – IP adresa a počet daných překladů. Díky RevIPu jsou analytici schopni poměrně snadno najít:

- k dané doméně domény jí podobné (překládající se na stejné IP adresy);
- domény překládající se na danou IP adresu;
- napadení DNS[1] (odlišné domény se překládají na stejné IP adresy např. *google.com* a *facebook.com*);
- nesprávně zablokované IP adresy (pokud se na adresu překládají zjevně legitimní doménová jména);
- nesprávně blokované domény (jedna zablokovaná doména mezi podobnými nezablokovanými u nezablokované IP adresy).

RevIP je velice důležitým zdrojem dat, ale jen těžko se daří zautomatizovat vytváření nových detekcí, a proto je potřeba manuální práce analytika.

Aby ji nebylo potřeba tolik, vyvíjí se nástroj jménem *Mopsus* [9]. Jedná se o klasifikátor založený na náhodných lesích rozhodovacích stromů. Modely jsou trénovány na datech z RevIPu a na lexikálních vlastnostech URL adres. Nástroj zatím čelí několika problémům, jako jsou nedostatečně označená čistá data v RevIPu nebo přeučení modelů, které by mohlo nastat ve chvíli, kdy se podle *Mopsus*u začnou vytvářet nové URL detekce a další modely se tak budou učit na datech, která jsou klasifikována podle výstupů těch starších. Kvůli těmto nedostatkům se ještě *Mopsus* nedostal do produkčního chodu.

[1] Nejčastějšími formami útoků na DNS protokol jsou *DNS hijacking* a *DNS cache poisoning*. Při *DNS hijacking* útočník pozmění TCP/IP konfiguraci klienta a ten se pak při dotazu na doménová jména obrací na podvržený DNS server. *DNS cache poisoning* je útok na cache paměť DNS serveru, kam útočník nahraje pozměněné překladové tabulky. Výsledkem obou útoků je nesprávný překlad doménových jmen na IP adresy.

Další zajímavá data o doménách lze vyextrahovat z WHOIS databází. Nově registrovaná doménová jména jsou kontrolována se slovníkem klíčových slov, který má odhalit pokusy o podvod založené na podobnosti s legitimními doménami. Dále lze v WHOIS databázi najít skupiny domén, které jsou zaregistrovány na stejné jméno, organizaci nebo e-mail. Poté lze klasifikovat takovou skupinu domén jako celek. Častým podezřelým jevem je například registrace doménového jména v exotických zemích nebo za poplatek vynechané osobní údaje v WHOIS záznamech.

I přes výše zmíněné nástroje probíhá stále větší část zpracování doménových jmen manuálně. Právě to je důvod, proč vyvíjet *Marlowe*, který má propojit databázi doménových jmen s databází souborů a kombinovat tak klasifikace obojího.

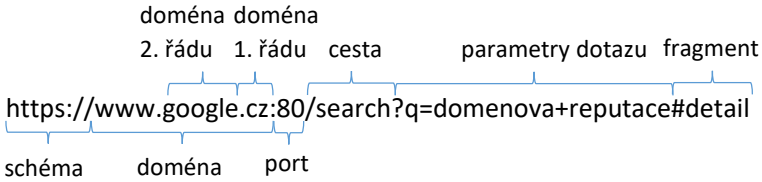
Kapitola 3
Analýza a návrh řešení

V této kapitole se budu zabývat doménovou reputací, jak se ji pokusím vypočítat a jak ji dále budu využívat. Dále zde dojde k návrhu systému pro sběr dat, která budou potřeba pro budování databáze závislostí mezi staženými soubory a jejich zdrojovými URL. Nakonec navrhnu databázi, která bude všechna tato data uchovávat a bude schopna obsloužit četné dotazy.

3.1 Doménová reputace

V sekci 2.2 jsem shrnul práce, které se snažily vybudovat doménovou reputaci na základě DNS, WHOIS nebo analýzy cest, které vedly uživatele ke stažení škodlivých souborů. Naše reputace doménových jmen bude budována podle klasifikace stažených souborů. Cílem bude dostat se do stavu, kdy všechny domény se *špatnou* reputací budou přidány na blacklist, tj. jejich veškerý obsah bude pro uživatele Avastu nedostupný. Na druhou stranu také budu směřovat k tomu, aby při stahování z domén s *dobrou* reputací nedocházelo k detekování souborů jako škodlivých. Pokud by k takovéto situaci došlo, *Marlowe* by ji měl vyřešit nebo alespoň upozornit na možnou FP detekci[1].

Nejprve je potřeba zmínit, že když budu mluvit o doménové reputaci, pokud nebude uvedeno jinak, bude myšlena reputace domény druhého stupně. Počítání reputace pro celou doménu včetně všech (případně podmnožiny) jejích subdomén by samozřejmě také mohlo být užitečné, ale aby vůbec bylo možné počítat reputace z nějakých nezanedbatelných čísel, bylo by nutné z každé stáhnout více souborů, což například při využití DGA, kdy jsou generované jenom subdomény, ale doména druhého řádku zůstává stejná, vůbec nemusí být možné. Pro rozdělení URL na její části budu používat standardní notaci, příklad URL a její rozdělení na části je zobrazeno na obrázku 3.1.



Obrázek 3.1. Příklad URL a její části

Doménovou reputaci DR potom definuji jako

$$DR(d) = (n_{cln}, n_{mal}), \qquad n_{cln}, n_{mal} \in \{0, 1, 2, ...\},$$

kde d je doména druhého řádu, n_{cln} resp. n_{mal} je počet čistých resp. škodlivých PE souborů stažených z URL na doméně d.

[1] FP - false positive se používá pokud je něco nesprávně označeno, že přísluší do určité skupiny. Pokud tedy označena detekce na souboru jako FP, potom je to detekce, která nesprávně detekuje čisté soubory jako škodlivé.

$n_{cln} >> n_{mal}$	$n_{cln} \sim n_{mal}$	$n_{cln} << n_{mal}$
dobrá	neutrální	špatná

Tabulka 3.1. Rozdělení doménové reputace podle reputačních koeficientů

Na základě takto definované reputace DR poté můžu rozdělit domény do tří základních skupin (tabulka 3.1).

Do skupiny domén s dobrou reputací by měly patřit domény, které sdílejí větší množství legitimních PE souborů. Každá malware detekce souborů stažených z těchto domén je podezřelá a měla by být vyřešena.

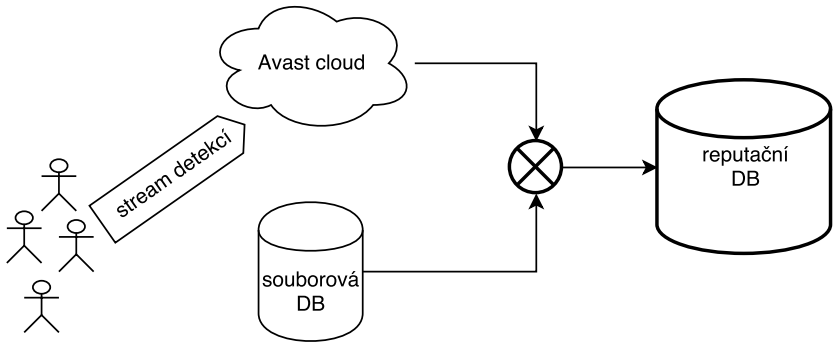
Do skupiny domén s neutrální reputací spadají domény, ze kterých se stahují čisté i škodlivé soubory. Typicky by mělo jít o internetová úložiště, která samozřejmě nejsou určená k šíření škodlivého softwaru, ale vždy nějakou dobu trvá, než se takovéto soubory odstraní. Dále sem budou patřit i například domény, které jsou již napadené a spolu s původním legitimním obsahem už šíří i ten škodlivý. Takovéto domény už jenom na základě jejich reputace nebudu schopen odhalit. Je nutné všechny tyto incidenty podchytit ještě předtím, než se počet stažených škodlivých souborů dokáže přiblížit počtu stažení souborů čistých.

Skupina domén se špatnou reputací by měla obsahovat domény s větším počtem stažených škodlivých souborů a nízkým počtem čistých PE souborů. Tyto domény by měly přidány na blacklist, aby na ně byl blokován přístup.

3.2 Vstupní data

Aby bylo možné počítat reputace domén, je nutné nejprve vytvořit databázi, která bude udržovat vztahy mezi URL adresami a soubory z nich stažených. Návrhem databáze se bude zabývat sekce 3.3.

Jako vstupní data bude použita databáze souborů, konkrétně její dva sloty: jeden obsahující škodlivé PE soubory (malware set) a druhý obsahující čisté PE soubory (clean set). Jako další zdroj bude stream detekcí na PE souborech od uživatelů. Zde budou zajímavé soubory stažené z webu, u kterých lze použít jejich vazbu na zdrojovou URL (obrázek 3.2).



Obrázek 3.2. Vstupní data do reputační databáze

Je důležité si uvědomit fakt, že škodlivých souborů a jejich zdrojových URL bude dostatek jenom z detekčních streamů, zatímco čisté soubory nejsou detekované, a tak je o nich méně záznamů. Kvůli tomu bude nutné zacházet s oběma skupinami dat jiným způsobem.

S ohledem na oba hlavní úkoly, které by měl *Marlowe* plnit, je klíčová část čistých souborů. Právě čisté soubory jsou důkazem toho, že doména slouží k legitimním účelům. Pokud se zaměřím na blokování domén se špatnou reputací, cílem bude je zablokovat v co nejranějším stádiu jejich životnosti. Proto je potřeba zpracovat detekované (škodlivé) soubory ideálně v reálném čase a stavět na nich reputace. K tomu poslouží stream detekcí od uživatelů. Zároveň je nutné mít sadu všech čistých souborů, aby nedošlo k zablokování domén, které tyto soubory hostují. Podobně i při hledání anomálních detekcí na doménách s dobrou reputací je potřeba mít dostatečnou množinu čistých souborů, abychom bylo možné identifikovat domény s dobrou reputací.

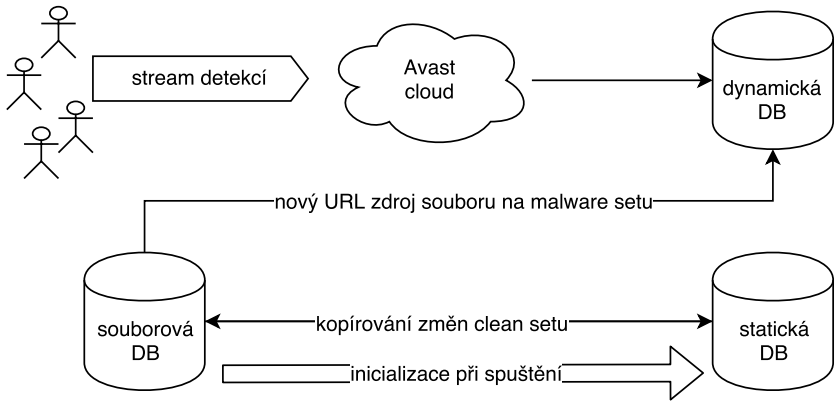
Z toho plyne, že čisté soubory musí být v reputační databázi „napořád“, zatímco škodlivých souborů se bude uchovávat pouze časové okno pro zajištění pokrytí současné situace.

3.3 Databáze Cobra

Jak již bylo zmíněno v 3.2, databáze bude rozdělena na dvě části. První z nich – *statická* – bude obsahovat čisté soubory, tj. celý PE clean set ze souborové databáze se svými zdrojovými URL. Databáze sice bude statická, tj. v čase se nebude příliš měnit, ale zároveň musí podporovat přidávání nových souborů stejně tak jako jejich mazání. Při inicializaci této databáze tedy dojde k jejímu naplnění aktuálním obrazem PE clean setu souborové databáze a dále bude sledovat její změny (přidávání a odebírání souborů).

Druhá část databáze – *dynamická* – bude konzumovat a uchovávat příchozí soubory se svými zdrojovými URL převážně ze streamu detekcí, ale bude také uchovávat soubory, které byly nově přidané na malware set souborové databáze. Pokud v souborové databázi přibude škodlivému souboru nový URL zdroj, potom i tato skutečnost bude vložena do dynamické databáze. Tato část databáze bude sloužit pro zachycení dat z posledního týdne. Incidenty starší sedmi dnů tedy budou z databáze mazány.

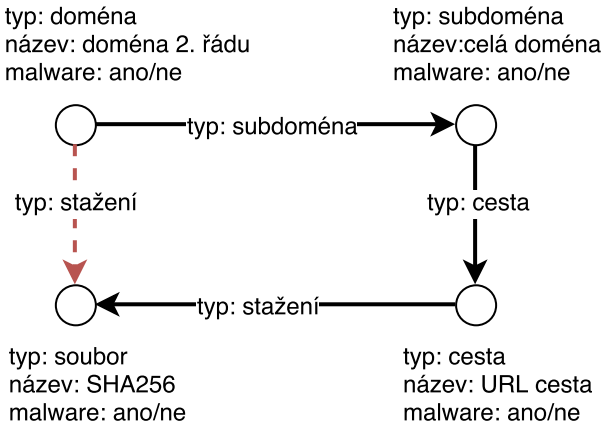
Schéma databáze a její připojení na zdrojová data je znázorněno na obrázku 3.3.



Obrázek 3.3. Schéma reputační databáze

Jedná se o grafovou databázi, u které jsou jednotlivé incidenty reprezentovány orientovanými grafy. Každý z orientovaných grafů představující jeden incident, tj. soubor stažený z dané URL, je tvořen čtyřmi uzly: doména, subdoména, cesta a soubor. Doménou je myšlena doména druhého řádu, zatímco uzel typu subdoména charakterizuje celou doménu i se všemi subdoménami. Cesta URL adresy je použita podle notace zavedené dříve (obrázek 3.1) a soubor je charakterizován svou SHA256 haší.

Znázornění incidentu pomocí grafu je vyobrazeno na obrázku 3.4. Každý uzel je kromě svého typu a názvu popsán také booleovskou proměnnou *malware*. Tato hodnota ukazuje, zda je tato část URL (případně soubor) blokována[1] Avastem.

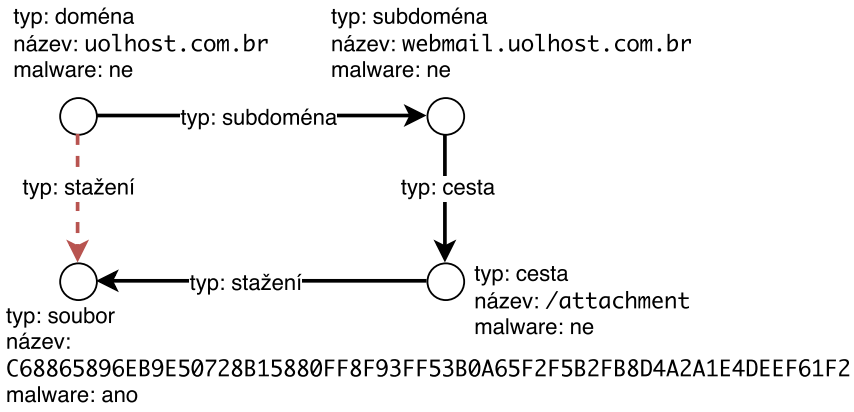


Obrázek 3.4. Zobrazení incidentu pomocí grafu

Z logiky postupování od nejobecnějšího uzlu k nejkonkrétnějšímu, tj. od domény 2. řádu přes subdoménu a cestu až k souboru, by stačily pouze 3 vazby (vyznačeny na obrázku černou barvou). Vzhledem k tomu, že nejčastější dotazy do databáze směřují právě na uzly typu *doména* a *soubor*, je vhodné přidat do grafu další vazbu typu *stažení* mezi tyto uzly. Tím se výrazně zrychlí dané dotazy, protože pak nebude nutné prohledávat zbytek grafu.

V této práci budu využívat pouze uzly typu *doména* a *soubor* a právě přidanou vazbu typu *stažení* (zobrazena červenou barvou), ale vzhledem k dalšímu využití této databáze je nutné počítat při jejím návrhu se všemi částmi URL.

Příklad jednoho incidentu (souboru staženého z URL) je na obrázku 3.5. Jde o stažení souboru s haší *C688...61F2* z *http://webmail.uolhost.com.br/attachment*. Z obrázku je vidět, že žádná část URL není blokována, zatímco soubor je klasifikován jako škodlivý.



Obrázek 3.5. Příklad grafu incidentu

[1] URL adresy lze blokovat podle jejich libovolné části. Podle domény nebo její části, podle cesty nebo její části i podle jejich libovolné kombinace. *Malware* bit u uzlů ukazuje pouze zda je daná konkrétní část URL zablokovaná, ne celá URL, jejíž je součástí. Například pokud by byla zablokována URL *google.com/search*, pak by u uzlu typu *cesta* s názvem */search* byla hodnota *malware* bitu 0. Pokud by ale na blacklistu byla cesta */search*, potom by hodnota tohoto bitu byla 1 atp.

Pro dotazování databáze je využita podmnožina jazyka *Cypher*[1]. Obecná struktura dotazu má následující podobu:

```
MATCH definice_vztahů RETURN seznam_proměnných LIMIT limit
```

Definicí vztahů je myšlen popis uzlů a hran, které má výsledek dotazu splňovat. Uzly se v dotazech definují

```
(alias{type='typ uzlu', name='název uzlu'})
```

a hrany

```
-[type='typ hrany']->
```

Všechny parametry uzlů i hran jsou volitelné. Například dotaz pro spočítání všech škodlivých a všech čistých souborů stažených z domény *google.com* by mohl vypadat následovně:

```
MATCH IN *
(domain{type='domain', name='google.com'})
-[type='download']->
(file{type='file'})
RETURN domain.name, MALCOUNT(file), CLNCOUNT(file)
LIMIT 1000
```

MALCOUNT a CLNCOUNT jsou agregační funkce počítající počet škodlivých respektive čistých uzlů (podle atributu *malware*) na které je funkce aplikována.

Databáze bude přijímat data pomocí tří vstupů. Dva z nich jsou frontové systémy, ve kterých jednotlivé zprávy obsahují data pro vložení daných incidentů (tedy definice uzlů a hran). Frontový systém směřující do *statické* databáze dále obsahuje také zprávy pro mazání grafů. Toho bude využito při odstranění souborů z clean setu souborové databáze. Pokud k tomu dojde, musí se ze *statické* databáze smazat všechny grafy obsahující uzel s tímto souborem.

Třetí vstup je určen k počáteční inicializaci *statické* databáze. Nahraje se do ní obraz clean setu souborové databáze a případné změny od doby jeho vytvoření se aktualizují z frontového systému. Kvůli tomu je nutné, aby retence frontového systému byla alespoň tak velká, jako bude perioda vytváření souboru obsahující obraz clean setu souborové databáze.

3.4 Klasifikace domén se špatnou reputací

Jedním z hlavních úkolů *Marlowe* je zablokovat domény, které mají špatnou reputaci (tj. obsahují výrazně více škodlivých PE souborů než těch čistých).

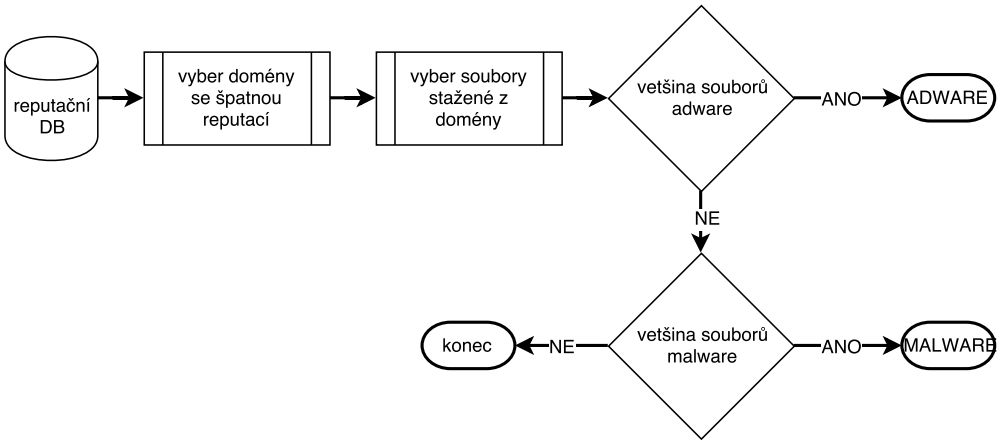
Nejprve je potřeba zmínit, že ne všechny soubory, které se do databáze dostanou s atributem *malware: ano*, musejí být nutně úplně škodlivé. Z pohledu souborového štítu antiviru jsou *adwarové*[2] soubory považované za nechtěné a tento štít je tedy blokuje. Na druhou stranu webový štít (tj. i blokování na základě URL) adware často neblokuje. Důvodem je snaha o nesnížení pohodlí uživatelů. Pokud už je jeho počítač infikovaný nějakým adwarem, je často méně rušivé pouze zavírat nově otevřená okna s reklamami,

[1] Cypher je dotazovací jazyk grafové platformy Neoj4, který svou syntaxí srozumitelně popisuje uzly a jejich vztahy. Více na https://neo4j.com/developer/cypher-query-language/.

[2] Adware je software šířící reklamy. Často jde o programy napadající internetový prohlížeč, které přesměrovávají uživatele na stránky s více či méně užitečným reklamním obsahem.

která se otevřela kvůli adwaru, než bojovat s vyskakovacími okny antiviru, který upozorňuje na zablokovanou komunikaci s doménou, která je oklasifikovaná jako adware. Jde o to, že ani zablokováním dané domény se počítač od adwarové infekce nevyléčí, pouze dojde k větší iritaci uživatelů.

Z toho plyne povinnost nepovažovat automaticky všechny domény jako nutně škodlivé z pohledu webového štítu, pokud se z nich stahují soubory škodlivé z pohledu souborového štítu. Pro jednoduché rozlišení adwaru od malwaru lze využít názvy detekcí, které dané soubory detekují. Názvy detekcí často obsahují klíčová slova, aby bylo na první pohled rozeznatelné, proti jaké hrozbě jsou cílené.



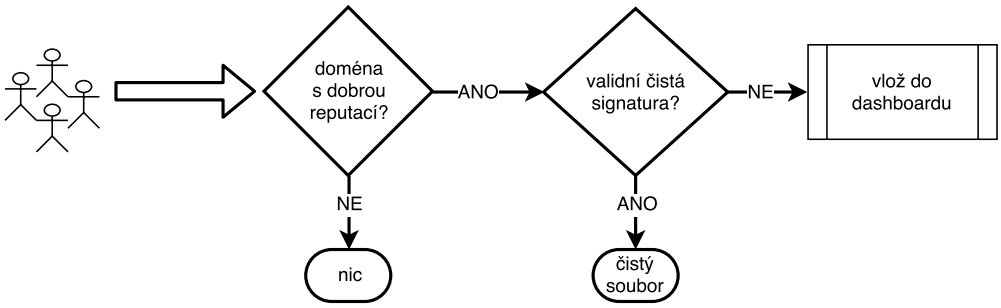
Obrázek 3.6. Schéma klasifikátoru domén

Na obrázku 3.6 je zobrazeno obecné schéma klasifikátoru domén. Nejprve se vyberou z databáze domény se *špatnou* reputací. Pro každou doménu se vyberou soubory z ní stažené a podle jejich detekcí se určí počet souborů klasifikovaných jako *malware*, resp. *adware*. Pokud většina souborů spadá do třídy *adware*, je takto oklasifikovaná i doména, pokud většina souborů patří do třídy *malware*, potom je doména přidaná na blacklist.

3.5 Indikace false positive detekcí

Druhým hlavním úkolem *Marlowe* je najít a nahlásit, popř. vyřešit situace, kdy je soubor stažený z domény s *dobrou* reputací detekován jako malware. Tato situace nastane v případě, kdy je vytvořena špatná detekce a detekuje soubor, který je neškodný, nebo v případě, kdy je doména infikována. Pokud jde o nesprávnou souborovou detekci, měla by být zrušena a soubor by měl být označen za čistý. Pokud je doména napadena a infikována škodlivými soubory, je potřeba na to upozornit.

Ani zde se nelze spoléhat pouze na doménovou reputaci, a proto všechny detekované soubory pocházející z domény s *dobrou* reputací považovat za neškodné a detekce je detekující za nesprávné. Například vůči již zmiňovanému zneužití zbytkové důvěry v doménu ([7]), která byla dříve používána k legálním účelům a nyní louží k šíření škodlivých souborů, by potom *Marlowe* nebyl odolný.



Obrázek 3.7. Schéma hledače anomálních detekcí

Na obrázku 3.7 je zobrazeno schéma pro hledání anomálních detekcí. Na počátku je stream detekcí od uživatelů. Pokud jde o detekci z webového štítu a soubor je stáhnutý z domény s *dobrou* reputací, dojde ke kontrole certifikátu souboru. Pokud má soubor validní[1] certifikát, který je označen jako důvěryhodný, je soubor označen jako čistý. Pokud jde o jinou signaturu, tj. pokud ji například vypršela doba platnosti nebo není důvěryhodná, nebo soubor podepsán není, je celý záznam o incidentu vložen do dashboardu, aby byl manuálně zkontrolován analytikem.

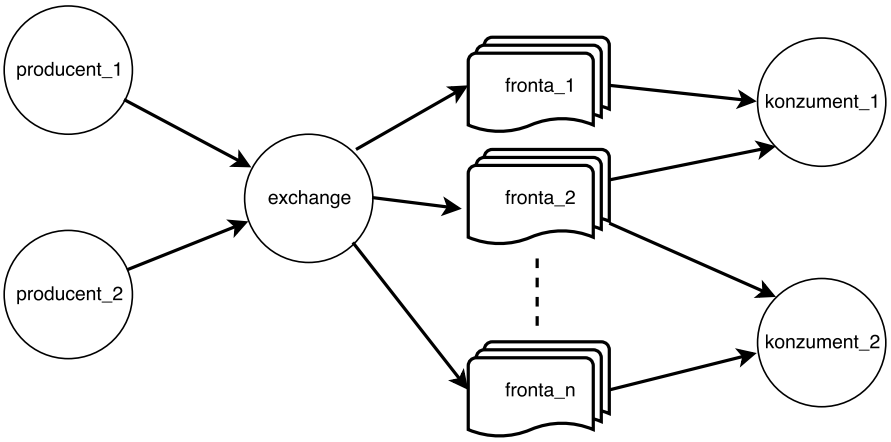
[1] Jako validní je považován certifikát, kterému nevypršela doba platnosti a nebyl zneplatněn.

Kapitola 4
Použité technologie

V této kapitole se krátce seznámíme se softwarovými technologiemi, které budou použity pro implementaci *Marlowe*.

4.1 RabbitMQ

RabbitMQ je open sourcový frontový systém sloužící k paralelizaci zpracování dat. RabbitMQ původně vznikl jako implementace AMQP[1] protokolu, který umožňuje distribuci zpráv mezi různými programovacími jazyky.



Obrázek 4.1. RabbitMQ

Na obrázku 4.1 je znázorněno schéma posílání zpráv do front a jejich následného konzumování. Producent posílá spolu s každou zprávou i tzv. směrovací klíč (*routing key*). Konzumenti při vytváření front nastaví, které klíče (tzv. přiřazovací – *binding keys*) je zajímají. Exchange po obdržení zprávy od producenta pošle zprávu do těch front, které mají přiřazovací klíč stejný jako je směrovací klíč zprávy. Může zprávu poslat do více front, ale také do žádné, pokud žádná z existujících front zprávu „nechce“.

Zprávy se z front posílají rovnoměrně mezi konzumenty, kteří jsou k frontě připojeni. Zprávy mohou být z fronty mazány hned po jejich odeslání jednomu z konzumentů, nebo se může čekat na potvrzení od konzumenta, pokud je potvrzení explicitně požadováno. V takovém případě se může zpráva odeslat znovu, i jinému konzumentovi, pokud potvrzení nedorazí.

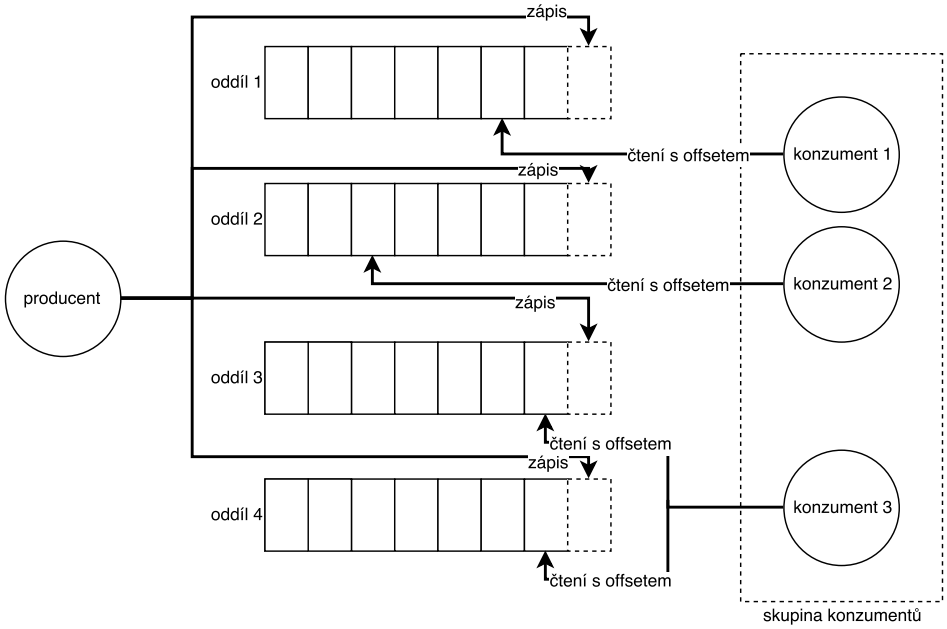
RabbitMQ je implementovaný v jazyce Erlang a nabízí API pro mnoho dalších programovacích jazyků jako je Java, Python, C# a další.

[1] AMQP (Advanced Message Queuing Protocol) je standard aplikační vrstvy pro komunikační prostředí pomocí zpráv zavádějící směrování, frontování, spolehlivost a bezpečnost posílání zpráv.

4.2 Apache Kafka

Kafka je open sourcový software pro zpracovávání streamů dat vyvíjený firmou Apache v jazyce Scala a Java. Jejím účelem je poskytnout unifikovanou platformu s vysokou datovou propustností a nízkou latencí pro práci s daty v reálném čase.

Zásadním rozdílem oproti RabbitMQ je způsob uchovávání zpráv. Kafka nepoužívá fronty. Zprávy příslušící do tzv. *topiců* jsou odesílány do jednotlivých oddílů, které jsou uchovávány na serverových klastrech (data jsou v nich replikována).



Obrázek 4.2. Apache Kafka

Na obrázku 4.2 je zobrazeno schéma komunikace mezi producentem a konzumenty prostřednictvím Kafky. Zpráva (skládající se z těla zprávy, klíče a časové značky) poslaná konzumentem do daného topicu je připojena na konec jednoho z oddílů, který je k topicu přiřazen (zprávy jsou rozdělovány mezi oddíly pomocí round-robin nebo pomocí hašovací funkce).

Po připojení konzumenta dojde k přebalancování oddílů tak, aby byly rovnoměrně rozděleny mezi konzumenty (jeden konzument může konzumovat více oddílů, ale jeden oddíl může být konzumován jenom jedním konzumentem). Konzument čte data z oddílu z pozice dané svým offsetem.

Doba, po kterou zůstávají zprávy uloženy v oddílech, je daná retencí, která je pro jednotlivé topicy nastavená. Pokud například dojde k chybě na jednom z konzumentů, stačí, když se změní jeho offset, a může zprávy zpracovat znovu. To je zásadní rozdíl oproti RabbitMQ, kde jsou zprávy z front mazány po jejich čtení a podruhé už je konzumovat nelze.

4.3 Python

Python je vyšší programovací jazyk, který v roce 1991 navrhl Guido van Rossum. Hlavní filozofií Pythonu je snadná čitelnost a pochopitelnost[1]. Python má interpretry

[1] Hlavní filosofie Pythonu je shrnuta v podobě 20 aforismů v tzv. *Zen of Python* Tima Peterese https://www.python.org/dev/peps/pep-0020/

pro mnohé operační systémy a podporuje několik programovacích paradigmat včetně objektově orientovaného, imperativního i procedurálního programování. Jde o open source software a díky široce rozvinuté komunitě uživatelů nabízí velké množství knihoven.

Python užívá pro oddělování bloků příkazů mezery místo častějších složených závorek a používání středníků pro ukončování příkazů je volitelné. V mnohých případech jsou místo znaků pro některé operace použity slova. Tím je podpořena čitelnost kódu a jeho celková podobnost anglickému jazyku.

Nevýhodou Pythonu je jeho nízká rychlost. Kvůli tomu, že jde o interpretovaný jazyk, nedosahuje takové rychlosti jako většina kompilovaných.

4.4 Scala

Scala je silně typovaný objektově orientovaný programovací jazyk. Zdrojový kód se kompiluje do byte kódu Javy, aby výsledný spustitelný soubor mohl být spuštěn na JVM[1]. Kód napsaný ve Scale je kompatibilní s kódem psaným v Javě, takže knihovny psané v obou jazycích mohou být vzájemně kombinovány.

4.4.1 Akka

Akka[2] je skupina open-source knihoven pro tvorbu robustních škálovatelných systémů s vysokou spolehlivostí a výkonem. Aby Akka knihovny umožnily uživateli vyvinout systém zvládající běžet v prostředí, kde mohou selhávat komponenty, ztrácet se zprávy a kolísat latence v síti bez nutnosti programovat na nízké úrovni mutexy atp., přináší několik paradigmat: použití více vláken bez nutnosti hlídání přístupu k paměti, transparentní vzdálená komunikace mezi systémy a jejich komponentami, elastická škálovatelná architektura umožňující vytváření vysoce responzivní systémy.

4.5 Jenkins

Jenkins[3] je open sourcový automatizační server, umožňující plánovat různé typy úloh spojené se sestavením, testováním a vydáním softwaru. Jenkins podporuje velké množství pluginů, díky nimž ho lze snadno integrovat se spoustou dalších nástrojů. Lze například po odeslání nové verze do gitu spustit automaticky testy, po jejich úspěšném doběhnutí začlenit novou větev do větve hlavní a poté celou novou verzi nasadit.

Při použití pro běžné spouštění skriptů lze nastavit adresa k repozitáři se zdrojovými kódy a kdy se mají spustit. Jenkins nabízí několik možností pro spuštění běhu. Od periodického přes dotaz na zvolenou URL až po kaskádovité spouštění několika akcí po sobě, které si mohou vzájemně předávat parametry.

Samozřejmostí je také grafické rozhraní, kde lze přehledně sledovat, v jakém stavu se všechny projekty nacházejí.

4.6 Puppet

Puppet je platforma pro správu konfigurace použité infrastruktury. Puppet je navrhnut tak, aby umožňoval nakonfigurovat prostředí nezávisle na architektuře, na které je

[1] JVM (Java Virtual Machine) je abstraktní stroj umožňující počítači spustit program psaný v Javě.
[2] Více informací v dokumentaci k poslední verzi Akka 2.5 na https://doc.akka.io/docs/akka/2.5/index.html.
[3] Detailnější informace o Jenkinsu na webových stránkách https://jenkins.io.

spuštěn. Toho je dosaženo vysokou mírou abstrakce, kdy je uživatel oproštěn od implementačních detailů a pouze zadává Puppetu instrukce jako „zařiď, že je přihlášen daný uživatel a že je spuštěná daná služba“. Vždy se pouze definuje stav, ve kterém se má systém na konci nacházet, ale nemusí se řešit kroky ani jejich pořadí, které jsou nutné, aby se do daného stavu systém dostal. Konfigurace probíhá ve vlastním deklarativním jazyce Puppet.

Díky tomuto nástroji lze poměrně snadno zařídit, že je aplikace spouštěna ve stejném prostředí pro testování i při produkčním chodu. Puppet dále také zajistí, že aplikace je opětovně spuštěna, pokud dojde k jejímu pádu.

Kapitola 5
Implementace

V této kapitole bude diskutována implementace jednotlivých částí *Marlowe*. Nejprve vytvořím aplikace pro sběr a posílání dat do *statické* a *dynamické* databáze. Poté bude následovat vývoj aplikací pro analýzu a hledání anomálií v databázi a v poslední části vytvořím dashboard pro zobrazování výsledků na webovém serveru.

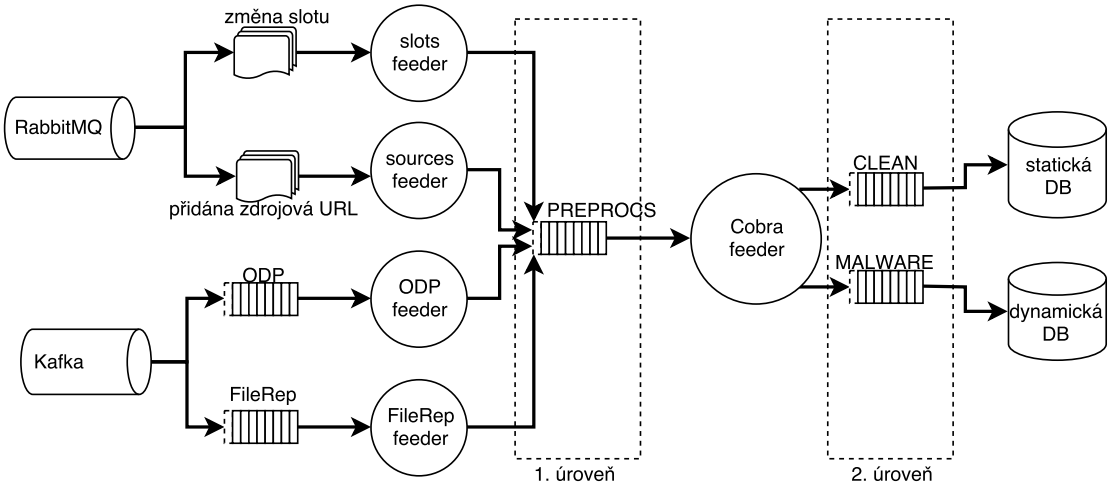
Části psané v jazyce Python budou koncipovány do formy instalačního balíčku a několika skriptů tak, aby pro spuštění stačilo naklonovat gitový repositář balíčku a spustit jeden ze skriptů. Aplikace psané v jazyce Scala naopak budou každá psaná jako samostatná aplikace, schopné být spuštěny samostatně na libovolném serveru podle nastavení konfiguračních souborů.

5.1 Zpracování vstupních dat

Tato sekce bude popisovat postup návrhu hierarchie aplikací a front, které budou sloužit k plnění reputační databáze daty a jejich následnou implementaci.

Jak již bylo zmíněno v 3.3, reputační databáze je plněna prostřednictvím dvou frontových systémů. Konkrétně půjde o dva Kafka topicy – *MALWARE* a *CLEAN*. V topicu *CLEAN* budou dva typy zpráv: pro přidávání grafu incidentu do *statické* databáze a pro odebrání grafu. Topic *MALWARE* bude obsahovat pouze zprávy pro přidávání grafů, protože jejich odebírání probíhá automaticky po vypadnutí z časového okna, tj. jeden týden po vložení.

Vkládání zpráv do těchto topiců bude prováděno několika aplikacemi, jelikož budou sbírána z více různých zdrojů prostřednictvím několika nástrojů jako Kafka a RabbitMQ. Pro větší distribuci a snazší správu aplikací bude výhodné rozdělit zpracování vkládaných dat do dvou úrovní (obrázek 5.1).



Obrázek 5.1. Hierarchie zpracování dynamických vstupních dat

Oproti použití pouze dvou Kafka topiců a posílání všech dat přímo do topiců, které jsou konzumovány reputační databází, má tento přístup výhodu ve své obecnosti. Do Kafka topicu v první úrovni budou posílány data o všech incidentech stažení PE souborů z daných URL. Tento topic pak může být použit i k jiným účelům. Mezi první a druhou úrovní bude aplikace *Cobra feeder* konzumující *PREPROC* topic. Data o incidentu budou dále obohacena o další informace jako jsou klasifikace domény a cesty URL. Tato data budou rozesílána do obou topiců druhé úrovně ve formátu definující jednotlivé grafy incidentů.

Na vstup Kafky v první úrovni jsou posílána data ze čtyř zdrojů – *ODP*, *FileRep* a další dva jsou ze změn v souborové databázi.

ODP (One Data Point) je Kafka topic, do kterého se posílají informace o detekcích od uživatelů. Aplikace *ODP feeder* konzumuje tento topic, filtruje zprávy týkající se PE souborů, které jsou klasifikovány jako škodlivé, a posílá je dále, tj. do *PREPROCS* Kafka topicu.

FileRep je reputační služba pro PE soubory. Kdykoli uživatel spustí nějaký PE soubor, který není klasifikovaný jako čistý, tj. pokud nejde o podepsaný soubor ověřenou signaturou, dochází k poslání haše a metadat souboru do *FileRepu*. Odpověď obsahuje informace z databáze *FileRepu* jako je prevalence souboru, několik klasifikačních atributů apod. pomáhající klientu rozhodnout, zda se jedná o škodlivý soubor. Právě tyto dotazy a odpovědi na ně jsou posílány do Kafky, odkud je konzumuje *FileRep feeder*, filtruje pouze ty zprávy obsahující dotaz na soubor označený jako škodlivý a posílá je dále.

Zprávy z databáze souborů jsou uchovávány v RabbitMQ. Pro nás jsou zajímavé zprávy se směrovacími klíči *AutomaticFileClassificationAdded* a *AutomaticFileClassificationRemoved*, jež obsahují data o souborech, které byly přidány resp. odebrány z některého ze slotů. Skript *slots feeder* konzumuje frontu s těmito klíči, vybírá takové zprávy, kde byl soubor přidán do slotu čistých resp. škodlivých souborů a nebo kde je soubor odebrán ze slotu čistých souborů. Zprávy se soubory přidanými do slotů jsou poslány do *PREPROCS* topicu, zatímco zprávy se soubory odebranými z čistého slotu jsou poslány přímo do *CLEAN* topicu, protože tento incident už není potřeba obohacovat o žádné další informace. Stačí znát haš souboru, který má být odebrán, aby byly následně odebrány ze *statické* databáze všechny grafy obsahující tento soubor.

Skript *sources feeder* konzumuje z RabbitMQ zprávy se směrovacím klíčem *FileSourceAdded*. Tyto zprávy obsahují soubory, ke kterým byl přidán nový zdroj. Jsou vybrány jenom zprávy týkající se souborů ze slotu s čistými resp. škodlivými soubory, pokud jde o nový zdroj z webu s validní URL adresou. Tyto incidenty jsou poté poslány do *PREPROCS* topicu k dalšímu zpracování.

5.1.1 Formát zpráv pro Kafku

Pro serializaci zpráv posílaných do Kafky jsem zvolil formát JSON. Jeho výhodou je snadná editace v případě, že by v budoucnu bylo potřeba do zpráv přidat nějaké další parametry.

PREPROCS Kafka topic obsahuje pouze informace o stažení PE souboru z dané URL adresy. Zprávy tedy budou obsahovat pouze čtyři parametry: URL, haš souboru, logickou proměnnou zda se jedná o škodlivý soubor a název zdroje, odkud incident pochází. Například pro incident zobrazený na obrázku 3.5 bude zpráva vypadat následovně:

```
{
    "source": "odp_malware",
    "url": "http://webmail.uolhost.com.br/attachment",
```

```
    "sha": "C688...61F2", /* zkráceno pro lepší čitelnost */
    "malware": true
}
```

CLEAN Kafka topic může obsahovat dva typy zpráv: pro přidávání grafů a pro jejich mazání. Zprávy pro odebírání grafů obsahují pouze haš souboru, který má být odstraněn, a parametr říkající o jaký typ zprávy jde. Pro odstranění grafů obsahující soubor s haší 200C...E9DA by byla použita následující zpráva:

```
{
    "type": "remove",
    "sha256": "200C...E9DA" /* zkráceno pro lepší čitelnost */
}
```

Zprávy pro přidávání grafů kromě specifikace typu zprávy a zdroje incidentu obsahují definici celého grafu pomocí seznamů uzlů a vazeb, které je spojují. Pro příklad takové zprávy opět použiji incident zobrazený na obrázku 3.5:

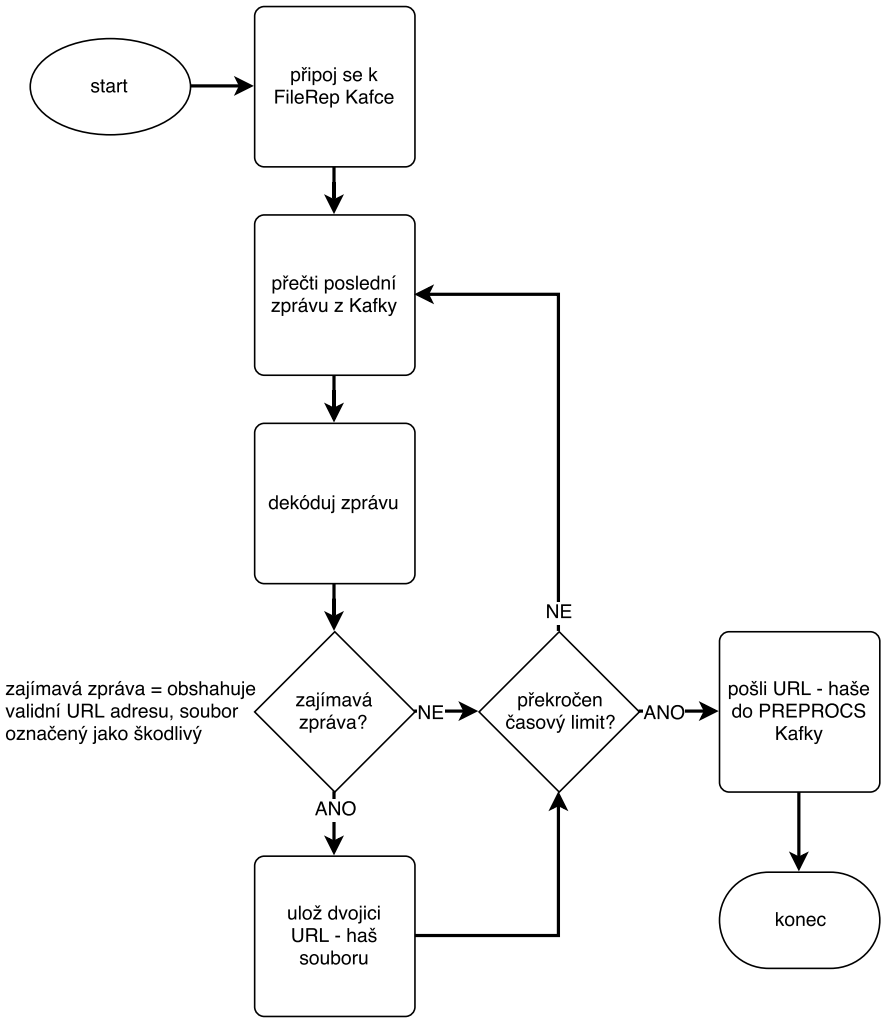
```
{
  "nodes": [
    {
      "type": "domain",
      "name": "uolhost.com.br",
      "malicious": false
    },
    {
      "type": "subdomain",
      "name": "webmail.uolhost.com.br",
      "malicious": false
    },
    {
      "type": "path",
      "name": "/attachment",
      "malicious": false
    },
    {
      "type": "file",
      "name": "C688...61F2", /* zkráceno pro lepší čitelnost */
      "malicious": true
    }
  ],
  "edges": [
    {
      "parent": 0,
      "child": 1,
      "type": "subdomain"
    },
    {
      "parent": 1,
      "child": 2,
      "type": "path"
    },
    {
      "parent": 2,
```

```
        "child": 3,
        "type": "download"
      },
      {
        "parent": 0,
        "child": 3,
        "type": "download"
      }
    ],
    "source": "odp_malware"
    "type": "add"
}
```

Topic *MALWARE* obsahuje pouze zprávy přidávací, protože *dynamická* část reputační databáze nepodporuje jiné mazání grafů než po vypadnutí z časového okna.

5.1.2 FileRep

Jak již bylo zmíněno výše, skript *FileRep feeder* bude konzumovat Kafka topic obsahující zprávy s dotazem na soubor a jeho následnou odpovědí od serveru.



Obrázek 5.2. Schéma skriptu pro konzumování FileRepu

Skript je implementován v jazyce Python. Algoritmus (zobrazen na obrázku 5.2) je založen na cyklickém spouštění skriptu pomocí Jenkinsu. Dochází ke konzumování zpráv

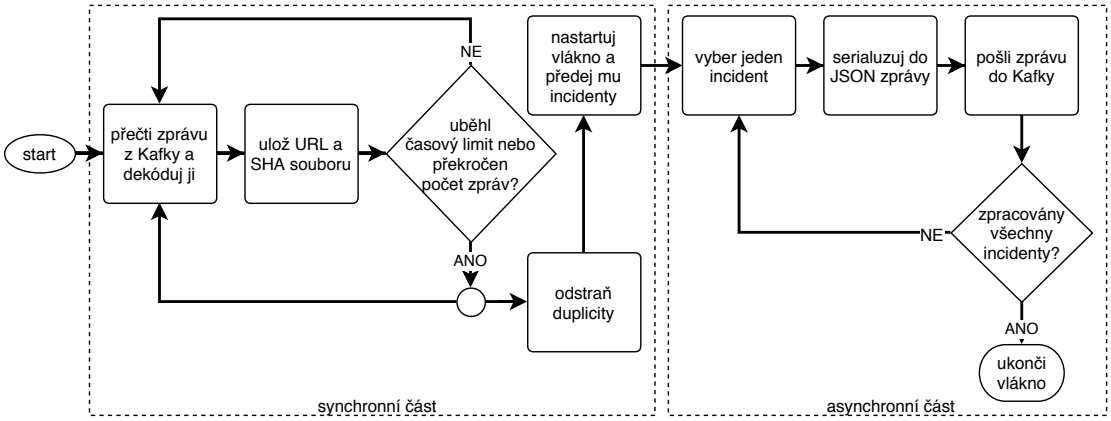
z Kafky po určitou časovou dobu (55 s). Zprávy je nutné nejprve dekódovat, protože jsou do Kafky posílány ve formátu tzv. *protobufu*[1]. Pokud zpráva obsahuje potřebná data, tj. validní URL adresu, ze které byl stažen soubor označený jako škodlivý, potom je uložena dvojice URL – haš souboru. Po uplynutí časového limitu se z dvojic souborové haše a její zdrojové URL vytvoří zprávy a ty jsou poslány do *PREPROCS* Kafky. Pokud je skript cyklicky spouštěn s periodou o několik vteřin delší než je doba pro konzumování (v našem případě tedy 60 s), stihnou se zpracovat zkonzumované zprávy a poslat do Kafky ve druhé úrovni. Zároveň lze díky cyklickému spouštění získat určitou odolnost vůči pádům skriptu vlivem chvilkové nedostupnosti clusterů Kafky atp. Skript se po periodě spustí znovu a konzumuje, kde přestal.

5.1.3 ODP

Pro implementaci aplikace konzumující ODP[2] použiji jazyk Scala.

Do ODP přichází až 700 zpráv za sekundu (během dne počet zpráv kolísá vlivem nerovnoměrného rozložení uživatelů po Zemi a jejich různé aktivitě v různých denních dobách). Proto je potřeba jejich zpracování paralelizovat, což značně usnadní knihovna Akka.

Celý algoritmus aplikace je zobrazen na obrázku 5.3. Paralelizace zpracování zpráv je zajištěna pomocí více vláken. Jedno vlákno konzumuje zprávy z Kafky a ukládá si dvojice URL – haš souboru, pokud jde o škodlivý PE soubor. Toto shromažďování dat pokračuje do chvíle, kdy uběhne časový limit (1 minuta) nebo se nashromáždí daný počet dat (10000 validních zpráv). K omezení jsou použita obě kritéria, aby se zajistilo dostatečné rozdělení práce mezi více vláken při větším objemu dat a zároveň aby nedocházelo ke zbytečnému zpoždění při menším objemu dat.



Obrázek 5.3. Schéma aplikace pro konzumování ODP

Po dosažení jednoho z kritérií dojde k odstranění duplicit a spuštění dalšího vlákna, které se postará o serializaci dat do JSON formátu a poslání zpráv do *PREPROCS* Kafky. Původní vlákno mezitím dále konzumuje nové zprávy.

[1] Protobuf (Protocol Buffers) je forma serializace strukturovaných dat vyvinutá firmou Google. Jde o platformově i jazykově nezávislý mechanizmus napsaný v jazyce C++. Pro použití v jiných jazycích slouží binární soubor, který pomocí souboru obsahujícího definici objektu obsaženého ve zprávě vytvoří skript v daném programovacím jazyce. Pomocí tohoto skriptu poté lze dekompilovat binární zprávu do objektu čitelného v daném jazyce a dále s ním pracovat.

[2] ODP – *One Data Point* je Kafka topic obsahující informace o detekcích, které se posílají z koncových zařízení, kde je nainstalovaný Avast.

5.1.4 Změny v souborové databázi

Pro udržení reputační databáze synchronizované se souborovou databází je potřeba posílat informace o změnách. O to se starají dva skripty. Jejich algoritmus by se opět dal popsat schématem z obrázku 5.2. Na rozdíl od Kafky se ale zprávy konzumují z RabbitMQ. Jde o dvě fronty. Do první se posílají informace o přidání zdroje k souborům a do druhé o přidání resp. odebrání souborů z nějakého slotu (soubory jsou rozděleny do slotů podle jejich klasifikace).

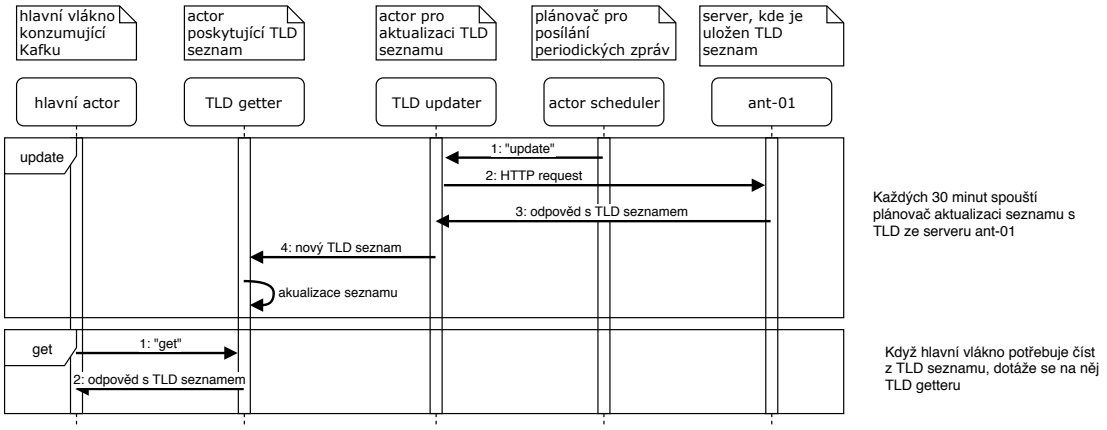
Skript konzumující frontu obsahující přidání zdrojů k souborům tak činí po dobu 20 sekund. Poté vybere pouze zprávy obsahující zdrojové URL adresy souborů ze slotu čistých nebo škodlivých souborů. Poté jsou zprávy odeslány do *PREPROCS* Kafky.

Skript konzumující frontu se soubory, které změnily svůj slot, opět pracuje periodicky. Zprávy konzumuje po dobu 10 sekund. Pro soubory, které byly přidány do slotu pro škodlivé soubory, jsou zjištěny všechny jejich URL zdroje a poté jsou opět poslány zprávy do *PREPROCS* Kafky. Soubory přidané do čistého slotu jsou zpracovány stejným způsobem. Důležité jsou ale i zprávy obsahující soubory, které byly odebrány z čistého slotu. V takové situaci se pošle zpráva obsahují haš souboru, který má být odebrán z reputační databáze, do *CLEAN* Kafky.

5.1.5 Cobra feeder

Cobra feeder je aplikace běžící mezi Kafka topicy první a druhé úrovně. Jde o Scala aplikaci konzumující *PREPROCS* topic a posílající zprávy do *CLEAN* a *MALWARE* topiců. Zatímco *PREPROCS* topic obsahuje pouze informace o jednotlivých incidentech tj. haš souboru, URL a název zdroje odkud incident přišel, topicy *CLEAN* a *MALWARE* už obsahují zprávy s celou definicí grafu (obrázek 3.4). Je tedy nutné zjistit, zda jsou jednotlivé části URL (doména 2. řádu, celá doména, cesta) blokované webovým štítem Avastu.

Na vstupu aplikace je tedy *PREPROCS* topic. Po rozbalení zprávy je potřeba rozdělit URL na její části. Problematické může být nalezení domény druhého řádu. Přestože většina domén prvního řádu je složena z jednoho slova, existuje stále více domén prvního řádu, které jsou složeny z více částí oddělených tečkou (např. *com.br*). Proto je nutné udržovat seznam domén prvního řádu a za jeho pomoci poté hledat domény druhého řádu. Tento seznam uchováváme ve formě textového souboru. Abychom mohli měnit tento soubor za běhu aplikace, potřebujeme ho dynamicky načítat za běhu. K tomuto účelu využijeme Akka aktory, které si mezi sebou budou posílat zprávy.



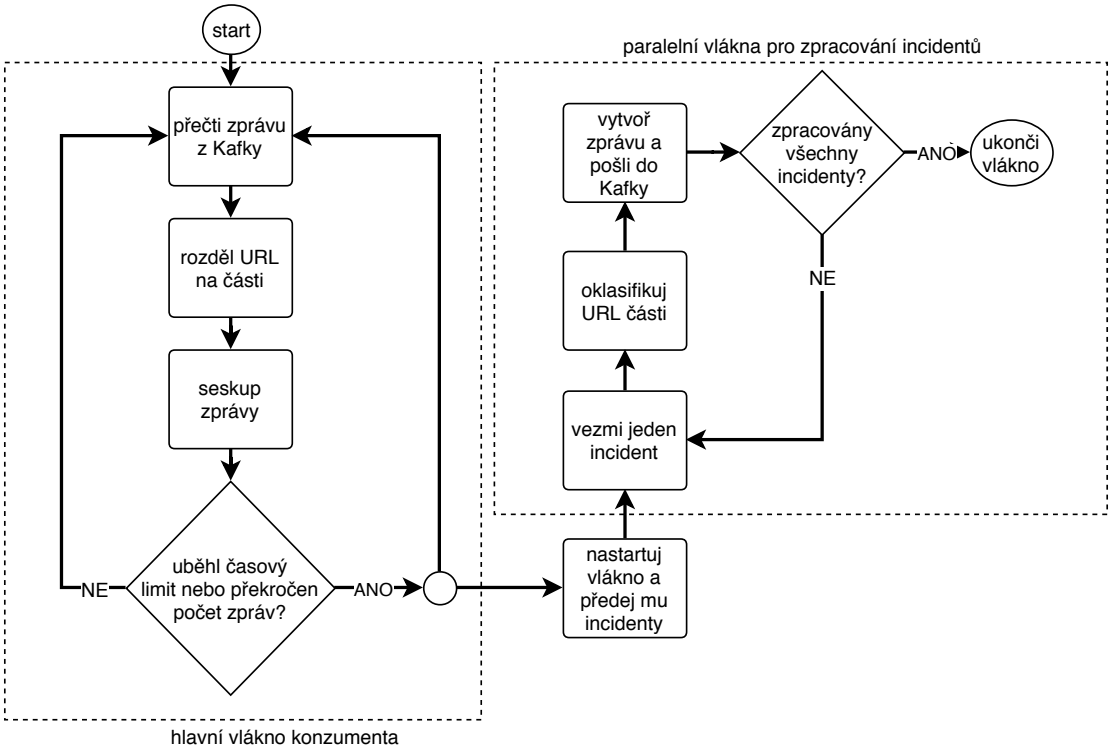
Obrázek 5.4. Diagram pro používání a aktualizaci TLD seznamu

Na obrázku 5.4 je zobrazen diagram komunikace aktorů při aktualizaci seznamu s doménami prvního řádu. Jsou použity dva aktory *TLD getter* a *TLD updater*. Po uplynutí periody (30 minut) pošle plánovač aktorů zprávu „update“ *updateru* a ten pomocí HTTP požadavku získá přes API aktuální seznam ze serveru. Tento seznam poté pošle aktoru *getter*, který si seznam lokálně uloží.

Pokud hlavní vlákno potřebuje seznam s domény prvního řádu, vyšle zprávu „get“ směrem k *getteru* a ten mu zpět pošle seznam. Tímto mechanizmem docílím možnosti aktualizovat seznam za běhu aplikace bez jakéhokoli čekání (kromě prvního načtení seznamu při startu aplikace).

Po rozdělení URL na jednotlivé části je ještě potřeba zjistit, zda jsou škodlivé z pohledu webového štítu. K tomu je využita webová služba, která pro URL předané pomocí HTTP post metody vrací, zda jsou zablokované. Poté se jednotlivé incidenty serializují do zpráv ve formátu JSON (5.1.1), které obsahují definice grafů. Tyto zprávy jsou podle klasifikace souboru poslány do Kafka topiců *CLEAN* nebo *MALWARE*.

Nejslabším místem této aplikace z hlediska rychlosti jsou právě HTTP dotazy na klasifikace částí URL. Proto využiji možností Akka a před těmito dotazy zprávy seskupím a pro jejich zpracování a následné posílání zpráv do Kafky využiji asynchronně spouštěná vlákna. Schéma algoritmu aplikace je zobrazeno na obrázku 5.5. Hlavní vlákno pro konzumování cyklicky konzumuje zprávy po danou dobu (60 sekund) nebo dokud nenashromáždí dostatečný počet zpráv (10000). Poté dochází k odstranění duplicit a předání zpráv jednomu z vláken z Thread Poolu, které po dokončení posílání zpráv opět zanikne.



Obrázek 5.5. Schéma aplikace *Cobra feeder*

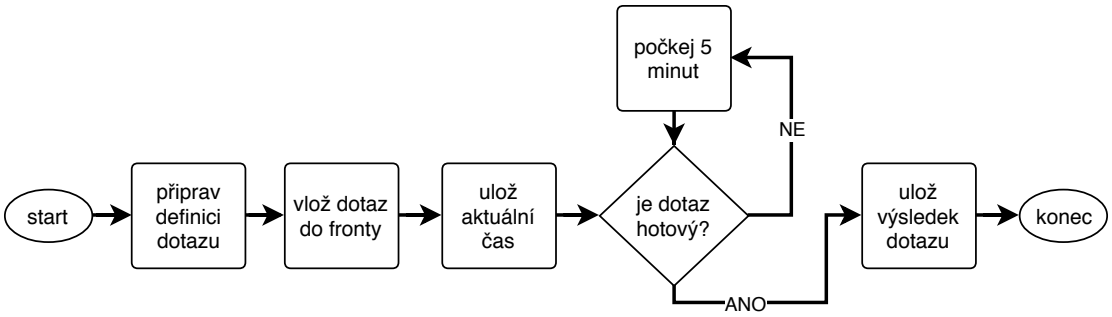
5.1.6 Obraz souborové databáze pro *statickou* databázi

Cobra má celou svou databázi uloženou v RAM paměti. Pokud dojde k jejímu ať už plánovanému nebo neplánovanému restartu, musí se celá databáze opět odněkud nahrát. *Dynamická* databáze obsahuje data pouze z posledního týdne, tudíž je možné

uchovávat všechna data v Kafce. Pokud je její retence nastavená na stejnou dobu jako je velikost časového okna, po které mají být grafy v databázi udržovány (tj. jeden týden), pak stačí zkonzumovat celý *MALWARE* topic a celá *dynamická* databáze může být opět v provozu.

Na druhou stranu *statická* databáze obsahuje veškeré čisté soubory a jejich zdrojové URL a *CLEAN* Kafka topic obsahuje pouze změny, které se za poslední týden staly v databázi čistých souborů. Retence *CLEAN* Kafky je též nastavena na 7 dní, proto je nutné alespoň jednou za týden vytvořit obraz celé databáze čistých souborů. Pokud dojde k restartu *Cobry*, načte se tento obraz, zkonzumuje se *CLEAN* topic od doby vytvoření obrazu do současnosti a poté se podle toho patřičně upraví i *statická* databáze.

O vytvoření obrazu souborové databáze se starají dva skripty. První z nich vyexportuje potřebná data ze souborové databáze a druhý je poté zpracuje a vytvoří soubor s definičními daty pro grafy reprezentující jednotlivé soubory a jejich zdrojové URL.

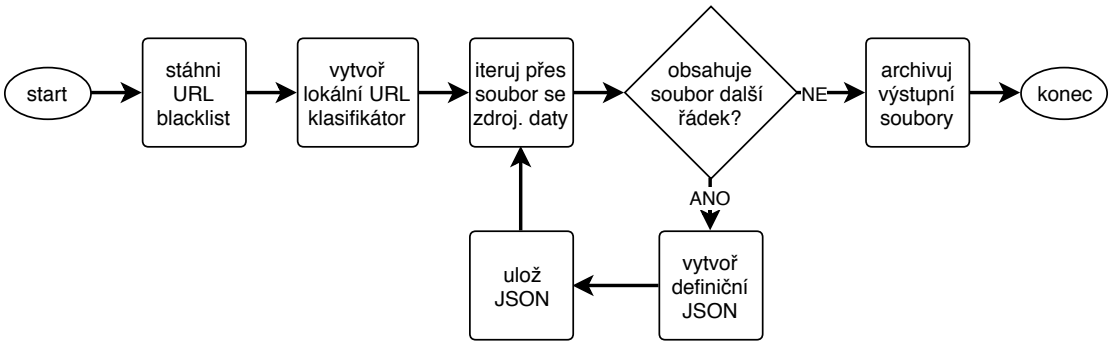


Obrázek 5.6. Schéma skriptu pro stažení dat k vytvoření obrazu souborové databáze

Schéma prvního skriptu je zobrazeno na obrázku 5.6. Pro přímé získávání většího množství dat ze souborové databáze slouží tzv. *analytické dotazy*, umožňující přesnou specifikaci dat, které se mají vybrat, a poté běží na jednom z pracovních serverů a výsledek uloží do textového souboru. Pro volání tohoto dotazu je potřeba vytvořit šablonu, která umožní vybrat všechny soubory ze slotu čistých PE souborů spolu se svými URL zdroji. Pomocí této šablony bude vložen dotaz do fronty pro zpracování. Poté dochází k cyklickému dotazování, zda je dotaz hotový. Spolu s kladnou odpovědí dostanu i cestu k souboru, kde je soubor uložen. Tento soubor si skript uloží do svého pracovního adresáře, aby mohl být pomocí Jenkinsu předán jako artefakt dalšímu skriptu, který s ním bude pracovat. Dále je nutné uložit si přesný čas, kdy se vytvářel obraz celé databáze, aby bylo možné při spouštění *Cobry* od tohoto okamžiku konzumovat Kafku obsahující změny od tohoto okamžiku.

Druhý skript má za úkol projít výsledek dotazu a vytvořit pro jednotlivé soubory data pro vložení do *Cobry*. Podobně jako v aplikaci *Cobra feeder* i zde bude nutné zjistit, zda jsou jednotlivé části URL blokovány webovým štítem. Zde by bylo časově velice náročné ptát se na každou URL zvlášť stejné HTTP služby pro klasifikaci webových adres jako v případě *Cobra feeder*, protože se zde pracuje s řádově desítkami milionů jednotlivých incidentů. Proto je v tomto případě využita možnost stáhnout si lokálně databázi URL detekcí ve formě textového souboru. Na základě tohoto souboru si vytvoříme objekt umožňující lokální klasifikace pro části URL (celou doménu, doménu druhého řádu i cestu). Poté je možné procházet soubor s výsledky dotazu a pro každý incident vytvořit definiční JSON grafu (ve stejném tvaru jako zprávy posílané do *CLEAN* a *MALWARE* Kafky, viz 5.1.1). Tento výsledný soubor obsahující zdrojová data pro vytvoření grafů reprezentující čisté soubory pro *statickou* databázi je spolu se souborem obsahující časovou značku vytvoření obrazu archivován jako artefakt Jenkinsem. Při spuštění Cobry

poté stačí z dané URL adresy stáhnout oba soubory, z jednoho nahrát grafy a z druhého určit počáteční datum, od kdy se má začít konzumovat Kafka s případnými změnami od vytvoření obrazu. Schéma skriptu je zobrazeno na následujícím obrázku 5.7.



Obrázek 5.7. Schéma skriptu pro vytvoření zdrojových dat k načtení statické databáze

5.2 Klasifikace domén se *špatnou* reputací

Tato sekce se bude zabývat implementací skriptu, který z reputační databáze vybere domény se *špatnou* reputací a následně podle detekcí detekujících soubory z nich stažených rozhodne, zda se jedná o doménu škodlivou, adwarovou či nelze udělat žádné rozhodnutí.

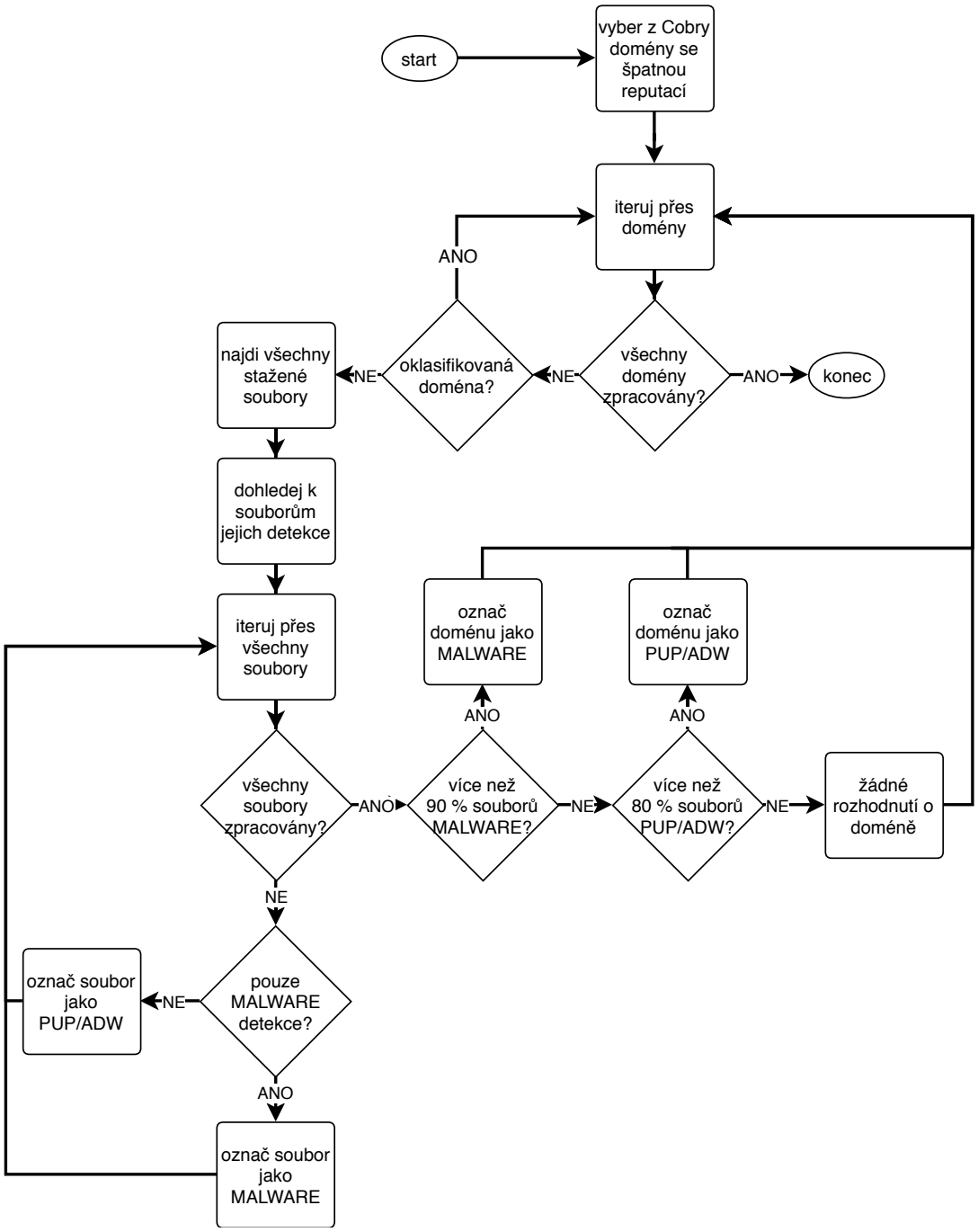
Schéma algoritmu skriptu je zobrazeno na obrázku 5.8. Nejprve jsou z reputační databáze vybrány domény se *špatnou* reputací. Každá doména je poté zpracována zvlášť.

Nejprve je potřeba ověřit, zda už není doména nějak oklasifikována. Pokud ano, může být přeskočena, pokud není, vyberou se opět z reputační databáze haše souborů, které byly z této domény stahovány. Poté je potřeba zjistit, zda jsou tyto soubory opravdu škodlivé (zda se například do databáze nedostaly jenom kvůli nějaké nesprávné detekci, která už je zrušená apod.). Za tím účelem jsou ze souborové databáze vybrány všechny detekce (Avastu i několika dalších antivirů), které daný soubor detekují. Díky tomu, že většina detekcí ve svém názvu obsahuje klíčová slova rozlišující jejich povahu[1], porovnáním jejich názvu s regulárním výrazem, který se shoduje právě s klíčovými slovy adware nebo PUP souborů, lze zjistit, zda se jedná o malware či nikoliv.

Abych se vyhnul nebezpečí false positive detekcí, bude považován soubor za *adware/PUP* i pokud jenom jediná z jeho detekcí bude splňovat regulární výraz. Po označení všech souborů stažených z dané domény lze rozhodnout o doméně samotné. Pokud je více než 90 %[2] všech stažených souborů označených jako malware, potom bude označena i celá doména jako škodlivá. Takové domény se také zablokují, aby je webový štít Avastu blokoval. Domény, ze kterých se stahuje více než 80 % souborů označených jako *adware/PUP*, budou oklasifikovány jako adware, aby nedošlo v budoucnu k jejich zablokování a aby je skript po dalším spuštění nezpracovával znovu.

[1] Například detekce směřující na adware obsahují v názvu často *ADW*, detekce potenciálně nebezpečných aplikací zase *PUP/PUA* (potentially unwanted program/application) atd.

[2] Zpočátku je použita tato velice konzervativní hranice. Teprve po řádném otestování a analýze výsledků bude možné číslo patřičně snížit, pokud bude potřeba

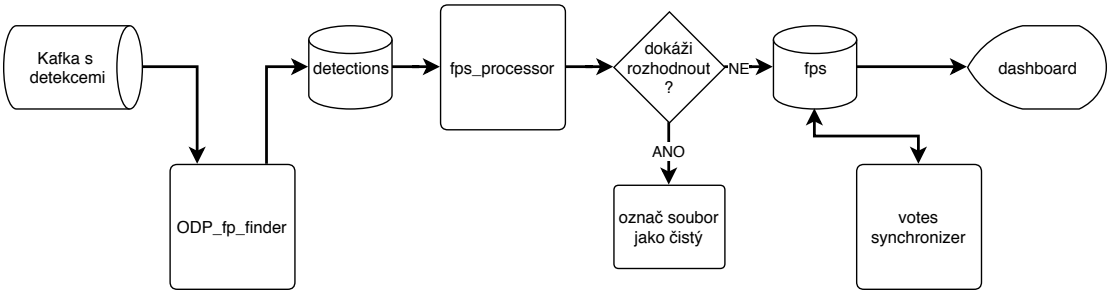


Obrázek 5.8. Schéma skriptu pro klasifikaci domén

5.3 Indikace false positive detekcí

Tato sekce bude popisovat implementaci systému pro hledání anomálních detekcí. Za anomální detekci je považována taková, která detekuje PE soubor stažený z domény, která podle reputační databáze slouží k šíření legitimního obsahu. Přehled tohoto systému je zobrazen na obrázku 5.9. Na počátku je Kafka topic obsahující detekce od uživatelů. Ten je konzumován aplikací (*ODP_fp_finder*), která hledá detekce týkající se souborů stažených z domén s *dobrou* reputací. Tyto incidenty jsou posílány do tabulky *detections* v MySQL databázi. Z této tabulky si bere data skript *fps_processor* a pomocí

dat ze souborové databáze se pokusí rozhodnout, zda se opravdu jedná o škodlivé soubory nebo ne. Když dojde k závěru, že je soubor čistý, označí jej tak, aby se na něm již nevytvářely detekce. Pokud nebude umět rozhodnout, pošle data o incidentech do tabulky *fps*, odkud budou zobrazena v dashboardu na HTTP serveru. Tam bude možné o souborech manuálně rozhodnout. Dále tabulku *fps* aktualizuje skript *vote_synchronizer*, který do ní propaguje změny ze souborové databáze týkající se manuálních klasifikací a zároveň maže řádky obsahující incidenty nezměněné za poslední týden.



Obrázek 5.9. Struktura systému pro hledání anomálních detekcí

5.3.1 Filtrace detekcí

Pro první filtraci detekcí vytvořím aplikaci psanou v jazyce Scala, která bude konzumovat stream detekcí od uživatelů Avastu. Pro každý incident týkající se PE souboru staženého z internetu se musí zjistit reputace domény, ze které byl soubor stažen. Pokud bude reputace domény považována za *špatnou*, poté bude celý incident poslán do MySQL databáze pro další zpracování.

Vstupem bude stejný Kafka topic jako v případě ODP feederu (5.1.3). Tím rostou nároky na rychlost zpracování těchto dat. Protože stejný zdroj dat je použit jak k plnění reputační databáze, tak i k testování vůči ní, je nutné, aby testování proběhlo ještě před vkládáním. Například pokud dojde ke stahování většího množství detekovaných souborů z domény s *dobrou* reputací (tato situace by mohla způsobit jedna špatná detekce, která detekuje instalátor specifický pro každého uživatele), mohla by dojít ke změně reputace z *dobré* na *špatnou*. Poté by se samozřejmě stažení škodlivého souboru z této domény nejevilo jako anomalita. Proto je cílem zajistit co možná nejrychleji dotaz do Cobry hned po přijetí incidentu. Tím se docílí nalezení anomálie, oklasifikování souboru a následného zrušení špatné detekce.

Zpracování incidentů probíhá v několika fázích. Během té první se dekóduje protobuf a dojde k rozhodnutí, zda je daná zpráva relevantní. To znamená, že jde o detekci na PE souboru, který byl označen jako škodlivý, a zpráva také obsahuje validní URL, ze které soubor pochází. Dále zde dochází k rozložení URL na její části. K oddělení domény druhého řádu je potřeba seznam domén prvního řádu a stejně tak jako v 5.1.5 i zde je použit pro jeho dynamické načítání za běhu aplikace algoritmus zobrazený na obrázku 5.4.

Ve druhé fázi dochází k seskupení incidentů, aby mohly být poté paralelizovaně zpracovávány. Při seskupování incidentů je nutné mít na mysli více kritérií. Pro zajištění co nejnižší latence ve zpracovávání jednotlivých událostí dává smysl seskupování do menších skupin, na druhou stranu ale v této fázi budou volány dotazy do *Cobry* pro zjištění reputace domén, ze kterých byly soubory staženy. Doba vykonávání dotazu v *Cobře* závisí na počtu souborů, které jsou s danou doménou spojeny. Typicky domény

s vyšším počtem souborů s nimi spojenými jsou doménami nejvíce prevalentními[1]. To znamená, že právě pro ty incidenty, které jsou zpracovávány nejčastěji, trvají dotazy nejdelší dobu. Z toho důvodu je vhodné si výsledky dotazů pro jednotlivé domény druhého stupně ukládat pro každé paralelní zpracování. Tím ale roste výhoda seskupování incidentů do větších skupin, čímž se docílí většího počtu hitů v cache výsledků.

Ve třetí a poslední fázi tedy dochází k dotazování na počty škodlivých n_{mal} a čistých souborů n_{cln} stažených z daných domén druhého stupně. Pokud se jedná o doménu s *dobrou* reputací (dochází tedy k šíření škodlivého souboru z domény, která běžně neslouží k tomuto účelu), jedná se o anomálii, a proto se tento incident vloží do MySQL databáze, aby mohlo dojít k podrobnějšímu prošetření. Doména s *dobrou* reputací je definována jako doména, pro kterou platí

$$n_{cln} > 1000 \quad \&\& \quad n_{cln} > 100 \cdot n_{mal}.$$

V takovém případě je zaručeno, že bylo z dané domény zaregistrováno velké množství stažených PE souborů a zároveň řádově menší počet souborů škodlivých.

5.3.2 Zpracování potenciálních false positive

Tabulka *detections* obsahuje informace o anomální incidentech. Jde o SHA256 souboru, URL, počet stažených čistých a škodlivých souborů z dané domény druhého stupně a časová známka vložení. Tato tabulka obsahuje události, kdy došlo k detekování škodlivého souboru staženého z domény, která v minulosti sloužila k šíření především čistých souborů. Jenom na základě toho nelze rozhodnout o tom, že se jedná o nesprávnou detekci. Může jít o napadnutí domény nebo o změnu vlastníka domény a její následné zneužití. Také se může jednat o domény, které slouží jako online úložiště, kde se vlastník často snaží škodlivé soubory po nahlášení odstranit. Ve všech těchto případech a mnohých dalších může nastat situace, kdy se z nich škodlivé soubory stahují. Proto je nutné soubory nejdříve analyzovat podle již nasbíraných dat a až poté o nich rozhodnout.

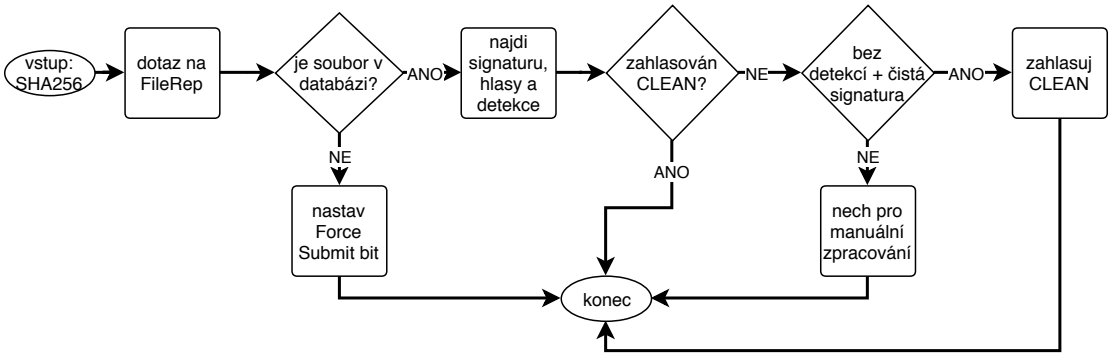
To má na starost skript *fps_processor* napsaný v Pythonu. Na jeho vstupu je tabulka *detections*. Konkrétně se vždy zpracovávají jenom záznamy, které v tabulce přibyly od posledního běhu skriptu. Toho je docíleno ukládáním časové značky posledního zpracovaného řádku. Když je poté skript puštěn znovu, načte tuto značku a vybere z tabulky jenom hodnoty novější.

Jednotlivé incidenty jsou zpracovávány postupně. Průběh zpracování a rozhodnutí o incidentu je zobrazen na obrázku 5.10. Nejprve se dotáži na soubor *FileRepu*, čímž zjistím prevalenci souboru[2] a zda je daný soubor v souborové databázi. Pokud soubor v databázi ještě není, nelze o něm rozhodnout. Proto se souboru pouze nastaví *ForceSubmit bit*, díky němuž se soubor pošle do souborové databáze, jakmile bude příště spatřen. Pokud soubor v databázi již je, vytáhnou se z ní k souboru další užitečné informace jako je jeho signatura, manuální klasifikace a detekce, které se na něm spouštějí. Pokud se zjistí, že soubor už někdo manuálně oklasifikoval jako čistý, už není třeba nic dále řešit, protože tím už se na něm žádná detekce spouštět nebude a false positive je tím vyřešen. Pokud se zjistí, že je soubor podepsán validní signaturou, která je oklasifikovaná jako čistá, a zároveň se na souboru nespouští žádné další detekce, potom se soubor oklasifikuje jako čistý a tím je také vyřešen. Pokud ovšem žádná z podmínek

[1] Například doména *google.com* je jednou z nejvíce prevalentních domén a ve *statické* databázi je velké množství čistých PE souborů z ní stažených. Zároveň ale také Google nabízí hostingové služby na webových adresách jako *sites.google.com/site/cesta/k/malware*, které útočníci mohou použít k šíření škodlivého softwaru.

[2] Prevalence souboru udává počet uživatelů Avastu, kteří soubor spustili.

výše není splněna, jsou všechny informace o incidentu včetně těch získaných ze souborové databáze (detekce, prevalence souboru, klasifikace signatury atd.) uloženy do *fps* tabulky, aby mohly být zobrazeny v dashboardu.



Obrázek 5.10. Schéma zpracování podezřelé detekce na souboru

5.3.3 Aktualizace *fps* tabulky

Jednou z nejdůležitějších hodnot v tabulce je pro každou řádku položka s manuální klasifikací souboru. Právě tato hodnota se většinou velmi rychle změní u nového souboru, který má vysokou prevalenci, protože u analytiků má samozřejmě vysokou prioritu. Pokud například dojde k manuálnímu označení souboru jako čistý, přestanou se na něm vytvářet detekce a už se znovu nedostane přes *ODP_fp_finder* do *fp_processoru*. Důsledkem toho by mohlo docházet tomu, že v tabulce *fps* zůstanou incidenty obsahující tento soubor, i když už je daný problém vyřešen.

Abych tomuto problému předešel, vytvořím skript, který bude změny v automatické klasifikaci souborů propagovat dále do *fps* tabulky. Na vstupu tohoto skriptu bude RabbitMQ fronta, do které jsou posílány všechny manuální klasifikace souborů. Z nich budou vybírány ty, co se týkají souborů z *fps* tabulky. Pokud se nějaký takový soubor s novou klasifikací najde, zpracuje se opět podle schématu na obrázku 5.10. Dále ještě má skript za úkol odmazávat z tabulky řádky staré jeden týden a více. Takovéto řádky obsahují málo prevalentní a staré incidenty, které není potřeba v tabulce držet.

5.4 Dashboard pro zobrazení anomálních detekcí

Aby byly výsledky z tabulky *fps* pro analytiky co nejsnáze čitelné, je potřeba vytvořit dashboard, který bude podezřelé incidenty zobrazovat a zároveň umožní zahlasovat na souborech a tím je patřičně oklasifikovat.

Na obrázku 5.11 je zobrazen dashboard obsahující deset nejvíce prevalentních škodlivých souborů stažených z domén s *dobrou* reputací. Pro každý soubor je zobrazeno několik informací, podle kterých lze v některých případech ihned rozhodnout o klasifikaci souboru. Vedle SHA256 souboru obsahující odkaz na detail souboru v grafickém zobrazení souborové databáze a domény, ze které byl soubor stažen, je v tabulce zobrazena i prevalence domény. Tato prevalence je počítána z databáze *Cobra*. To znamená, že zobrazuje počet počet unikátních čistých PE souborů stažených z domény za celou dobu její životnosti (*statická* databáze) a počet škodlivých souborů stažených z domény za poslední týden (*dynamická* databáze).

SHA	domain name	domain prevalence	file prevalence	signature category	other AV's detections	last vote	vote
BB8D...854A	M R hp.com	8.6k	9M	invalid	0 (0)		unknown malware PUP clean damaged adware
3A17...2AB2	M R asus.com	91k	6M		0 (0)		unknown malware PUP clean damaged adware
904A...4635	M R dsnetwb.com	2.8k	2.5M	invalid	0 (0)		unknown malware PUP clean damaged adware
1A4A...30FF	M R m6web.fr	7.8k	2.4M	invalid	9 (0)		unknown malware PUP clean damaged adware
1A4A...30FF	M R uptodown.com	80k	2.4M	invalid	9 (0)		unknown malware PUP clean damaged adware
3282...3BCA	M R apple.com	3k	2.4M	invalid	0 (0)		unknown malware PUP clean damaged adware
C4F8...5A2A	M R samsung.com	16k	1.7M		0 (0)		unknown malware PUP clean damaged adware
75A1...1874	M R asus.com	91k	1.6M		0 (0)		unknown malware PUP clean damaged adware
85F8...5A12	M R lavasoft.com	3.3k	1.6M	valid	0 (1)		unknown malware PUP clean damaged adware
46B7...AAB6	M R uptodown.com	80k	1.2M		0 (0)		unknown malware PUP clean damaged adware

Obrázek 5.11. Dashboard pro zobrazení podezřelých detekcí

V dalším sloupci je zobrazena prevalence souboru, což je počet uživatelů Avastu, kteří soubor spustili. Sloupec *signature category* zobrazuje klasifikaci signatury souboru a zda je signatura validní. *Other AV's detections* ukazuje kolik z vybraných jiných antivirů detekuje soubor jako škodlivý resp. jako PUP/adware (hodnota v závorkách). Ve sloupci *last vote* je zobrazena poslední manuální klasifikace souboru, pokud existuje. V posledním sloupci jsou tlačítka pro hlasování na souboru tj. pro přidání klasifikace. Barevná kolečka symbolizující manuální klasifikaci a signaturu nabývají stejných barev jako tlačítka pro hlasování. Červeně zobrazují kategorii *malware* a *adware*, žlutě značí *PUP*, zeleně *clean* a šedě *neznámé*.

Ve sloupci s doménovým jménem jsou dvě písmenka **M** a **R**, kde **R** je odkaz do grafického rozhraní databáze RevIPu (2.3) a **M** je odkaz na detail domény na základě analýzy pomocí *Marlowe*. Tato stránka obsahuje data z *Cobry* doplněná o další informace ze souborové databáze. Pro dotazovanou doménu je spočten počet škodlivých a čistých stažených souborů. Dále jsou ze souborové databáze vybrány všechny detekce, které detekují soubory z domény stažené. Na základě názvů detekcí je vypočteno *malware* resp. *adware/PUP* skóre podobně jako v 5.2. *Malware* skóre označuje procentuální část souborů, které jsou detekovány pouze malware detekcemi ze všech škodlivých souborů z domény stažených. *Adware/PUP* skóre na druhou stranu zobrazuje procentuální zastoupení souborů, které jsou detekovány i ne-malware detekcemi (takové soubory považujeme za *adware/PUP*).

Dále je zde vykreslen slovní mrak ze jmen rodin detekcí, který umožňuje na první pohled rozeznat, jaký typ škodlivých souborů se z domény stahuje. Čím více souborů daná rodina detekcí detekuje, tím větším písmem je její název zobrazen. Na obrázku 5.12 je zobrazena podoba stránky s detailem pro IP adresu *88.99.187.254*, která již byla podle kritérií automatického klasifikátoru domén (5.2) zablokována.

Obrázek 5.12. Grafické zobrazení *Marlowe* analýzy pro IP adresu *88.99.187.254*

Kapitola 6
Testování výsledného řešení

Tato kapitola se bude zabývat testováním výsledného řešení systému *Marlowe*. Nejprve budou popsány testy aplikací zpracovávajících vstupní data do *Cobry*. Zaměřím se na jejich datovou propustnost, tj. kolik zpráv dokáží zpracovat za jednotku času a za jak dlouho se dokáží vzpamatovat z výpadku. V další části bude následovat analýza výsledků automatického klasifikátoru domén a zpracování false positive detekcí od doby spuštění produkčního provozu.

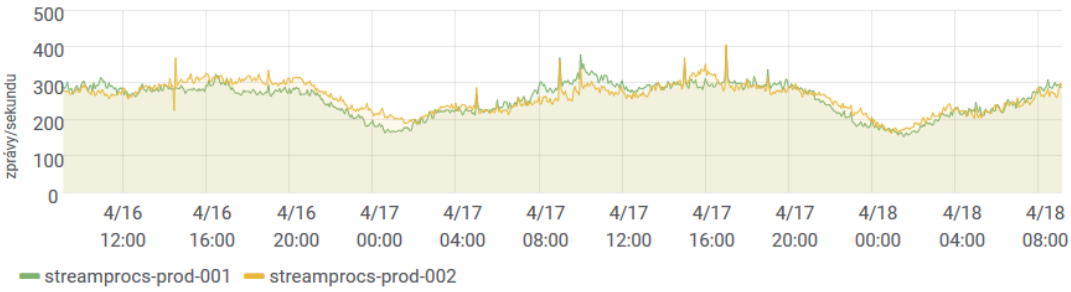
6.1 Testy propustnosti dat

Tato sekce se bude zabývat testováním datové propustnosti aplikací zejména pro přípravu vstupních dat pro databázi *Cobra*. Aplikace psané v jazyce Scala běží každá na dvou produkčních serverech, takže Kafka topic, který zpracovávají, je rovnoměrně rozdělen do oddílů tak, aby obě instance na obou serverech zpracovávaly stejné množství dat.

Test bude probíhat tak, že aplikace budou zastaveny po dobu přibližně jedné hodiny, aby se Kafka na jejich vstupu naplnila zprávami, a poté budou opět spuštěny. Tím dojde k simulaci zotavení aplikace po jejich pádu. Protože se v Kafce nahromadí větší množství dat, bude možné zjistit maximální rychlost zpracovávání.

6.1.1 ODP feeder

ODP feeder je aplikace konzumující *One Data Point*. Na obrázku 6.1 je zobrazena rychlost zpracování zpráv z Kafky pro obě instance aplikace.

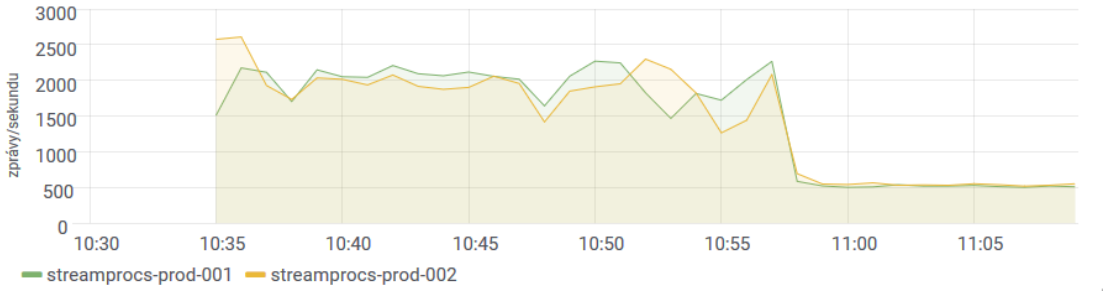


Obrázek 6.1. *ODP feeder* – rychlost zpracování vstupních zpráv za běžných okolností

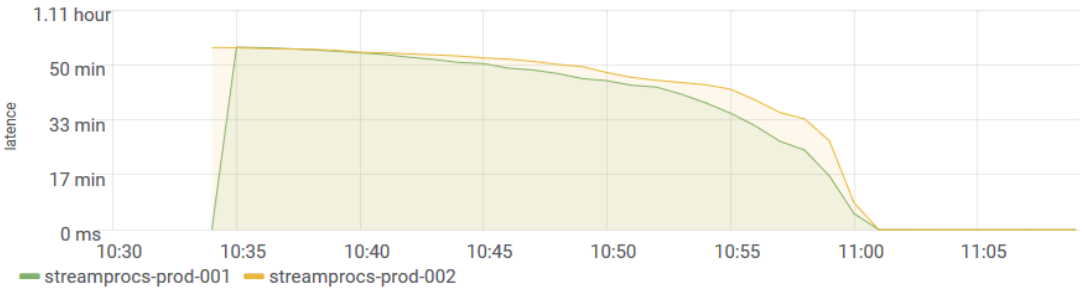
Je vidět, že počet zpracovaných zpráv jednou instancí se pohybuje zhruba mezi 150 a 350 za sekundu. Počet kolísá během dne v závislosti na denní době v oblastech, kde je největší zastoupení aktivních uživatelů Avastu. K nejvyššímu zatížení dochází v odpoledních hodinách středoevropského času a naopak nejméně uživatelů bývá aktivních krátce po středoevropské půlnoci.

Obrázky 6.2 a 6.3 zobrazují chování aplikace po jejím jednohodinovém výpadku. Na obrázku 6.2 je vidět, že obě instance aplikace jsou schopny zpracovat zprávy rychlostí přesahující 2000 zpráv za sekundu. Obrázek 6.3 demonstruje 99. percentil latence

(hodnota zobrazená v grafu vyčísluje maximální hodnotu zpoždění, se kterým je zpracováno minimálně 99 % všech příchozích zpráv). Z grafů lze vyčíst, že v 10:35, kdy došlo k nastartování aplikací a následnému přidělení oddílů Kafky oběma instancím, se zvedla latence na hodnotu 1 hodiny (po takovou dobu se zprávy v Kafce hromadily kvůli zastavení aplikace). Do 11:00 docházelo ke snižování latence až na hodnotu 3 sekund, kde se hodnota ustálila.



Obrázek 6.2. Zotavení po pádu – počet zpracovaných zpráv za sekundu



Obrázek 6.3. Zotavení po pádu – latence (99. percentil)

6.1.2 FileRep feeder

Stejně jako u *One Data Point*, také v topicu obsahujícím datazy na FileRep a odpovědi na ně kolísá počet zpráv během dne podle aktivity uživatelů Avastu. Jejich počet se pohybuje kolem 500 zpráv za sekundu v období nejmenšího provozu a v době špičky přesahuje 1300 zpráv za sekundu. FileRep feeder je napsaný v Pythonu, běží pouze v jedné instanci a za běžného provozu bez problémů stíhá zprávy zpracovat. Po nuceném výpadku délky přibližně jedné hodiny byl skript schopen zprávy zpracovávat rychlostí přes 3000 zpráv za sekundu a za 45 minut se opět vrátit do běžného stavu zpracovávání nově příchozích zpráv.

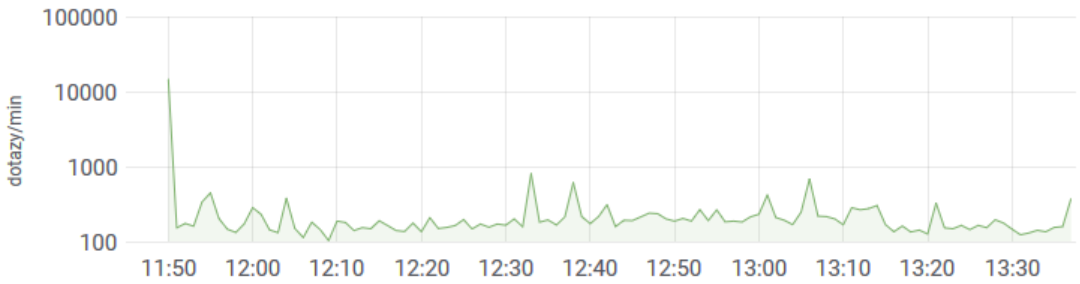
6.1.3 Změny v souborové databázi

Oba skripty pro zpracování událostí přidání souboru zdrojové URL nebo přiřazení souboru do čistého nebo malware slotu fungují na principu periodického spouštění. Skript se vždy spustí (dle aktuálního nastavení každou minutu) a zpracuje zprávy, které se ve frontě nahromadily od minulého běhu. Byly také podrobeny testu, kdy došlo k zastavení jejich spouštění na jednu hodinu a poté opět k jeho obnovení.

Skripty hned během jedné resp. dvou minut dokázaly zpracovat nahromaděných 50000 resp. 100000 zpráv a poté pokračovat se zprávami novými. V nejvyšším zatížení se skript pro zpracování změny souborových slotů dostal na rychlost 820 zpráv za sekundu, zatímco skript pro zpracování nových zdrojů souborů dosáhl rychlosti 1300 zpráv za sekundu.

6.1.4 Cobra feeder

Aplikace pracující mezi Kafka topicy prvního a druhého stupně zpracovává řádově mnohem méně zpráv. Během dne počet příchozích zpráv do *PREPROCS* topicu kolísá mezi 2 a 10 zprávami za sekundu. Vzhledem k tomu, že zde dochází pouze k rozdělení URL na doménu, doménu druhého stupně, cestu a poté k přeposlání do jednoho z topiců druhého stupně, aplikace (opět bežící ve dvou instancích) by měla být schopna zvládnout i mnohem vyšší zátěž.

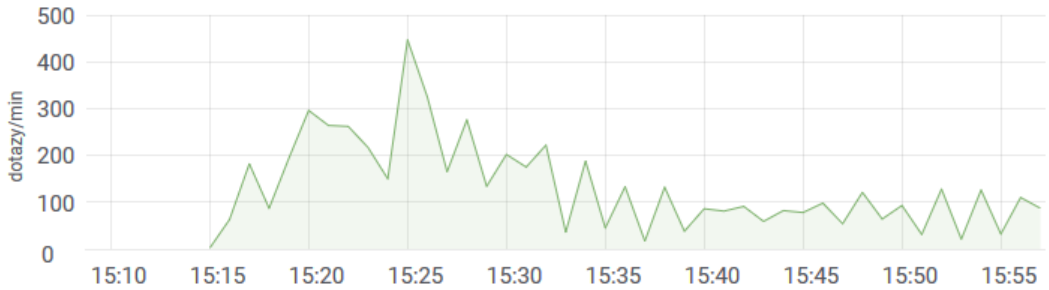


Obrázek 6.4. Zotavení po pádu - počet zpracovaných zpráv

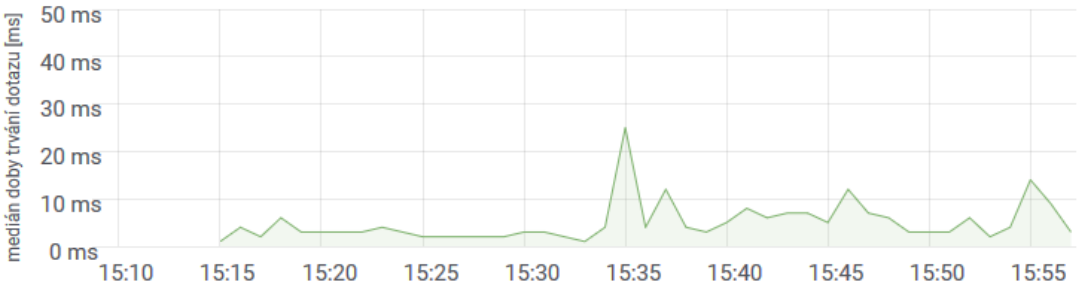
Ověření proběhlo opět testem spuštění po jednohodinové odstávce. Na obrázku 6.4 je zobrazen vývoj rychlosti zpracování zpráv po odstávce (v grafu je zobrazen součet pro obě instance). Lze vidět, že po první minutě, kdy došlo ke zpracování téměř 15000 zpráv, se rychlost snížila zpět k přibližně 200 zprávám za sekundu, což je normální stav.

6.1.5 Filtrace detekcí při hledání potenciálních false positive

Nejslabším místem aplikace pro filtrování detekcí při hledání false positive detekcí je dotaz do *Cobry* na získání doménové reputace. Protože jde o HTTP dotaz, ostatní fáze od vyjmutí zprávy z Kafky přes dekódování protobufu až po rozhodnutí, zda je detekce podezřelá, je možné z hlediska měření propustnosti zanedbat. Schopnost zvládnout velké množství dat bude demonstrováno na počtu obsloužených dotazů na reputaci do Cobry. Na obrázku 6.5 jsou zobrazeny počty jednotlivých dotazů za minutu.



Obrázek 6.5. Počet dotazů do *Cobry* na doménovou reputaci



Obrázek 6.6. Medián doby trvání dotazu na doménovou reputaci

Do doby 15:15 trvala odstávka aplikace, proto byl počet dotazů roven nule. Poté došlo k nárůstu až na hodnotu přesahující 400 dotazů za minutu a dále se již počet snižoval zpět na hodnotu kolem 100 dotazů za minutu, což je hodnota za běžného denního provozu.

Porovnáním obrázků 6.5 a 6.6 je vidět vliv mezimapěti výsledků dotazů na doménovou reputaci. Zatímco počet dotazů do Cobry v čase mezi 15:15 a 15:30 je vyšší než za normálního provozu, ve stejném časovém rozmezí je medián doby trvání dotazů pod hodnotou stálého průměru (3 ms oproti běžným 10 ms). Je to způsobeno tím, že nejdéle trvají dotazy na nejvíce prevalentní domény. Když aplikace zpracovává větší množství dat, stane se pro seskupování dat dominantní podmínka na velikost skupiny nad podmínkou časovou. Tím, že jsou skupiny větší, dostane se do ní i více méně prevalentních domén, pro které dotazy trvají kratší dobu. Díky mezipaměti výsledků se pomalé dotazy volají v rámci jedné skupiny incidentů jenom jednou a výsledný medián doby trvání dotazů se potom snižuje.

6.1.6 Shrnutí

Simulací odstávky aplikací, jejich opětovným spuštěním a sledováním chování při zpracovávání nahromaděných zpráv jsem otestoval maximální množství dat, která jsou schopny zpracovávat s aktuálním nastavením. Tabulka 6.1 znázorňuje porovnání aktuálního zatížení aplikací s jejich maximálním možným zatížením, které by měly zvládnout zpracovat.

název aplikace	běžná zátěž [zpráv/s]	maximální propustnost [zpráv/s]	procentuální zatížení
ODP feeder	300 – 700	4000	7,5 – 17,5
FileRep feeder	500 – 1300	3000	17 – 44
změny souborových slotů	20	820	2,5
nové zdroje souborů	70	1300	5,5
Cobra feeder	4 – 6	250	2
filtrace detekcí	300 – 700	3000	10 – 23

Tabulka 6.1. Porovnání aktuálního a maximálního naměřeného zatížení aplikací

Z výsledků je patrné, že všechny aplikace stíhají bez problému zpracovat nahromaděná data i po výpadku. Většina aplikací se drží za běžných podmínek pod 20 % své maximální propustnosti dat. Pro aplikace psané ve Scale by v případě nutnosti stačilo navýšit počet paralelně pracujících vláken, popřípadě pustit aplikace v další instanci.

Nejvyššího procentuální využití maximální propustnosti dosahuje *FileRep feeder*, ale i ten se drží pod 50 %. To znamená, že i více než dvojnásobné zvýšení množství zpracovaných dat by v současném stavu dokázal zpracovat. Případně lze zvýšení výkonu dosáhnout přepsáním z Pythonu do Scaly, která je k tomuto účelu, tj. ke zpracovávání dat z Kafky, vhodnější.

Pro ostatní části systému *Marlowe* (jako například skript pro klasifikaci domén atd.) není potřeba provádět testy propustnosti dat, protože nepracují v on-line režimu a případný nárůst množství dat je příliš nezatíží.

6.2 Výsledky

29. ledna 2018 došlo ke spuštění systému *Marlowe* do produkčního provozu. K vyhodnocení výsledků použiji období do 20. dubna 2018, kdy jsem výsledky zpracoval. Při analýze výsledků se zaměřím na klasifikaci domén a zamezení vytváření detekcí na čistých souborech.

6.2.1 Klasifikace domén

Během období, které jsem mapoval, identifikoval systém *Marlowe* 87 škodlivých domén. Tyto domény byly zároveň zablokovány webovým štítem Avastu.

Vytvořené detekce měly celkem 3 600 000 hitů[1] u více než 700 000 unikátních uživatelů Avastu. Pro lepší představu těchto hodnot lze říci, že v průměru jednou za sekundu se někdo pokoušel komunikovat s jednou ze zablokovaných domén. Všechny vytvořené detekce jsou v současné době ještě aktivní a žádná z nich nebyla označena jako nesprávná. V tabulce 6.2 je zobrazeno 10 domén, které byly označeny jako škodlivé a na které se snažilo přistupovat největší množství uživatelů. IP adresa *88.99.187.254* řádově převyšuje ostatní domény, protože se několik dalších domén sloužících ke stejnému škodlivému účelu překládá právě na ni. Vytvořená detekce na IP adresu blokuje i komunikaci se všemi doménami, které se na ni přeloží.

doména	počet hitů	počet uživatelů
88.99.187.254	600 000	500 000
bitstowertours.com	90 000	45 000
grabdownloadscapital.com	47 000	33 000
cabbagecomparison.bid	20 000	18 000
138.128.150.133	230 000	16 000
stockdownloadsdelivery.com	22 000	13 000
otwevka.pwm	42 000	11 000
farmupdatecurrent.com	18 000	11 000
cstatics.net	320 000	9 000
dreammy.info	520 000	9 000

Tabulka 6.2. Statistika pro 10 nejvíce prevalentních zablokovaných domén

Dále došlo ke klasifikaci 205 domén jako adware. Klasifikace domén, které nejsou blokovány, slouží ke kategorizaci a pro zabránění jejich nechtěného zablokování. Konkrétně

[1] Počet hitů udává počet incidentů, kdy došlo k zablokování přístupu uživatele na danou stránku. Například pokud se jeden uživatel snaží pětkrát stáhnout soubor ze zablokované domény, počet hitů je 5.

domény oklasifikované systémem *Marlowe* jsou domény, které slouží k šíření souborů více či méně agresivního reklamního softwaru nebo potenciálně nechtěných souborů.

6.2.2 Klasifikace čistých souborů

Kritéria pro klasifikaci souborů jsou nastavena velice přísně. Aby byl soubor oklasifikován jako čistý, musí mít validní čistý certifikát, musí být stažen z domény s *dobrou* reputací a nesmí být detekován žádným jiným antivirem. Tímto nastavením je cíleno jenom na ty nejjasnější případy, aby nedocházelo ke špatným rozhodnutím.

Během necelých tří měsíců se automaticky oklasifikovalo 234 čistých PE souborů. Následně došlo k oklasifikování dalších 2800 souborů, které se z nich rozbalily[1].

17 z těchto souborů má v současné době prevalenci přes 10 milionů (to znamená, že si je spustilo přes 10 milionů uživatelů Avastu). Pokud by tedy došlo k blokování takových souborů, negativní následky by byly značné.

[1] Pokud například dojde ke klasifikaci čistého instalačního souboru, automaticky se jako čisté označí i soubory, které se vytvoří během instalace.

Kapitola 7
Závěr

Cílem práce bylo vyvinout systém *Marlowe* schopný určit reputaci domény na základě klasifikace souborů, které se z ní v minulosti stahovaly. Na základě této reputace měl být systém schopen detekovat anomálie mezi jednotlivými incidenty stahování souborů. Anomálií je myšleno jednak stažení škodlivého souboru z domény, ze které se v minulosti stahovaly čisté soubory, a jednak existence nezablokované domény, ze které se stahuje velké množství škodlivých souborů. Dále bylo cílem vytvořit dashboard pro zobrazení výsledků a jejich případnou manuální analýzu.

Celá práce probíhala ve spolupráci s bezpečností společností *Avast Software, s.r.o.*

Nejprve jsem se věnoval analýze současného stavu této problematiky. Provedl jsem test dostupných služeb pro zjištění reputace domén. Analyzoval jsem také literaturu zabývající se problémem automatické klasifikace domén. Došel jsem k závěru, že žádné stávající funkční veřejné řešení reputace domén na základě analýzy stažených souborů dosud neexistuje.

Pro výpočet reputací domén je potřeba ukládat záznamy o stažených souborech. Za tímto účelem jsem navrhl grafovou databázi *Cobra*. Její implementace nebyla součástí této práce. *Cobra* má dvě části, jednu pro statické uchovávání čistých souborů a jejich zdrojových URL a druhou pro uchování časového okna informací o stažených škodlivých souborech. K tomuto rozdělení došlo kvůli rozdílnému množství dat o čistých a škodlivých souborech.

Pro zpracování různých druhů vstupních dat a jejich vkládání do databáze *Cobra* jsem vytvořil strukturu streamových aplikací, které si mezi sebou posílají zprávy pomocí několika Kafka topiců.

Navrhl a implementoval jsem skript, který z *Cobry* vybírá domény se špatnou reputací a soubory z nich stažené. Na základě analýzy těchto souborů dojde k rozhodnutí, zda je celá doména škodlivá a případně i k vytvoření detekce nad touto doménou.

Pro vyhledávání anomálně detekovaných souborů jsem vytvořil aplikaci, která filtruje detekce souborů stažených z domény s dobrou reputací. Následně jsou tyto přefiltrované soubory analyzovány dalším skriptem. Pokud nedojde k rozřešení souboru, tj. k jeho označení jako čistého, uloží se výsledek do databáze, aby se mohl ručně zpracovat.

Vytvořil jsem HTTP dashboard pro zobrazení nevyřešených souborů. K jednotlivým incidentům jsou dohledány doplňující informace, které jsou následně zobrazeny, aby byla manuální analýza souborů zjednodušena. Z dashboardu je také možné soubory manuálně klasifikovat. Je zde také stránka pro analýzu domény na základě stažených souborů z *Cobry*.

Otestoval jsem datovou propustnost výsledné implementace. Všechny aplikace dokázaly zpracovat několikanásobný počet dat oproti běžnému provozu, čímž jsem ověřil, že i při případném nárůstu vstupních dat bude systém schopen pracovat.

Celý systém *Marlowe* jsem spustil nad produkčními daty Avastu. Během prvních tří měsíců došlo k zablokování 87 domén, čímž došlo k ochránění více než 700 tisíc uživatelů před šířením nebezpečných souborů. Došlo ke klasifikaci více než 3 tisíc souborů, na kterých se díky tomu nebudou dále vytvářet žádné nové detekce.

7.1 Budoucnost projektu *Marlowe*

Současná verze *Marlowe* je nasazena v produkční verzi antivirového programu Avast. Do budoucna existuje několik možných rozšíření.

Aplikace pro zpracování vstupních dat a databáze *Cobra* podporují ukládání incidentů se všemi částmi URL, tj. celou doménou i cestou. V současné době dochází k analýze pouze na úrovni domén druhého stupně. Nabízí se tady možnost využít podobného principu i pro doménová jména s více subdoménami, popř. i pro cesty URL adres. Poté by bylo možné pro jednotlivé URL vypočítat více reputací z těchto částí.

Samotné rozdělení domén podle počtu stažených domén je nastaveno velice konzervativně. Při dostatečném času na testování by mohlo vést k nalezení dynamičtějších hranic a tím rozdělení zpřesnit.

Stejně tak kritéria pro označení domény jako škodlivé na základě analýzy stažených souborů jsou nastavena poměrně přísně, aby se zamezilo zbytečným nesprávným detekcím. I tato kritéria by se mohla v budoucnu nastavit poněkud benevolentněji.

Klasifikace souborů také nabízí prostor ke zvýšení automatizace. Na základě informací, která jsou poskytnuty analytikům v dashboardu pro manuální analýzu, by se mohl postavit automatický klasifikátor.

Pro zvýšení množství dat, které je *Marlowe* schopen zpracovat, by mohlo dojít k přepsání všech aplikací zpracovávajících zprávy z Kafky do jazyka Scala. Aplikace psané ve Scale jsou k tomuto účelu vhodnější a mnohem výkonnější.

Literatura

[1] Manos Antonakakis, Roberto Perdisci, David Dagon, Wenke Lee, and Nick Feamster. Building a dynamic reputation system for dns. In *Proceedings of the 19th USENIX Conference on Security*, USENIX Security'10, pages 18–18, Berkeley, CA, USA, 2010. USENIX Association.

[2] Manos Antonakakis, Roberto Perdisci, Wenke Lee, Nikolaos Vasiloglou, II, and David Dagon. Detecting malware domains at the upper dns hierarchy. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 27–27, Berkeley, CA, USA, 2011. USENIX Association.

[3] Manos Antonakakis, Roberto Perdisci, Yacin Nadji, Nikolaos Vasiloglou, Saeed Abu-Nimeh, Wenke Lee, and David Dagon. From throw-away traffic to bots: Detecting the rise of dga-based malware. In *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, pages 24–24, Berkeley, CA, USA, 2012. USENIX Association.

[4] Mark Felegyhazi, Christian Kreibich, and Vern Paxson. On the potential of proactive domain blacklisting. In *Proceedings of the 3rd USENIX Conference on Large-scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More*, LEET'10, pages 6–6, Berkeley, CA, USA, 2010. USENIX Association.

[5] Denis Konopiský. Avast blocked more than 34 million monero cryptomining attacks in one day, 2017. cit. 2018-03-07, dostupné z: https://blog.avast.com/avast-blocked-more-than-34-million-monero-cryptomining-attacks.

[6] Jakub Křoustek. Wannacry update: The worst ransomware outbreak in history, May 2017. cit. 2018-03-07, dostupné z: https://blog.avast.com/wannacry-update-the-worst-ransomware-outbreak-in-history.

[7] C. Lever, R. Walls, Y. Nadji, D. Dagon, P. McDaniel, and M. Antonakakis. Domain-z: 28 registrations later measuring the exploitation of residual trust in domains. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 691–706, May 2016.

[8] Terry Nelms, Roberto Perdisci, Manos Antonakakis, and Mustaque Ahamad. Webwitness: Investigating, categorizing, and mitigating malware download paths. In *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC'15, pages 1025–1040, Berkeley, CA, USA, 2015. USENIX Association.

[9] Jakub Puchýř and Martin Holena. Random-forest-based analysis of url paths. 2017.

[10] Abhishek Singh. *Portable Executable File Format*, pages 1–15. Springer US, Boston, MA, 2009.

Příloha
Obsah přiloženého CD

- **src/** – složka obsahující zdrojové kódy
  - **cobra-feeder/** – repositář aplikace pro zpracování vstupních dat do Cobry
  - **marlowe-feeder-odp/** – repositář aplikace pro zpracování dat z ODP Kafka topicu
  - **marlowe-odp-fp-finder/** – repositář aplikace pro filtrování FP detekcí
  - **marlowe-package/** – repositář Python balíčku obsahující modul pro komunikaci s Cobrou a souborovou databází a ostatní skripty psané v Pythonu
  - **Mravenec/** – repositář web serveru, složka **marlowe-app** obsahuje zdrojové kódy pro *Marlowe* frontend
- **tex/** – složka s textem diplomové práce a jejím zdrojovým kódem
- **readme.txt** – textový soubor obsahující seznam s obsahem CD