SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-16607-104631

POTVRDENIE INCIDENTU Z IDS LOGOV S VYUŽITÍM AI

DIPLOMOVÁ PRÁCA

2024 **Oliver Vraňák**

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-16607-104631

POTVRDENIE INCIDENTU Z IDS LOGOV S VYUŽITÍM AI

DIPLOMOVÁ PRÁCA

Študijný program:	Aplikovaná informatika
Názov študijného odboru:	Informatika
Školiace pracovisko:	Ústav informatiky a matematiky
Vedúci záverečnej práce:	Ing. Štefan Balogh, PhD.

Bratislava 2024 **Oliver Vraňák**

Slovenská technická univerzita v Bratislave
Ústav informatiky a matematiky

Fakulta elektrotechniky a informatiky
Akademický rok: 2023/2024
Evidenčné číslo: FEI-16607-104631

ZADANIE DIPLOMOVEJ PRÁCE

Študent:	**Bc. Oliver Vraňák**
ID študenta:	104631
Študijný program:	aplikovaná informatika
Študijný odbor:	informatika
Vedúci práce:	Ing. Štefan Balogh, PhD.
Vedúci pracoviska:	doc. Ing. Milan Vojvoda, PhD.
Miesto vypracovania:	Ústav informatiky a matematiky

Názov práce: **Potvrdenie incidentu z IDS logov s využitím AI**

Jazyk, v ktorom sa práca vypracuje: slovenský jazyk

Špecifikácia zadania:

Hrozby sofistikovaných a vysoko účinných kybernetických útokov sa stávajú rizikom v takej miere, že mnohé organizácie implementujú „Bezpečnostné operačné centrá“ (SOC) na detekciu a reakciu na incidenty.
S rastúcou sofistikovanosťou a objemom kybernetických útokov je pre bezpečnostných analytikov náročné identifikovať a správne reagovať na incidenty. Umelá inteligencia (AI) a veľké jazykové modely (LLM) sú čoraz sofistikovanejšie. Cieľom práce je analyzovanie možností využitia komerčných a open source LLM modelov v SOC prostredí.

Úlohy:
1. Analyzujte aktuálne LLM modely a vyberte vhodné na testovanie.
2. Analyzujte procesy v SOC centrách a navrhnite vhodný proces pre využitie LLM modelu a AI.
3. Vytvorte návrh riešenia s využitím LLM modelu.
4. Implementujte riešenie a vykonajte testovanie riešenia.
5. Vyhodnoťte úspešnosť pre komerčné a open source LLM modely.

Zoznam odbornej literatúry:

1. Motlagh, F. N., Hajizadeh, M., Majd, M., Najafi, P., Cheng, F., & Meinel, C. (2024). Large Language Models in Cybersecurity: State-of-the-Art. arXiv preprint arXiv:2402.00891.

SÚHRN

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný program:	Aplikovaná informatika
Autor:	Oliver Vraňák
Diplomová práca:	Potvrdenie incidentu z IDS logov s využitím AI
Vedúci záverečnej práce:	Ing. Štefan Balogh, PhD.
Miesto a rok predloženia práce:	Bratislava 2024

Cieľom diplomovej práce je preskúmať, navrhnúť a nakoniec implementovať vlastné riešenie na využitie umelej inteligencie pri identifikácii a klasifikácii bezpečnostných incidentov z IDS záznamov. Práca obsahuje stručný prehľad do problematiky monitorovania sieťovej prevádzky pomocou IDS riešení, ako aj popis veľkých jazykových modelov a ich využití. V práci si povieme taktiež aj o klasifikácii útokov a aké *frameworky* sa na to používajú. V druhej časti je možné nájsť návrh, postupy a aj samotnú implementáciu nášho nástroja. Tento nástroj pozostáva zo spracovania záznamov vygenerovaných monitorovacími nástrojmi, ich následnou koreláciou a nakoniec využitie LLM na ich analýzu. Nakoniec, v poslednej časti sú popísané výsledky testovania implementovaného návrhu pomocou viacerých jazykových modelov.

Kľúčové slová: detekcia, AI, IDS

ABSTRACT

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY

Study Programme:	Applied Informatics
Author:	Oliver Vraňák
Master's thesis:	Incident confirmation from IDS logs using AI
Supervisor:	Ing. Štefan Balogh, PhD.
Place and year of submission:	Bratislava 2024

The aim of this thesis is to investigate, propose and finally implement own solution for the use of artificial intelligence in the identification and classification of security incidents from IDS logs. The work contains a brief overview of the network traffic monitoring using IDS solutions as well as a description of large language models and their use. At this paper, we will also talk about the classification of attacks and what frameworks are used for it. In the second part, you can find the design, procedures and even the actual implementation of our tool. This tool consists of processing logs generated by monitoring tools, their subsequent correlation and finally using LLM for their analysis. Finally, the last part describes the results of testing the implemented design using several language models.

Keywords: detection, AI, IDS

Poďakovanie

Touto cestou by som chcel vyjadriť svoju vďaku vedúcemu mojej práce, Ing. Štefan Balogh, PhD., za odbornú pomoc, prístup a ochotu konzultovať akékoľvek problémy, na ktoré sme natrafili počas jej realizácie.

Obsah

Zoznam obrázkov a tabuliek

Zoznam skratiek

API	Application Programming Interface
CSS	Cascading Style Sheets
DDoS	Distributed Denial of Service
EVTX	Windows Event Logs
HTML	Hypertext Markup Language
ICMP	Internet Control Message Protocol
IDPS	Intrusion Detection and Prevention System
IDS	Intrusion Detection System
IP	Internet Protocol
IPS	Intrusion Prevention System
JSON	JavaScript Object Notation
LLM	Large Language Model
NLP	Natural Language Processing
NN	Neural Networks
RAM	Random-access memory
SIEM	Security Information and Event Management
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UI	Umelá Inteligencia
UTC	Universal Time Coordinated
XML	eXtensible Markup Language
XSS	Cross-site Scripting

Zoznam výpisov

Úvod

Neustále sa vyvíjajúce prostredie kybernetických hrozieb si vyžaduje neustály vývoj sofistikovaných bezpečnostných riešení. Analytici pracujúci v kybernetickej sfére čelia ťažkej úlohe monitorovania obrovského množstva údajov generovaných sieťovými a systémovými nástrojmi, identifikácie potenciálnych bezpečnostných incidentov, klasifikácie ich závažnosti a nakoniec aj ich zabráneniu. Tradičné systémy založené na pravidlách sa snažia držať krok s týmto dynamickým trendom, pričom ale často generujú falošné poplachy.

Veľké jazykové modely (LLM) sa ukázali ako sľubná technológia s potenciálom spôsobiť revolúciu v postupoch kybernetickej bezpečnosti. Tieto výkonné modely umelej inteligencie, trénované na obrovskom a rozsiahlom súbore dát, demonštrujú schopnosť porozumieť prirodzenému jazyku, generovať text a vykonávať zložité logické úlohy. Táto práca sa zaoberá aplikáciou LLM v oblasti kybernetickej bezpečnosti, konkrétne sa zameriava na potenciál identifikácie a následnej klasifikácie potenciálnych hrozieb z IDS záznamov.

Na začiatku sa ponoríme do problematiky umelej inteligencie a najmä veľkých jazykových modelov. Povieme si o ich obmedzeniach a možnostiach v kybernetickom priestore. Ďalej si povieme viac o možnostiach monitorovania sieťovej prevádzky, keďže tieto dáta nám budú slúžiť ako hlavný zdroj našich záznamov. V ďalších kapitolách si predstavíme návrh nášho nástroja na analýzu záznamov pomocou LLM a aj jeho implementáciu. Nakoniec podrobíme viacero jazykových modelov (komerčných, aj voľne dostupných) sade simulovaných útokov, aby sme si overili ich úspešnosť.

1 Intrusion Detection System (IDS)

Systém detekcie narušenia (IDS) je softvér, ktorý má na starosti monitorovanie sieťovej prevádzky a detegovanie známych podozrivých či škodlivých aktivít. O takýchto podozrivých aktivitách sú neskôr posielané upozornenia bezpečnostným tímom, ktoré podniknú ďalšie potrebné kroky. Hoci detekcia anomálií a ich následne hlásenie sú primárnymi úlohami IDS, niektoré systémy môžu ísť ešte ďalej a sú schopné prijať opatrenia v prípade zistenia škodlivej činnosti ako napríklad blokovanie požiadaviek z podozrivých IP adries [1].

1.1 Typy IDS

IDS riešenia vynikajú v monitorovaní sieťovej prevádzky a odhaľovaní podozrivých aktivít. Fungujú tak, že hľadajú podpisy známych typov útokov alebo zachytávajú aktivitu, ktorá sa odchyľuje od normálu. Následne upozornia alebo nahlásia tieto anomálie a potenciálne škodlivé akcie bezpečnostným tímom, aby ich bolo možné neskôr hlbšie analyzovať.

Existujú viaceré typy IDS, ktoré zisťujú podozrivé aktivity pomocou rôznych typov metód ako napríklad [2]:

- *Network intrusion detection system (NIDS)* - je strategicky nasadený v sieti, kde môže monitorovať prichádzajúcu a odchádzajúce sieťovú premávku.
- *Host intrusion detection system (HIDS)* - V tomto prípade IDS systém beží na všetkých zariadeniach v sieti s priamym prístupom do internej a externej siete. HIDS má výhodu oproti NIDS v tom, že môže byť schopný odhaliť anomálie v sieťových paketoch, ktoré pochádzajú zvnútra organizácie, alebo škodlivú komunikáciu, ktorú NIDS nedokázal odhaliť. HIDS môže byť tiež schopný identifikovať škodlivý prenos dát, ktorý pochádza od samotného klienta, napríklad keď bol infikovaný škodlivým softvérom.
- *Signature-based intrusion detection system (SIDS)* – SIDS monitoruje všetky pakety prechádzajúce sieťou a porovnáva ich s databázou signatúr útokov alebo atribútov známych škodlivých hrozieb, podobne ako antivírusový softvér.
- *Anomaly-based intrusion detection system (AIDS)* - Systém IDS založený na anomáliách je relatívne novšia technológia určená na odhaľovanie neznámych útokov. Tento typ detekcie využíva strojové učenie na analýzu veľkého množstva sieťových

údajov a prevádzky. Vytvára definovaný model normálnej aktivity a používa ho na identifikáciu anomálií. Je však na druhú stranu veľmi náchylný na falošné poplachy.

Historicky boli systémy detekcie narušenia kategorizované ako pasívne alebo aktívne. Pasívny IDS je ten, ktorý by zistil škodlivú aktivitu. Daný systém by neskôr vygeneroval upozornenia ale nevykonal by žiadnu akciu. Aktívny IDS, niekedy nazývaný aj ako systém detekcie a prevencie narušenia (IDPS), by generoval upozornenia ale navyše by mohol byť nakonfigurovaný aj na vykonávanie akcií, ako je blokovanie IP adries alebo zastavenie prístupu k určitým zdrojom zdrojom [2].

1.2 Výhody a nevýhody IDS

Riešenia IDS ponúkajú organizáciám veľké výhody, predovšetkým v oblasti identifikácie potenciálne škodlivých aktivít. Medzi najväčšie výhody patria napríklad [1]:

- Pochopenie rizika - IDS pomáha organizáciám bližšie priblížiť počet útokov, ich typ a riziko, ktorým na dennej báze čelia.
- Formovanie bezpečnostnej stratégie - Pochopenie rizika je kľúčové pre vytvorenie a vývoj komplexnej stratégie, ktorá dokáže obstáť v prípade hrozieb. IDS je tak možné použiť aj na identifikáciu chýb a potenciálnych nedostatkov v zariadeniach. Tie následne na posúdenie a prispôsobenie ich obrany tak, aby riešili riziká, ktorým môžu v budúcnosti čeliť.
- Rýchla odozva - Okamžité upozornenia, generované pomocou IDS pomáhajú organizáciám odhaliť a zabrániť útočníkom rýchlejšie, ako by to bolo pri ručnom monitorovaní siete.

Aj keď sú riešenia IDS dôležitými nástrojmi pri monitorovaní a zisťovaní potenciálnych hrozieb, nie sú bez problémové. Toto sú hlavné nevýhody [1]:

- *False positives* - Vďaka falošným poplachom sú riešenia IDS nie 100% spoľahlivé pri identifikácii potenciálnych hrozieb. Jedná sa o prípady, keď IDS deteguje anomáliu alebo škodlivú aktivitu ale reálne sa jedná o bežnú prevádzku.
- *False negatives* – Tento prípad predstavuje omnoho vyššie riziko, pretože je útočníkovi umožnené prejsť do siete organizácie, pričom IDS nezaznamená žiadnu anomáliu. To znamená, že systém považoval útočníkovo vniknutie do siete ako súčasť bežnej prevádzky.

Keďže sa prostredie hrozieb vyvíja a útočníci sa stávajú sofistikovanejšími, je vhodnejšie aby riešenia IDS poskytovali radšej falošne poplachy akoby nemali zaznamenať reálny útok.

1.3 Suricata

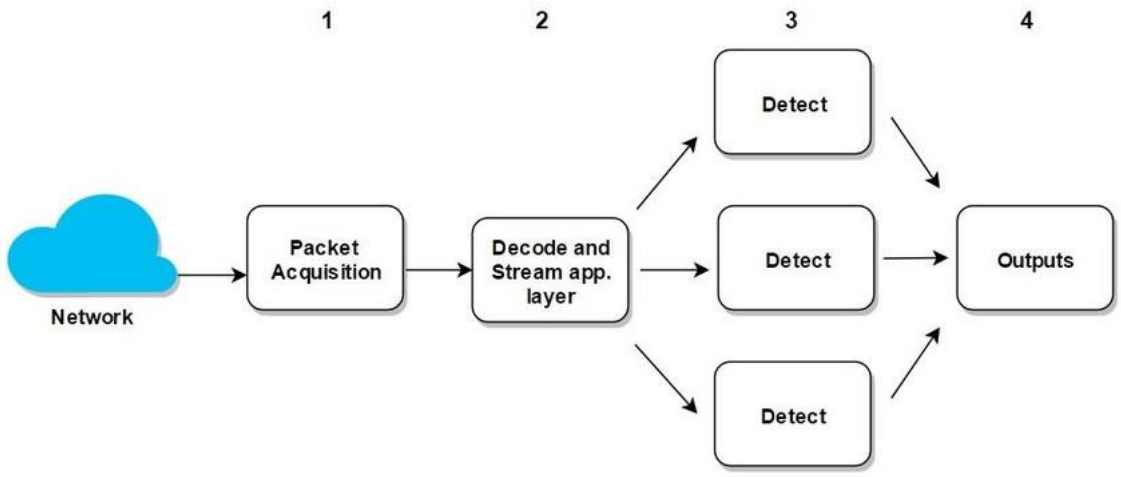
Suricata[1] je nekomerčný a všestranný softvér, ktorý funguje hlavne ako systém detekcie narušenia (IDS) ale je možné ho nakonfigurovať tak, aby v prípade útoku zakročil, čiže vie fungovať aj ako systém prevencie narušenia (IPS). Vytvorila ho nadácia Open Information Security Foundation (OISF)[2] a vyniká hlavne schopnosťou efektívne analyzovať sieťovú prevádzku, identifikovať hrozby v reálnom čase a dokonca im predchádzať v režime IPS.

1.3.1 Ako funguje Suricata

Suricata je nástroj nasadený v sieti, ktorý neustále monitoruje sieťovú prevádzku pre prípad podozrivých aktivít. Obsahuje nasledujúce základné funkcionality [3]:

- Zachytávanie a predspracovanie paketov - Suricata zachytáva sieťové pakety, čo sú v podstate malé jednotky dát prúdiace sieťou. Potom tieto údaje predspracuje na extrakciu dát a ďalej analyzuje.
- Priraďovanie pravidiel - Suricata obsahuje taktiež komplexný súbor pravidiel, ktoré definujú škodlivé aktivity. Neskôr v procese analýzy, Suricata porovnáva extrahované informácie zo sieťových paketov s týmito preddefinovanými pravidlami.
- Detekcia a analýza hrozieb - Ak sa nájde zhoda medzi sieťovým prenosom a pravidlom, Suricata vygeneruje hlásenie. Toto upozornenie môže obsahovať podrobnosti o type zisteného útoku, zdrojovej IP adrese a ďalších relevantných informáciách.
- Akcie (voliteľné) - Keď je Suricata nakonfigurovaná pre režim IPS, môže vykonávať preventívne opatrenia. Môže to zahŕňať blokovanie škodlivej prevádzky, resetovanie pripojení alebo upozorňovanie na iné bezpečnostné systémy.

[1] https://suricata.io/
[2] https://oisf.net/



Obr. 1: Spôsob fungovania Suricaty [4].

2 Sysmon Monitor

Sysmon[3], tiež známy aj ako *System Monitor*, je systémová služba na operačnom systéme Windows, ktorá hrá kľúčovú úlohu pri monitorovaní a zaznamenávaní aktivít vykonávaných v systéme. Vo veľkom sa používa na zvýšenie bezpečnosti v organizáciách a to najmä kvôli tomu, že poskytuje podrobné informácie o rôznych systémových operáciách vrátane vytvárania procesov, sieťových pripojení a zmien v systéme súborov [5]. Kľúčové vlastnosti:

- Vylepšené monitorovanie - Sysmon ponúka vyššiu a presnejšiu úroveň sledovania rôznych udalostí vykonavaných v systéme ako sú napríklad vytváranie procesov, sieťové pripojenia alebo úpravy vykonané na súboroch.
- Detekcia škodlivej aktivity - Pomocou monitorovania správania procesov a sieťovej prevádzky umožňuje Sysmon detekciu škodlivých aktivít, čo môže byť neskôr užitočné v procese vytvárania detekčných pravidiel.
- Podrobné zaznámenavanie - Sysmon všetky aktivity vykonávané v systéme zachytáva a podrobne zapisuje. Tieto záznamy poskytujú komplexné informácie o operáciách na úrovni systému vrátane ID procesu, ID nadradeného procesu, parametrov v termináli, *hashu* súborov, sieťových pripojení a pod.
- Flexibilita - Používateľ má veľké možnosti konfigurácie. Vie si, tak nastaviť čo má Sysmon monitorovať, ako detailné by mali byť záznamy a pod.

[3]https://learn.microsoft.com/en-us/sysinternals/downloads/sysmon

V prípade flexibility, Sysmon je možné nakonfigurovať pomocou XML konfiguračných súborov. Jeden z najpoužívanejších takých konfiguračných súborov je napríklad *sysmon-modular* [4]. Sysmon-modular je komunitou riadený projekt na platforme *GitHub*, ktorý ponúka modulárny prístup. Obsahuje veľké množstvo detekčných pravidiel založených na základe Mittre ATT&CK *frameworku* ako aj definované pravidlá na generovanie podrobných záznamov.

2.1 Fungovanie Sysmonu

Ako bolo spomenuté, Sysmon zaberá širokú škálu pri monitorovaní systému. Používa ovládač s názvom *SysmonDrv.sys* a službu bežiacu na pozadí na monitorovanie aktivít. Na zachytenie dodatočných informácií, *SysmonDrv.sys* registruje vykonané Windows API volania. Sysmon taktiež sleduje službu Event Tracing for Windows (ETW)[5], čo je natívna Windows služba, ktorá sleduje a zaznamenáva udalosti vyvolané aplikáciami v používateľskom režime a ovládačmi v režime jadra [5].

Sysmon je práve preto výkonný nástroj, ktorý výrazne zlepšuje monitorovanie a bezpečnosť systému tým, že poskytuje podrobné informácie o systémových aktivitách, čím výrazne pomáha bežpečnostným tímom efektívne odhaliť a reagovať na bezpečnostné hrozby.

[4]https://github.com/olafhartong/sysmon-modular

[5]https://learn.microsoft.com/en-us/windows-hardware/drivers/devtest/event-tracing-for-windows–etw-

3 Cyber Kill Chain

Ľudia pracujúci v oblasti kybernetickej bezpečnosti sa na dennom poriadku stretávajú s možnými hrozbami čí nástrahami. Aby sa týmto hrozbám mohli efektívne brániť, je dôležité porozumieť cieľu útočníka. V tejto kapitole si povieme hlavne o tzv. *Cyber kill chain-e*, ktorý popisuje životný cyklus kybernetického útoku a neskôr sa pozrieme aj na bežne používané *frameworky*, ktoré daný životný cyklus aj implementujú.

3.1 Pohľad na útok z vyššej perspektívy

Cyber Kill Chain, je v podstate model, ktorý zachytáva viaceré fázy, ktorými útočníci zvyčajne prechádzajú počas kybernetického útoku. Tento model bol vyvinutý spoločnosťou Lockheed Martin[6] v roku 2011 a ponúka zjednodušený pohľad na životný cyklus útoku, ktorý tak umožňuje ľuďom z bezpečnostných tímov ľahšie identifikovať potenciálne miesta zásahu.

Model sa dokopy skladá zo siedmich fáz a teraz sa pozrieme na každú z nich trochu bližšie [6]:

- *Reconnaissance* - V tejto fáze útoku útočníci zhromažďujú veľké množstvo informácií o svojom cieli, ktoré vedia neskôr použiť pri útokoch. Zväčša útočníci identifikujú zraniteľné miesta alebo získavajú cenné informácie ako sú emailové adresy, heslá a pod.

- *Weaponization* - V prípade identifikovania zraniteľných komponentov, útočník získava, vytvára alebo modifikuje škodlivý kód, tak aby zneužil objavené zraniteľnosti.

- *Delivery* - Škodlivý obsah je doručený do cieľového systému, často zamaskovaný ako legitímny softvér.

- *Exploitation* - Útočník využíva doručený škodlivý obsah nájdenej zraniteľnosti na získanie počiatočného prístupu do systému.

- *Installation* - V tomto kroku útočník v napadnutom systéme zvyčajne nainštaluje škodlivý softvér tzv. malvér.

- *Command and Control* - Vo fáze Commad and Control, útočník vytvorí komunikačný kanál na kontrolu infikovaného systému, prípadne na vykonávanie ďalších operácií.

[6] https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html

- *Actions on Objectives* - V tomto bode dochádza k záveru útoku a tým je dosiahnutie útočníkovho cieľa. Cieľ útoku môže byť rôzny, ale zväčša sa jedná o exfiltráciu dát, narušenie systému, eskaláciu privilégií a pod.

Najväčšia výhoda tohto modelu spočíva v jeho jednoduchosti. Poskytuje jednoduchý a zároveň jasný obraz o životnom cykle útoku. Toto ďalej napomáha sústrediť sa na narušenia konkrétnych fáz, a tak predísť možným útokom.

3.2 MITRE ATT&CK Framework

V predošlej kapitole sme si povedali o čom *Cyber Kill Chain* je, a že ponúka pohľad na kybernetické útoky z vyššej perspektívy. Na druhú stranu sa ale nezaoberá žiadnymi technikami, taktikami alebo spôsobmi, ktoré útočníci pri útokoch využívajú. Z tohto dôvodu vznikol *framework* s názvom **Mitre ATT&CK** [7] (Adversarial Tactics, Techniques and Common Knowledge). S týmto *frameworkom* prišla nezisková organizácia Mitre Corporation a slúži ako komplexný zdroj informácií o taktikách a technikách používaných útočníkmi pri reálnych kybernetických útokoch.

Mitre ATT&CK kategorizuje taktiky (ciele) a techniky útočníka (špecifické metódy používané na dosiahnutie týchto cieľov) na rôznych platformách a operačných systémoch. Momentálne Mitre zhromažďuje databázu obsahujúcu 14 základných taktík [7]:

- *Reconnaissance* - Zbieranie informácií o cieli.
- *Resource Development* - Vytváranie zdrojov, ktoré budú použité pri útoku.
- *Initial Access* - Získanie počiatočného prístupu k systému.
- *Execution* - Spustenie škodlivého kódu na cieľovom systéme.
- *Persistence* - Udržiavanie prístupu k cieľovému systému.
- *Privilege Escalation* - Získanie vyšších oprávnení v cieľovom systéme.
- *Defense Evasion* - Obchádzanie bezpečnostných mechanizmov.
- *Credential Access* - Získanie prístupu k citlivým údajom, ako sú napríklad heslá.
- *Discovery* - Získanie informácií o cieľovom systéme po získaní prístupu.
- *Lateral Movement* - Pohyb medzi systémami v cieľovej sieti.

[7] https://attack.mitre.org/

- *Data Exfiltration* - Odcudzenie citlivých údajov z cieľového systému.
- *Impact* - Spôsobenie škody na cieľovom systéme.
- *Command and Control* - Komunikácia s infikovaným systémom a riadenie jeho aktivít.
- *Collection* - Zhromažďovanie informácií.

Pre každú spomínanú taktiku sú definované rôzne techniky (konkrétne metódy na dosiahnutie cieľa). Momentálne existuje viac ako 400 techník a viac ako 200 procedúr (podrobné implementácie techník).

Mitre ATT&CK si svoju popularitu vydobyl najmä svojou komplexnosťou a špecifickosťou. Obsahuje rozsiahle informácie o technikách, ktoré pokrývajú širokú platformu operačných systémov a sú bežne používané pri kybernetických útokoch.

4 Generatívna umelá inteligencia

Generatívna umelá inteligencia je časť umelej inteligencie (UI), ktorá je schopná generovať nový obsah vo forme textu, zvuku, videí či obrázkov. Pracuje na princípoch strojového učenia, odvetvia UI, ktorá umožňuje strojom učiť sa z dát. Na rozdiel od tradičných modelov strojového učenia, ktoré sa učia vzory a robia predpovede alebo rozhodnutia založené na týchto vzoroch, generatívna UI robí toho viac – nielen sa učí z údajov, ale vytvára aj nové inštancie údajov, ktoré napodobňujú vlastnosti vstupných údajov.

Bežný postup pri generatívnej UI je [8]:

- Zber údajov - Nazhromažďuje sa veľká množina údajov s príkladmi typu obsahu, ktorý sa má generovať.
- Model tréningu - Generatívny model umelej inteligencie je konštruovaný pomocou neurónových sietí.
- Generovanie - Po natrénovaní modelu môže v závislosti od použitého modelu generovať nový obsah.
- Spresnenie - V závislosti od úlohy a jeho využitia sa vygenerovaný obsah môže podrobiť ďalšiemu spresneniu alebo následnému spracovaniu, aby sa zlepšila jeho kvalita.

Základným kameňom generatívnej UI je hlboké učenie, čo je typ strojového učenia , ktoré napodobňuje fungovanie ľudského mozgu pri spracovaní údajov a vytváraní vzorov pre rozhodovanie. Modely hĺbkového učenia využívajú neurónové siete, ktoré pozostávajú z mnohých vzájomne prepojených vrstiev a tie spracúvajú a prenášajú informácie a napodobňujú neuróny v ľudskom mozgu [8].

4.1 Využitie v kybernetickej bezpečnosti

Potenciál generatívnej umelej inteligencie ovplyvniť oblasť kybernetickej bezpečnosti je veľký. Rovnako ako sa dokáže učiť a replikovať vzory v texte, môže sa učiť aj zo vzorov nájdených v kybernetických útokoch alebo zraniteľnostiach. Model vycvičený na obrovskom množstve údajov z danej oblasti by mohol identifikovať vzory a trendy, čo by viedlo k schopnosti predvídať budúce hrozby. Namiesto reagovania na hrozby v okamihu ich výskytu by odborníci na kybernetickú bezpečnosť mohli využiť spomínané modely na predvídanie hrozieb ešte pred ich realizáciou, a tak maximalizovať svoju úspešnosť.

Generatívna UI môže pomôcť bezpečnostným tímom presnejšie, efektívnejšie a produktívnejšie brániť svoje organizácie a to nasledovne [9]:

- Detekcia hrozieb v reálnom čase - Detekcia hrozieb je dnes jedným z najčastejších prípadov využitia generatívnej UI. Jeho používaním na rýchlejšiu identifikáciu vzorov a anomálií, efektívnejšie filtrovanie upozornení na incidenty, môžu organizácie výrazne zrýchliť svoju schopnosť odhaliť nové vektory hrozieb.

- Zlepšenie informovanosti o hrozbách - Generatívna UI sa tiež dá použiť na získanie podrobnejších informácii o hrozbách. Momentálne je potrebné používať zložité dotazovacie jazyky, operácie a reverzné inžinierstvo na analýzu obrovského množstva údajov, aby bolo možné hrozby pochopiť. UI, tak môže automaticky skenovať kód a sieťovú prevádzku a poskytovať informácie, ktoré môžu pomôcť analytikom pochopiť správanie škodlivých skriptov alebo prípadného útoku.

- Automatizácia bezpečnostných záplat - Generatívna UI dokáže automatizovať analýzu a aplikáciu záplat. Je tak možné natrénovať sieť, ktorá bude skenovať zdrojové kódy na zraniteľné miesta a aplikovať, poprípade navrhovať vhodné záplaty.

- Zlepšenie reakcie na incidenty - Ďalším možným využitím generatívnej UI v kybernetickej bezpečnosti je reakcia na incidenty. UI môže poskytnúť bezpečnostným analytikom stratégie založené na úspešných taktikách použitých pri minulých incidentoch, čo môže pomôcť urýchliť reakciu na incidenty.

4.2 Limitácie

Generatívna UI v kybernetickej bezpečnosti prináša veľa výhod a riešenia na mnohé výzvy, ktorým dnes čelia odborníci v danej oblasti. Aj keď je jej využitie všestranné, je dôležité pozrieť sa aj na výzvy, ktoré s tým prichádzajú. Ako ku každej technológii, aj k jej používaniu treba pristupovať zodpovedne, aby sa zmiernili riziká a potenciálne zneužitie [10].

- Vysoké výpočtové zdroje - Tréning takýchto modelov si vyžaduje značný výpočtový výkon a úložisko. Pre menšie organizácie to môže byť limitujúci faktor.

- Riziko použitia UI útočníkmi - Modely generatívnej UI a súvisiace nástroje, sú čoraz dostupnejšie širokej verejnosti. To dáva útočníkom možnosť použiť modely na vývoj sofistikovaných útokov, ktoré sú schopné obchádzať bezpečnostne opatrenia.

- Etické hľadiská - Ďalším problémom sú otázky súvisiace so súkromím a kontrolou dát, najmä pokiaľ ide o typ dát, ktoré používajú modely pri trénovaní.

4.3 Veľké jazykové modely

Veľký jazykový model, tzv. Large Language Model (LLM), je algoritmus hlbokého učenia, ktorý dokáže vykonávať rôzne úlohy spracovania prirodzeného jazyka (NLP). Veľké jazykové modely používajú transformačné modely a sú trénované pomocou veľkých súborov plných dát. Toto im umožňuje rozpoznávať, spracovávať, prekladať, predpovedať alebo aj generovať text či iný obsah [11].

Veľké jazykové modely sa označujú aj ako neurónové siete (NN), čo sú počítačové systémy inšpirované ľudským mozgom. Tieto neurónové siete pracujú pomocou siete uzlov, ktoré sú navrstvené podobne ako neuróny. Rovnako ako ľudský mozog, aj LLM sa musia vopred natrénovať a potom ladiť, aby mohli riešiť špecifickejšie problémy ako napríklad klasifikácie textu, odpovedania na otázky, sumarizácie dokumentov alebo generovania textu [11].

4.3.1 Architektúra transformátora

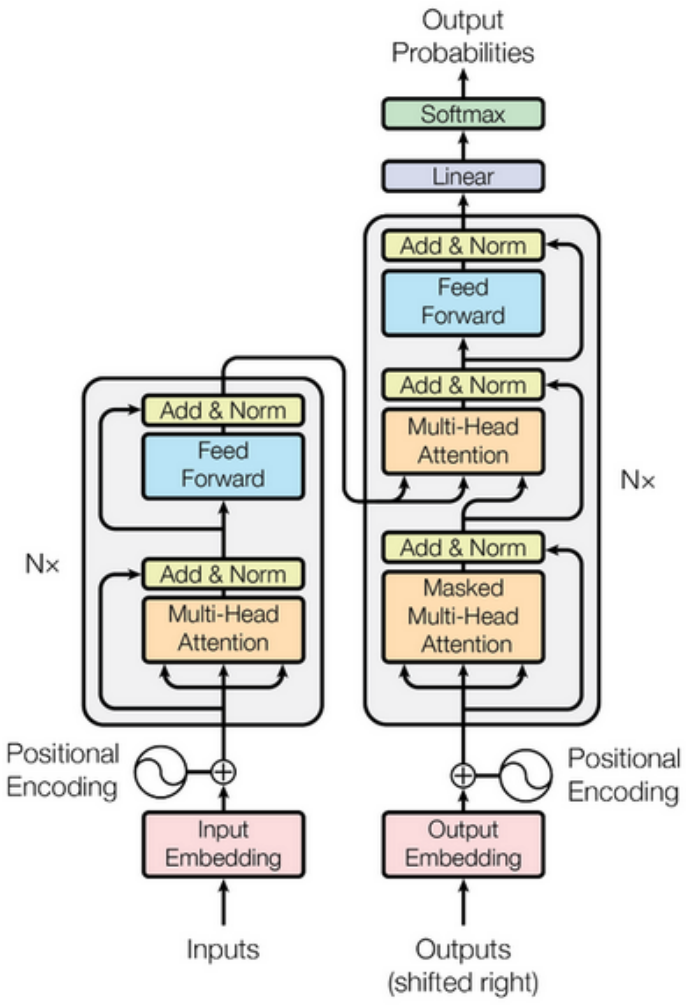
LLM, ako napríklad modely GPT, sú založené na architektúre transformátora. Jedná sa o mechanizmus sebapozorovania, ktorá umožňuje týmto modelom spracovávať vstupné dáta. Na rozdiel od predchádzajúcich modelov hlbokého učenia na spracovanie prirodzeného jazyka, ako sú RNN [8] a LSTMS [9], ktoré spracúvajú údaje sekvenčne, transformátory dokážu spracovávať vstupné dáta paralelne. Tým sa nielen zvyšuje efektivita tohto modelu, ale aj jeho schopnosť porozumieť kontextu. Kľúčové komponenty transformátora [12] [13]:

- Vkladacia vrstva - Táto vrstva konvertuje vstupné tokeny (slová) na vektory, ktoré sú číselnou reprezentáciou slov spolu so sémantickým významom a kontextom textu.
- Polohové kódovanie - Tento krok pridáva do vkladania informácie o pozícii každého tokenu v sekvencii. Účelom tohto postupu je kompenzovať nedostatok sekvenčného spracovania v transformátore.
- Enkóder a dekodér - Enkóder spracováva vstupný text a získava kontextové informácie, zatiaľ čo dekodér generuje súvislé odpovede predpovedaním ďalších slov v sekvencii.
- Mechanizmus vlastnej pozornosti - Mechanizmus vlastnej pozornosti umožňuje modelu zvážiť dôležitosť rôznych slov vo vstupnej sekvencii. To mu umožňuje pochopiť kontext a vzťahy medzi slovami v celej vstupnej sekvencii, čo je nevyhnutné pre prirodzený jazyk, kde sa význam slova môže meniť na základe jeho kontextu vo vete.

[8] https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/

[9] https://medium.com/analytics-vidhya/lstms-explained-a-complete-technically-accurate-conceptual-guide-with-keras-2a650327e8f2

- Napájacie neurónové siete - Enkóder a aj dekodér obsahujú neurónové siete na základe spätnej väzby, ktoré sú zodpovedné za aplikovanie ďalších transformácií na spracované údaje. To im umožňuje získať ešte ďalší prehľad o kontexte.
- Normalizácia vrstiev a zvyškové spojenia - Napokon ide o techniky používané v rámci blokov modelu na zvýšenie stability trénovania, zabránenie problému miznúceho gradientu a uľahčenie trénovania hlbších neurónových sietí.



Obr. 2: Architektúra transformátora [13].

Rôzne mechanizmy implementácie preto umožňujú jazykovému modelu zamerať sa na jednotlivé časti vstupného textu. Existujú preto tri hlavné typy LLM [12]:

- Všeobecné jazykové modely predpovedajú ďalšie slovo na základe jazyka v tréningových dátach. Tieto jazykové modely vykonávajú úlohy ako napríklad získavanie čí hľadánie potrebných informácií.
- Jazykové modely ladené podľa pokynov sú trénované na predpovedanie reakcií na pokyny uvedené vo vstupe. To im umožňuje vykonávať analýzu alebo generovať text či kód.
- Nakoniec, dialógovo ladené jazykové modely sú trénované na vedenie dialógu predpovedaním ďalšej odpovede.

4.3.2 Rozdiel medzi LLM a generatívnou UI

Ako už bolo spomínané, generatívna UI je širší pojem alebo obsiahlejšia kategória UI, ktorá sa vzťahuje na modely umelej inteligencie, ktoré majú schopnosť generovať obsah. Generatívna UI dokáže generovať text, kód, obrázky, video, hudbu či iný obsah. Medzi jej príklady patria napríklad DALL-E[10] [11].

LLM sú len podtypom generatívnej UI, ktoré sú trénované na základe textu a aj vytvárajú textový obsah. Jedným z príkladov je napríklad populárny chatbot ChatGPT[11] [11].

[10] https://openai.com/dall-e-2
[11] https://openai.com/chatgpt

5 Riešenia použité v práci

V práci budeme používať viaceré *frameworky* či programovacie jazyky, ktoré si teraz teoreticky viac priblížime.

5.1 Python

Jedná sa o vysoko úrovňový programovací jazyk, čo znamená, že je navrhnutý tak, aby sa ľuďom dal jednoducho čítať a písať. Za pomoci abstrakcie, sa tak vzďaľuje strojovému jazyku a približuje tomu ľudskému, čo zjednodušuje jeho použitie. Python bol navrhnutý programátorom Guidom van Rossom v roku 1989 ako nový programovací jazyk, ktorý mal nahrádzať chyby a nedostatky vtedajších programovacích jazykov [14].

5.2 Flask

Je ľahký, voľne dostupný webový vývojový *framework* napísaný v programovacom jazyku Python. Bol navrhnutý ako minimalistický, flexibilný a užívateľsky prívetivý nástroj na vytváranie webových aplikácií. Flask poskytuje jednoduchý spôsob vytvárania a nasadzovania dynamických webových aplikácií, čo umožňuje vývojárom sústrediť sa na aplikačnú logiku[15].

5.3 Html a CSS

HTML je relatívne jednoduchý jazyk, ktorý sa používa na to, aby informoval prehliadač o tom, ako má zobraziť webové stránky. Pozostáva zo série prvkov, ktoré sa používajú na uzavretie alebo zalomenie rôznych častí obsahu, aby vyzeral určitým spôsobom alebo pôsobil určitým spôsobom. Ohraničujúce značky môžu zo slova alebo obrázka urobiť hypertextový odkaz, môžu slová písať kurzívou alebo môžu zväčšiť alebo zmenšiť písmo. Pomocou HTML, je tak veľmi jednoduché vytvoriť si základnú štruktúru našej webovej stránky [16].

CSS sa používa ako doplnok pre HTML. Samotné HTML dokáže vytvoriť základnú štruktúru webovej stránky, avšak nedokáže zabezpečiť ako má stránka vyzerať graficky. Pomocou CSS, je tak možné zabezpečiť vizuálny vzhľad webového rozhrania.

6 Návrh riešenia

V tejto časti sa budeme venovať samotnému návrhu nášho riešenia. Cieľom práce je oboznámiť sa a odskúšať si detekčné systémy IDS ako je napríklad Suricata, a taktiež podporný monitorovací systém Sysmon. Následne si osvojiť Mitre ATT&CK *framework* a jeho techniky, procedúry, taktiky a celkové využitie. V neposlednom rade, oboznámiť sa s metódami generatívnej UI, jej výhody v kybernetickom priestore a taktiež limitácie. Nakoniec, sa na základe získaných poznatkov pokúsime navrhnúť a implementovať vlastné riešenie na identifikáciu a klasifikáciu potenciálnych bezpečnostných incidentov z IDS a Sysmon záznamov s využitím generatívnej UI.

Podstatou samotného riešenia je dokázať zakategorizovať potenciálne bezpečnostné incidenty z obdržaných záznamov. Naše riešenie v prvom rade akceptuje záznamy vygenerované nástrojmi Suricata a Sysmon, ktoré monitorujú sieťovú prevádzku, ale aj celkové dianie na úrovni systému. Tieto záznamy sú neskôr spracované aby bolo možné s nimi ďalej pracovať.

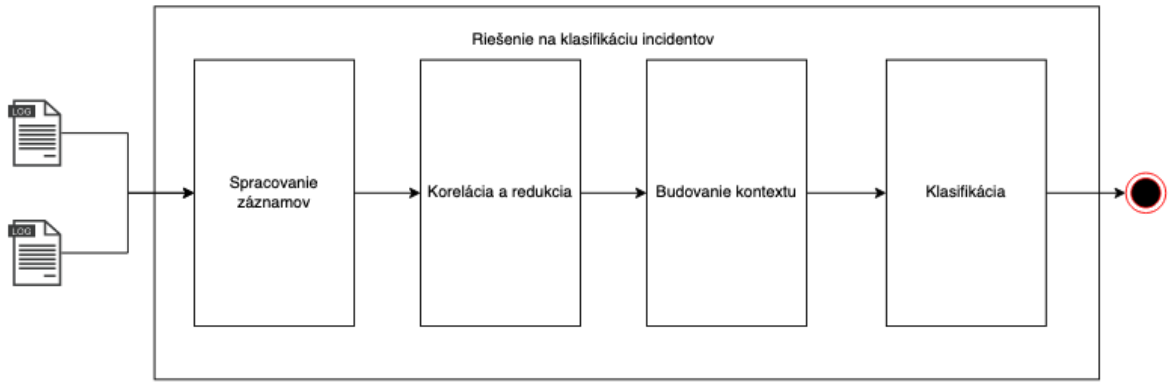
Pripravené dáta sa ďalej korelujú na základe času po určitých časových intervaloch. Takto sa vlastne vytvoria skupiny záznamov, ktoré reprezentujú čo všetko sa udialo za daný určitý okamih. Keďže záznamov sa pri určitých typoch útokov môže vygenerovať ohromné množstvo, ďalším krokom je redukcia záznamov v spomínaných skupinách.

Po redukcii pokračuje fáza využitia generatívnej UI. Presnejšie, v tejto fáze sa snažíme vytvoriť celkový kontext diania tým, že poskytneme odchytené, spracované a redukované záznamy LLM na analýzu. Výstup tejto analýzy spolu so spomínanými záznamami tvoria celkový kontext a ten ide znovu do LLM modelu, ktorý sa v poslednom kroku snaží zakategorizovať daný kontext podľa Mitre ATT&CK *frameworku*.

6.1 Architektúra riešenia

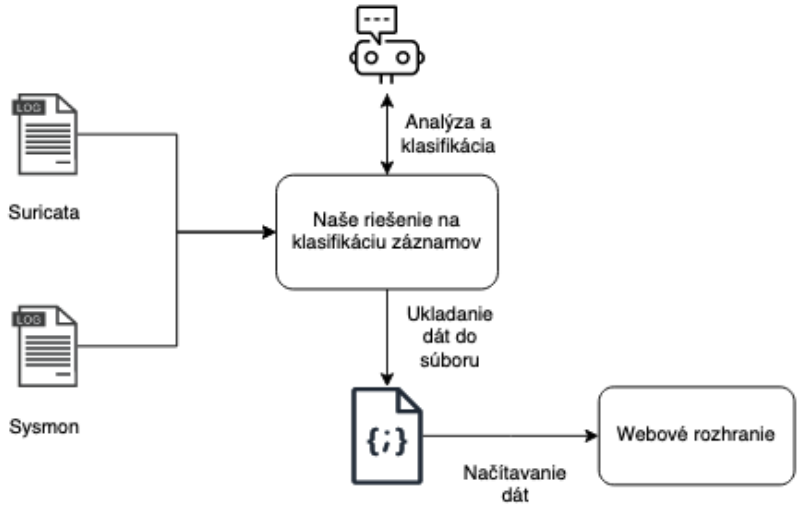
Čo sa týka architektúry nášho systému, ktorý sme vyššie popísali, bude sa skladať zo štyroch hlavných častí. V týchto častiach sa bude vykonávať celková manipulácia so záznamami ako aj finálna klasifikácia. Mimo daných častí naše riešenie bude obsahovať aj jednoduché webové rozhranie na vizualizáciu výsledkov.

V krátkosti, v prvej časti budeme zbierať a spracovávať záznamy vygenerované pomocou IDS riešenia Suricata a monitorovacieho systému Sysmon. V druhej časti, sa budeme snažiť spracované záznamy korelovať podľa času a redukovať. Následne začneme budovať kontext pre LLM pomocou analýzy daných záznamov. Nakoniec, sa pokúsime identifikovať a klasifikovať záznamy pre prípadne bezpečnostné incidenty.



Obr. 3: Návrh architektúry riešenia.

Všetky dáta, ako napríklad obdržané a spracované záznamy a ich analýzy a klasifikácie, sú ukladané do súboru. Vygenerovaný súbor, tak obsahuje veľké množstvo informácií o uskutočnených udalostiach vykonaných v sieti alebo v systéme. V našom prípade sme navrhli jednoduché webové rozhranie, ktoré bude načítavať postupne dáta zo spomínaného súboru a bude ich vizualizovať pre jednoduchší prehľad.



Obr. 4: Celkový návrh riešenia.

6.2 Spracovávanie záznamov

Na začiatku práce sme si podrobnejšie povedali o IDS/IPS systéme Suricata. Tá, sa široko používa na monitorovanie bezpečnosti počítačovej siete. Je navrhnutá tak, aby analyzovala sieťovú prevádzku v reálnom čase a zisťoval potenciálne bezpečnostné hrozby na základe vopred definovaných pravidiel. V jednoduchosti, Suricata funguje na základe analýzy sieťovej prevádzky na úrovni paketov pomocou rôznych protokolov, ako sú TCP, UDP a ICMP. Dokáže preto odhaliť sieťové anomálie a podozrivé vzory porovnaním sieťovej prevádzky s preddefinovanými pravidlami, signatúrami a protokolmi.

Na druhú stranu, výstrahy sa generujú na základe vopred definovaných pravidiel a signatúr. Tieto pravidlá sú definované v jazyku Suricata Rules Language (SRL) a používajú sa na detekciu rôznych typov sieťových hrozieb. V prípade, že Suricata deteguje potenciálnu anomáliu, vygeneruje tok, čo je kolekcia sieťových paketov, ktoré zodpovedajú rovnakým kritériám. Každý tok obsahuje metadáta o paketoch, ako sú zdrojové a cieľové IP adresy, porty a protokoly. Suricata generuje upozornenia na základe tokov, ktoré vytvára. Keď sa tok zhoduje so zadefinovaným pravidlom, Suricata vygeneruje upozornenie. Všetky záznamy, čo Suricata generuje, toky a aj výstrahy sa ukladajú do súboru vo formáte JSON.

Nás v našom riešení budú najviac zaujímať práve spomínané vygenerované výstrahy, keďže obsahujú informácie o toku, čiže IP adresy, čísla portov, pravidle, ktoré sa spustilo, a závažnosť upozornenia. Nižšie je možné vidieť ako vyzerá taká výstraha reprezentujúca podozrivú komunikáciu na porte číslo 1433, čo je číslo portu charakteristické pre MSSQL[12] databázu.

[12]https://en.wikipedia.org/wiki/Microsoft_SQL_Server

```
{"timestamp":"2024-02-21T20:30:29.438193+0200","flow_id":1600553847883771,
"in_iface":"\\Device\\NPF_{A0B14077-6B07-43C7-822D-17ACA68E4CFD}",
"event_type":"alert","src_ip":"192.168.100.5","src_port":60579,
"dest_ip":"192.168.100.10","dest_port":1433,"proto":"TCP",
"pkt_src":"wire/pcap","alert":{"action":"allowed","gid":1,
"signature_id":2010935,"rev":3,
"signature":"ET SCAN Suspicious inbound to MSSQL port 1433",
"category":"Potentially Bad Traffic","severity":2,
"metadata":{"created_at":["2010_07_30"],"former_category":["HUNTING"],
"updated_at":["2018_03_27"]}},"direction":"to_server",
"flow":{"pkts_toserver":1,"pkts_toclient":0,"bytes_toserver":60,
"bytes_toclient":0,"start":"2024-02-21T20:30:29.438193+0200",
"src_ip":"192.168.100.5",
"dest_ip":"192.168.100.10","src_port":60579,"dest_port":1433}}
```


Obr. 5: Ukážka generovanej výstrahy zo sieťových tokov.

V našom návrhu figuruje aj monitorovací systém Sysmon, ktorý možno použiť na monitorovanie a detekciu škodlivých aktivít na systéme Windows. Je navrhnutý tak, aby dopĺňal zvyšné monitorovacie nástroje ako napríklad systémy IDS/IPS, a to poskytovaním podrobných informácií o systémových udalostiach, ako je vytváranie procesov či sieťové pripojenia. Keďže sa v našom riešení snažíme odhaliť potenciálny incident a zaklasifikovať ho podľa Mitre ATT&CK *frameworku* za pomoci LLM, je dôležite si pre dané modely pripraviť dostatočný kontext. To znamená, získať čo najviac informácii čo sa v danom momente na systéme udialo aby bolo možné čo najpresnejšie obdržané záznamy zanalyzovať. Napríklad, predstavme si, že Suricata zistí podozrivé sieťové pripojenie, záznamy zo Sysmonu je možno použiť na poskytnutie informácií o procese, ktorý dané spojenie vytvoril, používateľovi, ktorý spustil proces, a akýchkoľvek iných systémových udalostiach, ktoré sa vyskytli v rovnakom čase.

Sysmon záznamy sú generované vo formáte EVTX a obsahujú podrobné informácie o systémových udalostiach. Keďže generovaný formát záznamov je špecifický a nie všetky nástroje sú schopné daný formát akceptovať, je preto potrebné tieto záznamy konvertovať. Pre jednoduchosť, keďže Suricata svoje výstrahy ukladá vo formáte JSON, v našom riešení budeme záznamy generované nástrojom Sysmon taktiež konvertovať do rovnakého formátu, tak aby sme docielili lepšiu integráciu a schopnosť ľahšej manipulácie s dátami. Kedže

EVTX formát je špecialna varianta XML formátu, pri konvertovaní je potrebné načítať si daný súbor a prechádzať všetkými elementami, ktoré sa v súbore nachádzajú. Na základe elementov sa vyextrahujú a priradia metadáta k svojím uzlom, respektíve elementom. Primárne nás budú zaujímať xml elementy *EventData* a *TimeCreated*. Tieto elementy, obsahujú všetky potrebné informácie o každom zázname, ktorý bol vygenerovaný, ako je čas udalosti, o akú udalosť sa jednalo a podobne. Sprocesované elementy s dátami, uložíme do *python* slovníka a nakoniec aj do súbora. Vygenerovaný súbor, bude takto obsahovať Sysmon záznamy prekonvertované do JSON formátu. Výhodou tohto formátu je jeho jednoduchosť a kompatibilita s programovacími jazykmi. Na obrázku nižšie, je možné pozorovať ako vyzerá taký záznam vygenerovaný systémom Sysmon.

```
{"Provider_Name": "Microsoft-Windows-Sysmon",
"Provider_Guid": "{5770385f-c22a-43e0-bf4c-06f5698ffbd9}", "Provider": null,
"EventID_Qualifiers": "", "EventID": "3", "Version": "5", "Level": "4",
"Task": "3", "Opcode": "0", "Keywords": "0x8000000000000000",
"TimeCreated_SystemTime": "2024-02-21 18:09:25.788668",
"TimeCreated": null, "EventRecordID": "134349",
"Correlation_ActivityID": "",
"Correlation_RelatedActivityID": "", "Correlation": null,
"Execution_ProcessID": "3176", "Execution_ThreadID": "4360",
"Execution": null,
"Channel": "Microsoft-Windows-Sysmon/Operational",
"Computer": "DESKTOP-CN8KBJS",
"Security_UserID": "S-1-5-18", "Security": null,
"EventData_RuleName": "technique_id=T1571,technique_name=Non-Standard Port",
"EventData_UtcTime": "2024-02-21 18:09:24.077",
"EventData_ProcessGuid": "{f3770eaa-3c50-65d6-3800-000000000f00}",
"EventData_ProcessId": "2656",
"EventData_Image": "C:\\Windows\\System32\\svchost.exe",
"EventData_User": "NT AUTHORITY\\NETWORK SERVICE",
"EventData_Protocol": "udp",
"EventData_Initiated": "True", "EventData_SourceIsIpv6": "False",
"EventData_SourceIp": "192.168.100.10",
"EventData_SourceHostname": "-",
"EventData_SourcePort": "5353", "EventData_SourcePortName": "-",
"EventData_DestinationIsIpv6": "False",
"EventData_DestinationIp": "224.0.0.251",
"EventData_DestinationHostname": "-", "EventData_DestinationPort": "5353",
"EventData_DestinationPortName": "-"}
```


Obr. 6: Ukážka generovaného hlásenia systémom Sysmon.

Po získaní všetkých potrebných záznamov v potrebnom formáte, sa dostávame do ďalšieho bodu a tou je filtrácia. Ako je uvedené v krátkom popise o fungovaní Suricaty, tá generuje dva typy výstupu, toky a výstrahy. Všetky toky a výstrahy sa ukladajú do jedného

súboru. Výstrahy sú vygenerované na základe tokov, ktoré aktivovali definované pravidlá. Obsahujú dôležité informácie ako napríklad názov aktivované pravidla, súhrn potenciálne škodlivých sieťových tokov a ďalšie metadáta. V riešení si načítame daný súbor a budeme prechádzať každý jeden záznam. V prípade, že dostaneme záznam, ktorý bude označený ako výstraha, *alert*, tak si ho uložíme pre ďalšiu manipuláciu. To isté platí aj v prípade záznamov generovaných systémom Sysmon. Aj Sysmon generuje svoje záznamy rovnakým spôsobom, až na rozdiel, že nepoužíva toky alebo výstrahy. V tomto prípade budeme prechádzať záznamy a filtrovať len tie, ktoré boli aktivované definovaným pravidlom. Tie sa odlišujú najmä tým, že v popise záznamu obsahujú názov takého pravidla.

Nakoniec, takto načítané, spracované a filtrované záznamy nám poskytnú dostatočný základ pre samotnú identifikáciu a klasifikáciu.

6.3 Korelácia a Redukcia

Posunieme sa do ďalšieho kroku, v ktorom sa budeme snažiť korelovať získané záznamy na základe času a pripraviť ich na nasledujúcu časť, kde budeme dané dáta analyzovať. Prvým krokom pri korelácií záznamov vygenerovaných pomocou nástrojov Suricata a Sysmon, bude ich zarovnanie podľa času. Po ich zoradení je potrebné si špecifikovať časový rámec. Jedným z návrhov je vytvoriť takzvané skupiny záznamov, kde každá skupina bude balíkom všetkých záznamov vygenerovaných pomocou Suricaty a Sysmonu, ktoré sa za časové okno udiali. Napríklad, ak si zvolíme, že chceme spojiť záznamy na základe 1 minútového časového rámca, tak jeden balík bude obsahovať všetky záznamy, ktoré sa udiali napríklad medzi 20:00 a 20:01. Takýmto spôsobom získame podrobnejší prehľad čo sa za daný čas udialo.

Dôležitou otázkou je ale správne nastavenie spomínaného časového rámca, keďže nemusí odrážať aktuálne dianie. Napríklad, korelácia záznamov za dlhšie obdobie môže viesť k vysokému počtu falošných poplachov, keď sú bežné udalosti nesprávne označené ako incidenty. Je to preto, že veľké časové okno môže zachytiť omnoho väčšie množstvo údajov, čo sťažuje rozlíšenie medzi normálnymi a abnormálnymi aktivitami. To znamená, že sa môže stať, že v prípade veľkej sieťovej prevádzky sa škodlivá sieťová komunikácia rýchlo stratí a nemusí byť odhalená. Naopak, malé časové okná nemusia ihneď zachytiť celý kontext bezpečnostného incidentu, pretože poskytujú iba úzky pohľad na vec. To môže viesť k neúplnej alebo aj nepresnej analýze.

Následne, po korelácií záznamov bude potrebné dané záznamy redukovať a to z viacerých dôvodov. Jedným z nich je vyhnutie sa duplikátom a tým najväčším faktorom je limitácia veľkosti vstupu pre LLM. V prípade, že nastavíme časový rámec na dlhšie

obdobie alebo v prípade určitých sieťových útokov, kde sa generujú stovky paketov za sekundu, ako to je v prípade DDoS útoku, veľké jazykové modely nebudú schopné, také obrovské množstvo dát spracovať. Práve preto bude redukcia záznamov nevyhnutná pri ich efektívnom spracovaní. Dôležite ale bude záznamy redukovať do spracovateľnejšieho tvaru a to s čo najmenšou stratou dôležitých detailov. To znamená, že v procese redukcie sa odfiltrujú nadbytočné alebo menej významné atribúty zo záznamov, aby sme sa mohli zamerať na tie najrelevantnejšie a najdôležitejšie informácie v nich. Mimo odstránenia menej podstatných atribútov v záznamoch sa zameriame aj na unikátne záznamy. Jednoduchšie si to vieme predstaviť tak, že pre naše záznamy spolu s počtom ich výskytov budeme používať takzvaný register. Zoberieme prvý záznam, a v prípade, že sa ešte nenachádza v našom registri, tak ho tam pridáme. Takto si vieme jednoducho overiť identifikované záznamy. V prípade, že budeme spracovávať záznam, ktorý sa už nachádza v našom registri, tak ho nepridáme, ale iba inkrementujeme jeho hodnotu výskytu. Dané číslo, tak bude reprezentovať frekvenciu výskytu daného záznamu za určitý čas. Takto môžeme spracovávať aj väčšie množstvo záznamov bez väčšej straty podstatných informácií.

Výstupom tejto fázy spracovania záznamov bude štruktúrovaný JSON zoznam, ktorý bude obsahovať korelované záznamy na základe časového rámca do skupín, a dané skupiny budú obsahovať všetky redukované unikátne záznamy aj s ich počtom výskytov, ktoré sa udiali v danom okamihu.

6.4 Budovanie kontextu

V predošlej podkapitole sme rozoberali korelovanie záznamov na základe času a taktiež ako je možné dané záznamy redukovať. V tejto fáze sa pozrieme na to ako tieto zhromaždené záznamy analyzovať pomocou LLM. Tieto modely sú obzvlášť účinné pri analýze dát vďaka svojej schopnosti porozumieť a interpretovať text v prirodzenom jazyku. To znamená, že sú schopné porozumieť kontextu správy, ktorú mu používateľ poskytne. Na základe daného kontextu vie model potom podať čo najpresnejšiu odpoveď. Z toho vyplýva, že kontext je najdôležitejšou zložkou pri týchto jazykových modeloch, keďže im umožňuje analyzovať a pochopiť sémantický význam slov a syntaktickú štruktúru viet. Širšie kontextové okno umožňuje modelom napríklad udržiavať súdržné príbehy a porozumieť zložitým vetám. Avšak, veľké kontextové okno výrazne zvyšuje výpočtovú zložitosť a teda aj rýchlosť pri interakcií [17]. V našom prípade je kľúčové vybudovať čo najpresnejší kontext, aby sme boli schopný s pomocou LLM klasifikovať potenciálne incidenty.

V tomto štádiu využijeme teda potenciál LLM na základnú analýzu získaných zá-

znamov. Zaujímať nás bude najmä to či záznamy zo Suricaty a Sysmonu spolu súvisia, a to z toho dôvodu, že sa môže stať, že v záznamoch sa nájdu falošné poplachy, ktoré vôbec nemusia súvisieť so zvyškom záznamov. Napríklad, Suricata nám vygeneruje a poskytne záznam o externej sieťovej komunikácii s portom 1433, čo je charakteristický port pre MSSQL databázu, avšak Sysmon nám môže poskytnúť záznam o tom, že interný používateľ si v danom čase práve spustil MSSQL klienta na vyťahovanie dát z databázy. Takéto falošné poplachy sú v bežnom živote normálne a je dôležité vedieť či je model schopný takéto nezrovnalosti identifikovať. Ďalej nás bude zaujímať, či je model schopný na základe daných záznamov zhruba odhadnúť či sa jedná o kybernetický útok, spomíname zhruba keďže zatiaľ nám ide hlavne o celkovú analýzu čo sa udialo a špecificky na incidenty sa budeme pýtať v ďalšom kroku. Keďže sme v predošlej kapitole popisovali redukciu záznamov, budeme taktiež dbať na to aby model bral do úvahy nielen typ záznamu, ale aj jeho celkový výskyt za dané časové obdobie. Nakoniec, budeme chcieť nech model vyhľadá všetky potenciálne indikátory kompromitácie v tých záznamoch a dohľadá k nim ďalšie informácie. Indikátory kompromitácie sú veľmi dôležité, pretože slúžia ako dôkaz škodlivej činnosti. Bežne zahŕňajú nezvyčajné vzory sieťovej prevádzky, neznáme aplikácie alebo procesy bežiace v systéme, podozrivú aktivitu a tak ďalej. Pri identifikácií známych indikátorov, získame dôležité informácie, ktoré ďalej len napomôžu potvrdiť a klasifikovať incident [18].

Výstupom tejto analýzy z tohto dôvodu teda bude podrobnejší popis korelovaných záznamov za daný časový rámec.

6.5 Klasifikácia

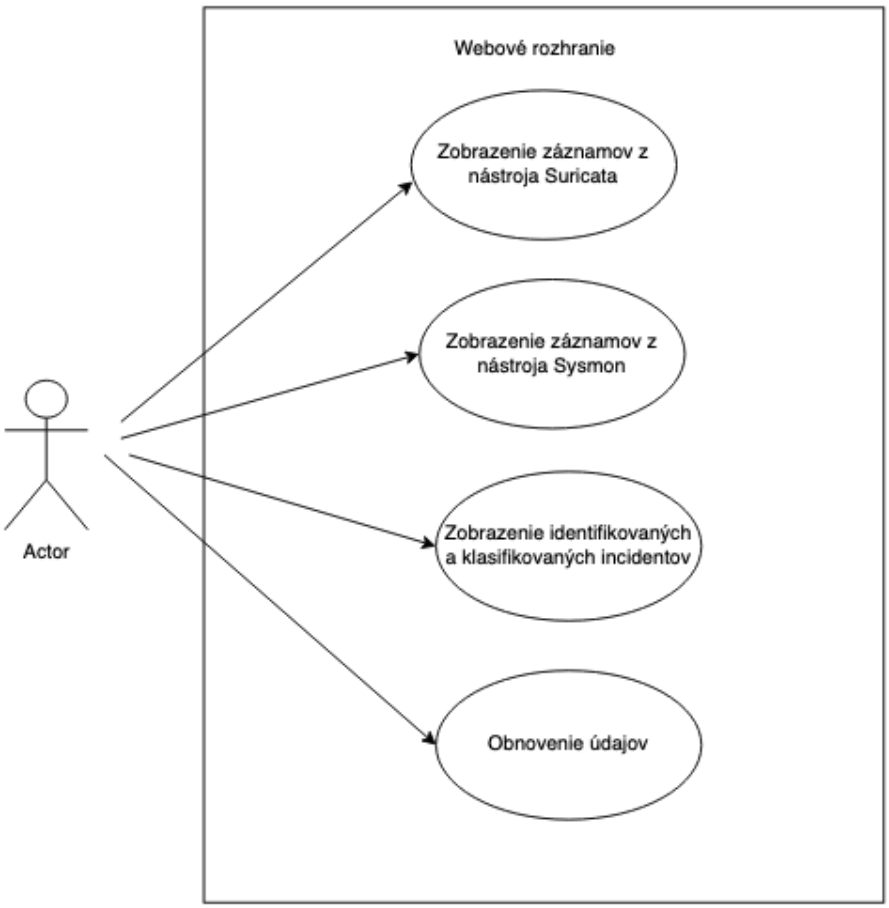
Doteraz sme prešli zber a spracovanie záznamov, ich koreláciu a zgrupovanie a nakoniec pomocou LLM aj získanie podrobnejšieho popisu čo sa udialo alebo či dané aktivity v záznamoch majú škodlivý charakter. Dostávame sa do poslednej časti. Po absolvovaní predošlých krokov, máme pripravené dostatočné základy, respektíve kontext na to, aby sme mohli znovu využiť veľký jazykový model. V tomto prípade poskytneme modelu znovu skupiny záznamov, kde rovnako ako v predošlom prípade dbáme na to, aby model bral do úvahy aj frekvenciu výskytu záznamov v stanovenom okamihu. Navyše ako doplnok mu poskytneme podrobnejšiu analýzu vykonanú modelom v predošlom kroku. Po načítaní celého vstupu budeme oproti predošlému kroku vyžadovať od modelu opätovnú analýzu záznamov. S tým, že budeme dbať na to, aby sa model zameral práve na identifikáciu rôznych kybernetických útokov. V prípade, že model zistí, že za daný časový okamih došlo k incidentu, chceme aby odhalený incident klasifikoval podľa Mitre ATT&CK *frameworku*.

Potrebné pre nás bude to, aká technika a procedúry v prípade incidentu boli vykonané a hlavne v akom štádiu útoku, sa útočník môže nachádzať. V opačnom prípade, ak ani po ďalšej analýze nebude potvrdený incident, napríklad ak sa jednalo v danom okamihu o bežnú aktivitu, nebudeme vyžadovať žiadnu klasifikáciu.

Výstupom tejto fázy bude JSON súbor obsahujúci všetky záznamy, s ich analýzami a aj klasifikáciami. Na základe týchto výsledkov bude neskôr možné jednoducho reagovať a predvídať ďalšie kroky v identifikovanom útoku a vďaka nájdeným indikátorom kompromitácie, bude pre bezpečnostné tímy jednoduché predísť podobnému incidentu.

6.6 Webové rozhranie

Posledným krokom v našej práci je vývoj jednoduchej webovej aplikácie na vizualizáciu záznamov generovaných Suricatou a Sysmonom, ako aj identifikovaných a klasifikovaných incidentov z nášho hlavného riešenia. Toto webové rozhranie bude poskytovať používateľom hlavne jednoduchý a intuitívny spôsob interakcie s údajmi pre získanie rýchlejšieho prehľadu o dianí na monitorovanom systéme. Webová aplikácia bude mať čistý a jednoduchý dizajn so zameraním na vizualizáciu záznamov. Čo sa týka funkcionalít, tých bude hneď viacero aj keď všetky sa budú týkať primárne zobrazovania dát. Používateľ si bude môcť zobraziť všetky výstrahy, ktoré boli vygenerované nástrojom Suricata vrátane zdrojových a cieľových IP adries, portov a protokolov. Keďže daných výstrah môže byť za určitý čas veľké množstvo, používateľovi sa zobrazia v skrátenom tvare a po kliknutí na jeden z nich, sa mu daný záznam rozbalí. To isté platí aj pre nástroj Sysmon , kde si používateľ bude taktiež vedieť zobraziť vygenerované záznamy o zachytených udalostiach vrátane vytvárania procesov, sieťových pripojení a vytvárania súborov. Primárnou časťou bude sekcia s korelovanými záznamami, keďže tá bude poskytovať prehľad o identifikovaných a klasifikovaných incidentoch vrátane typu útoku a štádia útoku. Nakoniec, webová aplikácia bude obsahovať aj možnosť aktualizovať údaje v reálnom čase.



Obr. 7: Diagram prípadov použitia.

7 Implementácia

Po prejdení si teoretickej časti a úspešnom dokončení návrhu nášho riešenia, sa v tejto kapitole pozrieme už na samotnú implementáciu. Povieme aké technológie sme použili a načo presne sme ich použili, ukážeme ako sme implementovali rôzne algoritmy a podobne.

7.1 Použité technológie

Počas vývoja nášho riešenia budeme využívať hneď niekoľko technológií, ktoré preto v tejto časti priblížime a vysvetlíme.

7.1.1 Suricata

Naša práca sa venuje využitiu UI pri kybernetických útokoch. Ako hlavný zdroj dát, respektíve záznamov na analyzovanie o tom čo sa práve udialo, nám bude slúži IDS systém.

Na monitorovanie sieťovej prevádzky a respektíve ako zdroj našich dát na analyzovanie, sme si zvolili práve nástroj Suricata. Rozhodli sme sa pre Suricatu najmä kvôli predošlým skúsenostiam s daným nástrojom, kde sa nám jeho používanie osvedčilo. Navyše, Suricata má veľkú a stále aktívnu komunitu ľudí, ktorí daný nástroj stále vyvíjajú. V našom rozhodnutí vybrať si Suricatu hrali hlavnú rolu aj jej výhody voči ostatným nástrojom IDS, ako je napríklad viacvláknová architektúra, čo jej umožňuje zvládnuť vyššie objemy prevádzky a poskytnúť rýchlejšie spracovanie paketov v porovnaní s jednovláknovými systémami. Vďaka tomu je vhodnejšou voľbou pre siete s vysokou sieťovou prevádzkou [19]. Suricata je taktiež nekomerčné riešenie, a poskytuje tak veľkú mieru modifikácie podľa potrieb používateľa. Nakoniec, všetky svoje záznamy, chybové hlásenia a podobne, ukladá do súboru, ktorý je kompatibilný s viacerými bežne používanými SIEM systémami.

Detekcia anomálií v prípade Suricaty je založená práve na pravidlách. V našom prípade sme použili odporúčanú sadu pravidiel *Proofpoint Emerging Threats Rules*[13]. Jedná sa o súbor pravidiel vyvinutý výskumným tímom zo spoločnosti Proofpoint. Tieto pravidlá sú navrhnuté tak, aby organizáciám umožňovali odhaliť a blokovať pokročilé hrozby. Výhodou týchto pravidiel je, že pokrývajú širokú škálu hrozieb vrátane malvéru, šírenia útokov, rôznych typov zneužití systému a zraniteľností, phishingu, distribuovaných útokov DoS, anomálií protokolov a aplikácií a podobne.

[13]https://rules.emergingthreats.net/

7.1.2 Sysmon

Nástroj Sysmon sme sa rozhodli použiť preto, aby sme obohatili sieťové záznamy z nástroju Suricata. Vďaka tomuto kroku sme získali väčší prehľad o tom čo sa udialo. Následne sme tak mohli jednoduchšie a presnejšie identifikovať a klasikovať prípadné hrozby.

Rozhodli sme sa použiť Sysmon najmä kvôli tomu, že poskytuje podrobné informácie o aktivitách vykonaných na systéme Windows, ktoré vieme použiť na potvrdenie potenciálneho incidentu, tým že doplníme kontext k IDS záznamom.

Najväčšia výhoda nástroja Sysmon spočíva v jeho schopnosti filtrovať udalosti. Môže byť nakonfigurovaný tak, aby odfiltroval známe udalosti, ktoré nie sú škodlivé alebo môžu reprezentovať falošný poplach. Tým pádom veľmi dôležitým aspektom pri implementácií Sysmonu je jeho konfigurácia. Predvolená konfigurácia pokrýva približne 60 % útokov popísaných vo *frameworku* MITRE ATT&CK s výraznými výnimkami pre sieťové pripojenia [20].

Preto sme pri konfigurácií nástroja Sysmon použili jeden z popredných a odporúčaných konfiguračných súborov, sysmon-modular [14], ktorý poskytuje ďalšie možnosti prispôsobenia spomínaného filtrovania udalosti, granulárny prístup k záznamom na základe ich identifikátora a nakoniec aj integráciu s viacerými poprednými SIEM systémami [20].

7.1.3 Konvertovanie EVTX záznamov

V predošlej kapitole sme si povedali výhody nástroja Sysmon a prečo ho plánujeme použiť ako doplnok k naším IDS záznamom. V práci sme aj uviedli, že Sysmon svoje záznamy neukladá do súboru vo formáte JSON, ako to je v prípade Suricaty, ale využíva formát EVTX. Sú to v podstate súbory, ktoré ukladajú údaje zaznamenaných udalostí v štruktúrovanom formáte v operačných systémoch Windows.

Z tohto hľadiska, bude pre nás potrebné EVTX záznamy konvertovať do formátu JSON. Týmto neskôr docielime jednotnosť v dátach a zároveň jednoduchšiu manipuláciu s nimi.

Na konvertovanie EVTX záznamov do JSON formátu existuje momentálne viacero nástrojov, preto sme sa rozhodli porozhliadnuť sa po dostupných riešeniach oproti programovaniu vlastného. Nakoniec, sme si zvolili nástroj evtx-toolkit [21]. Je to jednoduchý nástroj na čítanie a konvertovanie Sysmon záznamov z EVTX formátu do JSON formátu alebo taktiež aj do grafovej podoby. Vybrali sme si ho hlavne kvôli tomu, že je napísaný v programovacom jazyku Python, čo nám umožňuje upraviť si ho v prípade nejakých

[14] https://github.com/olafhartong/sysmon-modular

nedostatkov v kóde a taktiež vďaka jeho jednoduchosti.

7.1.4 Riešenia použité pri vývoji

Celé naše riešenie bude vyvíjané v programovacom jazyku Python. Jeho veľkou výhodou a zároveň aj dôvodom prečo sme sa rozhodli pre tento jazyk je jeho široké spektrum využitia od umelej inteligencie cez automatizáciu až k webovým aplikáciám.

Ďalej, po zanalyzovaní všetkých získaných záznamov, si ich budeme chcieť jednoducho prezerať, preto v našej práci implementujeme jednoduché webové rozhranie na vizualizáciu našich výsledkov. Takzvaný *back-end* tejto webovej aplikácie budeme implementovať pomocou *frameworku* Flask[15]. Zvolili sme si ho najmä kvôli tomu, že sa jedná o ľahký, voľne dostupný webový vývojový *framework*, ktorý je taktiež aj napísaný v programovacom jazyku Python.

Nakoniec, pre *front-end* sme sa rozhodli použiť základne webové programovacie jazyky, HTML a CSS, a to hlavne kvôli ich jednoduchosti.

7.2 Vývoj

V tejto časti sa budeme zaoberať vývojom nášho riešenia. Vysvetlíme si ako a kde sme použili spomínané technológie a ako sme implementovali náš návrh.

7.2.1 Spracovávanie záznamov

Prvým krokom v našom riešení je konvertovanie všetkých záznamov vygenerovaných nástrojmi Suricata a Sysmon, do rovnakého formátu. Keďže Sysmon svoje záznamy ukladá do EVTX formátu, budeme ich musieť konvertovať do prijateľnejšieho JSON formátu. Na toto konvertovanie použijeme spomínaný nástroj evtx-toolkit, ktorý je ďalej popísaný v predchádzajúcej kapitole. Ako vstup do tohto nástroja je potrebné vložiť exportované záznamy a ako výsledok spracovania, získame dáta v požadovanom JSON tvare. Po konvertovaní záznamov do potrebného tvaru, sa dostávame do ďalšieho bodu a tou je filtrácia. Hovorili sme si, že Suricata generuje dva typy výstupu, toky a výstrahy a všetky tieto výstupy sa ukladajú do jedného spoločného súboru. Keďže nás zaujímajú hlavne vygenerované výstrahy, po načítaní daného súboru budeme tieto záznamy filtrovať. Výstrahy sú Suricatou označované typom “alert”.

Definujeme si preto algoritmus vo funkcií *filter_suricata_alert*, ktorý začné tým, že si načíta súbor a bude prechádzať všetky jeho záznamy. V prípade, že typ záznamu je deklarovaný ako “alert”, tak si ho uloží a pokračuje ďalej, viď výpis čislo 1. V opačnom prípade, keď sa nejedná o výstrahu, algoritmus daný záznam ignoruje.

[15]https://flask.palletsprojects.com/en/3.0.x/

```
with open(input_file, 'r') as f:
        for line in f:
            json_obj = json.loads(line)
            if json_obj.get('event_type') == "alert":
                function()
```

Výpis 1: Filtrovanie Suricata záznamov

V prípade Sysmon záznamov je to veľmi podobné. Sysmon monitoruje každú zmenu, ktorá sa udeje na monitorovanom systéme. Nie však všetky záznamy, ktoré generuje sú relevantné. Avšak, niektoré anomálie na systéme spôsobia, že sú aktivované pravidlá, ktoré reprezentujú podozrivé aktivity. V takom prípade, vygenerovaný záznam obsahuje aj dané aktivované pravidlo. Z tohto dôvodu si definujeme algoritmus *filter_sysmon_alert*, ktorý slúži na filtrovanie Sysmon záznamov, viď výpis číslo 2. V princípe je totožný s algoritmom na filtrovanie Suricata záznamov. Načítame si Sysmon súbor prekonvertovaný do JSON formátu a budeme prechádzať všetky jeho záznamy. V prípade, že záznam obsahuje názov pravidla, ktoré ho spustilo, tak si ho uložíme. V opačnom prípade záznam ignorujeme.

```
with open(input_file, 'r') as f:
        for line in f:
            json_obj = json.loads(line)

            if json_obj.get('EventData_RuleName') != "-":
                add_hours(json_obj['EventData_UtcTime'], 2)
                function()
```

Výpis 2: Filtrovanie Sysmon záznamov

Pri týchto záznamoch je ešte dôležité spomenúť, že používajú UTC formát, ktorý je o 2 hodiny pomalší oproti časovému pásme na našom území. Preto, pri odchytení záznamu, modifikujme jeho časovú pečiatku aby bola pre nás relevantná.

Nakoniec, takto načítané, spracované a filtrované záznamy môžeme posunúť do ďalšej fázy.

7.2.2 Korelácia a Redukcia

Po načítaní a filtrovaní záznamov sa dostávame do ďalšej fázy, v ktorej sa budeme snažiť korelovať získané záznamy na základe času a pripraviť ich na nasledujúcu časť, kde budeme dané dáta analyzovať.

Prvým krokom pri korelácií vygenerovaných záznamov, bude vytvorenie takzvaných skupín záznamov, kde každá skupina bude balík všetkých záznamov, ktoré sa za vybraný časový rámec udiali. Najprv si vytvoríme globálnu premennú, ktorá bude reprezentovať

dĺžku časového rámca. Ďalej sme implementovali algoritmus vo funkcií *correlate*, ktorý si načíta filtrované záznamy zo Suricaty, viď výpis číslo 3. Na to aby vytvoril spomínané balíky záznamov, si najprv nastaví počiatočný čas, čo bude čas prvého vygenerovaného záznamu zo Suricaty, a konečný čas, ktorý bude jednoducho štartovací čas plus spomínaná globálna premenná udávajúca dĺžku časového rámca. Neskôr, algoritmus prechádza záznamy, a všetky záznamy, ktoré podľa ich časových pečiatok spadajú do uvedeného časového rozpätia, sa pridajú do balíka. V prípade, že algoritmus dostane na rad záznam, ktorého časová pečiatka je väčšia ako je momentálne nastavené časové rozpätie, tak sa štartovacia a konečná hodnota nastavia vzhľadom na daný záznam, a tak sa vytvorí nový balík. Algoritmus takto pokračuje až pokiaľ neprejde všetky záznamy.

```
...

start_time = datetime.strptime(suricata_logs[0]['timestamp'], "%Y-%m-%d %H:%M:%S.%f")
for suricata_log in suricata_logs:
    suricata_time = datetime.strptime(suricata_log['timestamp'], "%Y-%m-%d
        %H:%M:%S.%f")
    time_difference = abs(suricata_time - start_time)
    if time_difference < timedelta(seconds=time_threshold_seconds):
        function()
    else:
        start_time = datetime.strptime(suricata_log['timestamp'], "%Y-%m-%d
            %H:%M:%S.%f")

...
```

Výpis 3: Vytváranie Suricata balíkov na základe času

Takýmto spôsobom získame balíky záznamov, ktoré nám povedia čo všetko sa za určité časové obdobie udialo. Avšak, aby sme doplnili kontext k týmto záznamom, a dosiahli tak väčší prehľad, potrebujeme si podobným spôsobom spracovať aj Sysmon záznamy. Začneme získanými balíkmi Suricata záznamov. Algoritmus, ktorý je taktiež uvedený vo funckií *correlate*, viď výpis číslo 4, zoberie prvý balík záznamov a nastaví si počiatočný čas podľa prvého záznamu z balíka. Zároveň, jeho konečný bod bude počiatočný bod sčítaný s globálnou premennou vyjadrujúcou časové rozpätie. V predošlom prípade nám stačilo iba iterovať záznamy zo Suricaty, a tak si postupne vytvoriť dané balíky. Avšak, môže sa stať, že obdržíme Sysmon záznamy za omnoho dlhšie obdobie ako to bolo v prípade Suricaty. V takom prípade, lineárne prehľadávať hľadané Sysmon záznamy môže byť neefektívne. Z tohto dôvodu sme implementovali vyhľadávací algoritmus, taktiež nazývaný aj ako

binary-search[16]. To znamená, že využijeme algoritmus binary-search aby nám našiel index Sysmon záznamu, ktorý spadá do definovaného časového rozpätia. V prípade, že získame nenulový index, iterujeme záznamy od daného indexu pokiaľ ich časová pečiatka nebude väčšia ako nastavený časový rozsah. Všetky tieto záznamy sa pridajú do balíka a budú reprezentovať všetky udalosti, ktoré sa za ten čas udiali. Takto algoritmus pokračuje až pokiaľ neprejde všetky Suricata balíky.

```
...

for logs in batch_suricata:
    start_time = datetime.strptime(logs[0]['timestamp'], "%Y-%m-%d %H:%M:%S.%f")
    index = binary_search(sysmon_logs, start_time,
        timedelta(seconds=constants.SYSMON_THRESHOLD))
    while index != -1 and index < len(sysmon_logs):
        sysmon_log = sysmon_logs[index]
        sysmon_time = datetime.strptime(sysmon_log['EventData_UtcTime'], "%Y-%m-%d
            %H:%M:%S.%f")
        time_difference = abs(sysmon_time - start_time)
        if time_difference <= timedelta(seconds=constants.SYSMON_THRESHOLD):
            function()
        else:
            break

        index += 1
...
```

Výpis 4: Vytváranie Sysmon balíkov na základe času

Následne, po korelácií záznamov bude potrebné dané balíky záznamov redukovať aby LLM bol schopný tieto dáta spracovať. V tomto kroku, budeme znovu prechádzať balíky záznamov. Oba typy záznamov, aj tie z nástroja Suricata aj tie zo Sysmonu, obsahujú redundantné atribúty, ktoré neprinášajú žiadnu pridanú hodnotu. Napríklad, záznam zo Suricaty obsahuje atribút vyjadrujúci názov pravidla, ktoré bolo aktivované a taktiež obsahuje ďalší atribút, ktorý obsahuje identifikátor aktivovaného pravidla. Z tohto dôvodu, keď prechádzame záznamy z balíka, odstránime nadbytočné alebo menej významné atribúty zo záznamov, aby sme sa mohli zamerať na tie najrelevantnejšie a najdôležitejšie informácie v nich, viď výpis číslo 5. Zároveň, budeme si viesť zoznam unikátnych záznamov spolu s premennou vyjadrujúcou počet ich výskytov. Pri prechádzaní záznamov z balíka sa pozrieme či daný záznam sme už spracovávali, v prípade, že áno, tak len inkrementujeme

[16]https://en.wikipedia.org/wiki/Binary_search_algorithm

jeho hodnotu výskytu a pokračujeme na ďalší záznam. Ak sa v zozname nenachádza, pridáme ho a jeho výskyt nastavíme na hodnotu 1. Toto opakujeme pre všetky záznamy v balíku. Tento algoritmus, definovaný vo funckii *reduce_logs* takto pokračuje kým neprejde všetky balíky. Nižšie je uvedený príklad kódu ako to vyzerá algoritmicky. Uvedený je postup pre Suricatu ale v prípade Sysmonu je postup identický. Na konci dostaneme súbor balíkov, ktoré obsahujú všetky rôzne záznamy, ktoré boli vygenerované aj s počtom ich výskytov za dané časové rozpätie.

```
...
  for j in batch_suricata[i]:
          tmp = copy.deepcopy(j)
          if 'signature_id' in tmp['alert']:
              tmp['alert'].pop('signature_id')
          ...

          tmp = json.dumps(tmp)
          if tmp not in dic_suricata:
              dic_suricata[tmp] = {'log': j, 'occurrences': 1, 'start_time': j['timestamp']}
          else:
              dic_suricata[tmp]['occurrences'] += 1
...
```

Výpis 5: Redukovanie Suricata záznamov

7.2.3 Budovanie kontextu

V predošlej podkapitole sme si rozoberali korelovanie záznamov na základe času a taktiež, ako sme dané záznamy redukovali. Teraz sa pozrieme na to ako tieto zhromaždené záznamy analyzovať pomocou LLM. V prípade komerčných modelov, ako sú napríklad modely GPT-3.5-turbo-0125 alebo Gpt-4, využijeme knižnicu openai[17]. Vo výpise číslo 6 si môžeme pozrieť ako sme danú knižnicu použili na vytvorenie funkcie *gpt_request*, ktorá nám bude zabezpečovať posielanie príkazov a prijímanie odpovedi zo servera.

```
import openai

def gpt_request(message):
  try:
      response = client.chat.completions.create(
          model=constants.GPT_MODEL,
          messages=[
                  {"role": "user", "content": message}
```

[17]https://pypi.org/project/openai/

```
                ]
            )
        except Exception as e:
            exit(1)

        return response.choices[0].message.content
    ...
```

Výpis 6: Komunikácia s Openai

Pri testovaní budeme avšak používať aj nekomerčné modely ako je napríklad model Mistral-7b-instruct. Dané modely sa dajú jednoducho stiahnuť lokálne na zariadenie, avšak je potrebne si nastaviť vlastný lokálny server, na ktorom bude takýto model fungovať. Pre tieto modely využijeme voľne dostupný llama server[18]. Jedná sa o ľahký http server kompatibilný s OpenAI API, ktorý je možné využiť na hosťovanie a interagovanie s LLM.

V tomto prípade sme implementovali druhý spôsob interakcie s modelmi a to pomocou Python knižnice requests [19], viď výpis číslo 7. Tá nám umožňuje posielať POST požiadavky na špecifikovaný server a zároveň vieme cez danú knižnicu aj spracovať odpoveď zo servera. Daný kód je možné nájsť vo funckcií *llama_request*.

```
import requests

def llama_request(message):
    url = f"http://___URL___/completion"
    req_json = {
        "stream": False,
        "n_predict": 750,
        "temperature": 0.7,
        "stop": [
            "</s>",
            "L:",
            "U:"
        ],
        "repeat_last_n": 256,
        "repeat_penalty": 1.18,
        "top_k": 40,
        "top_p": 0.95,
        "tfs_z": 1,
        "typical_p": 1,
        "presence_penalty": 0,
```

[18]https://github.com/ggerganov/llama.cpp
[19]https://docs.python-requests.org/en/latest/index.html

```
    "frequency_penalty": 0,
    "mirostat": 0,
    "mirostat_tau": 5,
    "mirostat_eta": 0.1,
    "grammar": "",
    "n_probs": 0,
    "prompt":"This is a conversation between U and L.\nU:{}\n L:".format(message)
}
try:
    res = requests.post(url, json=req_json)
    result = res.json()["content"]
    return result
except:
    exit(1)
...
```

Výpis 7: Komunikácia s lokálnym modelom

Teraz keď už máme implementované funkcie na komunikáciu s modelmi, pozrieme sa aké vstupy a príkazy im budeme posielať. V tomto momente máme dostupné Suricata a Sysmon balíky rozdelené na základe špecifikovaného časového rozpätia. Postupne zoberieme každý balík a použijeme ho na analýzu pomocou LLM. V tomto kroku, budeme od modelu vyžadovať všeobecnú analýzu logov, či záznamy spolu súvisia a nech dohľadá indikátory kompromitácie. To dosiahneme tak, že si vytvoríme premennú *message*, ktorá bude obsahovať príkazy, čo má model robiť. Ďalej tieto príkazy zreťazíme s balíkom záznamov, viď výpis číslo 8. Tým, že sme v predchádzajúcom kroku záznamy redukovali, v príkaze musíme modelu jasne špecifikovať nech berie do úvahy aj počet výskytov analyzovaných záznamov. V skratke, takýto príkaz vyzerá nasledovne a je uvedenený vo funkcií *get_llm_alert_info*.

```
message = "Your task is to perform log analysis based JUST on provided logs which happened
    during {} seconds window.".format(constants.FILTER_THRESHOLD)
message += " Suricata logs that were identified: "
for key, value in dic_suricata.items():
    message += " LOG: \n"
    sur_log = value['log']
    sur_occur = value['occurrences']
    message += "\nOCCURRED {} times.\n".format(sur_occur)

message += " Sysmon logs that were identified: "
for key, value in dic_sysmon.items():
    message += " LOG: \n"
```

```
        sys_log = value['log']
        sys_occur = value['occurrences']
        message += "\nOCCURRED {} times.\n".format(sys_occur)
    message += '''
        Your task is to:
            - Analyze the suricata and sysmon logs if they have anything in common. In case
              sysmon/suricata logs are missing, continue without them.
            - Identify if the logs represent a security incident. Take into account also the number
              of occurences of each log.
            - Describe the potential incident.
            - Perform threat intelligence search on found IoCs.
        Use only provided information. Remember to reply shortly and precisely.V tomto
    '''
    ...
```

Výpis 8: Komunikácia s lokálnym modelom

Po analyzovaní záznamov modelom, obdržíme odpoveď obsahujúcu ich stručnú analýzu. Túto odpoveď uložíme do spracovaného balíka a algoritmus pokračuje ďalej až pokiaľ nespracuje všetky balíky záznamov.

7.2.4 Klasifikácia

Po absolvovaní predošlých krokov sa dostávame do poslednej časti. Na finálnu identifikáciu a klasifikáciu, znovu využijeme jazykový model rovnako ako v predošlom prípade. Postupne budeme iterovať všetky balíky záznamov. Vytvoríme si ďalšiu lokálnu premennú *message*, ktorá bude obsahovať príkazy a dáta, ktoré má model spracovať. Od modelu budeme očakávať, že znovu zanalyzuje všetky podklady a nech posúdi či sa v danom časovom okne stal bezpečnostný incident. Zároveň, v prípade, že sa jedná o incident, budeme chcieť nech nám poskytne techniku a takiež taktiku z Mitre ATT&CK, ktorá najlepšie popisuje identifikovaný incident, viď výpis číslo 9. Navyše, na doplnenie kontextu poskytneme modelu mimo záznamov aj ich vyhotovenú analýzu z predošlého kroku. Tento algoritmus je možné nájsť vo funkcií *get_llm_classification*.

```
message = "Analyze if a security incident is happening based on provided logs which happened
    during {} seconds window.".format(constants.FILTER_THRESHOLD)
message += " Suricata logs that were identified: "
for key, value in dic_suricata.items():
    message += " LOG: \n"
    sur_log = value['log']
    sur_occur = value['occurrences']
    message += "\nOCCURRED {} times.\n".format(sur_occur)
```

```
    message += " Sysmon logs that were identified: "
    for key, value in dic_sysmon.items():
        message += " LOG: \n"
        sys_log = value['log']
        sys_occur = value['occurrences']
        message += "\nOCCURRED {} times.\n".format(sys_occur)

    message += '''
        In case of a security incident, your task is to:
        - Decide if the logs represent a security incident/attack.
        - Based on Mitre Att&ck, tell me which technique best describes the discovered incident.
        - Based on choosen Mitre Att&ck technique, tell me which attack tactic/kill chain stage does
            the technique belongs to.
        - Give me score from 1 to 10 how critical it is.
        Remember to take into account also the number of occurrences of each log.
        Try to reply as precisely as possible. Reply just with the Mitre Att&ck information and
            score. In case it is NOT a incident, *ignore* the previous commands.
    '''
    message += ". For the analysis use also following additional information:
        {}".format(conf_message)
    ...
```

Výpis 9: Identifikácia a klasifikácia

Algoritmus takto pokračuje pokiaľ neprejde všetky balíky záznamov a všetky nespracuje. Konečný výstup tejto fázy bude JSON súbor, viď výpis číslo 10, obsahujúci zoznam balíkov so svojími záznamami, ich analýzami a aj klasifikáciami. Nižšie je možné vidieť ukážku, ako vyzerá štruktúra vygenerovaného súboru.

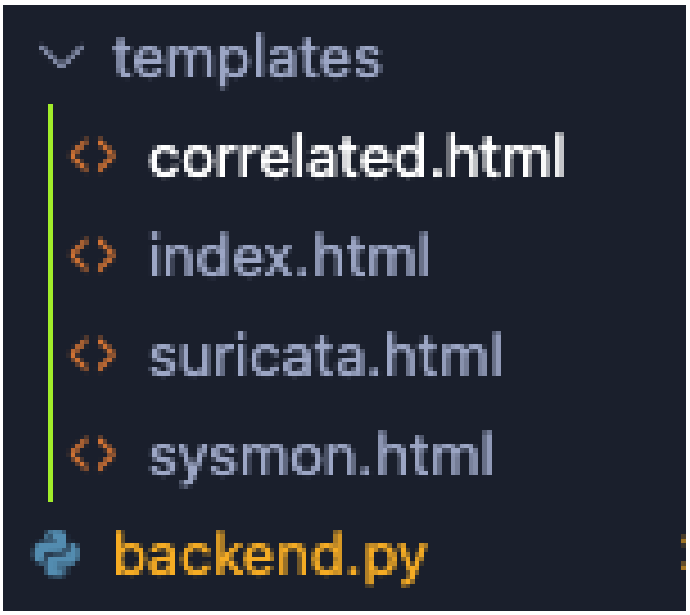
```
[
    {
        "start time": "2024-02-21 19:53:57.807117",
        "end time": "2024-02-21 19:55:57.807117",
        "info": "LOG INFO GENERATED FROM LLM",
        "mitre": "MITRE CLASIFICATION FROM LLM",
        "suricata_logs" :[],
        "sysmon_logs":[]
    },
    {},
    {},
    ...
]
```

Výpis 10: Vygenerovaný súbor

Na základe týchto výsledkov bude neskôr možné jednoducho reagovať a predvídať ďalšie kroky v identifikovanom útoku.

7.2.5 Webové rozhranie

V prípade webového rozhrania sme sa riadili spomínaným diagramom prípadov použitia. Toto rozhranie teda slúži na zobrazenie spracovaných záznamov do jednoduchšej podoby. Všetky dáta zobrazené na tomto rozhraní sú načítavané z JSON súboru, ktorý je vygenerovaný naším riešením po konečnom spracovaní a analýze. Celý *back-end* sme riešili pomocou *frameworku* Flask a *front-end* zase pomcou jazykov HTML a CSS. Štruktúra súborov teda tohto webového rozhrania vyzerá nasledovne:



Obr. 8: Suborová štruktúra.

Templates je priečinok obsahujúci HTML súbory pre každú sekciu, ktorú vykresľujeme a *backend.py* obsahuje celú *back-endovú* logiku.

Web má definovaných dokopy 5 ciest, viď výpis číslo 11.

```
@app.route('/')
def index():
    ...
@app.route('/suricata', methods=['GET'])
def suricata_json_data():
    ...
@app.route('/sysmon', methods=['GET'])
def sysmon_json_data():
    ...
```

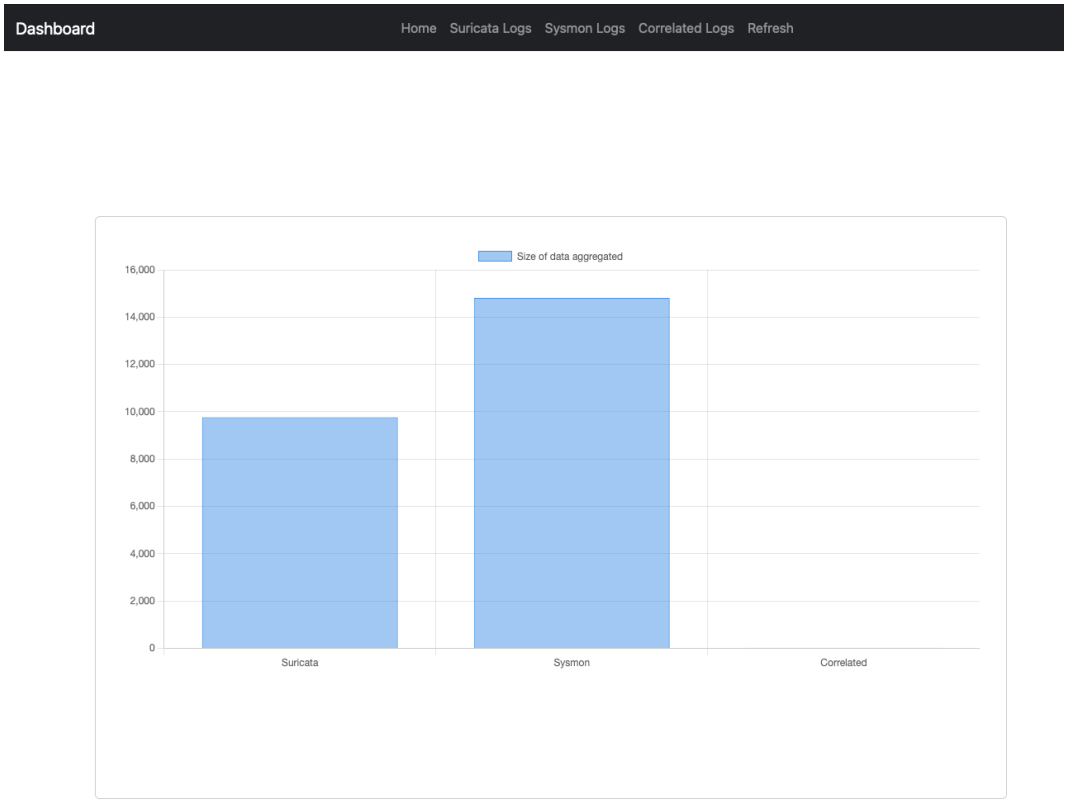
```
@app.route('/correlated', methods=['GET'])
def correlated_json_data():
    ...
@app.route('/refresh')
def refresh():
    ...
```

Výpis 11: Cesty webového rozhrania

Na úvodnej stránke, sa používateľovi vykreslí súbor *index.html*, ktorý zobrazí štatistiku spracovaných záznamov. Jedná sa o jednoduchý graf, ktorý sme implementovali pomocou programavacieho jazyka JavaScript, viď výpis číslo 12. Graf ukazuje celkový počet spracovaných Suricata záznamov, Sysmon záznamov a taktiež aj identifikovaných incidentov.

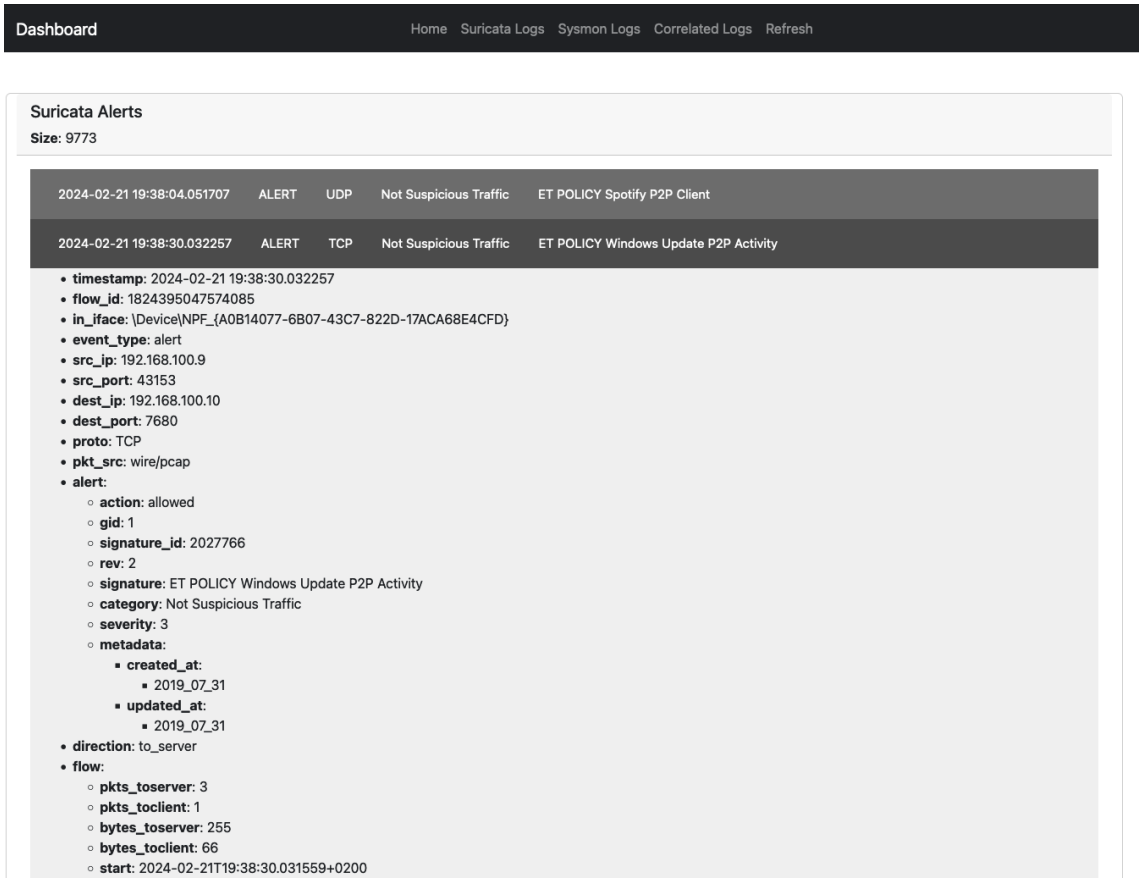
```
<script>
    const ctx = document.getElementById('myChart');
    new Chart(ctx, {
      type: 'bar',
      data: {
        labels: ['Suricata', 'Sysmon', 'Correlated'],
        datasets: [{
          label: 'Size of data aggregated',
          data: {{ lengths | safe }},
          borderWidth: 1
        }]
      },
      options: {
        scales: {
          y: {
            beginAtZero: true
          }
        }
      }
    });
  </script>
```

Výpis 12: vykreslenie grafu



Obr. 9: Úvodná stránka

V ďalšom okne, si používateľ bude môcť zobraziť všetky výstrahy, ktoré boli vygenerované nástrojom Suricata vrátane zdrojových a cieľových IP adries, portov a protokolov. Keďže daných výstrah môže byť za určitý čas veľké množstvo, všetky výsledky sa používateľovi zobrazia v skrátenom tvare. Následne po kliknutí na špecifický záznam, sa mu daný záznam rozbalí v štruktúrovanej forme.



Obr. 10: Vykreslenie záznamov z nástroja Suricata.

Na docielenie tohto výpisu voláme funkciu *get_attributes_suricata*, viď výpis čislo 13, ktorá si načíta JSON objekt reprezentujúci záznam a extrahuje odtiaľ základne atributy záznamu ako je jeho časová pečiatka, typ záznamu, použitý protokol, aktivované pravidlo a aj jeho kategóriu. Túto funkciu voláme pre každý záznam v cykle.

```
def get_attributes_suricata(data):
    attributes = []
    attributes.append(data['timestamp'])
    attributes.append(data['event_type'].upper())
    attributes.append(data['proto'])
    attributes.append(data['alert'].get('category'))
    attributes.append(data['alert'].get('signature'))
    return attributes
```

Výpis 13: Extrahovanie atributov

Tieto atributy sa ďalej posielajú do funkcie *print_suricata*, ktorá ich zobrazí v spomínanom zmenšenom okne. Po rozkliknutí daného okna sa používateľovi rozbalý celý záznam. Tento krok sme implementovali pomocou funkcie *print_full_json*. Najprv obdržíme na vstupe záznam, ktorý je reprezentovaný v tvare slovníka. Funkcia taktiež obsahuje premennú *html*, ktorá bude obsahovať základnú štruktúru ako bude záznam vyzerať vo webovom rozhraní. V cykle ďalej len iterujeme všetky jeho kľúče a hodnoty. Tieto dáta spolu s HTML príznakmi sa reťazia s premennou *html*, viď výpis číslo 14. Na výstupe, je tak HTML kód, ktorý sa posiela ďalej do súbora *suricata.html*.

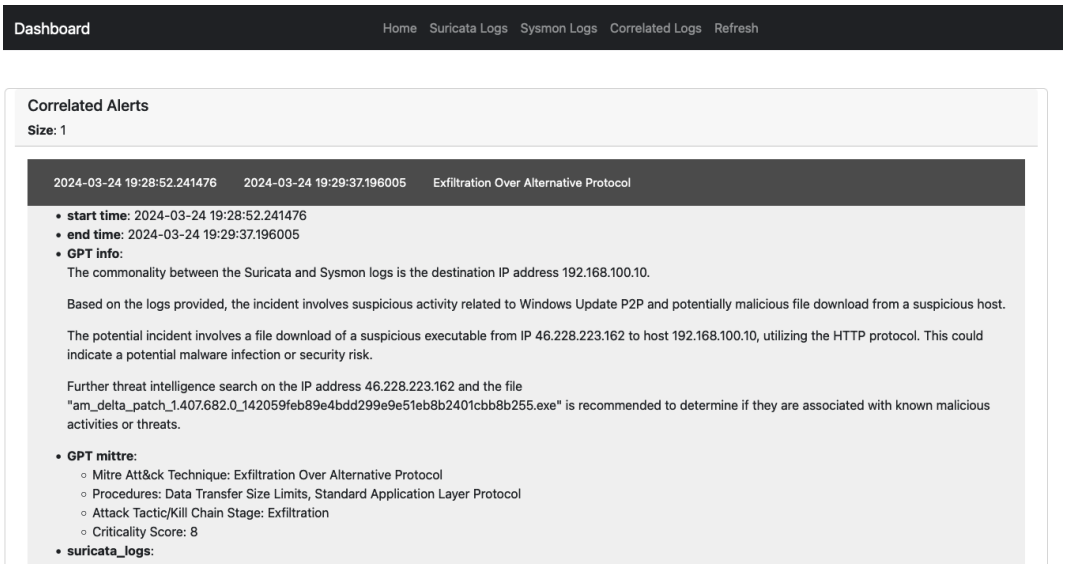
```
def print_full_json(data):
    html = '<div class="json">'
    html += '<ul>'
    for key, value in data.items():
        html += f'<li><span class="key"><strong>{key}</strong>: </span>'
        if isinstance(value, dict):
            html += print_full_json(value)
        elif isinstance(value, list):
            html += '<ul>'
            for item in value:
                html += f'<li>{item}</li>'
            html += '</ul>'
        else:
            html += str(value)
        html += '</li>'
    html += '</ul>'
    html += '</div>'
```

```
return html
```

Výpis 14: Vykreslenie celého záznamu

Veľmi podobným spôsobom sme implementovali aj okno na zobrazenie Sysmon záznamov. Aj v tomto prípade budú záznamy vykreslené v skrátenom tvare a po kliknutí rozbalené.

Používateľ má možnosť si zobraziť aj spracované záznamy, ktoré obsahujú identifikované a klasifikované incidenty s ich informáciami z analýzy. Aj v tomto prípade sa všetky dáta vykresľujú v skrátenom tvare rovnakým spôsobom ako to je v prípade Suricaty.



Obr. 11: Vykreslenie spracovných záznamov.

Poslednou funkcionalitou webového rozhrania je aktualizácia dát. Tá na pozadí zavolá funkciu *load_logs*, ktorá opätovné načíta všetky záznamy a údaje na stránke sa aktualizujú.

8 Testovanie a výsledky

V predošlej kapitole sme si povedali o použitých technológiách a ako sme implementovali celé naše riešenie. V tejto časti budeme popisovať výsledky našej práce.

8.1 Testovacie scenáre

V prípade testovania sa zameriame na 3 základné merania:

- V prípade komerčných modelov nás bude zaujímať cena za analýzu záznamov
- Podrobíme viaceré komerčné aj voľne dostupné modely sade záznamov, ktoré zachytávajú simulované kybernetické útoky
- S vybraným jazykovým modelom sa pozrieme aj na vplyv modifikácie časového rámca pri korelácií záznamov

8.2 Simulované útoky

Na otestovania LLM je potrebné nasimulovať viaceré kybernetické útoky aby sme zistili celkovú úspešnosť pri rôznych typoch útokov. V rámci testovania sme sa rozhodli simulovať nasledovné útoky:

- *Network sniffing (Mitm)* - technika používaná na zachytenie a analýzu sieťovej prevádzky. Dá sa použiť na rôzne účely, napríklad na krádež citlivých informácií alebo vloženie škodlivého kódu.
- *Denial of Service* - útok, ktorého cieľom je zneprístupniť službu tým, že ju zahltí prevádzkou alebo požiadavkami.
- Útok hrubou silou na RDP - útok, ktorý sa pokúša uhádnuť heslo služby protokolu vzdialenej pracovnej plochy (RDP) skúšaním rôznych kombinácií. Môže sa použiť na získanie neoprávneného prístupu do systému alebo siete.
- *Dir listing* - útok, ktorý využíva nesprávnu konfiguráciu webového servera na výpis obsahu adresára. Môže sa použiť na zhromažďovanie informácií o serveri alebo na sťahovanie citlivých súborov.
- *Cross-site scripting* - zraniteľnosť webovej aplikácie, ktorá umožňuje útočníkovi vložiť na webovú stránku škodlivý kód. Môže sa použiť na odcudzenie používateľských údajov, presmerovanie používateľov na škodlivé stránky alebo vykonávanie iných škodlivých akcií.

- Skenovanie portov - technika používaná na skenovanie rozsahu portov v cieľovom systéme alebo sieti s cieľom určiť, ktoré služby sú spustené, a ktoré sú zraniteľné.

8.3 Testovacie prostredie

Na otestovanie nášho riešenia je najprv potrebne si pripraviť a nakonfigurovať testovacie prostredie. Čo sa týka architektúri, tá sa skladá primárne z virtuálneho počítača s operačným systémom Windows. Virtuálny počítač bol nakonfigurovaný so 6 jadrami a 8 Gb pamäte RAM. Taktiež sme na počítačí povolili vzdialený prístup RDP aby sme boli schopný previesť útok hrubou silou. Na danom počítači sme inštalovali nástroj Suricata spolu s konfiguračným súborom *Proofpoint Emerging Threats Rules*[20] obsahujúcim pravidlá na detekciu škodlivých anomálií. V prípade Sysmonu sme stiahli a použili konfiguračný súbor *symon-modular*[21]. Aby sme boli schopný otestovať aj sieťové útoky na webové aplikácie ako je spomínaný XSS útok, v testovacom prostredí sme nainštalovali a spustili zraniteľný webový e-shop *Juice Shop*[22]. Takto nastevené prostredie obsahuje všetký komponenty na simulovanie útokov. Tie budú monitorované nástrojmi Suricata a Sysmon, ktoré následne vygenerujú svoje záznamy.

V prípade analýzy záznamov pomocou LLM sme sa rozhodli použiť výkonnejší počítať. Jednálo sa o macbook pro so 16 Gb RAM a 12 jadrami. Na tomto počítačí sme v prípade testovania nekomerčných modelov stiahli a nainštalovali *llama-server* [23], ktorý slúži na lokálne spustenie a použitie jazykových modelov.

8.4 Testovacie modely

Ako sme spomínali, budeme testovať najmä komerčné a voľne dostupné LLM. Komerčné modely vyvíjajú a udržiavajú spoločnosti ako OpenAI, Anthropic a Cohere. Tieto modely sú často doladené pre špecifické prípady použitia a môžu poskytnúť vysokokvalitné výsledky pre komerčné aplikácie. Na druhej strane, nekomerčné modely sú vyvíjané výskumnou komunitou a sú voľne dostupné na použitie a úpravu. Tieto modely môžu byť cenným zdrojom pre výskumníkov a vývojárov, ktorí chcú experimentovať s jazykovými modelmi a vytvárať vlastné riešenia.

Nižšie sú uvedené použité modely aj s ich stručným popisom:

- Gpt-3.5-turbo-0125 - Jedná sa o model vyvinutý spoločnosťou OpenAI[24] a teda ide o

[20] http://rules.emergingthreats.net/open/suricata/
[21] https://github.com/olafhartong/sysmon-modular
[22] https://github.com/juice-shop/juice-shop
[23] https://github.com/ggerganov/llama.cpp
[24] https://openai.com/

výkonný a všestranný jazykový model. Je vhodný pre analztické úlohy vďaka svojej schopnosti porozumieť a generovať zložité jazykové vzory, čo umožňuje extrahovať dáta zo záznamov.

- Gpt-4 - Ďalšia generácia zo série GPT, ktorá je výkonnejšia a všestrannejšia ako jej predchodcovia. Tento model je navrhnutý tak, aby zvládol širokú škálu jazykových úloh a dá sa doladiť pre špecifické aplikácie.

- Mistral-7b-instruct - Tento model, vyvinutý spoločnosťou Hugging Face[25], je variantom série Mistral, čiže časťou LLM. Je špeciálne navrhnutý pre úlohy súvisiace s inštrukciami, kde ho je možno napríklad naučiť extrahovať špecifické informácie zo záznamov.

- Claude-3-haiku - Tento model sa dá použiť na vytváranie stručných a zmysluplných súhrnov údajov z dát, čo z neho robí cenný nástroj na identifikáciu vzorov a trendov.

- Gemma-7b-it - Gemma je jeden z novších modelov od spoločnosti Google, ktoré sú postavené na rovnakom základe ako modely Gemini. Je vhodný pre rôzne úlohy generovania textu vrátane odpovedí na otázky, sumarizácie a uvažovania.

Stručne povedané, každý z týchto jazykových modelov prináša jedinečné silné stránky a schopnosti, vďaka čomu sú vhodné pre naše testovacie účely. Testovaním a porovnávaním týchto modelov môžeme identifikovať to najlepšie riešenie pri identifikácií a klasifikácii kybernetických útokov.

8.5 Porovnanie viacerých jazykových modelov

V tejto kapitole si predstavíme výsledky našej práce. V rámci testovania sme si pripravili virtuálny počítač s vyššie spomínanou konfiguráciou. Ďalej sme simulovali na daný počítač útoky, ktoré sú bližšie popísane v predchádzajúcej kapitole. Vygenerované záznamy nástrojmi Suricata a Sysmon, sme po simuláciách zozbierali a poskytli ich nášmu nástroju. Tam sa tieto záznamy spracovali, prefiltrovali, korelovali a nakoniec analyzovali. Cieľom bolo otestovať viacero veľkých jazykových modelov a porovnať ich úspešnosť pri takomto type identifikovania a klasifikovania incidentov.

8.5.1 Skenovanie portov

Nižšie uvedená tabuľka predstavuje výsledky analýzy každého modelu pre útok skenovania portov.

[25]https://huggingface.co/

Tabuľka 2: Výsledky pre skenovanie portov

Model	**Technique**	**Tactic**
GPT 4	Network Service Discovery	Discovery
GPT 3.5 turbo 0125	Network Service Discovery	Discovery
Claude-3-haiku	Network Service Discovery	Reconnaissance
mistral-7b-instruct	Scanning IP Blocks	Reconnaissance
gemma-7b-it	Bruteforce	Reconnaissance

Ako môžeme vidieť, všetky LLM okrem gemma-7b-it správne identifikovali incident skenovania portov. GPT 4, GPT-3.5-turbo-0125 a mistral-7b-instruct presne mapovali techniku na taktiku *Discovery*. Claude-3-haiku identifikoval *Network Service Discovery*, ale klasifikoval taktiku ako *Reconnaissance*, čo je oveľa širšia kategória. Aj keď je to technicky správne, *Discovery* je špecifickejšia taktika v rámci prieskumu.

8.5.2 Útok hrubou silou na RDP

Tabuľka 3: Výsledky útoku hrubou silou na RDP

Model	**Technique**	**Tactic**
GPT 4	Commonly Used Port	Command and Control
GPT 3.5 turbo 0125	Remote Services	Execution
Claude-3-haiku	RDP Bruteforce	Lateral Movement
mistral-7b-instruct	Bruteforce	Initial Access
gemma-7b-it	Remote Services	Execution

Ako vidíme, väčšina LLM presne neklasifikovala útok hrubou silou na protokol RDP. GPT 4, GPT-3.5-turbo-0125 a gemma-7b sa zamerali skôr na službu RDP, než na zlý zámer (hrubé vynútenie prístupu). Claude-3-haiku a mistral-7b-instruct sa najviac priblížili tým, že spomenuli použitú hrubú silu.

8.5.3 Denial of Service (Dos)

Tabuľka 4: Výsledky útoku DoS

Model	**Technique**	**Tactic**
GPT 4	Active Scanning	Reconnaissance
GPT 3.5 turbo 0125	Network Denial of Service	Impact
Claude-3-haiku	Network Denial of Service	Impact
mistral-7b-instruct	-	-
gemma-7b-it	-	-

V tomto prípade dosiahli najpresnejšie výsledky GPT-3.5-turbo-0125 a Claude-3-haiku. Správne identifikovali taktiku *Impact* spojenú s útokom DoS. Aj keď to nie je také presné, GPT-4 identifikovala techniku *Active Scanning*, ktorú by bolo možné použiť na prieskumné účely, ale priamo nesúvisí s útokom DoS. Mistral-7b-instruct a gemma-7b-it v tomto prípade nezvládli spracovať kvantum záznamov, ktoré boli počas tohto útoku vygenerované.

8.5.4 Network Sniffing

Tabuľka 5: Výsledky pre útok Network Sniffing

Model	**Technique**	**Tactic**
GPT 4	Amplification	Impact
GPT 3.5 turbo 0125	Network Sniffing	Reconnaissance
Claude-3-haiku	Denial of Service: Amplification	Impact
mistral-7b-instruct	-	-
gemma-7b-it	Data Exfiltration Through VPN	Execution

Tu GPT-3.5-turbo-0125 vyniká správnou identifikáciou taktiky *Reconnaissance* a príslušnej techniky *Network Sniffing*. Zatiaľ čo iné LLM sa zameriavali na aspekty ako je amplifikácia (DoS) alebo exfiltračné techniky, ktoré sú súčasťou útoku ale nepopisujú priamo útok, ktorý sme simulovali. V prípade modelu mistral-7b-instruct sme sa ani odpovede nedočkali, keďže nebol schopný spracovať veľké množstvo záznamov, ktoré boli vygenerované a poslane na analýzu.

8.5.5 Cross-site scripting (XSS)

Tabuľka 6: Výsledky pre XSS

Model	**Technique**	**Tactic**
GPT 4	Command and Scripting Interpreter:Javascript	Execution
GPT 3.5 turbo 0125	Exploitation for Client Execution	Initial Access
Claude-3-haiku	Command and Scripting Interpreter:Javascript	Execution
mistral-7b-instruct	Inject	Execution
gemma-7b-it	Inject	Execution

Všetky LLM až na menšiu výnimku pri modely GPT-3.5-turbo-0125, správne identifikovali a zaklasifikovali útok. Modely správne identifikovali taktiku *Execution* a buď špecifickú techniku *Command and Scripting Interpreter: JavaScript* alebo všeobecnú injekčnú techniku. GPT-3.5-turbo-0125 omylom klasifikoval taktiku ako *Initial Access*, pričom identifikoval relevantnú techniku, do ktorej taktiež spadá útok XSS.

8.5.6 Dir Listing

Tabuľka 7: Výsledky pre Dir Listing

Model	**Technique**	**Tactic**
GPT 4	Remote System Discovery	Reconnaissance
GPT 3.5 turbo 0125	Attempted Information Leak	Initial Access
Claude-3-haiku	Data from Information Repositories	Reconnaissance
mistral-7b-instruct	-	-
gemma-7b-it	-	-

Všetky LLM dosiahli čiastočný úspech pri klasifikácii tohto útoku. Všetky modely identifikovali taktiku buď ako *Reconnaissance* alebo *Initial Access*. Je to čiastočne správne, pretože pokus o výpis adresára zhromažďuje informácie o systéme, ktoré možno použiť na účely prieskumu alebo ako počiatočný krok na získanie prístupu. GPT-4 presne identifikoval

techniku súvisiacu s objavovaním vzdialeného systému. GPT-3.5-turbo-0125 sa zameral skôr na únik informácií, čo je potenciálny dôsledok úspešného zneužitia, ale nie nevyhnutne použitej techniky. Nakoniec, modely gemma-7b-it a mistral-7b-instruct nedokázali spracovať objem dát vygenerovaný počas útoku.

8.5.7 Zhrnutie

Experiment priniesol viaceré výsledky. Zatiaľ čo niektoré LLM dosahovali povzbudivé výsledky, iné zase nevedeli presne definovať o aký typ útoku ide. Taktiež je podstatné spomenúť, že voľne dostupné modely mali v niektorých prípadoch aj problém so spracovaním veľkého množstva záznamov. Presnosť klasifikácie LLM sa môže líšiť v závislosti od modelu a typu útoku. Niektoré modely fungovali dobre vo väčšine prípadov, zatiaľ čo iné vynikali iba v špecifických scenároch. Môže to byť spôsobené niekoľkými faktormi, ako sú obmedzenia LLM, zložitosť útokov, kvalita alebo množstvo záznamov. Pre komplexnú a kvalitnejšiu klasifikáciu bezpečnostných incidentov, môže byť napríklad potrebné dotrénovanie LLM na danú úlohu.

8.6 Cenové porovnanie veľkých jazykových modelov

Keďže sme sa rozhodli otestovať viaceré jazykové modely, medzi ktoré patria aj tie platené, dobrou otázkou pri pracovaní s nimi je ich cena. Ceny za používanie komerčných jazykových modelov sa môžu líšiť v závislosti od poskytovateľa a úrovne používania. Nižšie je uvedená tabuľka s plátennými modelmi, ktoré sme testovali a ich cena.

Tabuľka 8: Porovanie cien komerčných jazykových modelov

Model	**Vstup (Input)**	**Výstup (Output)**
Gpt-3.5-turbo-0125	0.5 eur / 1 000 000 tokenov	1.5 eur / 1 000 000 tokenov
Gpt-4	30 eur / 1000 000 tokenov	60 eur / 1 000 000 tokenov
Claude-3-haiku	0.25 eur / 1 000 000 tokenov	1.25 eur / 1 000 000 tokenov

Tabuľka obsahuje informácie o rôznych modeloch a taktiež aká je cena ich používania. Samotná cena súvisí s viacerými faktormi ale najmä od veľkosti vstupu, to znamená, koľko tokenov obsahuje správa, ktorú má model spracovať a taktiež od veľkosti výstupu, ktorý hovorí o veľkosti textu, ktorý model vygeneruje ako odpoveď pre zadaný vstup. Ako si môžeme všimnúť, najdrahší model zo spomenutých je práve model Gpt-4, kde 1 000 000 tokenov na vstupe stojí v prepočte 30 eur. Zároveň, vygenerovaná odpoveď vo veľkosti 1 000 000 tokenov stojí 60 eur. Najlepšie sú na tom Gpt-3.5-turbo-0125 a Claude-3-haiku, kde ich cena vstupu a výstupu je iba zlomok z ceny modelu Gpt-4.

Ako sme spomínali, pri analýze naším nástrojom sa posielajú balíky Suricata a Sysmon záznamov do jazykového modelu na spracovanie. Počet balíkov závisí od časového rámca, podľa ktorého budeme chcieť dané záznamy korelovať a taktiež aj za aké dlhé obdobie sme získali záznamy. Každá správa obsahujúca príkazy, ktorými sa má model riadiť spolu so záznamami na spracovanie je zložená v priemere z 12 000 tokenov. V procese spracovávania posielame dve takéto správy, prvú aby nám LLM zanalyzoval záznamy a potom druhú, v ktorej záznamy už identifikuje a klasifikuje. Tým pádom, každé kolo sa pošlú dve správy o priemernej veľkosti 12 000 tokenov. Do toho musíme zarátať aj veľkosť textu, ktorý nám vygeneruje LLM po spracovaní. Aj v tomto prípade v každom kole dostaneme dve odpovede s priemernou veľkosťou 300 tokenov. Pri našom testovaní sme v priemere analyzovali 20 balíkov záznamov, v ktorých sme sa snažili identifikovať simulované útoky. To znamená, že 20 krát sme posielali po dve správy modelu. Na výpočet priemernej ceny teda musíme vziať do úvahy počet spracovaných a vygenerovaných tokenov a náklady modelu.

*Celkovo_spracovaných_tokenov = (Priemerná_veľkosť_správy * Počet_správ) * Počet_analýz*

*Celkovo_spracovaných_tokenov = (12 000 * 2) * 20 = 480 000 tokenov*

*Celkovo_vygenerovaných_tokenov = (Priemerná_veľkosť_odpovede * Počet_správ) * Počet_analýz*

*Celkovo_vygenerovaných_tokenov = (500 * 2) * 20 = 20 000 tokenov*

Tabuľka nižšie obsahuje priemerne ceny použitia komerčných modelov naším nástrojom po dosadení cien z tabuľky 8.

Tabuľka 9: Celková cena modelov

Model	**Vstup (Input)**	**Výstup (Output)**	**Cena dokopy (Total Cost)**
Gpt-3.5-turbo-0125	0.25 eur	0.03 eur	0.28 eur
Gpt-4	15 eur	1.2 eur	16.2 eur
Claude-3-haiku	0.125 eur	0.025 eur	0.15 eur

Z tabuľky vyplýva, že najlacnejší model na použitie je model Claude-3-haiku, ktorého cena za spracovanie a analýzu záznamov pre identifikáciu a klasifikáciu prípadných incidentov je 0.15 eur. Najhoršie na tom je model Gpt-4, ten vďaka svojím vysokým nákladom spotreboval 16.2 eur za analýza dát. Táto cena je spôsobená najmä veľkosťou dát na ktorej

bol model Gpt-4 natrénovaný. Platí, že čím väčší model, tým lepšie výsledky ponúka, avšak za cenu výpočtovej sily.

8.7 Účinnosť LLM pri zväčšení časového rámca

V návrhu sme spomínali, že pri spracovávaní Suricata a Sysmon záznamov, prebieha aj ich korelácia na základe času. Táto korelácia zahŕňa zoskupovanie daných záznamov, ktoré sa vyskytnú v rámci určitého časového rámca. Táto technika pomáha analytikom identifikovať potenciálne súvislosti medzi zdanlivo izolovanými udalosťami. Taktiež sme si hovorili o možných problémoch, ktoré môžu nastať pri nesprávne nastavenom časovom rámci. V predošlom testovaní sme si overovali, ktoré LLM sú schopné identifikovať a klasifikovať simulované útoky. Dáta, ktoré im boli poskytnuté boli korelované s časovým rámcom vo veľkosti 1 minúty. 1-minútový časový rámec ponúka vysokú presnosť pri identifikácii súvisiacich udalostí. Ak sa napríklad skenovanie portov alebo pokus o útok hrubou silou na protokol RDP uskutoční do minúty, je pravdepodobnejšie, že budú súčasťou rovnakého útoku v porovnaní s 30-minútovým rámcom. Výhodou väčších časových okien je, že môžu zachytiť širší kontext potenciálne škodlivej aktivity. Napríklad, skenovanie portov sa môže uskutočniť niekoľko minút pred útokom DoS a ich spoločná analýza v rámci 15-minútového okna by mohla odhaliť koordinovaný vzor útoku. Na druhú stranu môže spôsobiť veľký šum, čo môže narušiť analýzu a úspešnosť pri odhalení.

V tejto fáze navyše budeme používať len jeden model oproti viacerým. Na základe nášho predchádzajúceho porovnania sa model GPT-3.5-turbo-0125 ukázal ako sľubný kandidát pre tento experiment, keďže preukázal dobrú schopnosť identifikovať a klasifikovať rôzne útoky vrátane skenovania portov, pokusov o DoS a podobne. To naznačuje jeho potenciál na spracovanie záznamov a rozpoznávanie škodlivých aktivít. Posledným faktorom bola jeho rýchlosť odpovedania, cena za výkon a taktiež veľkosť akceptovaného vstupu.

Tabuľky nižšie reprezentujú výsledky analýzy modelom gpt-3.5-turbo-0125. Záznamy sme korelovali podľa 4 rôznych časových rámcov a to 5, 10, 15 a 30 minút.

8.7.1 5 minútový časový rámec

Tabuľka 10: Výsledky pre 5 minútový časový rámec

Attack	**Technique**	**Tactic**	**Problem**
Port Scan	Remote Services	Initial Access	-
RDP Bruteforce	Remote Services	Execution	-
Denial of Service	-	Initial Access	-
Network Sniffing	-	-	Incident not identified
XSS + dir listing	-	-	Huge amount of data

V tomto prípade väčšina útokov nebola zmiešaná dohromady s inými útokmi. Jediný taký prípad bol pri záznamoch popisujúcich XSS útok a zneužitie miskonfigurácie *Dir listing*. Model správne identifikoval všeobecnejšiu techniku *Remote Services* pri útoku skenovania portov, avšak klasifikoval taktiku ako *Initial Access* namiesto *Reconnaissance*, čo je presnejší popis. V prípade útoku hrubou silou na RDP sa modelu úspešne podarilo identifikovať a klasifikovať incident. V ďalších scenároch model vykazoval problémy pri analýze. Najprv nedokázal správne identifikovať techniku ani taktiku pri útoku DoS a *Network Sniffing* a nakoniec pri spojenom útoku XSS a *dir listingu* nedokázal spracovať množstvo dát, ktoré mu bolo poskytnuté.

Ako môžeme vidieť, už menšia zmena v časovom rámci spôsobuje veľký šum v záznamoch a sťažuje celkovú identifikáciu.

8.7.2 10 minútový časový rámec

Pri zväčšenom časovom rámci na 10 minút, sa už viacero záznamov o útokoch pomiešalo. V rovnakom časovom rámci sa napríklad ocitol útok hrubou silou na RDP a taktiež aj útok DoS. To isté sa deje aj v prípade XSS a *Dir listingu*.

Tabuľka 11: Výsledky pre 10 minútový časový rámec

Attack	**Technique**	**Tactic**	**Problem**
Port Scan	Non-Standard Port, Remote Services	Command and Control, Lateral Movement	-
RDP Bruteforce + DOS	Remote Services	Execution	-
Network Sniffing	Network Sniffing	Collection / Reconnaissance	-
XSS + dir listing	-	-	Huge amount of data

Pri skenovaní portov, model správne identifikoval použitú techniku, avšak taktika *Command and Control/Lateral Movement* sa zdá byť čiastočne nepresná. Aj keď tieto taktiky možno použiť po úspešnom úvodnom prieskume, nie sú nevyhnutne jediným účelom skenovania. Pri kombinácií DoS útoku a útoku na protokol RDP, model správne identifikoval techniku *Remote Services*, ale priradil viac menej generickú taktiku *Execution*. Dôležité je ale spomenúť, že výsledná technika a taktika sa spája viac s útokom na protokol RDP ako na útok DoS. Z toho vychádza, že model prehliadol tento útok. Nakoniec, podobne ako pri 5-minútovom časovom rámci, model nedokázal spracovať objem dát pri útokoch XSS a *Dir listing*.

8.7.3 15 minútový časový rámec

Pri 15 minútovom časovom rámci sa veľa zmien oproti predošlému prípadu neudialo. Samozrejme, že záznamy obsahovali viacej šumu a bežnej aktivity ale útoky neboli viac spojené.

Tabuľka 12: Výsledky pre 15 minútový časový rámec

Attack	**Technique**	**Tactic**	**Problem**
Port Scan	Non-Standard Port	Discovery	-
RDP Bruteforce + DOS	Active Scanning	Discovery	-
Network Sniffing	Network Sniffing , Exfiltration Over Alternative Protocol	-	
XSS + dir listing	-	-	Huge amount of data

Pri skenovaní portov je výsledná technika *Non-Standard Port* spojená s neštandardnými portami, aj keď tento popis je veľmi všeobecný. Na druhú stranu, výsledná taktika *Discovery* sa zhoduje s účelom skenovania portov. Pri DoS útoku a útoku na protokol RDP, podobne ako pri 10-minútovom časovom rámci sa model snaží v tomto okne rozlíšiť dva typy útokov. Oboje zaraďuje do kategórie *Discovery* s technikou *Network Scanning*. To naznačuje na obmedzenia a problémy pri identifikácii niektorých metód útoku, keďže výsledná technika nie je presná. Čo sa ale týka útoku *Network Sniffing*, model správne identifikoval techniku aj keď na druhú stranu posúdil útok viac ako exfiltráciu dát. Čo nie je správne zaradenie.

V tomto scenári bolo oveľa viac viditeľné, že s väčším objemom dát a šumu, mal model problém rozlišovať čo sa vlastne v daný čas udialo. V prípade, že sa aj úspešne podarilo nájsť útok, jeho klasifikácia bola nejednoznačná.

8.7.4 30 minútový časový rámec

Nakoniec sa pozrieme na posledný scenár. V tomto prípade sme záznamy korelovali podľa 30 minútového časového rámca. Ako môžeme vidieť podľa tabuľky, hneď niekoľko útokov sa udialo v tom istom časovom okne.

Tabuľka 13: Výsledky pre 30 minútový časový rámec

Attack	**Technique**	**Tactic**	**Problems**
Port Scan + DOS + rdp Bruteforce	Reconnaissance	Discovery	Unable to detect all the attacks. No incident identified
Network Sniffing	-	-	No incident identified
XSS + dir listing	-	-	Huge amount of data

Pri analýze záznamov, ktoré obsahovali informácie o skenovaní portov, útoku na protokol RDP a DoS, model nedokázal správne rozpoznať spomínané útoky. Model na základe výsledkov mal problém presne určiť jednotlivé použité techniky. Navyše jedinú techniku, ktorú identifikoval bola *Reconnaissance*, čo je podľa Mitre *frameworku* taktika a nie technika. Okrem toho, síce sa snažil identifikovať útoky a klasifikovať ich, celkovo naznačoval, že sa nejedná v danom čase o žiadny incident. Rovnaký výsledok určil aj v prípade útoku *Network Sniffing*, kde taktiež po analýze došiel k záveru, že sa nejedná o žiaden útok. V poslednom prípade sa nič nezmenilo, vďaka veľkému objemu záznamov ich model nebol schopný spracovať.

8.7.5 Zhrnutie

Experiment v tomto prípade testovania priniesol zmiešané výsledky. Model preukázal určitý úspech pri identifikácii techník a taktík, najmä s kratšími časovými rámcami (1 a 5 minút). S čím mal ale problémy je rozlišovanie medzi rôznymi typmi útokov vo väčšom časovom rámci (10, 15, 30 minút). To mohlo byt zapríčinené zvýšeným šumom z nesúvisiacich udalostí, čo sťažuje určenie konkrétnych škodlivých aktivít. Navyše ďalším faktorom bolo spracovanie veľkého objemu dát, kde kombinácia záznamov z viacerých udalostí, najmä zložitých udalostí, ako je útok XSS alebo výpis adresárov, preťažila kapacitu modelu.

Tieto zistenia poukazujú na to, že zatiaľ čo LLM ako GPT-3.5-turbo-0125 naznačujú úspešnosť pri analýze textu, ich účinnosť v tomto odvetví môže byť obmedzená. Na druhú stranu, každým dňom pribúdajú nové a nové modely, ktoré sú dotrénované na určité úlohy a môžu byť pri takejto analýze omnoho kvalitnejšie.

Záver

Neustále napredovanie techník v kybernetických zločinoch si vyžaduje neustálu inováciu v oblasti kybernetickej bezpečnosti. Bezpečnostné tímy čelia ťažkej úlohe a to spracovávaniu obrovského množstva záznamov s cieľom identifikovať potenciálne incidenty, čo je proces, ktorý môže byť časovo náročný a náchylný na chyby. LLM sa ukázali ako sľubná technológia s potenciálom zmeniť spôsob, akým pristupujeme ku klasifikácii bezpečnostných incidentov.

V práci sme skúmali možnosť použitia LLM na analýzu Suricata a Sysmon záznamov s cieľom identifikovať a klasifikovať potenciálne incidenty. Simulovali sme rôzne kybernetické útoky vrátane skenovania portov, útoku hrubou silou na protokol RDP a iné. Záznamy z týchto simulácií boli potom spracované a vložené do LLM na analýzu. Úlohou LLM bolo identifikovať a klasifikovať potenciálne bezpečnostné incidenty v záznamoch pomocou *frameworku* MITRE ATT&CK, známeho štandardu na klasifikáciu kybernetických taktík a techník.

Experiment priniesol zmiešané výsledky, pričom zvýraznil silné stránky ale aj obmedzenia, ktorým LLM čelí. Pri analýze záznamov s kratšími časovými rámcami, preukázali modely určitý úspech pri identifikácii techník a taktík spojených s konkrétnymi útokmi. Avšak, výkonnosť LLM klesala s dlhšími časovými rámcami (10, 15 a 30 minút). To bolo spôsobené napríklad zvýšeným objemom dát, ktorý priniesol šum z nesúvisiacich udalostí. To sťažilo LLM rozlíšenie medzi špecifickými škodlivými aktivitami a neškodným sieťovým prenosom. Rovnako zvýšenie objemu dát viedlo aj k preťaženiu kapacity modelov, ktoré neboli schopné analyzovať dané množstvo. V 30-minútovom časovom rámci LLM dokonca niektoré útoky úplne vynechal, najmä kvôli obrovskému množstvu údajov.

Na základe týchto zistení je evidentné, že jazykové modely sú do určitej mieri schopné analyzovať bezpečnostné záznamy, najmä v prípade menších vzoriek. Jedným z možných návrhov na zlepšenie je dotrénovanie vybraného LLM. To ale prichádza so svojimi problémami ako je množstvo a kvalita dát na trénovanie, výpočtové zdroje a nakoniec čas. Takto dotrénovaný model by mal na druhú stranu vykazovať omnoho lepšie výsledky a tým pádom pomôcť predchádzať alebo zastaviť bezpečnostný incident ešte v jeho počiatku.

Zoznam použitej literatúry

1. FORTINET. What is an Intrusion Detection System (IDS)? *fortinet* [online]. [N.d.] [cit. 2024-03-18]. Dostupné z : https://www.fortinet.com/resources/cyberglossary/intrusion-detection-system.
2. LUTKEVICH, Ben. intrusion detection system (IDS). *TechTarget* [online]. 2021 [cit. 2024-03-19]. Dostupné z : https://www.techtarget.com/searchsecurity/definition/intrusion-detection-system.
3. LOWY, Kennith. SURICATA BASIC FUNCTIONALITY. *Medium* [online]. 2023 [cit. 2024-03-19]. Dostupné z : https://medium.com/@kennithlowy/suricata-basic-functionality-868135dac7cd.
4. SHAH, Syed a ISSAC, Biju. Performance Comparison of Intrusion Detection Systems and Application of Machine Learning to Snort System. *Future Generation Computer Systems*. 2018, roč. 80, s. 157–170. Dostupné z DOI: 10.1016/j.future.2017.10.016.
5. MONK, Scarred. Understanding Sysmon Events using SysmonSimulator. *RootDSE* [online]. 2022 [cit. 2024-03-20]. Dostupné z : https://rootdse.org/posts/understanding-sysmon-events/.
6. MARTIN, Lockheed. Gaining the Advantage: Applying Cyber Kill Chain® Methodology to Network Defense [online]. 2011 [cit. 2024-03-11]. Dostupné z : https://www.lockheedmartin.com/content/dam/lockheed-martin/rms/documents/cyber/Gaining_the_Advantage_Cyber_Kill_Chain.pdf.
7. CORPORATION, MITRE. MITRE ATTCK® Framework [online]. [N.d.] [cit. 2024-03-12]. Dostupné z : https://attack.mitre.org/.
8. SAP. *Generative AI: A Powerful New Technology* [online]. Online: SAP, [n.d.] [cit. 2024-03-12]. Dostupné z : https://www.sap.com/sk/products/artificial-intelligence/what-is-generative-ai.html.
9. FITZGERALD, Anna. Generative AI in Cybersecurity: How It's Being Used + 8 Examples. *Secureframe* [online]. 2023 [cit. 2024-03-14]. Dostupné z : https://secureframe.com/blog/generative-ai-cybersecurity.
10. STANHAM, Lucia. Generative AI (GenAI) in Cybersecurity. *Crowdstrike* [online]. 2023 [cit. 2024-03-14]. Dostupné z : https://www.crowdstrike.com/cybersecurity-101/secops/generative-ai/.

11. ELASTIC. What is a large language model (LLM)? *Elastic* [online]. 2024 [cit. 2024-04-02]. Dostupné z : https://www.elastic.co/what-is/large-language-models.

12. TARDIF, Antoine. Odhalenie sily veľkých jazykových modelov (LLM). *Unite AI* [online]. 2023 [cit. 2024-04-02]. Dostupné z : https://www.unite.ai/sk/ve%C4%BEk%C3%A9-jazykov%C3%A9-modely/.

13. FOY, Peter. Understanding Transformers & the Architecture of LLMs. *MLQ AI* [online]. 2024 [cit. 2024-04-02]. Dostupné z : https://www.mlq.ai/llm-transformer-architecture/.

14. AKASH. What is Python? Is it easy to learn? *Edrureka.* 2021. Dostupné tiež z: https://www.edureka.co/blog/what-is-python/.

15. BAHGAT, Ahmed. Flask vs Django: Let's Choose Your Next Python Framework. *Kinsta* [online]. 2023 [cit. 2024-04-09]. Dostupné z : https://kinsta.com/blog/flask-vs-django/.

16. DEVDOC. HTML basics. *DevDoc* [online]. 2017 [cit. 2024-04-09]. Dostupné z : https://devdoc.net/web/developer.mozilla.org/en-US/Learn/Getting_started_with_the_web/HTML_basics.html.

17. JATASRA, Sarvex. The Power of Context: Understanding the Importance of Context Window in Large Language Models. *LinkedIn* [online]. 2023 [cit. 2024-04-03]. Dostupné z : https://www.linkedin.com/pulse/power-context-understanding-importance-window-large-language-jatasra-vf8rf/.

18. AQUASEC. Indicators of Compromise (IoC): Examples, Lifecycle, and Security Impact. *Aquasec* [online]. 2022 [cit. 2024-04-03]. Dostupné z : https://www.aquasec.com/cloud-native-academy/vulnerability-management/indicators-of-compromise/.

19. DEVITO, Andrew. Suricata vs Snort: A Comprehensive Review. *StationX* [online]. 2023 [cit. 2024-04-05]. Dostupné z : https://www.stationx.net/suricata-vs-snort/.

20. TAMURA, Eito. Windows Sysinternals - Sysmon. *tier zero security* [online]. 2024 [cit. 2024-04-05]. Dostupné z : https://tierzerosecurity.co.nz/2024/02/27/microsoft-system-monitor.html.

21. PROJECT, MISP. *MISP evtx-toolkit* [https://github.com/MISP/evtx-toolkit/blob/main/README.md]. 2020.

Prílohy

A Štruktúra elektronického nosiča

/praca.pdf

· diplomová práca napísaná vo formáte pdf

/backend.py

· súbor obsahuje kód pre *back-end* webového rozhrania

/constants.py

· súbor obsahuje konštanty použité v našom nástroji

/correlation.py

· súbor obsahuje hlavnú funckionalitu

/gpt.py

· súbor obsahuje kód na komunikáciu s LLM

/events_filter.py

· súbor obsahuje kód na filtrovanie záznamov

/templates

· priečinok obsahuje zdrojové kódy pre webové rozhranie

B Používateľská príručka

Príručka pre spustenie a otestovanie nášho riešenia vo vlastnom prostredí.

Implementácia

Pre správne fungovanie systému je potrebné nainštalovať *Python* knižnice. Tie vieme doinštalovať pomocou nasledovných príkazov:

- **pip install openai**
- **pip install flask**
- **pip install markdown**
- **pip install requests**

Po doinštalovaní knižníc, vieme nástroj spustiť príkazom:

python correlation.py

Na konvertovanie Sysmon záznamov z EVTX formátu do JSON treba stiahnuť nástroj evtx-toolkit[26] a použiť nasledovný príkaz:

python evtx_dump.py sysmon.evtx –noepochconvert -o json > output.json

V prípade použitia komerčných modelov od OpenAI, je potrebné si zakúpiť API kľúč na stránke OpenAI [27]. Tento API kľúč je neskôr potrebne vložiť do súbora *constants.py* do premennej s názvom *API_KEY*. V prípade lokálneho LLM, je potrebne si stiahnuť už spomínaný llama server[28]. Za ďalšie, je potrebné si stiahnuť vybraný LLM model zo stránky Hugging face[29]. Model je potom možné spustiť nasledovne:

./server -m CESTA_K_MODELU

[26]https://github.com/MISP/evtx-toolkit/blob/main/README.md
[27]https://openai.com/api/
[28]https://github.com/ggerganov/llama.cpp
[29]https://huggingface.co/

Po nainštalovaní si *frameworku* Flask, je možné si spustiť webové rozhranie nasledovným príkazom:

python backend.py