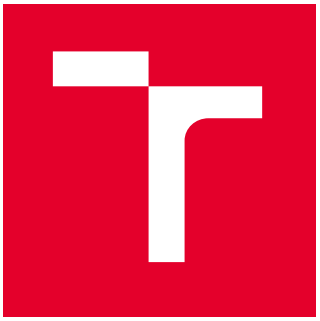


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA PODNIKATELSKÁ

FACULTY OF BUSINESS AND MANAGEMENT

ÚSTAV INFORMATIKY

INSTITUTE OF INFORMATICS

METODIKA A IMPLEMENTACE NÁSTROJŮ PRO ŘÍZENÍ KYBERNETICKÝCH INCIDENTŮ

METHODOLOGY AND IMPLEMENTATION OF CYBER INCIDENT MANAGEMENT TOOLS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Jakub Sedlák**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Petr Sedlák**
SUPERVISOR

BRNO 2025

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA PODNIKATELSKÁ

Zadání diplomové práce

Ústav:	Ústav informatiky
Student:	**Bc. Jakub Sedlák**
Vedoucí práce:	**Ing. Petr Sedlák**
Akademický rok:	2024/25
Studijní program:	Informační management

Garant studijního programu Vám v souladu se zákonem č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů a se Studijním a zkušebním řádem VUT v Brně zadává diplomovou práci s názvem:

Metodika a implementace nástrojů pro řízení kybernetických incidentů

Charakteristika problematiky úkolu:

Cíle práce
Teoretický úvod
Popis současného stavu
Návrh řešení
Ekonomické zhodnocení
Závěr

Cíle, kterých má být dosaženo:

Cílem práce je návrh datové architektury systému a její integrace s nástrojem Splunk.

Základní literární prameny:

DOUCEK Petr, Martin KONEČNÝ a Luděk NOVÁK, Řízení kybernetické bezpečnosti a bezpečnosti informací, Praha: Professional Publishing, 2020. ISBN 978-80-88260-39-4.

SEDLÁK Petr, Martin KONEČNÝ, Přeměna ISMS v manažerské informatice. Brno: CERM, Akademické nakladatelství, 2023. ISBN 978-80-7623-110-8.

SEDLÁK Petr, Martin KONEČNÝ a kolektiv, Kybernetická (ne)bezpečnost. CERM, Akademické nakladatelství, 2021. ISBN 978-80-7623-068-2.

SEDLÁK Petr, Martin KONEČNÝ a kolektiv, Případové studie řízení kybernetické bezpečnosti. CERM, Akademické nakladatelství, 2024. ISBN 978-80-7623-126-9.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2024/25

V Brně dne 9.2.2025

L. S.

doc. Ing. Miloš Koch, CSc. garant	prof. Ing. et Ing. Stanislav Škapa, Ph.D. děkan

Abstrakt

Tato diplomová práce se zabývá návrhem systému pro zpracování a agregaci síťových záznamů s cílem minimalizovat jejich objem při zachování vysoké informační hodnoty pro potřeby threat huntingu a forenzní analýzy. První část práce se zaměřuje na teoretická východiska, konkrétně na technologie používané v návrhové části. Druhá část práce popisuje výchozí stav existujícího řešení a definuje problémy spojené s objemem a redundancí síťových záznamů. Třetí část podrobně popisuje návrh a implementaci nového systému v kontejnerizovaném prostředí Podman, a to včetně jednotlivých komponent pro příjem, transport, normalizaci a agregaci dat pomocí nástroje Apache Spark a Apache Kafka. V závěru jsou shrnuty přínosy navrhovaného řešení z provozního i ekonomického hlediska.

Klíčová slova

síťová bezpečnost, detekce hrozeb, zpracování dat, Splunk, Apache Spark, Apache Kafka

Abstract

This master thesis focuses on the design of a system for processing and aggregation of network records in order to minimize their volume while maintaining high information value for threat hunting and forensic analysis. The first part of the thesis focuses on the theoretical background, specifically the technologies used in the design part. The second part of the thesis describes the initial state of the current solution and defines the problems associated with the volume and redundancy of network records. The third part details the design and implementation of the new system in the containerized Podman environment, including the individual components for data ingestion, transport, normalization, and aggregation using Apache Spark and Apache Kafka. It concludes with a summary of the benefits of the proposed solution from an operational and economic perspective.

Keywords

network security, threat detection, data processing, Splunk, Apache Spark, Apache Kafka

Bibliografická citace

SEDLÁK, Jakub. *Metodika a implementace nástrojů pro řízení kybernetických incidentů* [online]. Brno, 2025 [cit. 2025-05-17]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/167949. Diplomová práce. Vysoké učení technické v Brně, Fakulta podnikatelská, Ústav informatiky. Vedoucí práce Ing. Petr Sedlák.

Čestné prohlášení

Prohlašuji, že předložená diplomová práce je původní a zpracoval jsem ji samostatně. Prohlašuji, že citace použitých pramenů je úplná, že jsem ve své práci neporušil autorská práva (ve smyslu zákona č. 121/2000 Sb., o právu autorském a o právech souvisejících s právem autorským).

V Brně dne 17. 5. 2025

Bc. Jakub Sedlák

autor

Poděkování

Rád bych poděkoval svému vedoucímu práce panu Ing. Petru Sedlákovi za poskytnuté rady při zpracování této diplomové práce. Dále bych chtěl poděkovat všem kolegům z oddělení analýzy síťového provozu na NÚKIB, kteří mi vždy věnovali svůj čas, trpělivost a pomohli s řešením dané problematiky. Na závěr chci poděkovat své rodině a blízkým, kteří mi v průběhu celého studia byli motivací, a především velkou oporou.

OBSAH

ÚVOD

Monitoring síťového provozu a schopnost detekovat kybernetické bezpečnostní události představuje jeden z klíčových požadavků kladených na regulované subjekty podle zákona č. 181/2014 Sb., o kybernetické bezpečnosti. V praxi však implementace těchto opatření naráží na technická a provozní omezení. Častým problémem je práce s velkými objemy dat, která vznikají nepřetržitým sledováním síťového provozu.

Nástroje pro analýzu těchto dat bývají licencovány na základě množství indexovaných událostí, což omezuje možnosti jejich nasazení bez předchozího zpracování dat. Efektivita systému tak vychází ze schopnosti data filtrovat, transformovat a agregovat bez ztráty jejich informační hodnoty.

Tato diplomová práce se proto zaměřuje na návrh a realizaci detekčního systému, který zajišťuje efektivní předzpracování síťových toků. Řešení využívá distribuované proudové zpracování pomocí aplikací Apache Kafka a Apache Spark, které jsou provozovány v kontejnerizovaném prostředí Podman. Výsledná architektura by měla umožňovat normalizaci a agregaci, s cílem minimalizovat objem dat indexovaných do analytického nástroje Splunk. Systém je určen pro nasazení v prostředí Národního úřadu pro kybernetickou a informační bezpečnost a jeho partnerské organizace, které podléhají zákonu o kybernetické bezpečnosti.

CÍLE PRÁCE, METODY A POSTUPY ZPRACOVÁNÍ

Hlavním cílem této diplomové práce je návrh a implementace efektivního řešení pro agregaci síťových toků za účelem snížení objemu dat indexovaných v analytickém nástroji Splunk. Návrh je realizován v rámci Národního úřadu pro kybernetickou a informační bezpečnost jako součást systému detekce, určeného pro analýzu síťové komunikace u regulovaných subjektů.

Celý návrh navazuje na předchozí funkční systém detekce, který však musel být pozastaven s ohledem na původní licenci Splunku, která umožňovala 500 GB denní limit indexovaných dat. S přechodem na výrazně omezenější 100 GB licenci bylo nutné systém kompletně přepracovat. Způsob zpracovávání datových toků byl navržen tak, aby výsledný objem bylo možno indexovat do této nižší licence a zároveň nedošlo ke ztrátě informační hodnoty dat.

Základ řešení je kontejnerizovaná architektura, která propojuje nástroje pro sběr dat IPFIXcol2, streamovací platformu Apache Kafka a výpočetní aplikaci Apache Spark. Síťová data jsou převedena na binární formát Apache Avro, dále jsou agregována v časových oknech a následně připravena pro indexaci do Splunku ve formě optimalizovaných záznamů.

Samotný návrh řešení se také zaměří na dílčí klíčové cíle, které zahrnují návrh agregační logiky, která bude minimalizovat redundanci dat. Důraz byl kladen na to, aby struktura výstupních dat odpovídala konkrétním analytickým potřebám a detekčním scénářům. Pro analytiky bylo důležité, aby systém umožňoval proaktivní i retrospektivní vyhledávání pomocí dotazovacího jazyka SPL v prostředí Splunk.

Použitá metodika vychází z principů proudového zpracování dat v reálném čase. Návrh se zaměřoval primárně na požadavky na modularitu, a tedy možnost přidání různých typů vstupních dat. Architektura systému také umožňuje přesnou kontrolu nad každou fází datového toku od příjmu dat až po jejich indexaci. Poskytuje tak analytikům stabilní základnu pro rozšiřování systému na základě měnících se požadavků na analyzovaná data.

1 TEORETICKÁ VÝCHODISKA PRÁCE

Tato kapitola se zaměřuje na základní pojmy, nástroje a technologie, které tvoří základ návrhu systému detekce. Nejprve jsou zde popsány technologie NetFlow/IPFIX a IPFIXcol2, které slouží ke sběru síťových toků. Poté následuje představení pojmu Cyber threat hunting a nástroje typu SIEM, včetně jejich přínosů. Další část se věnuje analytickému nástroji Splunk a aplikacím Apache Spark a Apache Kafka. Je zde také popsána kontejnerizace prostřednictvím platformy Podman. Kapitola je uzavřena přehledem používaných nástrojů pro monitoring systému.

1.1 Technologie a sondy Flowmon

V prostředí moderních počítačových sítí je vyžadováno efektivní monitorování síťového provozu. Jedním z nejrozšířenějších přístupů k této problematice je využití tzv. flow-based monitoringu neboli analýzy síťových toků. Tento koncept popisuje komunikaci mezi dvěma koncovými body bez nutnosti uchovávání záznamů o jednotlivých paketech. Důležitými nástroji v této problematice jsou protokoly NetFlow a jeho standardizovaný nástupce IPFIX.

1.1.1 Netflow, IPFIX

Technologie NetFlow byla původně vyvinuta společností Cisco Systems jako proprietární metoda pro sběr informací o síťových tocích. Netflow je obvykle definován pěti základními parametry:

- IP adresa zdroje (Source IP Address)
- IP adresa cíle (Destination IP Address)
- Číslo portu zdroje (Source Port)
- Číslo portu cíle (Destination Port)
- Přenosový protokol (Transport Protocol, např. TCP/UDP) [9][14]

Jelikož se technologie rychle vyvíjela, bylo ji časem potřebné také standardizovat. V návaznosti na tuto potřebu vznikl v rámci IETF otevřený standard IPFIX (IP Flow Information Export), který je specifikován v dokumentaci RFC 7011. IPFIX zachovává architekturu NetFlow verze 9 a zároveň ji rozšiřuje. Klíčovým rozšířením je možnost definovat si vlastní informační prvky, čímž IPFIX zvyšuje své možnosti využití.[14]

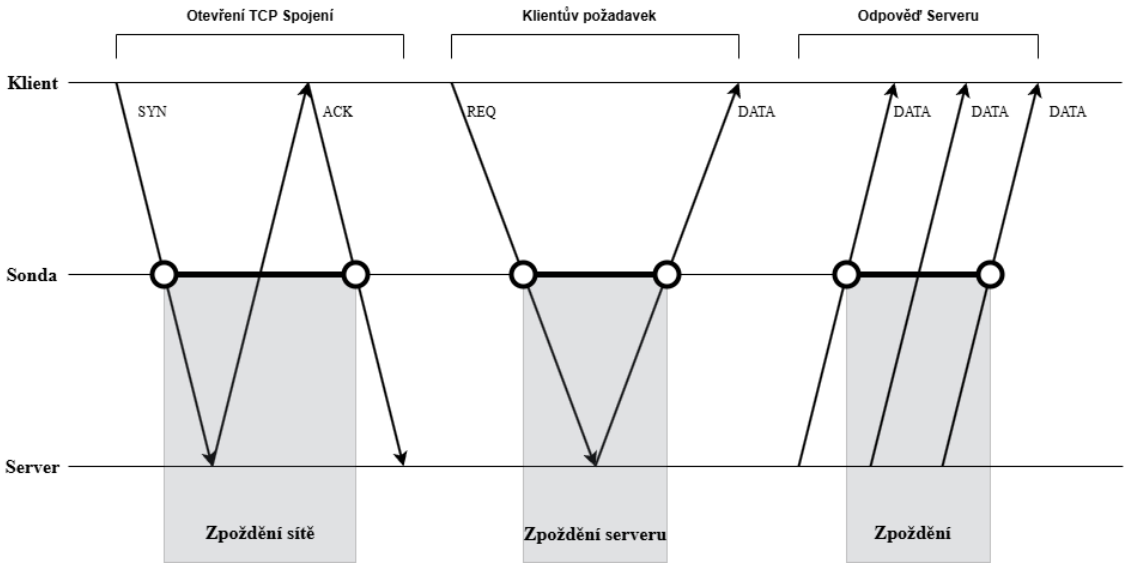
Standard IPFIX se obecně skládá ze tří základních komponent:

- **Exportér,** který shromažďuje a odesílá data o síťových tocích v předem definované struktuře
- **Kolektor**, jehož cílem je příjem a následné zpracování dat z exportéru
- **Analyzér**, který zde může být chápán jako aplikace pro analýzu síťových toků [14]

1.1.2 Flowmon Sondy

Jako exportér se v praxi často využívá specializovaných sond, mezi které patří i komerční Flowmon sonda. Ta představuje zařízení určené k pasivnímu monitorování síťového provozu. Síťový provoz může být zachycen z libovolného síťového segmentu pomocí TAP nebo SPAN portu a nedochází zde k zásahu do provozu samotného. Sondy primárně shromažďují informace vrstev L2-L4, jako jsou komunikující IP adresy, protokoly, doby zpoždění paketu a další. Využitím Flowmon IPFIX rozšíření je dále možné exportovat data z vyšších vrstev modelu ISO/OSI. Lze tak získat i hodnoty SNI z TLS handshake, hostname z HTTP komunikace, DNS dotazy a odpovědi a mnoho dalších. Tímto způsobem lze sondy nasadit nejen pro běžný provozní monitoring, ale také pro pokročilou detekci hrozeb.[25]

Sondy Flowmon jsou dostupné ve fyzické podobě, jako virtuální stroje nebo jako cloudové instance v prostředích typu AWS nebo Azure. To umožňuje jejich flexibilní nasazení v různých typech sítí.[25]



Obrázek č. 1: Diagnostika síťového výkonu pomocí Flowmon sondy
(Zdroj: Vlastní zpracování dle [25])

1.1.3 IPFIXcol2

Pro příjem a zpracování IPFIX záznamů je vhodný open-source kolektor IPFIXcol2. Tento kolektor byl vyvinutý sdružením CESNET. Jedná se o zájmové sdružení právnických osob, které poskytuje informační a komunikační služby pro vědu, výzkum a vzdělání v České republice. Velký důraz je u kolektoru kladen na modularitu. Architektura IPFIXcol2 je postavena na konceptu modulární pipeline, což umožňuje sestavit si vlastní řetězec pluginů podle specifických potřeb. I když je IPFIXcol2 optimalizován pro zpracování IPFIX a NetFlow v9, může být pomocí pluginů rozšířen o další datové formáty a přenosové protokoly. Je zde i možnost vývoje vlastního pluginu, a to s pomocí otevřeného API a kvalitní projektové dokumentace. [6][7]

1.2 Cyber threat hunting

Threat hunting nebo také aktivní vyhledávání hrozeb je proaktivní přístup k řízení kybernetických incidentů, jehož cílem je odhalovat skryté kybernetické hrozby nebo útočníky, kteří již pronikli do infrastruktury organizace a unikají tradičním bezpečnostním opatřením, jako jsou antivirové systémy, firewally či systémy pro detekci a prevenci narušení (*IDS/IPS*). Tato metodika zahrnuje manuální i automatizované techniky, které se zaměřují na vyhledávání indikátorů kompromitace (dále jen IoC), neobvyklých vzorců chování nebo jiných signatur ukazujících na potenciální kybernetický útok. [12][34]

1.2.1 Definice a význam threat hunting

Aktivní vyhledávání hrozeb je definováno jako metodický, iterativní proces, kdy bezpečnostní experti proaktivně hledají IoC uvnitř organizační infrastruktury. Narozdíl od klasických reaktivních opatření (např. incident response) se threat hunting snaží útoky odhalit dříve, než útočník způsobí závažnější škody. Využití aktivního vyhledávání hrozeb přispívá k identifikaci kybernetických útoků, které běžné detekční mechanismy často nedokážou odhalit, a významně snižuje čas mezi infiltrací útočníka a jeho detekcí.[4][12][34]

Threat hunting je v současném prostředí kybernetických hrozeb nejen důležitou metodikou, ale také strategickou součástí řízení kybernetické bezpečnosti. Jeho význam se netýká pouze interních bezpečnostních týmů organizací, ale i jejich zákazníků či koncových uživatelů. Význam threat huntingu lze tedy chápat z více úhlů pohledu,

přičemž mezi ty nejzásadnější patří jeho **schopnost preventivně odhalit skryté hrozby a minimalizovat dopady kybernetických útoků,** což s sebou nese řadu přínosů pro zákazníky i organizace samotné.[12][34]

1.2.2 Proces Threat huntingu

Formulace hypotéz

Formulace hypotéz je počáteční fází threat huntingu, která spočívá ve vytvoření jasně definovaných předpokladů o tom, jaké typy hrozeb by mohly v organizaci existovat, jakým způsobem by mohly působit a jaké stopy by mohly zanechat v datech. Hypotézy mohou vycházet z:

- informací o aktuálních kybernetických hrozbách a útocích (např. analýza reportů threat intelligence),
- znalostí o specifickém prostředí organizace (použité technologie, zranitelnosti, typické vzory chování),
- výsledků předchozích bezpečnostních auditů a incidentů.

Hypotézy určují další postup a zaměření analytických kroků. Jde o základ, na kterém stojí celý proces aktivního vyhledávání hrozeb.[12][24]

Sběr dat

Ve fázi sběru dat dochází k systematickému shromažďování relevantních dat z různých bezpečnostních nástrojů a systémů. Jedná se zejména o**:**

- síťová data (NetFlow, packet capture, metadata),
- data z logovacích systémů (SIEM nástroje jako Splunk),
- data o aktivitě koncových bodů, které shromažďují EDR (Endpoint Detection and Response) systémy
- data z aplikačních systémů a serverů

V praxi se často používají platformy jako Apache Kafka, které umožňují agregovat a streamovat velké objemy dat v reálném čase, a pak například Apache Spark pro následnou analýzu a detekci anomálií v těchto datech. [12][24]

Analýza dat a identifikace anomálií

Analytická fáze představuje jádro threat huntingu, během níž jsou využívány pokročilé metody pro odhalování podezřelých vzorců a indikátorů kompromitace (*IoC*). Využívají se zejména techniky jako:

- metody strojového učení
- techniky hlubokého učení (*Deep Learning*)
- analýza vzorců chování

Klíčovým prvkem této fáze je schopnost nástrojů (např. Apache Spark nebo platformy Splunk) zpracovávat velké množství heterogenních dat v reálném čase, což umožňuje včasnou identifikaci sofistikovaných a dříve neznámých hrozeb. [12][24]

Potvrzení a reakce na incident

Pokud jsou během analýzy nalezeny anomálie, podezřelé aktivity nebo indikátory kompromitace, následuje fáze potvrzení incidentu. Zde dochází k detailnímu přezkoumání podezřelé aktivity a jejímu ověření s cílem vyloučit falešně pozitivní nálezy (*false positives*).

V případě potvrzení incidentu následuje řízená reakce, která zahrnuje:

- izolaci postižených systémů
- eliminaci hrozby a zastavení útoku
- obnovu zasažených systémů
- analýzu dopadu incidentu
- informování relevantních zainteresovaných stran (management, bezpečnostní týmy) [10][24]

Dokumentace a zpětná vazba

Každá fáze threat huntingu je dokumentována. Dokumentace incidentů a analytických postupů poskytuje klíčové informace pro budoucí iterace procesu. Získaná zpětná vazba umožňuje neustálé zlepšování metodik, rozšiřování znalostí bezpečnostních týmů a aktualizaci používaných technologií.[4][24]

Tato fáze rovněž slouží jako základ pro splnění regulatorních požadavků jako je nová směrnice Evropské unie NIS2 a poskytuje transparentní podklady pro audity a reporting bezpečnostních incidentů.

Aktualizace bezpečnostních opatření a tvorba nových hypotéz

Poslední fáze cyklu spočívá v aplikaci získaných poznatků k vylepšení bezpečnostních opatření a generování nových hypotéz pro další iteraci threat huntingu. Tento proces je kontinuální a iterativní, což umožňuje pružně reagovat na nové hrozby a adaptovat bezpečnostní strategii organizace dle aktuálního vývoje v oblasti kybernetických hrozeb.[12][24]

1.2.3 Přínosy z pohledu zákazníka

Zákazník, který si threat hunting objedná jako službu či jako součást bezpečnostního outsourcingu, očekává zejména výrazné zvýšení schopnosti organizace detekovat a předcházet útokům dříve, než dojde k jejich eskalaci. Největším přínosem threat huntingu je právě zkrácení doby mezi infiltrací útočníka a jeho odhalením. Tato vlastnost je kritická zejména u útoků typu Advanced Persistent Threat (dále jen APT), kdy útočník může skrytě operovat v infrastruktuře po dlouhou dobu bez odhalení tradičními bezpečnostními nástroji. Zákazník proto získává větší jistotu, že i sofistikované hrozby, které běžné bezpečnostní systémy nezachytí, budou odhaleny dříve, než způsobí významné škody či ztrátu dat. [10][12][32]

Dalším podstatným přínosem je **minimalizace finančních a reputačních škod způsobených kybernetickými incidenty.** Organizace s implementovaným aktivním vyhledáváním hrozeb obvykle vykazují výrazně nižší celkové náklady spojené s řešením bezpečnostních incidentů a kybernetických útoků, a to díky kratší době reakce a rychlejšímu odstranění hrozeb z prostředí. Zákazník tak z pohledu managementu a akcionářů získává nejen lepší kontrolu nad svými finančními riziky, ale také ochranu hodnoty své značky a důvěryhodnosti. [10][32]

Dalšími přínosy threat huntingu pro zákazníka mohou být následující faktory.

Proaktivní přístup namísto reaktivního

Threat hunting představuje zásadní posun od tradičního reaktivního bezpečnostního modelu směrem k proaktivnímu. Tento přístup umožňuje nejen odhalit již existující kompromitace, ale také identifikovat bezpečnostní slabiny a chování, která by mohla být v budoucnosti zneužita. To poskytuje zákazníkům možnost efektivněji řídit svou bezpečnostní strategii a kontinuálně zlepšovat své procesy řízení rizik. [12][24][32]

Zlepšení souladu a regulatorních požadavků

V mnoha odvětvích jsou dnes organizace pod silným regulatorním tlakem (např. GDPR v Evropě, NIS2 směrnice). Aktivní vyhledávání hrozeb pomáhá organizacím nejen splňovat, ale i překračovat požadavky regulací a auditů, jelikož pravidelná dokumentace a reporting při threat huntingu jsou cenným zdrojem dat pro prokazování souladu s předpisy.[32]

Zvýšená důvěra mezi organizací a jejími zákazníky

Organizace, které otevřeně prezentují svou schopnost aktivně vyhledávat a odhalovat kybernetické hrozby, mohou snáze budovat důvěru u svých zákazníků. Transparentnost a schopnost demonstrovat pokročilou úroveň bezpečnosti je významnou konkurenční výhodou, zejména v sektorech s vysokou citlivostí na ochranu dat (např. bankovnictví, zdravotnictví, státní sektor).[32]

Podpora efektivního rozhodování o investicích

Pravidelný threat hunting generuje kvalitní a strukturované informace o aktuálním stavu kybernetické bezpečnosti organizace. Zákazníci mohou díky tomu lépe odůvodnit a směrovat investice do bezpečnostních technologií a lidských zdrojů, neboť výstupy z threat huntingu poskytují přesná data o tom, které oblasti vyžadují zvýšenou pozornost.[10][12]

1.2.4 Současné trendy

V současnosti lze identifikovat několik významných trendů, které formují očekávání a vnímání zákazníků vůči službám v oblasti aktivního vyhledávání hrozeb. Organizace kladou stále větší důraz na proaktivní zabezpečení, automatizaci a integraci pokročilých technologií, které by posílily schopnost organizací chránit své informační prostředí před sofistikovanými kybernetickými hrozbami.[10]

Zvýšená poptávka po automatizaci a integraci strojového učení

Jedním z nejvýznamnějších současných trendů je rostoucí zájem organizací o implementaci automatizovaných procesů v rámci threat huntingu. Tento trend vychází z potřeby rychlé a efektivní reakce na útoky a incidenty, což lidský faktor často nedokáže v dostatečné míře pokrýt. Stále větší důraz je proto kladen na využití strojového učení a umělé inteligence, které umožňují nejen efektivně analyzovat velké množství dat v reálném čase, ale také významně redukovat chybovost a zrychlit dobu odezvy

bezpečnostních týmů. Zákazníci tak častokrát očekávají, že poskytovatelé služeb nabídnou řešení, která jsou schopna kontinuálně vyhodnocovat bezpečnostní data s minimální nutností manuální intervence.[10]

Preference kontinuálního threat huntingu (*Continuous Threat Hunting*)

Dalším výrazným trendem vnímání zákazníků je posun od periodického k nepřetržitému aktivnímu vyhledávání hrozeb. Zákazníci si uvědomují, že pokročilé hrozby typu Advanced Persistent Threat vyžadují nepřetržitou ostražitost, a proto stále častěji upřednostňují řešení poskytující kontinuální monitoring a analýzu bezpečnostního stavu organizace. Kontinuální threat hunting umožňuje průběžně aktualizovat bezpečnostní hypotézy, rychle adaptovat analytické scénáře a okamžitě reagovat na jakékoli podezřelé chování, což výrazně snižuje riziko dlouhodobých infiltrací a jejich následných škod. [10][11]

Rostoucí zájem o modely spolupráce (*Collaborative Threat Hunting*)

Stále více zákazníků očekává spolupráci mezi různými subjekty v oblasti threat huntingu, například mezi dodavateli bezpečnostních řešení, bezpečnostními týmy organizace a bezpečnostní komunitou. Tato kolaborace často zahrnuje sdílení aktuálních indikátorů kompromitace (IoC), technik útočníků a metod odhalování nových hrozeb. Vznikají tak modely, kdy organizace propojují své bezpečnostní týmy s externími specialisty či komunitami (např. MITRE ATT&CK Framework), které jim poskytují aktuální informace a rozšiřují schopnosti efektivního threat huntingu. Tento trend odráží snahu zákazníků získat co nejkomplexnější pohled na aktuální hrozby, zvýšit transparentnost bezpečnostních operací a maximalizovat efektivitu investic do bezpečnosti. [10][11]

Důraz na vizualizaci dat a reporting

Dnešní zákazníci rovněž vyžadují pokročilou vizualizaci výsledků threat huntingu a detailní reporting. V kontextu rostoucího množství bezpečnostních dat zákazníci oceňují jasné, přehledné a srozumitelné vizuální výstupy, které umožňují snadnější interpretaci bezpečnostních incidentů, rizik a trendů v reálném čase.[12] Moderní nástroje jako Splunk, Elasticsearch či Kibana proto hrají klíčovou roli v oblasti threat huntingu, neboť umožňují přehlednou a efektivní prezentaci dat nejen technickým týmům, ale také manažerům a dalším netechnickým rolím uvnitř organizace. [10][12]

1.3 Security Information and Event Management (SIEM)

Security Information and Event Management (dále jen SIEM) představuje systém, který integruje správu bezpečnostních informací (*SIM*) a správu bezpečnostních událostí (*SEM*). Základní funkcí SIEM systémů je sběr, centralizace, normalizace a analýza bezpečnostních dat pocházejících z různých zdrojů, jako jsou systémové logy, síťové provozy, aplikační události nebo data z bezpečnostních nástrojů. Hlavním cílem těchto systémů je detekovat potenciální bezpečnostní incidenty, umožnit rychlou reakci na ně a poskytovat prostředí pro detailní vyšetřování již proběhlých kybernetických útoků. [30][13]

Systémy SIEM jsou využívány zejména k identifikaci bezpečnostních hrozeb na základě automatizovaných pravidel a analytických technik korelace událostí. Kromě standardních analytických metod využívají také pokročilé techniky založené na strojovém učení a detekci anomálií, což jim umožňuje detekovat také neznámé typy kybernetických útoků nebo složité cílené kampaně útočníků. [13][30][34]

1.3.1 Historický vývoj konceptu SIEM

Koncept SIEM se vyvinul na začátku 21. století jako odpověď na stále rostoucí složitost a objem logů generovaných různými informačními systémy. Původní oddělené přístupy SIM, který se orientoval zejména na dlouhodobou archivaci, správu a reportování bezpečnostních dat, a SEM, zaměřený na detekci incidentů v reálném čase, se postupně sjednotily do jednoho celku, označovaného právě jako SIEM. Spojení těchto dvou přístupů umožnilo bezpečnostním týmům efektivnější a rychlejší detekci i reakci na potenciální incidenty. [13][18][21]

1.3.2 Funkce systémů SIEM

SIEM systémy se vyznačují řadou funkcionalit a vlastností, jejichž vzájemné propojení umožňuje komplexní přístup k bezpečnostnímu monitoringu v organizacích. Jedním z klíčových úkolů těchto systémů je agregace a centralizace bezpečnostních dat. SIEM umožňuje shromažďovat data z různých typů zdrojů, jako jsou například firewally, servery, síťová zařízení či antivirové programy, a tato data následně ukládat do jednoho centrálního úložiště v normalizované a strukturované formě. Díky této centralizaci lze provádět efektivní analýzu a následnou korelaci jednotlivých událostí. [13][29][30]

Proces korelace je další zásadní vlastností SIEM systémů. Korelační techniky umožňují identifikovat vzájemné souvislosti mezi událostmi, které vypadají, že s žádnými dalšími událostmi nesouvisí. SIEM tak pomáhá odhalovat složité incidenty nebo útoky, které by mohly být jinak přehlédnuty při monitorování jednotlivých bezpečnostních zařízení. V poslední době je korelace často doplňována pokročilými metodami detekce anomálií s využitím strojového učení. Díky integraci technik analýzy chování uživatelů a entit (*UEBA*) mohou moderní SIEM systémy identifikovat nejen známé vzory útoků, ale i nové, dosud neobjevené hrozby na základě detekce neobvyklého chování uživatelů, zařízení nebo síťového provozu. [13][18][34]

Kromě těchto analytických funkcí plní SIEM také významnou úlohu v oblasti reportingu a již mnohokrát zmíněné správě souladu s předpisy (*tzv. compliance management*). Systémy umožňují generování detailních bezpečnostních zpráv a reportů, které jsou důležité nejen pro vnitřní potřeby organizace, ale také pro splnění požadavků legislativních norem a standardů jako GDPR nebo ISO 27001. Organizace tak mohou díky SIEM systémům efektivně dokumentovat svůj bezpečnostní stav a poskytnout transparentnost vůči regulatorním autoritám či auditorům. [13]

Další rozšířenou schopností moderních SIEM systémů je podpora automatizace reakce na bezpečnostní incidenty a integrace s dalšími bezpečnostními nástroji. SIEM může být propojen se systémy typu SOAR (*Security Orchestration, Automation and Response*), které umožňují částečně nebo úplně automatizovat reakce na specifické bezpečnostní události. Tyto reakce mohou zahrnovat například izolaci napadených zařízení, blokování podezřelých síťových připojení nebo notifikaci bezpečnostního týmu. Tato automatizace zkracuje dobu odezvy a minimalizuje dopady bezpečnostních incidentů na organizaci.

Všechny výše uvedené vlastnosti SIEM systémů tak společně tvoří robustní řešení, které pomáhá organizacím řídit kybernetickou bezpečnost a proaktivně reagovat na aktuální i budoucí bezpečnostní hrozby. [13][30][33][34]

1.3.3 Scénáře použití SIEM systémů v praxi

V praxi nalézají SIEM systémy široké uplatnění napříč různými odvětvími. Typické scénáře jejich využití se nejčastěji týkají proaktivní detekce a reakce na kybernetické bezpečnostní hrozby. Jsou ale užitečné i pro poskytování podkladů pro forenzní analýzu již proběhlých incidentů.[13][18]

Jedním z běžných scénářů použití SIEM je detekce narušení sítě. SIEM systémy analyzují data získaná ze síťových zařízení, firewallů, IDS/IPS systémů a aplikačních logů, a pomocí korelace těchto údajů identifikují podezřelou aktivitu, která naznačuje pokus o průnik nebo neoprávněný přístup. Typickým příkladem může být detekce pokusů o prolomení přihlašovacích údajů za pomoci hrubé síly *(„brute-force attack")* při přihlašování uživatelů. SIEM v tomto případě automaticky identifikuje nadměrný počet neúspěšných pokusů o přihlášení, spojí tyto události s dalšími podezřelými indikátory a upozorní bezpečnostní tým na potenciální útok. [13][18][34]

Dalším častým scénářem je využití SIEM pro **detekci APT útočníků**. Útočníci v případě APT používají dlouhodobě skryté techniky, jako je laterální pohyb v síti nebo exfiltrace citlivých dat. SIEM systémy v tomto případě využívají pokročilých analytických technik a strojového učení k identifikaci vzorců a anomálií. Může se jednat například o neobvyklý síťový provoz mezi interními systémy nebo podezřelé časy přístupů uživatelů ke kritickým systémům nebo komponentám. Tímto způsobem mohou bezpečnostní týmy včasně identifikovat útoky, které by jinak mohly dlouhodobě unikat pozornosti. [10][34]

Významnou oblastí je také použití SIEM pro **monitoring aktivit privilegovaných uživatelů**. Tento scénář zahrnuje sledování přístupů administrátorů nebo jiných privilegovaných účtů k citlivým systémům a datům. SIEM zde dokáže identifikovat neobvyklé aktivity, jako přístup mimo pracovní dobu, pokusy o eskalaci oprávnění, nebo provedení citlivých operací na klíčových systémech bez řádného povolení. Monitoring privilegovaných účtů tak pomáhá předcházet internímu zneužití přístupů nebo kompromitaci externími útočníky, kteří získali administrátorské přihlašovací údaje. [13][34]

SIEM systémy rovněž často slouží jako podpora při **vyšetřování bezpečnostních incidentů**. V případě, že dojde k bezpečnostnímu incidentu, SIEM poskytuje centralizovaný přehled historických dat, logů a síťových aktivit, což umožňuje bezpečnostním analytikům rychle identifikovat původ a rozsah útoku, zrekonstruovat jeho časovou osu a určit, které části infrastruktury byly ovlivněny. Tato schopnost nejen zkracuje dobu nutnou k reakci, ale také usnadňuje shromažďování důkazů pro následné forenzní vyšetřování a případnou komunikaci s regulatorními orgány. [13][18][34]

Tyto scénáře ukazují význam SIEM systémů v rámci moderní kybernetické bezpečnosti. V dnešní době **SIEM platformy představují nezbytný prvek bezpečnostní infrastruktury**, který umožňuje komplexní řízení bezpečnosti v organizacích.

1.3.4 Technologické výzvy a omezení

I přes své rozsáhlé možnosti a výhody se systémy typu SIEM potýkají s několika významnými technologickými výzvami a omezeními. Tyto problémy se nejčastěji týkají složitosti nasazení a konfigurace, kapacity efektivního zpracování velkého množství dat, omezené flexibility některých řešení, ale také náročnosti na lidské zdroje a provozní náklady.

Jednou z hlavních technologických výzev SIEM systémů je jejich **složitá implementace a počáteční konfigurace**. Správná integrace SIEM řešení do stávající IT infrastruktury vyžaduje značné úsilí při nastavení zdrojů dat, normalizaci a vytvoření vhodných upozorňovacích pravidel. Nesprávně nakonfigurovaná pravidla mohou vést buď k nadměrnému množství falešných poplachů (tzv. *false positives*), nebo naopak k přehlížení skutečných bezpečnostních incidentů (*false negatives*), což výrazně komplikuje činnost bezpečnostních týmů. [13][18]

Další zásadní technologickou výzvou je **rozšiřitelnost SIEM systémů**. Vzhledem k neustálému nárůstu objemu bezpečnostních dat, která je potřeba zpracovat, se organizace často potýkají s problémy rozšíření svých SIEM řešení. To se týká zejména schopnosti systému udržovat vysokou rychlost zpracování dat v reálném čase, aniž by docházelo k významným prodlevám nebo ztrátám důležitých bezpečnostních informací. U velkých a distribuovaných prostředí je nutné pečlivě plánovat architekturu řešení, jelikož nesprávně rozšířené prostředí může vést k omezenému výkonu a snížené hodnotě analýz. [13][18]

S rostoucí komplexností kybernetických útoků představuje další významnou technologickou překážku také **schopnost efektivní analýzy pokročilých a dosud neznámých hrozeb**. I když řada moderních SIEM systémů již obsahuje nástroje využívající strojové učení a UEBA, efektivita těchto nástrojů stále značně závisí na kvalitě vstupních dat, správné konfiguraci učících algoritmů a schopnosti bezpečnostních týmů správně interpretovat jejich výstupy. Některé sofistikované typy útoků mohou stále

unikat detekci SIEM, zejména pokud útočník využívá technik, které napodobují legitimní chování uživatelů nebo zařízení. [10][13]

Dalším relevantním omezením jsou **vysoké nároky SIEM systémů na lidské zdroje a provozní náklady**. SIEM vyžaduje kontinuální údržbu, aktualizaci korelačních pravidel, monitorování systému a neustálou validaci upozornění. Tento proces představuje značnou zátěž pro personální kapacity bezpečnostních týmů, které musí být nejen kvalifikované, ale také dostatečně početné pro efektivní práci s velkými objemy bezpečnostních dat a vyhodnocování vznikajících upozornění. **Pro menší nebo středně velké organizace tak náklady na pořízení a provoz SIEM systému bývají často obtížně obhajitelné.** [13][18]

1.3.5 Budoucí trendy

Oblast SIEM systémů je neustále se rozvíjejícím segmentem kybernetické bezpečnosti, jehož budoucí vývojové trendy jsou ovlivněny zejména potřebou efektivní detekce stále sofistikovanějších kybernetických hrozeb, rostoucím zájmem o cloudová řešení a integrací umělé inteligence a automatizace. Současné průmyslové analýzy identifikují několik zásadních směrů, které budou v příštích letech určovat vývoj a implementaci SIEM systémů. [21]

Jedním z klíčových trendů je zvýšená integrace SIEM s pokročilými technikami umělé inteligence a strojového učení. S využitím hlubokého učení a pokročilé behaviorální analytiky budou SIEM systémy schopné efektivněji detekovat složité, dosud neznámé útoky, které tradiční detekční techniky nemusejí včas zachytit. Tento trend zahrnuje zejména rozšíření již zmiňované UEBA, kde je cílem vytvořit přesnější modely chování uživatelů a systémů, a tak rychleji detekovat anomálie představující kybernetickou hrozbu. [10][13][21][30]

Dalším významným vývojovým směrem je posilování integrace SIEM s cloudovými technologiemi. Stále více organizací přechází na hybridní či plně cloudová IT prostředí, což přináší nové výzvy a požadavky na SIEM platformy. Výsledkem tohoto trendu je vzestup tzv. SIEM-as-a-Service, kde je systém poskytován formou cloudové služby, což umožňuje snížení počátečních investičních nákladů, rychlejší nasazení, vyšší flexibilitu a snadnou škálovatelnost řešení. Současně však tento trend klade nové požadavky na

zabezpečení dat, ochranu soukromí a nutnost řešit legislativní aspekty spojené s cloudovým ukládáním citlivých informací. [11][13][21][30]

Významnou oblastí, která bude dále ovlivňovat vývoj SIEM systémů, je stále větší automatizace bezpečnostních operací a integrace se systémy typu SOAR. Automatizace umožňuje bezpečnostním týmům rychleji reagovat na detekované hrozby, minimalizovat škody způsobené útoky a snižovat zátěž analytiků. Budoucí generace SIEM systémů se tak zaměří na užší integraci s nástroji automatizované odezvy, což nejspíš povede k vytvoření uceleného systému schopného automatické nebo alespoň poloautomatické reakce na bezpečnostní incidenty. [13][30][34]

V neposlední řadě je budoucí vývoj SIEM řešení ovlivněn tlakem na zvyšování schopností systémů si vzájemně poskytovat služby. Důraz je také stále více kladen na otevřenost systémů, tedy schopnost SIEM systémů efektivně spolupracovat s dalšími bezpečnostními technologiemi bez ohledu na konkrétního dodavatele. Otevřenost systémů umožní organizacím pružněji reagovat na měnící se kybernetické hrozby a snadněji integrovat nové technologie a zdroje dat, což omezí problém závislosti na jednom dodavateli. [13][21]

Vzhledem k těmto trendům je zřejmé, že SIEM systémy se budou i nadále významně rozvíjet, přičemž jejich budoucí podoba bude silně záviset na schopnosti efektivní integrace pokročilých analytických technologií, cloudových služeb, a především na automatizovaných reakcích.

1.4 Splunk

1.4.1 Historie nástroje Splunk

Splunk byl založen roku 2003 v San Franciscu třemi technologickými experty, Michaelem Baumem, Erikem Swanem a Robem Dasem. Jejich primárním cílem bylo vytvořit nástroj umožňující uživatelům efektivně analyzovat data produkovaná různorodými systémy, přičemž se zaměřovali na strojová data, která chtěli transformovat na jednoduše dostupné a prakticky využitelné informace. Už od svého počátku kladl Splunk důraz na jednoduchost vyhledávání a analýzy dat pomocí vyhledávacího jazyka SPL, který umožnil organizacím zrychlit řešení bezpečnostních incidentů. V roce 2009 uvedla společnost Splunk na trh významnou verzi Splunk Enterprise 4.0, která přinesla zásadní vylepšení uživatelského rozhraní a možnosti pokročilé vizualizace dat. V dalších

letech Splunk pokračoval v inovacích, zejména integrací pokročilých analytických funkcí jako behaviorální analýzy uživatelů, pokročilou automatizací reakce na bezpečnostní incidenty (SOAR) a rozšiřováním nabídky cloudových služeb Splunk Cloud Platform. [5][30]

V dnešní době je pro společnost nejdůležitější jejich poskytované SIEM řešení v podobě platformy Splunk Enterprise Security. Momentálně je toto řešení jeden z předních produktů na trhu kybernetické bezpečnosti. V době psaní tohoto textu je nejnovější verzí Splunk Enterprise 9.4.1, která přináší významná vylepšení v podobě zlepšení výkonu uložiště, které je pro organizace pracující s větším objemem dat klíčové.

1.4.2 Princip fungování Splunku

Splunk funguje na principu centralizovaného sběru dat z různých typů datových zdrojů, jejich následné normalizaci, indexaci a rychlé analytické dostupnosti uživatelům. Data ve Splunku jsou indexována formou událostí označených unikátními časovými razítky, což umožňuje jejich rychlou dohledatelnost za předpokladu, že využijeme dotazovacího jazyka SPL. [5]

1.4.3 Komponenty nástroje Splunk

Splunk Enterprise je složen z několika komponent, které jsou spolu vzájemně provázány a každá z nich plní specifické role v procesu zpracování dat.

První z těchto komponent je **Forwarder**. Jedná se o agenta, který jednoduše sbírá data přímo ze zdrojových systémů, kterými mohou být například servery, aplikace, databáze, nebo jakákoliv jiná síťová zařízení produkující záznamy. Tyto záznamy dále forwarder předává na další z komponent Splunku a tím je indexer. Forwarderů existují dva typy, a těmi jsou: [29][31]

- **Universal forwarder**

Jedná se o lehký agent, který sbírá záznamy, ale nejsou u něj požadavky na větší výkon. I když je považován za dobré řešení ve většině případů, lze u něj vidět silné nedostatky například v podobě nekompability s programovacím jazykem Python, což může být v určitých případech překážkou. Dalším nedostatkem tohoto forwarderu je možnost předávat pouze neparsovaná data, což znamená, že pro odesílání dat založených na událostech by univerzální forwarder nebyl kompatibilní a lepší volbou by byl druhý typ forwarderu. [29]

- **Heavy forwarder**

Tento typ forwarderu je taky někdy označován za „běžný forwarder“. Heavy forwarder je schopen data parsovat před jejich předáním. Další výhodou je možnost data směrovat na základě vstupních kritérií, jako jsou typ události nebo zdroj záznamu. V nastavení tohoto typu forwarderu je také možnost nechat data indexovat lokálně. [29]

V následující tabulce je shrnutí a porovnání obou typů na základě vybraných vlastností.

Tabulka č. 1*:* Srovnání funkcí obou typů forwarderů
(Zdroj: Vlastní zpracování dle: [29])

Vlastnosti a schopnosti	Universal forwarder	Heavy forwarder
Typ instance Splunk	Dedikovaný spustitelný soubor	Plná verze Splunk Enterprise s většinou funkcí vypnutých
Nároky na zdroje (paměť, CPU)	Nejmenší	Střední až vysoké (závislé na povolených funkcích)
Podpora jazyku Python	NE	ANO
Potvrzení od indexeru (garance doručení)	Volitelné	Volitelné
Filtrování jednotlivých událostí	NE	ANO
Parsování událostí	NE	ANO
Lokální indexace	NE	ANO
Vyhledávání/upozorňování	NE	Volitelné
Splunk Web	NE	Volitelné

Další komponentou je **Indexer**. Tato instance Splunku má za cíl indexovat data, která jsou mu předána forwarderem. Index je v tomto případě úložiště dat pro Splunk, ten příchozí data transformuje na události a ty potom uloží právě do indexů. [28]

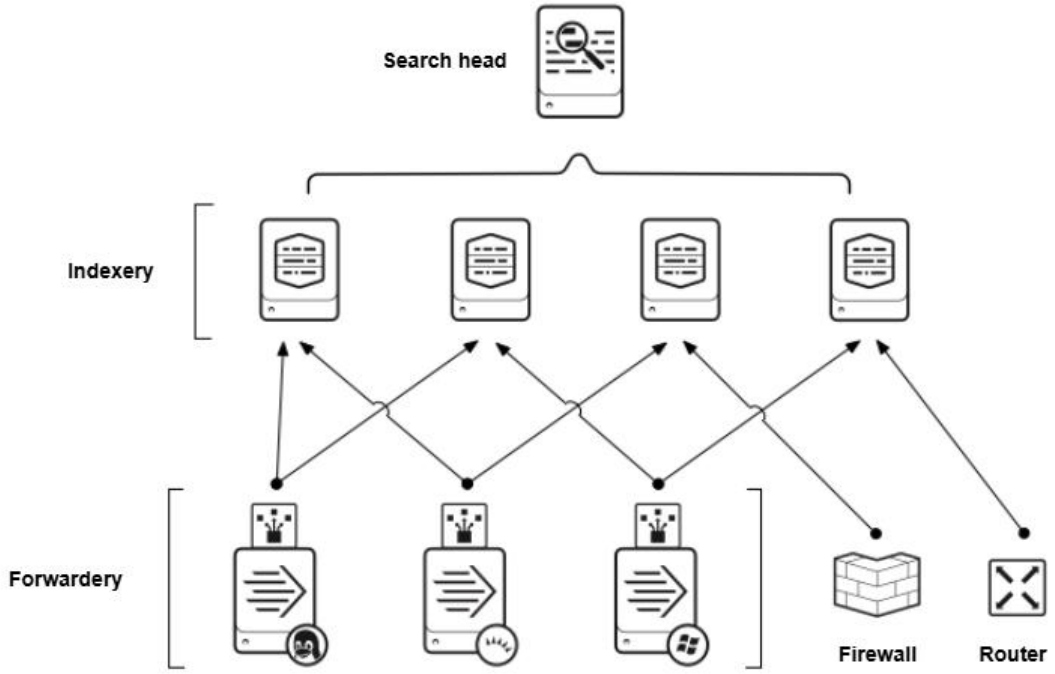
Splunk je v základu dodáván s několika indexy, ale je zde možnost vytvořit další vlastní indexy dle potřeby. Index obsahuje různé typy souborů, které se dělí do dvou hlavních kategorií: [28]

- Surová data v komprimované podobě (*tzv. rawdata*)
- Indexy, které ukazují na surová data a metadata soubory (soubory indexů, označované také jako soubory *tsidx*)

Tyto soubory jsou ukládány do adresářů, které jsou řazeny dle stáří. Adresáře jsou konkrétně nazývány buckets. Pomocí těchto indexů je tak Splunk schopen poskytnout

rychlé a flexibilní vyhledávání dat. Jelikož je vše zpracováno za pomoci tzv. plochých zdrojů (*flat file*), není zde potřeba žádného databázového systému třetí strany, který by musel běžet na pozadí. [28]

V případech, kdy je Splunk nasazen pouze na jednom stroji, tedy pokud se skládá pouze z jedné instance Splunk Enterprise, má indexer za úkol dvě funkce. Nejedná se tedy pouze o indexaci vstupních dat, ale jeho úkolem je také správa vyhledávání nad daty. Je zde ale nutno podotknout, že takový typ nasazení Splunk je vhodný pouze pro potřeby jednoho oddělení v organizaci. Pro potřeby většího rozsahu je indexování odděleno od funkce správy vyhledávání. V takových případech přebírají ostatní komponenty neindexovací role. [28]



Obrázek č. 2: Architektura distribuovaného nasazení Splunku
(Zdroj: Vlastní zpracování dle [28])

Poslední z hlavních komponent je **Search Head**. Tato komponenta představuje pro uživatele hlavní přístupový bod k indexovaným datům. Uživatelé tak skrze search head mohou nad těmito událostmi vyhledávat, analyzovat je nebo také vizualizovat. Search head tak tvoří uživatelské rozhraní a zároveň analytickou vrstvu systému.

Většina analytických operací se v search head provádí prostřednictvím vyhledávacího jazyka SPL (*Search Processing Language*). Search head zpracovává dotazy psané

v tomto jazyce transformuje je do distribuovaných vyhledávacích příkazů a následně je odešle na příslušný indexer. Následně výsledky dotazů shromáždí, vytřídí a prezentuje je zpět uživateli. [28]

1.5 Suricata

1.5.1 Úvod do Suricaty

Suricata je open-source systém pro detekci průniku (*IDS*), prevenci průniků (*IPS*) a monitorování sítě. Je vyvíjený a spravován komunitou OISF (*Open Information Security Foundation*). Původním cílem Suricaty bylo vyvinout systém pro detekci průniku založený na signaturách, který by zlepšil svého předchůdce Snort. Cílem bylo vytvořit IDS který by měl stejný detekční jazyk jako Snort a byl zaměření především na komunitu. [16][23]

1.5.2 Základní komponenty a architektura

Architektura Suricaty je navržena pro paralelní zpracování dat, což umožňuje efektivní analýzu síťového provozu i v prostředích s vysokou propustností. Mezi klíčové komponenty celého systému patří dekodér paketů, detekční modul a modul pro zpracování toků a správu pravidel. [16]

Dekodér po zachycení paketů identifikuje a analyzuje síťové protokoly, včetně HTTP, DNS, TLS a dalších. Jedná se o hloubkovou inspekci paketů, která umožňuje detekovat anomálie nebo hrozby specifické pro jednotlivé protokoly. [16]

Suricata využívá sofistikovaný detekční modul k identifikaci potenciálních bezpečnostních incidentů. Centrálním prvkem této detekce jsou pravidla, která jsou psána ve standardizovaném formátu. Pomocí těchto pravidel je možno popsat velké množství síťových aktivit, které by mohly odhalit přítomnost útoků nebo podezřelé chování.[16][23]

1.5.3 Struktura a role detekčních pravidel

Každé pravidlo je definováno podle jasné syntaxe a specifikuje konkrétní podmínky, při kterých dochází k vyvolání reakce. Pravidla obsahují následující základní části: [16][22]

- **Akce**, která popisuje reakci systému, pokud dojde ke splnění podmínky pravidla. Může se jednat o generování upozornění, zahození paketu nebo zamítnutí celého spojení

- **Protokol**, specifikuje druh protokolu, který je pravidlem sledován.
- **IP adresy a porty**
- **Klíčová slova,** která jsou využívána hlavně pro detailní vyhledávání v analyzovaných datech. Může jít o jednoduché řetězce, ale i složitější výrazy a regulární výrazy.

```
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"HTTP GET Request
Containing Rule in URI"; flow:established,to_server; http.method;
content:"GET"; http.uri; content:"rule"; fast_pattern; classtype:bad-
unknown; sid:123; rev:1;)
```

Obrázek č. 3: Příklad detekčního pravidla v systému Suricata
(Zdroj: [22])

1.5.4 Výstupy systému Suricata

Suricata umožňuje generovat výstupy v několika různých formátech. Každý z formátů má svá specifická využití. I tak všechny především sdílí stejnou myšlenku jednoduché kompatibility se systémy SIEM. [16]

Jedním z hlavních výstupních formátů je EVE (Extensible Event Format) JSON. Tento formát obsahuje podrobné informace o každé zachycené události, včetně identifikace útoku, obsahu paketů, informací o protokolech a metadat. [16]

Druhým typem je Syslog. Jedná se o tradiční textový formát, který je používaný pro centrální správu událostí u velkého množství systémů a aplikací. Výhodou syslogu je jeho univerzálnost a velmi snadná integrace se SIEM řešením. [16]

1.6 Apache Spark

1.6.1 Základní charakteristika Apache Spark

Apache Spark je open-source distribuovaná výpočetní platforma, která primárně slouží pro efektivní zpracování velkých objemů dat, *(tzv. Big Data)*. Tato platforma může provádět rychlé paralelní výpočty na velkých clusterech výpočetních uzlů. Hlavní výhodou, která Apache Spark odlišuje od jiných platforem pro analýzu velkých objemů dat, je schopnost provádět své výpočty primárně v operační *paměti (tzv. in-memory processing)*, pomocí čeho výrazně redukuje dobu pro zpracování dat. Spark nabízí možnost pro práci hned s několika programovacími jazyky, konkrétně se jedná o Scala, Python, Java a R. Integrace těchto jazyků je přínosná v rozsáhlejších technologických

systémech a usnadňuje práci, jak analytikům, tak datovým vědcům. Apache Spark je také modulární architekturou. Jeho jádro se skládá ze Spark Core, které zajišťuje základní funkcionalitu. Mezi takové funkce se řadí správa paměti, plánování úloh a přerozdělování výpočetních prostředků napříč celým clusterem. Dalšími moduly, které na toto jádro navazují jsou Spark SQL, které se používá především pro práci se strukturovanými daty, Spark Streaming pro real-time zpracování proudových dat. Posledními moduly pak jsou, knihovna MLlib, využívaná pro strojové učení a modul GraphX pro analýzu grafů. [2][8][17]

1.6.2 Historie a vývoj

Apache Spark vzniknul v roce 2009 jako výzkumný projekt na Kalifornské univerzitě Berkeley v rámci tamní AMP laboratoře, která se zabývá výzkumem v oborech jako jsou strojové učení, dolování dat, databáze, vyhledávání informací, zpracování přirozeného jazyka, rozpoznávání řeči, ale především Big Data. Hlavním průkopníkem v tomto oboru byl Matei Zaharia, který chtěl vytvořit alternativu k dřívější technologii Hadoop MapReduce, přestože byla ve své době zásadním nástrojem pro zpracování dat distribuovaným způsobem. Mezi velký nedostatek tohoto nástroje se ale řadil způsob, jakým docházelo k ukládání mezivýsledků, ty se totiž ukládaly přímo na disk, což způsobovalo velké zpomalení zejména při iterativních úlohách, které jsou právě hlavní součástí strojového učení a pokročilé analýzy dat. [1][2]

Jakmile se tento výzkumný projekt ukázal jako životaschopné řešení, které překonává MapReduce, byl v roce 2010 vydán pod licencí open-source a v roce 2013 se stal projektem nejvyšší úrovně Apache. Následně se skupina výzkumníků pracujících na tomto výzkumném projektu dala dohromady a založila společnost Databricks. Tato společnost je dnes hlavním komerčním správcem Sparku. V roce 2015 společnost IBM oznámila významnou investici do vybudování technologického centra Spark, které bude rozvíjet Apache Spark pomocí úzké spolupráce s open source komunitou a zabudováním Sparku do jádra analytických a obchodních platforem společnosti. [1]

1.6.3 Základní koncepty a architektura

Pro celkové porozumění Apache Spark je důležité nejprve získat základní přehled o klíčových konceptech a hlavních komponentách, které tento systém tvoří. Tato část se

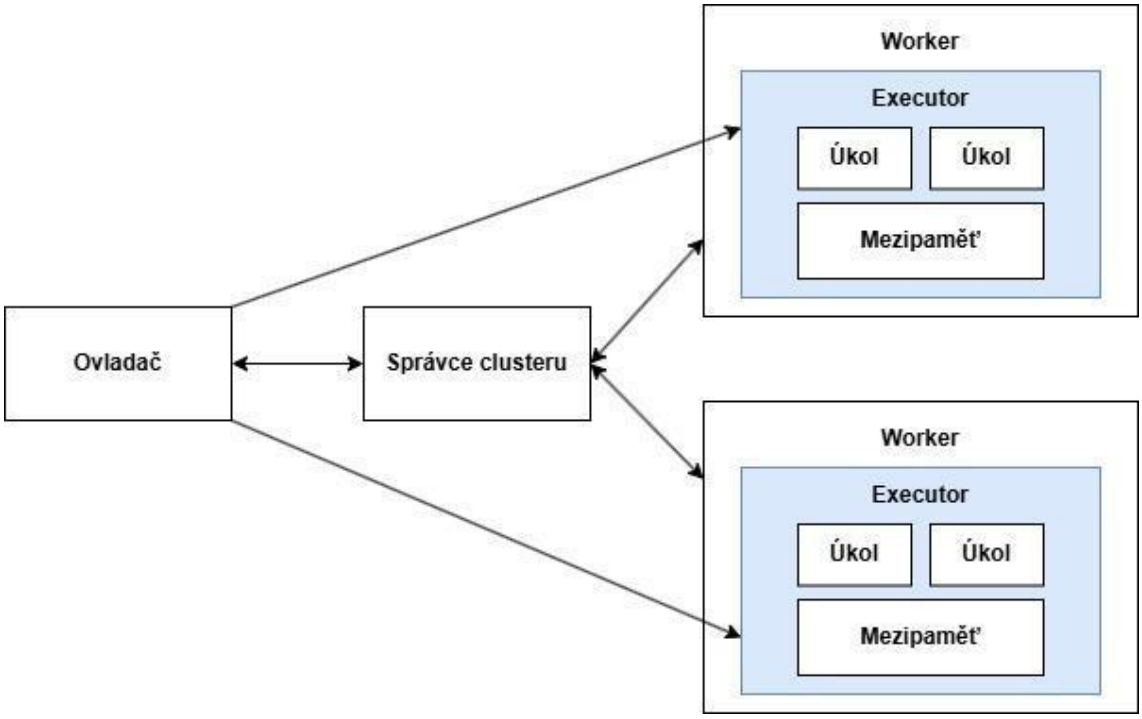
proto zaměřuje na vysvětlení principů Spark clusterů, systému pro správu prostředků, samotných Spark aplikací, dále na roli Spark driverů a funkcí Spark executorů. [8][15]

Spark Clustery a systém správy zdrojů

Apache Spark je v jádru distribuovaný systém navržený pro efektivní a rychlé zpracování velkého objemu dat. Tento systém je obvykle nasazen na množině strojů, označované jako **Spark cluster**. Velikost tohoto clusteru se může pohybovat od několika málo strojů až po tisíce uzlů. Největší veřejně známý Spark cluster na světě zahrnuje více než osm tisíc strojů. Pokud bychom chtěli efektivně spravovat takto rozsáhlou infrastrukturu strojů je zapotřebí inteligentního systému pro správu zdrojů. Příkladem takových správců jsou především Apache YARN nebo Apache Mesos. Typicky je systém správy zdrojů tvořen dvěma hlavními komponentami **správce clusteru (cluster manager)** a **pracovní uzly (workers)**. [8][15][17]

Správce clusteru uchovává informace o umístění workerů, množství dostupné paměti a počtu CPU jader na jednotlivých strojích. Jedním z hlavních úkolů je koordinace práce tím, že jednotlivým workerům přiřazuje konkrétní úkoly. [8][15][17]

Worker pak nabízí své prostředky, kterými jsou například paměť nebo CPU a vykonává přidělenou práci, jako spouštění konkrétních procesů a monitorování jejich stavu. Spark je navržen tak, aby snadno spolupracoval s těmito systémy pro správu zdrojů, v případě potřeby však nabízí i vlastní vestavěný cluster manager, který umožňuje spravovat dedikovanou sadu strojů přímo pro účely zpracování dat pomocí Apache Spark. [8][15][17]



Obrázek č. 4: Interakce mezi aplikací Spark a správcem clusteru
(Zdroj: Vlastní zpracování dle [8])

Aplikace Spark

Aplikace Spark je složena ze dvou částí. První částí je logika, jak dochází ke zpracování dat aplikace za pomoci Spark API. Tato zpracovací logika může být třeba reprezentována jen pár řádky kódu, které provádí jednoduché operace, ale i složitým řešením v podobě trénování rozsáhlého modelu pro strojové učení. [17]

Druhou částí je zde **ovladač Spark (spark driver)**. Ovladač funguje jako centrální koordinátor samotné aplikace. S jeho pomocí dochází ke komunikaci se správcem clusteru, aby zjistil, na kterých strojích má být spuštěna logika pro zpracování dat. Pro každý z těchto strojů následně ovladač požádá správce clusteru, aby spustil proces zvaný Spark executor. Dalším úkolem ovladače je jménem aplikace spravovat a přerozdělovat úlohy a operace na jednotlivé executory. [8][15][17]

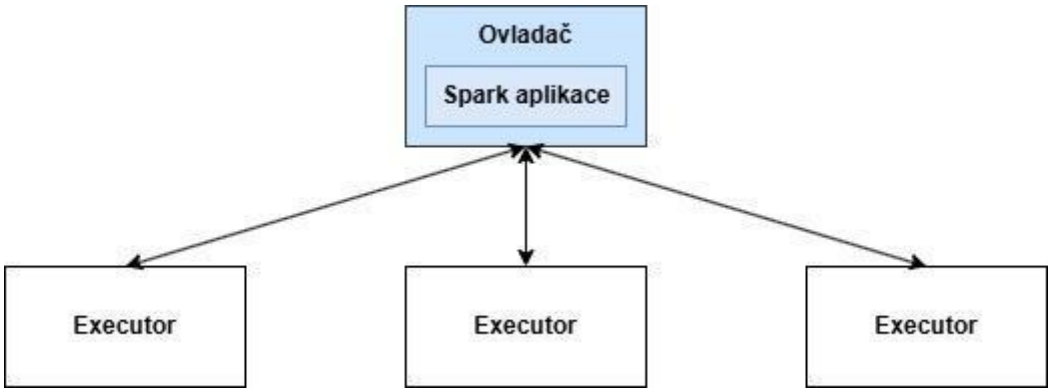
Vstupní bod do aplikace Spark je prostřednictvím třídy nazvané SparkSession, která poskytuje prostředky pro nastavení konfigurace a také API pro vyjádření a zadání logiky zpracováni dat, jako je třeba nahrání scriptu psaného v jazyce python.[8][17]

Spark Executor

Každý Spark executor je proces, který je přidělen výhradně konkrétní aplikaci Spark. Děje se tak za účelem, aby se zabránilo sdílení jednoho executoru mezi více aplikací. Výsledkem je, že jsou od sebe aplikace izolovány stylem, aby jedna špatně se chovající aplikace neovlivnila zbytek správně fungujících aplikací. Aby spolu mohly exekutoři sdílet data napříč jednotlivými aplikacemi, je zde zapotřebí zapisovat data do externího úložného systému, ke kterému mohou všichni přistupovat. [8][15][17]

Spark využívá architekturu typu master-slave, kde ovladač má roli hlavního řídícího prvku (*master*), zatímco executor vystupuje jako podřízená jednotka (*slave*). Obě tyto komponenty běží jako samostatné procesy v rámci Spark clusteru. Každá aplikace je tvořena vždy jedním Spark ovladačem a jedním nebo více exekutory. **Spark executor provádí přiřazené úkoly, které představují jednotlivé části logiky zpracování, které se zde nazývají Spark job**. Každý job je realizován na samostatném procesorovém jádru. Pomocí této paralelizace je Spark schopen výrazně urychlit zpracování rozsáhlých datových objemů. Jak již bylo řečeno v předchozí části, kromě samotného vykonávání logiky je také úkolem executorů také ukládání časti dat v operační paměti nebo na disku, pokud to vyžaduje aplikační logika. [8][15][17]

Při spuštění Spark aplikace existuje možnost specifikovat, přesně kolik executorů má být alokováno, jaké množství paměti může nebo má být každému z nich přiděleno a kolik CPU jader budou využívat. Aby bylo možné optimalizovat nastavení těchto parametrů je zapotřebí odhad, jak velký bude objem zpracovávaných dat, jak náročná bude samotná datová logika a za jakou dobu má být zpracování dokončeno. [8][15]



Obrázek č. 5: Spark cluster s třemi executory
(Zdroj: Vlastní zpracování dle [8])

1.7 Apache Kafka

1.7.1 Distribuované systémy

Pod pojmem distribuovaný systém si můžeme představit typ architektury, ve které jsou výpočetní zdroje rozprostřeny napříč vícero samostatnými fyzickými nebo virtuálními jednotkami. Výhodou této architektury je, že umožňuje horizontální škálování a celkově zlepšuje dostupnost systémů. Další výhodou může být například replikace dat napříč systémem, což má za výsledek zvýšení odolnosti vůči selhání jednotlivých komponent systému. [26]

Dnes tvoří distribuované systémy základ mnoha aplikací a služeb. Jako příklad moderních distribuovaných systému můžeme uvést streamovací platformy, cloudová uložiště nebo třeba velké datové sklady firem. Jelikož dochází k neustále se zvětšujícímu objemu generovaných dat, bývá tak stále více kladen důraz na schopnost efektivního přenosu, zpracování a ukládání dat v reálném čase. Problém nastává právě u „klasických“ přístupů k přenosu dat, které jsou založené na dávkovém zpracování dat. Tyto systémy přestávají vyhovovat potřebám, kterými mohou být schopnost přenášet obrovské množství zpráv za jednotku času, ale třeba také minimální latence (tj. časové zpoždění při doručení zprávy). [26]

Právě v reakci na tyto požadavky vznikla platforma Apache Kafka, jejíž hlavní výhodou je schopnost poskytovat distribuované zpracování dat, které je využitelné v reálném čase a zároveň vysoce spolehlivé. **Platforma je navržena tak, aby sloužila jako distribuovaný záznamový systém, umožňující asynchronní komunikaci mezi různými částmi systémů.** [26]

1.7.2 Historie a vývoj

Apache Kafka vznikla jako interní projekt v rámci společnosti LinkedIn, kde bylo původním cílem vytvořit odolný systém pro sběr a distribuci velkých objemů logovacích dat v reálném čase. V době vzniku, v roce 2010, čelil LinkedIn problémům s tehdejšími messaging systémy, které nebyly dostatečně výkonné a spolehlivé při škálování na úroveň velkých datových toků. [26]

Původní návrh systému byl založen na principu publish-subscribe architektury, která umožňuje oddělení producentů dat od jejich konzumentů, čímž je zajištěna vysoká míra modularity. Název Kafka byl zvolen na podnět jednoho ze zakladatelů systému, Jay

Krepsa, který měl v oblibě literárního autora Franze Kafky. V jednom z rozhovorů na toto téma také řekl, „Myslel jsem, že vzhledem k tomu, že Kafka je systém optimalizovaný pro psaní, dávalo použití jména spisovatele smysl“. [26]

V roce 2011 byl projekt vydán s licencí open-source pod Apache a zároveň byl zařazen do Apache Software Foundation. V následujících letech prošel vývojem, který jej dokázal transformovat z jednoduchého systému pro logování dat na komplexní platformu pro streamované zpracování zpráv a událostí. Jedním z významných milníků ve vývoji Apache Kafka bylo zavedení modulu Kafka Connect, který umožňuje snadné propojení Kafky s externími systémy bez nutnosti vývoje vlastního integračního přístupu. Tento krok výrazně rozšířil možnosti praktického využití Kafky při budování moderních datových kanálů v systémech. Dalším zlepšením byl přechod z dříve používaného externího koordinátora a správce metadat Apache Zookeeper, na dnes už zcela vlastní Kafka řešení nazvané KRaft. Došlo tak ke zjednodušení nasazování Kafka clusterů a správě metadat přímo v samotné Kafka infrastruktuře.[26][36]

Důležitou roli v rozvoji Apache Kafka také měla firma Confluent, kterou založili původní autoři projektu Jay Kreps, Jun Rao a Neha Narkhede. Confluent se podílí na podpoře, rozvoji a komerční distribuci Apache Kafka. Společnost také přináší do celého systému další nástroje a moduly, jako například ksqlDB (streamovací databáze, která využívá SQL syntaxi), nebo také Control Center, který slouží pro vizualizaci a správu clusterů. [26]

V dnešní době je Apache Kafka jedním z nejpoužívanějších nástrojů v oblasti zpracování datových toků v reálném čase a je tak součástí datových architektur velkých technologických společností jako jsou například Netflix, Spotify nebo Uber. Své využití, ale také může nalézt i u menších projektů, které usilují o implementaci datových pipeline a architektur založených na událostech. [26]

1.7.3 Základní koncepty a architektura

V této části budou představeny základní komponenty a koncepty Apache Kafka mezi které patří především **zpráva, schéma, broker, topic, oddíl, producent a konzument**. Všechny tyto komponenty společně tvoří základ škálovatelné a odolné infrastruktury.

Zprávy, Batches

Základní datovou jednotkou v rámci Apache Kafka je **zpráva.** Ekvivalentem v databázových systémech je řádek nebo záznam. Z pohledu Kafky se jedná o pole bajtů. Samotné bajty nemají pro Kafku specifický formát, a tedy ani kontext. Zpráva může obsahovat volitelný kus metadat, který je rovněž pole bajtů a označuje se jako klíč. Tyto klíče je možné používat pro přiřazování specifických zpráv do oddílů v topics. [26]

Pro efektivitu se zprávy do Kafky zapisují v **dávkách (batches)**. Dávka je jednoduše kolekce zpráv, která se zapisuje do stejného topicu. Kdyby se měla každá zpráva po síti posílat individuálně, vedlo by to k nadměrné zátěži. Tento problém právě shromažďování zpráv do dávek řeší. I tak se jedná o kompromis mezi latencí a propustností, tzn. Čím větší velikost dávek, tím více zpráv lze zpracovat za jednotku času, ale tím déle trvá přeposlání. [26]

Schéma, formát Avro

Jelikož samotná Kafka do zprávy, reprezentované polem bajtů, „nevidí“, je zapotřebí využití další struktury, které se jednoduše nazývá **schéma.** Schéma dává zprávám kontext, určuje datové typy a strukturu, aby je bylo možno pochopit a dále s nimi pracovat. Těchto schémat může být velké množství, které se odvíjejí od jednotlivých potřeb aplikací, do kterých Kafka své zprávy přeposílá. Zjednodušené formáty, jako je JSON a XML, jsou snadno použitelné a lidsky čitelné. Postrádají však funkce, jako je robustní zpracování datových typů a kompatibilita mezi verzemi schémat. Proto mnoho vývojářů při práci s Kafkou upřednostňuje využívání formátu **Apache Avro.** [26]

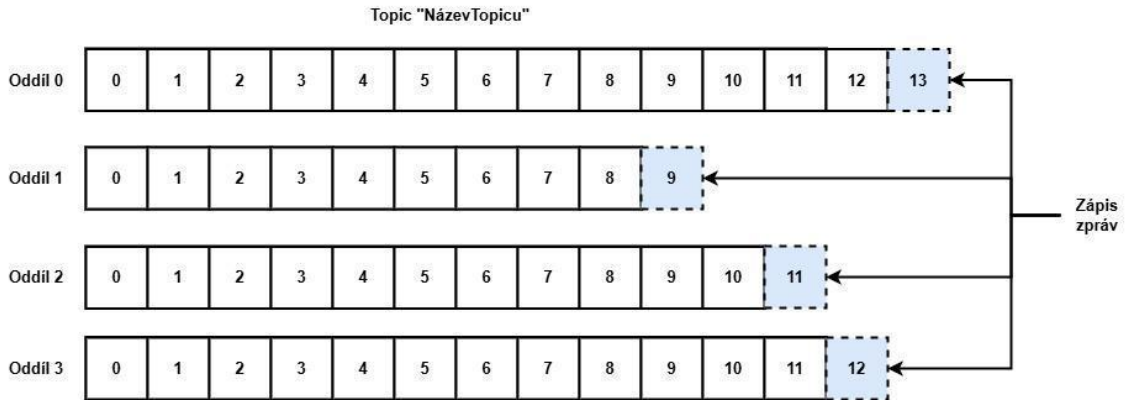
Apache Avro je datový binární formát navržený pro efektivní ukládání a přenos dat v distribuovaných systémech jako je právě Apache Kafka nebo Apache Spark. Jeho hlavní výhodou je schopnost definovat strukturu dat pomocí zmíněného schématu, které je uloženo v JSON formátu. V Apache Kafka se formát Avro často používá ve spojení se službou **Schema Registry**, které slouží jako centralizované uložiště pro schémata. Schema Registry také umožňuje producentům a konzumentům sdílet a spravovat schémata, což zajišťuje kompatibilitu při jejich předávání. Když producent posílá zprávu do Kafky, data nejprve serializuje dle definovaného schématu a toto schéma uloží do Schema Registry, pokud zde ještě není. Samotná zpráva pak již nenese kompletní definici schématu, ale pouze unikátní identifikátor tohoto schématu. Tento přístup významně snižuje objem přenášených dat. Konzument následně během příjmu zprávy využije

přiložený identifikátor k načtení odpovídajícího schématu ze Schema Registry, pomocí kterého přijatá data zase deserializuje. [26]

Konzistentní formát dat je v systému Kafka důležitý, protože umožňuje oddělit zápis a čtení zpráv. Kdyby tyto akce nebyly navzájem odděleny mohlo by pak docházet k problémům, kdy aplikace, ze které jsou zprávy zapisovány, bude mít starou verzi schématu než například aplikace, která zprávy čte a dojde tak nesprávné deserializaci. Právě proto, je použití a dobře definovaných schémat a jejich uložení ve společném uložišti klíčové, pro správný chod celého systému Kafka. [26]

Topic, Oddíly

V systému Apache Kafka jsou zprávy uspořádány do tzv. **topics**, které lze přirovnat k databázovým tabulkám nebo složkám v souborovém systému. Každý topic je dále rozdělen na několik oddílů. Každý oddíl si můžeme představit jako samostatný zásobník, do něhož jsou zprávy zapisovány sekvenčně, a to pouze formou přidávání na konec. Čtení zpráv probíhá ve stejném pořadí, v jakém byly zapsány. Každá zpráva, která je tímto způsobem zapsána do topicu dostane přiřazený číselný identifikátor, který je označován jako **offset**. Tento offset uchovává informaci o tom, na jaké pozici v rámci oddílu se zpráva nachází. Zprávy jsou zároveň uloženy na disku po předem stanovenou dobu, nebo dokud na disku nedojde místo. Tato logika dovoluje, že konzumenti nemusí zprávy ihned po jejich publikaci zpracovávat. Konzument tak může libovolně posouvat svoji pozici v toku zpráv za pomoci již zmíněného offsetu. Lze tak zpracovávat i historické zprávy, nebo opětovně vyčítat zprávy v případě, že by některá z aplikací vypadla. [26]



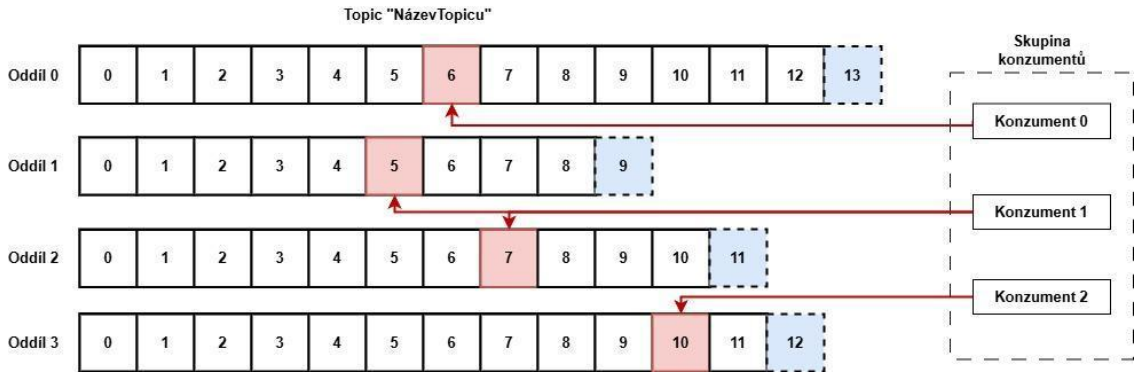
Obrázek č. 6: Znázornění topicu s více oddíly
(Zdroj: Vlastní zpracování dle [26])

Producent, Konzument

Klienti systému Apache Kafka představují uživatele této platformy a dělí se do dvou základních kategorií **producenti** a **konzumenti**.

Úlohou producentů je jednoduše vytvářet nové zprávy. V jiných systémech typu publish/subscibe, se můžeme setkat s podobným označením jako je například publisher nebo writer. Zprávy jsou publikovány vždy do konkrétního topicu. Ve výchozím nastavení se snaží producent vždy rovnoměrně rozdělit zprávu do jednotlivých oddílů v rámci zvoleného topicu. [26]

Konzumenti mají opačnou úlohu, a to číst zprávy. Opět se u obdobných systémů se můžeme setkat s názvy jako subscriber nebo reader. Konzumenti se totiž přihlašují k odběru jednoho nebo více topics a vyčítají z nich zprávy v pořadí, v jakém byly zapsány do oddílu, čehož je docíleno pomocí již zmíněných offsetů. Konzumenty lze také spojit do tzv. konzumentských skupin, kde jednotlivý členové spolupracují na odběru zpráv z topicu. Každý oddíl je v rámci skupiny čten vždy právě jedním konzumentem. Takto přiřazený oddíl se nazývá vlastněným. Tímto sloučením do jedné skupiny, je docíleno možnosti horizontálního škálování odběru zpráv z velmi rozsáhlých topiců. Další výhodou je, že pokud by došlo k selhání některého z konzumentů, jeho vlastněný oddíl, který konzumoval, by se automaticky převedl na zbývající členy skupiny, čímž je zajištěna vysoká odolnost celého systému. [26]



Obrázek č. 7: Odebírání zpráv z topicu skupinou konzumentů
(Zdroj: Vlastní zpracování dle [26])

Broker

Individuální server v rámci systému Apache Kafka se označuje jako **broker**. Broker má na starosti příjem zpráv od producentů, přiřazování offsetů a následné ukládání na diskové úložiště. Zároveň se stará o obsluhu konzumentů tím, že reaguje na jejich

požadavky ohledně načítání zpráv z konkrétních oddílů. Díky optimalizaci pro vysoký výkon je jeden broker schopen obsluhovat tisíce oddílů a zpracovávat miliony zpráv za sekundu, a to v závislosti na použitém hardwaru a jeho výkonnostních charakteristikách. Broker je navržen tak, aby byl provozován v clusteru. V každém clusteru se automaticky zvolí jeden broker jako řadič, jehož funkcí je řízení administrativních operací, například přidělování oddílům jednotlivým brokerům nebo monitorování dostupnosti brokerů. Každý oddíl je spravován jedním brokerem, který v této roli funguje jako vůdce. Pokud by se jednalo o replikovaný oddíl, jsou zprávy navíc ukládány na dalších brokerech, kteří se označují jako následovníci. [26]

Pokud chceme zajistit opravdu odolné řešení v rámci systému Apache Kafka, musí být zmíněna **replikace**. Replikace garantuje dostupnost dat, pokud by došlo k selhání vůdčího brokera. Při jeho výpadku, převezme správu celého clusteru právě jeden z jeho následovníků. Systém je navržen tak, že producenti při zápisu zpráv komunikují vždy přímo s vůdcem, zatímco konzumenti mohou číst zprávy jak od vůdce, tak od následovníka. [26]

Retence dat

Zásadní vlastností Apache Kafka je koncept retence, tedy uchovávání zpráv po předem definovanou dobu nebo do dosažení určitého objemu dat. Každý broker je nakonfigurován s výchozím nastavením retence, například uchováváním zpráv po dobu měsíce nebo do dosažení velikosti oddílů například 100 GB. Po překročení těchto limitů se starší zprávy automaticky smažou. Retenční politika tak definuje minimální množství dat, která budou v systému kdykoliv k dispozici. Pro jednotlivé topics lze nastavovat specifické retenční parametry, dle potřeb na zpracovávaná data. [26]

1.8 Technologie kontejnerizace

1.8.1 Kontejnerizace

Kontejnerizací se rozumí přístup k virtualizaci aplikací, který umožňuje izolovat a spouštět aplikace i s jejich závislostmi v odděleném prostředí. Tento způsob se liší od tradičních metod absencí požadavků na samostatný operační systém pro každý virtuální stroj. Kontejnery spotřebovávají méně systémových zdrojů, jako je paměť nebo CPU, což umožňuje efektivnější správu infrastruktury vybudované na těchto kontejnerech. [37]

Zásadní výhodou kontejnerů je jejich přenositelnost a konzistence spouštěcího prostředí napříč různými systémy. Kontejnery jsou izolované na úrovni operačního systému prostřednictvím mechanismů, jako jsou *namespaces* a *cgroups*. Tyto mechanismy zaručují, že aplikace běžící v kontejneru zůstávají izolované a neovlivňují další aplikace nebo hostitelský operační systém. [37]

Kontejnery lze stavět pomocí tzv. obrazů (*image*) kontejnerů. Jedná se o neměnné šablony obsahující aplikace a jejich závislosti. Obrazy jsou ukládány ve veřejných knihovnách kontejnerů jako je Docker Hub. V těchto knihovnách jsou jednotlivé obrazy verzovány a následně distribuovány mezi různé týmy a prostředí. [37]

Výraznou předností kontejnerů je také rychlost, jakou mohou být nasazeny a spuštěny. Nemusí zde docházet ke spouštění celého operačního systému, takže se spuštění může pohybovat i v řádu sekund. Kontejnerizace je nejvíce využívána v prostředích, která kladou důraz na modularitu a flexibilní nasazování aplikací. [37]

1.8.2 Platforma Podman

Podman je moderní open-source platforma pro správu a spouštění kontejnerů bez nutnosti běhu služby na pozadí systému. Na rozdíl od klasického přístupu, jaký nabízí třeba oblíbená platforma Docker, která využívá službu běžícím pod root oprávněním, **Podman nabízí bezrootový (*rootless*) přístup ke kontejnerizaci.** To znamená, že uživatel nemusí mít administrátorská práva pro vytváření a provoz kontejnerů. Pokud by útočník například získal přístup do kontejneru, díky izolaci by získal pouze omezená práva daného uživatele, nikoliv administrátora. Tento přístup výrazně zvyšuje bezpečnost celého systému a snižuje rizika spojená s eskalací oprávnění. [35][38]

Základním principem Podmanu je využívání tzv. OCI (*Open Container Initiative*) standardů pro kontejnery a jejich obrazy. Díky tomu je kompatibilní s obrazy vytvořenými i jinými nástroji, jako je již zmíněný Docker. Je zde tedy snadná migrace z Dockeru pomocí téměř plné kompatibiltě příkazů, což umožňuje vývojářům přechod bez nutnosti změny již zavedených postupů. [38]

Bezpečnost je zde také posílena možností monitorování kontejnerů, což zahrnuje logování a inspekci stavů. Platforma je tak vhodnou volbou pro open-source komunity ale i podnikatelský sektor, kde je kladen důraz na bezpečnost a dodržování otevřených standardů. [38]

2 ANALÝZA SOUČASNÉHO STAVU

Řešení, které jsem popsal v následujících kapitolách vznikalo v rámci odborné činnosti na Národním úřadu pro kybernetickou a informační bezpečnost (NÚKIB). Na tomto úřadě pracuji na pozici analytika síťového provozu, a to v rámci oddělení analýzy síťového provozu (OASP). Oddělení organizačně spadá pod odbor vládní CERT České republiky.

Obrázek č. 8: Logo Národního úřadu pro kybernetickou a informační bezpečnost
(Zdroj: [20])

2.1 Vládní CERT

Vládní CERT tým hraje významnou roli při ochraně kritické informační infrastruktury a významných informačních systémů, a to dle zákona o kybernetické bezpečnosti (181/2014 Sb.) a jeho prováděcích spisů. Hlavní úlohou tohoto týmu je účinně čelit bezpečnostním výzvám, reagovat na vzniklé incidenty a koordinovat činnost při jejich řešení. Stejně významnou roli také hraje předcházení incidentům, a to například v podobě osvěty, poskytování prvotních bezpečnostních informací a pomoci pro orgány státu, neregulované organizace, ale dokonce i občany. Vládní CERT pomáhá zvyšovat vzdělanost a povědomí v oblasti bezpečnosti na internetu. Orgány a osoby, na které se vztahuje zákon o kybernetické bezpečnosti, musí plnit své povinnosti právě vůči vládnímu CERT týmu. [19]

2.1.1 Poskytované služby

Koordinační činnost a pomoc při řešení incidentů

Při řešení bezpečnostních incidentů má vládní CERT připraven k dispozici tým odborníků, který poskytuje asistenci IT specialistům zasaženého subjektu po technické stránce. Tento tým je schopný také poskytovat dodatečné rady a navrhnout další preventivní opatření. Mezi takovou asistenci se může řadit analýza síťových dat, síťový threat hunting nebo forenzní analýza koncových stanic. Pokud nastane situace, kdy některý incident směřuje na vícero regulovaných subjektů, vládní CERT koordinuje

společný postup při jeho řešení. Vzhledem k rozsáhlé spolupráci s různými institucemi lze zprostředkovat kontakty jak s bezpečnostními týmy na území České republiky, tak se zahraničními partnery, kterými mohou být vládní CERT týmy jiných zemí. [19]

Penetrační testování

Penetrační testování je další službou, kterou vládní CERT poskytuje. Jedná se o simulaci útoků, které by mohly subjekt zasáhnout. Veškeré testování je prováděno na základě podepsané smlouvy. Výsledkem tohoto testování je zpráva o chybách v zabezpečení. Tato služba je bezplatná a v současnosti je poskytována především pro subjekty regulované zákonem o kybernetické bezpečnosti. Tým poskytuje více druhů těchto testů, jedná se především o interní testy, které mají simulovat útočníka, který se nachází již ve vnitřní síti útočníka. Externí testy, které naopak simulují útočníka, který útočí z vnější sítě a nemá přehled o infrastruktuře podniku.

Součástí služeb penetračního testování je také tvoření phishingových a vishingových kampaní na základě předchozí dohody, které mají za úkol otestovat, zdali jsou zaměstnanci schopni rozeznávat škodlivé emailové zprávy a jaké jsou jejich reakce. Výsledkem takových kampaní je zpráva a doporučení pro zlepšení. [19]

Forenzní laboratoř

V návaznosti, na již proběhlé nebo stále probíhající incidenty jsou specialisté z CERT týmu schopni poskytnout forenzní analýzu napadených koncových stanic nebo síťového provozu. Forenzní laboratoře se zaměřují na technické vyšetřování kompromitací systémů. Aby bylo možno tyto analýzy začít, je zapotřebí zajistit veškerá data, která by mohla v tomto vyšetřování pomoci. I s tímto je vládní CERT schopen asistovat subjektům. Jedná se tak o vytváření kopií disků, obrazů paměti a extrakci logů.

Výsledkem těchto služeb je podrobná analýza průběhu celého incidentu, ze kterého je vytvořena závěrečná zpráva obsahující zdokumentovaný postup analýzy. Tato zpráva může následně přispět k tomu, aby se podobným incidentům do budoucna předešlo kompletně nebo alespoň došlo k minimalizaci následků pro daný subjekt. [19]

Projekt systém detekce

Tato služba zahrnuje neustále se rozšiřující systém detekce, do kterého se mohou jednotlivé subjekty zapojit. Systém zpracovává kybernetické události a data o síťovém provozu a záznamy z perimetrů zapojených organizací. Cílem projektu je tak schopnost

detekovat globální hrozby a jako výsledek publikovat blacklisty nebo signatury pro zapojené partnery.

2.2 Oddělení analýzy síťového provozu (OASP)

Analýza síťového provozu je jednou z hlavních složek v rámci vládního CERT týmu. Hlavním posláním tohoto oddělení je provádění forenzních analýz kybernetických útoků směřujících na regulované subjekty. Dále se také zabývá vyhledáváním škodlivých aktivit a indikátorů kompromitace (*IoC*) v síťových záznamech. Důležitá je také pro toto oddělení spolupráce s reaktivním oddělením tzv. incident handlingem a analytickým oddělením, které se zabývá malware analýzou koncových stanic.

Významným přínosem pro OASP je využívání analytického nástroje Splunk, který zásadním způsobem analytikům zjednodušuje a zrychluje vyhledávání nad velkým objemem získaných logů.

2.3 Původní systém detekce

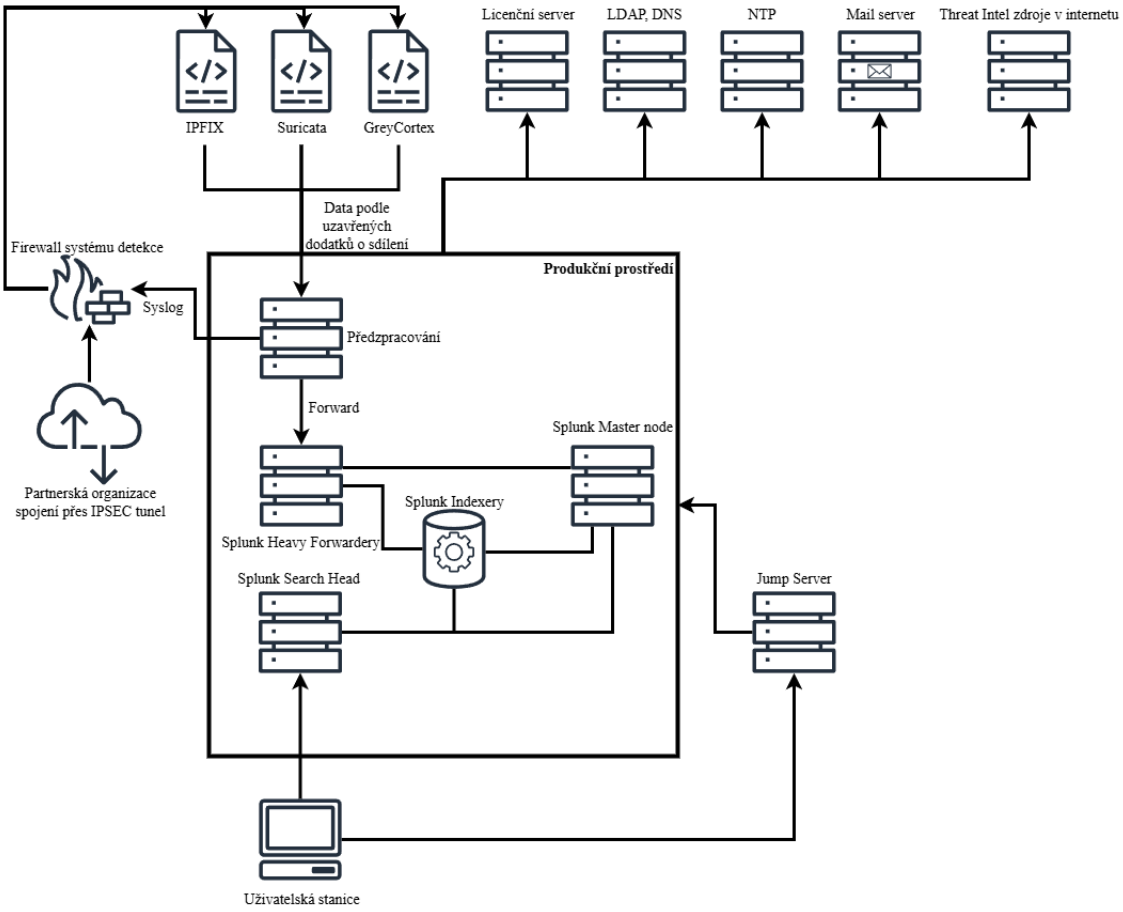
Cílem této kapitoly je detailně popsat výchozí stav systému detekce, identifikovat jeho slabiny a ukázat na konkrétní faktory, které vedly k rozhodnutí o jeho redesignu.

Projekt systém detekce vzniknul ve své prvotní verzi už v roce 2020 a byl sponzorován z dotací poskytnutých Evropskou unií. Jedná se o projekt, do kterého je momentálně zapojeno 17 organizací napříč celou Českou republikou, které z důvodu citlivosti dat nebudou v této práci blíže jmenovány. Všechny tyto organizace spadají pod regulované subjekty specifikované v zákonu o kybernetické bezpečnosti (181/2014 Sb.).

Cílem systému detekce je sběr síťových NetFlow/IPFIX dat od zapojených organizací, nad kterými dále oddělení analýzy síťového provozu provádí síťový threat hunting s využitím analytického nástroje Splunk. Tento threat hunt může sloužit k proaktivním účelům, kdy analytici monitorují komunikaci na perimetru organizace a snaží se nalézt možné indikátory kompromitace na základě blacklistů nebo malware signatur, ještě před tím, než se útočníkovi podaří jakákoliv škodlivá činnost, kterou může být například kompletní zašifrování infrastruktury nebo exfiltrace utajovaných dat. Pokud by k takovým nálezům došlo, organizace by byla včas automaticky kontaktována a byla by jí poskytnuta asistence s nápravnými opatřeními a doporučením. Další možností, jak tento systém využít je reaktivní threat hunting, kdy v návaznosti na již proběhlý incident, mohou analytici analyzovat historické záznamy napadené organizace a poskytnout jim

tak objasnění, jak celý útok proběhl, včetně vypracování časové osy a použitých technik dle matice MITRE ATT&ACK.

2.3.1 Topologie původního systému detekce



Obrázek č. 9: Topologie původního řešení systému detekce
(Zdroj: Vlastní zpracování)

2.3.2 Slabiny původního řešení

Původní podoba systému detekce byla do značné míry limitována její hlavní komponentou Splunk. Ačkoliv byla na začátku projektu funkčně dostačující, s rostoucím počtem zapojených organizací a nárůstem objemu síťových toků se ukázalo, že tento model není dlouhodobě udržitelný, a začal se plánovat přechod na levnější licenci Splunku.

V době vzniku systému detekce byla zakoupena licence, která umožňovala denní indexaci dat až na kapacitu 500 GB/den. Tato licence se ovšem v dlouhodobém horizontu jevila jako nevýhodná, a tudíž nastaly tři možnosti řešení. První z nich byla možnost, že dojde

ke kompletnímu zastavení provozu systému detekce. Ten se již ale osvědčil jako velice přínosný, a tak se tato možnost kompletně vyloučila. Druhou možností bylo sjednocení jednotlivých dohledových systémů na státní úrovni, které ale bylo v tomto případě komplikované. Vznikla tedy třetí možnost řešení, při kterém došlo ke snížení dosavadní licence pouze na 100 GB/den. Tato možnost tedy vedla k nutnosti systém detekce upravit tak, aby i přes nezměněný objem příchozích dat mohla být tato data indexována ve Splunku. S tímto schváleným krokem se však ukázaly i další dlouhodobé slabiny původního systému.

První z nich byly **vysoké nároky na indexaci**. Všechna data, která přicházela z organizací nebyla specificky selektována nebo agregována. Zároveň se tyto datově objemné záznamy ihned indexovaly do Splunku. Některé druhy záznamu přitom nebyly dostatečně analyticky přínosné.

Další slabinou systému byla **omezená retence dat**. Opět z důvodu velkého objemu dat nebylo možné data uchovávat ve struktuře, která by byla přínosná pro analytiky. Forenzní analýza při šetření incidentů s delším časovým odstupem byla velmi složitá nebo i kompletně nemožná. V návaznosti na předešlé nedostatky také **docházelo k nízké efektivitě Splunk SPL dotazů**. Jelikož nebyla data agregována bylo vyhledávání nad daty velmi pomalé. Pokud analytik zadal komplexní dotaz s více filtry, delším časovým oknem nebo s velkou korelací výsledků, mohl takový dotaz trvat i několik desítek hodin.

Jako poslední příklad nedostatku předchozího systému byl považován chybějící článek mezi úpravou dat a jejich indexací. Architektura umožňovala jen velmi jednoduché úpravy, než byla data nahrána do Splunku. Jakékoliv větší změny jako přejmenování polí nebo filtrování tak musely být řešeny až uvnitř Splunku což bylo velice neefektivní a pro uživatele často nepřípustné. V systému také chyběla možnost obohacovat záznamy například o geolokaci nebo informace o reputacích IP adres.

Výše uvedené nedostatky vedly k rozhodnutí vedení CERT, že systém musí projít zásadní transformací. Jejím cílem bylo vytvořit vícevrstvou architekturu s předzpracováním dat, jejich agregací a selekcí, čímž by se výrazně snížil objem indexovaných dat a umožnilo by se zachovat přínos systému i při nižších nákladech na Splunk licenci.

3 VLASTNÍ NÁVRH ŘEŠENÍ

V této části bude detailně popsán návrh nové verze systému detekce, který bude splňovat požadavky na agregaci dat a celkové snížení objemu indexovaných dat ve Splunku. Jelikož se jedná o projekt, který má za cíl především přispívat k bezpečnosti v partnerských organizacích, budou některé části z tohoto důvodu záměrně anonymizovány nebo nebudou uváděny explicitně, aby nemohlo dojít ke zneužití kontextu těchto informací.

3.1 Motivace a cíl návrhu

Návrh nové verze systému detekce byl motivován nedostatky a limity, které se v průběhu provozu projevily u původní architektury, jak bylo podrobně zmíněno v předchozí kapitole.

Důležité je zmínit, že obecně například problémy s pomalými SPL dotazy nelze řešit pouze navýšením výpočetních prostředků celého systému. **Bylo tedy nutné zredukovat celkový objem dat zavedením mechanismu normalizace v kombinaci s agregací, a to s cílem sice zmenšit objem indexovaných dat do Splunku, ale zároveň přitom neztratit informační hodnotu pro analytiky.**

Celý systém byl také od začátku navrhován tak, aby byl bezpečný a přehledně monitorovatelný. Významný podíl návrhu zahrnoval důraz na automatizaci a verzování jednotlivých komponent, aby byla zajištěna celková robustnost a bylo možné co nejvíce předcházet nečekaným pádům systému.

Realizace návrhu řešení byla nejen technickým ale také organizačním úkolem, který vyžadoval komunikaci napříč celým analytickým týmem a muselo zde být dodržováno důsledné testování každé fáze implementace.

Tabulka č. 2: Srovnání výchozího stavu systému a požadavků na nové řešení
(Zdroj: Vlastní zpracování)

Požadavek	Předchozí stav	Nové řešení
Indexovaný objem	100 % bez selekce dat	Výrazně redukovaný
Retence dat	Omezena na krátké období	Výrazně prodloužená dle možností systému
Možnost škálování	Velmi omezená	Plně modulární
Čas zpracování SPL dotazu	Několik desítek minut až hodiny	Jednotky minut

3.2 Architektura návrhu nového systému detekce

Celková architektura navrženého systému je navržena jako modulární, škálovatelné řešení, které zajišťuje příjem, transformaci, serializaci a agregaci síťových toků. Tato architektura popisuje přeposílání dat z IPFIXcol2 kolektorů, jejich zpracování pomocí nástrojů Apache Kafka a Apache Spark a následnou indexaci do Splunku.

Logická struktura systému je přehledně znázorněna v **Příloze č.1**, která zobrazuje jednotlivé komponenty, datové toky a zpracování jednotlivými agregačními skripty. Klíčovou roli zde hrají topicy v Kafce a specializované Spark joby pro různé typy vstupních dat, jejichž podrobný popis je uveden v následujících kapitolách.

3.3 Příjem a kategorizace dat

Úvodní část návrhu architektury se zabývá strukturou nového řešení na koncepční úrovni. Část je zaměřena na hlubší analýzu technických detailů, konkrétně popis způsobu přijímání, kategorizace a zpracování vstupních dat. Síťové NetFlow záznamy, které systém přijímá a dále zpracovává, jsou naprostým základem všech analytických úloh a následného Threat huntingu. Proto je důležité, aby byl zajištěn jejich správný příjem, důvěrnost, integrita a následná distribuce napříč dalšími komponentami systému. Zapojení organizace do systému je realizováno na základě formální smlouvy, která definuje technické a organizační parametry spolupráce včetně délky vzájemné spolupráce.

3.3.1 Datové zdroje a vstupy

Základním datovým vstupem pro navržený systém jsou IPFIX data převážně ze zařízení Flowmon. Primárním nástrojem, který tato data zpracovává a posílá dále do centrální vrstvy je kolektor IPFIXcol2. Jedná se o specializovaný kolektor určený pro příjem, zpracování a export metadat o síťových tocích dle standardu IPFIX, konkrétně definovaný v dokumentaci RFC 7011.

IPFIXcol2 umožňuje export ve více datových formátech, pro tento návrh byl zvolen výstupní formát Kafka JSON. Tento formát byl vybrán, jelikož se jedná o oblíbený textový formát, který je jednoduše interpretovatelný. Právě interpretace byla u vývoje tohoto systému důležitá z hlediska validace dat při jejich testování.

Vedle IPFIX dat systém také musí být schopen přijímat i jiné datové zdroje. Jedná se tak o již zmíněné Suricata záznamy síťového provozu.

Všechny tyto záznamy mají podobnou strukturu, což umožnilo používat jednotný formát JSON. Existence pouze jednoho typu formátu tak poskytla možnost ucelit zpracování dat napříč celým systémem.

3.3.2 Centralizace přes Apache Kafka

Jakmile jsou data připravena a přijata IPFIXcol2, jsou dále exportována přímo do Apache Kafka, kde probíhá jejich prvotní shromáždění a kategorizace do jednotlivých topics. Pro účel exportu je využíván speciální plugin Kafka Exporter. Tento plugin je nativně podporován IPFIXcol2 a umožňuje okamžitý a neustálý export přijatých dat přímo do vybraného Kafka brokeru bez nutnosti externích skriptů nebo dalších mezi prvků v systému. Kafka je distribuovaná streamovací platforma, která umožňuje zpracovávat velké objemy dat s nízkou latencí a zároveň zajišťuje spolehlivost a odolnost vůči výpadkům.

Jedním z prvotních problémů, který musel být řešen, byla otázka: **Jakým způsobem rozlišovat z jaké organizace data pochází?** V počáteční fázích vývoje bylo zvažováno hned několik možných přístupů, jak tento problém řešit.

Všechna data v jednom společném Apache Kafka topicu

Tento přístup se z počátku jevil jako nejjednodušší varianta, jak spravovat a udržovat záznamy. Veškeré záznamy by tak byly dostupné z jednoho centralizovaného streamu. Rychle se však ukázalo, že v praxi nelze takto jednoduše určit, od které organizace daný

záznam pochází. Jak již bylo zmíněno v předchozí kapitole, každá organizace má sice svoji vlastní instanci IPFIXcol2 exportéru, ale to bylo v tento moment jediné rozlišení záznamů. IPFIX formát totiž standardně neobsahuje žádný explicitní identifikátor původního subjektu. I přesto, že lze z obsahu zprávy (např. *ExporterIPv4Address*) v některých případech označení organizace zpětně odvodit, nejedná se žádným způsobem o bezpečnou variantu, která by zaručovala integritu dat a není tak systémově možná.

Další možností, jak označení organizace řešit bylo **Obohacování zpráv přímo v IPFIXcol2**

Tato varianta řešení zahrnovala upravit samotný výstupní plugin IPFIXcol2 tak, aby při exportu dat do společného Kafka topicu doplnil ke každému záznamu unikátní identifikátor organizace ve formě metadat. Například se jednalo o přidání nového pole s názvem *organization_id*. Za tímto účelem by ovšem byl zapotřebí vývoj a údržba vlastního rozšiřovacího pluginu pro Kafka Exporter. Tato možnost byla v závěru vyhodnocena jako technicky náročná, špatně udržovatelná a náchylná k chybám, především u aktualizací IPFIXcol2 z hlediska nekompability.

Variantou, která byla na konci těchto diskusí zvolena, bylo **vytvoření vlastních dedikovaných Apache Kafka topiců pro každou organizaci,** do kterých budou směřovat data z individuálních instancí IPFIXcol2. Tato varianta má hned několik zásadních výhod:

- **Jednoznačná identifikace původu dat**
- **Zlepšené škálování**

  Přidání nové organizace by znamenalo pouze vytvoření nového topicu bez nutnosti, jakkoliv upravovat stávající logiku
- **Modularita Spark jobů**

  Tento přínos byl v budoucnu velice užitečný, jelikož normalizační a agregační Spark joby mohly být nakonfigurovány pouze na konkrétní topicy, což umožnilo paralelní zpracování dat podle organizací.
- **Možnost nastavení individuálních retencí u topiců**

  Jelikož různé organizace posílají rozdílné objemy dat je vhodné mít možnost si nastavit individuálně kolik se může do jednotlivých topiců maximálně ukládat dat.

U organizací s větším objemem si tak jednoduše lze nastavit větší retenci pro uchovávání většího počtu záznamů.

Důležitou částí návrhu bylo také vymyšlení jednoznačné jmenné konvence pro Kafka topics, která byla stanovena následovně.

<typ_dat>_<formát>_<organizace>

například *flowmon_json_organizace1, suricata_json_organizace2*

Tato konvence také umožňuje při zpracovávání agregačních skriptů jednoduše rozlišovat o jaký stream dat se jedná.

Kafka broker běží na vlastním serveru, který zároveň hostuje i komponenty pro monitoring jako Prometheus, Kafka UI a Zabbix agent, což umožňuje efektivní dohled nad aktuálním stavem všech datových toků v reálném čase.

3.3.3 Bezpečnost přenosu

Bezpečnost přenosu dat z jednotlivých organizací do centrálního systému je řešena prostřednictvím šifrovaných IPsec tunelů, které zajišťují důvěrnost a integritu těchto dat. IPsec (Internet Protocol Security) je sada standardů, definovaných v rámci IETF, která umožňuje bezpečnou komunikaci na úrovni síťové vrstvy prostřednictvím šifrování a digitální autentizací IP paketů. V rámci minulého projektu již bylo rozhodnuto, že každá organizace bude mít zřízen samostatný tunel. Tunely jsou v naprosté většině případů navazovány prostřednictvím komunikační infrastruktury veřejné správy (KIVS/CMS), která poskytuje chráněné propojení mezi státními institucemi. IPsec tunel zajišťuje, že datový provoz mezi organizací a systémem detekce není přenášen v otevřené podobě přes veřejný internet, ale prochází šifrovanou a kontrolovanou cestou, jejíž koncové body jsou striktně určeny.

3.4 Normalizace dat

3.4.1 Účel a význam

Další klíčovou fází při zpracovávání dat v novém návrhu systému detekce je normalizace dat. Pod normalizací si v tomto případě můžeme představit především sjednocení formátu, struktury a obsahu vstupních dat ze zdrojů Flowmon a Suricata. Výsledkem této fáze by měla být specificky připravená data, která budou vhodná pro další efektivní zpracování, kterým je zde myšlena následná agregace.

Původní systém detekce byl navržen tak, že sbíral veškerá data bez jakéhokoliv předchozího výběru polí a předzpracování. V praxi to znamenalo, že všechny záznamy, které organizace generovaly a následně poskytovaly byly uloženy do systému bez ohledu na jejich analytickou hodnotu. Počítalo se s tím, že jakákoliv data by mohla být potenciálně užitečná do budoucna. Některé typy záznamů přesto však nikdy nebyly využity. Výsledkem byl výrazný objem mnohdy redundantních dat, který k ničemu nesloužil. Takový objem dat znamenal nejen nadměrné zatížení uložiště Splunku, ale také vedl k podstatnému zpomalení veškerých SPL dotazů.

Při návrhu nového systému detekce byla proto jedním z prvních kroků příprava analýzy, která zahrnovala identifikaci všech dostupných polí ze všech typů IPFIX záznamů. Tyto druhy se mohou od sebe lišit a poskytovat svá unikátní pole. Typy záznamů, které byly analýzou identifikovány byly především TLS, HTTP, DNS, AAAA, CNAME, NS, MX, PTR, HTTPS, TXT, SRV, SOA, CON.

Analytici z oddělení síťového provozu ve spolupráci vytvořili na začátku návrhu přehledný Excel soubor, ve kterém byla definována všechna dostupná pole, která se vyskytovala v záznamech poskytnutými organizacemi. Na základě tohoto dokumentu probíhaly detailní diskuse členů oddělení, při kterých bylo cílem praktické posouzení analytických potřeb. Analytici při výběru polí postupovali metodou kladení reálných otázek k forenzní analýze. Ptali se, jaká konkrétní data jsou nezbytně nutná pro provedení efektivní síťové forenzní analýzy nebo Threat huntingu. Zvažovali, jaké typy informací mohou i do budoucna usnadnit vyšetřování incidentů nebo urychlit jejich objasnění. Ovšem základním a hlavním cílem v této fázi bylo selekcí **výrazně zredukovat celkový objem ukládaných dat, při zachování jejich maximálního analytického potenciálu.**

3.4.2 Implementace

Implementace normalizace dat probíhá prostřednictvím aplikace Apache Spark, a to v rámci specializovaných Spark jobů, kterými jsou:

- *Spark-submit-flowmon_avro*
- *Spark-submit-suricata_avro*

Oba tyto joby provádí hned několik velice důležitých úkolů, bez kterých by nemohla data být dále agregována efektivně. Konkrétně se jedná o úkoly:

- **Parsování vstupních dat z JSON formátu**, která přichází z Apache Kafka topiců.
- **Jednotné přejmenování původních polí**
- **Sjednocení formátu IP adres**
- **Normalizace časových údajů (timestamps)**, aby byla zachována konzistence časových údajů.
- **Odstranění redundantních a nepotřebných polí**

U předchozího systému i u návrhu nového systému také docházelo občas ke zaznamenání datových anomálií. Takové anomálie mohly mít různou podobu, ale nejčastěji se jednalo o chybějící hodnoty v povinných polích, nesprávně formátované IP adresy nebo nekompletní záznamy. I když se jedná o velice vzácné případy, mohou mít tyto anomálie velký vliv na integritu celkových dat. Řešení takto hraničních záznamů je momentálně řešeno tak, že dochází pouze k jejich zápisu do dedikovaného kafka topicu *avro_invalid,* kde jsou jednoduše ukládány. Nad těmito daty dále neprobíhá analýza, ale zároveň je nechceme kompletně ignorovat. I když mohou některá pole v těchto záznamech být chybná, neznamená to, že jsou pro nás kompletně neužitečná. Stále mohou obsahovat některá IoC, která v případě proběhlého incidentu lze zpětně dohledat.

3.4.3 Spark job flowmon_avro

Veškerá normalizace, která byla popsána v předchozí části je realizována právě pomocí Spark jobu flowmon_avro. Jedná se o skript, jehož cílem je převod IPFIX surových síťových záznamů získaných z Kafka topiců jednotlivých organizací do normalizované podoby ve formátu Spark Avro.

Technická konfigurace

Spark Job je spuštěn jako samostatný kontejner prostřednictvím platformy Podman, konkrétně příkazem spark-submit. Jelikož tento skript byl v průběhu a vývoje a testování mnohokrát změněn, bylo nutné uchovávat jednotlivé verze v privátním GitLab repozitáři, které si oddělení analýzy síťového provozu samo spravuje.

Všechny normalizační i agregační Spark joby, běží na dedikovaných serverech. Na každém serveru je provozován samostatný Spark cluster v tzv. standalone režimu. To znamená, že na jednom Spark serveru běží instance spark-master, spark-worker a zároveň jeden nebo více spark-submit kontejnerů, které obsahují jednotlivé Spark joby.

Samostatný Spark cluster byl zvolen proto, že všechny Spark joby byly vyvíjeny v jazyce Python, pro který není plně podporováno sdílení mezi více nativními Spark clustery v klasickém distribuovaném režimu.

Výpočetní zdroje jako množství RAM nebo počet CPU jader pro jednotlivé Spark servery byly nastaveny manuálně během testovací fáze. Tyto zdroje byly individuálně upravovány na základě požadavků na výkon u jednotlivých jobů. Po dostatečném odladění již nebylo nutné tyto parametry dále upravovat.

Konfigurace pomocí souboru .env a proměnných

Všechny servery i jednotlivé joby jsou definovány a parametrizovány pomocí jednotného *.env* souboru. Tento soubor zahrnuje všechny technické informace o jednotlivých strojích, jobech, vstupních a výstupních topicích v Apache Kafka, názvech aplikací, definicích míst, kde se mají ukládat historické logy a mnoho dalšího. Soubor slouží pro účely centralizace všech konfigurací. Soubor je opět spravován v rámci repozitáře na GitLabu. Na následujícím obrázku je zobrazena vybraná část konfigurace serveru, na kterém běží normalizační skript flowmon_avro. Je zde vidět alokované výpočetní zdroje pro tento stroj, normalizační skript, který je na tomto stroji provozován a názvy jeho vstupního a výstupního kafka topicu.

```
####  tst003 (Spark2)
## Spark
SPARK2__VERSION=x
SPARK2__SPARKUI_EXTERNAL_PORT=x
SPARK2__WORKER_CORES=23
SPARK2__WORKER_RAM=28G
SPARK2__SCRIPTS=./spark_scripts
SPARK2__EVENTS=./data/spark-history-server
SPARK2__HISTORY_EXTERNAL_PORT=x

SPARK2__FLOWMON_AVRO_APP_NAME=flowmon_avro
SPARK2__FLOWMON_AVRO_SCRIPT=flowmon_avro.py
SPARK2__FLOWMON_AVRO_INPUT=^flowmon_json_.*
SPARK2__FLOWMON_AVRO_OUTPUT=flowmon_avro
SPARK2__FLOWMON_AVRO_INVALID=avro_invalid
SPARK2__FLOWMON_AVRO_CORES=10
SPARK2__FLOWMON_AVRO_RAM=6G
```

Obrázek č. 10: Ukázka části konfiguračního .env souboru pro Spark aplikaci
(Zdroj: Vlastní zpracování)

Průběh zpracování dat

Skript samotný je navržen tak, aby běžel dlouhodobě a nepřetržitě, jedná se tak tedy o Spark Structured Streaming aplikaci. Skript průběžně čte vstupní data z Kafka topiců ve formátu JSON, provádí jejich transformaci a následně provádí serializaci do **Avro formátu**. Oproti dávkovému zpracování je tento job určen k nepřetržitému běhu a zpracovává zprávy hned jak jsou dostupné, bez jakéhokoliv přerušení časových intervalů.

Na rozdíl od agregačních jobů, o kterých bude řeč v následující kapitole, neprovádí tento skript žádné seskupování v časových oknech. Jeho cílem je pouze normalizovat různé datové struktury do jednotné struktury. Po načtení dat z Kafka topiců jednotlivých organizací, dochází k první transformaci, kdy dochází ke spojení *sourceIPv4Adress* a *sourceIPv6Adress* do jednotného pole s názvem *src_ip*. Stejně tak dochází k úpravě u *dst_ip* u cílových adres. Jsou zde také sjednoceny transportní porty, protokoly nebo identifikátory IP verze. Tato základní pole jsou také v tento moment doplněna o časové značky začátku a konce spojení.

Jednou z klíčových částí normalizace je logika určení směru spojení. Vychází se zde z porovnávání hodnot *src_port* a *dst_port*. Pokud zde platí podmínka:

dst_port ≤ *src_port* předpokládá se, že se jedná o výstupní spojení, tedy směr ven. Na základě této logiky jsou potom odvozována následující pole, která pracují se směry: *bytes_out, bytes_in, packets_out, packets_in, flow_out_count, flow_in_count, tcp_flags_out, tcp_flags_in.*

Charakteristickou vlastností IPFIX záznamů je využívání tzv. *uniflow* modelu. Každý směr od jednoho koncového bodu ke druhému bývá zaznamenáván jako samostatný tok. V případě obousměrné komunikace tak typicky dochází k vytvoření dvou samostatných záznamů i přes to, že ve skutečnosti jde o jeden logický tok. Dalším specifikem IPFIX je způsob exportu dat. Exportér může v závislosti na konfiguraci rozdělovat záznamy o dlouhotrvajících tocích na více částí podle časového intervalu. Výsledkem je, že opět jeden logický tok může být reprezentován více záznamy rozptýlenými v čase. Pro účely analýzy je nezbytné provést agregační transformaci která tyto dílčí záznamy sloučí do jednoho. Součástí normalizační fáze je tak i sjednocení takto dělených záznamů. Tím dojde k odstranění redundance způsobené dvojitým zaznamenáváním stejné komunikace akorát v opačném směru.

U DNS části zprávy dochází k extrakci a dekódování odpovědí, včetně odvození IP adres, doménových jmen nebo MX poštovních záznamů. TLS metadata jsou zde obohacována o otisk *ja3* a doménu *dst_host*, která jde extrahovat z TLS handshake nebo HTTP hlavičky. K označení zdroje této informace zde slouží pole *dst_host_seen*, které může mít hodnotu 1 pokud se jedná o TLS nebo 2 pokud se jedná o HTTP.

Na úrovni tohoto skriptu také probíhá validace dat. Výsledky skriptu jsou ověřovány proti schématu, které se nachází v komponentě Kafka Schema Registry. Pouze záznamy, které odpovídají struktuře schématu mohou být serializovány do Avro formátu a poslány do další fáze zpracování. Tímto způsobem je tak zajištěna značná míra integrity celého datového toku.

Obrázek č. 11: Ukázka IPFIX záznamu ve formátu JSON
(Zdroj: Vlastní zpracování)

Výstup jobu

Po normalizaci, schématické validaci a serializaci do binárního formátu Avro, jsou data odeslána do jednotného výstupního Kafka topicu, který se nazývá stejně jako skript, tedy

flowmon_avro a je společný pro všechny organizace. Tato volba byla učiněna záměrně, jelikož v této fázi se již data nachází v jednotném formátu. To umožňuje efektivní zpracování v navazujících agregačních Spark jobech, bez nutnosti jakéhokoliv dalšího dělení topiců. Tento výstupní Kafka topic slouží výhradně jako vstupní bod pro agregační Spark joby, které na jeho základě provádějí výběr relevantních polí, sumarizace, obohacování dat a další operace.

3.4.4 Spark job suricata_avro

Dalším normalizačním Spark jobem je skript suricata_avro. Tento skript slouží k transformaci aplikačních událostí ze systému Suricata do sjednoceného formátu, který odpovídá struktuře zbytku datových toků. Skript nepřebírá síťová data typu IPFIX, ale zpracovává všechny síťové události včetně protokolu tls a http, které generuje nástroj Suricata.

Technická konfigurace

Stejně jako Flowmon_avro je i tento job provozován nepřetržitě uvnitř samostatného kontejneru v Podman. Konfigurace skriptu je opět řešena pomocí proměnných definovaných ve sdíleném .env souboru.

Průběh zpracování dat

Na rozdíl od IPFIX dat, jsou Suricata záznamy zapisovány do vnořených struktur, které jsou ve většině případů víceúrovňová. U Suricata záznamů je tak nutné jako první krok provést zploštění (tzv. flattening) těchto vnořených struktur. V rámci tohoto skriptu tak tedy dochází k následujícím úkolům.

- **Extrakci a přejmenování polí do standardizované podoby**
- **Výpočet směrových metrik**, kde se jedná o stejný postup jako tomu bylo u Flowmon_avro, což zahrnuje porovnávání portů.
- **Převod časových údajů do UNIX Timestamp podoby**
- **Extrakce doménového jména z pole SNI (TLS) nebo Hostname (HTTP)**
- **Validace a příprava pro serializaci do formátu Avro**

Zásadní rozdíl oproti IPFIX spočívá ve způsobu reprezentace síťových toků. Zatímco u IPFIX dat může být jeden síťový tok rozdělen na několik dílčích záznamů, Suricata pro každý detekovaný tok vytváří právě jeden záznam typu *biflow*. Tento záznam zahrnuje

celý průběh obousměrné komunikace mezi dvěma koncovými body bez ohledu na jejich délku nebo objem přenesených dat. Proto není v rámci normalizační fáze zapotřebí provádět agregaci více záznamů do jednoho, jak tomu je u IPFIX. Úloha této fáze v případě Suricata dat tak spočívá hlavně ve strukturálním sjednocení. Objem dat zůstává nezměněn, protože každému vstupnímu záznamu nadále odpovídá jeden výstupní záznam.

Obrázek č. 12: Ukázka Suricata záznamu ve formátu JSON
(Zdroj: Vlastní zpracování)

Výstup jobu

Jakmile je dokončena transformace dojde zde opět k validaci proti schématu ze služby Kafka Schema registry a následné serializaci do Avro formátu. Tyto binární data jsou zaslána do Kafka výstupního topicu s názvem suricata_avro. Tento topic je poté využíván v dalších krocích, konkrétně se jedná o agregaci, převod zpět do formátu JSON a navazující indexaci přímo do Splunku.

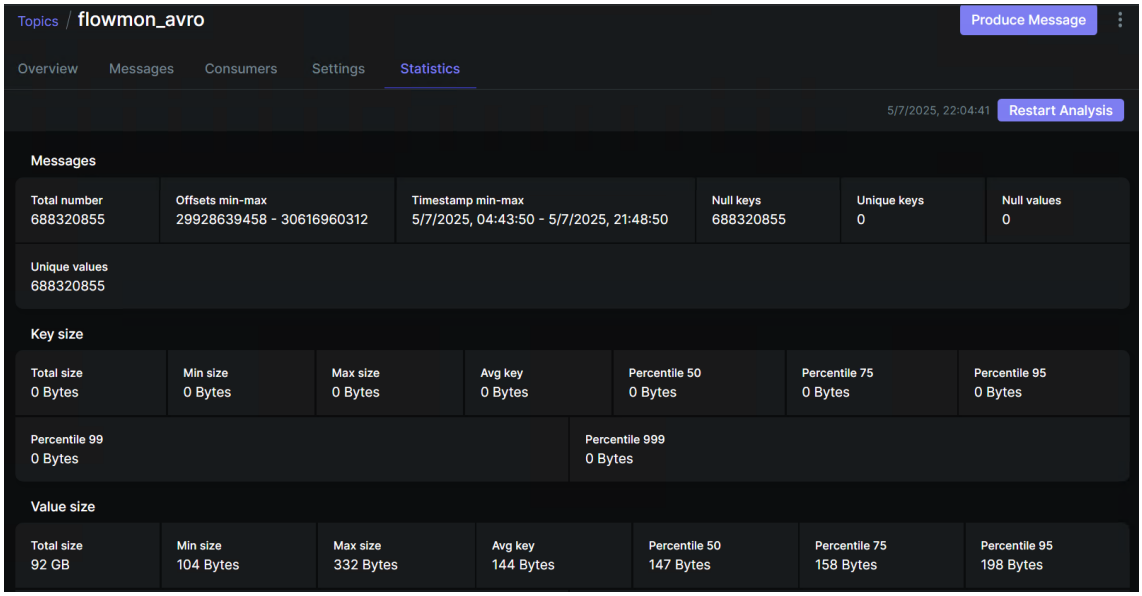
3.4.5 Objem dat po normalizaci

Jelikož jedním z hlavních cílů normalizační fáze je zredukovat objem dat, zaměří se tato část práce na vyčíslení úspory a porovnání před a po dokončení této normalizace. Úspora je zde doložena statistickými údaji získanými z monitorovacího nástroje Prometheus, který monitoruje objem vstupních dat pro jednotlivé organizace. Dále je podložena údaji z grafického rozhraní Apache Kafka, které poskytuje statistické údaje o výstupním topicu flowmon_avro.

Objem přijatých dat ze všech organizací (1 den)

Na základě statistik z Promethea, který je schopen sumarizovat počet *bytes_in* z jednotlivých topiců *flowmon_json_organizace* a *suricatasyslog_organizace* lze zjistit, že **Denně přichází do systému více než 400 GB dat** z jednotlivých IPFIX a Suricata exportérů.

Objem po normalizaci



Obrázek č. 13: Statistiky topicu flowmon_avro v Apache Kafka UI
(Zdroj: Vlastní zpracování)

Z Kafka UI pro výstupní topic flowmon_avro vyplývá, že za dobu měřeného intervalu, bylo vygenerováno celkem 688 320 855 zpráv o celkovém objemu 92 GB. Je zde ale nutné zmínit, že generovaný objem zůstává v přepočtu na celý den prakticky stejný, neboť provoz v nočních hodinách je provoz většiny zapojených organizací výrazně nižší. Kromě celkového objemu zpráv je v Kafka UI také viditelná průměrná velikost jednoho záznamu, která činí 144 bajtů. Pokud bychom tuto hodnotu srovnávali s původním JSON formátem, kde záznam mohl běžně dosahovat velikosti 400-600 bajtů, ihned vidíme že Avro formát výrazně přispívá k redukci objemu.

Pro výpočet úspory využijeme následující vzorec:

$$Úspora = \left(1 - \frac{Objemdatponormalizaci}{Objemdatpřednormalizací}\right) \times 100$$

Celkové porovnání úspor pro tuto fázi lze vyjádřit v následující tabulce:

Tabulka č. 3: Vyčíslení úspory objemu dat po normalizační fázi
(Zdroj: Vlastní zpracování)

Metrika (1 den)	Hodnota
Počet zpráv (normalizovaný topic)	≈ 688 320 855 zpráv
Objem vstupních dat před normalizací	≈ 400 GB
Objem dat po normalizaci	92 GB
Relativní úspora	≈ 77 %

Z tabulky lze vidět, že **samotná fáze normalizace umožnila zredukovat objem dat o více než 75 %** ještě před fází samotné agregace.

3.5 Agregace dat

Abychom dosáhli ještě větší redukce objemu dat, byla zavedena navazující agregace. Tato fáze je realizována třemi hlavními agregačními úlohami v rámci aplikace Apache Spark, které zpracovávají události z Kafka topicu připraveným normalizačním Avro skriptem.

Cílem této fáze bylo vytvořit „zhutněný“ výstup dat, který by snížil redundanci. Tím by došlo k ještě větší úspoře objemu dat a v této formě by již mohla být zaindexována do Splunku.

3.5.1 Spark job flowmon_conn_1h

Jako jeden ze zvolených parametrů, podle kterého bychom mohli data agregovat, byl čas. Základním cílem tohoto skriptu je tedy vytvářet pravidelnou časovou agregaci síťových událostí na úrovni jednotlivých spojení. Jak již z názvu vyplývá, agregace zde probíhá v hodinových intervalech. Je zde stále prioritou udržet vysokou informační hodnotu. Tento skript jako jediný je v rámci celé infrastruktury systému také dostupný v grafickém rozhraní Spark UI, což umožňuje jeho průběžné sledování výkonu pro identifikaci slabých míst. Agregace hraje v celkovém systému opravdu klíčovou roli. Skript *flowmon_conn_1h* se stal nejen technicky ale i operačně nejsložitější částí implementace celé datové pipeline.

3.5.2 Struktura a princip zpracování dat

Vstupní data z normalizovaného Kafka topicu ve formátu Apache Avro jsou přijímána skriptem *flowmon_conn_1h*. Po načtení těchto binárních zpráv, dochází k jejich deserializaci pomocí funkce *from_avro()* s využitím schématu načteného ze Schema Registry. Z takto upravených dat je následně vybrán pouze definovaný seznam relevantních polí, která jsou označena jako důležitá. Ty představují minimální kontext pro smysluplnou agregaci. Tato pole zahrnují atributy vyjádřené v následující tabulce.

Tabulka č. 4: Důležitá pole určená ke agregaci
(Zdroj: Vlastní Zpracování)

Pole	Popis
Src_ip	Zdrojová IP adresa zařízení, které zahájilo spojení
Dst_ip	Cílová IP adresa zařízení, na které bylo spojení navázáno
Src_port	Port zdrojového zařízení
Dst_port	Port cílového zařízení
Org	Identifikátor organizace
Proto	Protokol transportní vrstvy podle IANA
Packets	Celkový počet paketů přenesených v rámci spojení
Bytes	Celkový objem přenesených dat v bajtech (součet obou směrů)
Bytes_in	Bajty přijaté cílovou stranou (vstupní směr komunikace)
Bytes_out	Bajty odeslané zdrojovou stranou (výstupní směr komunikace)
Packets_in	Pakety přijaté cílovou stranou
Packets_out	Pakety odeslané zdrojovou stranou
Tcp_flgs	Určení TCP stavu napříč celým spojením (SYN, ACK, FIN atd.)
Ipv	Verze IP protokolu (IPv4, IPv6)
Flow_start	Čas začátku spojení
Flow_end	Čas konce spojení

Flow_in_count	Počet detekovaných příchozích toků
Flow_out_count	Počet detekovaných odchozích toků
Flow_count	Celkový počet toků detekovaných pro danou kombinaci adres, portů a protokolu.
Tcp_flags_in	Agregované TCP stavy příchozích toků
Tcp_flags_out	Agregované TCP stavy odchozích toků
Kafka_timestamp	Časová značka přijetí události do Kafky
Dst_host	Doménové jméno cílového serveru
Dst_host_seen	Indikátor zdroje doménového jména
Ja3	TLS otisk klienta ve formátu JA3

Z těchto atributů je věnována zvýšená pozornost u pole Kafka_timestamp, které vyjadřuje čas kdy událost byla přijata do Kafka systému. Bez tohoto pole by nebylo možné udělat časovou osu událostí. Timestamp totiž slouží jako základní referenční čas pro časová okna, která jsou zde nastavena na 1 minutu. Referenčním časem je tak tedy možno správně řídit agregaci, a Spark jobu umožňuje ignorovat události, přicházející po definovaném zpoždění.

Takto definovaná a načtená data jsou připravena na časové **seskupení na základě kombinace hodnot zdrojové a cílové IP adresy, portů, protokolu a identifikátoru organizace**.

3.5.3 Definice agregační logiky

Hlavní část skriptu *flowmon_conn_1h* tvoří agregační blok, který slouží k sumarizaci síťových událostí ve zvoleném časovém okně. Samotná agregace je implementována pomocí metody *groupBy()*, kde jsou použity následující parametry:

- **window (kafka_timestamp, Window_LEN)**, které představuje časové okno
- ***src_ip, dst_ip***
- ***src_port, dest_port,***
- ***proto***
- ***org***

Tato kombinace parametrů zajišťuje, že každý agregovaný záznam odpovídá jedinečné síťové komunikaci mezi dvěma koncovými body v daném směru a časovém rámci.

Následně jsou pomocí agregačních funkcí vypočteny metriky pro každou z těchto kombinací, přičemž nejvýznamnější z nich jsou následující.

Součtové agregace *sum():*
flow_count, flow_in_count, flow_out_count, bytes**,** bytes_in, bytes_out, packets, packets_in, packets_out.

Bitové operace:
tcp_flags, tcp_flags_in, tcp_flags_out.
Výstupem těchto operací je bitový součet všech TCP stavů za dané časové okno.

Minimální a maximální hodnoty:
flow_start (min), flow_end (max). Tyto hodnoty spolu zachycují celou délku trvání spojení
ipv (max). Skript volí vyšší IP verzi

Množinové agregace *collect_set()*:
ja3, dst_host.

Vytváří se zde seznam unikátních otisků ja3 a domén použitých v rámci spojení.

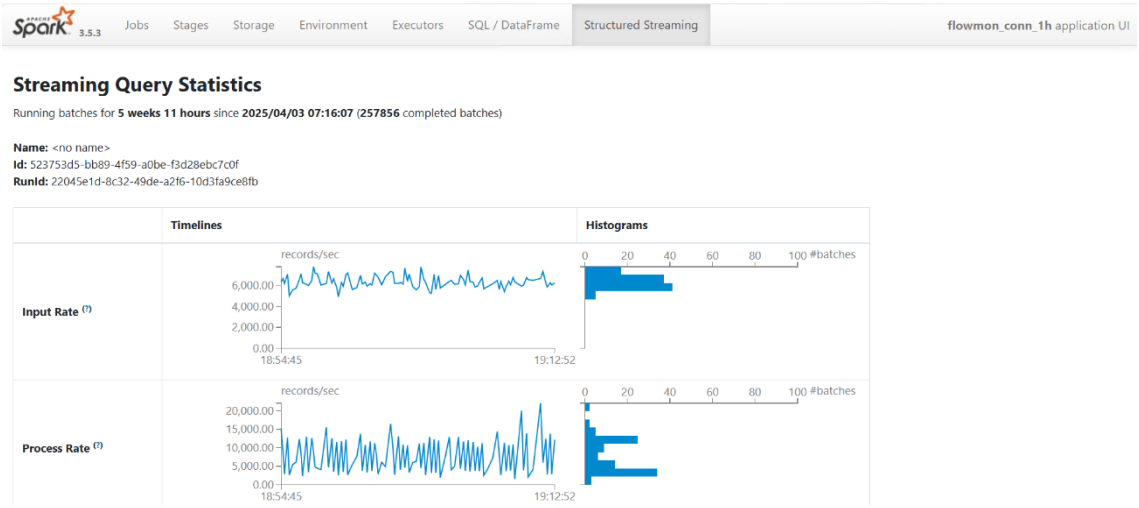
Výsledkem celé agregace je dataframe, ve kterém jsou sloupce *window.start* a *window.end* konvertovány na UNIX timestampy a uloženy do samostatných polí *window_start* a *window_end.* Samotný sloupec window je následně odstraněn, čímž se výsledná struktura zploští.

3.5.4 Výstupní serializace a zápis do Kafka topicu

Aby bylo možno výsledná data dále zpracovávat v rámci navazujících částí systému, je třeba je opět převést zpět do Avro formátu, který je kompatibilní se zbytkem infrastruktury. Data v binárním formátu jsou odeslána do Apache Kafka, kde jsou zapsána do topicu *flowmon_conn1h_avro*. Zápis do tohoto topicu probíhá průběžně, jakmile jsou výsledky časových oken dopočítávány. Pomocí tohoto návrhu je zajištěno, že datový tok je konzistentní. Zároveň zachováním Avro formátu až do poslední fáze agregace umožňuje flexibilně přepínat mezi binární a textovou reprezentací podle aktuálních potřeb.

3.5.5 Monitoring skriptu

Monitoring tohoto skriptu je realizován hned pomocí tři nástrojů, kterými jsou Apache Spark UI, Kafka UI, a Prometheus. Pomocí Spark UI je možno sledovat detailní metriky o samotném běhu streamovací aplikace. Je tak jednoduché provádět diagnostiku problémů, které by se mohly vyskytnout.



Obrázek č. 14: Spark UI pro aplikaci flowmon_conn_1h
(Zdroj: Vlastní zpracování)

3.5.6 Agregace Suricata

Vedle primárního využití skriptu *flowmon_conn_1h* pro agregaci IPFIX síťových událostí je stejný Spark job nasazen pro agregaci Suricata záznamů. V tomto případě se jedná o samostatnou instanci, která běží na dedikovaném serveru a slouží k agregaci záznamů ze systému Suricata. V této instanci skript načítá normalizovaná data z Kafka topicu *suricata_avro* a zapisuje výstup do samostatného Kafka topicu *suricata_conn_1h*.

I přesto, že datová struktura se liší od IPFIX záznamů, skript využívá stejnou agregační logiku. Seskupuje záznamy podle *src_ip, dst_ip, src_port, dst_port, proto, org* a časového okna a také počítá metriky již jmenované v části o *flowmon_conn_1h.*

Z technického hlediska tedy není nutné oddělovat agregaci IPFIX a Suricata záznamů do dvou různých instancí. Obě vstupní datové struktury jsou po normalizaci kompatibilní a zpracovatelné stejným skriptem. V praxi se ale oddělení ukázalo jako velice přínosné z provozního hlediska. Datová zátěž se totiž u obou typů záznamů výrazně liší. Suricata zpráv je méně ale jsou datově bohatší což má za důsledek, že *suricata_avro* topic má

zcela odlišný počet přísunu zpráv než *flowmon_avro*. Oddělením těchto dvou Spark jobů je umožněna lepší škálovatelnost, ladění výkonu, přidělování výpočetních zdrojů.

Jelikož je pomocí tohoto skriptu možno zpracovávat Suricata záznamy stejným způsobem jako IPFIX data, dochází ke sjednocení obou typů těchto záznamů v jedné struktuře bez ohledu na původní zdroj. To je velice výhodné pro výsledné vyhledávání ve Splunku.

3.5.7 Agregace DNS

Dalším agregačním skriptem je flowmon_avro_dns_cache.py. Tento skript je určen pro specifický typ agregace veškeré DNS komunikace. Jeho primární cíl není sumarizace metrik provozu, ale vytvoření efektivní mezipaměti DNS, která bude indexována ve Splunku a bude sloužit k obohacování analyzovaných dat o doménový kontext.

Agregační logika

Na vstupu skript přijímá události ze stejného Kafka topicu jako flowmon_conn1h skript, tedy flowmon_avro, ve kterém se nacházejí normalizované záznamy ve formátu Avro. Tento skript na rozdíl od conn1h, neprovádí agregaci spojení, ale soustředí se především na **extrakci a agregaci DNS dotazů a odpovědí**, která je součástí těchto záznamů.

Prvním krokem je zde filtrování pouze relevantních typů odpovědí, kterými jsou A, NS, CNAME, PTR, MX, AAAA. Tím se výrazně snižuje objem zpracovávaných zpráv a soustředí se pouze na doménové překlady. Následuje krok seskupení dat v časových oknech podobně jako ve flowmon_conn1h, ale tentokrát na základě polí:

- ***window(kafka_timestamp, WINDOW_LEN)***
- ***dns_query_name***
- ***dns_answer_data***
- ***dns_answer_type***
- ***dns_answer_octetbytes***
- ***dns_answer_packets***
- ***dns_answer_record_count***
- ***dns_authority_record_count***
- ***dns_additional_record_count***
- ***dns_question_count***

Pro každé takové seskupení jsou následně vypočítány agregační metriky, které jsou základ pro budování mezipaměti DNS.

Tabulka č. 5: Přehled agregačních metrik DNS mezipaměti
(Zdroj: Vlastní zpracování)

Metrika	**Popis**
count_dns_request	Počet dotazů na danou kombinaci doména-IP v daném okně
dns_ttl_min dns_ttl_max	Minimální a maximální hodnota TTL v odpovědích
flow_min_start flow_max_end	Rozsah časového výskytu v síti
org	Množina organizací, které dotaz zaznamenaly
unique_dns_servers unique_dns_clients	Počet unikátních IP adres zdrojů a cílů dotazů
dns_id_unicount dns_ttl_unicount	Počet unikátních identifikátorů DNS zpráv a TTL hodnot

Výsledná data jsou poté serializována do formátu Avro a zapsána do Kafka topicu *flowmon_avro_dns_cache*. Stejně jako ostatní výstupy jsou následně převedena pomocí Spark jobu *avro_to_json.py* do formátu JSON a odeslána do Splunku k jejich indexaci.

Využití výstupu

Data produkovaná tímto jobem nejsou určena pro přímou forenzní analýzu, ale slouží především jako zpětná doménová mezipaměť. V praxi to znamená, že v okamžiku, kdy se v jiných částech datového toku objeví IP adresa, je možné prostřednictvím této DNS mezipaměti zpětně dokázat, jaká doména byla k dané IP adrese v daném čase přiřazena.

3.5.8 Shrnutí agregační fáze

Agregační vrstva je v systému navržena tak aby momentálně fungovala na základě tří samostatných Spark jobů, které zpracovávají rozdílné typy datových toků, kterými jsou IPFIX záznamy, Suricata záznamy a DNS dotazy a odpovědi. Toto rozdělení nám umožňuje flexibilní přidávání dalších typů dat do budoucna, individuální monitoring a jednoduchou správu každé instance. Výstupní agregovaná data jsou základem pro následné indexování do systému Splunk.

3.6 Integrace se Splunkem

Poslední fází architektury systému detekce je samotné napojení na analytickou platformu Splunk.

3.6.1 Převod z Avro do JSON

Než jsou data přeposlána a zaindexována do Splunku, je nutné převést všechny výstupní agregační Kafka topics z Avro formátu do čistého JSON. Za tímto účelem byl vytvořen samostatný Spark job s názvem avro_to_json. Tento job provádí následující sérii úkolů.

- Načítá data ze vstupního Kafka topicu.
- Pomocí shématu ze Schema registry deserializuje zprávu.
- Provede přejmenování polí dle mapy field_aliases.
- Serializuje výstup jako JSON.
- Výsledek zapíše zpět do Kafka topicu určeného pro Splunk

V těchto krocích je zmíněno přejmenování polí dle mapy field_aliases. Tato mapa je jednoduchý seznam, ve kterém se vždy nachází originální název pole a jeho zkrácený alias, která bude použit ve Splunku. Příkladem takového aliasu je převod názvu scr_ip na „i“, bytes_out na „B“, a podobně. Mapovací soubor definuje v této době přibližně 30 klíčových aliasů. Zkrácení samotného názvu polí je tak tedy posledním krokem, který slouží k úspoře dat před indexací.

3.6.2 Připojení ke Splunk

Abychom nemuseli data do Splunku nahrávat ručně v určitých intervalech, je připojení ke Splunku je realizováno pomocí konektoru Splunk Kafka Sink Connector. Přenos dat probíhá pomocí běžného protokolu HTTP, přičemž jednotlivé Kafka zprávy v JSON formě jsou odeslány do Splunku pomocí rozhraní HEC (HTTP Event Collector)

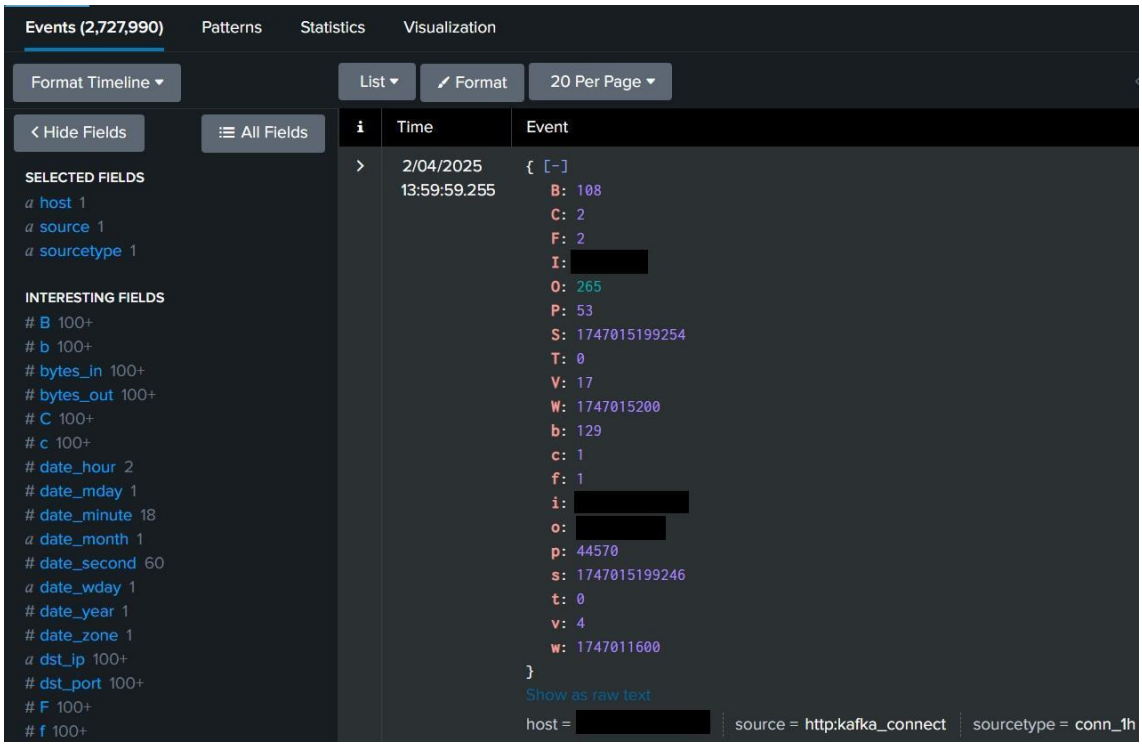
Splunk HEC je standardizovaný vstup, který umožňuje přijímat zprávy z Kafky pomocí HTTP. Konektor samotný zajišťuje, že každá zpráva z Kafka topicu je převedena na požadovaný formát, opatřena cílovými parametry a je následně odeslána do instance Splunku, kde je ihned zpracována a zaindexována. V systému detekce jsou tímto způsobem momentálně napojeny tři hlavní výstupní topicy, kterými jsou flowmon_conn_1h_json, flowmon_avro_dns_cache_json a suricata_conn_1h_json. Pro každý z těchto topiců musí být nakonfigurován samostatný konektor jehož parametry

udávají cílový index ve Splunku, identifikaci zdroje dat, přístupový token a adresu HTTP rozhraní. Konektor běží neustále jako služba připojená ke Kafka Connect REST API.

3.6.3 Indexace ve Splunk

Ve Splunku jsou přijatá data zaindexována do dvou specifických indexů podle svého analytického účelu. Pro všechna agregovaná síťová spojení Flowmon a Suricata byl vytvořen jednotný index nazvaný *conn_1h*. Jelikož mají oba typy dat po agregaci stejnou strukturu, bylo výhodné sloučit je do jednoho indexu. Tímto způsobem je zjednodušeno dotazování pomocí SPL příkazů a vyhledávání napříč záznamy je efektivnější.

Druhý index s názvem dns_cache slouží pro ukládání agregovaných DNS spojení. Tento index v rámci Splunku funguje jako tzv. pasivní DNS, které uchovává historické informace o doménových dotazech a odpovědích. Takové informace mají zásadní vliv na zpětnou analýzu domén a jejich korelace s IoC.



Obrázek č. 15: Ukázka agregace síťového záznamu ve Splunku ze zdroje conn_1h
(Zdroj: Vlastní zpracování)

3.7 Monitoring a dohled

Jelikož je systém detekce tvořen převážně distribuovanou sadou kontejnerizovaných komponent, bylo zde nutné zavést důkladný monitoring. Vzhledem ke kritičnosti celého řešení bylo zapotřebí pokrýt tímto monitoringem všechny vrstvy systému. Zatímco

u vstupních dat bylo zapotřebí sledovat především jejich dostupnost, u normalizace a agregace šlo převážně o výkonnostní a kapacitní parametry.

Realizace monitoringu je v systému provedena kombinací nástrojů Prometheus a Zabbix, které se navzájem doplňují. Prometheus má za cíl především sledovat metriky streamovacích a výpočetních komponent, Apache Kafka a Apache Spark. Zabbix se potom zaměřuje primárně sledování dostupnosti datových toků z jednotlivých organizací a monitoring jednotlivých fyzických serverů na kterých běží kontejnery Podman.

Vedle těchto nástrojů je v systému používáno také webové rozhraní Kafka UI, ve kterém lze sledovat statistiky jednotlivých topics. A dále pak Spark UI, které zase umožňuje sledovat detailní průběh agregačních jobů.

3.7.1 Monitoring Podman kontejnerů

Systém Zabbix je nakonfigurován tak, aby sledoval především dostupnost jednotlivých serverů, využívání jejich výpočetních zdrojů a stav diskového prostoru. Součástí dohledu je také kontrola jednotlivých kontejnerů. Když by mělo dojít k neočekávánému výpadku, kontejnery jsou nastaveny a spouštěny s parametrem *restart: unless-stopped,* který zajišťuje jeho automatické obnovení. Automatické zprávy a upozornění generované na úrovni Zabbixu aktuálně nejsou implementovány v rámci systému, jelikož běžné stavy jsou plně pokryty nastaveným monitoringem a konfigurací restartu v samotném Podmanu.

3.7.2 Monitoring datových toků

Veškeré datové toky jsou pod dohledem systému Prometheus, který běží v samostatném kontejneru Podman na serveru s instancí Apache Kafka. Mezi nejdůležitější sledované metriky v tomto systému patří bytesin_total, která udává objem přijatých dat, a messagein_total, která udává počet přijatých Kafka zpráv. Prometheus tyto metriky sbírá každých 10 sekund a uchovává je až do vyčerpání alokovaného úložného místa. Po zaplnění uložiště dochází k rotaci dat, kdy jsou nejstarší záznamy nahrazeny novými.

3.7.3 Monitoring Spark jobů

Spark joby, které zajišťují agregaci a normalizaci, jsou monitorovány prostřednictvím Spark UI a Spark History Serveru. Spark UI poskytuje přehled o běžících jobech, alokovaných zdrojích, době zpracování a chybových stavech. Abychom mohli chybové stavy analyzovat i zpětně, slouží zde k těmto účelům Spark History Server. Tento server

čte logy, které se ukládají v *tmp* adresáři serveru, kde daná instance Sparku běží. Opět zde platí, že logy jsou uchovávány až do doby, kdy dojde k zaplnění vyhrazeného uložiště a následně dochází k rotaci od nestarších logů až po nové. Aktivní monitoring pomocí Promethea je navíc doplněn o metriky přímo ze Spark aplikace, která má zapnuté rozhraní *spark.ui.prometheus.enabled=true.* To umožňuje sledovat doby zpracování jednotlivých toků, počet vstupních zpráv a výpočetní výkon i pomocí Promethea.

Pomocí správně nastaveného monitoringu má celý analytický tým k dispozici detailní přehled o celkovém zdraví systému. Je zde ale třeba zdůraznit, že jelikož není monitoring v této chvíli plně automatický a negeneruje upozornění, je zásadní, aby byl stav systému pravidelně kontrolován zkušeným členem analytického týmu. Bez tohoto pravidelného dohledu by monitoring ztratil svou funkci a smysl. Proto zde musí být monitoring vždy v kombinaci s lidským faktorem.

3.8 Nasazení a správa systému

Aby byl systém dlouhodobě udržitelný, byla zde zapotřebí systematicky navržená strategie konfigurace, testování, nasazení a správy. Tato podkapitola dává přehled o způsobu, jakým je systém implementovaný v produkčním prostředí, metodiku verzování a jak je řešena správa infrastruktury.

Za účelem spolehlivosti a izolace systémových komponent z provozního hlediska došlo k rozhodnutí, kde pro každou z částí systému bude dedikován samostatný fyzický server. Jednotlivé aplikace jako Apache Kafka nebo Apache Spark tedy fungují na těchto strojích v kontejnerizované formě přes platformu Podman. Toto řešení má hned několik výhod. První z nich je možnost jednoduše přidávat nové servery do infrastruktury. Tím může docházet k horizontálnímu škálování celého systému. Další výhodou je pak individuální správa přiřazených prostředků jako je úložný prostor nebo výpočetní výkon u každého serveru. Každý server je vybaven tak, aby pokrýval specifické nároky dané komponenty. Instance provozující Spark joby jsou vybaveny vícejádrovým CPU a dostatečným množstvím operační paměti, zatímco Apache Kafka je optimalizována pro zápis velkého množství zpráv.

Posledním přínosem je potom možnost přehlednější diagnostiky jednotlivých serverů při případných provozních anomáliích.

3.8.1 Automatizace konfigurace pomocí Ansible

Pro minimalizaci manuálních zásahů do nasazování a konfigurace systémových komponent byl použit nástroj Ansible. Tento nástroj umožňuje definovat požadovaný stav cílových serverů pomocí tzv. playbooků. Jedná se o scénáře, při kterých dochází k automatické distribuci zvolených konfiguračních souborů na definované servery. Pomocí Ansible je možné také připravovat adresářové struktury nebo nastavovat certifikáty pro bezpečnou komunikaci. Důležité je u těchto scénářů nastavování pravidla pro všechny Spark kontejnery, pomocí kterého jde automaticky restartovat jednotlivé aplikace, pokud by došlo k jejich neočekávanému pádu.

3.8.2 Verzování a správa kódu v GitLabu

Verzování celého systému je řešeno prostřednictvím platformy Gitlab. Oddělení analýzy síťového provozu si zde momentálně spravuje svůj vlastní repozitář, který slouží nejen jako centrální úložiště zdrojových kódů, ale také jako nástroj pro řízení celého vývojového cyklu. Každá změna kódu je realizována přes tzv. *merge request.* Tyto požadavky musí projít kontrolou ze strany ostatních vývojářů, kteří do kódu zasahují. Tímto způsobem je zajištěna auditovatelnost každé změny. Kdyby došlo k chybě, dají se takto provedené změny vrátit pomocí historických verzí. Součástí repozitáře nejsou jen samotné Python skripty, ale také veškeré konfigurační soubory pro kontejnery Podman, datová schémata pro Apache Avro včetně jejich integrace se Schema Registry a množství dokumentace, která popisuje závislosti mezi jednotlivými komponentami.

3.8.3 Testování konfigurací a skriptů

Před nasazením jakékoliv verze skriptu do produkce **muselo proběhnout testování v odděleném testovacím prostředí.** Prostředí pro testování bylo vytvořeno tak, aby co nejvíce odráželo konfiguraci skutečného produkčního prostředí. Zahrnuje testovací instance Apache Kafka, Apache Spark i kopii Schema Registry. V tomto prostředí se prováděly testy integrity dat, kdy se zkoumalo, zda se data procházející systémem úspěšně daří serializovat a následně deserializovat v Avro formátu. Byly provedeny testy výpočetní logiky, které ověřovaly správnost výstupních agregací. Vyskytly se zde také průběžné zátěžové testy, kdy se za pomoci simulovaných datových toků ověřovalo, jestli je systém stále schopný zpracovávat data v reálném čase.

Vyhodnocení výsledků probíhalo většinou manuálním porovnáním s referenčními daty. Kromě samotných dat byla testována i stabilita prostředí, včetně zotavení po výpadku.

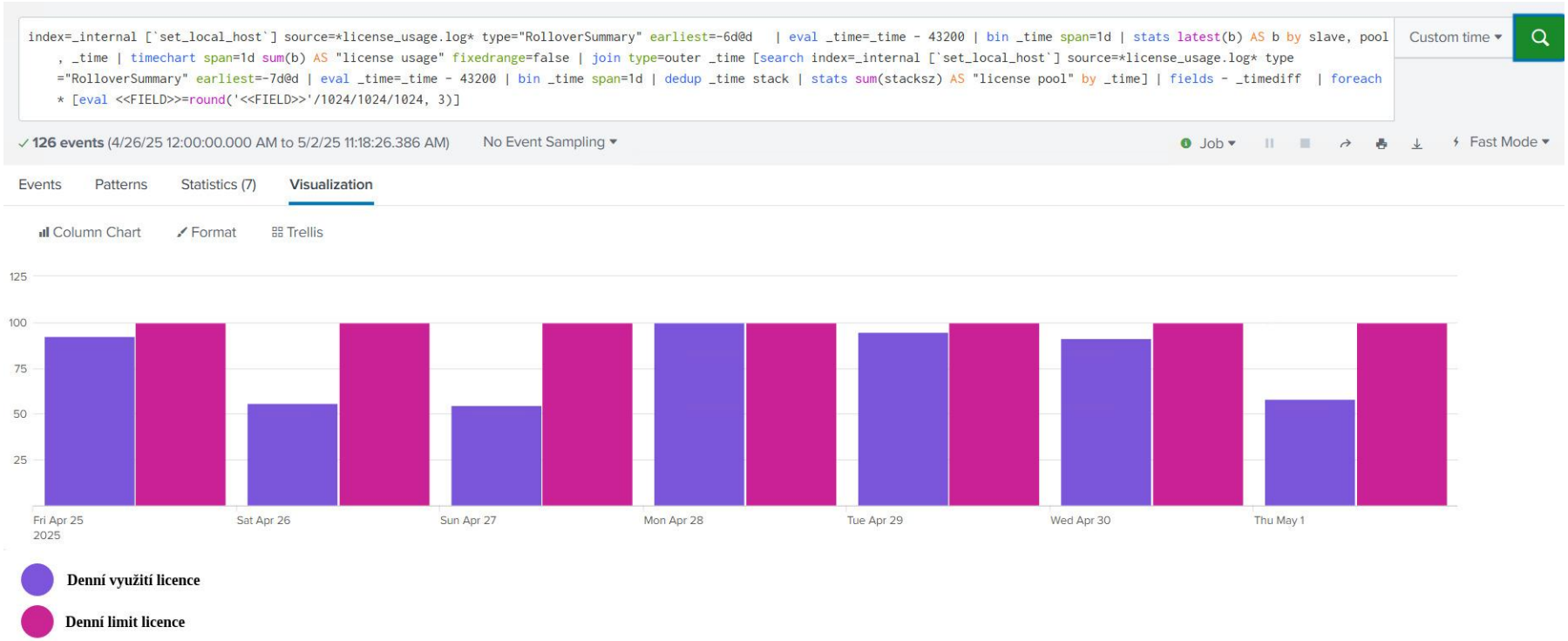
3.9 Možnosti rozšíření

Jelikož je systém od začátku navržen modulárně, umožňuje relativně jednoduché zavedení dalších typů datových zdrojů bez zásadních změn v navržené architektuře. V současné době systém zpracovává data primárně z Flowmon sond a Suricata syslogu. V budoucnu by tedy mohlo být možné začlenit i vstupy například z firewallu FortiGate nebo z platformy Greycortex Mendel. Oba vstupy by bylo možné přidat do normalizační a následně agregační fáze. Takové rozšíření by doplnilo současné analytické možnosti celého systému a zároveň by posílilo schopnosti proaktivní detekce kybernetických útoků.

V rámci rozšíření je také možnost vytvoření nových druhů agregačních skriptů podle délky časových oken. Konkrétním skriptem by mohl být *flowmon_conn_24h*, který by agregoval data po 24 hodinových oknech podle stejné logiky jako současný *flowmon_conn_1h*. Tímto způsobem by data mohla mít zvýšenou retenci a zároveň by se docílilo ještě větší optimalizace úložného prostoru.

3.10 Shrnutí přínosů

Výsledkem realizovaného projektu je efektivní systém pro sběr, zpracování a analýzu síťových dat. V průběhu implementace došlo k výrazné úspoře dat, kdy normalizace a agregace dat pomocí zavedeného Apache Spark snížila objem dat určených pro indexaci ve Splunku. Z původních hodnot přesahujících 400 GB dat za den bylo dosaženo redukce pod hranici 100 GB denně. Díky tomu **bylo možné snížit požadavky na licenci nástroje Splunk z původních 500 GB za den na současných 100 GB za den.** Využívání licence lze průběžně sledovat zadáním dotazu SPL na licenčním serveru Splunk.



Obrázek č. 16: Přehled využívání Splunk licence
(Zdroj: Vlastní zpracování)

Ekonomické zhodnocení projektu vychází z odhadovaných nákladů na vývoj a implementaci systému v období od září roku 2024 do března 2025. Při hodinové sazbě 500 Kč/h, kdy na systému pracovali 3 analytici a celkový odhadovaný časový rozsah činil přibližně 3000 hodin. Celkové náklady na vývoj a implementaci systému tak činí přibližně 1 500 000 Kč. Roční úspora času analytiků se díky efektivnějšímu zpracování dat odhaduje na přibližně 600 hodin, což při stejné hodinové sazbě odpovídá částce 300 000 Kč ročně. Systém byl realizován jako interní zakázka pro předem stanovené potřeby a není proto možné tyto hodnoty dále kvantifikovat.

Podstatným přínosem implementace tohoto systému je také **podstatně levnější provoz a servisní podpora**. Díky architektuře postavené na kontejnerizaci, automatizaci procesů skrze Apache Kafka, Apache Spark a monitoringu prostřednictvím nástrojů Zabbix a Prometheus jsou náklady na údržbu a podporu systému znatelně nižší než u původního řešení.

Posledním dílčím přínosem bylo **výrazné zrychlení vyhledávání nad daty pomocí dotazovacího jazyka SPL ve Splunku.** Reálným odhadem došlo ke zrychlení dotazů v rozmezí o 30-40 % oproti starému řešení. Tento přínos tak dále navyšuje produktivitu celého analytického týmu do budoucna.

4 ZÁVĚR

Cílem této diplomové práce bylo navrhnout a implementovat efektivní systém pro zpracování síťových toků, který by umožnil výrazně snížit objem dat indexovaných do nástroje Splunk a tím by byla zajištěna udržitelnost provozu při přechodu na nižší licenční model. Navržené řešení tento cíl úspěšně splnilo.

Klíčovým prvkem byla implementace agregační logiky pomocí Apache Spark, která pomohla zachovat datovou hodnotu z pohledu reálných analytických scénářů. Zároveň přidanou hodnotou bylo snížení počtu nadbytečných informací a bylo tak zrychleno vyhledávání nad indexovanými daty pomocí dotazovacího jazyka SPL.

Jelikož se jedná o návrh systému složený převážně z open-source nástrojů a vycházející z praktických zkušeností z provozu, je tato metodika snadno přenositelná do prostředí bezpečnostních dohledových center SOC. Může tak sloužit jako vhodný základ pro implementaci podobných řešení i v jiných organizacích, které usilují o efektivní zprávu síťových dat a optimalizaci nákladů.

Výsledný systém byl nasazen v produkčním prostředí Národního úřadu pro kybernetickou a informační bezpečnost a jeho přínos je ověřen v rámci běžného provozu. Momentálně plně splňuje požadavky analytiků z oddělení síťového provozu.

SEZNAM POUŽITÝCH ZDROJŮ

[1] AMPLab. *About AMPLab-Algorithms, Machines, and People Laboratory* [online]. University of California, Berkeley, 2025. Dostupné z: https://amplab.cs.berkeley.edu/about/ [citováno 2025-04-03]

[2] Apache Software Foundation, *Apache Spark Version History* [online]. 2025. Dostupné z: https://spark.apache.org/history.html [citováno 2025-04-03]

[3] Apache Software Foundation. *Monitoring and Instrumentation. Spark Documentation*. [online]. 2025. Dostupné z: https://spark.apache.org/docs/latest/monitoring.html [citováno 2025-04-05]

[4] Bejtlich, R. *The Practice of Network Security Monitoring: Understanding Incident Detection and Response*. San Francisco: No Starch Press, 2013. ISBN 9781593275099.

[5] Carasso, D. *Exploring Splunk: Search Processing Language (SPL) Primer and Cookbook*. San Francisco: CITO Research, 2012. ISBN 9780982550670. Dostupné z: https://www.splunk.com/en_us/pdfs/ebooks/exploring-splunk.pdf

[6] CESNET. *Dokumentace ipfixcol2 – IPFIX Collector Framework*. [online] GitHub, 2025. Dostupné z: https://github.com/CESNET/ipfixcol2 [citováno 2025-03-12]

[7] CESNET. *Oficiální web organizace CESNET*. [online]. 2025 Dostupné z: https://www.cesnet.cz/ [citováno 2025-03-12]

[8] Chambers, B. a Zaharia, M. *Spark: The Definitive Guide: Big Data Processing Made Simple*. Sebastopol: O'Reilly Media, 2018. ISBN 9781491912201. Dostupné z: https://www.oreilly.com/library/view/spark-the-definitive/9781491912201/

[9] Claise, B., Trammell, B. a Aitken, P. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information*. RFC 7011. Internet Engineering Task Force, 2013. [online] Dostupné z: https://datatracker.ietf.org/doc/html/rfc7011 [citováno 2025-03-12]

[10] Costa-Gazcon, V. *Practical Threat Intelligence and Data-Driven Threat Hunting: A hands-on guide to threat hunting with the ATT&CK™ Framework and open source tools*. Birmingham: Packt Publishing, 2012. ISBN 9781838556372. 348 s.

[11] CrowdStrike Inc. *SOC-as-a-Service (SOCaaS)* [online]. 2025. Dostupné z: https://www.crowdstrike.com/en-us/cybersecurity-101/managed-security/soc-as-a-service-socaas/ [citováno 2025-03-26]

[12] CrowdStrike Inc. *Threat Hunting* [online]. 2025. Dostupné z: https://www.crowdstrike.com/en-us/cybersecurity-101/threat-intelligence/threat-hunting/ [citováno 2025-03-26]

[13] González-Granadillo, G., González-Zarzosa, S. a Diaz, R. *'Security Information and Event Management (SIEM): Analysis, Trends, and Usage in Critical Infrastructures'*, *Sensors* 2021, 21(14), s. 4759. Dostupné z: https://doi.org/10.3390/s21144759

[14] IBM. What is NetFlow? [online]. 2025. Dostupné z: https://www.ibm.com/think/topics/netflow [citováno 2025-04-14]

[15] Karau, H. a Warren, R. *High Performance Spark: Best Practices for Scaling and Optimizing Apache Spark*. Sebastopol: O'Reilly Media, 2017. ISBN 9781491943205.

[16] Leblond, É. a Manev, P. *The Security Analyst's Guide to Suricata*. Indianapolis: Stamus Networks, 2023. ISBN 979-8-9871510-1-3. Dostupné z: https://www.stamus-networks.com/hubfs/Library/Suricata-4-Analysts%20Book/TheSecurityAnalystsGuidetoSuricata.pdf

[17] Luu, H. *Beginning Apache Spark 2: With Resilient Distributed Datasets, Spark SQL, Structured Streaming and Spark Machine Learning Library*. New York: Apress, 2018. ISBN 9781484235799. Dostupné z: https://www.oreilly.com/library/view/beginning-apache-spark/9781484235799/

[18] Miller, D. R. *Security Information and Event Management (SIEM) Implementation*. New York: McGraw-Hill Education, 2010. ISBN 0071701095. 464 s.

[19] Národní úřad pro kybernetickou a informační bezpečnost. *GovCERT.CZ Vládní CERT tým*. [online]. 2025 Dostupné z: https://nukib.gov.cz/cs/kyberneticka-bezpecnost/vladni-cert/ [citováno 2025-04-22]

[20] Národní úřad pro kybernetickou a informační bezpečnost. *Oficiální logo NÚKIB*. [online]. 2025. Dostupné z: https://nukib.gov.cz/ [citováno 2025-04-22]

[21] Oltsik, J. *4 key trends reshaping the SIEM market*. [online]. CSO Online, 2023. Dostupné z: https://www.csoonline.com/article/3829750/4-key-trends-reshaping-the-siem-market.html [citováno 2025-03-16]

[22] Open Information Security Foundation. *Introduction to Rules – Suricata Documentation*. [online]. 2025. Dostupné z: https://docs.suricata.io/en/latest/rules/intro.html [citováno 2025-04-14]

[23] Open Container Initiative. *What is the mission of OCI?* [online] In: *Open Container Initiative FAQ,* 2025. Dostupné z: https://opencontainers.org/faq/#what-is-the-mission-of-the-oci [citováno 2025-04-16]

[24] Palo Alto Networks. *Threat Hunting Methodologies*. [online]. 2024 Dostupné z: https://www.paloaltonetworks.com/cyberpedia/threat-hunting#methodologies [citováno 2025-03-26]

[25] Progress Software Corporation. *Flowmon sonda, produktová brožura*. [online]. 2022. Dostupné z: https://www.progress.com/docs/default-source/flowmon-resources/flowmon-probe-cz.pdf [citováno 2025-03-12]

[26] Shapira, G., Palino, T., Sivaram, R. a Petty, K. *Kafka: The Definitive Guide. 2nd ed.* Sebastopol: O'Reilly Media, 2021. ISBN 9781492043089. Dostupné z: https://www.oreilly.com/library/view/kafka-the-definitive/9781492043072/ [citováno 2025-04-06]

[27] Somerford Associates. *What is Splunk's Universal Forwarder?* [online] Somerford Blog, 2023. Dostupné z: https://www.somerfordassociates.com/blog/what-is-splunks-universal-forwarder/ [citováno 2025-03-25]

[28] Splunk Inc. *Managing Indexers and Clusters of Indexers* [online]. In: *Splunk Documentation, version 9.4.1.*, 2025. Dostupné z: https://docs.splunk.com/Documentation/Splunk/9.4.1/Indexer/Aboutindexesandindexers [citováno 2025-03-25]

[29] Splunk Inc. *Types of forwarders* [online]. In: *Splunk Documentation, version 9.4.1.*, 2025. Dostupné z: https://docs.splunk.com/Documentation/Splunk/9.4.1/Forwarding/Typesofforwarders [citováno 2025-03-25]

[30] Splunk Inc. *SIEM: Security Information and Event Management*. [online]. Splunk Learn Blog, 2025. Dostupné z: https://www.splunk.com/en_us/blog/learn/siem-security-information-event-management.html [citováno 2025-03-25]

[31] Splunk Inc. *Components of a Splunk Enterprise deployment*. [online]. Splunk Documentation, version 9.4.1., 2025 Dostupné z: https://docs.splunk.com/Documentation/Splunk/9.4.1/Capacity/ComponentsofaSplunkEnterprisedeployment [citováno 2025-03-25]

[32] TeamT5. *Benefits of Threat Hunting Strategies for Enterprises*. [online]. 2023 Dostupné z: https://teamt5.org/en/posts/5-benefits-of-threat-hunting-strategies-for-enterprises/ [citováno 2025-03-26]

[33] TechTarget. *SOAR (Security Orchestration, Automation and Response)*. [online]. SearchSecurity, 2025. Dostupné z: https://www.techtarget.com/searchsecurity/definition/SOAR [citováno 2025-03-12]

[34] Thomas, A. E. *Security Operations Center: SIEM Use Cases and Cyber Threat Intelligence*. Charleston: CreateSpace Independent Publishing Platform, 2018. ISBN 9781986862011. 376 s.

[35] Walsh, D. *Podman in Action*. Shelter Island: Manning Publications, 2023. ISBN 9781633439689. Dostupné z: https://www.manning.com/books/podman-in-action [citováno 2025-04-20]

[36] Apache Software Foundation. *Apache Kafka Documentation: KRaft Mode* [online]. 2025. Dostupné z: https://kafka.apache.org/documentation/#kraft [citováno 2025-04-12]

[37] Merkel, D. Docker: lightweight Linux containers for consistent development and deployment. Linux Journal, 2014 (239). Dostupné z: https://dl.acm.org/doi/10.5555/2600239.2600241 [citováno 2025-04-20]

[38] Red Hat, Inc. Podman documentation (latest version) [online]. 2025. Dostupné z: https://docs.podman.io/en/latest/ [citováno 2025-04-20]

SEZNAM POUŽITÝCH OBRÁZKŮ

SEZNAM POUŽITÝCH TABULEK

SEZNAM POUŽITÝCH ZKRATEK

AMP	Algorithms, Machines, People
API	Application Programming Interface
APT	Advanced Persistent Threat
ATT&CK	Adversarial Tactics, Techniques & Common Knowledge
AWS	Amazon Web Services
CERT	Computer Emergency Response Team
CMS	Centrální místa služeb
CNAME	Canonical Name record
DNS	Domain Name System
EDR	Endpoint Detection and Response
GDPR	General Data Protection Regulation
HEC	HTTP Event Collector
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IoC	Indicator of Compromise
IP	Internet Protocol
IPFIX	Internet Protocol Flow Information Export
IPS	Intrusion Prevention System
IPsec	Internet Protocol Security
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
JSON	JavaScript Object Notation
KRaft	Kafka Raft Metadata mode
KIVS	Komunikační infrastruktura veřejné správy
NIS2	Směrnice Evropské unie (Network and Information Security)

NÚKIB	Národní úřad pro kybernetickou a informační bezpečnost
OASP	Oddělení Analýzy Síťového provozu
OCI	Open Container Initiative
OISF	Open Information Security Foundation
PTR	Pointer (ukazatel u DNS záznamu pro reverzní překlad adresy)
SEM	Security Event Management
SIEM	Security Information and Event Management
SIM	Security Information Management
SNI	Server Name Indication (rozšíření protokolu TLS)
SOA	Start of Authority (část DNS záznamu)
SOAR	Security Orchestration, Automation and Response
SOC	Security Operations Center
SPAN	Switched Port Analyzer
SPL	Search Processing Language
SQL	Structured Query Language
SRV	Service record (DNS záznam)
TAP	Test Access Point
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TTL	Time to Live
TXT	Textový záznam
UDP	User Datagram Protocol
UEBA	User and Entity Behavior Analytics
XML	eXtensible Markup Language
YARN	Yet Another Resource Negotiator

SEZNAM PŘÍLOH