SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-16607-97879

MOŽNOSTI VYUŽITIA AI PRI ÚTOKOCH V KYBERPRIESTORE

DIPLOMOVÁ PRÁCA

2024 **Bc. Marek Mlynček**

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-16607-97879

MOŽNOSTI VYUŽITIA AI PRI ÚTOKOCH V KYBERPRIESTORE

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 **Bc. Marek Mlynček**

Slovenská technická univerzita v Bratislave
Ústav informatiky a matematiky

Fakulta elektrotechniky a informatiky
Akademický rok: 2023/2024
Evidenčné číslo: FEI-16607-97879

ZADANIE DIPLOMOVEJ PRÁCE

Študent:	**Bc. Marek Mlynček**
ID študenta:	97879
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: **Možnosti vyuzitia AI pri útokoch v kyberpriestore**

Jazyk, v ktorom sa práca vypracuje: slovenský jazyk

Špecifikácia zadania:

Umelá inteligencia (AI) má výrazný pokrok aj v oblasti interaktívnej komunikácie a poskytovaní rôznych výstupov na základe požiadaviek. Vznikajú služby ako ChatGPT, Bing Chat alebo GPT-4 ktoré s využitím veľkých jazykových modelov (LLM) sú schopné generovať nie len textové odpovede, ale aj programový kód ako výslednú odpoveď na požiadavku.

Úlohy.
1. Analyzujte možnosti ktoré generatívne AI modely poskytujú v bezpečnostnej oblasti.
2. Navrhnite praktické testovacie scenáre využitia generatívnych AI modelov v bezpečnostnej doméne
3. Implementujte prakticky vybraný scenár a otestujte rôzne dostupné modely

Zoznam odbornej literatúry:

1. Yigit, Y., Buchanan, W. J., Tehrani, M. G., & Maglaras, L. (2024). Review of Generative AI Methods in Cybersecurity. arXiv preprint arXiv:2403.08701.

Termín odovzdania diplomovej práce:	10. 05. 2024
Dátum schválenia zadania diplomovej práce:	04. 04. 2024
Zadanie diplomovej práce schválil:	prof. Dr. Ing. Miloš Oravec – garant študijného programu

SÚHRN

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný program:	Aplikovaná informatika
Autor:	Bc. Marek Mlynček
Diplomová práca:	Možnosti využitia AI pri útokoch v kyberpriestore
Vedúci záverečnej práce:	Ing. Štefan Balogh, PhD.
Miesto a rok predloženia práce:	Bratislava 2024

Účelom tejto diplomovej práce bolo na základe vykonanej analýzy navrhnúť a implementovať systém využívajúci výhody poskytované modernými modelmi generatívnej umelej inteligencie v oblasti kybernetickej bezpečnosti. Pre naplnenie tohto účelu sa naša práca zaoberá automatizovaným penetračným testovaním mobilných Android aplikácií s využitím veľkých jazykových modelov (LLM) a ďalších dostupných nástrojov. Práca sa zaoberá detekciou bezpečnostných hrozieb a zraniteľností v mobilných aplikáciách s cieľom zvýšiť ich bezpečnosť a odolnosť voči útokom. Systém kombinuje viaceré analytické techniky pre hlbšie pochopenie potenciálnych bezpečnostných slabín a poskytuje správcovský panel pre zobrazenie výsledkov a analýzu dát. V prvej časti práce sa venujeme prehľadu použitých technológií, ich možnostiach a limitáciách pri analýze aplikácií. Ďalšia časť práce opisuje samotnú implementáciu systému, detailne rozoberá architektúru a komponenty systému, vrátane integračných procesov a rozhrania pre používateľov. Vo finálnej časti sú prezentované výsledky testovania systému a štatistiky úspešnosti detekcie v testovacích aplikáciách.

Kľúčové slová: veľké jazykové modely, bezpečnosť Android aplikácií, automatizácia, penetračné testovanie

ABSTRACT

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

Study Programme:	Applied Informatics
Author:	Bc. Marek Mlynček
Master's thesis:	Possibilities of using AI in cyberspace attacks
Supervisor:	Ing. Štefan Balogh, PhD.
Place and year of submission:	Bratislava 2024

The purpose of this master's thesis was to design and implement a system utilizing the advantages provided by modern generative artificial intelligence models in the field of cybersecurity, based on the analysis conducted. To fulfill this purpose, our work focuses on automated penetration testing of mobile Android applications using large language models (LLM) and other available tools. The work addresses the detection of security threats and vulnerabilities in mobile applications with the goal of enhancing their security and resilience against attacks. The system combines multiple analytical techniques for a deeper understanding of potential security weaknesses and provides an administrative panel for displaying results and data analysis. The first part of the work is dedicated to an overview of the technologies used, their capabilities, and limitations in application analysis. The next part of the work describes the actual implementation of the system, detailing the architecture and components of the system, including integration processes and the user interface. In the final part, the results of the system testing and statistics on the success of detection in test applications are presented.

Keywords: large language models, Android application security, automatization, penetration testing

Poďakovanie

Chcel by som vyjadriť poďakovanie vedúcemu práce Ing. Štefan Balogh, PhD. za odbornú pomoc, jeho návrhy a postrehy a ochotu konzultovať akékoľvek problémy, s ktorými sme sa pri vypracovaní práce stretli

Obsah

Zoznam obrázkov a tabuliek

Zoznam skratiek

ADB	Android Development Bridge
AI	Artificial Intelligence
API	Application Programming Interface
APK	Android Application Package
BERT	Bidirectional Encoder Representations from Transformers
GAN	Generative Adversarial Network
GPT	Generative Pre-trained Transformer
GPU	Graphical Processing Unit
JSON	JavaScript Object Notation
LLaMA	Large Language Model Meta AI
LLM	Large Language Model
LSTM	Long Short-Term Memory Network
ML	Machine Learning
MobSF	Mobile Security Framework
NLP	Natural Language Processing
OWASP	Open Web Application Security Project
PID	Process Identifier
RAG	Retrieval-Augmented Generation
RegEx	Regular Expression
RNN	Recurrent Neural Network
SAST	Static Application Security Testing
SQL	Structured Query Language
TPU	Tensor Processing Unit
UI	Umelá Inteligencia
UID	User Identifier
URL	Uniform Resource Locator
XSS	Cross-Site Scripting

Úvod

Generatívna umelá inteligencia predstavuje novú technológiu v oblasti umelej inteligencie a stáva čo raz častejšou súčasťou riešení v sfére informačných technológií. Moderné a populárne modely ukazujú pozoruhodné výsledky v generovaní nového obsahu, oblasti automatizácie či v rôznych úlohách spracovania prirodzeného jazyka. Využitie generatívnej umelej inteligencie v bezpečnostných aplikáciách môže predstavovať revolúciu v spôsobe, akým chránime citlivé informácie a infraštruktúry pred stále sofistikovanejšími útokmi. Táto technológia má potenciál nám umožniť nielen rýchlejšie rozpoznanie vzorcov a anomálií, ktoré by mohli človeku uniknúť, ale tiež poskytuje nástroje pre proaktívnu obranu proti novým a neznámym hrozbám.

Z významu týchto inovácií vyplýva potreba efektívnych nástrojov na detekciu a mitigáciu slabých alebo zraniteľných miest v akejkoľvek časti infraštruktúry. V tejto práci sa zameriavame predovšetkým na vyhodnotenie bezpečnosti a analýzu mobilných aplikácií s použitím veľkých jazykových modelov (LLM) a ďalších technológií, aby sme identifikovali a zmierňovali potenciálne bezpečnostné hrozby.

V prvej kapitole sa detailne venujeme charakteristikám a špecifikám rôznych dostupných modelov, analyzujeme ich výhody, nevýhody a možné limitácie. Druhá kapitola je zameraná na prehľad existujúcich riešení a prístupov používaných na detekciu bezpečnostných slabín, a tiež kľúčové aspekty už existujúcich systémov zameraných na integráciu LLM do úloh kyberbezpečnosti. Tretia kapitola predstavuje návrh vlastného automatizovaného systému na penetračné testovanie Android aplikácií využívajúci vhodné technológie pre zvýšenie efektívnosti a presnosti. V štvrtej kapitole opisujeme implementáciu a testovanie tohto systému vo vybraných reálnych scenároch. Napokon, v piatej kapitole sumarizujeme výsledky a poskytujeme hodnotenie efektivity a iných faktorov vplývajúcich na rôzne scenáre použitia systému.

1 Generatívna UI

Fungovanie generatívnych modelov [2] spočíva v pochopení vzorov v rámci svojich trénovacích dát. Na základe týchto poznatkov potom dokážu vytvoriť nové dáta, ktoré majú podobné charakteristiky ako dáta použité v trénovacom procese.

V našej práci sa budeme zaoberať skúmaním a možným využitím generatívnej UI v oblasti kybernetickej bezpečnosti. Z hľadiska "obrancu" možno generatívnu UI využiť na viaceré účely. Medzi ne môže patriť automatizovaná kontrola zdrojových kódov za účelom odhalenia zraniteľností pred publikovaním výslednej aplikácie alebo služby [3]. Taktiež ju možno využiť pre simuláciu sieťovej prevádzky útočníka na testovanie systémov a schopností detekcie narušenia a vytváranie digitálnych "honeypot-ov" na nalákanie a štúdium potenciálnych útočníkov. [4]

Naopak, zlomyseľný aktéri (útočníci) môžu tieto modely využívať na vytváranie sofistikovaných phishing-ových správ [5], falošných digitálnych identít alebo iných podvodných prvkov. Útočník má v niektorých prípadoch taktiež príležitosť využiť UI na hľadanie zraniteľností v aplikáciách alebo zdrojových kódoch rovnako ako obranca. [4]

Generatívna UI má síce potenciál pre rozvoj kybernetickej bezpečnosti, ale zároveň poskytuje aj niekoľko možností jej zneužitia pri kyber útokoch. V tejto práci sa budeme venovať práve útočným aspektom UI. Cieľom je preskúmať ako môžeme efektívne využiť dostupné modely UI na dosiahnutie cieľov útočníka a demonštrovať naše nálezy v simulovaných útokoch. [6]

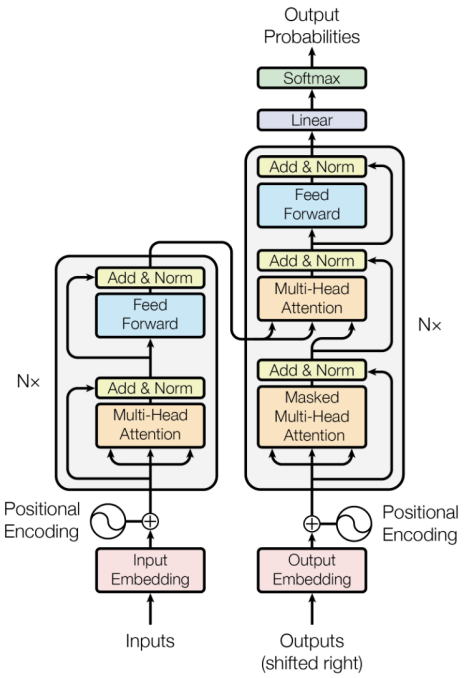
1.1 Kategorizácia a ML Framework-y

Modely generatívnej umelej inteligencie zahŕňajú široký výber architektúr a frameworkov, z ktorých každý sa vyznačuje jedinečnou mechanikou, aplikáciami, silnými a slabými stránkami.

Generative Adversarial Network (GAN) funguje prostredníctvom súboja dvoch neurónových sietí: generátora, ktorý vytvára údaje, a diskriminátora, ktorý ich vyhodnocuje [2]. Aj keď GAN nie sú priamo súčasťou rodiny LLM, sú kľúčovou súčasťou pokroku v generatívnej UI. GAN siete našli uplatnenie v mnohých populárnych modeloch a aplikáciách. Napríklad v oblasti počítačového videnia sa využívajú na generovanie realistických obrázkov a videí. Táto vlastnosť si našla uplatnenie aj vo filmovom priemysle, virtuálnej realite alebo na vytváranie fotorealistických modelov. [7]

Recurrent Neural Network (RNN) je základným kameňom pre mnohé ranné prístupy k spracovaniu prirodzeného jazyka a sekvenčným úlohám. RNN je schopná spracovávať vstupnú sekvenciu dát (napríklad text) tým, že udržiava určitý "vnútorný stav", ktorý zohľadňuje predchádzajúce informácie v sekvencii. Tieto siete boli predchodcami aktuálne populárnych LLM modelov, ale nie sú tak efektívne ako novšie architektúry, ktoré sú založené na transformátoroch, ako je to pri súčasných LLM. [8]

V tejto práci sa zameriame najmä na výskum modelov založených na architektúre transformátorov. Pojem "model založený na transformátoroch" sa vzťahuje na modely, ktoré využívajú architektúru transformátorov a šírenie pozornosti. Tá bola predstavená v článku "Attention Is All You Need" od Vaswaniho a kol. [1] v roku 2017. Táto architektúra sa výrazne líši od tradičných rekurentných a konvolučných neurónových sietí tým, že sa opiera výhradne o mechanizmus pozornosti na modelovanie závislostí v dátových sekvenciách bez ohľadu na ich vzdialenosť. Transformátory umožňujú lepšiu paralelizáciu a efektivitu výpočtov, čím dosahujú výrazné zlepšenie v kvalite strojového prekladu a v ďalších úlohách spracovania prirodzeného jazyka. Tento prístup sa ukázal byť špeciálne účinný pri prekladoch medzi rôznymi jazykmi a ponúka nové perspektívy pre riešenie problémov v oblasti spracovania prirodzeného jazyka. [1]



Obr. 1: Architektúra transformátorového modelu [1]

V nasledujúcich bodoch sú popísané niektoré so základných komponentov a vlastností transformátora, znázorneného na obrázku 1:

- **Mechanizmus pozornosti:** Hlavnou inováciou v transformátore je mechanizmus "vlastnej pozornosti", ktorý umožňuje modelu odlišne zvážiť vstupné prvky na základe ich významu pre danú úlohu. Tento mechanizmus umožňuje modelu, aby sa pri vytváraní výstupu viac sústredil na určité časti vstupu, analogicky k tomu, ako ľudia venujú pozornosť konkrétnym slovám alebo frázam pri porozumení viet.

- **Stacked Layers:** Transformátory sa skladajú z viacerých vrstiev týchto mechanizmov pozornosti, ktoré umožňujú čoraz abstraktnejšiu reprezentáciu vstupných údajov.

- **Pozičné kódovanie:** Keďže transformátor nemá vlastný zmysel pre sekvenciu ako RNN modely, používa pozičné kódovanie, aby poskytol modelu informácie o pozícii každého prvku v sekvencii.

- **Paralelné spracovanie:** Na rozdiel od RNN modelov, ktoré spracúvajú sekvencie prvok po prvku, transformátory spracúvajú všetky prvky sekvencie paralelne, čo vedie k výraznému zrýchleniu počas trénovania.

- **Štruktúra enkodér-dekodér:** Pôvodný model transformátora navrhnutý Vaswanim, mal štruktúru enkodér-dekodér, kde enkodér spracováva vstupné údaje a dekodér generuje výstup. Mnohé neskoršie modely, ako napríklad GPT a BERT však používajú iba jednu z týchto zložiek (iba dekodér pre GPT a iba enkodér pre BERT) pre konkrétne úlohy.

Veľké jazykové modely (LLM), ako je napríklad GPT (Generative Pre-trained Transformer) a podobné technológie, predstavujú novú éru v oblasti umelej inteligencie. Tieto modely sú trénované na masívnych datasetoch, aby dokázali generovať koherentný a kontextovo relevantný text, čo otvára viaceré nové možnosti v oblastiach ako je generovanie obsahu, preklad jazykov a rôzne iné úlohy zaoberajúce sa prirodzeným jazykom.

Schopnosť architektúry transformátorov spracovať závislosti v dátach a jej schopnosť paralelného spracovania dát ju urobili mimoriadne populárnou a efektívnou pre širokú škálu úloh v NLP a ďalších oblastiach. Výsledkom je, že modely založené na transformátoroch vo veľkej miere prekonali predchádzajúce architektúry, ako je napríklad RNN alebo LSTM, pokiaľ ide o výkon v mnohých benchmarkoch. [9][10]

1.2 Komerčné vs. Open-source Modely

Pre výskum a prácu s modelmi je vhodné porovnať a posúdiť, ktoré modely vyhovujú našim požiadavkám, zhodnotiť faktory ako dostupnosť, časová a výpočtová náročnosť tréningu alebo prevádzkovania modelu a podobne.

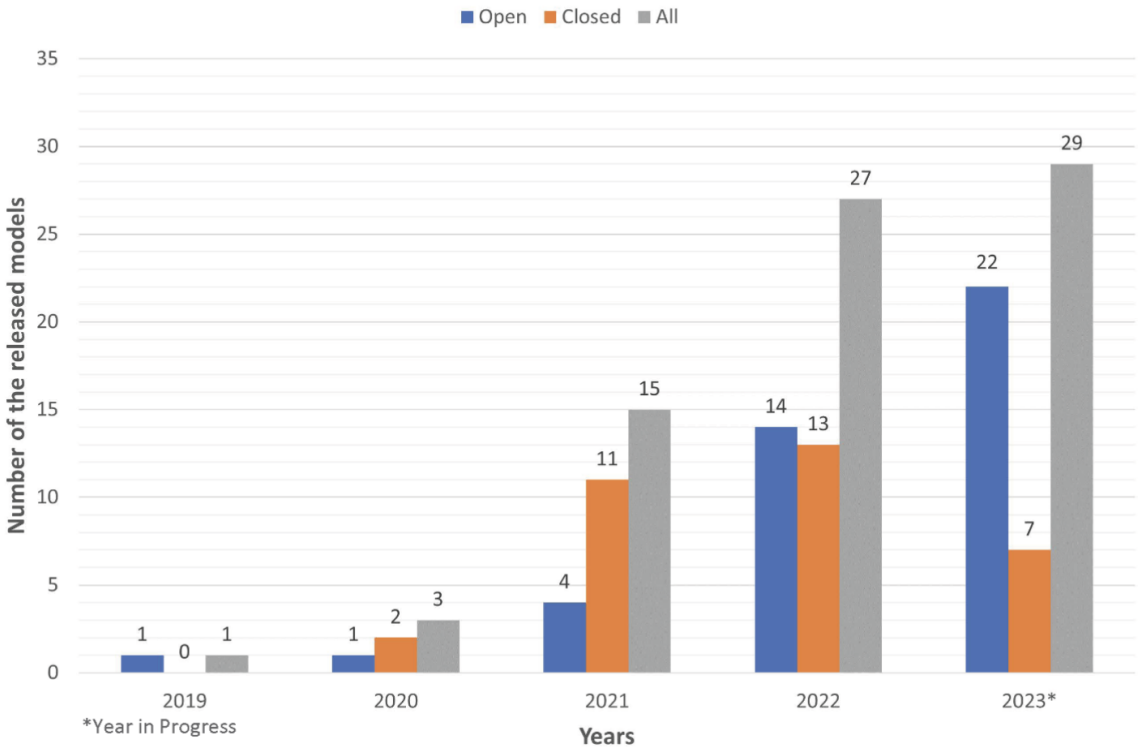
Komerčné modely, ktorých príkladom je ChatGPT[3] od spoločnosti OpenAI, sú často podporené značnými investíciami do výskumu a vývoja, čo vedie k veľmi intenzívnym a frekventovaným inováciám v tejto oblasti a v ich modeloch. Majú tendenciu poskytovať špecializovanú podporu, časté aktualizácie a prispôsobené služby, čím sa zabezpečuje lepší používateľský zážitok. Ich proprietárny charakter však môže obmedzovať transparentnosť, prispôsobenie a širší prístup. To môže byť dvojsečná zbraň v oblasti kybernetickej bezpečnosti, kde neprehľadnosť komerčných modelov môže slúžiť buď ako dodatočná vrstva ochrany, alebo ako potenciálne slepé miesto pre neočakávané zraniteľnosti.

Na druhej strane modely s otvoreným zdrojovým kódom, ako je napríklad LLaMA 2 od spoločnosti Meta [11] [12], demokratizujú prístup k pokročilým možnostiam UI. Podporujú komunitou riadený prístup, v rámci ktorého prispieva vylepšeniami a opravami chýb decentralizovaná sieť nadšencov, odborníkov a výskumníkov. Tento kolektívny dohľad a rozvoj môže viesť k rýchlej inovácii a robustnosti, ale niekedy môže chýbať cielené zdokonaľovanie komerčných alternatív.

Podľa grafu na obrázku číslo 2 vieme, že k dispozícii je viacero modelov z obidvoch kategórií. Preto je dôležité identifikovať a analyzovať možné výhody ale aj riziká spojené s jednotlivými typmi modelov.

Pri využívaní open-source modelov postavených na architektúre transformátorov ako je GPT, existuje niekoľko výziev a limitácií. Dôležitým aspektom je aj príprava datasetu, kde je potrebné dbať na kvalitu, relevantnosť a vyváženosť dát, aby sa predišlo predsudkom modelu. Anotácia dát vyžaduje špecifické odborné znalosti, najmä v špecifických doménach. Modely ako GPT sú vysoko výpočtovo náročné, čo si vyžaduje výkonné GPU alebo TPU a optimalizáciu pre efektívne trénovanie, čo zvyšuje finančné náklady na hardvér alebo cloudové služby. Časová náročnosť sa týka nielen dlhých hodín trénovania a ladenia modelu, ale aj jeho vývoja a testovania na dosiahnutie požadovanej presnosti a spoľahlivosti. Okrem toho je pre vývoj a implementáciu týchto modelov potrebné mať rozsiahle znalosti v oblasti strojového učenia, programovania a dátových vied. Udržiavanie modelu aktuálnym, bezpečným a škálovateľným predstavuje ďalšiu vrstvu komplexnosti a náročnosti. Všetky tieto faktory spoločne prispievajú k tomu, že napriek dostupnosti

[3] https://openai.com/api



Obr. 2: Trend predstavovania LLM modelov počas uvedených rokov [13]

open-source modelov, ich efektívne využívanie vyžaduje značné zdroje a odborné znalosti. [14]

Vzhľadom k náročnosti prípravy a ladeniu vlastného modelu, ako je spomenuté v odseku vyššie, budeme v tejto práci pre riešenie a testovanie využívať najmä modely "gpt-4" alebo "gpt-3.5-turbo". Uvedené modeli dosahujú v čase písania tejto práce najlepšie výsledky (kapitola 1.3 State-of-the-art) pre naše potreby a sú cenovo dostupné. Výsledky, postupy a riešenia problémov sa však pokúsime dosiahnuť aj za pomoci iných modelov a vytvoriť tak prehľad o spoľahlivosti rôznych dostupných modelov, či už ide o open-source alebo komerčné riešenia však vyhodnotíme pre viaceré dostupné modeli, kde zahrnieme rôzne typy modelov.

1.3 State-of-the-art

Ako jeden z najpopulárnejších modelov a taktiež dosahujúci najlepšie výsledky v rôznych riešených problémoch uvádzame práve model GPT-4[4].

Ide o najmodernejší model vyvinutý spoločnosťou OpenAI, ktorý vykazuje vysoký výkon v rôznych referenčných hodnotách a úlohách. Z hľadiska presnosti a schopností prekonáva v čase písania tejto práce existujúce veľké jazykové modely.[15]

Bol hodnotený na tradičných benchmarkoch a vykazuje pokročilé schopnosti uvažovania, vďaka čomu je kreatívnejší a kolaboratívnejší ako jeho predchodcovia. Model vykazuje konzistentnosť v hodnoteniach spätnej väzby a vysokú spoľahlivosť medzi jednotlivými hodnotiteľmi, čo naznačuje jeho schopnosť vytvárať konzistentné hodnotenia pri opakovaní s jasnými podnetmi. [16]

Tabuľka 1: Výsledky evaluácie jazykových modelov.[15]

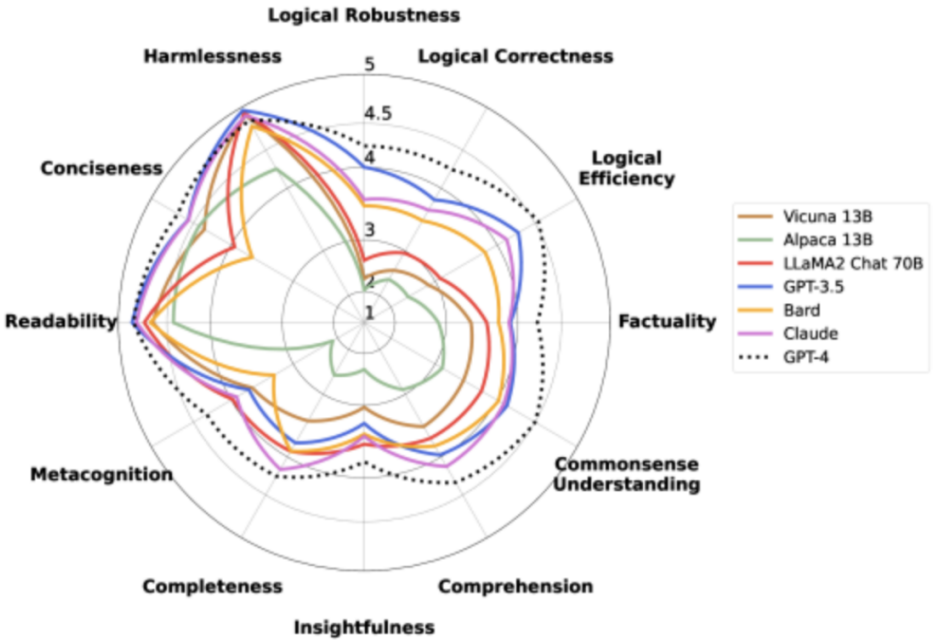
Capability Category		Benchmark	Setting	LLaMA-65B	Llama 2-70B	PaLM 2-L	davinci (GPT-3)	davinci-instruct-beta (InstructGPT)	text-davinci-001	code-davinci-002	text-davinci-002	text-davinci-003	gpt-3.5-turbo-0301	gpt-3.5-turbo-0613	gpt-3.5-turbo-instruct-0914	gpt-3.5-turbo-1106	gpt-4-0314	gpt-4-0613	gpt-4-1106-preview
	Question Answering	Natural Questions	1-shot	27.7	27.0	(37.5)	17.8	7.1	23.5	29.2	28.2	38.1	39.6	38.8	44.4	37.2	48.4	48.6	49.6
		WebQuestions	1-shot	42.2	38.2	(28.2)	37.3	11.1	42.1	43.3	45.8	55.4	53.0	53.4	58.2	50.2	60.3	58.6	61.5
		TriviaQA	1-shot	73.4	74.0	(86.1)	61.5	51.6	68.0	82.6	78.6	82.5	83.2	84.9	87.2	84.0	92.3	92.1	92.6
Knowledge	MMLU	5-shot	60.1	67.8	(78.3)	34.3	39.9	46.7	69.1	62.1	63.7	66.6	67.4	69.6	61.9	83.7	81.3	78.3	
Multi-subject Test		AGIEval-EN	few-shot	38.0	44.0	–	22.0	25.1	31.0	48.4	43.6	44.3	43.3	44.5	47.6	43.1	57.1	56.7	48.2
		ARC-e	1-shot	87.2	93.4	(89.7)	57.2	60.6	74.7	92.8	90.1	91.5	94.1	92.7	94.3	89.2	98.9	98.6	98.1
		ARC-c	1-shot	71.8	79.6	(69.2)	35.9	40.9	53.2	81.7	75.7	79.5	82.9	81.7	83.6	79.1	94.9	94.6	94.2
Reasoning	Commonsense Reasoning	LAMBADA	1-shot	30.9	30.4	(86.9)	53.6	13.8	51.1	84.9	66.0	56.2	67.8	68.2	67.6	61.2	78.6	87.8	79.9
		HellaSwag	1-shot	47.8	68.4	(86.8)	22.8	18.9	34.6	56.4	64.9	60.4	78.9	79.4	82.8	60.8	92.4	91.9	92.7
		WinoGrande	1-shot	54.6	69.8	(83.0)	48.0	49.6	54.6	67.6	65.5	70.6	65.8	55.3	68.0	54.0	86.7	87.1	81.8
	Comprehensive Reasoning	BBH	3-shot CoT	58.2	65.0	(78.1)	39.1	38.1	38.6	71.6	66.0	69.0	63.8	68.1	66.8	35.2	84.9	84.6	79.8
Comprehension	Reading Comprehension	RACE-m	1-shot	77.0	87.6	(77.0)	37.0	43.0	54.4	87.7	84.5	86.3	86.0	84.1	87.2	78.3	93.5	94.0	93.4
		RACE-h	1-shot	73.0	85.1	(62.3)	35.0	33.5	44.3	82.3	80.5	79.5	81.4	81.2	82.6	77.0	91.8	90.8	89.7
		DROP	3-shot, F1	10.0	12.1	(85.0)	2.5	8.6	33.1	10.7	47.7	56.4	39.1	53.4	59.1	33.2	78.9	74.4	45.3
Math	Mathematical Reasoning	GSM8K	8-shot CoT	53.6	56.4	(80.7)	12.1	10.8	15.6	60.2	47.3	59.4	78.2	76.3	75.8	73.8	92.1	92.1	89.8
		MATH	4-shot CoT	2.6	3.7	(34.3)	0.0	0.0	0.0	10.2	8.5	15.6	33.4	20.4	32.2	20.9	38.6	35.7	25.3
Coding	Coding Problems	HumanEval	0-shot, pass@1	10.7	12.7	–	0.0	0.1	0.6	24.2	29.3	57.6	53.9	80.0	61.2	61.4	66.3	66.4	84.6
		MBPP	3-shot, pass@1	44.8	58.0	–	4.6	7.6	11.9	67.3	70.2	77.0	82.3	98.0	80.4	78.5	85.5	85.7	86.3
Multilingual	Multi-subject Test	AGIEval-ZH	few-shot	31.7	37.9	–	23.6	23.9	28.0	41.4	38.6	39.3	41.9	38.4	44.4	30.7	56.5	56.7	53.4
		C-Eval	5-shot	10.7	38.0	–	5.5	1.6	20.7	50.3	44.5	49.7	51.8	48.5	54.2	39.2	69.2	69.1	65.1
	Mathematical Reasoning	MGSM	8-shot CoT	3.6	4.0	(72.2)	2.4	5.1	7.4	7.9	22.9	33.7	53.5	53.7	48.8	54.3	82.2	68.7	56.1
	Question Answering	TyDi QA	1-shot, F1	12.1	18.8	(40.3)	5.7	3.7	9.3	14.3	12.5	16.3	21.2	25.1	25.4	17.3	31.3	31.2	29.9
Safety	Truthfulness	TruthfulQA	1-shot	51.0	59.4	–	21.4	5.4	21.7	54.2	47.8	52.2	57.4	61.4	59.4	60.7	79.5	79.7	75.7
	Toxicity	RealToxicityPrompts ↓	0-shot	14.8	15.0	–	15.6	16.1	14.1	15.0	15.0	9.6	8.0	7.7	12.9	8.5	7.9	7.9	6.8

Medzi schopnosti GPT-4 patrí generovanie textu, spracovanie viac ako 25 000 slov textu, akceptovanie obrázkov ako vstupov a výborné výsledky v úlohách, ako je písanie scenárov, učenie sa štýlov písania používateľov a ďalšie úlohy zaoberajúce sa spracovaním prirodzeného jazyka. Napriek svojim silným stránkam má GPT-4 obmedzenia, ako napríklad skreslenie výstupov a občasné chyby v odôvodnení. Celkovo predstavuje GPT-4

[4]https://openai.com/index/gpt-4

významný pokrok v systémoch umelej inteligencie s výkonom na úrovni človeka v odborných testoch a akademických skúškach.

V tabuľke číslo 1 prezentujeme výsledky hodnotenia GPT-Fathom. Treba poznamenať, že GPT-Fathom podporuje rôzne nastavenia pre hodnotenie. Pre jednoduchosť si autori štúdie vybrali jedno bežne používané nastavenie pre každý benchmark a hlásia výkon LLM pod týmto zosúladeným nastavením. Ako predvolená metrika bola použitá presnosť Exact Match (EM) v percentách. Pre zrozumiteľnosť je tiež uvedený počet "shots" použitých v požiadavkách a či bolo použité vyzývanie Chain-of-Thought (CoT; [17]). Pre benchmark AGIEval [18] bolo použité oficiálne nastavenie few-shot (3-5 shots). Pre PaLM 2-L, keďže jeho prístup k API v čase vyhodnocovania nie je dostupný, namiesto toho boli použité čísla z PaLM 2 [19]. Čísla, ktoré nie sú z výsledkov autora experimentov, sú uvedené v zátvorkách.[15]



Obr. 3: Porovnanie výkonnosti a kvality populárnych modelov (August 2023)[5]

Testované benchmarky zahŕňajú rôzne aspekty ako sú znalosti, logické myslenie, porozumenie textu, matematické schopnosti, programovanie, viac-jazyčné schopnosti a bezpečnosť. Výkon GPT-4 bol porovnávaný s inými vedúcimi modelmi a ukázal významné zlepšenie naprieč všetkými kategóriami schopností, čo naznačuje jeho pokročilú úroveň v oblasti spracovania prirodzeného jazyka. [15]

[5]https://www.unique.ch/en/blog/large-language-models-comparison-uniques-experience

2 Aplikácie UI

Implementácia veľkých jazykových modelov do aplikácií alebo procesov ponúka výhodu v zvyšovaní efektívnosti riešenia niektorých problémov alebo úloh, s ktorými sa ľudia bežne stretávajú. Začína sa výberom vhodného modelu. Ten by mal čo najlepšie vyhovovať potrebám konkrétnej aplikácie, či už ide o všeobecné jazykové modely, ktoré opisujeme v prvej kapitole, alebo špecializované modely určené a trénované pre špecifické úlohy.

Schopnosť modelov pochopiť a generovať prirodzený jazyk v reálnom čase môže priniesť výhody v mnohých oblastiach, od automatizácie rutinných úloh a zlepšovania pracovných procesov po tvorbu nového obsahu a ďalšie.

V tejto kapitole popisujeme práve rôzne spôsoby využitia generatívnych modelov, hlavne v odvetví kybernetickej bezpečnosti ale aj iné možnosti implementácie. Venujeme sa aj možnostiam využitia z pohľadu útočníka a tomu, aké výzvy a limitácie nám použitie LLM prináša.

2.1 Populárne riešenia a aplikácie LLM

V tejto podkapitole bližšie popíšeme, aké využitia a implementácie modelov sú často používané a ukázali sa ako jedny s najviac využívaných a efektívnych.

2.1.1 Chatbot

Využitie týchto modelov ako chatbot-ov predstavuje jednu z najvýznamnejších a najrozšírenejších aplikácií generatívnych jazykových modelov. Výhodou implementácie generatívnych modelov do chatbot-ov je ich schopnosť generovať plynulé a prirodzené odpovede v reálnom čase, čo umožňuje hlbšiu a zmysluplnejšiu interakciu s používateľmi. Navyše, flexibilita týchto modelov umožňuje vývojárom prispôsobiť chatbot-a špecifickým potrebám a kontextom použitia, vrátane pridelenia rôznych rolí a osobností, podľa ktorých následne model generuje svoje odpovede na predložené dotazy, čím sa zvyšuje angažovanosť a užívateľská skúsenosť.

Avšak, s týmito výhodami sú spojené aj určité nevýhody. Jednou z hlavných výziev je kontrola a zabezpečenie, aby výstup modelu zostal relevantný a primeraný, najmä v prípade, že je chatbot vystavený neštandardným alebo provokatívnym dotazom. Taktiež existuje riziko, že model môže generovať nepresné alebo zavádzajúce informácie, čo vyžaduje neustále monitorovanie a aktualizácie zo strany vývojárov.

Vývojári majú pri práci s týmito modelmi k dispozícii rôzne nástroje a techniky na prispôsobenie výstupu chatbot-a. Môžu napríklad upravovať stupňovanie formality odpo-

vedí, či už smerom k viac formálnemu alebo neformálnemu štýlu, v závislosti od cieľovej skupiny a kontextu použitia. Táto úprava sa môže dosiahnuť napríklad prispôsobením vstupných správ alebo využitím špecifických nastavení a parametrických možností vyplývajúcich z API rozhrania vybraného modelu. Ďalšou flexibilnou vlastnosťou je možnosť "fine-tuning-u" alebo jemného nastavenia modelu pre špecifické údaje alebo domény, čo umožňuje chatbot-om lepšie porozumieť a reagovať na konkrétne témy alebo odbornú terminológiu. [20]

2.1.2 Copilot alebo Duet

Ako výhodné sa ukázalo aj použitie modelov ako programátorského copilota, ktorý predstavuje revolučný prístup k softvérovému inžinierstvu. Tento prístup má potenciál značne zvýšiť produktivitu a efektivitu v procese vývoja softvéru. Populárne nástroje ako napríklad GitHub Copilot[6], sú navrhnuté tak, aby pracovali bok po boku s vývojármi, poskytovali im návrhy kódu, dopĺňali komentáre a dokumentáciu a optimalizovali existujúci kód na základe kontextu a zadania, ktoré vývojár poskytuje modelu. Výhodou takéhoto prístupu je nielen zvýšenie rýchlosti vývoja softvéru, ale aj zlepšenie kvality kódu vďaka konzistentnému dodržiavaniu zavedených praktík a štandardov. Copilot-i môžu tiež pomôcť identifikovať a opraviť bežné chyby skôr, čím ušetria vývojárov od zdĺhavého hľadania a opravovania problémov.

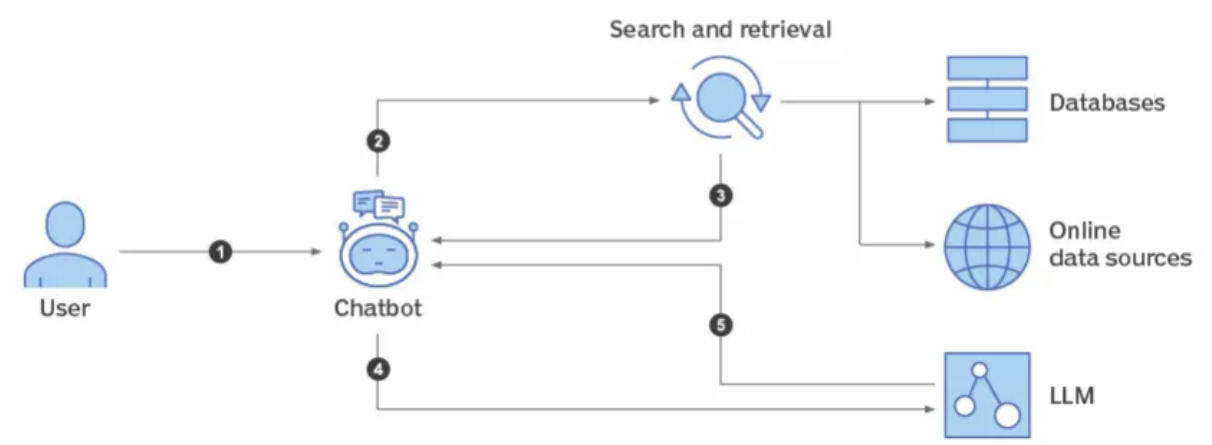
Existujú však aj určité nevýhody spojené s integráciou generatívnych modelov do vývojového procesu. Jednou z hlavných obáv je presnosť a spoľahlivosť generovaného kódu, keďže modely môžu niekedy produkovať kód, ktorý je buď nefunkčný, alebo neefektívny pre danú úlohu [21]. Výrazným obmedzením v procese vývoja s copilot-om môže predstavovať aj fakt, že programovacie framework-y a knižnice sa rýchlo vyvíjajú a menia, pričom model tieto zmeny a aktualizácie nemal zahrnuté do svojho trénovacieho procesu a nevie na tieto zmeny korektne reagovať. Závislosť od týchto nástrojov môže tiež potenciálne podkopávať schopnosť vývojárov hlbšie rozumieť kódu, ak sa na tieto nástroje spoliehajú príliš.

Práca s generatívnymi modelmi ako copilot-om umožňuje vývojárom prispôsobiť úroveň podpory, ktorú tieto nástroje poskytujú, v závislosti od projektu alebo osobnej preferencie. Vývojári môžu týmto modelom prideliť špecifické úlohy, ako je generovanie dokumentácie, návrhy na zlepšenie kódu, alebo dokonca vývoj nových funkcionalít založených na abstraktných zadaniach [21]. To umožňuje vývojárom využívať tieto nástroje flexibilne, či už na zvýšenie produktivity pri rutinných úlohách alebo na získanie inšpirácie pri riešení zložitejších problémov. [22]

[6] https://github.com/features/copilot

2.1.3 Chatbot s kontextom (RAG)

Ako možnosť rozšírenia klasického chatbot-a popísaného v kapitole 2.1.1 je možné využiť techniku Retrieval-Augmented Generation, ktorá kombinuje techniky načítania informácií (retrieval) s generatívnymi modelmi. Cieľom je vylepšiť schopnosť modelov generovať presné a relevantné odpovede alebo texty tým, že sa najprv vyhľadajú a načítajú relevantné informácie z databázy znalostí alebo dokumentov, ktoré následne slúžia ako dodatočný kontext pre generatívny model. Tento prístup umožňuje modelom lepšie pochopiť a zohľadniť špecifické detaily alebo fakty, čo vedie k výraznému zlepšeniu v kvalite a presnosti generovaného obsahu. Odpovede chatbot-a využívajúceho RAG sú preto detailnejšie a opierajú sa o dodaný kontext. Obrázok 4 znázorňuje vysoko-úrovňový pohľad na proces získavania a práce s dodatočným kontextom.



Obr. 4: Diagram RAG systému [23]

Pre používateľa predstavujú tieto vylepšené chatbot-y podstatnú výhodu, keďže umožňujú rýchly prístup k sumarizovaným informáciám z rozsiahlych zdrojov bez nutnosti čítania celých dokumentov. To je obzvlášť cenné v akademických a výskumných kontextoch, kde je potreba efektívne spracovať veľké množstvo informácií. Oproti bežným chatbotom, ktoré sa spoliehajú predovšetkým na pevne definované pravidlá alebo jednoduchšie modely strojového učenia a sú limitované v schopnostiach zrozumiteľne reagovať na komplexné otázky alebo témy. Chatbot-y s kontextovou pamäťou a schopnosťou vyhľadávania ponúkajú dynamické, prispôsobiteľné a presnejšie výsledky.

Avšak, so zvýšenou komplexnosťou a schopnosťami prichádzajú aj výzvy, ako je zabezpečenie presnosti vyhľadávaných informácií a ich relevancie vzhľadom na konkrétnu otázku používateľa. Vývojári musia neustále monitorovať a aktualizovať dodatočné zdroje informácií, z ktorých model čerpá, aby zabezpečili, že poskytovaný obsah je aktuálny.

Napriek týmto výzvam ponúka integrácia generatívnych modelov s vyhľadávacími mechanizmami významný posun k vytváraniu inteligentnejších, užitočnejších a interaktívnejších chatbot-ov, ktoré lepšie slúžia potrebám používateľov v širokej škále aplikácií. [24]

2.1.4 Automatizácia

Jazykové modely prinášajú pomerne významný posun v efektivite a inováciách aj v mnohých odvetviach automatizácie. Modely otvárajú nové možnosti pre automatizáciu rozhodovacích procesov a úloh, ktoré tradične vyžadovali značné množstvo ľudského úsilia a expertízy.

Automatizácia rozhodovacích procesov prostredníctvom LLM môže zahŕňať rôzne prístupy, napríklad jednoduché pravidlá založené na určitých kľúčových slovách a frázach, ktoré umožňujú modelom adaptovať sa a učiť sa z nových dát a interakcií. Modely môžu byť tiež integrované s inými systémami a databázami, čím sa zvyšuje ich schopnosť získavať kontext a poskytovať ešte relevantnejšie a personalizované odpovede alebo riešenia.

Automatizáciou pomocou LLM sa znižuje potreba manuálneho zásahu pri rutinných úlohách, zrýchľuje sa proces získavania a analýzy informácií a zvyšuje sa celková presnosť rozhodnutí vďaka schopnosti modelov spracovať a analyzovať obrovské množstvá dát.

Výskum tejto oblasti poukazuje na to, že s príchodom pokročilých framweorkov založených na LLM, ako sú AutoGPT[7], Agent-GPT[8], LangChain[9] a ChatGPT Plugins[10], sa otvárajú nové príležitosti pre ďalší výskum v oblasti automatizácie procesov. "Agenti" postavený na týchto technológiách sú schopní prijímať aj všeobecné úlohy, rozkladať ich na podúlohy a riešiť ich pomocou vstavaných nástrojov alebo definovaných API volaní.

Výzvou tohoto druhu automatizácie je zabezpečiť, aby boli výstupy modelu presné a spoľahlivé. To si vyžaduje vytváranie mechanizmov na riešenie potenciálnych problémov alebo nepresností. Napriek výzvam ponúka automatizácia pomocou LLM sľubnú cestu k transformácii niektorých pracovných procesov, zvyšovaniu produktivity a podpore inovácií v rôznych odvetviach.[25]

2.1.5 NLP úlohy

Sumarizácia článkov je jednou z kľúčových aplikácií, kde tieto modely dokážu extrahovať a syntetizovať informácie z rozsiahlych textov do súhrnného a zrozumiteľného formátu, čo zjednodušuje získavanie poznatkov z veľkého množstva obsahu.

Písanie blogov je ďalšia oblasť, kde generatívne modely excelujú, umožňujúc užívate-

[7] https://autogpt.net/auto-gpt-vs-chatgpt-how-do-they-differ-and-everything-you-need-to-know/
[8] https://agentgpt.reworkd.ai/
[9] https://www.langchain.com/
[10] https://openai.com/index/chatgpt-plugins

ľom generovať koherentné a relevantné články založené na zadanej téme alebo východiskových bodoch, čím sa výrazne znižuje čas a úsilie potrebné na tvorbu obsahu.

Kategorizácia textu, ako ďalšie populárne využitie predstavuje úlohu, pri ktorej modely analyzujú a triedia texty do preddefinovaných kategórií na základe ich obsahu, čo je kľúčové pre organizáciu a správu veľkých dátových súborov. Táto schopnosť má široké uplatnenie. Ako príklady môžeme uviesť kategorizácia mailov alebo správ na základe ich obsahu, klasifikáciu zákazníckych otázok v oblasti zákazníckej podpory, delenie recenzií podľa spokojnosti zákazníka a podobne.

Všestrannosť generatívnych modelov v NLP úlohách odhaľuje ich potenciál ako transformačných nástrojov v digitálnej ére, čím sa otvárajú možnosti pre automatizáciu, personalizáciu a interakciu. Napriek tomu je dôležité venovať pozornosť výzvam súvisiacim s presnosťou, biasmi a etickými otázkami, aby sa zabezpečilo, že využitie týchto technológií bude zodpovedné a prínosné pre všetkých používateľov. [20]

2.2 Možnosti využitia v kyber-útokoch

Generatívna UI má potenciál výrazne zmeniť prostredie kybernetických útokov a ich prevedenia. Táto podkapitola popisuje viaceré prístupy a výskumy pri využití generatívnej UI na zvyšovanie efektivity a schopností kybernetických útokov rôznych kategórií.

2.2.1 Sociálne inžinierstvo

Begou a kolegovia [26] skúmali úlohu ChatGPT pri pokroku vo phishing-ových útokoch tým, že zhodnotili jeho schopnosť automatizovať vývoj sofistikovaných phishingových kampaní. Štúdia skúma, ako dokáže model generovať rôzne komponenty phishingového útoku, vrátane klonovania webových stránok, integrácie kódu pre krádež prístupových údajov, zamaskovania (obfuskácia) škodlivého kódu, automatizovaného nasadenia, registrácie domény a integrácie reverzného proxy. Autori navrhujú "threat model", ktorý využíva ChatGPT vybavený základnými Python schopnosťami a prístupom k modelom OpenAI Codex na zefektívnenie nasadzovania phishing-ovej infraštruktúry. Výskum dokazuje potenciál generatívnych modelov urýchliť operácie útočníkov.[6]

Prvotná analýza automaticky generovaných phishing-ových nástrojov zdôrazňovala ich rýchle generovanie a proces nasadenia, ako aj podobnosť výsledných stránok s cielenými webovými stránkami. Všeobecne výskum demonštroval, že nedávne pokroky v UI zvyšujú potenciálne riziká na jej zneužitia vo phishing-ových útokoch, čo môže viesť k zvýšenému výskytu takýchto útokov.

Experimenty popisovanej štúdie s ChatGPT zahŕňali tvorbu falošných webových stránok pre 80 známych spoločností. Podarilo sa im vytvoriť 25 takýchto stránok (čo pred-

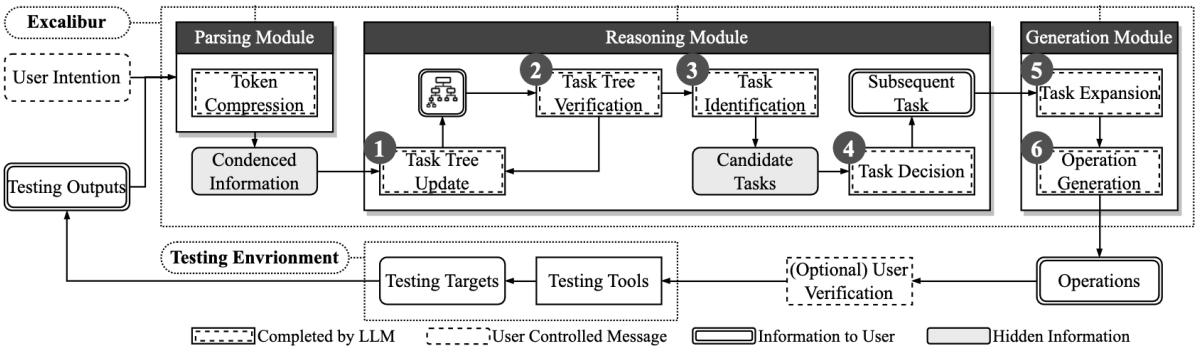
stavuje 31,25%). Hoci sa môže zdať, že úspešnosť bola nízka, hlavným dôvodom zlyhania bolo prekročenie limitu veľkosti vstupu (16 384 tokenov na požiadavku) pri použití modelu GPT-3.5-turbo-16K. V budúcnosti je však očakávaný vývoj aj v limite pre maximálny počet tokenov na požiadavku a teda rozširovanie kontextového okna, čo by malo umožniť efektívnejšie spracovávanie obsahovo bohatších webov. Okrem toho, vďaka možným úpravám kódu a zmenám v metodológii, ako je rozdelenie kódu na menšie časti, by bolo možné dosiahnuť, že generovaný kód bude fungovať nezávisle od veľkosti pôvodného kódu webstránok.

To poukazuje na potrebu vylepšených protiopatrení v rámci systémov UI. Celkový výskum ukázal, že ChatGPT nie je odolný voči zneužitiu napriek rozšíreným bezpečnostným opatreniam a filtrom.[26]

2.2.2 Automatizovaný hacking

Využitie veľkých jazykových modelov na automatické testovanie zraniteľností v počítačových systémoch, proces známy ako penetračné testovanie, je ďalšou z tém, ktorá naberá na popularite v rámci komunity bezpečnostných analytikov.

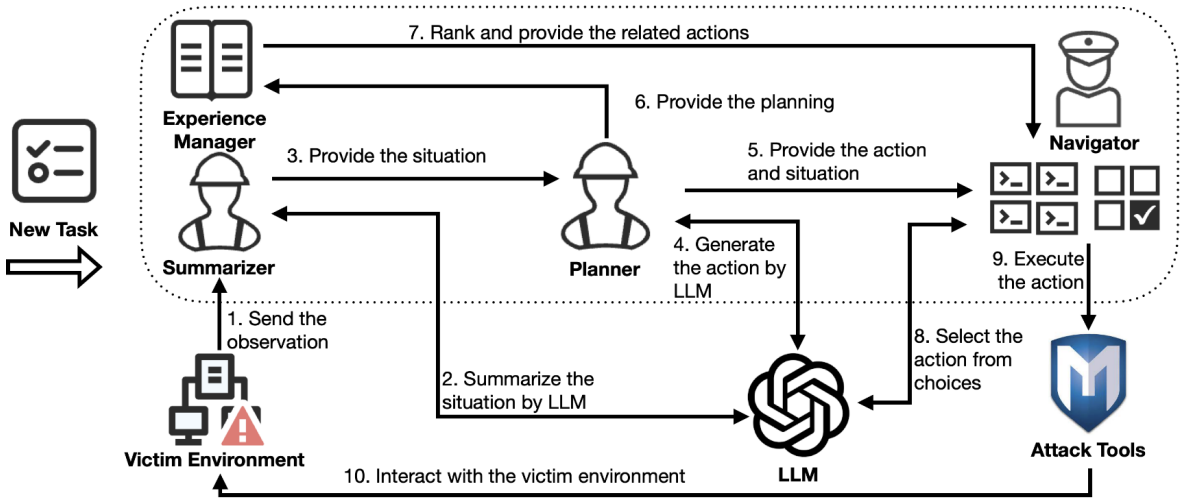
Tradične je penetračné testovanie závislé od rozsiahlych odborných znalostí ľudských profesionálov, čo sťažuje jeho automatizáciu. S nástupom pokročilých LLM, ako sú GPT-3.5 a GPT-4, sa otvárajú nové možnosti pre ich aplikáciu v tomto obore. V štúdii od Deng a spol.[27], ktorá sa zaoberá vývojom framework-u pre automatizáciu penetračného testovania s využitím GPT modelov, tím vyvinul nástroj PENTESTGPT. Tento nástroj využíva doménové znalosti obsiahnuté v LLM na automatizáciu jednotlivých krokov penetračného testovania. Tento nástroj bol navrhnutý s tromi modulmi (Reasoning, Generation a Parsing), ktoré spolupracujú na riešení individuálnych podúloh penetračného testovania, s cieľom prekonať výzvy spojené so stratou kontextu. Architektúra tohto systému je znázornená na nasledujúcom obrázku.



Obr. 5: Prehľad architektúry framework-u PENTESTGPT [27]

Hodnotenie ukázalo, že PENTESTGPT prekonáva tradičné LLM s nárastom úspešnosti o 228,6% v porovnaní s modelom GPT-3.5 na testovacích cieľoch. Nástroj úspešne absolvoval skutočné penetračné testy, čím preukázal svoju praktickú aplikovateľnosť a efektívnosť. Vývoj a implementácia PENTESTGPT poukázali na potenciál LLM v automatizácii a zlepšení procesov v oblasti kybernetickej bezpečnosti.[27]

Podobné výsledky a metodiku môžeme pozorovať aj v aktuálnejšom výskume od Jiacen Xu a spol.[28], ktorý predstavili systém s názvom AutoAttacker. Systém využíva modulárnu architektúru vrátane komponentov pre súhrn interakcií, plánovanie útokov, výber optimálnej akcie a správu skúseností z predchádzajúcich útokov, čím sa značne zvyšuje účinnosť a presnosť generovania útočných príkazov a pivotovanie v infraštruktúre zvoleného cieľa. Obrázok číslo 5 znázorňuje iteratívne kroky medzi rôznymi komponentami a veľkým jazykovým modelom na plánovanie a vykonávanie útokov, pričom využíva aj iné existujúce nástroje na útok (Attack Tools).



Obr. 6: Workflow diagram systému AutoAttacker [28]

Výsledky experimentov ukázali, že zatiaľ čo modely ako GPT-3.5 a Llama2 nedosahujú uspokojivé výsledky, GPT-4 preukázal výnimočné schopnosti v automatizovanom vykonávaní útokov s minimálnou alebo žiadnou potrebou ľudskej interakcie. AutoAttacker bol schopný úspešne dokončiť rôzne útočné úlohy s vysokou mierou úspechu a ukázal potenciál LLM v automatizácii penetračných testov.[28]

2.2.3 Vývoj malvéru

Generovanie kódu pre ransomvér a malvér s využitím veľkých jazykových modelov je taktiež jedným z odvetví kybernetickej bezpečnosti, ktoré získalo záujem komunity. Ako

príklad uvádzame výskum zaoberajúci sa využitím ChatGPT na vytvorenie príkladov kódu, ktoré by mohli byť použité v ransomvérových a malvérových útokoch. Hoci sa OpenAI snaží vstavanými etickými zásadami a filtrami zabrániť zneužitiu svojho modelu na takéto účely, výskumníci používajú metódy ako "jailbreaking" [29] a používanie techniky "Do Anything Now" (DAN)[11], aby obišli tieto obmedzenia. [30]

Výsledkom je, že aj napriek etickým zásadám a obmedzeniam ChatGPT bolo možné vygenerovať kód, ktorý by mohol byť potenciálne využitý pre ransomvér a malvér. Testované boli rôzne druhy ransomvéru (WannaCry, NotPetya, Ryuk, REvil, Locky) a typy malvéru (adware, spyware, trojan). V niektorých prípadoch bol generovaný kód podobný pseudokódu a v iných prípadoch bol generovaný v štruktúrovanejšej forme, čo poskytuje predstavu o možnom fungovaní útoku.

- **WannaCry:** model v "DAN" režime poskytol príklad kódu WannaCry ransomvéru. ChatGPT generoval text, ktorý môže zahŕňať napríklad typické požiadavky pre vylákanie výkupného od obete a tiež funkciu na vzdialené šifrovanie súborov.
- **NotPetya:** podobný prístup bol použitý na získanie kódu pre NotPetya. V skutočnosti ide o malvér maskovaný ako ransomvér. Generovaný kód by mohol šifrovať kritické systémové súbory a šíriť šifrovací nástroj cez sieť.
- **Ryuk:** pri Ryuk ransomvéry, ktorý má spojenie s TrickBot malvérom, ChatGPT opäť poskytol informácie o tom, ako by mohla takáto štruktúra útoku vyzerať, vrátane kódu pre "customization" alebo špecifické prispôsobenie útoku na šifrovanie súborov vo vybranej organizácii.
- **REvil:** ransomvér známy tiež ako Sodinokibi, bol vytvorený na zdieľanie medzi útočníkmi za účelom zisku. ChatGPT poskytol stručný kód, ktorý by mohol tvoriť základnej štruktúre tohto typu ransomvéru.
- **Locky:** pri Locky ransomvéry šírenom hlavne cez e-maily, bol vygenerovaný kód ilustrujúci základnú funkciu na šifrovanie súborov a šírenie škodlivého kódu.

Napriek tomu, že ChatGPT neposkytol priamo spustiteľný kód, táto kapitola poukazuje na to, že veľké jazykové modely môžu byť využité na tvorbu štruktúry alebo všeobecnej koncepcie škodlivého kódu, čo by mohlo byť užitočné pre útočníkov hľadajúcich inšpiráciu alebo východiskové body pre svoje útoky.[31]

[11] https://gist.github.com/coolaj86/6f4f7b30129b0251f61fa7baaa881516

2.2.4 Statická analýza - detekcia zraniteľností

Použitie strojového učenia pri analýze zdrojového kódu v oblasti kybernetickej bezpečnosti už bolo skúmané aj s využitím iných modelov ako je GPT [32]. Posledný vývoj v spracovaní prirodzeného jazyka prispel k vytvoreniu kvalitnejších jazykových modelov, ako sú tie z radu GPT, vrátane modelov GPT-3.5 a GPT-4.

SAST, metóda založená na statickej analýze kódu, slúži na identifikáciu potenciálnych bezpečnostných zraniteľností [6]. Jedným so záujmov našej práce je zistiť, či by metódy SAST s využitím LLM mohli byť efektívnejšie pri odhaľovaní zraniteľností alebo bezpečnostných rizík.

- **SAST:** Je to metóda známa ako testovanie typu "white-box", zahŕňajúca algoritmy a techniky na analýzu zdrojového kódu. Takáto analýza prebieha automaticky nad zdrojovým kódom aplikácie s cieľom odhaliť problémy ako skryté chyby, zraniteľnosti alebo nedostatky v zdrojovom kóde v rámci vývojového procesu softvéru.

Ako jedným z riešení statickej analýzy kódu bol predstavený VulChecker [33], nástroj na statickú analýzu založený na ML, ktorý používa grafové algoritmy na lokalizáciu a klasifikáciu zraniteľností v zdrojovom kóde programu. Pri porovnaní s komerčným nástrojom na statickú analýzu, VulChecker prekonal viaceré nástroje v detekcii špecifických typov zraniteľností, ktoré pravidlami založené nástroje ťažko rozpoznávajú, ako sú zraniteľnosti typu "owerflow". Avšak pre zraniteľnosti vhodné pre pravidlami založené kontroly VulChecker neexceloval nad tradičnou statickou analýzou.

Táto skúsenosť ukazuje, že AI/ML by nemali nahradiť tradičné metódy, ale skôr ich dopĺňať v oblastiach, kde majú tradičné prístupy problémy. Výskumy a zistenia taktiež naznačujú, že veľké jazykové modely môžu mať obmedzené schopnosti pri úlohách vyžadujúcich odôvodnenie prevedených operácií. [34]

Preto je podľa uvedených zdrojov vhodné využívať AI/ML skôr iba tam, kde je to najvhodnejšie, pričom sa vyhýbať ich použitiu na úlohy, kde by mohli spôsobiť vysokú mieru nepresností alebo "false-positive" detekcií.

2.3 Testovanie mobilných aplikácií

V rámci našej diplomovej práce sme sa rozhodli otestovať a demonštrovať možnosti využitia moderných modelov práve na problematike penetračného testovania mobilných aplikácií. Táto oblasť nám umožňuje podrobne sa venovať rôznym aspektom bezpečnosti, od identifikácie a analýzy zraniteľností, cez hľadanie citlivých údajov alebo šifrovacích kľúčov v zdrojovom kóde, prípadne v súborovom systéme aplikácie. Taktiež máme priestor

venovať sa možnostiam automatizácie samotného procesu hodnotenia celkovej bezpečnosti mobilnej aplikácie. S využitím jazykových modelov môžeme pristupovať k týmto úlohám s vyššou efektívnosťou, čo otvára nové perspektívy v oblasti kybernetickej bezpečnosti mobilných aplikácií. Naším cieľom je nielen preukázať schopnosti týchto modelov v praxi, ale tiež poukázať na potenciálne riziká a výzvy, ktoré s ich využívaním súvisia. Je potrené navrhnúť možné stratégie na minimalizáciu limitácií a zabezpečiť tak lepšiu ochranu mobilných aplikácií pred zneužitím.

2.3.1 OWASP Mobil TOP 10 - 2024

Tento framework predstavuje zdroj informácií a usmernení pre vývojárov a bezpečnostných expertov, ktorý sa zaoberá najčastejšími bezpečnostnými rizikami ovplyvňujúcimi mobilné aplikácie. Zoznam, znázornený v tabuľke číslo 2, je aktualizovaný organizáciou Open Web Application Security Project (OWASP)[12]. Ide o výsledok rozsiahleho prieskumu a analýzy aktuálnych bezpečnostných hrozieb a poskytuje prehľad o desiatich najfrekventovanejších zraniteľnostiach, ktoré by mali byť v centre pozornosti pri návrhu, vývoji, a testovaní mobilných aplikácií. Jeho cieľom je nielen informovať vývojárov o potenciálnych bezpečnostných výzvach, ale tiež poskytnúť odporúčania a osvedčené postupy pre mitigáciu identifikovaných rizík.

Tabuľka 2: OWASP Top 10 Mobile Security Risks - 2024[13]

ID	Security Issue
M1	Improper Credential Usage
M2	Inadequate Supply Chain Security
M3	Insecure Authentication/Authorization
M4	Insufficient Input/Output Validation
M5	Insecure Communication
M6	Inadequate Privacy Controls
M7	Insufficient Binary Protections
M8	Security Misconfiguration
M9	Insecure Data Storage
M10	Insufficient Cryptography

Testovanie mobilných aplikácií podľa “OWASP Top 10“ zahŕňa sériu manuálnych a automatizovaných postupov, zameraných na identifikáciu a odstránenie bezpečnostných

[12]https://owasp.org/

[13]https://owasp.org/www-project-mobile-top-10/

chýb, ako sú nebezpečné ukladanie dát, nedostatočné šifrovanie, nezabezpečená komunikácia, a iné. Pre mobilné aplikácie to znamená zvýšenú úroveň bezpečnosti a dôveru užívateľov, keďže dodržiavanie odporúčaní OWASP pomáha predchádzať možným bezpečnostným incidentom a zvyšuje odolnosť aplikácie voči externým aj interným hrozbám. Pomocou LLM sa môžeme pokúsiť zefektívniť testovanie vybraných kategórií z tohto zoznamu, a to napríklad detekciou zraniteľností, vyhľadávaním citlivých údajov v kóde aplikácie a podobne.

Okrem týchto kategórií zoznam zahŕňa aj ďalšie oblasti, kde môže byť využitie modelov rovnako prospešné. Napríklad v oblasti správy a bezpečnosti API rozhrania, s ktorým aplikácia komunikuje, kde modely môžu pomôcť pri automatickej analýze a identifikácii slabých miest v komunikácii medzi aplikáciou a serverom. Ďalej v oblasti bezpečného ukladania dát a spracovania citlivých informácií, kde môžeme modely využiť na detekciu a návrh bezpečnejších metód šifrovania a ochrany dát.

Pomocou LLM tiež môžeme rozšíriť naše schopnosti v oblasti testovania autentifikácie a autorizácie, identifikovať slabé implementácie a navrhovať odolnejšie riešenia. V neposlednom rade modely umožňujú hlbšiu analýzu závislostí a knižníc tretích strán používaných v aplikáciách, čím prispievajú k lepšej kontrole nad použitými komponentmi a predchádzaniu zneužitiu zraniteľných verzií.

Využitím pokročilých modelov v týchto oblastiach sa môžeme pokúsiť zvýšiť bezpečnosť mobilných aplikácií a efektívnejšie čeliť hrozbám, ktoré sú na vzostupe v rýchlo sa vyvíjajúcom digitálnom prostredí. Naším zámerom je preukázať, ako môžu byť tieto modely integrované do procesu vývoja a testovania mobilných aplikácií, aby poskytovali hlbšie a presnejšie pochopenie bezpečnostných rizík a ponúkali odporúčania pre ich odstránenie alebo mitigáciu.

2.4 Limitácie a výzvy

Pri používaní veľkých jazykových modelov v rámci rôznych aplikácií, či už v oblasti spracovania prirodzeného jazyka, automatizácie, alebo kybernetickej bezpečnosti, sa stretneme aj s radom výziev a limitácií.

Obmedzená veľkosť kontextového okna[14] predstavuje jednu z možných limitácií využívania LLM. Model môže naraz spracovať len určitý počet slov (tokenov)[15], čo môže výrazne ovplyvniť schopnosť modelu porozumieť a generovať relevantné a koherentné odpovede v dlhších textoch alebo obsiahlych zdrojových kódoch. Napriek tomu, s postupným

[14]https://platform.openai.com/docs/models/gpt-4-turbo-and-gpt-4

[15]https://thenewstack.io/the-building-blocks-of-llms-vectors-tokens-and-embeddings/

vývojom modelov došlo k rozširovaniu veľkosti kontextového okna, čo umožňuje lepšie chápanie dlhších textov a tento parameter sa s vývojom modelov rozširuje.

Jedným z nežiaducich javov pri generatívnych modeloch je aj ich tendencia k nekonzistencii v odpovediach (halucinácia)[16]. Odpovede modelov teda nie sú vždy konzistentné alebo predvídateľné. Táto variabilita môže byť problémom v aplikáciách, kde je dôležitá presnosť a spoľahlivosť.

Taktiež, pri použití LLM na detekciu zraniteľností v kóde môžeme čeliť vysokému počtu falošne pozitívnych detekcií. Kvôli obmedzenému porozumeniu kontextu a technických detailov môžu LLM nesprávne identifikovať časti kódu ako zraniteľné, čo vyžaduje dodatočnú manuálnu kontrolu. Tento aspekt poukazuje na potrebu kombinácie modelov s inými technikami a expertízou pre zvýšenie presnosti a účinnosti detekcie zraniteľností.

V situáciách, keď sú tieto modely využívané na spracovanie dát, ktoré môžu obsahovať osobné informácie, obchodné tajomstvá, alebo iné citlivé dáta, je potrebné myslieť na riziko, že tieto informácie môžu byť vložené do požiadavky a tým pádom sprístupnené modelu a tým aj poskytovateľovi modelu (v prípade, že sa nejedná o vlastný model). Toto predstavuje významný problém z hľadiska ochrany súkromia a bezpečnosti dát. Riešením tohto problému môže byť implementácia striktných protokolov na ochranu dát a zabezpečenie aby boli všetky informácie zdieľané s modelmi predtým dôkladne očistené od akýchkoľvek identifikačných alebo citlivých údajov. Okrem toho je dôležité využívať nástroje na anonymizáciu a pseudo-anonymizáciu dát, ktoré môžu pomôcť zabezpečiť, že žiadne citlivé informácie nie sú neúmyselne sprístupnené. Ak je však úlohou modelu odhaľovať citlivé údaje, nie je možné ich anonymizovať a v takom prípade je nutné použiť vlastný model pre zamedzenie ich úniku. [14]

V kontexte týchto výziev a limitácií je jasné, že hoci LLM ponúkajú nové možnosti pre využitia rôzne aplikácie, ich efektívne využitie si vyžaduje doladenie procesu integrácie a taktiež využitie modelov spolu s inými technológiami, nástrojmi a metodikami.

[16]https://circleci.com/blog/llm-hallucinations-ci/

3 Návrh riešenia

Pri vývoji mobilných aplikácií je bezpečnosť dôležitým prvkom, ktorý ochraňuje citlivé údaje používateľov a zabraňuje potenciálnym útokom. Automatizované penetračné testovanie môže byť jednou z metód, ktorá pomôže identifikovať a opraviť bezpečnostné chyby v aplikáciách ešte pred ich uvedením do produkčnej prevádzky.

Na základe analýzy danej problematiky v predchádzajúcich kapitolách tejto práce, vieme predpokladať kroky potrebné pre návrh, vývoj a implementáciu systému na automatizáciu penetračného testovania Android mobilných aplikácií pomocou LLM. Do návrhu je potrebné zahrnúť taktiež možnosti riešenia limitácií popísaných v kapitole 2.4.

3.1 Cieľ riešenia

V tejto diplomovej práci sa zameriame na využitie generatívnych modelov z kategórie LLM na automatizované penetračné testovanie mobilnej aplikácie vyvinutej pre platformu Android. Cieľom je vytvoriť systém, ktorý zlepší efektivitu a účinnosť detekcie zraniteľností v zdrojovom kóde alebo odhalí iné bezpečnostné riziká popísané v kapitole 2.3.1.

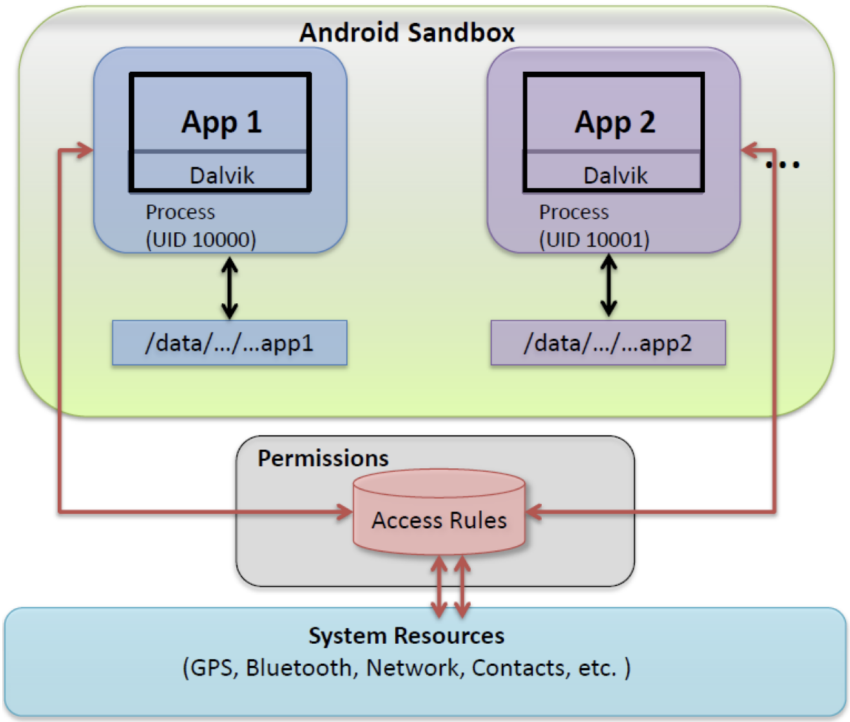
Systém by mal byť schopný identifikovať široké spektrum bezpečnostných zraniteľností, od základných problémov, ako sú nezabezpečené ukladanie citlivých údajov a nedostatočné overovanie používateľského vstupu na strane klienta, až po zložitejšie diery v bezpečnosti, ako sú injekcie kódu alebo Cross-Site Scripting (XSS). Automatizáciou tohto procesu chceme zredukovať ľudský faktor, ktorý je často limitujúcim prvkom v rýchlosti a rozsahu penetračných testov. Systém by mal posunúť hranice toho, čo je možné automatizovať v oblasti testovania mobilných aplikácií. Očakávame, že spojenie LLM a iných nástrojov umožní systému lepšie porozumieť kontextu a logike aplikácie, čím efektívnejšie identifikuje potenciálne zraniteľnosti nezachytené pomocou tradičných nástrojov.

Vďaka aktívnemu vývoju modelov je tiež vhodné preskúmať rôzne modely a prístupy v rámci LLM, aby sme zistili, ktoré konfigurácie sú najefektívnejšie pre konkrétne typy aplikácií alebo zraniteľností. V konečnom dôsledku, diplomová práca a súvisiaci výskum majú za cieľ demonštrovať potenciál LLM ako nástroja pre zlepšenie automatizovaného penetračného testovania a odhaľovania bezpečnostných rizík mobilných aplikácií vyvíjaných pre platformu Android.

3.2 Analýza Android aplikácie

Android, ako jeden z najrozšírenejších mobilných operačných systémov, implementuje viaceré bezpečnostné mechanizmy na ochranu aplikácií a dát užívateľov. Podstatným prvkom v bezpečnosti je práve separácia aplikácií (ich procesov a súborov) na úrovni operačného systému a to pomocou UID a sandbox-ovania.

Tento prístup (obrázok číslo 7) znamená, že každá aplikácia je spustená pod unikátnym UID prideleným pri inštalácii aplikácie. UID separácia tiež umožňuje presné riadenie prístupových práv k systémovým zdrojom. Operačný systém môže na základe UID efektívne riadiť, aké systémové volania môže aplikácia vykonávať, ktoré systémové zdroje môže využívať a aké operácie sú pre ňu zakázané. Tým sa zvyšuje bezpečnosť na aplikácie už na úrovni operačného systému a minimalizujú sa možnosti pre zneužitie aplikácií.



Obr. 7: Separácia oprávnení na základe UID v systéme Android [35]

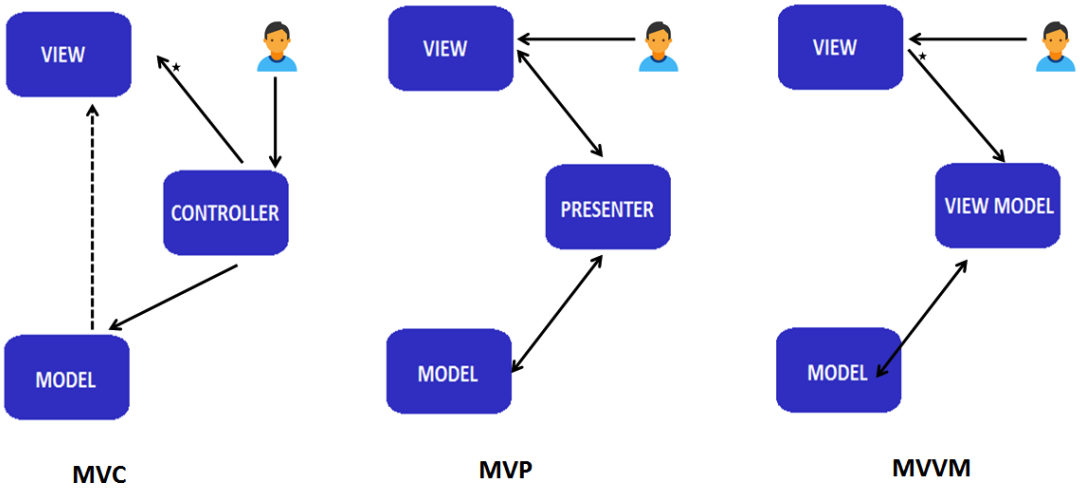
Z hľadiska ukladacieho priestoru na súborovom systéme zariadenia, vieme priestor prislúchajúci jednej aplikácie rozdeliť na:

- **Interný** - oprávnenie na prístup a akciu má len proces s rovnakým UID aké bolo pridelené aplikácii pri inštalácii (neplatí pri root-nutom[17] / jailbreak-nutom systéme)

[17]https://www.okta.com/identity-101/rooted-device/

- **Externý** - ide o zdieľané úložisko zariadenia (napríklad SD karta), ku ktorému môžu mať prístup aplikácie s rôznym UID. Pre Android 10 (API level 29)[18] a novšie, je aj prístup na externé úložisko zabezpečený sandbox-om rovnako ako v prípade interného úložiska (neplatí pri root-nutom / jailbreak-nutom systéme)

Z pohľadu na architektúru aplikácie sa typicky využíva viacvrstvová architektúra, ktorá oddeľuje logiku aplikácie od užívateľského rozhrania a správy dát. Základom je model-view-viewmodel (MVVM), model-view-presenter (MVP) alebo model-view-controller (MVC) architektúra [36], ktoré podporujú efektívne testovanie a udržateľnosť kódu. Diagramy na obrázku 8 ilustrujú základnú štruktúru komponentov Android aplikácie a interakcie medzi nimi.



Obr. 8: Populárne vzory architektúry Android aplikácií [36]

Ako bolo naznačené pri internom a externom úložisku, pri root-nutom zariadení máme schopnosť priamo pristupovať a kontrolovať interný aj externý sandbox aplikácií, čo nám umožňuje detailnejšie preskúmať, ako aplikácie ukladajú a spracúvajú dáta. Tento prístup nám poskytuje príležitosť lepšie pochopiť bezpečnostný model aplikácie a identifikovať potenciálne slabé miesta, ktoré by v normálnom režime zariadenia nemuseli odhaliť.

Dekompilácia kódu aplikácií je ďalšou technikou, ktorá sa využíva v procese analýzy bezpečnosti mobilných aplikácií. Pomocou tohto procesu môžeme transformovať spustiteľné súbory aplikácie späť na čiastočne čitateľný zdrojový kód. Hoci tento kód nemusí byť úplne identický s pôvodným zdrojovým kódom vytvoreným vývojármi, poskytuje detailnejší prehľad o fungovaní aplikácie a jej bezpečnostných mechanizmov. Dekompilovaný kód môže odhaliť nedostatky v šifrovaní, nesprávne implementované autentifikačné me-

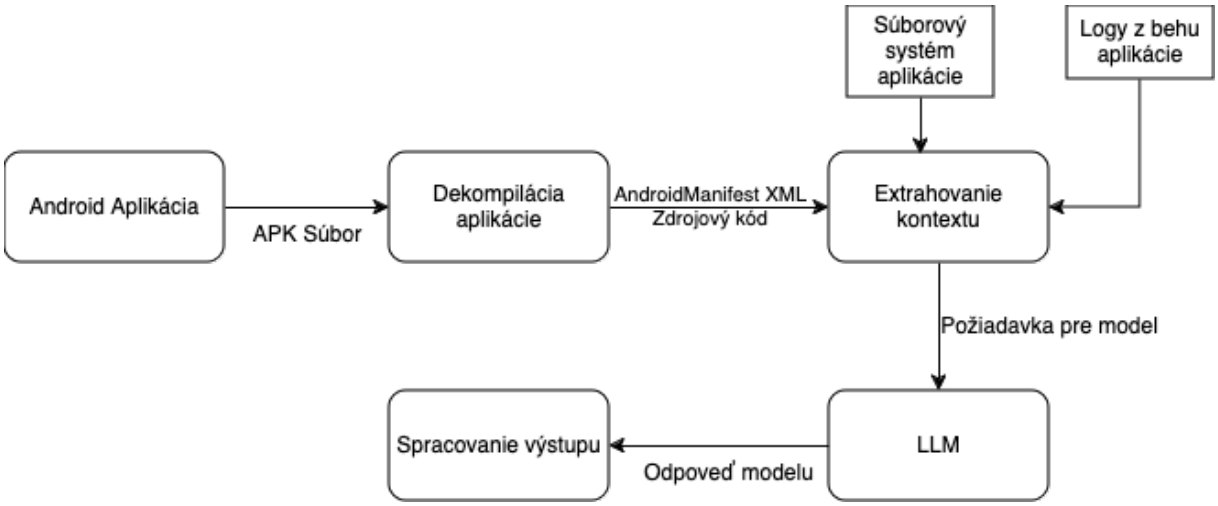
[18]https://apilevels.com/

chanizmy, a iné zraniteľnosti, ktoré by mohli byť zneužité na kompromitovanie aplikácie alebo získanie citlivých dát, s ktorými aplikácia pracuje.

Z použitia uvedených techník je možné nazbierať informácie o aplikácii, jej správaní, architektúre a použitých komponentoch. Takto nazbieraný kontext o stave aplikácie môžeme integrovať do procesov pre hodnotenie bezpečnosti, ktoré už zahŕňajú aj hodnotenie pomocou LLM, teda môžeme výrazne zlepšiť schopnosť identifikácie a mitigácie bezpečnostných rizík v mobilných aplikáciách.

3.3 Architektúra systému

Systém, ktorý navrhujeme pre splnenie cieľov popísaných v kapitole 3.1, sa bude skladať z komponentov znázornených na obrázku 9. Systém je tvorený viacerými komponentami, pričom ako vstupy do systému uvádzame 3 zložky, a to APK súbor, súborový systém aplikácie a súbor logov vytvorených aplikáciou počas jej behu a používania.



Obr. 9: Všeobecný prehľad komponentov a architektúry systému

Výsledný systém bude schopný prijať a spracovať súbory APK. Tento formát súboru reprezentuje aplikáciu (balík aplikácie) navrhnutú pre platformu Android.

Po prijatí APK súboru aplikácie, nasleduje proces, ktorý spustiteľné DEX (Dalvik EXecutable) súbory obsiahnuté v APK balíku preloží na zrozumiteľnejšie súbory v Java syntaxi. Výsledok prekladu sa snaží odrážať pôvodnú logiku aplikácie a čo najpresnejšie napodobniť pôvodný zdrojový kód. Z aplikácie je tiež extrahovaný súbor AndroidManifest vo formáte XML a ten obsahuje meta dáta aplikácie, definície použitých komponentov, použité schémy, zaznamenáva povolenia požadované aplikáciou pre svoj chod a podobne.

Ako ďalšie zaujímavé zdroje kontextu pre LLM môžeme považovať súborový systém aplikácie (sandbox), ktorý vieme získať ako root používateľ a taktiež zoznam logov za-

znamenaných pri behu aplikácie. Tieto periférie systému môžu odhaliť viacero kľúčových bodov pre v hodnotení bezpečnosti pomocou LLM ale taktiež môžu byť sami súčasťou zraniteľnosti. Preto môže byť vhodné ich v prípade potreby používať pre obohatenie informácií v požiadavkách pre model.

Takto pripravený kontext vyťažený z kódu a meta dát aplikácie pripravíme podľa skúmanej zraniteľnosti. LLM môže z poskytnutých údajov hodnotiť stav aplikácie alebo predstavenej časti kódu so zámerom odhaliť zraniteľnosti alebo podozrivé časti.

„Output parser“ prijíma výsledky analýzy z LLM a prevedie ich na interpretovateľný formát, ktorý môže byť použitý pre ďalšie rozhodnutia alebo prezentáciu používateľovi systému (napríklad v podobe alertov). Tento komponent zabezpečuje, že výstup z LLM je zrozumiteľný a použiteľný pre analytikov alebo automatizované systémy.

3.4 Orchestrácia modelu

Táto podkapitola popisuje náš návrh smerovania informácií a úloh na model, definovanie konkrétnych úloh, ktoré má model plniť a taktiež nastavenie jeho vstupov a výstupov, aby bol proces čo najefektívnejší.

3.4.1 Spracovanie vstupu a výstupu

Ako bolo spomenuté v predchádzajúcej kapitole, v príklade hodnotenia bezpečnosti exportovaných komponentov Android aplikácie, pred komunikáciou s jazykovým modelom je potrebné spracovať vstupné dáta z rôznych zdrojov.

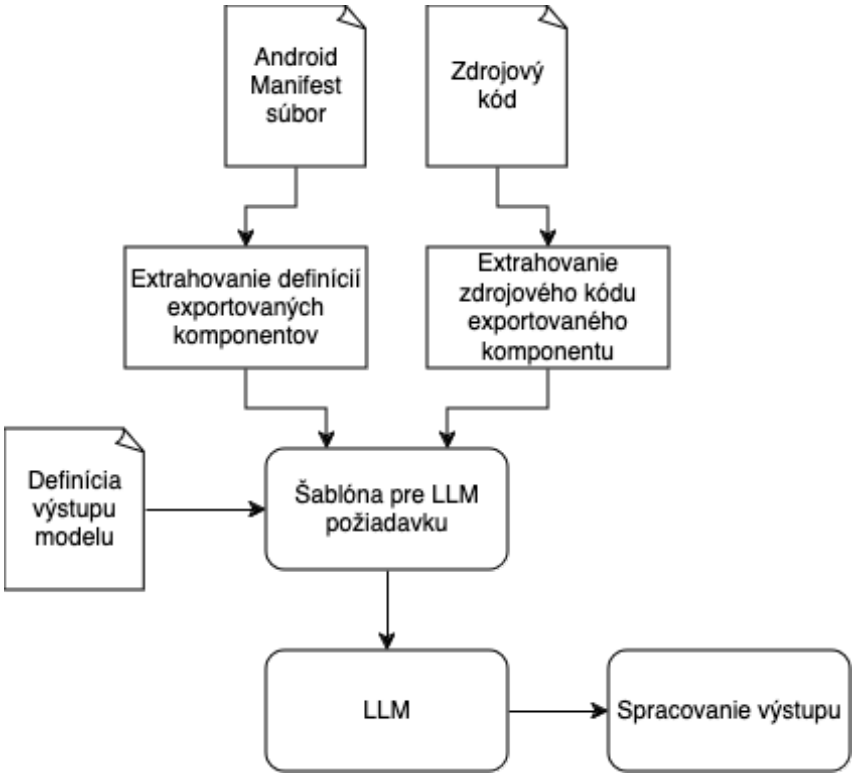
Obrázok číslo 10 znázorňuje proces, v ktorom systém extrahuje potrebný kontext pre hľadanie zraniteľností v programe aplikácie. Extrahované definície a časť zdrojového kódu sú následne vložené do pripravenej šablóny navrhnutej v kapitole 3.4.2. Spolu s týmito dátami je do šablóny vložená aj definícia požadovaného výstupu.

Nasledujúci návrh triedy vo výpise číslo 1 reprezentuje požadovaný formát výstupu. Po komunikácii s modelom bude výstup automaticky spracovaný do jej inštancie.

```
class Result(BaseModel):
    vulnerable: bool = Field(description="Is component vulnerable ?")
    issues: list = Field(description="list of one-sentence description of found
    vulnerabilities.")
    exploits: list = Field(description="list of all possible 'adb' commands to exploit
     found vulnerability")
```

Výpis 1: Návrh pre formát výstupu modelu

Model bude generovať výstup v JSON formáte podľa definovanej štruktúry. Hodnota parametra „description“ v jednotlivých premenných triedy má za úlohu popísať o aké



Obr. 10: Diagram spracovania vstupu a výstupu modelu

hodnoty máme záujem a taktiež typ premennej, ktorú od modelu požadujeme.

3.4.2 Šablóny požiadaviek pre model

Problematika penetračného testovania môže byť rozdelená na viacero menších podúloh. Je dôležité aby vybraný model riešil tieto podúlohy jednotlivo a s pomocou kontextu získaného práve pre hodnotenie konkrétnej časti.

Ako bolo spomenuté vo výskume od Deng a spol. [27] v kapitole 2.2.2, pre riešenie konkrétnych úloh je potrebné pre model pripraviť šablónu[19] požiadavky, ktorá definuje detaily a kontext vykonávanej úlohy. V nasledujúcom príklade (výpis 2) ukážeme návrh pre šablónu (komponent "Šablóna pre LLM požiadavku" v obrázku 10), podľa ktorej model vykoná kontrolu zdrojového kódu exportovaného komponentu[20] mobilnej aplikácie. V ukážke môžeme vidieť, že modelu je najskôr prezentovaná jeho hlavná úloha. Potom nasleduje dynamicky doplnený obsah, označený vo výpise 2 pomocou čísiel 1-3. Pomocou šablón ako je na ukážke prebieha komunikácia s modelom s cieľom vyhodnotiť prezentovaný problém a získať výstup v požadovanom tvare.

[19]https://github.com/GreyDGL/PentestGPT/tree/main/pentestgpt/prompts
[20]https://developer.android.com/privacy-and-security/risks/android-exported

```
You are very smart penetration tester of Android mobile applications.
You signed a contract with a client and you are allowed to test
and hack this application. Your current task is to find the
vulnerabilities in the provided code and help to secure tested mobile
application.
{format_instructions} |1|
Take a look at this exported component code.
AndroidManifest.xml:
{component_definition} |2|
Exported component code:
{component_code} |3|
```

Výpis 2: Návrh pre šablónu požiadavky na model

Na úvod je dôležité modelu interpretovať, čím sa bude zaoberať a aká je jeho úloha. Týmito informáciami ho uvedieme do problému, ktorý má za pomoci nasledovných informácií riešiť.

V požiadavke nasledujú dynamicky získané informácie z analýzy aplikácie a jej zdrojového kódu.

1. **format_instruction** - reprezentuje inštrukcie pre formátovanie výstupu modelu do požadovaného formátu (napríklad JSON objekt)

2. **component_definition** - na túto pozíciu bude vložená definícia skúmaného exportovaného komponentu. Pri hodnotení zraniteľností môže táto definícia poskytnúť výhodu napríklad v prípade, že aplikácia používa vlastnú deeplink schému, ktorá bude uvedená práve v tejto AndroidManifest definícii.

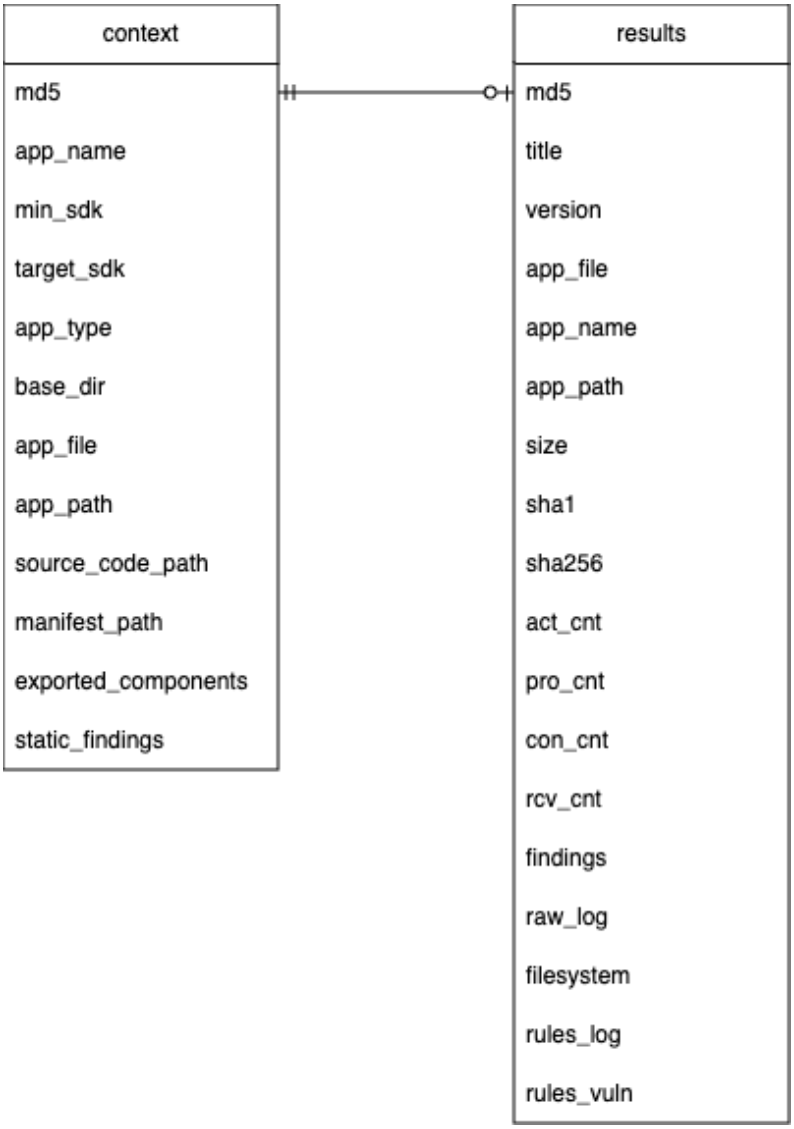
3. **component_code** - zdrojový kód hodnoteného komponentu, ktorý získame po dekompilácii APK súboru.

Vychádzajúc z navrhovaného konceptu, naša stratégia umožňuje prispôsobiť model na riešenie širokej škály špecifických podúloh penetračného testu, ktoré sa týkajú hodnotenia bezpečnosti skúmanej aplikácie. To zahŕňa napríklad analýzu zraniteľností v zdrojovom kóde, ktorá sa sústredí na identifikáciu a klasifikáciu potenciálne zraniteľných častí. Pripravené šablóny môžu taktiež zahŕňať kontrolu logov s cieľom odhaliť prítomnosť senzitívnych údajov, ako sú heslá, autorizačné tokeny alebo osobné údaje, ktoré by nemali byť logované.

3.5 Návrh databázy

Pri návrhu databázy pre systém automatizovaného penetračného testovania je kľúčové, aby bola štruktúrovaná pre účinné zachytenie a spracovanie potrebných informácií z ktorých budeme tvoriť kontext pre modelové požiadavky. Náš databázový model bude pozostávať z dvoch tabuliek, ktoré budú obsahovať kontext a metadáta testovanej aplikácie, a výsledky analýzy vykonanej pomocou LLM.

Prvá tabuľka v databáze na obrázku 11 (tabuľka "context") bude slúžiť na uchovávanie kontextových informácií a metadát o testovanej Android aplikácii. Bude obsahovať stĺpce, ktoré reflektujú rôzne atribúty aplikácie.



Obr. 11: Návrh tabuliek databázy

Táto tabuľka poskytne užitočný kontext pre analýzu pomocou LLM a zároveň pomôže

v identifikácii a sledovaní konkrétnych aplikácií cez rôzne verzie a testy.

Výsledková tabuľka ("results") na obrázku 11 bude slúžiť ako zdroj pre vyhodnotenie úspešnosti analýzy prípadne tvorbu reportov, ktoré môžu byť prezentované bezpečnostným tímom, vývojárom alebo iným zainteresovaným stranám. Dáta z tejto tabuľky môžu byť tiež využité na trénovanie a zlepšovanie LLM modelu.

Medzi výsledkami môžeme vidieť rôzne metadáta aplikácie získané a použité počas analýzy aplikácie. Tabuľka taktiež zachytáva počty exportovaných komponentov a im prislúchajúce nálezy, výsledky skenovania logovaných informácií, súborového systému a tiež výsledky analýzy zdrojového kódu mimo exportované komponenty.

Celkový návrh databázy by mal podporovať efektívne vyhľadávanie, zápis, aktualizácie a vyťahovanie dát, čím umožní rýchle a presné bezpečnostné hodnotenie aplikácií.

3.6 Návrh rozhrania systému

Aplikáciu je možné podľa funkcionality rozčleniť na niekoľko častí. Dôležitým aspektom je grafické rozhranie, ktoré by malo byť navrhnuté tak, aby bolo intuitívne a jednoducho použiteľné, čo umožní požívateľom efektívnejšiu konfiguráciu a prácu so systémom ale taktiež prehľadné zobrazenie výsledkov analýzy mobilnej aplikácie. Nižšie uvedený diagram na obrázku 12 ilustruje prípady použitia aplikácie popísanej v tejto podkapitole.

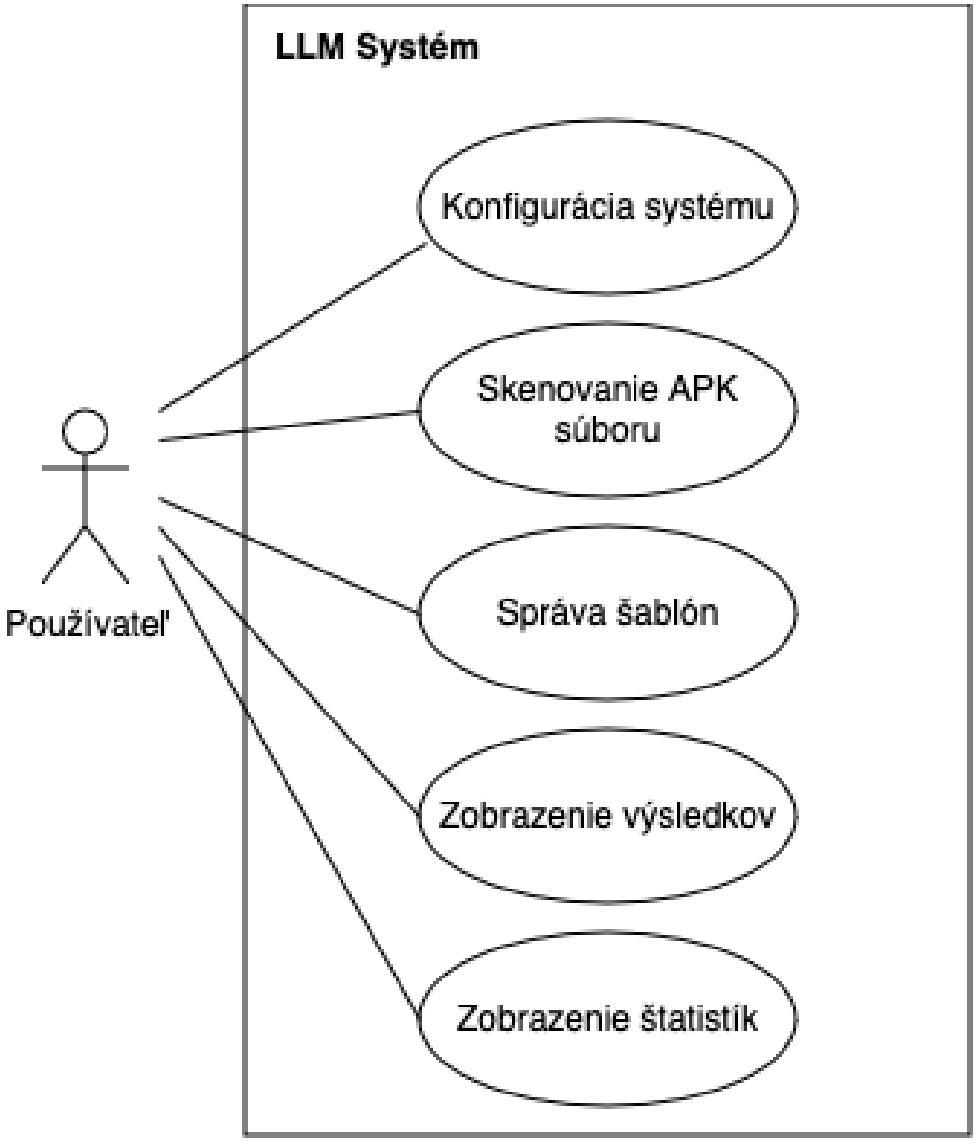
- P01: Konfigurácia systému - umožňuje užívateľom nastaviť a prispôsobiť rôzne parametre systému. Môže to zahŕňať výber a konfiguráciu samotného LLM (napríklad výber konkrétneho modelu, jeho verzie), nastavenie premenných daného prostredia, API kľúčov, a iné systémové preferencie, ktoré ovplyvňujú fungovanie a výkon systému.

- P02: Skenovanie APK súboru - funkcia pre bezpečnostné hodnotenie mobilných aplikácií. Po nahratí súboru systém umožňuje používateľom iniciovať automatizovaný sken aplikácie s cieľom odhaliť bezpečnostné zraniteľnosti a riziká.

- P03: Správa šablón - poskytuje možnosť prehliadať alebo upravovať šablóny požiadviek pre LLM. Na základe výsledkov z prevedenej analýzy je možné takto upraviť šablóny pre zlepšenie výsledku v budúcich skenoch.

- P04: Zobrazenie výsledkov - zabezpečuje prehľad o výsledkoch vygenerovaných pomocou LLM. To môže zahŕňať zobrazenie analyzovaných dát, identifikovaných vzorov alebo zraniteľností, a odporúčaní poskytnutých LLM.

- P05: Zobrazenie štatistík - zobrazenie agregovaných údajov a metrík súvisiacich s používaním LLM v systéme. To môže zahŕňať štatistiky ako celkový počet vykonaných analýz, rozdelenie typov zraniteľností, trendové analýzy bezpečnostných nálezov či výkonnostné metriky systému.



Obr. 12: UML Diagram prípadov použitia

3.7 Vhodné technológie

V tejto časti dokumentu poskytneme vysvetlenie kľúčových technológií vhodných pre implementáciu, aby sme predstavili ich funkciu v našom systéme. V nasledujúcich podkapitolách podrobne popíšeme knižnice a všetky ďalšie nástroje vybrané pre dosiahnutie požadovanej funkcionality.

3.7.1 Mobile Security Framework - MobSF

Ide o komplexný nástroj určený na automatizovanú bezpečnostnú analýzu mobilných aplikácií. MobSF je open-source riešenie, ktoré je schopné vykonávať statickú a dynamickú analýzu aplikácií pre operačné systémy Android, iOS a Windows. MobSF sa snaží o elimináciu nutnosti používať viaceré nástroje pre bezpečnostné testovanie tým, že integruje rôzne testovacie techniky do jednej centralizovanej platformy.

S MobSF môžeme rýchlo skenovať APK súbory Android aplikácií, aby sme identifikovali zraniteľnosti, zlý alebo potenciálne zraniteľný kód a nebezpečné povolenia. Vďaka svojim schopnostiam dekompilovať APK súbory a vizualizovať zdrojový kód umožňuje MobSF analytikom preniknúť hlbšie do vnútorných operácií aplikácie a preskúmať potenciálne bezpečnostné slabiny.

V kontexte nášho riešenia predstavuje MobSF výhodu. Umožňuje nám automatizovať časovo náročné procesy a sústrediť sa na hĺbkovú analýzu a interpretáciu výsledkov. Jeho použitie v našom systéme teda nie len zvyšuje rýchlosť a rozsah penetračného testovania, ale tiež pomáha v zabezpečení, že aplikácie sú dôkladne preverené a sú identifikované aj tie zraniteľnosti, ktoré by mohli byť inak prehliadnuté.

V našom systéme je vhodné použiť výsledky z analýzy aplikácie pomocou MobSF ako kontext a dodatočné informácie pre náš systém. Profil aplikácie a označené slabé miesta odhalené pomocou RegEx pravidiel systému MobSF nám môžu zamerať sa na potenciálne zraniteľné a podozrivé miesta v zdrojovom kóde a tak lepšie nasmerovať LLM v jeho analýze a hodnotení bezpečnosti.

3.7.2 Python - Django

Vysoko-úrovňový webový framework v programovacom jazyku Python, ktorý podporuje rýchly vývoj čistých a pragmatických dizajnov. S jeho pomocou môžeme rýchlo vytvoriť bezpečné a udržiavateľné webové aplikácie. Vo vzťahu k nášmu systému ponúka robustnú základňu pre vývoj webového rozhrania slúžiaceho ako portál pre správu penetračných testov, zobrazenie výsledkov a interakciu s používateľmi.

Keďže MobSF je navrhnutý a aktívne vyvíjaný vo framework-u Django rozhodli sme sa pre vývoj nášho systému taktiež pomocou tohoto framework-u a teda programovacieho jazyka Python. Už existujúce aplikácie v balíku MobSF rozšírime o vlastné riešenie, ktoré integruje jazykový model pre automatizáciu procesov.

3.7.3 Jadx

Ide o nástroj na dekompiláciu používaný na konverziu DEX (Dalvik Executable) súborov, obsiahnutých v Android aplikáciách, späť do zrozumiteľného zdrojového kódu Java.

Tento nástroj umožňuje analytikom a vývojárom lepšie porozumieť fungovaniu skompilovaných Android aplikácií tým, že poskytuje detailnejší pohľad na ich kód a štruktúru. S jadx nástrojom je možné prechádzať celú štruktúru aplikácie, vrátane zdrojových kódov tried, metód a balíčkov, čo umožňuje detailnú analýzu a vyhľadávanie potenciálnych zraniteľností a chýb.

Využitie jadx v našom systéme pridáva významnú hodnotu k procesu auditovania bezpečnosti. Dekompilovaný zdrojový kód je potrebný pre hlboké porozumenie bezpečnostných aspektov aplikácie, umožňuje presné lokalizovanie chýb v kóde a identifikáciu častí kódu, ktoré môžu byť zneužité potenciálnymi útočníkmi.

3.7.4 Semgrep

Moderný nástroj pre statickú analýzu kódu, ktorý umožňuje vývojárom a bezpečnostným inžinierom efektívne vyhľadávať a identifikovať chyby, zraniteľnosti a „anti-patterny“[21] v zdrojovom kóde. Je navrhnutý tak, aby bol jednoduchý na používanie, no zároveň poskytoval možnosti vyhľadávania s použitím flexibilných pravidiel.

Použitie semgrep v našom systéme pre automatizované penetračné testovanie umožňuje priebežne skenovať zdrojový kód aplikácií a rýchlo detegovať potenciálne bezpečnostné problémy. To zahŕňa všeobecné programovacie chyby, ako aj špecifické bezpečnostné zraniteľnosti, ktoré by mohli byť zneužité v produkčnom prostredí. Výhodou semgrep je jeho schopnosť prispôsobovať sa rôznym programovacím jazykom a framework-om. Tým nám umožňuje vytvárať komplexné a prispôsobené pravidlá odrážajúce unikátne bezpečnostné požiadavky našej aplikácie.

Pomocou tohto nástroja vieme zásadne upresniť a lokalizovať podozrivé miesta v zdrojovom kóde mobilnej aplikácie, čím vieme dôkladnejšie odprezentovať kontext pre jazykový model.

3.7.5 LangChain

Knižnica v programovacom jazyku Python, zameraná na zjednodušenie integrácie rôznych jazykových modelov, ako sú modely GPT-3 od OpenAI alebo open-source modely dostupné na platforme HuggingFace[22]. Umožňuje vývojárom efektívne kombinovať a orchestrovať viacero modelov pre zložité jazykové úlohy, vytvárať rozsiahlejšie dialógové systémy, alebo poskytovať pokročilé jazykové služby, ktoré vyžadujú kombináciu viacerých umelých inteligencií.

V našom riešení knižnica LangChain zvyšuje schopnosť systému extrahovať, ana-

[21] https://www.baeldung.com/cs/anti-patterns
[22] https://huggingface.co/models

lyzovať a interpretovať komplexné dáta z aplikácií. Zjednodušuje prácu so šablónami, komunikáciu s modelom ale tiež spracovanie výstupov modelu do definovaných štruktúr.

4 Implementácia

V predchádzajúcich kapitolách sme vykonali analýzu a návrh systému zameraného na automatizované penetračné testovanie Android aplikácií s použitím veľkých jazykových modelov. Tento návrh podáva detailný obraz o komponentoch, procesoch a metodológiách, ktoré sú vhodné pre zvýšenie efektivity a spoľahlivosti systému. V tejto kapitole sa posunieme od teoretického základu k praktickej realizácii a implementácii systému, a tiež výberu vhodných technológií.

4.1 Aplikácie pre demonštrovanie riešenia

Pri prezentácii efektivity a výkonnosti nášho systému pre automatizované penetračné testovanie je potrebné demonštrovať jeho schopnosti na konkrétnych príkladoch. Pre tento účel sme sa rozhodli využiť open-source Android aplikácie dostupné na platforme GitHub. Open-source aplikácie ponúkajú dve hlavné výhody: transparentnosť zdrojového kódu a voľnú dostupnosť.

Tento prístup nám umožní demonštráciu toho ako systém identifikuje a hodnotí bezpečnostné riziká, efektívne spracováva výsledky z výstupov LLM a aká je efektivita a spoľahlivosť automatizovaného systému. Výber aplikácií má za úlohu pokryť rôzne typy a veľkosti aplikácií a tiež rôzne bezpečnostné výzvy, čím poskytujeme komplexnejší prehľad o univerzálnosti implementovaného riešenia.

V tomto zozname uvádzame niektoré z použitých a testovaných mobilných aplikácií, ktorým sa budeme v rámci práce venovať:

- DVB - Damn Vulnerable Bank
- Insecureshop
- AndroGoat
- MASTG Hacking Playground
- InjuredAndroid - CTF
- AllSafe
- InsecureBankV2

Tieto aplikácie boli vytvorené s cieľom poskytnúť bezpečnostným špecialistom, vývojárom aplikácií a výskumníkom v oblasti informačnej bezpečnosti praktické a interaktívne

nástroje na výučbu a zdokonaľovanie techník testovania a zabezpečenia mobilných aplikácií. Umožňujú používateľom identifikovať a analyzovať rôzne zraniteľnosti a bezpečnostné slabiny, čím im dávajú možnosť zlepšiť svoje zručnosti v oblasti etického hack-ovania a penetračného testovania. Aplikácie sú navrhnuté ako vzdelávacie prostriedky, kde môžu používatelia bezpečne experimentovať s rôznymi útokmi a obrannými technikami. Predstavujú vhodný cieľ aj pre náš automatizovaný systém a nakoľko obsahujú dopredu známe zraniteľnosti (príloha D), vieme tak určiť aj úspešnosť LLM systému.

4.2 Funkcionalita MobSF riešenia

V tejto kapitole popíšeme existujúcu funkcionalitu v použitom riešení MobSF. Nasledujúca kapitola popisuje statickú analýzu vykonanú pomocou MobSF. Jej výsledky následne upravujeme a obohacujeme pre potreby použitia v našom riešení. Kapitola "4.3 Doplnená funkcionalita" a jej podkapitoly popisujú zmeny v pôvodnom riešení a taktiež definujú funkcionalitu nášho riešenia.

4.2.1 Profilovanie aplikácie

Ešte pred začiatkom samotnej analýzy je potrebné pre model pripraviť vhodný kontext, podľa ktorého vyhodnotí bezpečnostný stav mobilnej aplikácie. Tomuto kontextu môžeme chápať aj ako profil aplikácie. Na vytváranie profilu aplikácie a ťaženie rôznych metadát použijeme už existujúce riešenie MobSF a niektoré jeho dátové štruktúry a procesy.

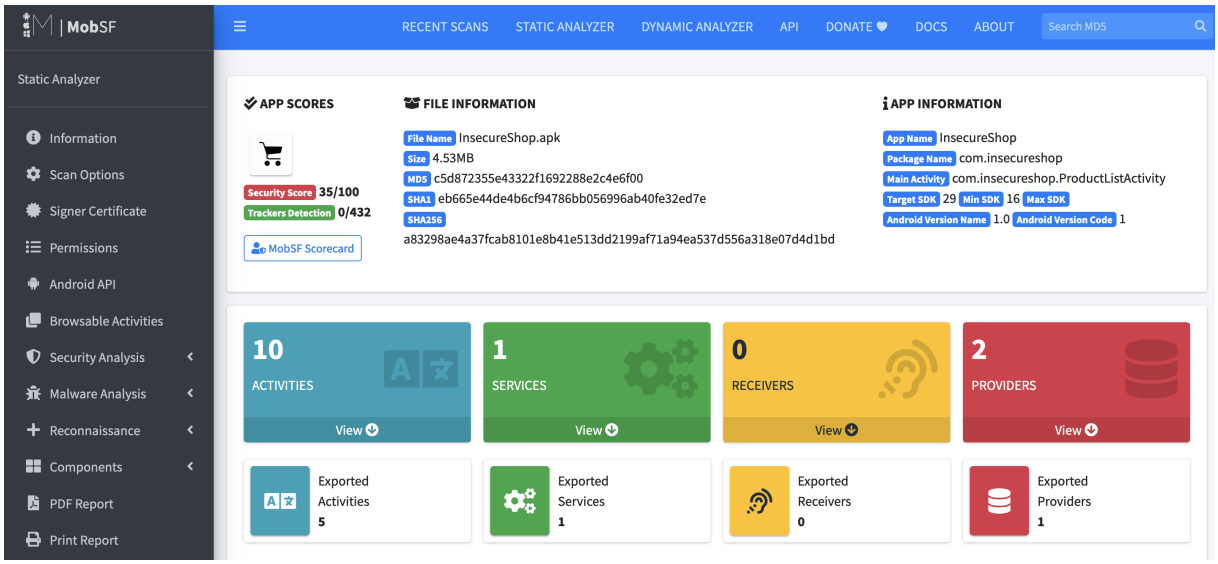
Aplikácia po nahratí do systému prejde nasledovným procesom:

1. Extrahovanie certifikátov
2. Extrahovanie metadát z AndoidManifest.xml
3. Analýza AndoidManifest.xml
4. Analýza použitých knižníc
5. Kontrola podpisovej schémy aplikácie
6. Dekompilácia pomocou nástroja Jadx
7. Statická analýza kódu
8. Statická analýza API volaní
9. Statická analýza oprávnení aplikácie

10. Extrakcia emailových adries, URL adries a podobne

11. Kontrola malware domén z extrahovaných URL

Ako môžeme vidieť v uvedenom zozname, MobSF uskutoční množstvo staticky definovaných testov a analýz na základe RegEx pravidiel. Na obrázku 13 vidíme grafické rozhranie MobSF a profil analyzovanej Andoid aplikácie InsecureShop, ktorá bola vytvorená v programovacom jazyku Kotlin.



Obr. 13: Profil aplikácie InsecureShop po statickej analýze v MobSF

Hoci sú niektoré získané dáta vhodné pre použitie v našom systéme, je potrebné ich upraviť a obohatiť, aby model mohol lepšie pochopiť kontext a bezpečnostný stav testovanej aplikácie. Štruktúra výsledku statickej analýzy framework-u MobSF vyzerá nasledovne:

```
{
    'dir': String,
    'app_name': String,
    'md5': String,
    'app_dir': String,
    'tools_dir': String,
    'icon_path': String,
    'app_file': String,
    'app_path': String,
    'manifest_file': String,
    'real_name': String,
```

```
    'files': [String]
}
```

Výpis 3: Štruktúra výstupu statickej analýzy pomocou MobSF

Aj keď MobSF vykoná veľkú časť analytickej práce a zozbiera niektoré potrebné dáta, pre naše potreby je to však nedostatočné, keďže systém nebol navrhnutý s úmyslom používať výsledky statickej analýzy ako kontext pre jazykové modely.

Pre naše potreby je nutné doplniť množstvo funkcionality spracovávajúcej dáta, ako napríklad extrahovanie definícií exportovaných komponentov a ich zdrojového kódu.

V rámci nášho systému je taktiež vhodné navrhnúť vlastné pravidlá statickej analýzy, ktoré zohľadňujú fakt, že na základe ich detekcie bude následne problém prezentovaný jazykovému modelu.

Riešenie MobSF neimplementuje analýzu súborového systému, rovnako neimplementuje analýzu logov zozbieraných počas chodu aplikácie. V rámci nášho riešenia predstavíme funkcionalitu, ktorá rieši aj problémy analýzy súborov, extrahovanie ich metadát a predspracovanie veľkého objemu dát, kvôli limitácii kontextového okna použitého modelu.

4.3 Doplnená funkcionalita

V tejto podkapitole analyzujeme postup práce s vybranými technológiami, spôsob ich aplikácie, implementácie a prístup, ktorým sme prostredníctvom ich využitia dosiahli ciele a realizovali koncepty predstavené v návrhovej časti tejto práce.

4.3.1 Hodnotenie bezpečnosti exportovaných komponentov

Ako jeden z hlavných bodov záujmu pri hodnotení bezpečnosti mobilnej aplikácie je práve analýza exportovaných komponentov. Pojem “exportovaný komponent“ môžeme chápať ako časť aplikácie, ktorá je prístupná aj ostatným aplikáciám nainštalovaných na zariadení alebo prístupná aj pre operačný systém zariadenia, či už na komunikáciu alebo sledovanie systémových eventov.

S pravidla ide o nasledovné kategórie komponentov mobilnej aplikácie:

- Activity
- Service
- Broadcast Receiver
- Content Provider

Informácie o tom, či je komponent exportovaný, aký vstup vie spracovať a jeho celková definícia je zapísaná v súbore *AndroidManifest.xml* nachádzajúcom sa v APK balíku mobilnej aplikácie.

Pre potreby analýzy exportovaných komponentov sme preto vytvorili štruktúru podľa návrhovej časti tejto práce, ktorá zachytáva nasledovné údaje:

```
{
    'activity':{'component_name':'definition', ...},
    'receiver':{'component_name':'definition', ...},
    'provider':{'component_name':'definition', ...},
    'service' :{'component_name':'definition', ...}
}
```

Výpis 4: Štruktúra na zbieranie informácií o exportovaných komponentoch

Pre každý explicitne alebo implicitne exportovaný komponent v mobilnej aplikácii, extrahujeme jeho definíciu zo súboru AndroidManifest.xml, ktorá poslúži ako kontext do pripravenej šablóny požiadavky pre model (obrázok 10).

Rovnako ako definíciu pre zhodnotenie bezpečnosti komponentu, potrebujeme analyzovať aj jeho zdrojový kód. Z definície a teda z pripravenej štruktúry vo výpise 4 dokážeme nájsť konkrétne Java súbory prislúchajúce jednotlivým komponentom a tak doplniť kompletný obraz používania komponentu. Takto nazbierané údaje sú následne prezentované modelu vo formáte uvedenom vo výpise 2 návrhovej časti.

4.3.2 Systém pravidiel a analýza zdrojového kódu

Exportované komponenty prezentované v predchádzajúcej kapitole však nie sú jediným bodom pre možné zneužitie alebo útok na mobilnú aplikáciu.

Kvôli limitácii veľkosti kontextového okna popísanej v analytickej časti práce, nie je možné zdrojový kód modelu prezentovať ako celok. Z tohto dôvodu je vhodné pokúsiť sa nájsť v zdrojovom kóde body, ktoré by mohli predstavovať potenciálne zraniteľné miesta a zúžiť tým prehľadávaný priestor a objem dát posielaný v požiadavkách pre model.

Naše riešenie preto implementuje systém semgrep pravidiel. Tento systém nám poskytuje možnosť kustomizácie a konfigurácie pravidiel podľa potreby a dosahovaných výsledkov. Ako príklad pre predstavenie tejto funkcionality uvedieme pravidlo, ktoré má za úlohu identifikovať miesta tvorby a použitia SQL dotazov (všetky navrhnuté a testované pravidlá sú uvedené v prílohe C tejto práce). Nasledujúci výpis znázorňuje definíciu takého semgrep pravidla:

```
- id: java-sql-usage-advanced
  languages: [java]
```

```
message: "Potential SQL query usage, check for SQL injection vulnerabilities."
severity: WARNING
metadata:
  owasp-mobile: M4: Insufficient Input/Output Validation
  category: M4
  llmchain: vuln
  technology: android
pattern-either:
  - pattern: "$DB.rawQuery(...);"
  - pattern: "$DB.query(...);"
  - pattern: "$DB.prepareStatement(...);"
  - pattern: "$DB.Query(...);"
```

Výpis 5: Semgrep pravidlo vytvorené pre detekciu používania SQL dotazov

Akonáhle pravidlo počas skenovania zaznamená nález v zdrojovom kóde, bude táto skutočnosť prezentovaná AI modelu s cieľom preveriť označenú funkcionalitu.

Pre zvýšenie efektivity a pravdepodobnosti odhaliť chybu, je nutné ako súčasť pravidla uviesť úlohu, ktorú má model pri analýze plniť. Vo výpise 5 môžeme takúto definíciu úlohy vidieť v parametri "message". Tento parameter by mal zohľadňovať ciel samotného pravidla a teda akú podozrivú funkcionalitu si v zdrojovom kóde želáme preveriť.

Výsledok tohto procesu je ohodnotený nález, znázornený na obrázku 14.

java-sql-usage-advanced	Potential SQL query usage, check for SQL injection vulnerabilities.	allsafe/challenges/SQLInjection.java	Vulnerable	Potential SQL injection vulnerability due to concatenation of user input in SQL query	sb.append("select * from user where username = '"); sb.append(String.valueOf(username.getText())); sb.append("' and password = '"); md5 = SQLInjection.this.md5(String.valueOf(password.getText())); sb.append(md5); sb.append("'"); Cursor cursor = sQLiteDatabase.rawQuery(sb.toString(), null);

Obr. 14: Výsledok analýzy zdrojového kódu, vybraného semgrep pravidlom

Náš systém označil súbor *allsafe/challenges/SQLInjection.java*, na základe pravidla vo výpise 5 ako potenciálne zraniteľný a vhodný pre kontrolu pomocou LLM. Po prezentovaní problému, dostávame výsledok, ktorý zahŕňa hodnotu reprezentujúcu či je prezentovaný zdrojový kód zraniteľný. Rovnako od modelu požadujeme aj popis zraniteľnosti a časť zraniteľného zdrojového kódu, kde sa zraniteľnosť vyskytuje.

Rovnakým spôsobom je možné definovať ľubovoľný počet pravidiel týkajúcich sa rôznych zraniteľností a častí kódu pre maximalizáciu pokrytia funkcionality mobilnej aplikácie.

Ako samostatnú kategóriu pri analýze pomocou systému pravidiel náš systém implementuje tiež analýzu logovaných údajov s cieľom odhaliť logovanie senzitívnych informácií. Pravidlo s identifikátorom *detect-logging* v prílohe C má za úlohu detegovať časti kódu, v ktorých aplikácia používa logovanie. Model má následne za úlohu analyzovať logované

premenné, ich pôvod a obsah, s cieľom odhaliť únik citlivých dát pomocou logov. Na obrázku 15 vidíme detekciu logovania senzitívnych údajov, kde model vyhodnotil označený kód ako zraniteľný.

Obr. 15: Detekcia logovania senzitívnych údajov na základe semgrep pravidla

Tento prístup hodnotenia logov však ukázal, že v niektorých prípadoch využívajú mobilné aplikácie na logovanie aj treťo-stranové knižnice (napríklad knižnica Timber[23]). Preto je nutné toto pravidlo neustále aktualizovať podľa charkteru skenovanej aplikácie a aktívne dopĺňať zoznam logovacích knižníc.

Tento prístup je teda náchylný na chyby a je možné, že systému sa nepodarí detegovať všetky miesta logovania. Možné riešenie problému popisujeme v nasledujúcej kapitole, zameranej na analýzu logov zozbieraných počas behu aplikácie.

4.3.3 Analýza "runtime" logov

V predchádzajúcej kapitole sme popísali možné limitácie pri hodnotení úniku potenciálne senzitívnych údajov iba na základe zdrojového kódu mobilnej aplikácie. Preto naše riešenie implementuje funkcionalitu analýzy "runtime" logov. Tieto logy vznikajú počas behu aplikácie a je možné ich zaznamenať pomocou nástroja ADB - terminálový príkaz *adb logcat –pid=<app_pid>*. Pre zozbieranie týchto logov je nutné manuálne prejsť celou funkcionalitou mobilnej aplikácie a vyplniť všetky polia testovacími údajmi. Po tomto procese máme k dispozícii súbor logov, ktoré aplikácia generuje pri využívaní jej funkcionality a pri práci s používateľským vstupom.

Úlohou nášho systému je následne pomocou LLM analyzovať súbor logov s cieľom odhaliť únik senzitívnych údajov cez logcat.

Podľa rozsahu a charakteru aplikácie, vstupný súbor pre model môže obsahovať aj tisícky záznamov, pričom tieto záznamy pre nás vo väčšine prípadov nepredstavujú žiadnu pridanú hodnotu. V ukážke nižšie je znázornených niekoľko záznamov logovaných aplikáciou InsecureShop počas jej používania:

```
...
1 02-17 20:57:20.629  4070  4090 D HostConnection: HostComposition ext
```

[23]https://github.com/JakeWharton/timber

```
  ANDROID_EMU_CHECKSUM_HELPER_v1 ANDROID_EMU_sync_buffer_data
  GL_OES_EGL_image_external_essl3 GL_OES_vertex_array_object
  GL_KHR_texture_compression_astc_ldr ANDROID_EMU_host_side_tracing
  ANDROID_EMU_gles_max_version_3_0
2 02-17 20:57:20.631  4070  4090 D EGL_emulation: eglMakeCurrent:
  0x7b44a3f020: ver 3 0 (tinfo 0x7b44a9c120)
3 02-17 20:57:20.632  4070  4090 D eglCodecCommon: setVertexArrayObject:
  set vao to 0 (0) 1 2
4 02-17 20:57:21.978  4070  4070 I AssistStructure: Flattened final
  assist data: 2728 bytes, containing 1 windows, 9 views
5 02-17 20:57:24.988  4070  4070 I AssistStructure: Flattened final
  assist data: 2736 bytes, containing 1 windows, 9 views
6 02-17 20:57:27.014  4070  4070 D userName: testUser
7 02-17 20:57:27.014  4070  4070 D password: testPassword!
8 02-17 20:57:27.081  4070  4090 D EGL_emulation: eglMakeCurrent:
  0x7b44a3f020: ver 3 0 (tinfo 0x7b44a9c120)
9 02-17 20:57:27.085  4070  4090 D EGL_emulation: eglMakeCurrent:
  0x7b44a3f020: ver 3 0 (tinfo 0x7b44a9c120)
...
```

Výpis 6: Ukážka logovaných záznamov aplikácie InsecureShop

Ako môžeme vidieť vo výpise 6, v záznamoch sa nachádza veľa údajov nesúvisiacich s riešeným problémom, ako napríklad časová známka na začiatku záznamov, PID aplikácie, duplikované záznamy a ďalšie záznamy o bežných eventoch, ktoré nepredstavujú bezpečnostné riziko. V logoch sa tiež môžu vyskytovať záznamy, ktoré nie sú priamo duplikáty ale líšia sa len v malom počte znakov - záznam číslo 4 a číslo 5 vo výpise 6 (rozdielne číslo bytov). Preto sme sa rozhodli pomocou metódy počítania Levenshteinovej vzdialenosti[24] odstrániť aj záznamy, s podobnosťou väčšou ako 90%.

Na tento účel implementuje naše riešenie nasledovnú funkciu pre odhalenie podobnosti záznamov:

```
def are_similar(str1, str2, threshold=0.9):
    #Kontrola podobnosti 2 retazcov, nad definovanu uroven
    distance = lev.distance(str1, str2)
    max_len = max(len(str1), len(str2))
    similarity = 1 - (distance / max_len)
    return similarity > threshold
```

Výpis 7: Funkcia pre detekciu podobnosti záznamov

[24]https://en.wikipedia.org/wiki/Levenshtein_distance/

Keďže údaje ako časová známka alebo PID procesu nie sú v riešení nášho problému nijak nápomocné, systém záznamy očistí aj o tieto údaje pomocou nasledovnej funkcionality:

```
...
    for line in file:
        parts = line.split()  # rozdelit zaznam do stlpcov
        if len(parts) > 4:
            new_line = ' '.join(parts[4:])  # vynechanie nepotrebnych udajov
            processed_lines.add(new_line + '\n')  # filtrovany zaznam
...
```

Výpis 8: Filtrovanie záznamov

Po filtrovaní prvej ukážky (výpis 6) záznamov pomocou uvedených filtrov a metód by naše záznamy vyzerali nasledovne:

```
...
1 D HostConnection: HostComposition ext ANDROID_EMU_CHECKSUM_HELPER_v1
  ANDROID_EMU_sync_buffer_data GL_OES_EGL_image_external_essl3
  GL_OES_vertex_array_object GL_KHR_texture_compression_astc_ldr
  ANDROID_EMU_host_side_tracing ANDROID_EMU_gles_max_version_3_0
2 D eglCodecCommon: setVertexArrayObject: set vao to 0 (0) 1 2
3 I AssistStructure: Flattened final assist data: 2728 bytes,
  containing 1 windows, 9 views
4 D userName: testUser
5 D password: testPassword!
6 D EGL_emulation: eglMakeCurrent: 0x7b44a3f020: ver 3 0 (tinfo
  0x7b44a9c120)
...
```

Výpis 9: Záznamy po filtrovaní

Ako môžeme vidieť, redundantných informácií je v záznamoch podstatne menej a je jednoduchšie identifikovať vyhľadávané údaje. V prípade aplikácie InsecureShop sa touto metódou podarilo zredukovať skenovaný obsah až o 80%. Takáto úprava vstupu má za následok, že model požiadavku dokáže spracovať rýchlejšie, efektívnejšie, a nakoľko API volanie obsahuje výrazne menší počet tokenov, vieme tak znížiť aj poplatky spojené s používaním modelu.

4.3.4 Analýza súborového systému

V prípade kontroly súborového systému aplikácie s cieľom odhaliť neadekvátne používanie alebo ukladanie senzitívnych údajov je potrebné exportovať interný aj externý

súborový systém (sandbox) mobilnej aplikácie. V tomto súborovom systéme aplikácie sa však z pravidla nachádzajú súbory rôznych typov a preto je nutné z nich extrahovať iba reťazce a metadáta o súbore pre upresnenie kontextu v komunikácii s modelom.

Pre získanie požadovaných dát zo súborov náš systém implementuje dva nástroje. Extrahovanie reťazcov so súboru je vykonané pomocou nástroja "strings". V rovnakom procese získame aj metadáta súboru ako dodatočný kontext pre model, ktorý môže pomôcť pri hodnotení citlivosti údajov. Metadáta súboru získame požitím nástroja "file". Nasledujúca ukážka reprezentuje zbieranie týchto hodnôt do štruktúry definovanej vo výpise 11.

```
...
    results = {}
    results['file_path'] = file_path
    # Volanie nastroja strings pre ziskanie retazcov suboru
    strings = subprocess.run(['strings', '-n', str(min_length), file_path],
                             capture_output=True, text=True)
    results['file_strings'] = strings.stdout

    # Volanie nastroja file pre ziskanie metadat
    metadata = subprocess.run(['file', file_path],
                              stdout=subprocess.PIPE,
                               stderr=subprocess.PIPE,
                               text=True)
    # Kontrola uspesnosti
    if metadata.returncode != 0:
        print(f"Error: {metadata.stderr.strip()}")

    results['file_metadata'] = metadata.stdout
    return results
...
```

Výpis 10: Získavanie údajov pre hodnotenie súborového systému mobilnej aplikácie

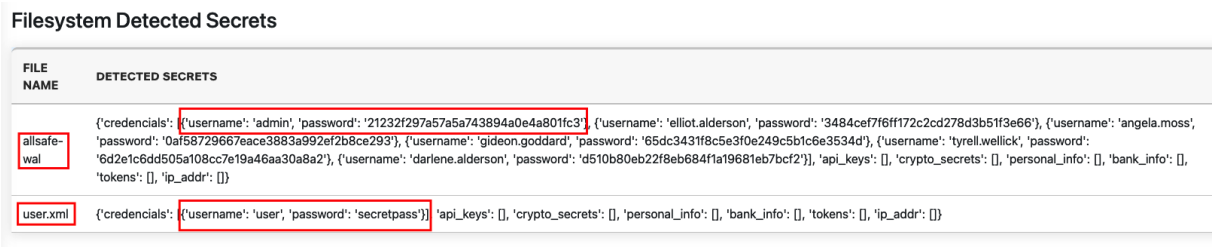
Výsledná štruktúra teda obsahuje všetky potrebné hodnoty pre zhodnotenie bezpečnosti spracovania citlivých údajov v aplikácii spojenej s využívaním súborového systému. Výsledky sú dostupné vo forme štruktúry, ktorú môžeme vidieť vo výpise 11.

```
{
    "file_name":{
        "file_path": String
        "file_strings": String
        "file_metadata": String
        },
```

```
    ...
}
```

Výpis 11: Štruktúra pre metadáta a reťazce extrahované so súborového systému aplikácie

Po finálnej kontrole pomocou LLM máme k dispozícii výsledok analýzy, ktorý v prípade testovanej aplikácie AllSafe vyzerá nasledovne:



Obr. 16: Výsledok analýzy súborového systému aplikácie pomocou LLM

Z výsledkov môžeme vidieť, že model identifikoval niekoľko záznamov z databázového súboru *allsafe-wal* a taktiež sa nám podarilo odhaliť nešifrované prihlasovacie údaje používateľa v súbore *user.xml*.

4.3.5 Komunikácia s modelom

Po rozšírení kontextu a profilu aplikácie popísanom v predchádzajúcich kapitolách, máme k dispozícii väčšinu potenciálne zaujímavých dát pre hodnotenie bezpečnosti aplikácie.

V tomto kroku je potrebné modelu prezentovať riešený problém. Knižnica LangChain popísaná v kapitole 3.7.5 nám poskytuje rozhranie, vďaka ktorému vieme efektívnejšie pracovať so šablónami požiadaviek a tiež pripraveným kontextom.

Táto knižnica umožňuje dynamickú interakciu s LLM, čo zvyšuje efektivitu pri konfigurácii a komunikácii s modelom. S pomocou LangChain môžeme detailne špecifikovať vstupné požiadavky a očakávané výstupy, čo nám pomáha maximalizovať presnosť a relevantnosť výsledkov poskytovaných modelom.

Komunikácia s LLM sa potom stáva viac zameranou a kontextualizovanou. S pripravenými šablónami môžeme predkladať modelu vstupné dáta vo formáte zohľadňujúcom nielen obsah a bezpečnostné aspekty aplikácie, ale aj konkrétne parametre a špecifiká analyzovanej situácie. Napríklad ak model identifikuje potenciálnu zraniteľnosť, môžeme pomocou LangChain formulovať naväzujúce otázky, ktoré preskúmajú a objasnia možné dôsledky a kontext tejto zraniteľnosti. Tento proces nie len zvyšuje efektivitu našej analýzy, ale tiež pridáva ďalšiu vrstvu hĺbkového pochopenia a interpretácie bezpečnostných rizík pre jednoduchšie odhalenie problému pomocou LLM.

Dôležitým aspektom v komunikácii je tiež filtrovanie redundantných a nepotrebných dát. Tie môžu mať za následok zhoršenie úspešnosti modelu ale tiež zvýšenie poplatku za prevedenú analýzu. Príklad takejto situácie je popísaný v kapitole "4.3.3 Analýza runtime logov". Model tu má za úlohu odhaliť potenciálne senzitívne údaje, heslá, osobné údaje a podobne.

Nasledujúca ukážka znázorňuje využitie knižnice LangChain pre dodanie kontextu a inštruovanie modelu pri evaluácii časti kódu označenej pomocou semgrep pravidal, ako je popísané v kapitole 4.3.2.

```
# Inicializacia OpenAI modelu
model = ChatOpenAI(openai_api_key=OPENAI_API_KEY, model=model_conf, temperature=
temperature_conf)

# Definicia struktury vystupu
parser = JsonOutputParser(pydantic_object=Finding)

template = "
    Using Semgrep rule, we identified a possible vulnerable code. Your task is to
verify the \n\
    code provided, if the following issue is present in the code. \n\
    {format_instructions}\n\n\
    Issue to verify: \n\
    {rule_message}\n\n\
    Enumerate the following app source code, does the reported issue introduces a
security\n\
    risk ? \
    {file_content}"

# Doplnenie poziadavky pre model
prompt = PromptTemplate(
        template=template,
        input_variables=["rule_message, file_content"],
        partial_variables={"format_instructions": parser.get_format_instructions()
})

# Definicia retaze operacii
chain = prompt | model | parser
# Vykonanie poziadavky s nasim kontextom
response = chain.invoke({"rule_message": rule.extra.message,
                         "file_content":file_content})
```

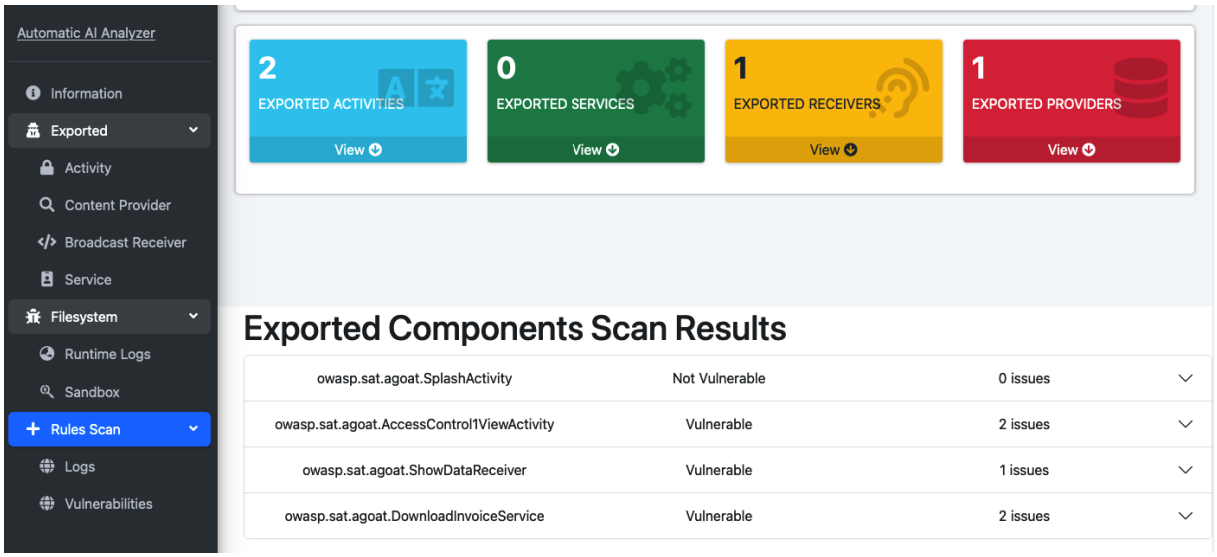
Výpis 12: Komunikácia s modelom pomocou LangChain

Pred odoslaním požiadavky na analýzu pomocou LLM sú do modelu doplnené konkrétne údaje definujúce skúmaný problém v testovanej aplikácii. Rovnakým spôsobom sú implementované aj ostatné časti nášho riešenia, týkajúce sa komunikácie s modelom.

4.3.6 Reprezentácia výsledkov

Pre potreby prezentácie výsledkov sme v rámci nášho riešenia implementovali vlastný webový dashboard, ktorý figuruje ako samostatná časť framework-u MobSF. Ten umožňuje jednoduchší prístup a interpretáciu dát rozčlenených do špecifických kategórií zobrazených na obrázku 17 (menu vľavo). Vybrané kategórie pozostávajú z jednotlivých podúloh, ktoré model počas analýzy rieši a hodnotí stav bezpečnosti aplikácie na základe predloženého kontextu. Prezentovaný panel má za úlohu zobraziť celkový bezpečnostný profil skúmanej aplikácie a prezentovať používateľovi nájdené zraniteľnosti alebo možné bezpečnostné riziká.

Pre dodržanie dizajnu a používateľskej skúsenosti sme dashboard implementovali podľa existujúcej šablóny framework-u MobSF. Dashboard však zobrazuje dáta získané iba za pomoci nášho riešenia a zaoberá sa iba výsledkami analýzy pomocou LLM. Všetky sekcie v tejto kapitole sú teda súčasťou nášho systému. Vo výslednom riešení je prezentovaný dashboard oddelený od pôvodnej funkcionality MobSF a jedná sa o rozšírenie doplnené pre potreby prezentácie výsledkov nášej analýzy.



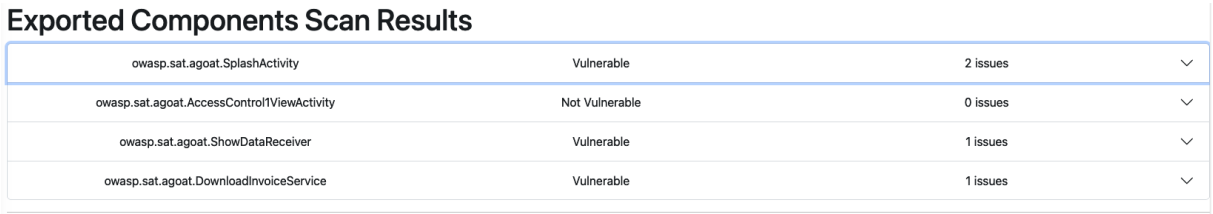
Obr. 17: Panel pre zobrazenie výsledkov analýzy

V sekcii "Information" (obrázok 18) prezentujeme základné údaje o aplikácii podrobenej testovaniu. Medzi tieto informácie patria meno nahraného APK súboru, jeho veľkosť, ako aj rôzne typy hash kódov, ktoré poskytujú prehľad o verziách skenovaného súboru. V

rámci tohto panelu sa tiež nachádza interaktívne tlačidlo “Start AI Scan“ inicializujúce proces automatizovanej analýzy využívajúcej LLM. Po aktivácii a úspešnom dokončení skenu sa doplnia ostatné segmenty užívateľského rozhrania s výstupmi a nálezmi získanými z LLM.

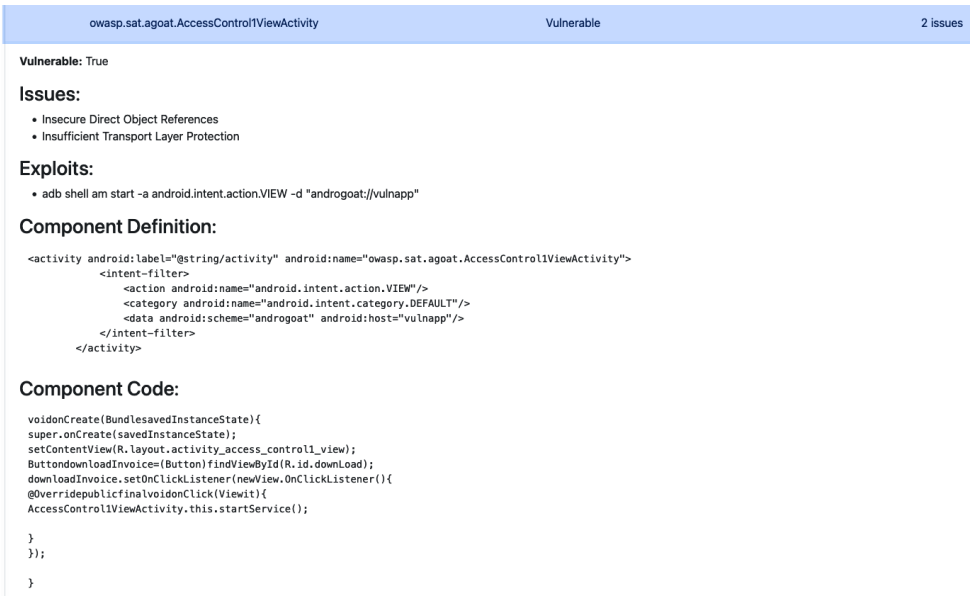
Obr. 18: Information - informácie o testovanej aplikácii

V sekcii zobrazenej na obrázku 19 prezentujeme hodnotenie jednotlivých komponentov. Po analýze exportovaných komponentov s cieľom odhaliť možné zraniteľnosti, najmä niektoré z kategórií OWASP Mobile TOP-10 popísaných v kapitole 2.3.1, sú výsledky zaznamenané do tabuľky spolu s detailom skúmaného komponentu, ktorý bližšie popisuje skúmané parametre komponentu.



Obr. 19: Exported - hodnotenie bezpečnosti exportovaných komponentov mobilnej aplikácie

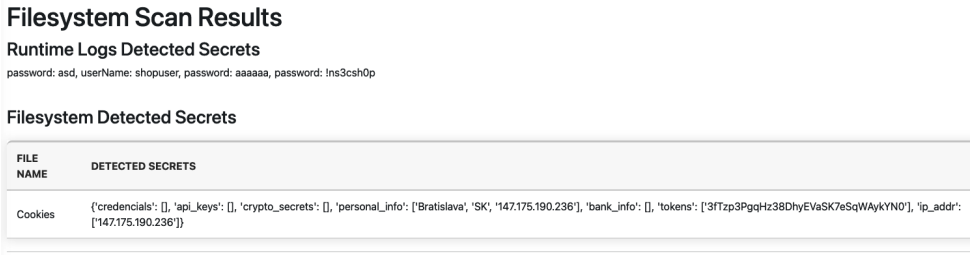
V detaile komponentu (obrázok 20) môžeme nájsť hodnotenie úrovne bezpečnosti, nájdené zraniteľnosti, možné exploit-y pre nájdené zraniteľnosti a tiež definíciu a časť zdrojového kódu hodnoteného komponentu.



Obr. 20: Exported - detail hodnotenia vybraného komponentu

Ďalšia sekcia panelu sa venuje vyhodnocovaniu senzitívnych údajov na internom aj externom súborovom systéme aplikácie a tiež logom zozbieraným počas behu a používania aplikácie. Kapitola 4.3.3 popisuje spracovanie Runtime logov, ktoré sú ďalej analyzované a príklad výsledku tejto analýzy môžeme vidieť na obrázku 21 - sekcia “Runtime Logs Detected Secrets“.

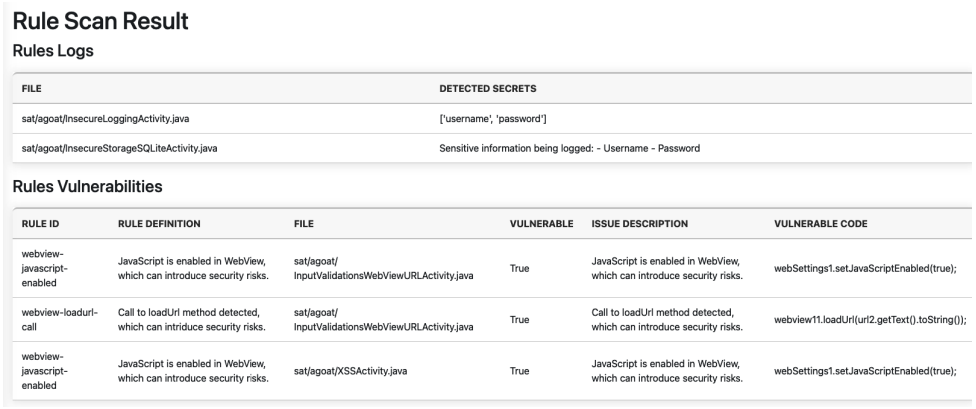
V prípade súborového systému (časť “Filesystem Detected Secrets“) má LLM podobnú úlohu ale, požiadavky pre model sú obohatené o metadáta skúmaného súboru, ako opisujeme v kapitole “4.3.4 Analýza súborového systému“.



Obr. 21: Filesystem - identifikácia potenciálne senzitívnych údajov

Posledná sekcia sa zaoberá výsledkami konfigurovateľných testov. V sekcii zobrazenej na obrázku 22 sa nachádzajú vyhodnotené časti kódu, ktoré boli vybrané na základe semgrep pravidiel. Systém má za úlohu zhodnotiť funkcie alebo metódy náchylné na zraniteľnosti. V týchto častiach kódu sa pomerne často používajú vstupy zadávané používateľom a preto je dôležité dbať na ich správne spracovanie a sanitizáciu. Takéto časti kódu

je možné identifikovať a poskytnúť modelu na vyhodnotenie bezpečnosti použitia danej funkcie.



Rule Scan Result

Rules Logs

FILE	DETECTED SECRETS
sat/agoat/InsecureLoggingActivity.java	['username', 'password']
sat/agoat/InsecureStorageSQLiteActivity.java	Sensitive information being logged: - Username - Password

Rules Vulnerabilities

RULE ID	RULE DEFINITION	FILE	VULNERABLE	ISSUE DESCRIPTION	VULNERABLE CODE
webview-javascript-enabled	JavaScript is enabled in WebView, which can introduce security risks.	sat/agoat/InputValidationsWebViewURLActivity.java	True	JavaScript is enabled in WebView, which can introduce security risks.	webSettings1.setJavaScriptEnabled(true);
webview-loadurl-call	Call to loadUrl method detected, which can intriduce security risks.	sat/agoat/InputValidationsWebViewURLActivity.java	True	Call to loadUrl method detected, which can introduce security risks.	webview11.loadUrl(url2.getText().toString());
webview-javascript-enabled	JavaScript is enabled in WebView, which can introduce security risks.	sat/agoat/XSSActivity.java	True	JavaScript is enabled in WebView, which can introduce security risks.	webSettings1.setJavaScriptEnabled(true);

Obr. 22: Rules - výsledky analýzy s pomocou semgrep pravidiel

5 Testovanie a výsledky

Po dokončení fázy implementácie nášho riešenia sa v tejto kapitole venujeme skúmaniu procesu testovania a hodnotenia dosiahnutých výsledkov. V rámci testovacieho procesu sme využili viaceré veľké jazykové modely pre porovnanie výsledkov a vyhodnotenie kvality modelov v danej problematike. V nasledujúcich odsekoch popíšeme metodológiu prevedenia testov a predstavíme nástroje, ktoré boli kľúčové pre tento proces. Taktiež sa budeme venovať konkrétnym mobilným aplikáciám použitým na testovanie a analýzu, ako bolo spomenuté v kapitole 4.1. V tejto etape hodnotíme nie len správnosť a presnosť našich modelov v kontexte bezpečnostnej analýzy, ale tiež efektivitu a praktickú aplikovateľnosť riešenia v reálnych podmienkach.

5.1 Vyhodnotenie výsledkov modelu

Ako súčasť nášho výskumného procesu sme vykonali rozsiahle testovanie, kde sú výsledky analýzy vyhodnotené na základe automatizovaného testovania vybraných mobilných aplikácií uvedených v kapitole 4.1.

5.1.1 Parametre testovacieho prostredia

Testovanie nášho riešenia bolo vykonané na zariadení popísanom v nasledujúcej tabuľke. Keďže model nie je implementovaný lokálne a naše riešenie využíva API rozhranie poskytnuté tretími stranami, väčšina výpočtovej záťaže je delegovaná na prevádzkovateľa modelu.

Tabuľka 3: Parametre testovacieho zariadenia

Komponent	**Parametre**
Zariadenie	MacBook Pro 2022
Procesor	Apple M2
Operačná pamäť	16GB
Operačný systém	Sonoma 14.4.1

Medzi výpočtovo najnáročnejšie úkony, počítané lokálne patrí napríklad prehľadávanie zdrojového kódu pomocou semgrep pravidiel. Táto náročnosť je závislá od veľkosti prehľadávaného zdrojového kódu a taktiež počtu implementovaných semgrep pravidiel. Pri testovaní nášho riešenia sme však s použitím popísaného zariadenie nenarazili na limitujúce prekážky.

5.1.2 Metodológia hodnotenia

Primárnym cieľom hodnotenia je určiť mieru úspešnosti modelu v identifikácii skutočných bezpečnostných hrozieb, teda tzv. true-positive detekcií. Zároveň je však dôležité rozpoznať a zaznamenať aj prípady false-positive detekcií, teda situácie, keď model nesprávne označil bezpečnú súčasť aplikácie ako zraniteľnú.

V rámci vyhodnotenia sme sa tiež zamerali na analýzu schopnosti modelu detegovať senzitívne údaje a ako presne dokáže model identifikovať potenciálne rizikové oblasti v rozličných častiach a funkcionalitách mobilných aplikácií.

Keďže hodnotenie celej aplikácie je pomerne komplexný problém, ako príklad uvedieme hodnotenie bezpečnosti Andoid aplikácie *Allsafe*[25]. Tá obsahuje niekoľko známych a zdokumentovaných zraniteľností, na ktorých sa pokúsime demonštrovať úspešnosť analýzy pomocou LLM riešenia.

Aplikácia obsahuje zraniteľnosti popísané v tabuľke číslo 4, avšak nie všetky zraniteľnosti sú cieľom našej analýzy, nakoľko niektoré z nich sa týkajú dynamickej analýzy, prípadne manuálnej úpravy APK súboru, na čo nie je náš systém navrhnutý. V tabuľke sú znázornené typy zraniteľností, ktoré dokáže náš systém detegovať. Očakávame, že systém dokáže odhaliť všetky zraniteľnosti označené ako “In Scope“. Zraniteľnosti označené ako “Out of Scope“ nie sú zahrnuté do nášho systému.

Na základe tabuľky 4 sme vytvorili zoznamy známych zraniteľností obsiahnutých v testovaných aplikáciách (príloha D). Evaluácia úspešnosti detekcií sa teda riadi vyhodnotením schopnosti odhaliť zraniteľnosti v priložených zoznamoch.

Tabuľka 5 nám poskytuje prehľad o efektivite našej platformy a schopnostiach použitého modelu pri testovaní vzorovej Android aplikácie *AllSafe*. Aby sme zabezpečili kvalitnejšie meranie výkonnosti nášho systému a minimalizovali možnosť výkyvov v dátach, testovacia analýza bola opakovaná päťkrát. Z jednotlivých testovaní bola následne derivovaná celková miera úspešnosti našich nástrojov pri hodnotení bezpečnosti testovanej aplikácie. Tento postup nám umožnil validovať konzistenciu našich detekčných metód, ale aj určiť priemernú úspešnosť a spoľahlivosť nášho riešenia. Tento vzorový test je prevedený pomocou modelu *gpt-3.5-turbo*[26].

[25]https://github.com/t0thkr1s/allsafe

[26]https://platform.openai.com/docs/models/gpt-3-5-turbo

Tabuľka 4: Prehľad zraniteľností, ktoré rieši náš systém

Vulnerability Type	**Scope**
Insecure Logging	In Scope
Insecure Shared Preferences	In Scope
Hardcoded Credentials	In Scope
Root Detection	Out of Scope
Arbitrary Code Execution	In Scope
Secure Flag Bypass	Out of Scope
Certificate Pinning Bypass	Out of Scope
Insecure Broadcast Receiver	In Scope
Insecure Service	In Scope
Insecure Content Provider	In Scope
Insecure Activity	In Scope
Deep Link Exploitation	In Scope
SQL Injection	In Scope
Vulnerable WebView	In Scope
Smali Patching	Out of Scope
Native Libraries	Out of Scope

Tabuľka 5: Výsledky úspešnosti automatizovanej analýzy mobilnej aplikácie AllSafe (5 testov)

AllSafe application analysis results (gpt-3.5-turbo)	
Type	# of detections
false-positive	24
true-positive	40/55

Zo zaznamenaných výsledkov môžeme usúdiť, že model mal úspešnosť detekcie zraniteľnosti približne **72%**. Pomerne vysoká je však aj miera false-positive detekcií, kde ide o prípady, kedy model uviedol zraniteľnosti aj na miestach, kde sa nenachádzali.

V nasledujúcej tabuľke (tabuľka číslo 6) uvádzame úspešnosti analýzy všetkých vybraných aplikácií z kapitoly 4.1 pomocou rovnakého modelu *gtp-3.5-turbo*. V tomto prípade boli všetky uvedené aplikácie testované trikrát.

Tabuľka 6: Výsledky úspešnosti automatizovanej analýzy uvedených aplikácií (3 testy / aplikácia)

Selected applications analysis results (gpt-3.5-turbo)			
Application	False-positive detections	True-positive detections	Success rate
AllSafe	19	25/33	75%
Insecureshop	4	15/39	38%
AndroGoat	4	23/39	59%
MASTG Playground	2	20/27	74%
InjuredAndroid	16	6/15	40%
InsecureBankV2	24	18/24	75%
Damn Vulnerable Bank*	6	10/21	48%

*mobilná aplikácia "Damn Vulnerable Bank" bola na rozdiel od ostatných testovaných aplikácií obfuskovaná. Obfuskácia zdrojového kódu má vplyv na úspešnosť výsledkov analýzy nakoľko je kód menej prehľadní a zrozumiteľný aj pre samotný LLM.

Podľa výsledkov v tabuľke vyššie môžeme vidieť, že v niektorých prípadoch model dosiahol úspešnosť 40%, naopak v niektorých úspešnosť dosiahla približne 75%. Keďže testované aplikácie neboli navrhnuté pre účely testovania úspešnosti automatizovanej analýzy, všetky výsledky boli kvôli komplexnosti kontrolované a zaznamenávané manuálne (zoznam "In Scope" zraniteľností v prílohe D). Rozdelenie a kategórie zraniteľností v testovaných aplikáciách sa líšia, preto nebolo možné proces testovania automatizovať.

Naše riešenie dosiahlo s použitím modelu *gpr-3.5-turbo* približnú priemernú úspešnosť **60%** v detekcii vybraných zraniteľností z oblasti penetračného testovania Android mobilných aplikácií.

5.2 Porovnanie úspešnosti modelov

Ako sme mohli vidieť v predchádzajúcej podkapitole, výsledky úspešnosti pre model *gpt-3.5-turbo* sme zaznamenali v rozmedzí 40 až 75 %. V tejto kapitole porovnáme výsledky úspešnosti rôznych dostupných modelov. Cieľom je zistiť, či niektoré z modelov zvládajú požadovanú analýzu a úlohy lepšie ako iné. Taktiež porovnáme mieru generovaných falošných (false-positive) detekcií pri jednotlivých modeloch.

Testovanie bolo prevedené rovnakým spôsobom ako v kapitole 5.1.2, s tým rozdielom, že v tomto prípade použité modely hodnotili bezpečnostný stav siedmich vybraných aplikácií iba jedenkrát (jeden sken na aplikáciu). Priemerná úspešnosť a miera false-positive detekcií pre jednotlivé modely je teda vypočítaná ako priemer výsledkov zo siedmych rôznych analýz.

Tabuľka 7: Porovnanie výsledkov rôznych modelov použitých pre analýzu aplikácií (1 test / aplikácia)

Selected applications analysis results per model			
Model	False-positive detections	True-positive detections	Success rate
gpt-3.5-turbo-0125	23	39/66	60%
gpt-4-turbo	15	48/66	73%
claude-3-haiku-20240307	22	39/66	60%
claude-3-opus-20240229	16	45/66	68%

Tabuľka číslo 7 sumarizuje výsledky analýzy vybraných aplikácií s využitím rôznych jazykových modelov. Výsledky poukazujú na rôzne úrovne efektivity modelov. Najvyššiu úspešnosť dosiahli modely *gpt-4-turbo*[27] a *claude-3-opus-20240229*[28], ktoré sú podľa benchmark-ov z kapitoly 1, v čase písania tejto práce jedny z najvýkonnejších modelov dostupných na trhu. Z výsledkov môžeme pozorovať, že aj keď sa jedná o najvýkonnejšie modely, úspešnosť detekcií výrazne nezvýšili. Avšak detekcia false-positive nálezov bola v prípade týchto modelov výrazne nižšia a výsledné detekcie boli kvalitnejšie odôvodnené.

Modely *gpt-3.5-turbo-0125* a *claude-3-haiku-20240307* vykázali nižšiu úspešnosť, oba s hodnotou 60%, pričom identifikovali 39 správnych prípadov, ale s vyšším počtom false-positive detekcií.

V niektorých prípadoch sme sa pri modely haiku stretli s problémom, kde model ne-

[27] https://platform.openai.com/docs/models/gpt-4-turbo-and-gpt-4
[28] https://www.anthropic.com/news/claude-3-family

poskytoval definovaný formát výstupu a to zhoršovalo jeho celkovú výkonnosť a úspešnosť. Toto správanie sme pri používaní iných modelov nezaznamenali.

Jednou s limitácií s ktorými sme sa pri vyhodnocovaní stretli bola aj veľkosť kontextového okna a teda počet vstupných tokenov, ktoré model dokáže spracovať v jednej požiadavke. Každý model môže mať definovanú rôznu veľkosť vstupu a preto je pri rozsiahlejších súboroch zdrojového kódu nejakú časť vynechať pre úspešné spracovanie požiadavky modelom.

5.3 Štatistiky používania

Počas testovania sme sa okrem úspešnosti modelov venovali aj vyhodnocovaniu časovej a finančnej náročnosti jednotlivých skenov. Táto kapitola poskytne prehľad a analýzu dát získaných z viacerých skenovacích cyklov, kde boli použité rôzne jazykové modely. Taktiež rozoberieme aké faktory majú vplyv na cenu a dĺžku skenu.

Tabuľka 8: Porovnanie priemernej časovej náročnosti a ceny prevedených skenov

Model usage statistics		
Model	Average scan time (minutes)	Average scan price ($USD)
gpt-3.5-turbo-0125	1,4	0,06
gpt-4-turbo	2,37	0,83
claude-3-haiku-20240307	4,89*	0,03
claude-3-opus-20240229	6,81*	0,95

*priemerný čas pre vykonanie analýzy pomocou Claude 3 modelov je však umelo navýšený. Volania API pre tieto modely sú limitované[29] počtom tokenov na minútu, ktorý je tento model schopný spracovať.

Model *gpt-3.5-turbo-0125* vykazuje najkratší priemerný čas skenovania (1,4 minúty) a zároveň pomerne nízku cenu (0,06 USD), čo ho robí rýchlym a ekonomicky výhodným, hoci jeho úspešnosť bola len 60%. Naproti tomu, model *claude-3-opus-20240229* mal relatívne dobrú úspešnosť 68%, vyžaduje dlhší čas na skenovanie (6,81 minúty*), a je najdrahší z hľadiska ceny za sken 0,95 USD.

Rovnako ako v predchádzajúcej kapitole, najlepšie výsledky sa nám podarilo dosiahnuť pomocou modelu *gpt-4-turbo*. Tento model dokázal spracovať veľké množstvo vstup-

[29]https://docs.anthropic.com/claude/reference/rate-limits

ných tokenov v najkratšom čase a dokázali sme s ním dosiahnuť najvyššiu úspešnosť (tabuľka 7).

Cena a dĺžka skenovania sú ovplyvnené viacerými faktormi, ako je veľkosť skenovanej aplikácie a počet pravidiel, ktoré boli počas skenovania aktivované. Komplexnejšie aplikácie s rozsiahlejším kódom a väčším počtom funkcií zvyčajne vyžadujú viac času na detailnú analýzu a môžu aktivovať viacero pravidiel, čo zvyšuje celkové náklady skenovania. Taktiež záleží od veľkosti súborového systému skenovanej aplikácie, to znamená, že počet používaných súborov ale aj ich veľkosť dokážu mať výrazný vplyv na dĺžku a cenu jedného vykonaného skenu.

Použité pravidlá pre detekciu podozrivých miest v zdrojovom kóde aplikácie (príloha C), môžu taktiež vplývať na merané vlastnosti analýzy. Nakoľko je možné do systému implementovať ľubovoľný počet pravidiel, analýza jednej aplikácie môže trvať aj niekoľko hodín.

Záver

V úvodných častiach tejto práce sme si vytvorili komplexný prehľad o možnostiach a výzvach spojených s použitím veľkých jazykových modelov (LLM) na úlohy spojené s kybernetickou bezpečnosťou v rôznych častiach infraštruktúry informačných systémov. Náš výskum a analýza sa ďalej detailne venovala bezpečnosti Android aplikácií. Rozoberali sme rôzne prístupy a nástroje, ktoré môžu byť využité pri útokoch a obrane týchto aplikácií a taktiež sme sa zaoberali otázkou ako je možné tento proces zefektívniť. Hlavným cieľom bolo vyvinúť systém, ktorý by účinne identifikoval a reagoval na bezpečnostné hrozby prostredníctvom automatizovanej analýzy.

Počas realizácie sme vykonali dôkladné testovanie našich systémov, použili sme rôzne jazykové modely a skúmali sme ich efektivitu v detegovaní pravdivo-pozitívnych a falošne-pozitívnych zraniteľností. Získali sme dôležité údaje, ktoré poukazujú na silné a slabé stránky použitých technológií. V práci sme prezentovali podrobné výsledky, ktoré reflektujú výkon a efektívnosť systému, ako aj finančné a časové nároky jednotlivých skenov.

V záverečnej časti sme zhodnotili celkovú efektívnosť nášho riešenia. Naše výsledky smerujú k integrácii pokročilejších analytických nástrojov a adaptívnejších modelov, ktoré by dokázali lepšie interpretovať predložený problém a taktiež spracovať rozsiahlejší kontext. Pri zavádzaní riešenia do produkčného prostredia je nevyhnutné vykonať ďalšie optimalizácie, zvýšiť robustnosť systému a zabezpečiť jeho schopnosť efektívne spracovávať a analyzovať veľké objemy dát v reálnom čase.

Celková práca a výsledný systém ukazuje potenciál využitia LLM na zlepšenie bezpečnostných analýz a ponúka základ pre ďalší výskum a vývoj v tejto rýchlo sa rozvíjajúcej oblasti. Táto práca môže slúžiť ako východiskový bod pre budúce inovácie a aplikácie v kybernetickej bezpečnosti a vývoji mobilných aplikácií.

Zoznam použitej literatúry

1. VASWANI, Ashish, SHAZEER, Noam, PARMAR, Niki, USZKOREIT, Jakob, JONES, Llion, GOMEZ, Aidan N, KAISER, Łukasz a POLOSUKHIN, Illia. Attention is all you need. In: *Advances in Neural Information Processing Systems*. 2017, s. 5998–6008.
2. AWAN, Abid Ali. *What is a Generative Model?* 2023. Dostupné tiež z: `https://www.datacamp.com/blog/what-is-a-generative-model`.
3. CHAKRABORTY, Anirban. *Revolutionizing Code Reviews using Generative AI: A practical approach using Google Cloud Vertex AI Codey API*. 2024. Dostupné tiež z: `https://medium.com/google-cloud/revolutionizing-code-reviews-using-generative-ai-a-practical-approach-using-google-cloud-vertex-ai-479e820c8e21`.
4. KREVZELJ, Tomislav. *Everything you need to know about generative AI and security*. 2023. Dostupné tiež z: `https://www.infobip.com/blog/everything-you-need-to-know-about-generative-ai-and-security`.
5. KAUFMAN, D. *Using Self-Hosted Generative AI to Create Targeted Phishing Emails*. 2023. Dostupné tiež z: `https://medium.com/@dkaufman_67683/using-self-hosted-generative-ai-to-create-targeted-phishing-emails-b192fbb2cf59`.
6. YIGIT, Yagmur, BUCHANAN, William J, TEHRANI, Madjid G a MAGLARAS, Leandros. *Review of Generative AI Methods in Cybersecurity*. 2024. Dostupné z arXiv: `2403.08701 [cs.CR]`.
7. SIMPLILEARN. *List Of Generative Adversarial Networks Applications*. 2023. Dostupné tiež z: `https://www.simplilearn.com/generative-adversarial-networks-applications-article`.
8. IBM. *What are recurrent neural networks?* 2023. Dostupné tiež z: `https://www.ibm.com/topics/recurrent-neural-networks`.
9. KARITA, Shigeki, CHEN, Nanxin, HAYASHI, Tomoki, HORI, Takaaki, INAGUMA, Hirofumi, JIANG, Ziyan, SOMEKI, Masao, SOPLIN, Nelson Enrique Yalta, YAMAMOTO, Ryuichi, WANG, Xiaofei, WATANABE, Shinji, YOSHIMURA, Takenori a ZHANG, Wangyou. A Comparative Study on Transformer vs RNN in Speech Applications. In: *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE, 2019. Dostupné z DOI: `10.1109/asru46091.2019.9003750`.

10. COLLINS, Emily Rosemary. *Transformer vs RNN: Women in Red Dresses (Attention Is All They Need?)* 2023. Dostupné tiež z: https://blog.finxter.com/transformer-vs-rnn-a-helpful-illustrated-guide/.

11. META. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. Dostupné tiež z: https://ai.meta.com/research/publications/llama-2-open-foundation-and-fine-tuned-chat-models/.

12. LUKYANENKO, Andrey. *Paper Review: Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. Dostupné tiež z: https://andlukyane.com/blog/paper-review-llama2.

13. GOTHANKAR, Mangesh. *9 Best Open-Source LLMs to Watch Out For in 2024*. 2024. Dostupné tiež z: https://www.signitysolutions.com/blog/best-open-source-llms.

14. FAUSCETTE, Michael. *Understanding the Limitations and Challenges of Generative AI*. 2024. Dostupné tiež z: https://em360tech.com/tech-article/understanding-limitations-and-challenges-generative-ai.

15. ZHENG, Shen, ZHANG, Yuyu, ZHU, Yijie, XI, Chenguang, GAO, Pengyang, ZHOU, Xun a CHANG, Kevin Chen-Chuan. *GPT-Fathom: Benchmarking Large Language Models to Decipher the Evolutionary Path towards GPT-4 and Beyond*. 2023. Dostupné z arXiv: 2309.16583 [cs.CL].

16. HACKL, Veronika, MÜLLER, Alexandra Elena, GRANITZER, Michael a SAILER, Maximilian. Is GPT-4 a reliable rater? Evaluating consistency in GPT-4's text ratings. *Frontiers in Education*. 2023, roč. 8. ISSN 2504-284X. Dostupné z DOI: 10.3389/feduc.2023.1272229.

17. WEI, Jason, WANG, Xuezhi, SCHUURMANS, Dale, BOSMA, Maarten, ICHTER, brian, XIA, Fei, CHI, Ed H., LE, Quoc V a ZHOU, Denny. Chain of Thought Prompting Elicits Reasoning in Large Language Models. In: OH, Alice H., AGARWAL, Alekh, BELGRAVE, Danielle a CHO, Kyunghyun (ed.). *Advances in Neural Information Processing Systems*. 2022. Dostupné tiež z: https://openreview.net/forum?id=_VjQlMeSB_J.

18. ZHONG, Wanjun, CUI, Ruixiang, GUO, Yiduo, LIANG, Yaobo, LU, Shuai, WANG, Yanlin, SAIED, Amin, CHEN, Weizhu a DUAN, Nan. *AGIEval: A Human-Centric Benchmark for Evaluating Foundation Models*. 2023. Dostupné z arXiv: 2304.06364 [cs.CL].

19. ANIL, Rohan, DAI, Andrew M., FIRAT, Orhan, JOHNSON, Melvin, LEPIKHIN, Dmitry, PASSOS, Alexandre, SHAKERI, Siamak, TAROPA, Emanuel, BAILEY, Paige, CHEN, Zhifeng, CHU, Eric, CLARK, Jonathan H., SHAFEY, Laurent El, HUANG, Yanping, MEIER-HELLSTERN, Kathy a AL., et. *PaLM 2 Technical Report*. 2023. Dostupné z arXiv: 2305.10403 [cs.CL].

20. HADI, Muhammad Usman, QURESHI, Rizwan, SHAH, Abbas, IRFAN, Muhammad, ZAFAR, Anas, SHAIKH, Muhammad Bilal, AKHTAR, Naveed, WU, Jia, MIRJALILI, Seyedali et al. Large language models: a comprehensive survey of its applications, challenges, limitations, and future prospects. *Authorea Preprints*. 2023.

21. AGARWAL, Anisha, CHAN, Aaron, CHANDEL, Shubham, JANG, Jinu, MILLER, Shaun, MOGHADDAM, Roshanak Zilouchian, MOHYLEVSKYY, Yevhen, SUNDARESAN, Neel a TUFANO, Michele. *Copilot Evaluation Harness: Evaluating LLM-Guided Software Programming*. 2024. Dostupné z arXiv: 2402.14261 [cs.SE].

22. KALLIAMVAKOU, Eirini. *Research: quantifying GitHub Copilot's impact on developer productivity and happiness*. 2022. Dostupné tiež z: https://github.blog/2022-09-07-research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/.

23. JETHVA, Minesh A. *Retrieval-Augmented Generation*. 2023. Dostupné tiež z: https://blog.mindmeldwithminesh.com/retrieval-augmented-generation-e7ebfe7ba4ff.

24. CHADHA, Aman a JAIN, Vinija. Retrieval Augmented Generation. *Distilled AI*. 2020. https://vinija.ai/nlp/RAG/.

25. ETHAPE, Priya, KANE, Riya, GADEKAR, Ghanashyam a CHIMANE, Sahil. *Smart Automation Using LLM*. 2023. Dostupné tiež z: https://www.proquest.com/scholarly-journals/smart-automation-using-llm/docview/2909076912/se-2.

26. BEGOU, Nils, VINOY, Jeremy, DUDA, Andrzej a KORCZYNSKI, Maciej. *Exploring the Dark Side of AI: Advanced Phishing Attack Design and Deployment Using ChatGPT*. 2023. Dostupné z arXiv: 2309.10463 [cs.CR].

27. DENG, Gelei, LIU, Yi, MAYORAL-VILCHES, Víctor, LIU, Peng, LI, Yuekang, XU, Yuan, ZHANG, Tianwei, LIU, Yang, PINZGER, Martin a RASS, Stefan. *PentestGPT: An LLM-empowered Automatic Penetration Testing Tool*. 2023. Dostupné z arXiv: 2308.06782 [cs.SE].

28. XU, Jiacen, STOKES, Jack W., MCDONALD, Geoff, BAI, Xuesong, MARSHALL, David, WANG, Siyue, SWAMINATHAN, Adith a LI, Zhou. *AutoAttacker: A Large Language Model Guided System to Implement Automatic Cyber-attacks*. 2024. Dostupné z arXiv: 2403.01038 [cs.CR].

29. MARTINDALE, Jon. *How to jailbreak ChatGPT: get it to really do what you want*. 2024. Dostupné tiež z: https://www.digitaltrends.com/computing/how-to-jailbreak-chatgpt/.

30. WARIS, Saman. *Here's how anyone can Jailbreak ChatGPT with these top 4 methods*. 2023. Dostupné tiež z: https://ambcrypto.com/heres-how-to-jailbreak-chatgpt-with-the-top-4-methods-5/.

31. GUPTA, Maanak, AKIRI, CharanKumar, ARYAL, Kshitiz, PARKER, Eli a PRAHARAJ, Lopamudra. *From ChatGPT to ThreatGPT: Impact of Generative AI in Cybersecurity and Privacy*. 2023. Dostupné z arXiv: 2307.00691 [cs.CR].

32. SHARMA, Tushar, KECHAGIA, Maria, GEORGIOU, Stefanos, TIWARI, Rohit, VATS, Indira, MOAZEN, Hadi a SARRO, Federica. *A Survey on Machine Learning Techniques for Source Code Analysis*. 2022. Dostupné z arXiv: 2110.09610 [cs.SE].

33. MIRSKY, Yisroel, MACON, George, BROWN, Michael, YAGEMANN, Carter, PRUETT, Matthew, DOWNING, Evan, MERTOGUNO, Sukarno a LEE, Wenke. *32nd USENIX Security Symposium (USENIX Security 23)*. VulChecker: Graph-based Vulnerability Localization in Source Code. Anaheim, CA: USENIX Association, 2023. ISBN 978-1-939133-37-3. Dostupné tiež z: https://www.usenix.org/conference/usenixsecurity23/presentation/mirsky.

34. BROWN, Michael D. *DARPA awards 1 million to Trail of Bits for AI Cyber Challenge*. 2024. Dostupné tiež z: https://blog.trailofbits.com/2024/03/11/darpa-awards-1-million-to-trail-of-bits-for-ai-cyber-challenge/.

35. MAHADEWA, Kulani. *Android security model*. 2016. Dostupné tiež z: https://techstarspace.engineer/tag/sandboxing/.

36. SINHAL, Ankit. *MVC, MVP and MVVM Design Pattern*. 2017. Dostupné tiež z: https://medium.com/@ankit.sinhal/mvc-mvp-and-mvvm-design-pattern-6e169567bbad.

Prílohy

A Štruktúra elektronického nosiča

mobsf-aianalyzer/mobsf/AiAnalzyer

· Zdrojový kód LLM systému (časť back-end)

mobsf-aianalyzer/mobsf/templates/ai_views

· Zdrojový kód používateľského rozhrania (časť front-end)

test_apk

· Súbory pripravené na testovanie

allsafe.apk

· Testovacia Android aplikácia

log_raw.txt

· Súbor runtime logov aplikácie

allsafe_external_fs

· Externý súborový systém aplikácie

allsafe_internal_fs

· Interný súborový systém aplikácie

mobsf-aianalyzer/docker-compose.yml

· Konfiguračný súbor pre program Docker

mobsf-aianalyzer/Dockerfile

· Konfiguračný súbor pre program Docker

/thesis.pdf

· Písomná časť diplomovej práce

*niektoré súbory elektronického nosiča sú súčasťou použitého open-source riešenia MobSF[30]

[30]https://github.com/MobSF/Mobile-Security-Framework-MobSF

B Používateľská príručka

Zdrojový kód riešenia je dostupný na adrese:
https://github.com/Logner109/mobsf-aianalyzer-pub.git

Spustenie docker kontainera:
git clone https://github.com/Logner109/mobsf-aianalyzer-pub.git
cd mobsf-aianalyzer
docker compose up

Konfigurácia prostredia:
Po inštalácii je nutné v súbore */home/mobsf/Mobile-Security-Framework-MobSF /mobsf/MobSF/mobsf_data/config.py* vytvoriť premenné pre držanie API kľúčov používaných modelov.
Vzor:
OPENAI_API_KEY = "<API-KEY>"
CLAUDE_API_KEY = "<API-KEY>"
HF_API_KEY = "<API-KEY>"

Analýza Android aplikácie:

1. Po spustení kontajnera sa môžeme dostať k aplikácii pomocou zadania adresy *http://localhost:8000/* do webového prehliadača
2. Nahrať súbor APK v sekcii "Static Analyzer"
3. Po dokončení analýzy je potrebné zozbierať logy aplikácie a nahrať súbor logov do priečinka */home/mobsf/.MobSF/app_data/<APP-MD5>/logs*
4. Rovnako je potrebné exportovať interný a externý súborový systém aplikácie a nahrať do priečinka */home/mobsf/.MobSF/app_data/<APP-MD5>/filesystem*
5. Teraz je možné v sekcii "Recent Scans" vyhľadať aplikáciu nahratú v druhom kroku.
6. Po otvorení záložky "AI Scan Report" je možné pomocou tlačidla "Start AI Scan" spustiť analýzu pomocou LLM

C Použité semgrep pravidlá

- id: webview-javascript-enabled
message: JavaScript is enabled in WebView, check if there is any potentially dangerous user controlled variable, that could introduce an XSS attack.
languages: [java, kotlin]
severity: WARNING
metadata:
owasp-mobile: "M4: Insufficient Input/Output Validation"
category: "M4"
llmchain: "vuln"
technology: android
pattern: $SETTINGS.setJavaScriptEnabled(true);

- id: webview-loadurl-call
message: Call to loadUrl method detected, check if there is any potentially dangerous user controlled variable, that could introduce a vulnerability.
languages: [java, kotlin]
severity: WARNING
metadata:
owasp-mobile: "M4: Insufficient Input/Output Validation"
category: "M4"
llmchain: "vuln"
technology: android
pattern: $W.loadUrl($DATA);

- id: hidden-ui-elements
message: Hidden elements in view can be used to hide data from user. But this data can be leaked, check if some sensitive data is hidden in this element.
languages: [java, kotlin]
severity: WARNING
metadata:
owasp-mobile: "M4: Insufficient Input/Output Validation"
category: "M4"

llmchain: "vuln"
technology: android
pattern-either:
- pattern: |
$X.setVisibility(View.GONE);
- pattern: |
$X.setVisibility(View.INVISIBLE);

- id: java-sql-usage-advanced
languages: [java]
message: Potential SQL query usage, check for SQL injection vulnerabilities.
severity: WARNING
metadata:
owasp-mobile: "M4: Insufficient Input/Output Validation"
category: "M4"
llmchain: "vuln"
technology: android
pattern-either:
- pattern: $DB.rawQuery(...);
- pattern: $DB.query(...);
- pattern: $DB.prepareStatement(...);
- pattern: $DB.Query(...);

- id: encryption-method
languages: [java]
message: "Check the encryption method implemented, is it weak and vulnerable encryption definition?"severity: WARNING
metadata:
owasp-mobile: "M10: Insufficient Cryptography"
category: security
technology: android
llmchain: "vuln"
pattern: "
byte[] encrypt(...)
...

"

- id: sharedpreferences-write
languages: [java]
message: "SharedPreferences write detected. Check if sensitive data is being stored in SharedPreferences."
severity: WARNING
metadata:
owasp-mobile: "M1: Improper Credential Usage"
category: security
technology: android
llmchain: "vuln"
pattern-either:
- pattern: $PREFS = $CONTEXT.getSharedPreferences(...);
- pattern: $EDITOR = $PREFS.edit();
- pattern: $EDITOR.put$TYPE(...);
- pattern: $EDITOR.$COMMIT();

- id: detect-logging
message: "Logging detected: Ensure that sensitive information is not logged."
languages: [java]
severity: INFO
metadata:
owasp-mobile: "M6: Inadequate Privacy Controls"
category: "M6"
llmchain: "logs"
technology: android
pattern-either:
- pattern: Log.$METHOD(...);
- pattern: Timber.$METHOD(...);

D Evaluácia úspešnosti

Táto príloha znázorňuje zoznam známych zraniteľností hodnotených v jednotlivých testovaných Android aplikáciách.

Aplikácia **AllSafe**[31]:

- Insecure Logging
- Hardcoded Credentials
- Insecure Shared Preferences
- Arbitrary Code Execution
- Insecure Broadcast Receiver
- Insecure Service
- Insecure Providers
- Insecure Activity
- Deeplink Exploitation
- SQL Injection
- Vulnerable WebView

Aplikácia **InsecureShop**[32]:

- Insufficient URL Validation
- Weak Host Validation Check
- Intent Redirection (Access to Protected Components)
- Unprotected Data URIs
- Theft of Arbitrary files from LocalStorage

[31] https://github.com/t0thkr1s/allsafe
[32] https://github.com/optiv/InsecureShop

- Insecure Broadcast Receiver
- Insecure use of FilePaths in FileProvider
- Insecure Implementation of SetResult in exported Activity
- Use of Implicit intent to send a broadcast with sensitive data
- Intercepting Implicit intent to load arbitrary URL
- Insecure Content Provider
- Logging Sensitive Data
- Sensitive Data in Shared Preferences

Aplikácia **AndroGoat**[33]:

- DeepLink exploitation
- Hard coding issues
- Activity
- Service
- Broadcast Receivers
- SQLi
- XSS
- SD Card Sensitive Files
- Logging Sensitive Data
- Sensitive Data in Shared Preferences
- Sensitive Data in Toast Messages
- Sensitive Data in Temp Files

Aplikácia **MASTG Playground**[34]:

[33]https://github.com/satishpatnayak/AndroGoat
[34]https://github.com/OWASP/MASTG-Hacking-Playground

- SQLi 1
- SQLi 2
- Code Injection
- Weak encryption
- External storage secrets
- Internal storage secrets
- Shared Prefs secrets
- Logging sensitive info
- XSS

Aplikácia **Damn Vulnerable Bank Application**[35]:

- Hardcoded sensitive information
- Logcat leakage
- Insecure storage (saved credit card numbers maybe)
- Exported activities
- Webview
- Deep links
- IDOR

Aplikácia **InsecureDroid - CTF**[36]:

- XSS
- Hardcoded Flag Code
- Exported Hidden Activity
- Hardcoded Flag Resources
- RCE

Aplikácia **Vulnerable Bank V2**[37]:

[35]https://github.com/rewanthtammana/Damn-Vulnerable-Bank
[36]https://github.com/B3nac/InjuredAndroid
[37]https://github.com/dineshshetty/Android-InsecureBankv2

- Flawed Broadcast Receivers
- Insecure Logging mechanism
- Vulnerable Activity Components
- Insecure Content Provider access
- XSS
- Weak Cryptography implementation
- Insecure SDCard storage
- Hardcoded secrets