VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

ZPRACOVÁNÍ LOGOVACÍCH INFORMACÍ NA PLATFORMĚ TESKALABS

LOG ANALYSIS USING TESKALAB PLATFORM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **PATRIK KOCINEC**
AUTHOR

VEDOUCÍ PRÁCE **Doc. Ing. ONDŘEJ RYŠAVÝ, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Kocinec Patrik**
Program: Informační technologie
Název: **Zpracování logovacích informací na platformě TeskaLabs**
Log Analysis Using TeskaLab Platform
Kategorie: Počítačové sítě
Zadání:

1. Seznamte se s problematikou zpracování logovacích informací pro účely bezpečnostního monitorování.
2. Nastudujte existující možnosti zpracování zdrojů dat pro použití metod strojového učení.
3. V prostředí TeskaLabs si vyzkoušejte možnosti zpracování dat pomocí metod strojového učení.
4. Navrhněte a realizujte experimentální nástroj pro zpracování logovacích informací za použití vybrané metody a pro několik zdrojů dat.
5. Prototyp otestujte a vyhodnoťte jeho vlastnosti.

Literatura:

- Anton Chuvakin, Kevin Schmidt, Chris Phillips: Logging and Log Management, Syngress, 2013.
- Jiawei Han, Micheline Kamber, Jian Pei: Data Mining Concepts and Techniques, Morgan Kaufmann Publishers, 2012.

Pro udělení zápočtu za první semestr je požadováno:

- Splněné body 1 a 2.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/
Vedoucí práce: **Ryšavý Ondřej, doc. Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 14. května 2020
Datum schválení: 21. října 2019

Abstrakt

Tato práce se zabývá využitím metod strojového učení na zpracování logovacích infomací v systému LogMan.io. Práce zahrnuje popis způsobů zpracování logovacích informací pro účely bezpečnostního monitoringu, dále pak metody strojového učení a principy zpracování dat. Následně se práce zaměřuje na představení systému LogMan.io a jeho komponent. Poté je navržena a implementována aplikace pro zpracování logovacích informací, která využívá metod strojového učení pro detekci nebezpečných adres nad systémem LogMan.io. Při implementaci aplikace pro vytrénování modelu bylo využito vícero metod, které byly otestovány se zaměřením na přesnost detekce.

Abstract

This work describes the usage of machine learning methods for processing logging information on LogMan.io system. The work includes a description of methods of processing logging information for the purposes of security monitoring, as well as machine learning methods and principles of data processing. Subsequently, the work focuses on the introduction of the LogMan.io system and its components. Then, an application for processing logging information is designed and implemented on LogMan.io system, which uses machine learning methods to detect malign domains. When implementing the application for model training, several methods were used focusing on the accuracy of detection.

Klíčová slova

TeskaLabs, LogMan.io, logování, bezpečnostní monitoring, strojové učení

Keywords

TeskaLabs, LogMan.io, logging, security monitoring, machine learning

Citace

KOCINEC, Patrik. *Zpracování logovacích informací na platformě TeskaLabs*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. Ing. Ondřej Ryšavý, Ph.D.

Zpracování logovacích informací na platformě Teska-Labs

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Doc. Ing. Ondřeje Ryšavého Ph.D.

Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Patrik Kocinec
28. května 2020

Poděkování

Děkuji panu Doc. Ing. Ondřeji Ryšavému Ph.D. za vedení mé bakalářské práce. Dále bych chtěl poděkovat Mgr. Dávidu Kosťovi za cenné rady, připomínky a obohacující myšlenky.

Obsah

Kapitola 1

Úvod

Logovací záznamy jsou v dnešní době zdrojem důležitých informací. Ať už pro detekci neoprávněných přístupů na systém, nedostupnosti služby či chybě v konfiguraci aplikace, je nutné tato data shromažďovat a analyzovat je. K určení chyby je nutno stále využívat složitější, sofistikovanější a rychlejší metody, jelikož vyžadujeme co největší přesnost a rychlost.

Standardní přístupy k analýze těchto dat jsou již pomalé, neefektivní a vyžadují obrovské množství zkušeností a znalostí k odhalení potenciální chyby. Každá operace, každý děj za sebou zanechává stopu. Vygenerovaných dat je obrovské množství a důležité informace se většinou ztratí v tomto chaosu. Navíc tato data ztrácí většinu své hodnoty po pár chvílích svého prvního výskytu.

Běžným přístupem v produkčním prostředí je detekce na základě vzorů, která ale naráží na své limity, jelikož nové způsoby útoků se vyvíjejí rychleji, než dokážeme vytvořit vzory na jejich detekci. Dalším přístupem je použití statistických metod, které si na základě pozorování dokážou vytvořit profil normálního chování a pokud se momentální aktivita diametrálně liší od standardu, je hlášena anomálie. Z těchto důvodu je nutné zavést nový způsob či mechanismus detekce anomálií za použití strojového učení, které nás včas a spolehlivě upozorní na nějakou chybu.

Práce je rozdělena na dvě hlavní části. V první, teoretické, rozeberu základní pojmy a informace k pochopení principů bezpečnostního monitoringu, zpracování dat a strojového učení. Z hlediska strojového učení jsou vysvětleny jednotlivé typy a jejich metody. V druhé, praktické části, je popsán postup nasazení systému LogMan.io, příprava dat pro model, implementace aplikace pro vytrénování modelu a navržení a implementace nástroje pro detekci nad systémem LogMan.io za použití vytrénovaného modelu. Na závěr bude aplikace otestována a budou vyhodnoceny výstupy.

Cílem této práce je představit systém LogMan.io od společnosti TeskaLabs, prozkoumat jeho možnosti v oblasti detekčních metod za použití strojového učení, navrhnout možné řešení a to otestovat vytvořením prototypu nástroje. Cílem není vytvořit nový způsob pro detekci, ale pouze pro demonstraci možností systému LogMan.io.

Kapitola 2

Logy

Je mnoho definic popisujících co to jsou logy, logovací data či logovací zprávy a hodně z nich mají vágní, či zavádějící významy. Obsahem logovacích dat jsou logovací zprávy, nebo-li logy. Log je záznam o nějaké akci, nebo události, kterou systém, zařízení nebo software vykoná. Akcí může být prakticky cokoliv, určujícím faktorem je zde zdroj logu. Například to může být přihlášení nebo odhlášení ze systému, přístup k souboru v operačním systému, spuštění služby a mnoho dalších [12].

Tyto události popisují jednoduché či komplexní děje probíhající na různých systémech. Na základě logů o těchto událostech získáváme určitou viditelnost do probíhajících či proběhlých dějů. Právě viditelnost do dějů na monitorovaném systému je jedna ze základních potřeb v kyberbezpečnosti a vedle správy aktiv také nutností pro bezpečnostní monitoring. Logy jsou jeden ze zdrojů dat, ze kterých můžeme tuto viditelnost získat. Díky této viditelnosti můžeme říci, jaké děje probíhají na monitorovaném zařízení [24].

Kromě popisu daného děje většina logů obsahuje Timestamp, což je časové razítko vygenerování logu. Tato informace je velice důležitá pro seřazení posloupnosti událostí. Další částí zprávy je zdroj, ze kterého záznam pochází, abychom znali původce zprávy. Posledním dílkem jsou data, to je samotné tělo zprávy, které obsahuje informace o proběhlém ději. Spojením těchto částí máme ucelenou zprávu o události.

Jak už bylo řečeno výše, viditelnost je pro bezpečnostní monitoring nezbytná, což v konečném důsledku znamená, aby všechny komponenty informačních technologií dokázaly generovat logy, takzvaně logovat. S touto skutečností ale přichází i obrovská výzva bezpečnostního managementu. Je nutné je rychle a správně zpracovat, jelikož obsahují enormní množství detailních informací a k nalezení potenciálních bezpečnostních událostí je vyžadováno rozsáhlých znalostí a zkušeností.

Logy všeobecně

Typů logů existuje obrovské množství, můžeme je rozdělit podle obsahu, zdroje či formátu. Z hlediska obsahu můžeme rozdělit logy na logy z koncových stanic, kde se monitorují například jaké procesy se spouštějí, jaká síťová spojení se navazují a mnoho dalších. Může se jednat o aplikační logy, které jsou vygenerovány nějakou aplikací, informují například o startu aplikace nebo provedeném úkonu aplikace. Systémové logy, které jsou vygenerovány operačním systémem, mohou informovat o nenačteném ovladači, či chybě hardwaru. Dále se může jednat o data ohledně síťové komunikace, například logy z firewallu nebo z dedikovaných síťových sond, či jiných zařízení s viditelností do síťového provozu. Bezpečnostní logy nás informují o bezpečnosti na systému, běžně to jsou neúspěšná přihlášení, pokusy

o neoprávněný přístup k souboru a další.

Každá událost může spadat do určité kategorie závažnosti:

- Informace – Událost popisuje úspěšnou operaci úlohy, například načtení ovladače, spuštění aplikace a další.
- Upozornění – Událost popisuje možnou chybu. Nemusí se zpravidla jednat o kritickou chybu, ale může upozornit na budoucí možný problém. Například ubývající místo na disku.
- Chyba – Událost popisující značný problém jako je ztráta dat. Může se jednat o neúspěšné spuštění služby při zapínání počítače či selhání hardwaru.

Data z endpointů

V dnešní době je čím dál tím víc složitější detekovat anomálie nebo zachycovat malware pomocí síťového provozu. Většina komunikace je zašifrována a monitorovat můžeme pouze to, kam se naše stanice připojuje, s kým komunikuje, kolik dat přenáší, či na jaké porty přistupuje. V dohledné době bude standardní DNS přes HTTPS, čímž se objem informací znovu sníží. Z těchto informací je velice obtížné identifikovat potenciální bezpečnostní události, proto je nutné se přesunout na koncové stanice a monitorovat tam.

Někdo by mohl namítat, že monitorování koncových stanic je zbytečné, pokud se na nich nevyskytují důležitá data. Tento názor je v zásadě mylný, jelikož velké množství útoků pochází právě skrz koncové stanice. Základem je ovládnutí této stanice, což je možné mnoha způsoby. Může se jednat o phishingový email s přílohou nebo třeba nákaza z napadeného webu. Většinou je na vině selhání lidského faktoru, z tohoto důvodu je velmi žádoucí a nutné školit uživatele a informovat je o možných hrozbách [25].

Ochrana před takovými útoky není jednoduchá, samotný antivir je základní, byť neúplnou ochranou, jelikož z velké části detekuje pouze na základě signatur souborů. To znamená, že pokud budete cílem v první vlně libovolné malwarové kampaně, váš antivirus je zbytečný, protože danou signaturu ještě nikdy neviděl. Navíc antivirus je spíše věcí prevence a iniciální detekce průniku, na další sledování pohybu útočníka v systému není většinou stavěný.

Logy z koncových stanic můžeme získat hlavně pomocí vestavěného logování operačního systému, případně pomocí dalšího softwaru, který umí poskytnout hlubší viditelnost a zalogovat více událostí, jako je například Sysmon. Jednotlivé záznamy můžeme rozřadit do kategorií. Mezi nejdůležitější patří:

- Vytvoření procesu – tento záznam popisuje to, že se v systému spustil nějaký proces. Díky této informaci můžeme dále prozkoumat, zda-li je spuštěný proces hrozbou, nebo ne.
- Síťové spojení – tento záznam informuje o vytvořeném TCP nebo UDP spojení na koncové stanici. Obsahuje v sobě zdrojovou i cílovou adresu a čísla portů.
- Vytvoření souboru – pokud se v systému vytvoří nebo přepíše soubor, vygeneruje se o tom záznam. Můžeme monitorovat složky, ve kterých se malware většinou vyskytuje při počáteční infekci.
- Událost v registru - zaznamenává operace související s vytvářením a úpravou klíčů a hodnot v registru. Používá se často k monitorování změn registrů souvisejících se spuštěním procesů po startu systému či specifické modifikace malwarem.

- Přístup k procesu - událost informuje o spuštění procesu jiným procesem. Většinou následuje informací o čtení či zápisu do adresního prostoru cílového procesu. Používá se pro detekci hackovacích nástrojů, které čtou části paměti, ve kterých se mohou nacházet přihlašovací údaje či jejich hashe [32].

Díky výše zmíněným událostem můžeme kontrolovat například spouštění rodičovských procesů a neobvyklých potomků. Praktickou ukázkou nám poslouží malware s názvem Emotet. Jedná se o malware, který byl navrhnut aby z vašeho počítače ukradl přihlašovací údaje nebo další citlivé informace. Počáteční šíření probíhalo primárně přes e-mail, ve kterém byla příloha s Word dokumentem s makrem, které spustilo PowerShell a malware se spojil s útočníkovým serverem, kde stáhl virus, který následně ukradl citlivé údaje nebo působil další škody, například instaloval další malware. Ve spuštěných procesech jsme mohli pozorovat proces PowerShellu s rodičovským procesem onoho Word dokumentu.

Tyto detekce jsou možné za použití systémů EDR[1], které se nespoléhají jen na signatury známých hrozeb, ale poskytují komplexní řešení pro odhalení a identifikaci potenciální hrozby. EDR systémy fungují jako vhodné doplnění klasických antivirů. Zprostředkovávají viditelnost do koncových stanic, analyzují data a děje na nich a často využívají různé techniky strojového učení na detekci anomálií. Jsou schopné odhalit probíhající hrozby hned na začátku a informovat uživatele, aby byly co nejvíce zmírněny škody.

Síťová data

Pod tímto pojmem si můžeme představit prakticky všechna data, která se nám objeví na síti. Může se jednat o samotné soubory, logovací záznamy, zprávy o jednotlivých zařízeních a tak dále. Všechny tyto informace můžeme použít pro monitorování infrastruktury.

Velmi efektivním přístupem se jeví sběr NetFlows, definovaném v RFC 3954 [13], na rozdíl od Full Packet Capture, čili zachycování úplného síťového provozu. NetFlow je otevřený protokol od firmy CiscoSystems. Účelem tohoto protokolu je monitorování síťového provozu na základě IP toků. Jednoduše se z hlaviček paketů vyberou určité informace a díky nim můžeme poskytnout administrátorům podrobný pohled do provozu v reálném čase. S využitím NetFlows dokážeme počítat počet spojení, objem přenesených dat, nová zařízení v síti, můžeme monitorovat cestu toků a tím odhalit například používání neautorizovaného DNS nebo DHCP serveru.

Přístup Full Packet Capture se zdá být hodnotnější na informace, ale je velice náročný na prostředky. Jeho efektivita záleží na objemu šifrovaného provozu, který nelze bez klíče analyzovat. Navíc je k tomu potřeba značný výpočetní výkon i velké množství úložiště. Pro představu, na 100 megabitové lince se zaplní disk o velikosti 1 terabajt za 22 hodin.

Existují ale i řešení, která kombinují oba dva uvedené přístupy. Při normálním provozu sbírají pouze NetFlows a veškerý síťový provoz ukládají do paměti, kde se po určité době zahodí. V momentě detekování nějaké anomálie je zachytáván veškerý provoz pro další analýzu.

2.1 Formáty logů

Pod pojmem formát rozumíme v tomto kontextu syntaxi, uspořádání a formu textové podoby jednotlivých částí logovacího záznamu. Díky nim je zajištěna konzistence záznamů, kterou můžeme využít pro rozdělení logu k následnému uložení ve strukturované podobě.

[1] Endpoint Detection and Response

V praxi je rozšířeno vícero formátů, například Common Log Format[2], který využívá hodně webových serverů, například Apache Web Server, IBM HTTP Server, a je hojně podporován. Dále se můžeme setkat s Combined Log Format, Extended Log Format i některými vlastními formáty. V zásadě se jedná o Common Log Format s přidanými položkami.

Příkladem pro Common Log Format může být GET požadavek na Apache server verze 2.4:

10.10.1.1 – admin [10/Jan/2010:14:33:02 +0100] "GET /index.html HTTP/1.0" 200 3458

- *10.10.1.1* – IP adresa uživatele.
- *admin* – uživatelské jméno.
- *10/Jan/2010:14:33:02 +0100* – přesný datum a čas včetně časového pásma.
- *GET /index.html HTTP/1.0* – GET požadavek protokolu HTTP verze 1.0. Požadovaný soubor byl */index.html*.
- *200* – návratový kód požadavku.
- *3458* – velikost souboru v bajtech.

Jako další příklad uvedu logy zasílané protokolem Syslog, který je definován v RFC 5424 [17]. Syslog zasílá logy přes síť v určitém formátu, který obsahuje v hlavičce Timestamp, závažnost, IP adresu nebo hostname stanice a v těle zprávy je obsažena zpráva o události.

<30>Oct 11 12:13:14 server.local su: 'su root' failed for bfu on /dev/15

- *<30>* – hodnota priority, reprezentuje zařízení a závažnost.
- *Oct 11 12:13:14* – přesný datum a čas.
- *server.local* – hostname zařízení, na kterém tato zpráva vznikla.
- *su: 'su root' failed for bfu on /dev/15* – tělo zprávy, příkaz o udělení vyšších práv.

Posledním příkladem je uchovávání logovacích informací v JSON formátu. JSON, jakožto způsob zápisu dat, byl původně navrhnut pro přenos dat. Je vhodný pro více záměrů. Je nezávislý na počítačové platformě a téměř každý programovací jazyk ho umí číst. Informace uchovává ve formě klíč-hodnota, což umožňuje rychlé vyhledávání a ukládání. Díky této vlastnosti nepotřebujeme implementovat další syntaktické analyzátory, které dovolí snadnější vyhledávání. Příklad je log z web serveru Apache2, popisující GET požadavek.

{"time":"[11/Aug/2014:17:21:45 +0000]", "remoteIP":"10.10.1.1", "host":"admin",
"request":"/index.html", "method":"GET", "status":"200",
"userAgent":"Mozilla/5.0 (Windows NT 10.0; Win64; x64)", "referer":"-"}

- *"time":"[11/Aug/2019:17:21:45 +0000]"* – přesný datum a čas včetně časového pásma.
- *"remoteIP":"10.10.1.1"* – IP adresa zařízení, ze kterého tento požadavek vzešel.

[2]Více viz. http://publib.boulder.ibm.com/tividd/td/ITWSA/ITWSA_info45/en_US/HTML/guide/c-logs.html#common

- *"host":"admin"* – hostname zdroje.
- *"request":"/index.html"* – Požadovaný dokument byl */index.html*.
- *"method":"GET"* – Dotazovací metoda HTTP protokolu.
- *"status":"200"* – Stavový kód HTTP protokolu, je součástí hlavičky odpovědi serveru na uživatelský požadavek. Kód 200 signalizuje úspěch.
- *"userAgent":"Mozilla/5.0 (Windows NT 10.0; Win64; x64)"* – User-Agent informuje server o aplikaci, operačním systému a jeho verzi, ze kterého požadavek vzešel.
- *"referer":"-"* – Označení pro URI, ze kterého byla webová stránka navštívena.

2.2 Sběr logů

Každá organizace, bez ohledu na její zaměření, by měla sbírat logy na centrální úložiště. Díky němu je možné zajistit archivaci a retenci dat. Archivace se provádí k zajištění uchování informací k dispozici po dobu v řádech let. Centralizovaný sběr logů také výrazně usnadňuje a urychluje případné prohledávání těchto dat. Ať už pro svou informaci o jednotlivých prvcích, či pro možnost detekce a analýzy kybernetických bezpečnostních incidentů. V případě některých organizací[3] to vyžaduje i zákonná úprava. Více o této problematice je napsáno v kapitole 2.7. Z tohoto důvodu je potřebné, aby se záznamy ze všech zdrojů shromažďovaly na centrálním úložišti, které je zálohované a zabezpečuje všechny požadavky na uložení dat i dostupnosti.

Existují dvě možnosti sběru dat, jedna je sběr agentem a druhá sběr bez agenta. Sběr agentem nazýváme lokálním sběrem logů a sběr bez agenta se nazývá vzdáleným sběrem logů.

Lokální sběr logů

Tento přístup je vcelku jednoduchý. Jedná se o to, že na každé zařízení se nainstaluje a nakonfiguruje agent, který sbírá logy a zasílá na určené místo. Nicméně tento přístup je náročnější na režii, jelikož je nutné agenta udržovat aktualizovaného i správně spravovat konfiguraci, což na desítkách, ne-li stovkách stanicích může být velmi obtížné.

Na rozdíl od sběru bez agenta je možné zde nakonfigurovat šifrované a spolehlivé spojení. Je tu i možnost zasílat strukturovaná data, ať už v JSON formátu, binární podobě, nebo mnoho dalších, taktéž tato data filtrovat nebo normalizovat na straně klienta.

Vzdálený sběr logů

Druhým přístupem je vzdálený sběr logů, nebo použití standardních nástrojů, které jsou již nainstalovány na konkrétním stroji. Typické pro operační systém Linux je Syslog, kde směrování logů je již nakonfigurováno. Taktéž logy můžeme přečíst přes SSH.

Nevýhodou tohoto přístupu, se kterou se můžeme setkat, je to, že zařízení zasílá logovací data přes UDP, čímž není zajištěna bezpečnost a spolehlivost. Data poslaná přes UDP se zahodí, pokud je logovací server restartován, nebo je v nějakou dobu nedostupný. Šifrované TLS spojení je podporováno jen zřídkakdy, navzdory tomu, že spolehlivý a zašifrovaný přenos může být mandatorním požadavkem na splnění.

[3] Více viz. § 3 vyhlášky č. 181/2014 Sb.

Další možností je sběr logů přes API. Tento způsob je možné využít ve spojení s přenosem přes HTTPS, čímž zajistíme šifrovaný i spolehlivý přenos. Touto cestou je možné i filtrovat sbíraná data, čímž můžeme ušetřit zdroje logovacího serveru. Například není nutné vyžadovat všechna data, ale jen ty s určitou závažností, vygenerované konkrétní službou a další.

2.3 Zpracování logů v reálném čase

Prvním druhem zpracování logů je zpracování v reálném čase. Jak už název napovídá, logovací záznam se zpracuje v okamžiku, kdy ho systém přijme. Výhody tohoto přístupu jsou takové, že jakýkoliv systém může data zasílat na vstup a výstup je okamžitě dostupný pro kohokoliv. Toto se stává užitečným pro režim ladění a monitorování, protože potřebujeme vědět co se děje ihned. Na rozdíl od dávkového zpracování nám tento způsob umožňuje v okamžiku získat výstup a zkontrolovat jeho správnost, spočítat si statistiky, nebo si data vizualizovat.

Z hlediska bezpečnostního monitoringu nám to umožňuje získat velice rychle povědomí o možné bezpečnostní události, což je velmi důležité, jelikož čas je rozhodujícím faktorem, jak moc se dokáže útočník rozšířit po síti a jaké škody způsobí. Firma CrowdStrike vydala dokument za rok 2018 se statistikou, jaký průměrný čas potřebuje která státem podporovaná skupina, na to, aby dosáhla `lateral movement`[4] po počátečním prolomení zařízení. Na prvním místě s obrovským náskokem figurovali útočníci asociovaní s Ruskou federací, kteří na přesun od iniciálního průniku do zařízení k rozšíření po síti potřebovali v průměru necelých devatenáct minut. Následovaly skupiny ze Severní Korei s průměrným časem lehce nad dvě hodiny dvacet minut. Za nimi týmy z Číny, Íránu a ostatní soukromé skupiny [14]. Z těchto důvodů je opravdu nutné zpracovávat logy ihned, abychom se dozvěděli o možném průniku útočníkem co nejdříve.

2.4 Dávkové zpracování logů

Druhým základním druhem zpracování logů je dávkové zpracování. K tomu, abychom ho mohli popsat, je zapotřebí si nejdřív vyjasnit, co znamená pojem dávka v této souvislosti. V praxi se můžeme setkat se dvěma přístupy. Jeden je dávkové zasílání a druhý dávkové zpracování.

V prvním případě se jedná o to, že daný procesor, který je určen ke zpracování dat, přijme v jeden moment objem dat za uplynulou specifikovanou dobu. V praxi se snažíme držet daná okna co nejmenší, jelikož o možné bezpečnostní události se dozvíme až na konci časového okna.

V druhém případě služba přijímá logy v reálném čase v rámci určitého časového okna, ale jejich vyhodnocení provádí až na jeho konci. Čili data si dokážeme zobrazit okamžitě, ale výstup z aplikační logiky až po uplynutí časového okna. Jako příklad nám poslouží produkt Flowmon, který obsahuje jednotku na sběr NetFlows a modul pro detekci anomálií na síti, ve kterém jsou sesbíraná data vyhodnocována pomocí různých detekčních metod. Jedna z takových metod je pro detekci přenosů velkých objemů dat na síti. Zmíněný modul si počítá průměr přenesených dat na celé síti v určitém časovém okně. Jestliže jedno zařízení přenese v rámci časového okna větší objem dat, než je spočítaný průměr na sítí, je vygenerována událost pomocí této metody.

[4] Útočník se začne šířit po dalších zařízeních v síti.

2.5 Fronty zpráv

V dnešní době se používají různé druhy front zpráv pro sběr či zpracování logovacích záznamů. Díky nim je možné rychleji a spolehlivěji reagovat na samotné logy. Podnětem pro hledání způsobů, které dovolují výměnu informací mezi systémy byl zřejmě velký rozvoj distribuovaných systémů, aby byla zajištěna jistá interoperabilita mezi nimi. Většina architektur je rozdělena na menší, mezi sebou nezávislé jednotky a fronty zpráv dovolují aplikacím a systémům komunikovat mezi sebou pomocí zasílání zpráv. Můžou také fungovat jako dočasné úložiště, pokud je cíl zaneprázdněný nebo nedostupný. Tím jsme schopni zajistit určitou spolehlivost.

Společným rysem většiny front je asynchronní komunikace. To znamená, že dva systémy spolu nekomunikují napřímo, ale prostřednictvím dané fronty. Zdrojový systém do ni odešle zprávu a cílový systém si ji v době své aktivity vyzvedne.

Podobným principem je `publish-subscribe`, který se oproti klasické frontě liší v některých vlastnostech. Jednotlivé fronty můžeme zpřístupnit více producentům či konzumentům, čímž z nich vytvoříme takzvané topicy, nebo-li témy. Nejsme tak nuceni odesílat jednu zprávu na vícero destinací, ale pouze na jednu, ze které si ji odeberou ti, kteří jsou připojení na danou tému. Dále také zajišťuje to, aby jednotliví konzumenti získali zprávy z témy ve správném pořadí [10].

Apache Kafka

Apache Kafka je open-source distribuovaná streamovací platforma, která funguguje na principu `publish-subscribe`. Původně byla tvořena společností LinkedIn a od července roku 2011 je vyvíjena pod záštitou Apache Software Foundation [28].

Kafka byla nejdříve navržena se záměrem sbírat a sledovat obrovské objemy dat generované webovou stránkou, například uživateli zobrazené a navštívené stránky a další aktivity za účelem analýzy a zpracování v reálném čase.

Kafka je podobná frontě zpráv a dovoluje vytvářet streamované pipelines pro spolehlivý přenos mezi systémy a aplikacemi. Zároveň umožňuje i transformovat přijatá data, či na ně reagovat. Konzumentům dovoluje přistupovat ke zprávám v libovolném pořadí. Konzumentem může být jakákoliv aplikace, či systém, který odebírá zprávy z fronty nebo témy.

RabbitMQ

RabbitMQ je open-source message-oriented middleware (MOM)[5]. MOM je střední vrstvou pro distribuovaný systém a zaopatřuje interní komunikaci mezi jednotlivými komponenty. Navíc dovoluje, aby moduly byly rozšířeny na vícero platforem či operačních systémů.

RabbitMQ zajišťuje hlavně směrování a bezpečnost, ale také podporuje asynchronní komunikaci, tím je myšleno to, že producent a konzument nemusí být aktivní v jednu chvíli. Pakliže konzument není aktivní, zprávy zůstávají ve frontě, dokud je nepřečte. Tímto způsobem je zajištěna bezztrátovost zpráv. Jelikož RabbitMQ funguje na principu fronty, tak zachovává i pořadí zpráv, v jakém byly přijaty.

Rozdílů oproti Apache Kafka je několik. RabbitMQ je čistá fronta a zprávy se zahazují po přečtení. Navíc v něm je možné dávat zprávám určitou prioritu, takže důležité zprávy jsou zpracovány jako první.

[5]Softwarová nebo hardwarová infrastruktura, která poskytuje zasílání a přijímání zpráv distribuovaným systémům.

2.6 Analýza logů

Analýza logů je proces posouzení, pochopení a zhodnocení logovacích záznamů. Pro tuto činnost existuje mnoho automatizovaných nástrojů, které například indexují jednotlivé záznamy, čímž usnadňují jejich prohledávání, nebo rovnou provádějí detekci na základě určených pravidel. Nicméně i přesto se stále využívá manuální procházení záznamů, abychom pochopili i okolnosti celého děje. Například nás systém upozorní na nefunkční službu a my následnou analýzou zjistíme, že před vypnutím služby se na systém někdo přihlásil z neznámé adresy a zmíněnou službu vypnul.

Pro takové analýzy je nutné splnit některé důležité prerekvizity. Jedna z hlavních je mít synchronizovaný čas na všech zdrojích, aby bylo možné vytvořit časovou osu. Analytici musí zajistit, že se logy skládají z kompletního rozsahu zpráv a jsou správně interpretovány. Logovací prvky by měly být normalizovány a měly by obsahovat stejnou terminologii, abychom se vyhnuli zmatkům. Například se může jednat o špatně nastavené časové pásmo, že jeden systém pracuje s koordinovaným světovým časem(UTC) a druhý s centrálním evropským časem(CET).

V praxi jsou často logy majoritním zdrojem informací pro detekci anomálií v síti. Základním bodem při detekci je schopnost odlišit neautorizovanou, nebo-li nedovolenou, aktivitu od autorizované [26]. Typickým případem použití analýzy logů může být odstraňování problémů v systému nebo síti, pochopení chovaní uživatelů, nebo forenzní analýza bezpečnostního incidentu.

2.7 Logování z pohledu kyberzákona a standardů NIST

V sekci 2.2 jsem se zmínil o některých zákonných požadavcích na logování a považuji za důležité objasnit určité pojmy z právního hlediska. Některé systémy mohou pracovat s citlivými údaji, jako jsou například osobní údaje, bankovní údaje, smlouvy a další. Proto je nutné dodržovat určité zákonem dané pravidla.

Vyhláška 82/2018 Sb. přikazuje povinným osobám, aby udržovali rozsah aktiv, u kterých je zaznamenávání bezpečnostních a provozních událostí prováděno. Tím rozumíme seznam zařízení, která logují. Dále je nutné, aby povinná osoba zajistila jednoznačnou síťovou identifikaci zařízení původce, sběr informací o bezpečnostních a provozních událostech, zejména datum a čas včetně specifikace časového pásma, typ činnosti, identifikaci technického aktiva, které činnost zaznamenalo, jednoznačnou identifikaci zařízení původce a úspěšnost či neúspěšnost činnosti. Také musí zajistit ochranu informací před neoprávněným čtením a jakoukoli změnou. Musí zaznamenat veškeré přihlašování a odhlašování ke všem účtům, ať úspěšné či neúspěšné, činnosti provedené administrátory, jakoukoli manipulaci s účty, pokus o provedení činnosti bez dostatečného oprávnění a mnoho dalších. Další povinností je zajistit synchronizaci jednotného času technických aktiv nejméně jednou za 24 hodin [7].

Doporučení od National Institute of Standards and Technology(NIST) popisuje celou problematiku logování. Od vysvětlení základních pojmů, důležitosti dat, přes samotný sběr logovacích záznamů, jejich uložení i analýzu. Souvislosti se zákonem o kybernetické bezpečnosti najdeme v synchronizaci časového razítka, zajištění důvěrnosti, dostupnosti a integrity [23].

Retence dat

Retence dat znamená pravidelnou archivaci logovacích záznamů jako součást standardního provozu [23]. Vyhláška 82/2018 Sb. specifikuje přesnou dobu, po jakou mají být určité logy ukládány a archivovány. Pro správce a provozovatele kritické informační infrastruktury a informačního systému základní služby to je nejméně 18 měsíců, po kterou mají být záznamy k dispozici. Pro správce a provozovatele významného informačního systému to je doba 12 měsíců.

Národní úřad pro kybernetickou a informační bezpečnost v roce 2016 vydal dokument[6], který popisuje doporučení na minimální požadavky pro logovací záznamy, které musí být zajištěny pro spolehlivou ex-post analýzu kybernetických bezpečnostních incidentů. V něm rozděluje zdroje logů na skupiny:

- SEC – bezpečnostní software a nástroje, jako jsou antivirus, IPS a IDS systémy, VPN, web proxy, firewally a routery a další.
- OS – servery, pracovní stanice a síťové prvky.
- APP – logování chodu aplikací.

Dobu, po kterou data musí být k dispozici dále určuje podle toho, zda-li se jedná o významný informační systém[7] či kritickou informační infrastrukturu[8]. Pokud se jedná o významný informační systém, je doporučeno zálohovat logy minimálně po dobu tří měsíců. V případě kritické informační infrastruktury to je dvojnásobek, čili šest měsíců.

Ochrana osobních údajů

V roce 2018 vešlo v platnost nařízení GDPR, které ovlivňuje některé body zpracování logovacích informací. Správce a zpracovatel je povinen provést technická a organizační opatření za účelem ochrany osobních údajů, zajistit jejich důvěrnost, bezpečnost a zamezit neoprávněným přístupům k nim. Tato ochrana je míněna velice zeširoka, v neomezeném časovém rámci a v jakékoliv situaci, která by mohla vést ke zneužití osobních údajů. Zajištěné opatření musí být řádně zdokumentováno za účelem doložitelnosti. Dále by správce měl být schopen doložit, že ochranná opatření zodpovědně reviduje a aktualizuje [22].

Navzdory tomu, že zmíněné nařízení přísně vyžaduje komplexní zabezpečení spravovaných dat, ponechává volnost ve stanovení konkrétních opatření a prostředků, které správce a zpracovatel využijí k provedení daného úkolu. Při tvorbě ochranných opatření, se přihlíží k povaze, rozsahu a účelu zpracování [29]. Nejvhodnějším způsobem je pseudononymizace a šifrování dat.

Pseudononymizace znamená nahrazení identifikačních údajů osob nějakým identifikátorem, například číslem. Jejich úkolem je ochránit soubory dat s citlivými osobními údaji tak, aby nebylo možné spárovat tyto údaje s konkrétními lidmi, citlivé údaje tak zůstanou přiřazeny pouze zástupnému číslu – pseudonymu. Soubor, který páruje pseudonymy s konkrétními osobami, je od souboru s citlivými daty oddělen. Jen ten, kdo má k dispozici oba soubory, může správně přiřadit citlivé údaje k lidem [8].

[6] Více viz. https://www.govcert.cz/cs/informacni-servis/doporuceni/2485-doporuceni-na-minimalni-pozadavky-pro-logy-ktere-musi-byt-zajisteny-pro-spolehlivou-ex-post-analyzu-kybernetickych-bezpecnostnich-incidentu/

[7] Vyhláška 205/2016 Sb., Vyhláška, kterou se mění vyhláška č. 317/2014 Sb., o významných informačních systémech a jejich určujících kritériích

[8] Zákon 315/2014 Sb., Nařízení vlády, kterým se mění nařízení vlády č. 432/2010 Sb., o kritériích pro určení prvku kritické infrastruktury

2.8 Big data versus small data

Big data

Není žádná přesná a obecná definice, co to jsou big data. Pod pojmem big data si ale můžeme představit data tak veliké a komplexní, že jejich zpracování běžnými databázovými prostředky je velice obtížné. Z těchto důvodů je nutné zavádět nové technologie pro sběr, zpracování, ukládání, vyhledávání a analýzu [20].

Tento pojem byl zaveden z toho důvodu, že se generují data mnohonásobně vyšší rychlostí, než tomu bylo v minulosti. A tato rychlost se pořád zvyšuje. Přibývají nové typy dat, ať už biometrická data, geografická data a další. S touto skutečností samozřejmě roste i objem dat, které je potřeba zpracovat. Nicméně jak už bylo řečeno v úvodu, stávající způsoby už jsou nedostačující a je nutné zavést nové.

Nejrozšířenější vlastnosti Big data jsou podle písmene "V". Charakterizují je anglickými slovy začínající na zmíněné písmeno "V". Jejich počet se liší podle zdroje, ale nejčastěji se jedná o Volume, Velocity a Variety. Jako další se uvadí Veracity a Value [15].

- Volume – objem dat, popisuje množství, které je potřeba uložit a analyzovat. Jedná se o velikosti v řádech terabajtů. Obrovský objem dat je jeden z hlavních důvodů, proč na jejich zpracování nestačí klasické databázové systémy. Z tohoto důvodu je nutné zavést nové přístupy.

- Velocity – rychlost produkce a přenosu dat. Každým rokem se mnohonásobně zvětšuje a velký podíl na tom mají například sociální sítě.

- Variety – rozmanitost dat. Data mohou pocházet z různých zdrojů, jsou v rozdílných formátech a obsahují všemožné datové typy.

- Veracity – přesnost a důvěryhodnost dat. Data nemusí pocházet z důvěryhodných zdrojů, mohou být nějakým způsobem nepřesná. Systémy na zpracování Big data musí být schopny rozpoznat tuto vlastnost.

- Value – hodnota dat. Nejedná se o vlastnost čistě Big data, ale všeobecně je nutné, aby námi zpracovávaná data měly pro nás nějakou hodnotu a jejich zpracování bylo přínosné.

Small data

Small data je takové množství dat, které můžete vhodně uložit a zpracovat na jednom stroji, zejména high-end laptopu nebo serveru [30]. Data jsou na nosiči a ve formátu, kdy jsou přístupná, nesou nějakou informaci a jsou zpracovatelná.

Na rozdíl od Big data je dokážeme ukládat a zpracovávat na běžném stroji s běžnými technologiemi a softwarem. Většinou jsou ve známém formátu. Může se jednat o textové dokumenty, fotky, videa, databáze uživatelů či produktů, technické dokumenty a mnoho dalších.

Kapitola 3

Strojové učení a zpracování dat

Pod pojmem zpracování dat chápeme takové operace, které přemění prvotní data do smysluplné informace. Aby byl tento proces možný, potřebujeme k jeho vykonání odborníky, kteří ovládají techniky pro analýzu a zpracování dat. Tento přístup je velmi časově a finančně náročný, jelikož se může jednat o obrovské množství dat, které nelze manuálně zpracovat. Mimo velkého množství můžeme také mluvit o různorodosti těchto dat, například obrázky, textová data, geografická data, data z bank, informačních systémů, biometrická data a další. Tato data neustále rostou a mění se. Prakticky splňují všechny vlastnosti Big data. Abychom byli schopni z nich efektivně získat hodnotu, kterou nesou, je potřeba použít strojové učení, které nám umožní zautomatizovat tento proces a poskytne přesnější výsledky.

Strojové učení neznamená, že se stroj sám něco naučí. Spíše tato oblast dává počítači schopnost se učit, aniž by byl explicitně naprogramován [33]. Typické strojové učení znamená, že najdeme nějaký matematický vzorec, který při aplikaci na soubor vstupů, trénovacích dat, vyprodukuje požadovaný výstup. Obvykle tento vzorec generuje správné výstupy na většinu ostatních vstupů za předpokladu, že jsou podobné trénovacím datům. Trénovací data je soubor dat určených pro učení algoritmu. Nejedná se o typické učení, protože pokud bychom jemně deformovali vstupní data, výstup bude pravděpodobně nesprávný [11]. Cílem procesu učení je vytvoření modelu. Existuje několik definic, ale v podstatě se jedná už o vytrénovaný algoritmus, který produkuje správné výstupy pro trénovací data s požadovanou úspěšností. Strojové učení je podmnožinou umělé inteligence, která představuje mnohem komplexnější přístup a řešení problémů.

Hlavní výzvou k vyřešení pro nasazení umělé inteligence je to, že počítače nedokážou zpracovávat data a vnější podněty tak, jako lidé. Představme si to například na jízdě autem. Jako řidič přijímáte a zpracováváte podněty, které si ani neuvědomujete a aktivně o nich nepřemýšlíte. Vnímáte vozovku, krajnice, ostatní řidiče, jejich řízení a snažíte se bezpečně dopravit do nějakého bodu. Počítače musí přijímat a zpracovávat všechny informace a rozhodnout, co je důležité a co ne. Dalším příkladem je typická věc ve vyjadřování významu slova. Pokud vám váš partner poví, že je venku zima, pravděpodobně tím chce říct, že byste mu měli donést teplejší oblečení. Člověk to dokáže pochopit, ale naučit počítač rozeznávat význam slov, je složité. A to je právě cílem umělé inteligence, nahradit člověka počítačem tam, kde to jde.

Podle expertů je 50% šance, že umělá inteligence překoná člověka ve všech úkolech do roku 2063. V následující dekádě mají autonomní vozidla nahradit miliony řidičů. Kromě možné nezaměstnanosti ale přibudou nové úkoly, jako je například přestavění infrastruktury, zajištění kyberbezpečnosti vozidel a ostatní. Další odvětví, které budou nejvíce ovlivněny

umělou inteligencí bude překládání jazyků, které už teď využívá mnohé techniky strojového učení [18].

3.1 Typy strojového učení

Strojové učení může být vhodné k řešení vícero typů úloh nad různými daty. Podle vstupních dat a řešeného problému můžeme použít jeden ze dvou základních skupin algoritmů strojového učení. V základě se jedná o supervised learning a unsupervised learning.

Supervised learning pracuje s datasetem, který je kolekce označených příkladů. Typickým druhem supervised learningu je regrese a klasifikace. Při regresi se stroj učí na základě vstupních dat určit výstupní hodnotu. Při klasifikaci se stroj snaží najít funkci, která rozdělí vstupní data do dvou nebo více tříd.

Na rozdíl od supervised learning, unsupervised learning pracuje s datasetem, který nemá označené příklady. Nejznámějším typem unsupervised learningu je clustering, při něm se stroj snaží zařadit položky do skupin s podobnými vlastnostmi. Je vhodný při použití na neznámá data, když v nich chceme najít nějakou strukturu.

Specifickým typem strojového učení, který se nedá zařadit do žádné z výše uvedených skupin, jelikož jejich přístupy může využívat, je deep learning. Deep learning je podmnožinou strojového učení a znamená to, že algoritmus pracuje s více než jednou vrstvou mezi vstupem a výstupem. Vrstvou rozumíme jakousi samostatnou část, která přijme vstup, transformuje ho sadou většinou nelineárních funkcí a výstup pošle další vrstvě. Hloubka modelu udává počet vrstev.

Dalším speciálním typem učení je reinforcement learning. Reinforcement learning je metoda, ve které stroj interaguje s okolím a funguje na principu pokus-omyl. Tudíž se opravdu "učí" na základě vlastní zkušenosti.

Supervised learning

Supervised learning se snaží určit funkci z trénovacích dat. Každý element z vstupních dat je označen správným výstupem a algoritmus si vytvoří model, který na základě trénovacích dat se bude snažit pomocí určené funkce odhadnout výstup pro neoznačené data. Například budeme chtít určit druh zvířete na základě určitého popisu. Jako popis může být použita váha, velikost, počet končetin, barva srsti či kůže a další. V trénovacím datasetu musíme mít označené příklady, takže u každého popisu bude i informace, o jaké zvíře se jedná. Algoritmus se bude snažit najít nějaký vztah mezi jednotlivými informacemi a poté se pokusí odhadnout neoznačená zvířata.
Na základě požadovaného výstupu můžeme použít dva základní typy supervised learningu:

- Klasifikace – Forma supervised learningu, při které se stroj učí na základě vstupních dat určit výstupní hodnotu v podobě označení. Může se jednat o přiřazení označení, které může nabývat například typu žena/muž, zdravý/nemocný, nebo může být výstupem číslo, například pravděpodobnost, které může analytik použít k přiřazení označení. Při dvou možnostech označení mluvíme o binární klasifikaci. Pokud můžeme vstup označit více možnostmi, jedná se o vícetřídní klasifikaci. Typickým příkladem klasifikace je jednoduchá detekce spamových e-mailů.

- Regrese – Forma supervised learningu, při které se stroj učí na základě vstupních dat určit výstupní hodnotu v podobě reálného čísla. Například se může jednat o odha-

dování hodnoty domu na základě jeho vlastností, jako je poloha, počet místností a další.

Unsupervised learning

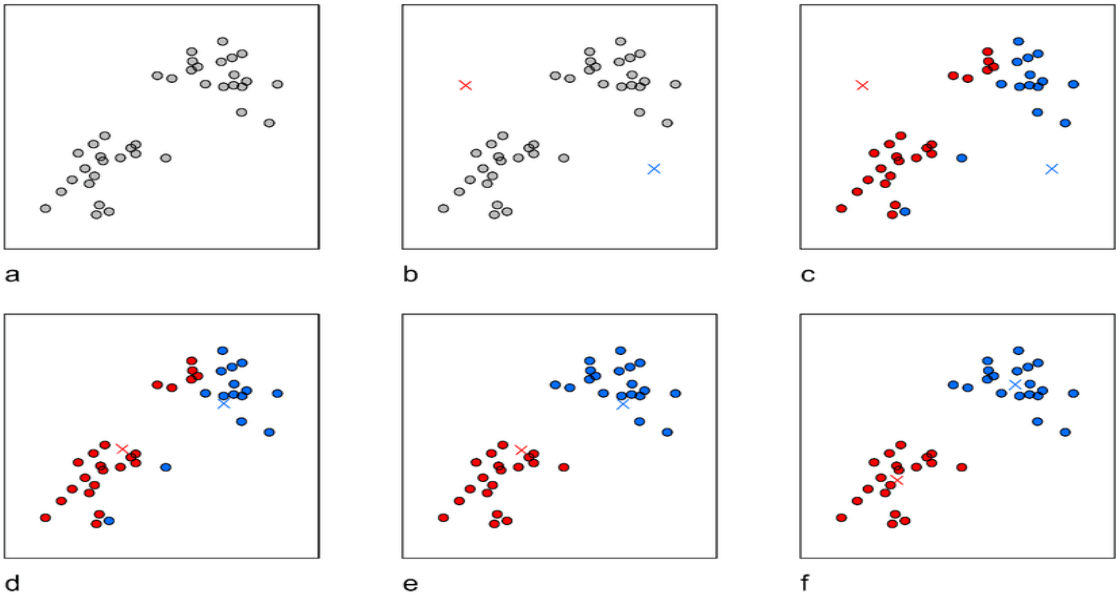
Unsupervised learning je metoda strojového učení, kde nepotřebujeme nebo nemůžeme učit daný model. Například budeme pracovat s neznámými daty nebo tak složitými, že nebude reálné je všechny označit. Namísto toho si model najde vlastní způsob, jak objevit informace. Vstupem pro tuto metodu je neoznačený dataset příkladů. To je ale velmi problematické pro velkou část praktických použití, jelikož chybí označení dat, které představuje nějaký bod, podle kterého se můžeme rozhodnout, zda-li model produkuje takové výstupy, jaké chceme. Podstatou činnosti unsupervised learningu je hledání společných vlastností vstupních dat. Používá se většinou v případech, kdy nejsme schopni určit, co přesně v datech hledáme. Nejběžnější typy unsupervised learningu jsou:

- Clustering – neboli shluková analýza, která seskupuje entity do skupin na základě vzájemné podobnosti či odlišnosti. Za pomocí různých technik se snaží hledat struktury v datech a umístit je do určité skupiny [11]. Je to běžná technika pro statistickou analýzu dat. Dokáže nalézt vzory, které bychom sami nebyli schopni určit. Nejznámějším algoritmem je K-means, kde analytik zvolí k – počet tříd(clusterů) a následně náhodně centroidy, které představují střed jedné třídy. Poté vypočítáme euklidovu vzdálenost každého bodu k centroidu a přiřadíme je k nejbližšímu centroidu. V dalším kroku můžeme centroidy upravit a pokračovat znovu. Kroky tohoto algoritmu jsou znázorněný na obrázku 3.1. Můžeme si to představit na určování druhu květin. V našem datasetu budeme mít několik druhů květin s označením počtu okvětních lístků a jejich velikostí. Ačkoliv se každá jednotlivá květina bude odlišovat, jeden druh by měl znázorňovat jednu třídu. Tudíž na grafu bychom uviděli určité skupiny bodů a každá skupina by měla být jeden druh květiny. V bezpečnosti se clustering používá ve forenzní analýze nebo behaviorální analytice. Díky němu dokážeme klasifikovat všechny aktivity a nalézt anomálie.

- Asociace – Forma unsupervised learningu, která nám dovolí objevovat vztahy mezi jednotlivými daty v obrovských datasetech. Používá se například v e-shopech pro nabízení doporučených produktů. Například pokud si kupujete chléb, je větší šance, že si k němu koupíte máslo nebo marmeládu. Algoritmy pro asociaci jsou Apriori, ECLAT a FP-Growth.

Deep learning

Deep learning je podmnožinou machine learningu a odkazuje na učení neuronové sítě s více než jednou vrstvou mezi vstupem a výstupem. Neuronová síť vznikne složením perceptronů, což je základní prvek celé sítě. Ty jsou navzájem propojeny a předávají si signály. V dnešní době se pro něj používá spíše označení neuron, jakožto inspirace struktury lidské nervové soustavy. Perceptronem rozumíme matematický model biologického neuronu. Deep learning se učí najít struktury v obrovských datasetech pomocí algoritmu Backpropagation, který indikuje změnu interních parametrů stroje, které se používají k výpočtu reprezentace v každé vrstvě z předchozí vrstvy [27]. Snaží se napodobit činnost lidského mozku při zpracovávání dat, vytváření vzorů a rozhodování. Běžně se používá v překladačích, rozpoznávání řeči, autonomních vozidlech a generování textu či obrazu.



Obrázek 3.1: **K-Means algoritmus.** Obrázek popisuje kroky algoritmu K-Means. Převzato z [21].

Reinforcement learning

Reinforcement learning je metoda strojového učení, fungující na způsobu pokus-omyl a systému odměn. Od supervised learningu se liší v tom, že nemá žádné trénovací data s korektními výstupy, ale na základě zadaného úkolu se rozhoduje, jaké akce podnikne. Stroj se snaží dělat takové akce, které mu přinesou co nejvíce odměn a co nejméně penalizací. Což je svým způsobem velice obtížné, ukážeme si to na příkladu s hrou Pong. V této 2D hře se proti sobě utkávají dva hráči a na způsob tenisu se snaží, aby protihráč neodehrál míček a dostal gól. Ovládání je pouze nahoru a dolů. Pokud bychom místo jednoho hráče nasadili stroj s reinforcement learningem a nechali ho učit se, tak by si při úspěšné hře zapamatoval sekvenci pohybů a na konci zaznamenal odměnu. Pokud by gól dostal, zaznamenal by penalizaci. Byla by ale celá sekvence pohybů nahoru-dolů špatně při obdrženém gólu? Pravděpodobně ne a tak se dostáváme k pojmu `Reward Shaping`, který nemusí dávat odměnu za celou sekvenci akcí, ale třeba za část. Takže při obdrženém gólu bychom označili sekvenci všech pohybů, až na poslední, jako úspěšnou. Nicméně implementace `Reward Shaping` je velice složitá a často vede k `The Alignment Problem`, kde se stroj naučí nějakou sekvenci akcí, která mu generuje velké množství odměn, ale nedělá to, co my chceme.

Reinforcement learning se používá k řešení takových problémů, kde je sekvenční rozhodování[1] a dlouhodobý cíl. Typickým příkladem je hraní her, práce s roboty nebo logistika.

3.2 Pyramida potřeb

Pokud chceme vytvořit kvalitní model strojového učení, či použít umělou inteligenci a nahradit, či zautomatizovat nějaký proces, musíme si uvědomit pár základních věcí. První z těchto bodů je zjistit, jaké je stávající řešení. Není užitečné vytvářet model pro aktivitu,

[1] Jedinec zvažuje dané možnosti jednu po druhé a nemá možnost se k předchozím možnostem vrátit.

která zabere hodinu práce týdně, měly by to být práce, které jsou nutné dělat často, opakovaně a s určitou přesností. Je nutné přesně specifikovat a určit si, které výstupy budou považovány za správné a špatné. Toto nás vede k toleranci chyb. Pokud budeme mít systém, který je používán ve zdravotnictví pro vyšetřování rakoviny, bude pro nás lepší, pokud se 100% přesností nás informuje o třech nemocných, ale vynechá dalších pět, nebo upozorní na deset podezřelých pacientů, u kterých se rakovina potvrdí pouze ve dvou případech? Proto je nutné si určit hranici pravděpodobnosti, kdy se už případ vyhodnotí, nebo ne.

Pokud si uvědomíme a vezmeme v potaz výše uvedené informace, můžeme se přesunout k hierarchii základních potřeb:

- Sběr dat – Na začátku celého procesu je nejdřív potřeba získat správná data pro strojové učení. Jejich důležitost je popsána o podkapitolu níže. Pro určité potřeby jsou vhodná různá data a pokud bychom nepracovali se správným datasetem, mohl by náš model produkovat nepřesné výstupy. Taková data můžeme získat z různých senzorů, logovacích zpráv, uživatelem generovaná data a mnoho dalších.

- Uložení dat – Pro správné zacházení a přístup k datům je důležité je mít správně uložené. Může se jednat o strukturovanou databázi, která nám umožní rychle a spolehlivě přistoupit k určitým datům.

- Příprava dat – Po splnění předchozích dvou bodů je možné začít připravovat data. To zahrnuje čištění dat. Pod tímto pojmem se myslí manuální procházení dat, během kterého rozhodneme, která data jsou vhodná nebo jestli nám nechybí velká část dat. Tento krok je velmi důležitý pro celý proces. Konkrétní příklady jsou uvedeny níže.

- Agregace/označování – V tomto kroku si definujeme metriky, na základě kterých budeme měřit citlivost modelu na určité faktory. Dále označíme testovací data, na kterých budeme model učit a optimalizovat.

- Učení/Optimalizace – V posledním kroku máme připravená data pro experimenty a optimalizaci modelu. Začne se jednoduchými algoritmy, které mohou objevit určité rysy, které mají vliv na výsledek [31].

Důležitost dat

Data jsou základem pro strojové účení. Na kvalitě vstupních dat záleží kvalita výstupu z algoritmu strojového učení. Musí být dodány v takové formě, aby jim daný algoritmus rozuměl, jinak náš model může produkovat nekvalitní výsledky, což může způsobit nemalé potíže. Hlavní funkcí algoritmu strojového učení je získat důležité informace z těchto dat [16].

Čištění dat a normalizace

Strojové učení je o trénování a vkládání dat do algoritmů k provádění různých úkolů. Složité ale je vyčistit vstupní data od nerelevantních a náchylných k chybám. Tento krok je snad z celého procesu nejdelší. Nejběžnější praktiky jsou doplňování chybějících hodnot, odstraňování řádků či snižování objemu dat. Pokud bychom tento krok ignorovali, tak by náš model, ačkoli by byl správný, mohl generovat špatné výstupy, jelikož jsme mu neposkytli správná data. Například budeme chtít vytvořit již zmíněný model na odhadování hodnoty

domu. Jeden dům by měl jako lokaci vyplněnou adresu, druhý zeměpisné souřadnice, rozloha by byla uvedena u jednoho v metrech čtverečních, jinde ve stopách. U rozpoznávání obrázků to může být špatné rozlišení, kombinace barevných a černobílých obrázků a další.

Feature selection a extraction

Feature selection je proces vybrání důležitých prvků z datasetu. Jak už bylo řečeno výše, data jsou pro model velmi důležitá a my musíme označit a poskytnout správné prvky. Například bychom chtěli vytvořit model na odhadování, zda-li člověk dokáže uběhnout maraton. V popisu člověka bychom mohli použít výšku, váhu, věk a jak často sportuje, ale informace, že má dvě ruce a dvě nohy, nebo že rád poslouchá hudbu, nám moc nepomůže a byly by nadbytečné v tomto případě.

Feature extraction je proces redukování prvků z existujícího datasetů do menších, lépe zpracovatelných skupin. Na rozdíl od feature selection, které vybírá existující prvky, feature extraction je transformuje. Zmenšuje počet dat, které musí být zpracovány, ale pořád přesně reprezentují původní dataset.

3.3 Strojové učení v bezpečnosti

Množství a rozsah možných kybernetických hrozeb nutí organizace, aby neustále monitorovaly obrovské množství datových bodů ve své infrastruktuře. To není jednoduše proveditelné, obzvláště, pokud jsou omezené zdroje, zejména ty lidské. V této záležitosti může pomoct strojové učení, které dokáže rozeznat vzory a předpovídat hrozby ve velkých zdrojích dat. Zautomatizováním analýzy mohou kyberbezpečnostní týmy rychle detekovat možné bezpečnostní události a vykonat určitou protiakci k zamezení škod.

Většina analýz bezpečnostních událostí či incidentů začíná tím, že zjistíme co se stalo a proč se to stalo. Pokud najdeme odpověď na tyto dvě otázky, můžeme se pokusit předpovědět, co bude následovat a rozhodnout se, jaké kroky podnikneme k eliminaci rizik. V dalším případě vyvineme maximální úsilí, aby se daná situace už neopakovala. V těchto situacích většinou značnou roli hraje čas a strojové učení nám dovolí ho získat.

Strojové učení si získalo i svou roli v detekci malware. Při správném trénovacím datasetu je schopné detekovat s nízkou chybovostí hodně druhů malware. Nicméně pro správný model pro detekci a vlastně jakýkoliv model použitý v kyberbezpečnosti, je nutné zajistit několik vlastností.

- Trénovací dataset – Jak bylo zmíněno výše, tréninková fáze je nejdůležitější. Tu zajistíme získáním a vložením správných dat. Musíme mít velké množství dat rozlišných druhů, aby náš model byl komplexní a spolehlivý. Pokud bychom například opomenuli v našem trénovacím datasetu, že každý malware je spustitelný soubor s příponou .exe, náš model by mohl založit detekci na této vlastnosti, ačkoliv by každý soubor s příponou .exe nebyl malware. Tímto by docházelo k velkému počtu nenahlášených malware souborů a falešně pozitivních hlášeních.

- Interpretovatelný model – Model nesmí být černou skříňkou. Pokud bychom seděli jako operátoři za monitorovacím systémem, který by nám zobrazil nějaké upozornění a my neměli jakoukoli šanci zjistit, za jakých podmínek bylo toto upozornění vyhodnoceno, nemáme dostatek šancí na analýzu a porozumění daného problému.

- Přizpůsobení a vývoj – Veškeré útoky a hrozby se neustále vyvíjí a jejich tvůrci se snaží, aby se vyhnuli detekcím. To způsobuje problém, jelikož například nové verze

malwaru se značně liší od těch předchozích. Benigní programy jsou také neustále vyvíjeny a jejich struktura je jiná od těch, které jsme měli v trénovací části, ale model je musí označit jako neškodné. Toto způsobuje degradaci modelu, což znamená, že se náš model zhoršuje v čase a nemá takovou úspěšnost. Průměrně po dvou měsících od vytrénovaní modelu se jeho schopnost detekce sníží o dvacet až třicet procent, a my musíme vhodně daný model trénovat znovu s aktualizovaným datasetem [9].

Stávající řešení

Strojové učení samozřejmě ovlivňuje i bezpečnost a hodně firem se vydává i tímto směrem. Ačkoliv většina nástrojů postavených na strojovém učení nejsou navrhnuty tak, aby nahradily tradiční řešení, mohou být užitečným doplňkem, pokud je jejich použití správně zaměřeno, například klasifikace dat, hledání vzorů či detekce anomálií.

Nejvhodnějším způsobem je použít strojové učení k naučení charakteristik prostředí, které chceme chránit, a charakteristik toho, co je špatné. Tímto můžeme podpořit práci analytiků. Nicméně všechna řešení nejsou ideální a jak bylo zmíněno výše, hodně nástrojů je implementováno jako černá skříňka, což nedovoluje analytikům pochopit důvody vyhodnocení anomálií. Tyto nástroje nám také nedávají stoprocentní jistotu a budou generovat určité množství falešně pozitivních výstupů.

Kapitola 4

LogMan.io

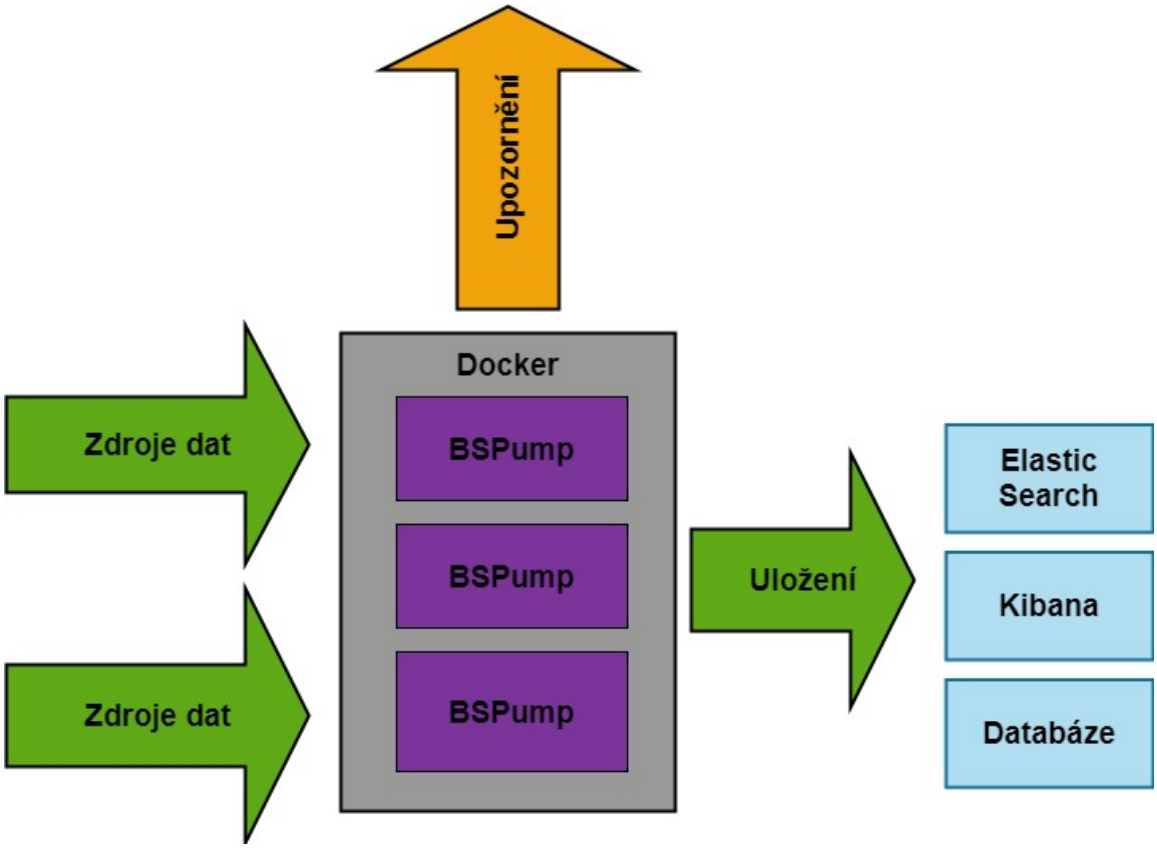
LogMan.io je produkt, který je určený pro real-time proudové zpracování dat a využívá událostmi řízenou architekturu. Jeho hlavními komponenty jsou tzv. real-time Processory, které dokážou zpracovávat data, v našem případě bezpečnostní události, mnoha možnými způsoby. Tyto processory běží paralelně a jsou schopny pracovat se stovky datových proudů z rozdílných zdrojů. Můžeme je využít k detekci anomálií, hledání vzorců v datech a dalších situacích.

Podstatná část zdrojových kódů LogMan.io je zveřejněna pod nerestriktivními open-source licencemi a umožňuje tak pokročilým uživatelům samostatně dotvářet a rozšiřovat své aplikace založené na tomto systému. Navržení architektury LogMan.io je koncipováno tak, aby podporovala vývoj nových funkcí, které lze snadno navázat na existující funkční celky. Tímto řešením získáváme univerzální platformu, která je škálovatelná a jednoduše rozšiřitelná.

LogMan.io je vyvinut v jazyce Python, což dovoluje využít celou škálu různých knihoven a lehce rozšířit celý systém, ať už v oblasti datové analýzy či strojového učení.

4.1 Architektura

LogMan.io se skládá z datových pump, které umožňují získavat data z různých zdrojů a technologií, jako je například Kafka, RabbitMQ či souborů v uložišti. Tyto pumpy jsou schopné data odesílat do určitých systémů, mezi které patří například ElasticSearch, či jiná databáze, nad kterou lze dále tvořit vizualizace.



Obrázek 4.1: **Architektura LogMan.io.**

LogMan.io se skládá z několika klíčových komponent, které utváří celý systém platformy a zároveň mu umožňují další rozšíření.

- BSPump – Jedná se o implementaci proudového analyzéru v jazyce Python, který poskytuje platformu pro vytváření datových pump. Datové pumpy se skládají z pipeline, které určují způsob získávání, zpracování a uložení dat, dále z konektorů poskytující integraci s ostatními systémy a platformami. Datových pump může být spuštěno libovolné množství zároveň, což umožňuje celý systém škálovat.
- ElasticSearch – ElasticSearch se řadí mezi fulltextové bezschémové databáze, které umožňují rychle vyhledávat jednotlivé události. Není třeba definovat přesnou strukturu databáze, protože se vytvoří sama na základě vložených dat. K fulltextovému vyhledávání využívá Apache Lucene, čímž je zajištěno napovídání, automatické doplňování a další usnadnění. ElasticSearch implementuje také REST API, které slouží k vytváření, čtení, editování nebo mazání informací pomocí HTTP požadavků. Ukládá data v podobě JSON dokumentů uložených v rámci indexů [19].
- Kibana – Kibana je vizualizační nástroj s rozhraním, který umožňuje interaktivně vyhledávat data v databázi ElasticSearch. Je implementována jako webová aplikace, čímž je zajištěn jednoduchý přístup z jakéhokoliv webového prohlížeče a systému. Zobrazuje data v reálném čase a uživatel si může snadno nastavit časové okno a filtry dat. Kibana také umožňuje vyhledávat podle klíčových slov, tvorbu vizualizací, časových grafů, tabulek nebo diagramů.

- Docker – Docker kontejnery umožňují izolovat jednotlivé aplikace do vlastního kontejneru v prostředí operačního systému. LogMan.io využívá Docker kontejnery ke snadnému a rychlému nasazení datových pump a dalších komponent. Kontejnery se dají jednoduše hromadně spravovat a využívají se pro distribuci nových verzí systému.

BSPump

BSPump je základní komponentou systému LogMan.io. Jedná se o open-source projekt, který může být použit k vytvoření aplikací pro zpracování dat v reálném čase. BSPump je implementován v jazyce Python.

Architekturu aplikace tvoří určité jednotky. Těmito jednotkami mohou být:

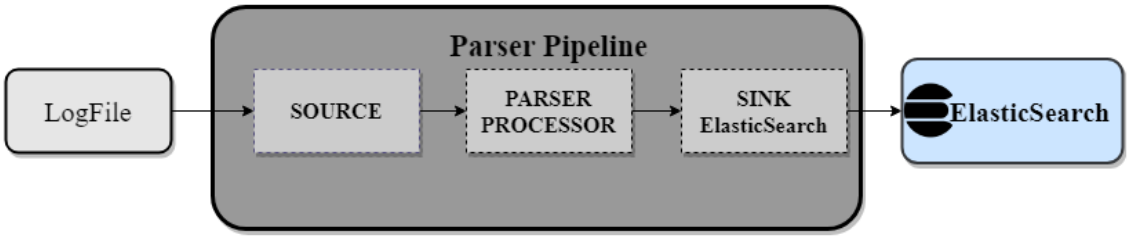
- Connection - Tato jednotka slouží k připojení externího zdroje dat nebo jiné služby v infrastruktuře. Je to základ každé pipeline pro zpracování dat. Příkladem může být konfigurace spojení do Kafky, RabbitMQ, MySQL databáze či ElasticSearche a mnoho dalších.

- Lookup - Lookup se využívá k rychlému vyhledání dat ve slovnících typu klíč-hodnota. Mohou se nacházet a být použity v pipeline objektech. Každý Lookup vyžaduje staticky či dynamicky vytvořený slovník. Využít se může například pro spojení do ElasticSearch a vytáhnutí dat s určitým indexem.

- Pipeline - V BSPump jsou data zpracovávány v takzvaných pipelines. Každá pipeline pracuje samostatně a nezávisle na ostatních pipelines, pokud explicitně nedefinujeme určitou závislost. Data zpracované v jedné pipeline, můžeme předat druhé, například bude existovat pipeline na rozdělení logu, ve které na začátku log přijmeme, rozdělíme ho podle určitých pravidel a než ho uložíme do databáze, tak ho pošleme do jiné pipeline, která má na starost analýzu. Takto ho můžeme zaslat vícero analyzačních pipeline, pokud potřebujeme jeden log použít ve více detekcích.

  Pipeline se může skládat ze tří hlavních komponent. Těmi jsou `Source`, což definuje zdroje dat, na kterých je pipeline připojena. Zdrojem může být již zmíněné spojení do databáze, fronty zpráv, TCP tok či CSV nebo JSON soubor v úložišti a další. Po Source většinou následuje `Processor`, který může například upravovat nebo obohacovat data. Mezi další možnosti patří počítání metrik, vytváření agregací, detekce anomálií a další. Konkrétní typy Processorů budou představeny a rozebrány později. Posledním typem komponentu je `Sink`, což je objekt, který slouží jako cíl v každé pipeline. Příkladem Sinku mohou být stejné typy jako u Source, čili MySQL databáze, ElasticSearch, Kafka, soubory a další.

  Každá pipeline začíná jedním nebo více zdroji a je ukončena právě jedním objektem typu Sink. Například se může jednat o pipeline na rozdělení logovací zprávy z logovacího souboru na serveru. Architektura této pipeline tedy bude taková, že na začátku bude Source s nakonfigurovanou cestou k souboru s logy, následovat bude Processor pro rozdělení logu a na konci pipeline bude Sink do ElasticSearche, nad kterým můžeme dále provádět vizualizace, či logy prohledávat například v Kibaně. Architektura je znázorněna na obrázku 4.2.



Obrázek 4.2: **Příklad architektury parsovací pipeline**

Log Management

Cyklus log managementu v LogMan.io se skládá ze tří fází:

- Sběr logů a dalších událostí
- Proudové zpracování a analýza nasbíraných dat
- Archivace nasbíraných dat

LogMan.io sbírá a zpracovává různé druhy informací o sledovaných systémech. Může se jednat o logovací informace, auditní zprávy, NetFlow a jiné protokoly pro sledování síťového provozu a další strukturované i nestrukturované zdroje informací. Formát pro centralizovaný sběr je definován na vstupu do systému sběru dat a nevyžaduje ustálený popis datové struktury. Sběr dat probíhá pomocí integrace standardních systémů a technologií jako je například Syslog, Windows Event Log, HTTP logy, NetFlow, Common Event Format a další. Není jakkoliv omezen v připojování zdrojů dat a chybějící konektory lze jednoduše doplnit.

Systém obsahuje software pro lokální sběr dat, který je možné využít pro získávání událostí ze systémů, které například nelze integrovat pomocí standardních protokolů a nástrojů. Další metodou je získávání dat ze souborů, které můžeme do systému nahrát pomocí běžných protokolů jako jsou například FTP, FTPS, SCP a další. Data lze do systému dodávat i z relačních databází, jako je MS SQL, Oracle, MySQL a další.

Pro proudové zpracování je vybaven již zmíněnými processory, které můžeme využít k transformaci dat. Dále pro analýzu je možné využít analyzérů, což jsou speciální typy processorů, které nejsou určeny pro změnu či úpravu daného záznamu, pouze pro vykonání analýzy nad ním. Po zpracování a vykonání analýzy log putuje do databáze pro archivaci.

4.2 Návrh řešení

V předešlých částech byly popsány komponenty systému LogMan.io a jeho základní funkce. Tato část bude věnována návrhu aplikace, která bude schopna z určitého logovacího záznamu provádět detekci pomocí modelu strojového učení za využití komponentů LogMan.io. Aplikace by měla být rozšiřitelná a být prototypem pro plně produkční nasazení.

Prvním krokem bude specifikace přijetí logu. Jelikož se v produkčním prostředí můžeme setkat s mnoha typy zdrojů logů, ať už sběr pomocí agenta, ze souboru, z databáze a mnoho dalších, bude nutné zajistit sběr na jedno místo, ze kterého bude možné data dále zpracovávat. Abychom splnili tyto podmínky, můžeme využít frontu zpráv. Po přijetí logu

je nutné zajistit, aby aplikace zaslala log do správného Parseru. Jeho úlohou je převést log ze zdrojové podoby do interního formátu, který je dále používaný a zajistit, že všechny potřebné informace jsou přenesené. Posléze Parser log rozešle dále na analýzu a sám ji archivuje v databázi pro možnou revizi. Analyzérů, které mohou provádět detekci nad jedním logem, může být více, tudíž musíme rozdělený log naklonovat a rozeslat ho. Po analýze se buď log zahodí, nebo se vytvoří upozornění, které je možné dále zpracovat a podniknout další akce.

4.3 Instalace LogMan.io

V této části je popsán postup při nasazení produktu LogMan.io, hardwarové a softwarové požadavky a základní obsluha.

Softwarové požadavky

Produkt LogMan.io vyžaduje prostředí s:

- operačním systémem Linux
- Python verze 3.5 nebo vyšší.

Díky Docker kontejnerům je toto prostředí implicitně zajištěno. Proto je zapotřebí, aby bylo vnější prostředí vybaveno aplikacemi:

- Docker
- Docker compose

Linux

Pro maximální efektivitu v rámci produkčního prostředí a zajištění kompatibility vyžaduje produkt LogMan.io operační systém Linux, nejlépe v distribuci Ubuntu či CentOS. Přehled podporovaných verzí:

- Ubuntu 16 LTS nebo vyšší
- CentOS 7 nebo vyšší
- RedHat Linux 5 nebo vyšší

Hardwarové požadavky

Požadavky na hardware se odvíjejí od množství zpracovávaných dat a jednotlivých připojených datových proudů. Propustnost jednoho CPU jádra se pohybuje mezi tisíci až dvaceti tisíci událostmi za sekundu, konečné číslo hodně záleží na složitosti vstupních dat. Na jedno CPU jádro je potřeba přiřadit 2GB paměti RAM. LogMan.io ukládá primárně data do ElasticSearch databáze, kterou je možné dimenzovat na ukládání desítek až stovek terabajtů dat. Tato vlastnost umožňuje prakticky neomezeně rozšiřovat úložný prostor pro data. Je také možné replikovat uložená data a zajistit odolnost systému proti výpadku jednoho, či více datových uzlů. Pro systémy s extrémními nároky na počet událostí za sekundu, které souvisí s ukládáním či načítáním dat, je doporučeno realizovat diskové úložiště na lokálně připojených SSD discích. Použití síťově připojeného úložiště se doporučuje pro archiv událostí.

Instalace

Celá instalace probíhala na serveru s operačním systémem CentOS 7.

Produkt LogMan.io je dodáván v podobě obrazů Docker kontejnerů, které je zapotřebí stáhnout a spustit z distribuční URL adresy pomocí aplikace Docker compose, ke které je poskytnut příslušný konfigurační soubor. Databázi ElasticSearch je nutné nainstalovat a připravit zvlášť.

Pro instalaci LogMan.io je zapotřebí stáhnout Docker kontejnery s konfiguračním souborem `docker-compose.yml` pro Docker compose, který přesně definuje zdroj jednotlivých kontejnerů, jejich nastavení, jako jsou čísla portů, cestu k datovým adresářům a další. Díky tomuto souboru příkazem `docker-compose up -d` automaticky nasadíme a spustíme kontejnery.

ElasticSearch lze nainstalovat pomocí následujících příkazů. Nejdříve je nutné připravit prostředí před vlastní instalací.

```
yum install java-1.8.0-openjdk-headless
rpm --import https://artifacts.elastic.co/GPG-KEY-elasticsearch
```

Další příkaz spustí instalaci ElasticSearch.

```
yum install elasticsearch
```

Upravte konfiguraci ElasticSearch a Java VM dle specifických požadavků.

```
vi /etc/elasticsearch/elasticsearch.yml
vi /etc/elasticsearch/jvm.options
```

Spusťte ElasticSearch.

```
service elasticsearch start
```

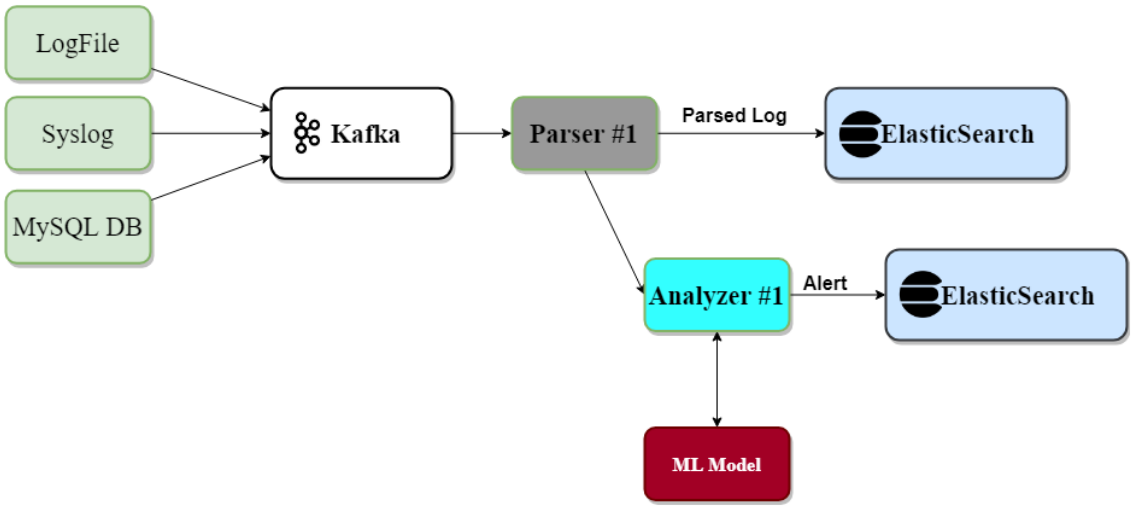
Je doporučené zamknout verzi ElasticSearch, aby nedošlo k automatické aktualizaci.

```
yum install yum-plugin-versionlock
yum versionlock elasticsearch
```

Kibana spolu s Kibana Lookup pluginem je dostupná jako Docker kontejner. Instalace je velmi snadná, uživatel si pouze spustí daný kontejner a nakonfiguruje podle potřeby. Nejdříve se vytvoří soubor `Dockerfile`, kde bude specifikována cesta ke kontejneru. Poté je zapotřebí vytvořit ve stejné složce soubor `kibana.yml`, kde se nakonfiguruje Kibana dle požadavků. Nakonec se vytvoří kontejner pomocí `docker build -t kibana .` příkazu.

4.4 Implementace

V této části je popsána konkrétní implementace prototypu aplikace nad systémem LogMan.io pro zpracování logovacích informací za využití metod strojového učení. Zjednodušená architektura je ilustrována na schématu 4.3, detailní schéma je v příloze B. Pro tento účel byl vytrénován model na detekci nebezpečných a benigních domén či adres. Jeho implementace je popsána v části 4.4. Pro provádění detekce nad vytrénovaným modelem je možné použít například logovací záznamy z DNS serveru, ze kterých se získá dotazovaná doména, nebo za použití sondy na sběr NetFlow by se daly získat informace o překládaných doménách, či celá URL, pokud by se jednalo o HTTP provoz. V tomto řešení byly použity logy z DNS serveru BIND9 verze 9.11.4.



Obrázek 4.3: **Architektura aplikace**

LogMan.io

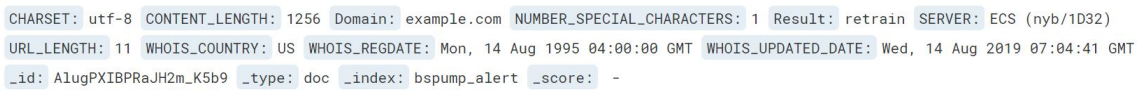
Jak už bylo zmíněno v části 4.2, prvním krokem je specifikovat přijetí logu. LogMan.io umožňuje široké možnosti v napojení externích zdrojů, avšak pro účely prototypu nám postačí vyčíst log ze souboru. Na začátku tedy bude pipeline, které za Source bude mít soubor. Dále log zašle do Enricheru, což je typ Processoru, který má za úkol přidat relevantní informace ke zpracovávané události. Běžně se může jednat o přidání lokace k IP adrese. V tomto případě přidá k logu informaci o jeho typu, díky kterému později určí, do kterého Parseru bude poslán. Po této operaci je zavolán Sink do Kafky do určitého topicu, ze kterého bude později rozeslán do Parseru na rozdělení. Detail tohoto kroku je vyobrazen v příloze B v části `SourceToKafka Pipeline`.

Aby bylo možné poslat logovací zprávu do správného Parseru, je využita informace o typu, které byla doplněna před uložením do Kafky. V praxi se využívá buď tento způsob, nebo systém větvení Parserů, kdy první Parser zkontroluje hlavičku logu, další poté typ aplikace a následně zašle do správné parsovací pipeline. Abychom zajistili rozeslání do určité pipeline, je využit RouterSink, což je speciální typ Sinku, který dokáže na základě poskytnutých informací zaslat událost do korektní pipeline. RouterSink tak díky uvedenému typu logu k němu namapuje cílovou pipeline na rozparsování. V detailním schématu se jedná o část s názvem `Ingest Pipeline`.

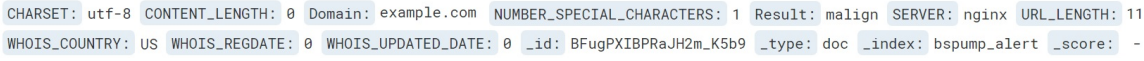
Pro přijetí události z jiné pipeline v rámci jedné aplikace se může použít zdroj InternalSource. Tento zdroj je využit v parsovací pipeline pro přijetí logu. Následně log putuje na rozdělení do určitého strukturovaného schématu. Když je log rozdělen do slovníku typu klíč-hodnota, pokračuje do RouterProcessoru, který má na starost ho naklonovat a zaslat Analyzérům, které provádějí detekci. Na konci parsovací pipeline je odeslán do databáze ElasticSearch. Tato část je na detailním schématu zobrazena jako `NamedLine Pipeline`.

Obrázek 4.4: **Ukázka logu v Kibaně**

Pro přijetí do Analyzéru je využit DirectSource, což je druhý způsob, jak zaslat události z jedné pipeline do jiné. Na rozdíl od InternalSource je o poznání rychlejší, jelikož neobsahuje frontu a není zatížen režií, na druhou stranu ale může způsobit problémy při nevhodném využití. Po přijetí je předán do Analyzéru, který má předat doménu do modelu na detekci. Jeden způsob, jak nasadit vytrénovaný model by bylo zasadit ho přímo do analyzéru. Byl by to jednodušší způsob na implementaci, ale bylo by velmi náročné model přetrénovat, jelikož by se musela restartovat celá aplikace, aby znovu načetla objekty. Druhým způsobem je využít Flask framework[1], díky kterému je možné model nasadit jako aplikaci a komunikovat s ní přes API. Tato aplikace dokáže pružně reagovat na změny, což umožňuje přetrénovat model, aniž by byla hlavní aplikace pozastavena, či přerušena. V tomto případě byl využit druhý způsob, jelikož při tomto typu modelu bude nutné ho často přetrénovat, ať už z důvodu špatného odhadu, či chybějících dat. Flask aplikace si k obdržené doméně sama doplní informace, které používá model k detekci. Všechny doplňované informace jsou popsány v následující části. Pokud detekce proběhla, je zaslán výsledek s informací, zda-li se jedná o nebezpečnou, nebo benigní adresu, který se dále ukládá do databáze ElasticSearch. Pokud detekce neproběhla z důvodu neaktivity adresy, je zaslána chybová hláška, která se nikam dále nepropaguje. Dále se může stát, že se budeme snažit převést data na označení, které nebyly přítomny v trénovacím datasetu. Tato událost je speciálně označena tak, aby byla zahrnuta do trénovacích dat a model byl přetrénován. Analyzér je v detailním schématu zobrazen jako `MBDomain Analyzer`.



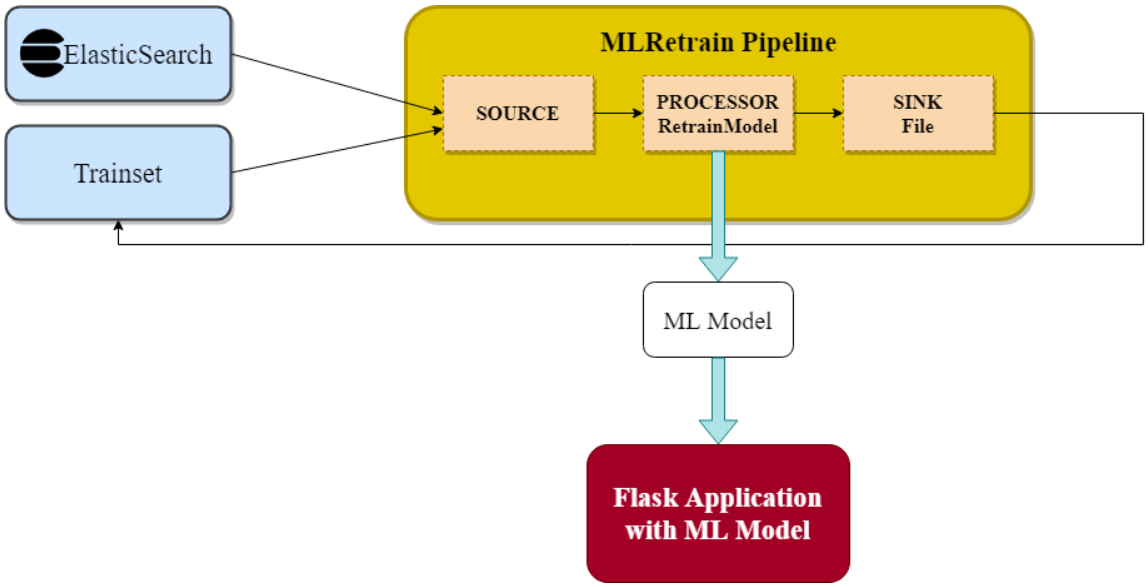
Obrázek 4.5: **Událost k přetrénování v Kibaně**



Obrázek 4.6: **Ukázka alertu v Kibaně**

[1] Více viz. https://flask.palletsprojects.com/

Jako doplnění celého systému je vhodné navrhnout i aplikaci, která bude přetrénovávat model s novými daty. Jelikož se prostředí, ve kterém náš model funguje, může celé změnit, proto je nutné zavést procesy, které budou udržovat model aktuální. Nasazení modelu není jednorázová záležitost, ale kontinuální proces. Navržené schéma této aplikace je na obrázku 4.7. Označená data by se vytáhla z databáze ElasticSearch, k nim by se doplnil původní trénovací dataset a v processoru by proběhlo přetrénování modelu, který by nahradil původní model ve Flask aplikaci. Nakonec by se všechna data uložila do souboru s původním trénovacím datasetem.



Obrázek 4.7: **Navrhované schéma aplikace pro přetrénování modelu.**

Model

V této části je popsán návrh aplikace strojového učení pro detekci škodlivých adres. Aplikace by měla být schopná vytvořit model, podle kterého bude rozdělovat jednotlivé adresy na škodlivé a benigní. Bude otestováno více druhů algoritmů se zaměřením na přesnost detekce.

V praxi se již využívá mnoho způsobů, jak se bránit proti škodlivým adresám. Nejběžnější způsob je na základě blacklistingu, což znamená vytvoření databáze, ve které budou adresy či domény, které byly identifikovány jako škodlivé. Existuje mnoho společností, které poskytují takové seznamy, ale většina z nich nemusí být spolehlivých, jelikož neuvádí informaci, podle které rozhodli, že je daná adresa škodlivá. Takže se jen můžeme domnívat, zda-li proběhla nějaká analýza, nebo ji označili jako nebezpečnou na základě nahlášení od uživatelů, což se běžně děje a hodně volně dostupných seznamů IOCs[2] se zakládají na reportingu, například PhishTank[5] či URL-haus[6]. Na druhou stranu fungují i společnosti, které se nespoléhají na externí zdroje a používají své vlastní algoritmy pro detekci, například Google Safe Browsing[1] či OpenPhish[4].

[2]Indicators of Compromise, může se jednat o IP adresy, MD5 hashe souborů, URL, domény a další.

Příprava dat

Pro vytrénování modelu byl použit volně dostupný dataset označených příkladů na maligní a benigní adresy[34] doplněný o část maligních domén z výše uvedených zdrojů, které nebyly vytvořeny pomocí DGA[3]. Důvod vybrání modelu tohoto typu byl takový, aby bylo možné na základě logu provést detekci.

Z původního datasetu byly pro zjednodušení vybrány určité sloupce, které lze získat pouze na základě dané adresy. Těmi jsou počet znaků v adrese, počet speciálních znaků, znaková sada, verze a typ webového serveru, velikost HTTP hlavičky, dvouznakový kód státu, ve kterém se webový server nachází a informace z registru Whois, konkrétně datum registrace a datum obnovení.

Délka adresy a počet speciálních znaků byl spočítán z poskytnutého řetězce, znaková sada, webový server a velikost HTTP hlavičky byla získána z hlavičky odpovědi na základě GET požadavku. Pro získání informace o geografickém umístění byla použita online databáze IPinfo[2]. Většina takových databází je buď placená, nebo mají omezený počet dotazů za měsíc. IPInfo poskytuje zdarma padesát tisíc dotazů měsíčně, což je i pro malou firmu nedostačující. Řešením by bylo použití offline databáze, například GeoIP[3], což by vyřešilo problém s počtem dotazů, ale na druhou stranu je nutné ji stále obnovovat. Datumy ohledně registrace a obnovení v registru Whois byly získány za pomocí knihovny Python-whois[4], která poskytuje odpovědi ve formě slovníku, takže je jednoduché získat konkrétní data, jelikož klíče jsou vždy stejné. Stejně jako způsob získávání informace o geografickém umístění, i toto řešení má své omezení. Většina poskytovatelů informací z Whois mají buď placenou službu, nebo omezený počet požadavků za měsíc, tudíž v reálném provozu to je nevyhovující a spousta dotazů se vrací bez odpovědi.

Po získání dat je nutné je zpracovat do podoby vhodné pro algoritmus na trénování modelu. Celkový počet záznamů je 1815, z toho 1562 označených jako benigní a 253 jako maligní, což se nejedná o velmi reprezentativní vzorek, jelikož je to málo domén, ale postačující pro demonstrativní účely. Chybějící či neposkytnuté údaje byly nahrazeny nulou. Časové údaje byly převedeny na počet dnů a textové údaje byly označeny číslem pomocí funkce `LabelEncoder`, který označí každý unikátní popisek za číslo. Nevýhoda tohoto přístupu je taková, že při pokusu o transformaci řetězce, který nebyl v trénovacích datech obsažen, se objeví chybová hláška informující o nemožnosti této akce. Proto je nutné ho označit a model přetrénovat.

Trénovací fáze

Po zpracování a přípravě dat je na řadě trénovací fáze. Byly použity tři metody supervised learningu pro vytrénování modelu a porovnání jejich přesnosti. Rozdělení dat pro trénování a testování bylo vždy 80/20.

První testovanou metodou byl Random Forest(RF), což je metoda pro klasifikaci a regresi, která vytvoří více rozhodovacích stromů, které následně spojí pro větší přesnost predikce. Rozhodovací stromy jsou stavebním kamenem pro vytvoření Random Forest modelu. Jejich cílem je vytvořit stromový graf, který na základě jednoduchých pravidel zařadí zkoumaný objekt do určité třídy. Random forest vytvoří jisté množství takových stromů a při predikci každý z nich nějak označí testovaný objekt. Na konci je vybrána ta predikce, která má větší množství označení. Výhodou této metody jsou takové, že dokáže automa-

[3]Domain generation algorithm, útočníci používají tento algoritmus na vygenerování náhodných názvů domén, které používají k malwarovým útokům.

[4]Více viz. https://pypi.org/project/python-whois/

ticky zpracovat chybějící hodnoty a z důvodu použití rozhodovacích stromů není náročný na výpočetní výkon. Nicméně při větším počtu stromů může být komplexnější a zabrat více času při trénování.

Druhou testovanou metodou byl Support Vector Machine(SVM), neboli metoda podpůrných vektorů pro klasifikaci. Při binární klasifikaci si Support Vector Machine nejdřív zobrazí data v N–dimenzionálním prostoru a poté se snaží nalézt nadrovinu, která rozdělí data na dva poloprostory tak, že v jednom budou, v našem případě, benigní adresy a v druhém budou maligní adresy s co největší vzdáleností bodů od roviny.

Poslední použitou metodou byla Logistic Regression. Navzdory svému jménu se nejedná o regresi, ale o klasifikační algoritmus. Nejčastěji se používá k řešení problémů s binární klasifikací tak, že se snaží odhadnout pravděpodobnost jevu na základě jeho charakteristik.

Vyhodnocení metod

Vyhodnocení testovaných metod se zaměřovalo na to, jak model dokáže rozpoznat škodlivé adresy s co nejmenším počtem falešně pozitivních a falešně negativních hlášení. Pro tento účel byla použita matice záměn, což je tabulka, která se běžně používá k popisu přesnosti klasifikačních modelů na sadu testovacích dat, pro které známe hodnotu označení. Pro binární klasifikaci se jedná o tabulku se čtyřmi hodnotami o predikovaných a skutečných výstupech. Predikovanou třídou rozumíme to, jakou hodnotou model daný objekt označil a skutečnou třídou to, do jaké třídy skutečně patří.

- True Positive(TP) - Hodnota, která určuje počet adres, které model správně odhadl jako maligní.
- True Negative(TN) - Hodnota, která určuje počet adres, které model správně odhadl jako benigní.
- False Positive(FP) - Hodnota, která určuje počet adres, které model špatně odhadl jako maligní.
- False Negative(FN) - Hodnota, která určuje počet adres, které model špatně odhadl jako benigní.

Dále jsou použity hodnoty jako přesnost, což je výsledek celkového podílu predikcí, které odhadl správně a které odhadl špatně. Tímto určíme poměr adres, které klasifikujeme správně. Preciznost, která se vypočítá pomocí správně odhadnutých adres k celkovému počtu odhadovaných adres v odhadované třídě. Čím vyšší, tím je náš odhad důvěryhodnější. Dále úplnost, nebo též senzitivita, která určuje podíl správně odhadnutých adres k součtu adres ve skutečné třídě. Poslední hodnotou je F1-score, což je vážený průměr preciznosti a senzitivity.

Random Forest				
Třída adres	Preciznost	Senzitivita	F1-score	Počet adres
Benigní	0.97	0.99	0.98	316
Maligní	0.93	0.78	0.84	47

Tabulka 4.1: Výsledky RF

Support Vector Machine				
Třída adres	Preciznost	Senzitivita	F1-score	Počet adres
Benigní	0.91	1.00	0.95	313
Maligní	1.00	0.38	0.55	50

Tabulka 4.2: Výsledky SVM

Logistic regression				
Třída adres	Preciznost	Senzitivita	F1-score	Počet adres
Benigní	0.91	0.97	0.94	312
Maligní	0.71	0.39	0.51	51

Tabulka 4.3: Výsledky Logistic Regression

Obrázek 4.8: **Matice záměn RF**

Obrázek 4.9: **Matice záměn SVM**

Obrázek 4.10: **Matice záměn Logistic regression**

Celková přesnost u metody Random forest byla 97.52%, u Support Vector Machine 91.46% a u Logistic regression 89.26%. Ačkoliv by ve většině případů byla tato čísla skvělá, u tohoto modelu to není tak určitě, jelikož třída maligních adres není zastoupená v tak velké míře. Z tohoto důvodu je lepší se zaměřit na senzitivitu u třídy maligních adres. V tomto ohledu se zdá být nejlepší použití metody Random Forest, jelikož dokázala ze 47 maligních adres v testovací fázi označit správně 40 jako škodlivé a 7 jako benigní. V rozpoznávání benigních adres byl jen o zanedbatelný kousek za metodou SVM, když správně označil 314 adres a skutečně benigních označil jako maligní pouze 2. Logistická regrese se SVM byly velmi nepřesné v rozpoznávání škodlivých adres s úspěšností pod 40%.

4.5 Vyhodnocení

Byla implementována aplikace pro detekci maligních adres za použití metod strojového učení nad systémem LogMan.io. Tato aplikace využívá komponenty systému LogMan.io a může být prototypem pro plně produkční nasazení. V praxi se některé způsoby budou lišit, například se log nebude číst ze souboru, ale bude zaslán pomocí Syslogu, nebo upozornění na anomálii se nebude propagovat do databáze ElasticSearch, ale do speciální konzole. Nicméně obecný způsob použití zůstane stejný.

Tento konkrétní případ modelu strojového učení má určitá omezení v získávání dat pro predikci z důvodu licencování některých služeb, ať už v přiřazení geografického umístění serveru, na kterém se nacházela doména, či obdržení informací z registru Whois.

Kapitola 5

Závěr

Cílem této práce bylo seznámení se s problematikou zpracování logovacích informací pro účely bezpečnostního monitorování, nastudovat existující možnosti zpracování dat pro použití metod strojového učení a navrhnout prototyp nástroje na detekci za použití metod strojového učení nad systémem LogMan.io. Práce obsahuje teoretickou část, která je v první části zaměřená na uvedení problematiky zpracování logovacích informací pro účely bezpečnostního monitorování. Byly uvedeny základní pojmy a způsoby spojené s problematikou logování a zpracování logů z pohledu technického i právního. V druhé části byly představeny typy strojového učení a jejich využití v bezpečnosti pro detekci anomálií, vzorců v datech a detekci malware.

V praktické části byl představen systém LogMan.io, který je určen pro proudové zpracování dat v reálném čase. Byly představeny jeho komponenty a funkce, ze kterých byla následně navrhnuta a implementována aplikace pro detekci nebezpečných domén z logů DNS serveru BIND9. Tato aplikace k detekci využívala vytrénovaný model strojového učení nasazený jako Flask aplikace, kde komunikuje s jejím API. Bylo otestováno více algoritmů metod učení s učitelem při trénování modelu a byla vybrána ta s nejlepší přesností při detekování maligních domén. Dále bylo navrhnuto schéma aplikace, díky které by se dal celý model přetrénovat a zajistit tak větší úspěšnost při detekci.

Byl představen způsob, jakým se dá využít strojové učení v systému LogMan.io a to tak, že byl navrhnut a implementován nástroj, který využíval metody strojového učení k detekci, čímž byl záměr práce splněn.

Práce mi byla velmi přínosná, jelikož jsem se díky ní seznámil s metodami strojového učení a jeho využití v rámci bezpečnostního monitoringu. V práci bych chtěl pokračovat implementací aplikace pro přetrénování modelu a vytrénování dalších modelů pro jiné případy použití, například k detekci anomálií.

Literatura

[1] *Google Safe Browsing : Blacklist service provided by Google*. Dostupné z: https://safebrowsing.google.com/.

[2] *IPInfo:The Trusted Source for IP Address Data*. Dostupné z: https://ipinfo.io/.

[3] *MaxMind's GeoIP*. Dostupné z: https://dev.maxmind.com/geoip/.

[4] *OpenPhish : Automated self-contained phishing blacklist*. Dostupné z: https://www.openphish.com/.

[5] *PhishTank : Phishing blacklist operated by OpenDNS*. Dostupné z: https://www.phishtank.com/.

[6] *UrlHaus : Malware blacklist operated by abuse.ch*. Dostupné z: https://urlhaus.abuse.ch/.

[7] *Vyhláška č. 82/2018 Sb. o bezpečnostních opatřeních, kybernetických bezpečnostních incidentech, reaktivních opatřeních, náležitostech podání v oblasti kybernetické bezpečnosti a likvidaci dat ze dne 21. května 2018*.

[8] Pseudonymizace. *ManagementMania.com*. duben 2018. Dostupné z: https://managementmania.com/cs/pseudonymizace-pseudonymisation.

[9] *Machine Learning Methods for Malware Detection*. Prosinec 2019. 18 s. Dostupné z: https://media.kaspersky.com/en/enterprise-security/Kaspersky-Lab-Whitepaper-Machine-Learning.pdf.

[10] Bock, J. Comparing Publish-Subscribe Messaging and Message Queuing. leden 2018. Dostupné z: https://dzone.com/articles/comparing-publish-subscribe-messaging-and-message.

[11] Burkov, A. *The Hundred-Page Machine Learning Book*. 2019. ISBN 978-1999579500.

[12] Chuvakin, A., Schmidt, K. J. a Phillips, C. *Logging and log management : the authoritative guide to understanding the concepts surrounding logging and log management*. Waltham, Mass: Syngress, 2013. ISBN 978-1-59749-635-3.

[13] Claise, B. *Cisco Systems NetFlow Services Export Version 9* [RFC 3954]. RFC Editor, říjen 2004. Dostupné z: https://rfc-editor.org/rfc/rfc3954.txt.

[14] Crowdstrike. Adversary Tradecraft and The Importance of Speed. 2019.

[15] Demchenko, Y., Membrey, P., Grosso, P. a Laat, C. Addressing Big Data Issues in Scientific Data Infrastructure. In:. Květen 2013, s. 48–55.

[16] Desik, A. Making the Foundation Strong: The Importance of Data Processing in Machine Learning/Artificial Intelligence. *Tata Consultancy Services*. srpen 2019. Dostupné z: https://www.tcs.com/blogs/making-the-foundation-strong-importance-of-data-processing-in-machine-learning.

[17] Gerhards, R. *The Syslog Protocol* [RFC 5424]. RFC Editor, březen 2009. Dostupné z: https://rfc-editor.org/rfc/rfc5424.txt.

[18] Grace, K., Salvatier, J., Dafoe, A., Zhang, B. a Evans, O. Viewpoint: When Will AI Exceed Human Performance? Evidence from AI Experts. *Journal of Artificial Intelligence Research*. Červenec 2018, roč. 62, s. 729–754.

[19] Guiliano, S. Elasticsearch Distributed NoSQL Database – What Is It and Should You Use It? *Atlantic.Net Blog*. únor 2016. Dostupné z: https://www.atlantic.net/hipaa-compliant-database-hosting/elasticsearch-distributed-nosql-database/.

[20] Hruška, T. *BIG DATA - VELKÁ DATA*. Dostupné z: https://www.fit.vutbr.cz/study/courses/WAP/private/podklady/Prednasky/PIS/PIS3060LAP.pdf.

[21] Hudson, R. *Big climate data analytics: Effective knowledge-discovery from Colombia's weather data*. Disertační práce.

[22] Janečková, E. *GDPR : praktická příručka implementace*. Praha: Wolters Kluwer, 2018. ISBN 978-80-7552-248-1.

[23] Kent, K. a Souppaya, M. P. *SP 800-92. Guide to Computer Security Log Management*. Gaithersburg, MD, USA, 2006.

[24] Kosť, D. Hierarchie potřeb v kyberbezpečnosti – 2. část. *IT SECURITY Network News*. září 2019. Dostupné z: https://www.itsec-nn.com/hierarchie-potreb-v-kyberbezpecnosti-2-cast/.

[25] Kosť, D. Hierarchie potřeb v kyberbezpečnosti – 4. část. *IT SECURITY Network News*. listopad 2019. Dostupné z: https://www.itsec-nn.com/hierarchie-potreb-v-kyberbezpecnosti-4-cast/.

[26] Kosť, D. Hierarchie potřeb v kyberbezpečnosti – 5. část. *IT SECURITY Network News*. leden 2020. Dostupné z: https://www.itsec-nn.com/hierarchie-potreb-v-kyberbezpecnosti-5-cast/.

[27] LeCun, Y., Bengio, Y. a Hinton, G. Deep Learning. *Nature*. Květen 2015, roč. 521, s. 436–44.

[28] Narkhede, N. First Apache release for Kafka is out! *LinkedIn Engineering*. září 2019. Dostupné z: https://engineering.linkedin.com/kafka/first-apache-release-kafka-out.

[29] Pattynová, J., Suchánková, L. a Černý, J. *Obecné nařízení o ochraně osobních údajů (GDPR)*. Praha: Leges, 2018. ISBN 978-80-7502-288-2.

[30] Pollock, R. What Do We Mean By Small Data. *Open Knowledge Foundation Blog*. duben 2013. Dostupné z: https://blog.okfn.org/2013/04/26/what-do-we-mean-by-small-data/.

[31] Rogati, M. The AI Hierarchy of Needs. *Hacker Noon*. srpen 2019. Dostupné z: https://hackernoon.com/the-ai-hierarchy-of-needs-18f111fcc007.

[32] Russinovich, M. a Garnier, T. *Sysmon*. Prosinec 2019. Dostupné z: https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon.

[33] Samuel, A. L. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*. 2000, roč. 44, 1.2, s. 206–226.

[34] Urcuqui, C. *Malicious and Benign Websites: Classify by application and network features*. 2017. Dostupné z: https://www.kaggle.com/xwolf12/malicious-and-benign-websites.

Příloha A

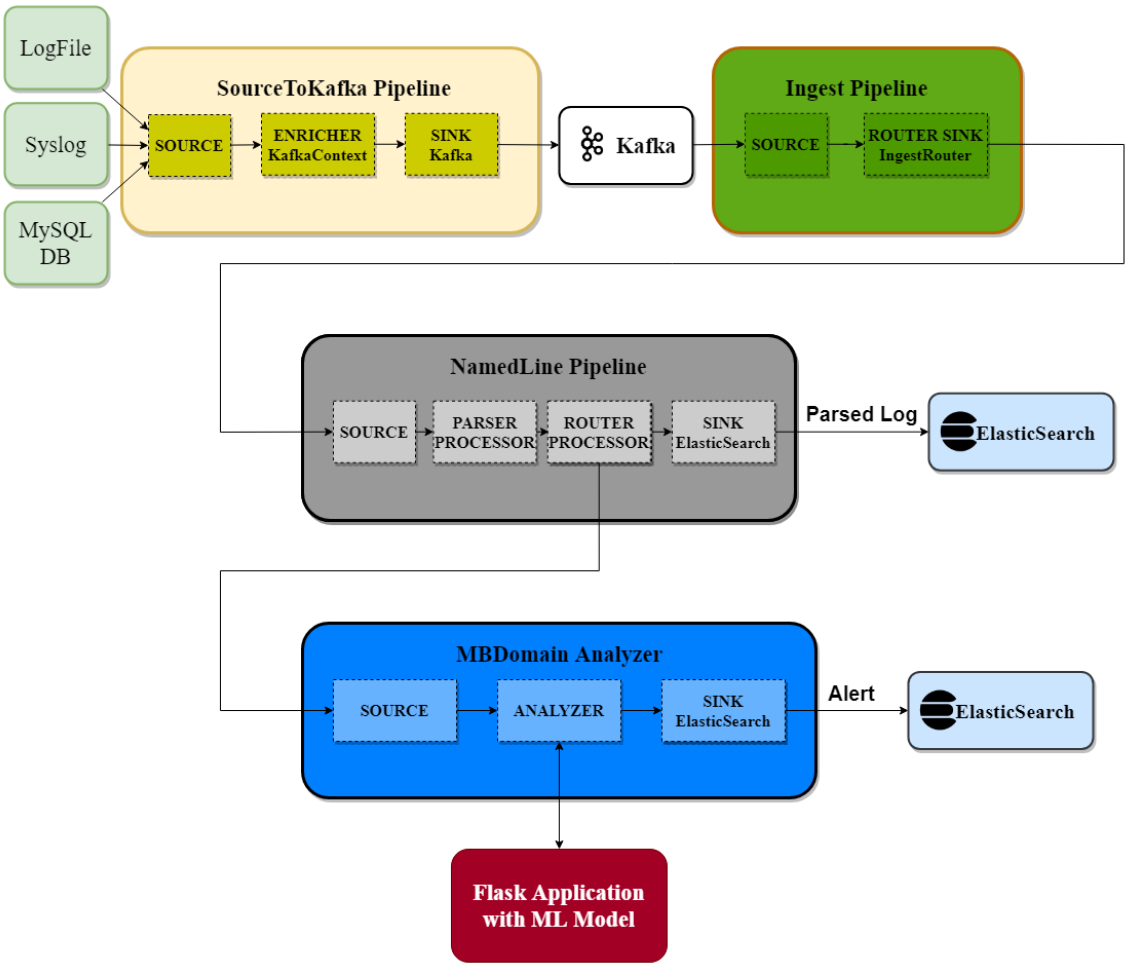
Obsah přiloženého paměťového média

K této práci je přiloženo CD s doplňujícími zdroji v elektronické podobě. Jeho obsahem je:

- Elektronická verze textu diplomové práce ve formátu PDF se zdrojovým kódem zprávy.
- Zdrojový kód aplikace pro vytrénování modelu pro detekci maligních adres.
- Zdrojové kódy aplikace pro detekci za použití metod strojového učení nad systém LogMan.io.
- Zdrojový kód aplikace pro nasazení modelu pro predikce.

Příloha B

Detailní schéma aplikace



Obrázek B.1: **Detailní schéma aplikace**