Sběr a analýza vzorků mobilního malwaru se zaměřením na aplikační zdroje

Bc. Marek Maťcha

Diplomová práce
2023

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2022/2023

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Marek Maťcha**
Osobní číslo:	**A21182**
Studijní program:	**N0613A140022 Informační technologie**
Specializace:	**Kybernetická bezpečnost**
Forma studia:	**Prezenční**
Téma práce:	**Sběr a analýza vzorků mobilního malwaru se zaměřením na aplikační zdroje**
Téma práce anglicky:	**Collection And Analysis Of Mobile Malware Samples With A Focus On Application Resources**

Zásady pro vypracování

1. Vypracujte literární rešerši, která mapuje současný stav řešené problematiky.
2. Proveďte sběr vzorků mobilního malwaru z vybraných torrentů a file share serverů.
3. Realizujte srovnání vzorků získaných z torrentů vůči vzorkům, které jsou z file share serverů.
4. Proveďte analýzu všech validních vzorků a určete detekční potenciál aplikačních zdrojů.
5. Vyhodnoťte potenciál mobilního malwaru se zaměřením na aplikační zdroj.

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. E. B. KARBAB, M. DEBBAI, A. DERHAB a D. MOUHEB. Android Malware Detection using Machine Learning: Data-Driven Fingerprinting and Threat Intelligence [online]. Springer, 2021. ISBN 3030746631.
2. X. JIANG. a Y. ZHOU. Android malware [online]. Springer, 2013. ISBN 978-1461473930.
3. R. MATELES, D. REJABEK, O. Margalit a R. MOSKOVITCH. Decompiled APK based malicious code classification. Future Generation Computer Systems: Volume 110 [online]. 2020, 135-147. ISSN 0167-739X. Dostupné z: doi: https://doi.org/10.1016/j.future.2020.03.052.
4. D. MUGISHA, ANDROID APPLICATION MALWARE ANALYSIS. International Journal of Mobile Learning and Organisation 12 (Android Malware) [online]. 2019. ISSN 1746-7268.
5. A. A. ALI a A. S. H. ABDUL-QAWY. Static Analysis of Malware in Android-based Platforms: A Progress Study. IJCDS Journal [online]. 2020, 1-10. Dostupné z: doi:10.12785/ijcds/100132.
6. Y. LI, S. C. SUNDARAMURTHY, A. G. BARDAS, X. OU, D. CARAGEA, X. HU a J. JANG. Experimental Study of Fuzzy Hashing in Malware Clustering Analysis [online]. Washington, D.C.: Conference: 8th Workshop on Cyber Security Experimentation and Test (CSET 15), 2015.
7. S. HUH, S. CHO, J. CHOI, S. SHIN a H. LEE. A Comprehensive Analysis of Today's Malware and Its Distribution Network: Common Adversary Strategies and Implications. In: IEEE Access, volume 10 [online]. 2022, s. 49566-49584. ISSN 2169--3536. Dostupné z: doi: 10.1109/ACCESS.2022.3171226.
8. R. CUEVAS, M. KRYCZKA, R. GONZÁLES, A. CUEVAS a A. AZCORRA. TorrentGuard: Stopping scam and malware distribution in the BitTorrent ecosystem. In: Computer Networks: Volume 59 [online]. 2014, s. 77-90. ISSN 1389-1286. Dostupné z: doi: https://doi.org/10.1016/j.bjp.2013.12.007.

Vedoucí diplomové práce: **Ing. Milan Oulehla, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **2. prosince 2022**
Termín odevzdání diplomové práce: **26. května 2023**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 7. prosince 2022

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne

Marek Maťcha v.r.
podpis studenta

ABSTRAKT

Tato práce se zabývá analýzou vzorků mobilních aplikací běžících pod operační systém Android. V teoretické části jsou popsány základní pojmy a technologie související s operačním systémem Android. Dále jsou definovány jednotlivé kroky analýzy, které jsou následně aplikovány v praktické části na vytvořenou datovou sadu. Praktická část popisuje proces vytvoření datové sady a použití jednotlivých analytických postupů k určení infekčnosti zkoumaných aplikací a analýze všech maligních vzorků. Hlavním výsledkem práce je datová sada, která umožňuje navazující výzkum. Práce rovněž přináší analýzu, která se týká aktuálního stavu aplikací pro operační systém Android distribuovaných prostřednictvím file share serverů a torrentů. Dalším přínosem práce je zhodnocení detekčního potenciálu aplikačních zdrojů.

Klíčová slova: Android, Torrent, File share, Aplikační zdroje, Analýza, Malware, Hash, Virové skenery

ABSTRACT

This thesis deals with analyzing mobile application samples running under the Android operating system. In the theoretical part, the basic concepts and technologies related to the Android operating system are described in detail. Furthermore, the individual steps of the analysis are defined, which are then applied in the practical part to the created dataset. The practical part describes creating the dataset and using each analytical procedure to determine the infectivity of the examined applications and to analyze all malignant samples. The main result of the work is a dataset that enables follow-up research. The thesis also presents an analysis that covers the current state of Android applications distributed via file share servers and torrents. Another contribution of the thesis is the evaluation of the detection potential of the application resources.

Keywords: Android, Torrent, File share, Application sources, Analysis, Malware, Hash, Virus scanners

Rád bych poděkoval svému vedoucímu práce panu Ing. Milanu Oulehlovi, Ph.D., za jeho zkušené vedení mé diplomové práce a cenné rady, které mi v průběhu studia poskytnul. Dále bych chtěl poděkovat členům laboratoře PTLAB při Univerzitě Tomáše Bati ve Zlíně za výpomoc s některými částmi práce. Samozřejmě, nesmím zapomenout na moji rodinu, která mi vždycky poskytovala největší podporu, nejen ve studiu, ale i v dalších oblastech mého života.

OBSAH

ÚVOD

S neustálým vývojem technologií se čím dál více klade důraz na bezpečnost. Ne jinak tomu je u mobilních telefonů. Dnes snad již každý uživatel vyspělejší země mobilní telefon vlastní a s tím se otevírá otázka, zda je jeho manipulace se zařízením bezpečná. Z důvodu největšího podílu mobilních telefonů s operačním systémem Android na trhu, který činí asi 70% [1], se práce zaměřila právě na tuto problematiku. Tyto telefony obsahují aplikaci, která se spojí s distribuční platformou Google Play, ze které si je možné stáhnout jejich aplikaci, kterou chce vlastník využít. Jedná se o službu, která nepřetržitě provádí kontrolu aplikací pomocí vestavěných antivirových skenerů. Z tohoto důvodu je možné tvrdit, že stahování prostřednictvím Google Play je považováno za nejbezpečnější variantu [2]. Nicméně, některé aplikace jsou placené, a to je klíčový faktor, který ovlivňuje bezpečnost. Mnoho uživatelů si přeje získat aplikace zdarma a proto se obrací na distributory, kteří je nabízejí bez nutnosti platby. Mezi takové se bezpochyby řadí distribuce aplikací prostřednictvím torrentových stránek a file share serverů. S tímto přístupem však přicházejí i bezpečnostní rizika, protože soubory na těchto stránkách nejsou během nahrávání nikterak kontrolovány a mnohdy obsahují riziko, že se jedná o malware. Ten následně může způsobit problémy nejen se zařízením, ale také například odcizení soukromých dat uživatele. A z tohoto důvodu je tvorba komplexní datové sady jedním z prvních klíčových faktorů této diplomové práce.

Druhou, ne méně klíčovou částí diplomové práce, byly aplikační zdroje. Kontrola a analýza adresářů res, souborů XML ad. byly nezbytnou součástí k následnému určení jejich detekčního potenciálu.

I. TEORETICKÁ ČÁST

1 LITERÁRNÍ REŠERŠE

Jako první bylo nutné provést důkladnou analýzu stávajících řešení, tzn. rešerši literárních pramenů. K důkladné analýze řešené problematiky mobilního malwaru bylo nutné zahrnout několik kroků, které bylo zapotřebí provést. Těmito kroky byly sběr dostatečné datové sady z různých zdrojů, zjištění infekčnosti vzorků, analýza infekčních vzorků a v neposlední řadě realizovat srovnání mezi jednotlivými zdroji aplikací, a to vzhledem k unikátnosti vzorků. V následujícím textu byly jednotlivé kroky popsány. Konkrétně byly analyzovány dostupné přístupy řešených problémů.

1.1 Analýza stávajících přístupů k řešené problematice

Pro analýzu jednotlivých vzorků mobilních aplikací je nejprve nutné vyřešit situaci týkající se jejich sběru. Tato data je možné získat několika způsoby. Jedním z nejlehčích a nejvíce využívaných je stažení datové sady z některého Android úložiště malwaru, kterými jsou např. Drebin, Datová sada AMD, Android Gnome ad. [3]. Důvodem využívaní těchto úložišť je, že disponují velkým množstvím infikovaných aplikací, viz. [3]. Tento způsob byl využit ve velkém množství odborných článků pro detekci mobilního malwaru, viz. [4], [5] a [6]. Existují také jiné, náročnější způsoby, jak datovou sadu získat a jedním z nich je, že dojde k jejímu vytvoření pomocí stažení dostatečného množství vzorků. Zde se objevují dva možné postupy, kterými jsou stažení vzorků prostřednictvím file share serverů nebo torrentů. Jelikož se jedná o časově náročný postup, viz. [7], tak využití těchto přístupů není v literárních pramenech v souvislosti s mobilními aplikacemi řešeno.

V datových sadách se ale obvykle objevuje větší množství benigních aplikací, než infikovaných, viz. [8] a [9]. Z tohoto důvodu je nutné vzorky otestovat, a to nejčastěji pomocí některého z řad virových skenerů. Mezi nejvyužívanější skenery patří VirusTotal, který určuje, zda je aplikace benigní nebo se jedná o malware, viz. [10] a [11].

Na otestované vzorky je dále možné využít statickou nebo dynamickou analýzu, jako je popsáno například v [12]. Statickou analýzou se rozumí zkoumání a analýza aplikace bez jejího spuštění. Dochází zde především ke zkoumání aplikační logiky a zdrojů aplikací (XML, ad.). Jedním z kroků této analýzy může být hledání podobnosti mezi jednotlivými vzorky pod jiným názvem pomocí hashovacích algoritmů, jako například v [13]. Nejefektivnějším způsobem v oblasti hashovacích algoritmů je využití fuzzy hashů, jelikož na rozdíl od klasických hashů (např. rodiny SHA) dokážou určit podobnost vzorků a nejen jejich přímou duplicitnost, viz. [14]. Dalším krokem statické analýzy je procházení

jednotlivých souborů, které jsou obsaženy v Android Appliacation Package (APK) balíčku, což je instalační soubor ve formátu zip využívaný operačním systémem Android [15]. K tomuto kroku je nutné soubory APK dekompilovat. Zde existuje několik nástrojů, přičemž mezi jeden z nejznámějších patří APKTool, viz. [16]. V takto dekompilovaném souboru je poté možné prozkoumávat jednotlivé části, jako je např. Androidmanifest.xml, viz [17]. Dalším možným postupem analýzy je zkoumání souboru ve formátu .dex. U tohoto typu souboru následně dochází ke zkoumání hlaviček, zda se v obsahu neobjevují jiné spustitelné soubory, viz. [18], a to pomocí některého z hexadecimálních editorů. Dále se v mnohých publikacích vyskytují jiné možnosti pro statickou analýzu, kterými jsou metody s využitím strojového učení, viz. [19], kde z jedním konkrétních případů mohou být neuronové sítě, viz. [20].

Při detekci mobilního malwaru je využívána také dynamická analýza. Ta se zabývá chováním aplikace za běhu, a to bez zkoumání zdrojového kódu dané aplikace. Tímto přístupem se zabývá několik publikací, a to především tvorbou vlastního frameworku pro dynamickou analýzu, viz. [21] a [22]. Dále se také vyskytují publikace využívající hybridní analýzu, viz. [23], která kombinuje statickou a dynamickou analýzu.

V tabulce 1 je uveden přehled uvedených literárních pramenů v kontextu s identifikací a analýzou mobilního malwaru. Pro každou část jsou v tabulce uvedeny dvě možnosti, kterými jsou plné kolečko, jenž označuje, že se daná problematika v publikaci vyskytuje, naopak prázdné kolečko znázorňuje, že se v publikacích tato část nenachází. Tabulka 1 jasně dokládá, že oblasti, související s aplikacemi pro systém Android, zahrnující statickou i dynamickou analýzu s využitím virových skenerů, představují komplexní problémy, které jsou často řešeny. V důsledku toho jsou v tabulce 1 uvedeny pouze vybrané publikace zaměřené na tyto specifické aspekty. Naopak téma analýzy aplikací pro operační systém Android ve vztahu k souboru dat tvořenému vzorky získaných prostřednictvím torrentových a file share serverů se v žádné publikaci neobjevuje, což poukazuje na významnou mezeru v této oblasti.

Tabulka 1 - Přehled literární rešerše

Zdroje	Android app	Tvorba datové sady	Torrent File share	Virový skener	Fuzzy hash	Statická analýza	Dynamická analýza
[3]5	●	○	○	●	●	●	○
[22]	●	○	○	○	○	●	●
[3]	●	○	○	●	○	●	●
[10]	●	○	○	●	○	●	●
[12]	●	○	○	○	○	●	●
[7]	○	●	●	○	○	○	○
[16]	●	○	○	○	○	●	●
[4]	●	○	○	●	●	●	●
[14]	●	○	○	○	●	●	●
[20]	●	○	○	●	○	●	●
[8]	●	○	○	○	○	●	●
[9]	●	○	○	●	○	●	●
[6]	●	○	○	●	○	●	●
[19]	●	○	○	●	●	●	●
[11]	●	○	○	●	○	●	●
[18]	●	○	○	○	○	●	●
[23]	●	○	○	○	○	●	●
[21]	●	○	○	●	○	●	●
[13]	●	○	○	○	●	●	●
[17]	●	○	○	○	○	●	○

Z přehledu literatury vyplývá, že práce zahrnují jednotlivé dílčí přístupy při analyzování potenciálních hrozeb mobilních aplikací. Problematika související s datovými sadami z file share serverů a torrentových stránek však v publikacích v souvislosti s mobilním malwarem do potřebné hloubky chybí. Tyto datové sady jsou avšak z hlediska bezpečnosti uživatelů mobilních aplikací klíčové. Rovněž nejsou diskutovány tvorby vlastních datových sad, na rozdíl od využívání již vytvořených sad z Android úložišť malwaru.

1.2 Aktuálnost řešené problematiky

Hlavním důvodem pro řešení problematiky spjaté s operačním systémem Android je, že se jedná o nejpoužívanější mobilní operační systém na planetě [24]. S tím souvisí i závažná bezpečnostní situace, jelikož operační systém Android je velmi atraktivní pro tvůrce malwaru [25].

Z tohoto důvodu byla provedena rešerše s cílem zjistit aktuální stav řešené problematiky. Z literární rešerše, která je uvedena v tabulce 1, vyplývá, že v současné době se výzkum převážně soustředí na statickou a dynamickou analýzu vzorků, mnohdy za pomocí umělé inteligence. Existující postupy se zaměřují na analýzu dex souborů, oprávnění a další prvky, ale nevěnují se zkoumání aplikačních zdrojů, které mohou mít detekční potenciál. Přičemž je také důležité zmínit, že většina analýz se soustředí na vzorky stažené z volně dostupných Android úložišť malwaru, ale nezabývají se tvorbou vlastních datových sad a řešením bezpečnostních otázek týkajících se file share serverů a torrentů.

Na základě zjištěných informací z literární rešerše se tato práce zaměřuje právě na neprozkoumané části. Konkrétně došlo k vytvoření vlastní datové sady prostřednictvím dříve nezanalyzovaných zdrojů aplikací, kterými jsou file share servery a torrenty. Na vytvořené datové sadě byla následně provedena bezpečnostní analýza, díky které došlo ke zjištění bezpečnostní situace týkající se mobilních aplikací na file share serverech a torrentech. Dále také byla provedena analýza se zaměřením na aplikační zdroje, kde byl zjišťován jejich detekční potenciál. Konkrétně došlo ke zkoumání jednotlivých adresářů a souborů, jako jsou classes.dex, textové řetězce, multimediální soubory související s GUI (Graphic User Interface), porovnávání struktury zdrojových adresářů, XML layouty, AndroidManifest.xml a ostatní XML soubory.

Přínosem této práce je tedy v první řadě vytvořená vlastní datová sada validních vzorků z dříve nezanalyzovaných zdrojů aplikací. Dále byla vyhodnocena bezpečnostní situace na file share serverech a torrentech se zaměřením na aplikace běžících pod operačním systémem Android. A v neposlední řadě došlo ke zkoumání aplikačních zdrojů a zjišťování jejich detekčního potenciálu.

Výzkum provedený v rámci této práce bude využit v penetrační laboratoři PTLAB při Univerzitě Tomáše Bati (UTB) ve Zlíně.

1.3 Časová náročnost řešené problematiky

Tato podkapitola se zabývá časovou náročností jednotlivých částí práce, včetně možných překážek a limitů, které se v průběhu vyskytovaly. Postupně byly rozebrány všechny vykonané úkony zahrnující sběr datové sady, kontrolu infekčnosti, vyhodnocení duplicitních vzorků a analýzu infikovaných vzorků s primárním zaměřením na aplikační zdroje.

Úvodní částí práce bylo stažení vzorků prostřednictvím file share serverů a torrentů. V prvním kroku bylo nutné vyhledat jednotlivé zdroje obsahující dostatečné množství vzorků. Následně již došlo ke stahování aplikací z vybraných zdrojů. Celková doba těchto úkonů zabrala okolo 200 hodin čistého času. To bylo primárně zapříčiněno důvodem omezené rychlosti stahování z jednotlivých zdrojů. Pokud se jedná o file share servery, tak zde byla průměrná rychlost stahování okolo 250 $kilobajtů\ za\ sekundu$. U torrentů byla rychlost stahování ovlivněna počtem uživatelů, jenž jeho část sdílí s ostatními. Celkově bylo za tuto dobu staženo 1771 vzorků s celkovou velikostí okolo 62 gigabajtů.

Dalším krokem byla kontrola infekčnosti stažených vzorků, k čemuž byla využita online služba VirusTotal. Postupně do VirusTotal byly nahrávány všechny vzorky z datové sady, které byly vyhodnocovány dle počtu jejich pozitivních detekcí. Tento úkon zabral okolo 17 hodin čistého času, což bylo zapříčiněno limitací možných kontrol souborů. Služba umožnuje nahrát a zkontrolovat pouze 20 vzorků v časovém rozsahu 10 minut.

Následně proběhla kontrola unikátnosti vzorků pomocí hashovacích algoritmů. V prvním kroku byly využity k vyhodnocení klasické hashovací algoritmy, konkrétně nekolizní funkce SHA-256. Pro tento krok byla vybrána volně přístupná aplikace HashMyFiles, která po nahrání souboru vypočítá jeho libovolně zvolenou hashovací funkci. Nahrání a vyhodnocení duplicitních vzorků, včetně jejich „odstranění“ z datové sady, kvůli její unikátnosti zabralo okolo 2 hodin čistého času. V druhém kroku došlo k vyhodnocení podobností jednotlivých vzorků pomocí fuzzy hashovacích algoritmů. Zpracování proběhlo v jazyce Python, konkrétně v jeho open-source vývojovém prostředí Spyder, pomocí knihovny python-ssdeep. Porovnání všech vzorků, včetně jejich analýzy trvalo okolo 3 hodin.

V dalších krocích již docházelo k analýze jednotlivých adresářů či souborů, které aplikační balíček obsahuje. Z tohoto důvodu bylo nutné všechny vzorky dekompilovat. Dekompilace proběhla v penetrační laboratoři PTLAB při UTB ve Zlíně pomocí dekompilačního nástroje Perfect Knife. Data byla poskytnuta laboratoři, která vzorky dekompilovala a poté v archivu ZIP zpět poskytnula. Jedinou překážkou byla dekomprimace složky obsahující všechny

dekompilované aplikace, jelikož obsahovala miliony souborů v celkové velikosti okolo 250 gigabajtů. Z důvodu omezeného výpočetního výkonu tento úkon trval zhruba 6 dnů čistého času, což je v přepočtu okolo 144 hodin.

Díky dekompilovaným aplikacím již mohlo dojít k analýze jednotlivých adresářů a složek. Primárním úkolem byla analýza aplikačních zdrojů a určení jejich detekčního potenciálu. Zde došlo ke kontrole adresářů obsahujících grafiku, textových řetězců a porovnání adresářů u čistých aplikací vůči potenciálně infikovaným aplikacím. Tento úkon zabral okolo 40 hodin čistého času. Dále byly řešeny dílčí analýzy práce, jako bylo ověření dex souborů. Zde byla kontrolována binární struktura souboru, včetně hlaviček a jestli počet souborů má závislost na infekčnosti aplikace. Dále byl analyzován soubor AndroidManifest.xml, konkrétně určována Application Programming Interface (API) úroveň aplikace, která charakterizuje stáří aplikace, dále povolená oprávnění a jednotlivé názvy labelů a aplikací. Tato analýza zabrala okolo 30 hodin čistého času. Celková časová náročnost praktické části, která neobsahovala zápis jednotlivých postupů a vyhodnocení, je znázorněna v tabulce 2.

Tabulka 2 - Detailní rozložení časové náročnosti jednotlivých úkonů práce

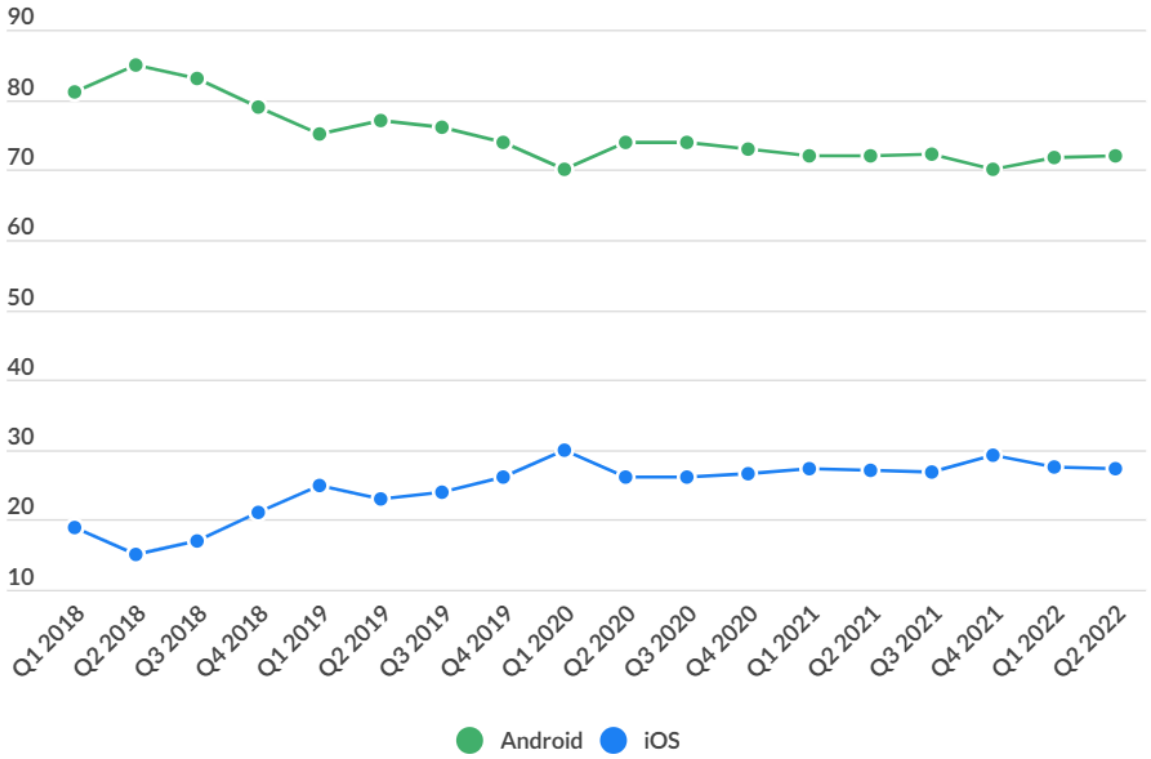
	Sběr, rozřazení a dekompilace datové sady			**366 hodin**
Sběr datové sady	Torrenty 70 hodin	File share servery 130 hodin		
Kontrola infekčnosti	Torrenty 6 hodin	File share servery 11 hodin		
Zjištění duplicit vzorků	Klasické hash funkce 2 hodiny	Fuzzy hash funkce 3 hodiny		
Dekomprimace	144 hodin			
	Analýza infekčnosti vzorků			**69 hodin**
Aplikační zdroje	Porovnání adresářů 0,5 hodiny	Textové řetězce 18,5 hodiny	Grafické části 20 hodin	
Dílčí problémy	AndroidManifest.xml 20 hodin	dex soubory 10 hodin		

Jak je možné v tabulce 2 vidět, tak celková časová náročnost týkající se jednotlivých úkonů práce činila v součtu okolo 435 hodin. Od této hodnoty je ale nutné odečíst 144 hodin, které představují dekomprimaci, a to z důvodu pouze pasivního čekání na dokončení úkonu. Celková doba aktivní práce tedy činila 291 hodin. Nutné říct, že tento čas zabral pouze sběr a následná práce s daty, nikoli celkové sepsání diplomové práce. Dále bylo nutné všechny postupy a výsledky sepsat do uspořádané podoby včetně teoretického základu.

Práce vznikala v průběhu dvou let, kde výstupem byly výsledky obsahující důležité informace, jež byly využity v diplomové práci. Výsledky práce budou použity pro následný výzkum penetrační laboratoře PTLAB při UTB ve Zlíně.

2 OPERAČNÍ SYSTÉM ANDROID

Android je open source [26] operační systém založený na Linuxovém jádře, jenž byl vyvinut společností Android Inc. a následně zakoupen společností Google Limited liability company (LLC). Není určen pouze pro chytré telefony, ale v nynější době je využíván i v ostatních zařízeních, jako jsou tablety, televize, hodinky apod. Procentuální podíl tohoto operačního systému je přes 70% [1] všech využívaných mobilních zařízení na světě, díky čemuž dominuje na celosvětovém trhu. Na obrázku 1 je možné vidět porovnání operačního systému Android vůči druhému nejpoužívanějšímu operačnímu systému, kterým je iPhone OS (iOS). IOS je mobilní operační systém pro zařízení vyráběná společností Apple. iOS funguje na zařízeních iPhone, iPad, iPod Touch a Apple TV [27].



Obrázek 1 - Podíl Android vs. iOS na globálním trhu (%) [1]

Android souvisí s americkou technologickou společností Android incorporated (Inc.), za kterou stojí zakladatelé Andy Rubin, Rich Miner, Nick Searsem a Chris White. Společnost byla založena v říjnu roku 2003 s cílem vyvinout operační systém pro digitální fotoaparáty [28]. Po několika měsících došlo ke změně, jenž vedla k zaměření na operační systémy mobilních zařízení, který bude konkurenceschopný ostatním operačním systémům. S rostoucí představou obrovského potenciálu projektu byla společnost následně v roce 2005 odkoupena společností Google LLC za více než 50 milionů dolarů [29]. Následně roku 2007

bylo oznámeno založení Open Handset Alliance, konsorcia desítek technologických a mobilních telefonních společností za účelem vývoje a propagace Androidu jako bezplatného open source operačního systému s podporou třetích stran. Roku 2008 došlo k představení mobilního zařízení s operačním systémem Android na trh. Jednalo se konkrétně o T-Mobile G1, známý pod názvem HTC Dream, díky němuž se poprvé Android stal hlavním konkurentem pro Apple [30]. Prvním oficiálním veřejným kódovým označením pro Android bylo až v dubnu 2009 po vydání verze 1.5 Cupcake.

Tímto krokem začal vzestup operačního systému a bylo vydáno do dnešní doby (10.2.2023) 33 dalších verzí, které jsou přehledně vidět v tabulce 3, viz. sloupec Úroveň API, včetně jejich názvu a dnešního procentuálního využití uživateli operačního systému Android.

Tabulka 3 - Verze operačního systému Android [31]

Verze	**Název**	**Datum vydání**	**Úroveň API**	**Využití [%]**
1.5	Cupcake	Duben 2009	3	0
1.6	Donut	Září 2009	4	0
2.0–2.1	Eclair	Říjen 2009	5-7	0
2.2	Froyo	Květen 2010	8	0
2.3	Gingerbread	Září 2010	9-10	0
3.0	Honeycomb	Únor 2011	11-13	0
4.0	Ice Cream Sandwich	Říjen 2011	14-15	0
4.1–4.3	Jelly Bean	Červen 2012	16-18	0,3
4.4	KitKat	Září 2013	19-20	0,9
5.0–5.1	Lollipop	Listopad 2014	21-22	2,6
6.0	Marshmallow	Srpen 2015	23	3,5
7.0	Nougat	Srpen 2016	24-25	4,5
8.0	Oreo	Březen 2017	26-27	10,9
9.0	Pie	Srpen 2018	28	14,5
10	Android Q	Září 2019	29	22,3
11	Red Velvet Cake	Září 2020	30	27,0
12	Material you	Říjen 2021	31-32	13,3
13	Tiramisu	Duben 2022	33	0,2
14	Upside Down Cake	2023?	?	0

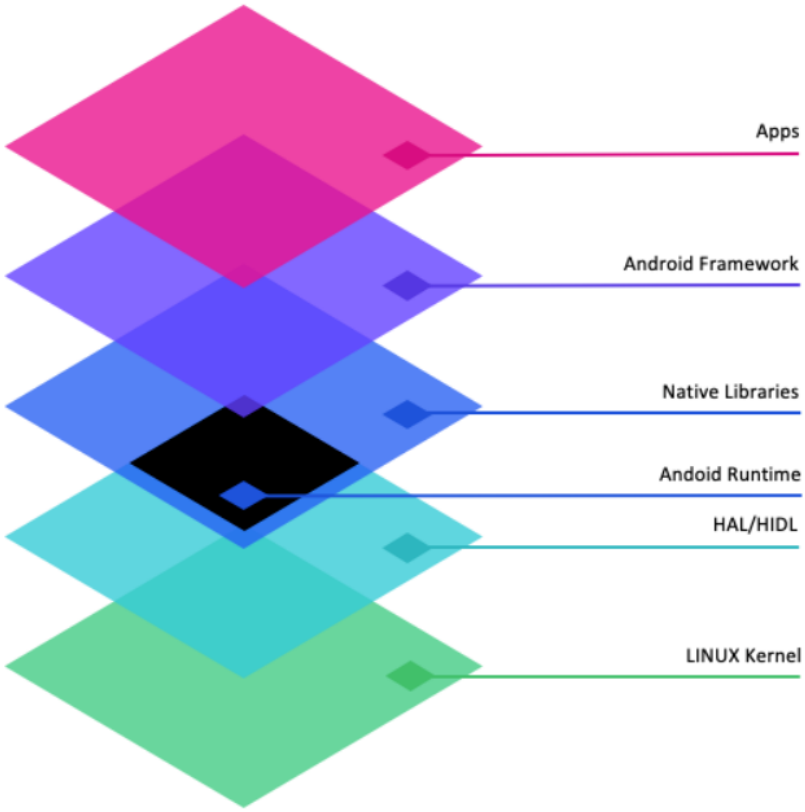
V tabulce 2 je možné vidět, že v průběhu 13 let bylo vydáno 33 verzí operačního systému, přičemž jedna již byla představena pro rok následující. V tabulce se vyskytuje jejich celočíselná hodnota API úrovně, která přesně identifikuje verzi sady API, které může aplikace používat [32]. Tedy srozumitelně určuje, zda je daná aplikace kompatibilní se systémem před její instalací. Zajímavostí je, že nejvíce využívané verze operačního systému se vyskytují mezi staršími verzemi, které byly vydány v rocích 2019 a 2020 [31].

2.1 Architektura

Jedná se o systém, který je založen na vrstvách pro podporu potřeb mobilních zařízení. Operační systém Android obsahuje vrstvy, kterými jsou [33]:

- Linux Kernel
- Hardware Abstraction Layer (HAL)
- Android Runtime
- Android Framework
- Apps

Na obrázku 2 je možné vidět tyto hlavní vrstvy platformy Android.



Obrázek 2 - Architektura operačního systému Android [34]

2.1.1 Linux Kernel

Základ operačního systému Android je tvořen Linuxovým jádrem. Jedná se o nejnižší vrstvu, která poskytuje základní funkcionalitu a všechny ostatní vrstvy využívají její služeb. Linuxové jádro je odladěné, díky čemuž je dosaženo vyšší stability systému, minimalizace

chyb apod. Odladění zahrnuje testování, opravu chyb (bugů) a optimalizaci kódu. Proces odladění zahrnuje detailní analýzu chyb, testování různých scénářů a situací, včetně extrémních podmínek, a zlepšování výkonu prostřednictvím optimalizací kódu. Z tohoto důvodu je vyvinutí vlastního jádra velmi složité [35]. Linuxové jádro také zajišťuje jednotlivé klíčové funkce Androidu, kterými jsou např.:

- Správy procesů – zajišťuje funkčnost aplikací na zařízení,
- správy paměti – zajištění volnosti pro vývoj aplikací,
- sítě – zajištění komunikace v určité síti a
- zabezpečení – zajištění ochrany mezi aplikacemi a systémem.

Právě na bezpečnost je na zařízeních kladen velký důraz. Poskytuje a prosazuje různá opatření, která využívají vícevrstvé zabezpečení k ochraně uživatelských dat na mobilních zařízeních. Existují určitá bezpečná výchozí nastavení, která ochrání uživatele, a také některé možnosti, které může vývojářská komunita použít k vytvoření bezpečných aplikací. Pro tato nastavení slouží v jádře především jednotlivé níže zmíněné bezpečnostní moduly pro Linux [36]:

SELinux

Tento modul definuje řízení přístupu pro aplikace, procesy a soubory v systému. Využívá bezpečnostní politiky, což je sada pravidel, která SELinuxu říkají, k čemu lze nebo nelze přistupovat, aby vynutil přístup povolený politikou [37]. Jedná se tedy o modul, který kontroluje, zda jsou oprávnění přístupu povoleny, v opačném případě je krok odepřen.

Smack,AppArmor,TOMOYO

Jako v předchozím případě, tak moduly Smack, AppArmor a TOMOYO se převážně zabývají řízením přístupu v systému. I přes to, že SELinux více zatěžuje systém svým během, tak se jedná o nejrozvinutější bezpečnostní modul, který je nejčastěji využíván [38].

Dalšími komponenty související se zabezpečením jsou zejména:

Audit

Linuxové jádro disponuje komplexním auditním systémem, který slouží k analýze chování systému a mohou pomoci odhalit pokusy o kompromitaci systému [36].

Seccomp

Jedná se o souhrn funkcí linuxového jádra, které mohou fungovat jako filtr systémových volání [39].

Integrity management

Subsystém integrity jádra je používán ke zjištění, zda u souboru nedošlo ke změně, a to jak vzdáleně, tak lokálně. Jedná se o porovnání hash otisků daného souboru s již dříve uloženou hodnotou [40].

Hardening and Platform Security

Jedná se o techniky pomáhající ke snížení rizika kompromitace systému. Mezi jednu z technik spadá randomizace adresního prostoru, která umisťuje různé paměťové oblasti spustitelného souboru uživatele do náhodných umístění, což pomáhá předcházet určitým třídám útoků [36].

2.1.2 Hardware Abstraction Layer

Hardwarová abstrakční vrstva (HAL) je klíčovou součástí systému Android, která poskytuje standardizované prostředí a zpřístupňuje možnosti hardwarových zařízení nadřazenému rámci Java API [41]. Skládá se z několika knihovních modulů, z nichž každý poskytuje rozhraní pro určitý typ hardwarové komponenty, například fotoaparát nebo modul Bluetooth. Když rámec API provede volání umožňující přístup k hardwarovému zařízení, systém Android načte knihovní modul pro vybranou hardwarovou komponentu [41].

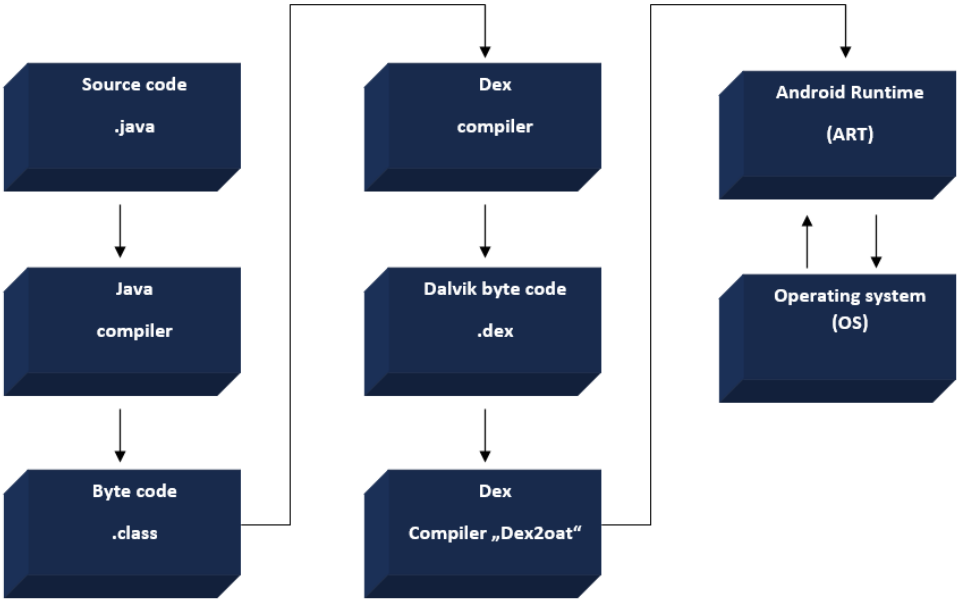
2.1.3 Native Libraries

Mnoho klíčových součástí a služeb systému Android, například ART a HAL, je vytvořeno pomocí nativního kódu, který vyžaduje nativní knihovny napsané v jazycích C a C++. Platforma Android poskytuje rozhraní API frameworku Java, které aplikacím zpřístupňuje funkce některých těchto nativních knihoven. Například k rozhraní OpenGL ES je možné přistupovat prostřednictvím rozhraní Java OpenGL API frameworku Android a přidat tak do aplikace podporu pro kreslení a manipulaci s 2D a 3D grafikou [42].

Pokud dochází k vyvíjení aplikace, která vyžaduje kód v jazyce C nebo C++, je možné k přístupu k některým z těchto nativních knihoven platformy přímo z nativního kódu použít Android NDK. [42].

2.1.4 Android Runtime

Android runtime (ART) je spravované běhové prostředí používané aplikacemi a některými systémovými službami v systému Android. Běhové prostředí pro Android je v architektuře na stejné vrstvě jako nativní knihovny. ART a jeho předchůdce Dalvik Virutal Machine (DVM) byly původně vytvořeny speciálně pro projekt Android. ART aplikuje několik technologií, jako například Ahead-of-time (AOT) a Just-in-time (JIT, který již byl implementován i v DVM). V dnešní době ART již využívá hybridního přístupu, který spočívá v aplikování profilové kompilace (JIT i AOT) [43]. Princip fungování spočívá v tom, že u částí aplikační logiky, která je spuštěna poprvé se provede JIT kompilace a následně jsou přidány do profilu. Taktéž se do profilu dostávají často používané části dané aplikace. Poté pokud je zařízení nečinné a nabíjí se, tak kompilační démon provádí AOT kompilaci na základě aktualizovaného profilu. Což znamená, že u dalších spuštění aplikace je využíván AOT zkompilovaný kód a nemusí docházet k JIT kompilaci. Výhodou je rychlost při instalaci aplikací a rychlost u často využívaných aplikací. Obrázek 3 názorně představuje celý průběh ART [44].



Obrázek 3 - Android Runtime

Pro detailnější vysvětlení jsou níže v textu postupně jednotlivé části rozebrány.

Virtual Machine (VM)

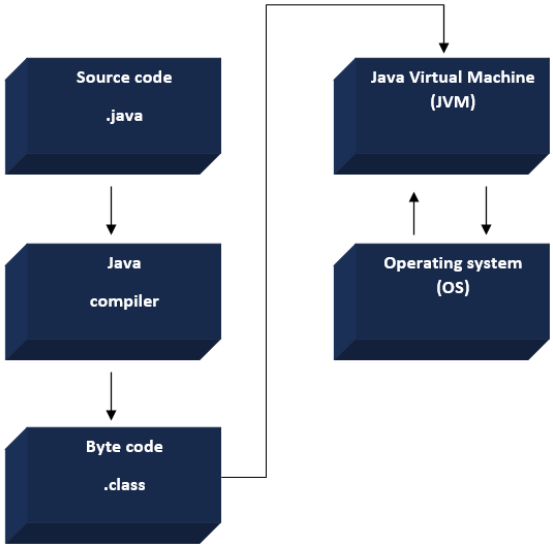
Virtuální stroj je abstrakce nativního stroje podporovaná zdroji nativního stroje. Jeho úkolem je převést kód specifický pro daný jazyk do formátu kompatibilního s provozem na

virtuálním počítači. Virtual Machine umožňuje spuštění stejného kódu na více platformách nezávisle na základním hardwaru [45].

Java Virtual Machine (JVM)

JVM slouží pro spouštění Java aplikací a je založen na zásobníku. JVM je ten, který ve skutečnosti volá hlavní metodu přítomnou v kódu Java. Java aplikace se nazývají WORA (Write Once Run Anywhere). To znamená, že programátor může vyvíjet kód Java na jednom systému a může očekávat, že bude fungovat na jakémkoli jiném systému s podporou Java bez jakýchkoli úprav. To vše je možné díky JVM. Když kompilujeme soubor .java, kompilátor Java vygeneruje soubory .class (obsahující bajtkód) se stejnými názvy tříd jako v souboru .java [46].

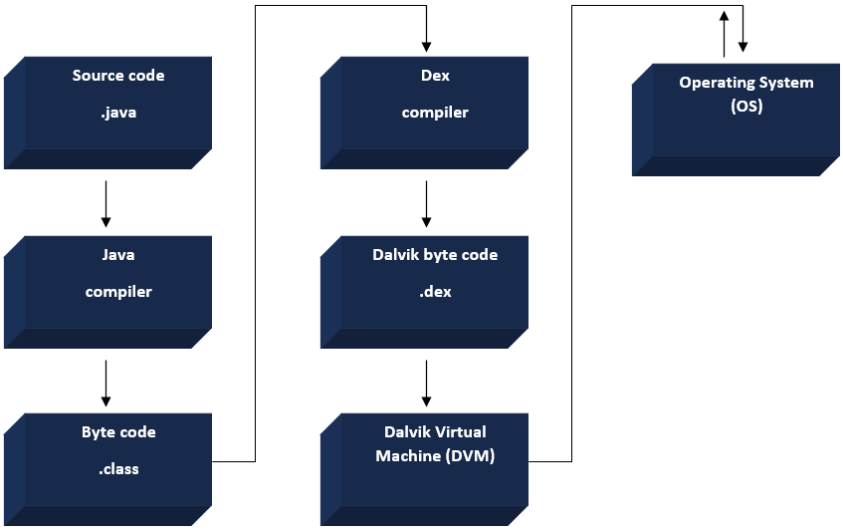
Popsané kroky je možné vidět na níže znázorněném obrázku 4.



Obrázek 4 - Java Virtual Machine

Dalvik Virtual Machine

Jedná se o virtuální stroj, který je určen pouze mobilním zařízením Android, který je podobný JVM. Na rozdíl od JVM je DVM založen na registrech, a to hlavně z důvodu, aby běžel s nízkou pamětí při primárním využití na mobilních zařízení Android. Hlavně z důvodu jeho rychlosti a využití méně paměti se tvůrci DVM rozhodli využívat DVM na úkor JVM [45]. Java Compiler (javac) převádí zdrojový kód Java na Java bajtkód (.class). Potom Dex compiler převede tento soubor (.class) na Dalvik Executable soubor (.dex) [47]. Na obrázku 5 je možné celý postup vidět.



Obrázek 5 - Dalvik Virtual Machine

2.1.5 Android Framework

Jedná se o skupinu tříd, rozhraní a dalšího předkompilovaného kódu jazyka Java, na kterém jsou postaveny aplikace. Části frameworku jsou veřejně přístupné prostřednictvím rozhraní Android API. Jiné části frameworku jsou dostupné pouze výrobcům OEM prostřednictvím systémových rozhraní API [48].

2.1.6 Apps

Aplikace jsou nejvyšší vrstvou architektury systému Android. Tyto aplikace jsou psány v jazycích Java nebo Kotlin a následně jsou zpracovány navazujícími systémy. V době běhu, aplikace využívá služby poskytované Android Frameworkem.

Vrstva Apps se skládá z aplikací napsaných týmem Androidu a z některých programů třetích stran stažených z Obchodu Play [49] a následně nainstalovaných v zařízení. Uživatelské rozhraní lze poměrně snadno a bez námahy přepracovat tím, že se umožní přístup vývojářů třetích stran k této vrstvě. Ve všech zařízeních je předinstalovaná řada standardních aplikací, jako např: aplikace pro přístup k telefonnímu systému, aplikace pro správu kontaktů, aplikace pro SMS klienty a aplikace pro webový prohlížeč. Lze napsat nové aplikace, které mohou nahradit stávající systémové aplikace. Android tak svým vývojářům poskytuje velké množství příležitostí [50].

2.2 Komponenty aplikace

Aplikační komponenty jsou základními stavebními kameny aplikace pro Android. Tyto komponenty jsou definovány souborem manifestu aplikace AndroidManifest.xml, který popisuje jednotlivé komponenty aplikace [51]. Každá aplikace má speciální startovní aktivity, které se poznají ve zmíněné souboru Androidmanifest.xml, jinak se jedná o způsob interakce s uživatelem. [52].

Existují čtyři typy komponent aplikace:

- Activities
- Services
- Broadcast receivers
- Content providers

Každý typ slouží odlišnému účelu a má odlišný životní cyklus, který definuje, jak je komponenta vytvořena a zničena. Následující části popisují typy komponent aplikace [41].

2.2.1 Activities

Aktivita je základní součástí architektury aplikace a funguje jako rozhraní mezi uživatelem a aplikací [50]. Aktivita představuje jednu obrazovku s uživatelským rozhraním, například seznam e-mailů nebo zobrazení pro čtení e-mailů v e-mailové aplikaci. Pokud má aplikace více aktivit, jedna z nich musí být označena jako startovní aktivita. Tato aktivita se zobrazí při spuštění aplikace jako první [51].

2.2.2 Services

Služba je jednou z kritických komponent aplikace, která může provádět dlouhotrvající operace na pozadí. Může po určitou dobu běžet, i když uživatel pracuje s jinými aplikacemi [52].

Existují tři typy služeb, které říkají systému, jak spravovat aplikaci a těmi jsou:

Foreground services

Provádí nějakou operaci, která je pro uživatele viditelná. Například zvuková aplikace použije službu na popředí k přehrávání zvukové stopy. Služby na popředí musí zobrazovat oznámení. Služby na popředí pokračují v činnosti, i když uživatel s aplikací neinteraguje [53].

Background services

Služby na pozadí jsou softwarové komponenty, které běží v pozadí bez viditelného uživatelského rozhraní. Jsou určeny k provádění úkolů, které nevyžadují interakci s uživatelem, jako je aktualizace aplikací, poslech hudby nebo stahování souborů. Tento typ služby běží obvykle nepřetržitě na pozadí. Mohou být spuštěny automaticky při startu zařízení a mohou být nastaveny k periodickému nebo trvalému běhu, dokud nejsou zastaveny. [53]

Bound services

Když je komponenta aplikace svázána se službou pomocí volání metody bindService(), je služba svázána. Vázaná služba nabízí rozhraní klient-server, které umožňuje komponentám komunikovat se službou, odesílat požadavky, přijímat výsledky a dokonce komunikovat mezi procesy pomocí meziprocesové komunikace (IPC). Vázaná služba běží pouze po dobu, kdy je k ní vázána jiná komponenta aplikace. Ke službě může být najednou vázáno více komponent, ale jakmile se všechny odváží, služba je zničena. [53]

2.2.3 Broadcast Reciever

Správce systémových balíčků zaregistruje reciever při instalaci aplikace. Reciever se pak stane samostatným vstupním bodem do aplikace, což znamená, že systém může spustit aplikaci a doručit broadcast, pokud aplikace není právě spuštěna [54].

Systém vytvoří nový objekt komponenty BroadcastReceiver, který bude zpracovávat každý broadcast recieve. Tento objekt je platný pouze po dobu trvání volání příkazu onReceive(Context, Intent). Jakmile se kód vrátí z této metody, systém považuje komponentu za již neaktivní [54].

2.2.4 Content providers

Content provider je primárně určen, když nějaká aplikace potřebuje poskytovat data či je naopak dostávat. Jedná se tedy o nástroj, pomocí něhož aplikace může nabízet data jiným aplikacím. Obvykle funguje na dvou scénářích, kterými jsou [55]:

- **Content provider –** Aplikace má content providera a nabízí data.
- **Content provider klient –** ten využívá přístupové rozhraní aplikace a může získávat data.

Content provider prezentuje data externím aplikacím jako jednu nebo více tabulek, které jsou podobné tabulkám v relační databázi. Řádek představuje instanci určitého typu dat, která poskytovatel shromažďuje, a každý sloupec v řádku představuje jednotlivé údaje shromážděné pro danou instanci.

Content provider koordinuje přístup k vrstvě ukládání dat v aplikaci pro řadu různých rozhraní API a komponent, mezi ně patří např. [55]:

- Sdílení přístupu k datům vaší aplikace s jinými aplikacemi,
- odesílání dat do widgetu,
- vracení vlastních návrhů vyhledávání pro vaši aplikaci prostřednictvím vyhledávacího rámce pomocí SearchRecentSuggestionsProvider,
- synchronizace dat aplikace se serverem pomocí implementace AbstractThreadedSyncAdapter,
- načítání dat v uživatelském rozhraní pomocí CursorLoader.

2.2.5 Soubor manifestu

Než může systém Android spustit komponentu aplikace, musí vědět, že komponenta existuje, přečtením souboru manifestu aplikace AndroidManifest.xml. Tento soubor obsahuje informace o APK balíčku, včetně součástí aplikace, jako jsou Activity, Services, BroadcastReciever, ContentProvider ad. [56]. Na obrázku 6 je vidět výstřižek znázorňující dané části v manifestu.

```
</service>
<service android:name="com.lib.gcm.MyFirebaseInstanceIDService">
    <intent-filter>
        <action android:name="com.google.firebase.INSTANCE_ID_EVENT" />
    </intent-filter>
</service>
<provider android:authorities="com.ustv.player.fileprovider" android:export
    <meta-data android:name="android.support.FILE_PROVIDER_PATHS" android:r
</provider>
<activity android:launchMode="singleTask" android:name="com.ustv.player.ui.
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
    </intent-filter>
</activity>
<receiver android:enabled="true" android:name="com.ustv.player.receiver.Boo
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED" />
    </intent-filter>
</receiver>
```

Obrázek 6 - Soubor manifestu

Jedná se tedy o naprosto klíčový soubor v mnoha ohledech, kterými jsou:

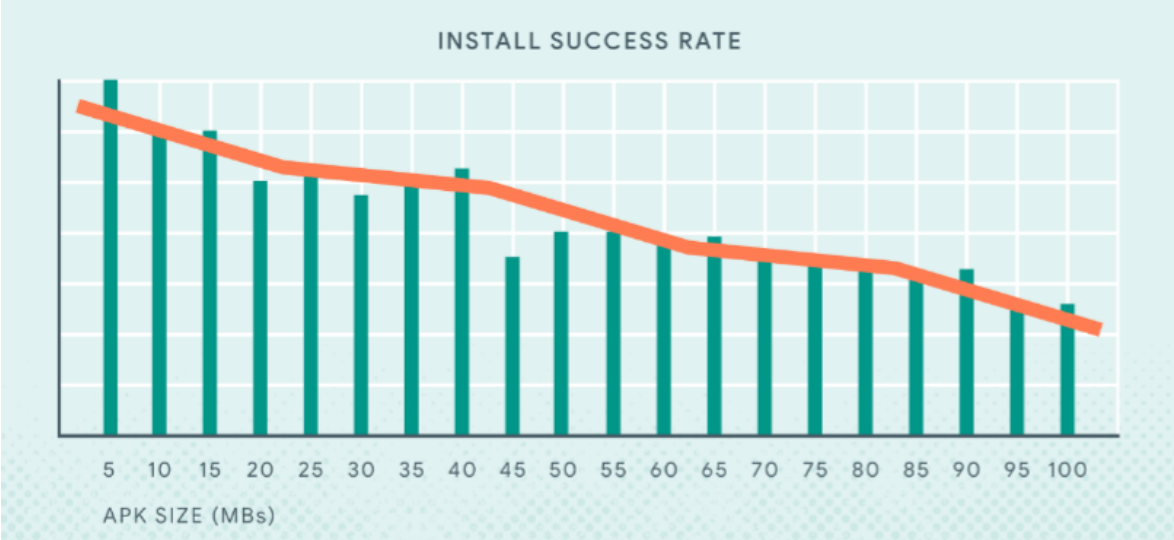
- Obsahuje zásadní informace pro chod aplikace, tudíž je potřeba k jejímu běhu,
- začíná zde většina penetračních testů, antivirových testů a malware útoků,
- analyzují jej bezpečnostní skenery a jedná se o místo, kde malware nemůže lhát.

2.3 Balíčky pro android

Operační systémy Android využívají od roku 2021 dva balíčky Android Appliacation Package (APK) a Android Application Bundle (AAB). AAB mají formát publikování, zatímco APK je formát, který bude nakonec nainstalován do zařízení. Google používá AAB ke generování a poskytování optimalizovaných souborů APK pro konfiguraci zařízení každého uživatele, takže si uživatelé stahují pouze aplikační logiku a zdroje, které potřebují ke spuštění aplikace [57]. Uživatelé tak mohou získat menší a optimalizovanější aplikaci. Tento postup je možné vidět na níže uvedeném obrázku 7.

Obrázek 7 - AAB a APK [57]

Původně byl APK balíček využíván pro distribuci, tak publikaci aplikací. Zde ale docházelo k problémům s jejich velikostí. To mělo za příčinu hlavně neustále rostoucí trend zvětšování aplikací, kvůli rozšiřování hardware portfolia a s tím spjatým přidáváním prvků. S tím ale také přicházely různé komplikace uživatelů operačního systému Android, kterým tyto aplikace zabíraly mnoho úložiště v zařízeních a stahování bylo příliš dlouhé a datově náročné. Díky uvedeným potížím poté docházelo ke klesající míře instalací větších aplikací [58]. Problém je možný sledovat na obrázku 8.



Obrázek 8 - Míra instalace u větších aplikací [58]

K řešení tohoto problému Google přistoupil zavedením AAB a od srpna roku 2021 je každý vývojář povinen publikovat nové aplikace na Google Play pomocí tohoto balíčku.

2.3.1 APK

Jak již bylo uvedeno výše, jedná se o balíček používaný k distribuci aplikací v operačním systému Android. Soubory balíčku nemají žádný speciální formát a jsou ukládány v formátu ZIP. APK obsahuje důležité soubory pro chod dané aplikace, jako jsou:

AndroidManifest.xml, který popisuje základní informace o aplikaci [59]. Více viz. oddíl 2.2.5. – Soubor manifestu.

Classes.dex - obsahuje aplikační logiku, která je nakonec spuštěna běhovým prostředím Android [60]. Více viz. oddíl 2.1.4 – Android Runtime

resources.arsc – jedná se o soubor, který obsahuje informace pro propojení logiky aplikace (classes.dex) se zdroji (res), například odkazy aplikační logiky na texty dialogů. Tyto texty jsou pak obsaženy ve zdrojích ve všech jazycích. Operační systém Android poté vybere správný jazyk na základě konfigurace lokálního zařízení [61].

META-INF – složka, jenž obsahuje soubory [62], znázorněno na obrázku 9:

- MANIFEST.MF – obsahuje pro každou položku JAR, chráněnou proti problémům s integritou, oddíl s odpovídajícím názvem, který obsahuje digest nekomprimovaného obsahu položky. Všechny tyto digesty jsou ověřeny [63].
- .SF – obsahuje souhrn celého souboru META-INF/MANIFEST.MF a výtahy každé sekce META-INF/MANIFEST.MF [63].

- .RSA – obsahuje všechny podpisy [64].

Obrázek 9 - Složka META-INF

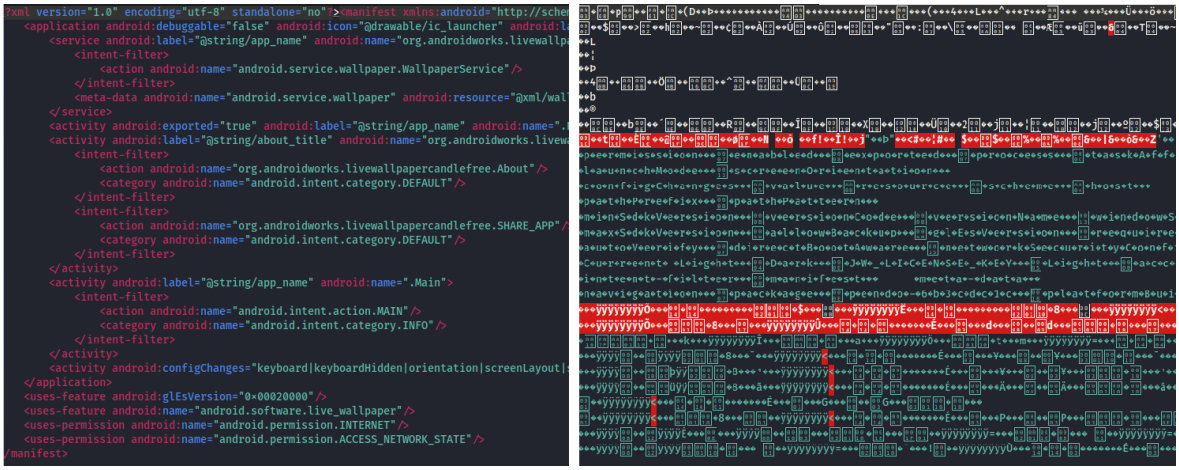
Adresář res zahrnuje mimo jiné:

- XML layout – jak jednotlivé aktivity budou vypadat,
- grafika – obrázky, ikony, apod.,
- řetězce ad [64].

Rozvržení balíčku je možné vidět na uvedeném obrázku 10.

Obrázek 10 - Soubory v APK

Jelikož se ale jedná o formát ZIP a aby bylo možné k jednotlivým souborům přistupovat, tak je nutné balíček dekomprimovat. K tomu jsou určené jednotlivé nástroje, jako jsou 7-zip, WinZip, FilZip ad. Přestože se tento postup zdá jednoduchý, tak není pro bezpečnostního analytika vyhovující, jelikož některé soubory, především AndroidManifest.xml, který obsahuje důležitá data, zůstává po dekomprimaci nečitelný. Z tohoto důvodu je pro statickou analýzu vhodné balíčky dekompilovat, čímž se stává čitelný. Následující rozdíl je možný vidět na obrázku 11.



Obrázek 11 – Struktura manifestu při dekompilaci a dekomprimaci balíčku

Dekompilace probíhá pomocí některého z APK dekompilačních nástrojů, jako jsou např. JADx, Android Decompiler, Apk Charger ad. [65]. Avšak nejvíce rozšířeným nástrojem je APKTool. Jedná se o nástroj reverzního inženýrství APK souborů. Disponuje dvěma možnými operacemi, kterými jsou dekompilace (d – decode) a znovu sestavení balíčku pomocí kompilace (b – build), což je možné vidět na obrázku 12.

```
$ apktool d test.apk
I: Using Apktool 2.7.0 on test.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: 1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
$ apktool b test
I: Using Apktool 2.7.0 on test
I: Checking whether sources has changed...
I: Smaling smali folder into classes.dex...
I: Checking whether resources has changed...
I: Building resources...
I: Building apk file...
I: Copying unknown files/dir...
```

Obrázek 12 - Nástroj APKTool [54]

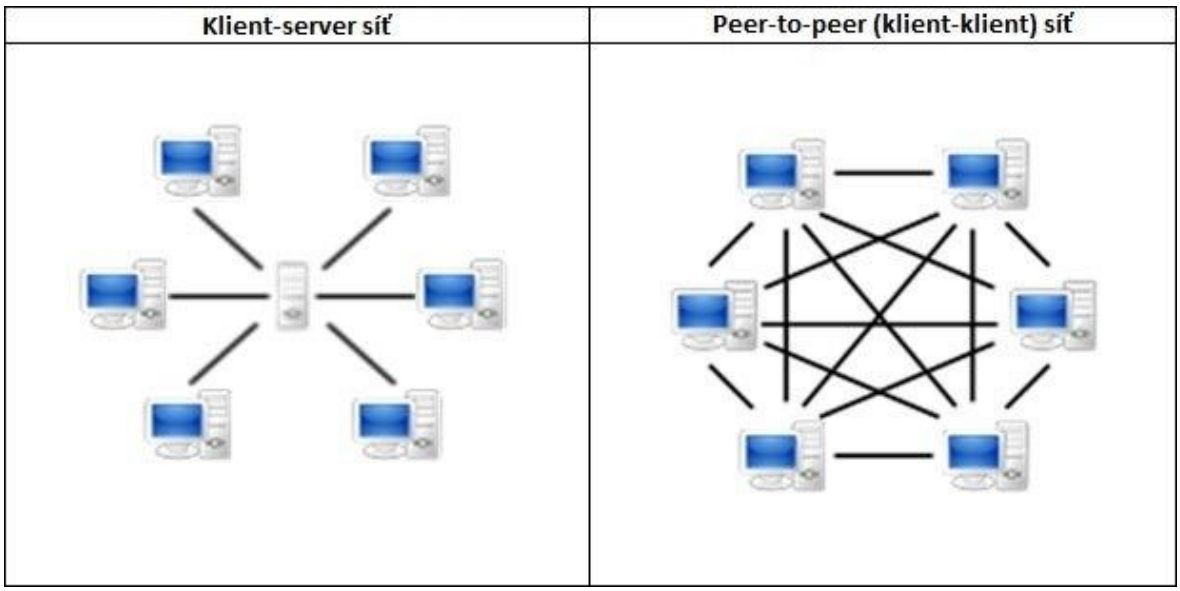
3 ZDROJE APLIKACÍ

Tato kapitola se zabývá pohledem na rozdělení, popis a rizika zdrojů aplikací, jež následně budou využívány při tvorbě datové sady. Jak již bylo zmíněno v podkapitole 1.1 – Analýza stávajících přístupů k řešené problematice, tak využití zdrojů aplikací v problematice tvorby APK datové sady se v odborné literatuře téměř nevyskytuje.

V tomto případě je možné aplikační zdroje rozdělit na dvě skupiny, kterými jsou torrentové stránky a file share servery.

3.1 Torrenty

Soubor metadat obsahující informace o sdílených souborech, jejich velikostech a kontrolních součtech se nazývá torrent. Sdílení probíhá prostřednictvím decentralizované sítě P2P (peer-to-peer). Tento typ sdílení souborů umožňuje uživatelům vyměňovat si soubory bez nutnosti jejich nahrávání na centrální server [66]. Každý takový uživatel je součástí daného procesu stahování se nazývá leecher. Na druhou stranu uživatel, jenž poskytuje souboru je charakterizován jako seeder. Tyto dvě skupiny dohromady tvoří roj (swarm). S tím tedy přichází výraz torrenting, jenž charakterizuje tento akt procesů mezi jednotlivými uživateli. Díky této povaze je rychlost stahování mnohem rychlejší než u serverů, jelikož není závislá na rychlosti centrálního serveru, ale na počtu peerů, jenž se procesu účastní [67]. Následující obrázek 13 poukazuje na rozdíl mezi jednotlivými zmíněnými sítěmi.



Obrázek 13 - Komunikace přes server vs P2P síť [68]

Důležitým prvkem pro P2P distribuci je BitTorrent. Jedná se o internetový přenosový protokol, jenž zaručuje právě distribuovaný přenos mezi více uživateli. Komunikační

protokol P2P, jako je BitTorrent, rozděluje soubory na části a přesouvá je od odesílatelů (seeders) ke stahovačům (leechers) prostřednictvím torrentového klienta. Tento klient, jenž daný protokol implementuje, se vyskytuje pod stejným názvem BitTorrent. Slouží ke čtení všech informací v souboru .torrent a spojuje uživatele za účelem výměny dat [69].

Torrentový soubor neobsahuje obsah určený k distribuci, ale obsahuje pouze informace o těchto souborech, jako jsou jejich názvy, struktura adresářů a velikosti získané pomocí hodnot hashovací funkce pro ověření integrity souborů [70]. Strukturu torrentového souboru je možné vidět na obrázku 14.

```
{
    'announce' :  'http://bttracker.debian.org:6969/announce' ,
    'info' :
    {
        'length' :  678301696 ,
        'name' :  'debian-503-amd64-CD-1.iso' ,
        'piece length' : 262144 , 'pieces' : <binary SHA1 hash> } }
```

Obrázek 14 - Struktura torrentového souboru [70]

Soubory torrent slouží jako obsah (index), který umožňuje počítačům vyhledávat informace klientem BitTorrent. Uživatelé, kteří si již soubor stáhnuli a sdílí jeho části (peers) umožňují stahování souborů souběžně. Soubory torrent jsou obvykle pojmenovány příponou ".torrent". Účelem torrentových souborů a jejich používání je odlehčit centralizovaným serverům. Namísto odesílání souborů na vyžádání umožňují hromadné využití šířky pásma potřebné pro přenos souborů, čímž zkracují dobu potřebnou ke stažení velkých souborů [70].

3.2 Torrentové stránky

V dnešní době existuje mnoho webových stránek, které indexují torrenty, tato podkapitola se zabývá jen malou skupinou těch, které jsou dle statistik z roku 2022 považovány za nejlepší [71].

Torrent_1: Jedná se o nejznámější torrentovou stránku, za jejímž vznikem roku 2003 stojí švédská anti-autorská organizace Piratbyrån ve složení Peter Sunde, Fredrik Neij, Gottfrid Svartholm a Carl Lundström [72]. I přes několik problémů, kterým tato stránka v průběhu své existence čelila, kvůli možnému porušování autorských práv se i nadále řadí mezi nejnavštěvovanější a nejoblíbenější torrentovou stránku [73].

Torrent_2: Jedná se obecný indexer torrentů s většinou ověřených torrentů, který vznikl roku 2009. Pyšní se pravidelnou aktualizací obsahu díky nichž se řadí mezi jedny

z nejlepších torrentových stránek, co se týká bezpečnosti. Dokonce disponuje pravidelnými seznamy, na nichž se objevuje stovka nejpopulárnějších a nejstahovanějších souborů. I přes to, že není tak oblíbený, jako některé jiné torrentové stránky, což je možná zapříčiněno malému počtu seederů, tak jeho počet souborů je dokonce větší než u Torrent_1 [74].

Torrent_3: Z této skupiny se jedná o nejnovější torrentovou stránku, jenž vznikla roku 2013. I přes to, že její existence není tak dlouhá, tak je známá jako jedna z největších torrentových stránek s více než 3,5 miliony ověřených torrentů a další tisíce torrentů jsou přidávány každý den. Torrent_3 se pyšní velkým množstvím her a aplikací určených především na osobní počítače a mobilní telefony [75].

Torrent_4: Další využívaná torrentová stránka vznikla roku 2010. Torrent_4 se pyšní svojí bezpečností, a to nejen slovy. Stránka platí svým uživatelům 1 USD za každý nahlášený falešný torrent. Ve své knihovně má zejména velké množství hudby, filmů, seriálů a elektronických knih, ale je zde možné stáhnout i jiné typy souborů, jako například aplikace [71].

3.3 File share servery

Stránky pro sdílení souborů zdarma nebo za poplatek poskytují uživateli možnost stažení digitálních médií, fotografií, aplikací ad., za předpokladu připojení k síti. Jedná se o síťový model klienta a serveru, kde počítače, jako jsou servery, poskytují síťové služby ostatním počítačům, zvaným klienti, pro provádění uživatelských úkolů (stahování + sledování multimediálních souborů). Model je známý jako síťový model klient-server [76]. Tento model je možné sledovat na dříve znázorněném obrázku 13 v porovnání s P2P sítí v podkapitole 3.1 – Torrenty. Jak již bylo uvedeno, tak do procesu vstupují dvě strany, kterými jsou klient-server.

Klient je program, jenž běží na lokálním stroji a žádá o službu server. Tento program je spuštěn zahájením činnosti na serveru a ukončen jejím dokončením.

Server je služba, jenž běží na vzdáleném počítači a nabízí službu klientům, kterou po požadavku od klienta poskytne. Tato služba je spuštěna bez časového omezení, až dokud ji nepřeruší nějaký problém či vypnutí serveru.

Model klient-server by se měl tedy řídit některými strategiemi, mezi které patří [77]:

- Serverový program běží vzdáleně, zatímco klientský program běží lokálně.

- Serverový program je neustále v běhu, zatímco klientský program běží pouze v době potřeby dané služby ze serveru.
- Mnoho klientů může využít služby jediného serveru (vztah many-to-one), přičemž musí být kontrolován počet připojených klientů, aby nedošlo k přetížení serveru.

3.3.1 Využívané file share servery

V dnešní době existuje mnoho služeb, ze kterých je možné neoficiálně stahovat soubory. Díky statistickému zpracování z roku 2016 jsou níže některé z českých služeb popsány [78]. Mezi nejrozšířenější služby patří:

Fileshare_1: Jedná se o hostingový web pro bezplatné stahování souborů. Na tomto serveru je možné bezplatné sériové stahování pouze jednoho souboru, než bude možné začít stahovat další, s omezenou rychlostí. Avšak některé soubory bez prémium účtu není možné stahovat vůbec. Při zakoupení prémium účtu je možné stahovat soubory paralelně s neomezenou rychlostí [79].

Fileshare_2: Jedná se o vůbec první web svého druhu. Bezplatné stahování s limitací aktuálně jednoho souboru, jako u Fileshare_1 sevreru, probíhá rychlostí 340kb/s [80].

Fileshare_3: Jedná se o jeden z nejnavštěvovanějších webů v České republice vůbec. Je zde možné stahovat různorodý obsah, ať už se jedná o filmy, muziku či aplikace. Princip bezplatného stahování je totožný, jako u Fileshare_1 serveru, a to s omezenou rychlostí 300kb/s. Pokud si klient koupí kredity, tak může docházet k paralelnímu stahování s neomezenou rychlostí [80].

Fileshare_4: Jedná se o službu, kde je možné soubory stahovat, ale také slouží jako videoportál, na kterém lze přímo sledovat filmy, seriály apod. Na tomto serveru si lze vybrat 3 druhy úrovní. První úroveň u které není nutná registrace disponuje rychlostí stahování 128kb/s. Pokud dojde k registraci (úroveň dva), tak je možné soubory stahovat s omezenou rychlostí 512kb/s. Třetí variantou je prémiový účet, jenž lze pořídit na časově omezenou dobu nebo na konkrétní velikost stažených dat. V této variantě je možné stahování bez omezení rychlosti [78].

4 VIROVÉ SKENERY

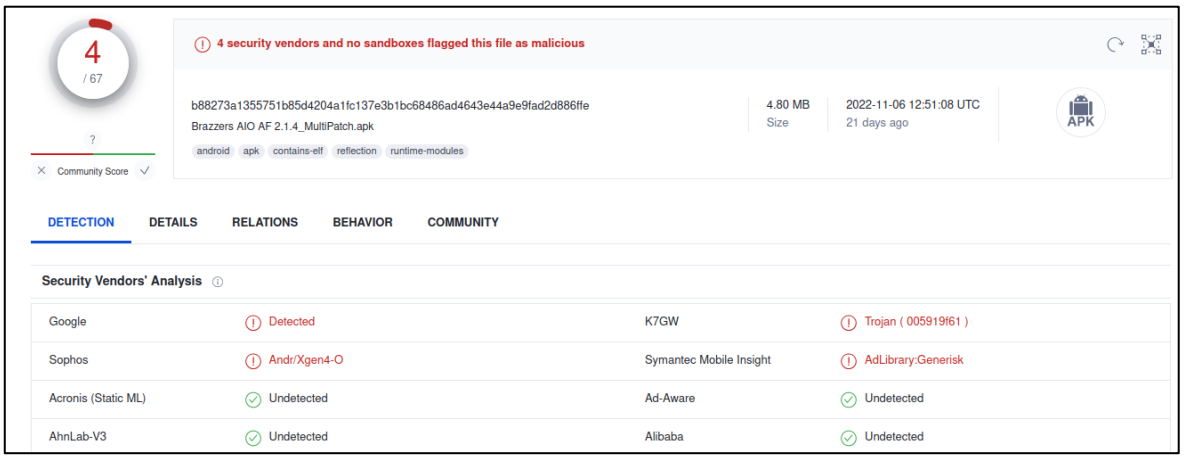
Tato kapitola se zabývá virovými skenery, které slouží k určení, zda je aplikace benigní či maligní (infikovaná). Jsou to služby, do kterých je možné nahrát libovolnou aplikaci či její zdroj, jenž je na základě signatur analyzována pomocí vybraných antivirových programů. Mezi nejznámější volně přístupný virový skener se bezpochyby řadí VirusTotal.

VIRUSTOTAL: Jedná se službu, založenou roku 2004, její část je bezplatná. Tato služba je schopna analyzovat soubory na přítomnost škodlivého obsahu. VirusTotal kontroluje položky pomocí více než 70 antivirových skenerů a služeb pro blokování adres jednotného lokátoru zdroje/domény, navíc s nesčetnými nástroji pro extrakci signálů ze studovaného obsahu [81]. Každý uživatel může vybrat soubor ze svého zařízení pomocí prohlížeče a odeslat jej do VirusTotal, což je možné vidět na obrázku 15.

Obrázek 15 - Nahrávání souborů do VirusTotal [81]

VirusTotal nabízí řadu metod odesílání souborů, včetně primárního veřejného webového rozhraní, desktopových uploaderů, rozšíření prohlížeče a API. Webové rozhraní má nejvyšší prioritu skenování mezi veřejně dostupnými způsoby odesílání. Příspěvky mohou být skriptovány v jakémkoli programovacím jazyce pomocí veřejného API založeného na http [81].

Po nahrání souboru proběhne jeho kontrola několika antivirovými programy. Výsledkem je následně výpis detailů souboru, jeho hash otisk pomocí SHA-256, kdy byl soubor vytvořen a komunitní skóre, do kterého uživatelé VirusTotalu mohou přidávat komentáře k danému souboru. Tento celkový výpis je možný vidět na obrázku 16.



Obrázek 16 - Výpis infekčnosti souboru

VirusTotal se pyšní svojí aktuálností a to díky aktualizaci v reálném čase. Signatury malwaru jsou často aktualizovány společností VirusTotal, protože jsou distribuovány antivirovými společnostmi, což zajišťuje, že služba používá nejnovější sady signatur.

VirusTotal není jedinou službou, která se zaujímá touto problematikou. Existuje i mnoho dalších, podobných aplikací, které řeší infekčnost nahraných souborů. Mezi další využívané skenery spadají [82]:

- Safety Detective Scanner zranitelností, ESET Scanner,
- F-Secure Scanner, Bitdefender Virus Scanner,
- Norton Security Scan, Bullguard Virus Scan ad.

Většina těchto skenerů požadují stažení a zakoupení licencí, což ani jedno u VirusTotalu není potřeba. Ovšem na rozdíl od VirusTotalu tyto skenery dokáží nejen zjistit zranitelnost daného problému, ale vzorek vyhodnotí a poskytnou souhrnnou zprávu o daném problému a v případě akutní hrozby zařízení jej dokážou odstranit [82].

5 HASHOVACÍ FUNKCE

Tato kapitola se zabývá hashovacími algoritmy a hash otisky, které jsou pro praktickou část a vyhodnocování duplicit souborů klíčové. Hashovací funkce převádí libovolně dlouhý vstup na výstup, který má fixní délku [83]. Takto vytvořenou zprávu nazýváme hash nebo otisk původního vstupu a má několik důležitých vlastností:

- Hashovací funkce pro daný vstupní řetězec vygeneruje vždy stejný výstupní hash.
- Hashovací funkce pro různě dlouhá data vždy produkuje hash o stejné délce.
- Z výstupního hashe nelze nijak získat původní vstupní data, tj. jednosměrnost.
- Nelze nalézt různá vstupní data pro tentýž výstupní hash (pokud by k tomu došlo, vzniká tzv. kolize a hashovací funkce se považuje za nebezpečnou).
- Malá změna na vstupu udělá velkou změnu na výstupu, tj. lavinový efekt.

Hashovací funkce prošly značným vývojem, a to především díky zranitelnostem, které měly. V následující tabulce 4 je možné vidět přehled jednotlivých známých hashovacích algoritmů.

Tabulka 4 - Přehled hashovacích algoritmů [84]

Hashovací algoritmus	**Rok vydání**	**Bitová velikost**
MD5	1992	128
SHA-1	1995	160
SHA-2	2002/2004	224,256,384 nebo 512
SHA-3	2008	Libovolná
BLAKE	2008	224,256,384 nebo 512
BLAKE2	2012	Max. 512
BLAKE3	2020	Libovolná (256 – výchozí)
RIPEMD	1992	128,160,256 nebo 320

Níže je detailněji popsáno využití následujících algoritmů z tabulky 4.

MD5 – Původně se sice používal pro bezpečnostní účely, časem se ale ukázalo několik zásadních zranitelností a dnes už je MD5 pro bezpečné využití nedostatečný. Hlavní zranitelnost MD5 představují případy, v nichž je dvěma rozdílným vstupním řetězcům

přidělen totožný hash. Vznikají tak kolize, které je navíc možné dohledat i na běžně výkonném počítači. Přestože jej bezpečnostní experti jednoznačně označili jako nevhodný, stále je na mnoha stránkách pro mnohé účely používán [84].

Rodina SHA algoritmů – První variantou této rodiny je SHA-1, která nahradila již nebezpečný MD5, ale stále byla založena na stejných principech. Postupem času se zjistilo, že SHA-1 je náchylná ke kolizím a není vhodná pro bezpečnostní účely. Národní institut pro standardy a technologie (National Institute of Standarts and Technology - NIST) doporučil nahradit algoritmus SHA-1 algoritmem SHA-2, který splňuje bezpečnostní požadavky a netrpí kolizemi. Následně byl v roce 2015 vytvořen nejnovější algoritmus z rodiny SHA, nazvaný SHA-3. SHA-3 není odvozen z předchozích algoritmů SHA, ale vychází z algoritmu Keccak a využívá jeho sílu ve vícekolové permutaci. Cílem algoritmu SHA-3 není nahradit dosud bezpečný algoritmus SHA-2, ale spíše jej doplnit a přinést nové možnosti [85].

Blake – Algoritmy z rodiny BLAKE zpracovávají vstup v několika kolech. Hlavní rozdíl mezi novým BLAKE3 (7 kol) a jeho předchůdci BLAKE2 (12 kol) a původním algoritmem BLAKE (16 kol) spočívá v počtu kol. Díky tomuto rozdílu počítá BLAKE3 výslednou hash mnohem rychleji, aniž by byla ohrožena bezpečnost. [85].

Další hashovací algoritmy – Existuje mnoho dalších hashovacích algoritmů, kterými jsou například RIPEMD, jenž vychází z podobných principů jako MD5 či SHA-1. Díky stále nedostatečně zpracovaným bezpečnostním analýzám tyto algoritmy nejsou tak využívány, jako algoritmy rodiny SHA [85].

V nynější době mezi nejrozšířenější hashovací funkci bezpochyby spadá SHA 256, u které prozatím nedochází ke kolizím [86]. Z tohoto důvodu je následně využita v praktické části při hledání duplicit jednotlivých vzorků.

Bohužel kvůli vlastnosti, která zaručuje při jakékoli změně vstupu vznik zcela odlišného výstupního otisku, jsou klasické hashovací funkce při bezpečnostní analýze v některých případech nedostačující. Z tohoto důvodu je nutné využití pokročilejších funkcí, mezi které se bezpochyby řadí fuzzy hashovací funkce, jejichž vlastnosti jsou popsány v následující podkapitole 5.1 – Fuzzy hashovací funkce.

5.1 Fuzzy hashovací funkce

Jedná se jako v předchozím případě o funkce, které z libovolného vstupu vypočítají výstupní otisk. Na rozdíl od klasických hashovacích funkcí mají určitou toleranci ke změnám v daném souboru. Při porovnání dvou odlišných souborů není vyhodnocována celková duplicita, ale míra podobnosti v procentech. [87]. Tento krok usnadňuje analytikům řešení v oblasti malwaru, a to hlavně z důvodu, že jednotlivé využívané infikované soubory mají více variant, které provádějí přesně stejnou sadu chování a spadají do stejné rodiny [88]. Nicméně v tomto případě se výstupní otisky částečně liší, což by klasické hashovací funkce vyhodnotily jako zcela unikátní vzorky. V tomto ohledu je výhodou právě vlastnost fuzzy hashovacích funkcí, které dokážou detekovat částečnou podobnost a vyhodnocovat ji v procentech. Mezi nejrozšířenější fuzzy hashovací algoritmy patří ssDeep. Princip tohoto algoritmu spočívá v tom, že vypočítá hash pro každý fragment dodaných dat [89]. Výsledný otisk je pak ve specifickém formátu:

$$\boldsymbol{Velikost\ bloku : Hlavni\ hash : Vedlejsi\ hash}$$

Velikost bloku: reprezentuje velikost bloku, který byl použit při generování fuzzy hashe. Tato hodnota je odvozena ze vstupního souboru a je vypočítána jako $3 \cdot 2^n$, kde n je celé číslo a je zvoleno tak, aby velikost bloku byla nejmenší hodnotou, která je větší nebo rovna velikosti vstupního souboru v bajtech dělené 64. [90]

Hlavni hash: představuje samotný fuzzy hash souboru, který byl vypočítán pomocí rolling hashe založeného na funkcích Adler-32 a Fowler-Noll-Vo. Rolling hash se používá k identifikaci trigger bodů v souboru, které se vypočítávají na základě posuvného okna sedmi bajtů. Poté se mezi dvěma trigger body vypočítají hashovací hodnoty, které jsou zakódovány jako řetězce Base64 a vytvoří fuzzy hash. [90]

Vedlejsi hash: reprezentuje druhý fuzzy hash, který byl vygenerován stejným postupem jako první hash, ale s použitím dvojnásobné velikosti bloku. Tento druhý hash se nazývá vedlejší hash a slouží k porovnání souborů, které mají odlišné velikosti bloků a mohou mít podobné fuzzy hashe. Tento přístup umožňuje ssdeep srovnávat a hledat shody mezi soubory s vyšší úspěšností. [90]

Pro demonstraci je níže uvedený konkrétní výraz.

98304:3yy32niTlfO7gUpYh/Dgb0zPKP1Ht0t90oAIgw55:**332niTlG7gR/Eb0ONHtE**

Ve výrazu je možné sledovat tři dané části:

- Číslo v rámečku charakterizují danou velikost bloku, kterou je 98 304b
- Podtržená část znázorňuje hodnotu hlavního hashe
- Tučně zvýrazněná část znázorňuje hodnotu vedlejšího hashe

Z tohoto tvaru je zřejmé, že porovnání výsledných hodnot hash funkce může být učiněno pouze za předpokladu, že velikost bloku u souborů je stejná, jejím *n* násobkem či podělena číslem *n*, kde *n* je přirozené číslo. To z důvodu, že pokud by tomu tak nebylo, tak výsledný otisk je naprosto odlišný a podobnost činí 0 %.

6 ANALYTICKÉ METODY PŘI DETEKCI MALWARU

Tato kapitola se zabývá analytickými metodami. Mezi využívané metody při detekci malwaru se především řadí statická a dynamická analýza, které jsou následně popsány v dalších podkapitolách. Dále také dochází k využívání hybridní analýzy, která je kombinací právě těchto dvou za účelem překonání jednotlivých nedostatků každé z nich [91].

6.1 Statická analýza

Statická analýza, mnohdy nazývána analýza statického kódu, zkoumá aplikační logiku popsanou pomocí zdrojových kódů a instrukcí pro virtuální stroj, aniž by muselo dojít k jeho spuštění [92]. Díky tomuto kroku disponuje značnou výhodou vůči analýze dynamické, jelikož je možné analyzovat veškerý kód aplikace [93]. Dochází tedy k postupnému prozkoumávání všech jednotlivých částí souborů, jako je AndroidManifest.xml, classes.dex, resources ad. Mezi nevýhody se především řadí její náročnost na výpočetní výkon, jelikož v mnoha případech se jedná o spuštění komplexních modelů (softwarů) založených na umělé inteligenci (vstupní vektory s velkým množstvím složek). A nutné také zmínit její provázání s metodami umělé inteligence, které v mnoha případech usnadňují analytikovi klasifikovat škodlivý kód [94]. Další nevýhoda ruční statické analýzy je velká časová náročnost, a vysoké odborné nároky kladené na analytika. Při procesu statické analýzy dochází k pochopení struktury daného kódu a zjištění jednotlivých anomálií, které mohou vykazovat různé infekce. Mezi jednotlivé anomálie se např. řadí chyby v kódu, nedefinované hodnoty, povolená oprávnění, nalezení spustitelných souborů v APK ad.

V praktické části diplomové práce byla statická analýza využita na vlastně vytvořenou datovou sadu z neoficiálních zdrojů, kterými jsou torrentové stránky a file share servery.

6.2 Dynamická analýza

Dynamická analýza je opakem statické analýzy, kdy je vyhodnocováno chování aplikace v reálném čase při jejím běhu. Z tohoto důvodu je nutné při analýze infekčnosti provádět daný typ analýzy v kontrolovaném prostředí [95]. S tímto procesem jsou spjaté hlavní nevýhody, kterými jsou emulace lidského chování a hrozba škodlivého kódu, který by mohl uniknout z kontrolovaného prostředí [96]. Naopak od statické analýzy zde dochází ke spuštění aplikací v emulátoru, u čehož není kladen důraz na velkou výpočetní náročnost.

II. PRAKTICKÁ ČÁST

7 TVORBA DATOVÉ SADY

Tato kapitola se zabývá vytvořením datové sady pro její další využití. Jelikož se práce zaměřuje na bezpečnostní analýzu stažených APK prostřednictvím torrentů a file share serverů, tak v prvním kroku bylo nutné udělat selekci stránek těchto zdrojů. Vybrány byly celkově 4 torrentové stránky a 3 file share servery. Detailnější informace o fungování těchto zdrojů aplikací jsou vysvětleny v kapitole 3 – Zdroje aplikací. Poté byly v těchto zdrojích vyhledány APK s využitím klíčových slov nebo možností vyhledávání v kategoriích. Aby mohla být provedena kvalitní analýza, tak bylo nutné mít dostatečný počet vzorků. Konkrétně tedy bylo dohromady staženo 1771 vzorků, a to 661 prostřednictvím file share serverů a 1110 prostřednictvím torrentů. Kvůli anonymitě konkrétních zdrojů byly datové sady rozděleny do jednotlivých skupin znázorněných v následující tabulce 5:

Tabulka 5 - Zdroje aplikací s danými vzorky

Zdroj aplikace	**Počet vzorků**
Fileshare_1	92
Fileshare_2	319
Fileshare_3	247
Torrent_1	322
Torrent_2	247
Torrent_3	100
Torrent_4	441

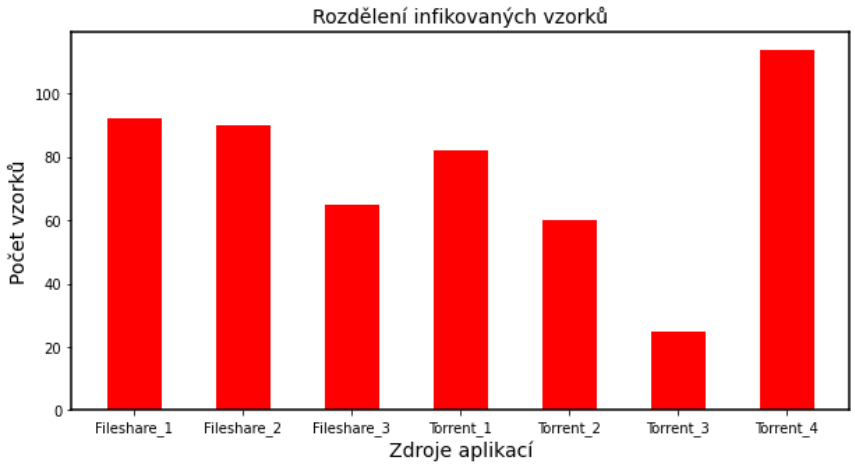
Z hlediska bezpečnosti je důležité, aby ke stahování daných souborů docházelo ve virtuálním prostředí nebo při nejmenším mít dostatečně kvalitní antivirovou ochranu. Důvodem je, že i když se jedná o aplikace určenou pro mobilní telefony operačního systému Android, tak mohou obsahovat infikovaný kód, který může být spuštěn na operačním systému osobního počítače. Na obrázku 17 je možné tento problém zpozorovat.

Obrázek 17 - Hrozba při stahování aplikace

Jak je z obrázku vidět, jedná se o hrozbu s názvem Trojan:Win32/AgentTesla!ml. Malware AgentTesla lze klasifikovat jako spyware. Tento druh malwaru se používá ke sledování různých částí činnosti oběti a shromažďování některých citlivých dat. V budoucnu mohou podvodníci tato data (hesla, osobní konverzace, důležité soubory) prodávat v Darknetu nebo je použít k vydírání oběti. Existuje mnoho případů, kdy byl Trojan:Win32/AgentTesla!ml distribuován společně s ransomwarem [97].

7.1 Kontrola infekčnosti vzorků

Následně byla všechna data vyhodnocena pomocí virového skeneru, a to zda se jedná o benigní aplikaci nebo malware. K tomuto kroku byl využit virový skener VirusTotal, jenž je popsán v kapitole 4 – Virové skenery. Po prozkoumání celkem 1771 vzorků bylo zjištěno, že 520 z nich bylo identifikováno alespoň jedním antivirovým programem jako potenciálně infikované aplikace. Zbývajících 1251 vzorků bylo klasifikováno jako bezpečné, tedy benigní aplikace. Konkrétní rozložení infikovaných aplikací dle jejich zdrojů je vidět na obrázku 18.



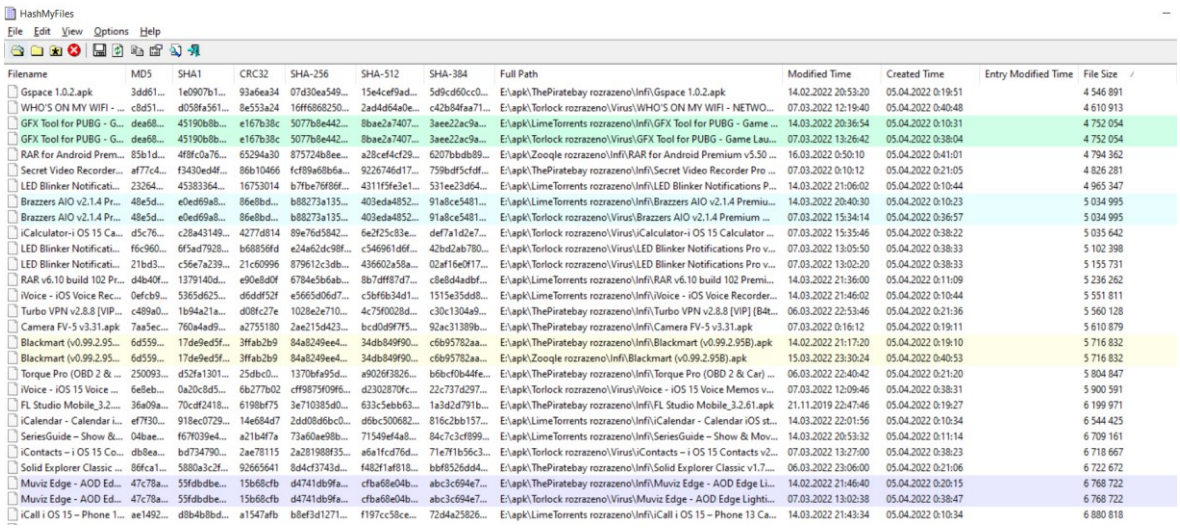
Obrázek 18 - Rozložení infikovaných vzorků

8 KONTROLA UNIKÁTNOSTI VZORKŮ

V této kapitole je popsána kontrola jednotlivých souborů, a to konkrétně práce s jejich hash otisky. Díky tomu byla zajištěna kontrola, zda při stahování z vybraných zdrojů nedošlo ke stažení APK, které mají sice jiný název, ale jejich obsah je totožný. K tomuto úkolu byly použity dvě metody, kterými jsou kontrola unikátnosti pomocí klasických hashovacích funkcí a pomocí fuzzy hashovacích algoritmů.

8.1 Klasické hashovací algoritmy

V rámci tohoto kroku pro kontrolu a porovnání hashů byla použita aplikace HashMyFiles. Tato aplikace provedla výpočet hashových otisků. HashMyFiles umožňuje uživateli vypočítat hashovací funkce dle vlastního výběru z poskytnutého seznamu. V aplikaci je k dispozici celkem šest různých hashovacích funkcí, včetně SHA-256, která byla v této situaci použita. Aplikace disponuje jednoduchým grafickým uživatelským rozhraním, do kterého pomocí tlačítka nahrajete soubory k následnému vyhodnocení. Po vyhodnocení také dochází k barevnému znázornění duplicit, které byly klíčové. Aplikace včetně konkrétního příkladu se znázorněním duplicit je možné vidět na obrázku 19.



Obrázek 19 - HashMyFiles

Pro kontrolu korektnosti výsledků byla také využita aplikace napsána v programovacím jazyce Python, která měla za úkol dané soubory taktéž porovnat. V prvním kroku byly zkoumány duplicitní soubory vždy z jednoho zdroje aplikací. Z výsledků je zřejmé, že jednotlivé stažené soubory, konkrétně z jednoho zdroje jsou duplicitní pouze u Fileshare_1. Zde z 92 infikovaných vzorků bylo pouze 12 unikátních a zbylých 80 tvořily aplikace, který sice měly jiný název, ale obsahovaly tutéž aplikaci. V následující tabulce 6 jsou znázorněny

některé případy těchto duplicit a počet antivirových programů, které tyto aplikace vyhodnotily za infikované.

Tabulka 6 - Duplicitní soubory

První název	**Druhý název**	**Počet detekcí**
Bulánci	Brawl Stars	15
AVAST Mobile APK Free CZ	AVG Antivirus 2019	24
Mozilla Firefox APK	MINER BITCOIN FREE CZ	29
Live For Speed 0.6G	LibreOffice 2019 APK	29
TotallComander APK	Minecraft Game Free CZ APK	32
Angry Birds	AnyMP4 Blu-ray Player 6.3.30	18

V tabulce 6 jsou pro demonstraci znázorněny pouze dva názvy duplicitních aplikací, ale duplicitních názvů mohou aplikace obsahovat více. Tyto vzorky byly vyhodnoceny velkým množstvím antivirových programů jako infikované. Z tohoto hlediska je možné říct, že dané aplikace s velkou pravděpodobností disponují infekcí, která může při stažení a následné instalaci zasáhnout uživatelovo zařízení. Jak je také z názvu aplikací zřejmé, tak se převážně jedná o hry, aplikace pro práci s kryptoměnou a antivirové programy. Právě tyto aplikace jsou velkým lákadlem jak z pohledu uživatele, který si je může stáhnout zdarma, ale také útočníka, jelikož zde může docházet k odcizení dat uživatele.

Následně je tedy možné říct, že u všech ostatních zdrojů aplikací se objevují pouze unikátní vzorky. V dalším kroku byly stejným způsobem zkontrolovány soubory mezi jednotlivými skupinami, do kterých se aplikace řadí, dle toho, z jakých zdrojů byly staženy. Aplikace z Fileshare_1, které obsahovaly velké množství duplicit, byly následně do analýzy vyfiltrovány a brány pouze jako jeden unikátní vzorek. Z této analýzy bylo vyhodnoceno 50 infikovaných aplikací, které jsou duplicitní mezi jednotlivými kategoriemi. Z toho 45 souborů je totožných i včetně názvu, zbylých 5 se mezi jednotlivými kategoriemi objevuje s názvem odlišným. Jednotlivé soubory s odlišným je možné vidět v následující tabulce 7.

Tabulka 7 - Duplicity s odlišným názvem mezi kategoriemi

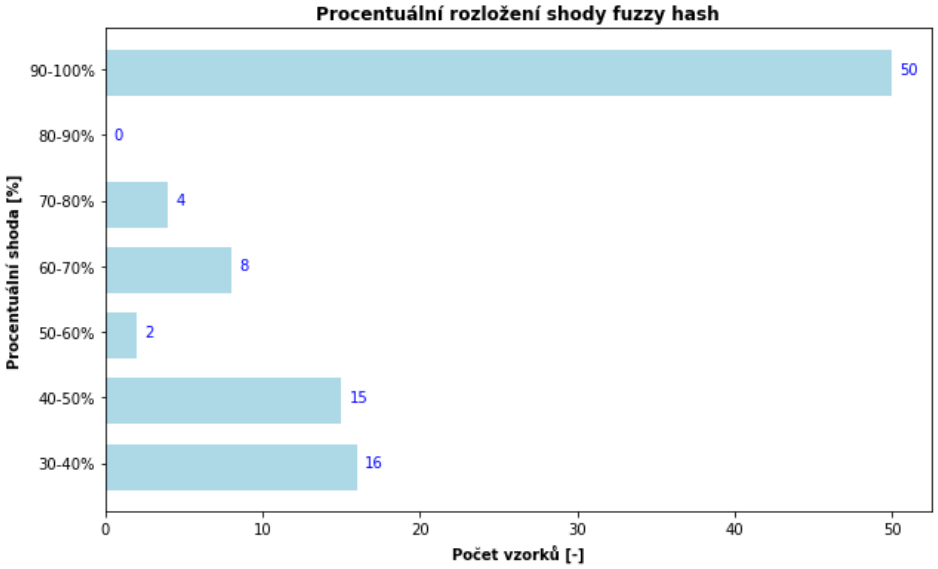
První název	**Druhý název**
Wifi Unlocker	HappyKids v5.8 Build77
GFX Tool for PUBG	Fifa Unlocker
Brazzers AIO v2.1.4 Premium	AdultsWebsite
Fake GPS Pro v4.9.4_build_66	Top Car Navigation Pro
Trusted VPN v10.9.0	Secure Web Search

Jak je z názvu některých vzorků zřejmé, tak se z části jedná o aplikace, které zaručují placenou funkcionalitu, ať už jsou to hry, které jsou placené nebo prémiové účty na stránky pro dospělé. Jedná se tedy o lákavé aplikace pro stahujícího a právě z tohoto důvodu se na tyto názvy útočníci zaměřují. Při důkladnějším zkoumání těchto vzorků bylo vyhodnoceno, že infekce vyskytující se v těchto aplikací jsou ve většině případů trojské koně, díky nimž útočník může získat citlivé informace o uživateli.

8.2 Fuzzy hashovací algoritmy

V této podkapitole byly využity fuzzy hashovací algoritmy ke zkoumání duplicitních infikovaných vzorků, či jejich částí. Jak již bylo řečeno v podkapitole 5.1 – Fuzzy hashovací funkce, tak fuzzy hashovací algoritmy jsou pro bezpečnostního analytika nástavbou klasických hashovacích algoritmů. To hlavně z důvodu, že klasické hashovací algoritmy dokážou rozeznat pouze naprosto totožné aplikace, zatímco fuzzy hashovací funkce jsou schopné najít shodu některých jejich částí, což může být klíčové při bezpečnostní analýze malwaru.

Pro tuto část byla využita fuzzy hashovací funkce ssDeep za pomocí programovacího jazyka Python. Pro účel stanovení minimální shody byla vytvořena předběžná analýza na základě datové sady, jejímž výsledkem bylo určení na 30%. Tímto způsobem byly porovnány všechny jednotlivé vzorky mezi všemi zdroji aplikací. Stejně jako v předchozím případu, tak duplicitní data ze zdroje Fileshare_1 byly vždy brány jako jeden unikátní vzorek. Vyhodnocení těchto vzorků ukázalo, že 95 vzorků obsahuje duplicitní části. Jednotlivé procentuální shody s počtem vzorků jsou znázorněny na obrázku 20.



Obrázek 20 - Procentuální rozložení shod pomocí fuzzy hashovacích funkcí

Z grafu je vidět, že pomocí fuzzy hashovacích algoritmů došlo k témuž nalezení 50 naprosto shodných aplikací, jako u klasických hashovacích funkcí. Avšak na rozdíl od toho je zde ještě 45 vzorků s částečnou shodou, které by mohly vykazovat tutéž infekci jen v jiné aplikaci. Z tohoto důvodu je vhodné se na všechny tyto soubory v následujících částech práce zaměřit.

9 ANALÝZA INFEKČNÍCH SOUBORŮ

Tato kapitola se zabývá analýzou aplikací, jež byly vyhodnoceny v pomocí VirusTotalu za infekční, viz. podkapitola 7.1 – Kontrola infekčnosti vzorků. Před tím, než k analýze mohlo dojít, tak bylo zapotřebí roztřídit dané potenciálně infekční aplikace do dvou skupin. Ze všech infekčních souborů byly vytvořeny dvě skupiny, a to podle toho z jakého zdroje aplikace byly staženy. V případě plně duplicitních vzorků, které byly vysvětleny v kapitole 8.1 – Klasické hashovací algoritmy, byly vybrány pouze unikátní vzorky. Dále dochází k rozdělení daných skupin na kategorie určující počet pozitivních detekcí antivirovými programy. Toto rozdělení je možné vidět v následující tabulce 8.

Tabulka 8 - Rozdělení infekčních aplikací

Zdroj aplikace	File share servery	Torrentové stránky
1-2 detekce	122	245
3+ detekcí	19	18

Jak již v předešlém textu bylo zmíněno, tak potenciálně nebezpečné aplikace byly mimo jejich zdroje také rozděleny do dvou skupin dle počtu detekcí. Důvodem tohoto rozdělení bylo, že aplikace s jednou či dvěma detekcemi mohou být mylně považovány za infikované a tím pádem mohou mít velmi malý detekční potenciál. Pro analýzu popsanou v této kapitole byla klíčová převážně druhá skupina, do nichž spadají s třemi a více detekcemi. Analýza probíhala v několika krocích:

- Analýza souboru Androidmanifest.xml

V této části došlo k prozkoumání souboru Androidmanifest.xml, ve kterém bylo řešeno pod jakým názvem je aplikace uložena, jaká jsou požadovaná oprávnění a v neposlední řadě pro jakou verzi Androidu je daná aplikace navržena.

- Analýza aplikačních zdrojů res

V této části došlo k analýze aplikačních zdrojů, při kterých budou hledány shody a potenciálně nebezpečné a nezvyklé části, jež se mohou v souborech objevovat.

- Analýza dex souboru

V této části došlo ke kontrole anomálií souborů classes.dex.

9.1 Analýza souboru Androidmanifest.xml

V této kapitole došlo k prozkoumání souboru Androidmanifest.xml, a to konkrétně k analýze názvu package, zjištění požadovaných oprávnění daných aplikací a názvu štítku (labelu). Na obrázku 21 je možné vidět barevně označené části, které právě tyto vlastnosti charakterizují.

```
package="com.android" platformBuildVersionCode="23" platformBuildVersionName="6.0-2438415">
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <application android:icon="@mipmap/ic_launcher" android:label="AndroidClient">
```

Obrázek 21 - Androidmanifest.xml - package a oprávnění

Kontrola názvu package

V prvním kroku tedy byly kontrolovány názvy package, což je jedinečný identifikátor aplikace v zařízení či v obchodě Google Play. Pokud se jedná o aplikace od společnosti Google LLC a jde o systémovou součást/aplikaci operačního systému Android, tak se v názvu vždy objevuje com.android. Konkrétním případem je například com.android.contacts, což je název pro aplikaci kontaktů. Z tohoto důvodu bylo nutné zkontrolovat, zda tvůrci aplikací z třetích stran nevyužívají právě tento název package.

Při prozkoumání všech potenciálně nebezpečných aplikací bylo zjištěno, že package v několika případech nevykazují žádné potenciálně nezvyklé názvy. Avšak v ostatních případech se převážně objevovaly dvě nestandartní formy názvu, kterými jsou:

- Nesrozumitelné názvy package

V šesti případech se v aplikacích objevovaly nesrozumitelné názvy package. Zde bylo navíc zkontrolováno, zda tento název není způsoben obfuskací. Jedná se o zcela nestandartní způsob pojmenování v porovnání vůči čistým aplikacím neobsahující malware. Záměrem vydavatele aplikací tedy může být maskování obsaženého malwaru pomocí těchto názvů. Jeden z případů je vidět na obrázku 22.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?><manifest
xmlns:android="http://schemas.android.com/apk/res/android"
package="com.dwnldpknw.cvlgwtt" platformBuildVersionCode="22"
platformBuildVersionName="5.1.1-1819727">
```

Obrázek 22 - Nesrozumitelný název package

- Obsažena oficiální část názvu package

Jak již bylo řečeno, oficiální názvy package mají pevně stanovenou strukturu pojmenování. V patnácti případech se u potenciálně nebezpečných aplikací opakovaně objevovaly stejné

dvě slova na začátku názvu package, která jsou právě typická pro oficiální aplikace. Tento případ je u stažených aplikací zcela nestandartní a tudíž zde dochází k falzifikaci jejich názvů package. Jeden z těchto konkrétních případů je možný vidět na obrázku 23.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?><manifest
xmlns:android="http://schemas.android.com/apk/res/android"
android:compileSdkVersion="29" android:compileSdkVersionCodename="10"
package="com.android.gspace." platformBuildVersionCode="29"
platformBuildVersionName="10">
```

Obrázek 23 - Názvy package obsahující oficiální část

Kontrola oprávnění

V druhém kroku došlo ke zjištění a následné analýze všech požadovaných oprávnění aplikací. Důvodem této analýzy je, že pokud chce vydavatel, aby aplikace měla přístup k některým chráněným částem nebo zdrojům zařízení běžícím pod operačním systémem Android, tak je nutné deklarovat příslušná oprávnění. Tak musí být právě i u aplikací, jenž potenciálně obsahují škodlivý kód. Při analýze byly zjištěny čtyři různé potenciálně nebezpečné kombinace požadovaných oprávnění. Těmi jsou:

- WRITE_EXTERNAL_STORAGE

V deseti případech se objevovalo pouze jedno oprávnění, kterým je právě WRITE_EXTERNAL_STORAGE. Díky tomuto oprávnění má vydavatel aplikace povolen přístup k externímu úložišti, kde je možné zapisovat a mazat data [98]. I přes to, že toto oprávnění se objevuje i v zcela čistých aplikacích, tak je to vždy v kombinaci s jinými oprávněními. Z tohoto důvodu je využití pouze daného oprávnění zcela nestandartní způsob a je možné aplikaci považovat za potenciálně nebezpečnou.

- READ_SMS, CALL_PHONE, SEND_SMS

Dalším výsledkem analýzy je, že ve čtyřech případech jsou v aplikaci opakovaně objevující se povolená oprávnění k využívání SMS, kontaktů, telefonních čísel, ale také externího úložiště. Díky tomuto oprávnění má vydavatel přístup ke kontaktům uživatele. Dále jsou povolena oprávnění, jenž poskytují všechny možné čtení komunikací. Ať už se jedná o čtení SMS pomocí READ_SMS, ale také READ_CALL_LOG, jenž umožnuje přistupovat do logů . Další oprávnění se zaobírají manipulací s kontakty bez vědomí uživatele. Těmi jsou CALL_PHONE, jenž poskytuje aplikaci volat na telefonní čísla bez interakce uživatele. Následně také SEND_SMS, jenž poskytuje možnost zasílání SMS zpráv bez vědomí uživatele. Následně se zde také objevují další potenciálně nebezpečná oprávnění, jako

QUERY_ALL_PACKAGES, které slouží k zobrazení aplikací, jež jsou v zařízení nainstalované a RECEIVE_BOOT_COMPLETED, má právo poslouchat, zda systém odeslal Broadcast, že zařízení bylo spuštěno [98]. Seznam často vyskytujících se oprávnění je možný vidět na obrázku 24.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.READ_CONTACTS" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.READ_PRIVILEGED_PHONE_STATE" />
<uses-permission android:name="android.permission.READ_SMS" />
<uses-permission android:name="android.permission.QUERY_ALL_PACKAGES" />
<uses-permission android:name="android.permission.READ_CALL_LOG" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.SET_WALLPAPER" />
<uses-permission android:name="android.permission.SEND_SMS" />
<uses-permission android:name="android.permission.CALL_PHONE" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.READ_PHONE_NUMBERS" />
<uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
```

Obrázek 24 - Oprávnění ohledně komunikací

- Příliš mnoho povolených oprávnění

Následně dalších pět aplikací vykazovalo nestandartní množství požadovaných oprávnění aplikace. Jedná se o aplikace, u kterých se objevuje počet požadovaných oprávnění v řádech několika desítek. Část tohoto příkladu je možný vidět na obrázku 25.

```
<uses-permission android:name="com.android.vending.BILLING" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.BROADCAST_PACKAGE_ADDED" />
<uses-permission android:name="android.permission.BROADCAST_PACKAGE_CHANGED" />
<uses-permission android:name="android.permission.BROADCAST_PACKAGE_INSTALL" />
<uses-permission android:name="android.permission.BROADCAST_PACKAGE_REPLACED" />
<uses-permission android:name="android.permission.RESTART_PACKAGES" />
<uses-permission android:name="android.permission.GET_TASKS" />
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS" />
<uses-permission android:name="android.permission.SET_WALLPAPER" />
<uses-permission android:name="android.permission.USE_FULL_SCREEN_INTENT" />
<uses-permission android:name="android.permission.READ_SYNC_SETTINGS" />
<uses-permission android:name="android.permission.WRITE_SYNC_SETTINGS" />
<uses-permission android:name="android.permission.AUTHENTICATE_ACCOUNTS" />
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
<permission android:name="com.gspace.android.internal.broadcast.permissions" android:protectionL
<uses-permission android:name="com.gspace.android.internal.broadcast.permissions" />
<instrumentation android:name="okhttp3.internal.platform.inner.PowerInstrumentation" android:ta
<uses-permission android:name="android.permission.DISABLE_KEYGUARD" />
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_SOCIAL_STREAM" />
<uses-permission android:name="android.permission.READ_SOCIAL_STREAM" />
<uses-permission android:name="android.permission.READ_PROFILE" />
<uses-permission android:name="android.permission.WRITE_PROFILE" />
<uses-permission android:name="android.permission.READ_USER_DICTIONARY" />
<uses-permission android:name="android.permission.WRITE_USER_DICTIONARY" />
<uses-permission android:name="android.permission.USE_CREDENTIALS" />
<uses-permission android:name="android.permission.ACCESS_MEDIA_LOCATION" />
```

Obrázek 25 – Velké množství oprávnění

Při analýze vzorků s velkým množstvím požadovaných oprávnění, viz. obrázek 25, bylo zjištěno, že požadují často ta, která jsou nebezpečná [98]. Těmi byla již zmíněná oprávnění týkající se SMS a kontaktů. Dále se ale také vyskytovala jiná oprávnění, jako např. ACCESS_MEDIA_LOCATION, která umožnují přístup ke geografickým lokalitám, jež jsou uloženy ve sdílené kolekci dat a GET_ACCOUNTS poskytující možnost přístupu k seznamu účtů v Accounts Service [98].

- Kombinace výše zmíněných nebezpečných oprávnění

Dalších deset aplikací představovalo bezpečnostní hrozbu v podobě oprávnění, která požadují, jelikož se jich vždy vyskytovalo několik, mezi nimiž se objevovala již zmíněná nebezpečná oprávnění z předchozích bodů.

Při prozkoumání všech potenciálně nebezpečných vzorků bylo tedy zjištěno, že více jak 75% z nich představuje bezpečnostní hrozbu v podobě oprávnění, která požadují. Díky těmto oprávněním mohou útočníci totiž uživateli například bez jeho vědomí ukrást citlivá data a dále je využívat.

Kontrola názvu labelu

V posledním kroku při analýze Androidmanifest.xml došlo ke kontrole názvu štítku (label) aplikace, která slouží k zobrazení názvu aplikace v seznamu nainstalovaných aplikací. V šesti případech se jednalo o opakované pojmenování v souvislosti s názvem Android, jako např. AndroidClient, Android ad. Následně dvacet pět aplikací je vždy odkazováno na název labelu do aplikačních zdrojů, konkrétně do souboru strings.xml pomocí @string/app_name. Jak je zjištěno i v porovnání s čistými aplikacemi, tak se jedná o standartní způsob. Z tohoto důvodu byla při analýze aplikačních zdrojů zahrnuta i kontrola názvů labelu, více v oddílu 9.2.3 – Analýza textových řetězců – strings.xml.

9.2 Analýza aplikačních zdrojů

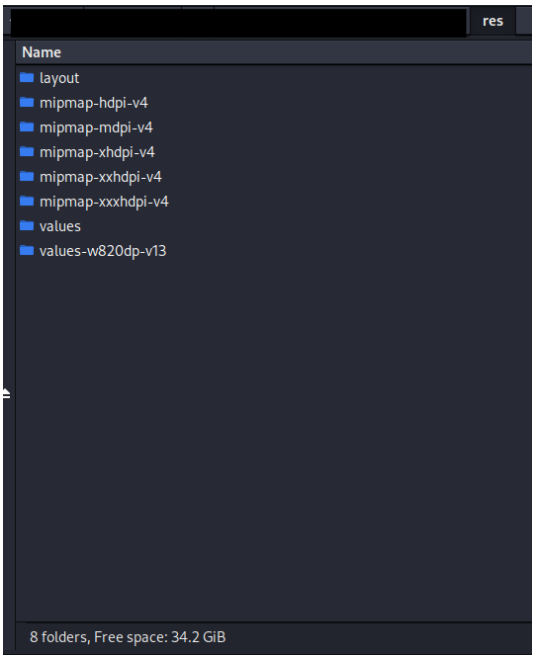
Jak již bylo řečeno v oddíle 2.3.1 – APK, tak aplikace neobsahuje pouze aplikační logiku, ale i využívá také zdrojové soubory. Mezi tyto zdroje spadají grafické soubory, textové řetězce, animace, využívané barvy, definice rozvržení uživatelského rozhraní, různé využívané celočíselné hodnoty ad., které se nacházejí v adresáři res. Jelikož zdroje jsou oddělenou částí od aplikační logiky, tak je snadná modifikace jejich hodnot. Z tohoto důvodu bylo klíčovým krokem analýzy právě prozkoumání tohoto adresáře, zda se v něm neobjevují nestandartní či modifikované hodnoty, jež by mohly charakterizovat malware.

V následujících krocích tedy došlo k podrobné analýze adresáře jako celku, ale také jednotlivých jeho částí, jako jsou ikony aplikací či textové řetězce.

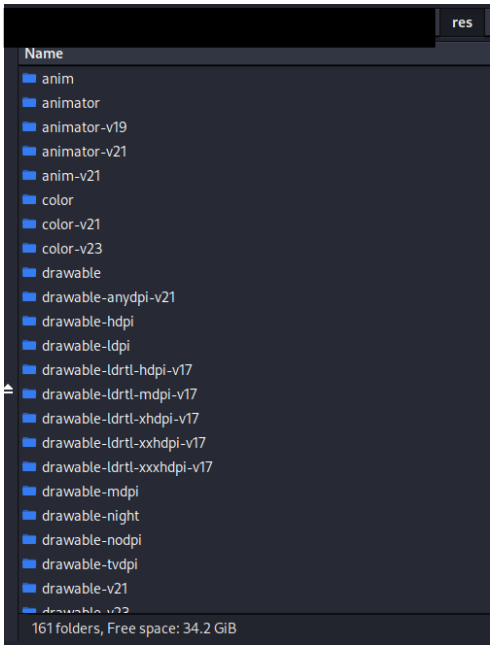
9.2.1 Porovnání adresářů res

V tomto kroku došlo k porovnání adresářů res potenciálně nebezpečných aplikací vůči čistým aplikacím. Konkrétně byly hledány možné opakující se shody v aplikačních zdrojích, tedy zda se v potenciálně nebezpečných aplikacích neobjevují nějaké soubory či složky navíc nebo naopak zda nějaké nechybí. Při porovnání všech potenciálně nebezpečných aplikací vůči čistým aplikacím bylo zjištěno, že v mnoha případech se obsah adresáře res shoduje. Avšak u devíti případů byla objevena pouze část adresářů a zbytek chyběl. Tento příklad je možný vidět na obrázku 26.

Potenciálně nebezpečná aplikace



Čistá aplikace



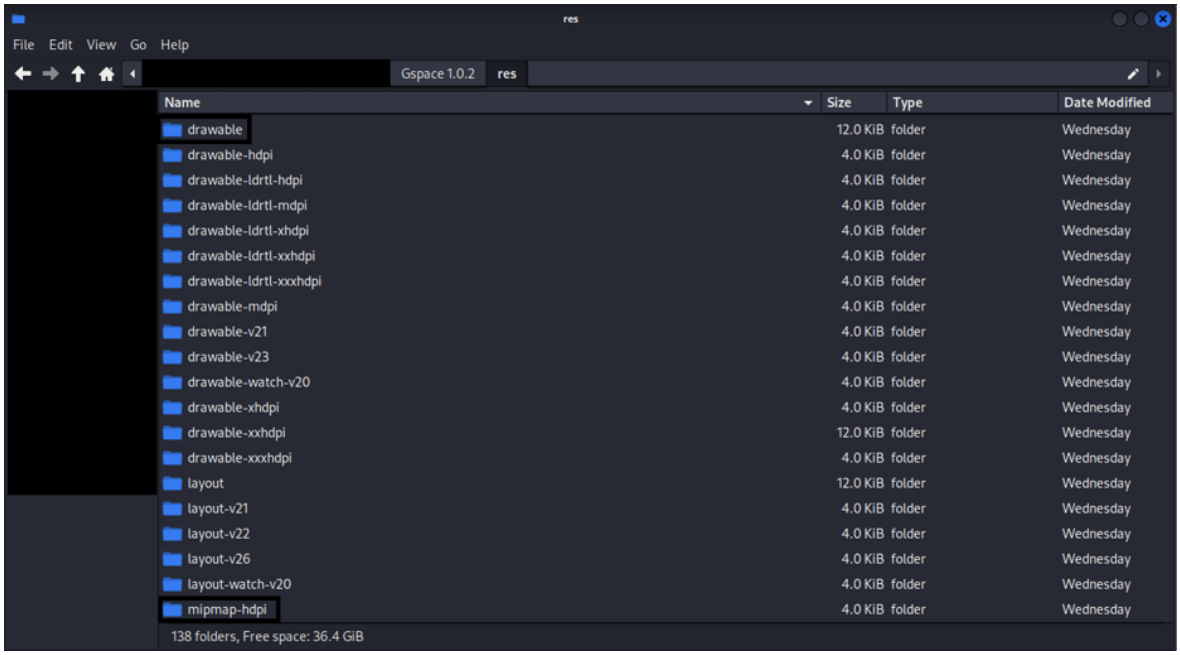
Obrázek 26 - Porovnání adresáře res

Konkrétně jak je na obrázku 26 vidět, tak u potenciálně nebezpečné aplikace se objevuje v adresáři res pouze 8 složek, zatímco čistá aplikace disponuje 161 složkami. Z tohoto důvodu je možné tvrdit, že může jít o charakteristiku s detekčním potenciálem.

9.2.2 Ikony aplikací

Tento oddíl se zabývá zkoumáním ikon jednotlivých aplikací. Jedním z několika možných identifikátorů nebezpečné aplikace může být právě změnění či podvržení ikony dané aplikace. Krokem pro nalezení ikon aplikací bylo zapotřebí v již rozbaleném APK balíčku

prozkoumat adresář s názvem res, a to konkrétně následující souborové složky drawable a mipmap, ve kterých jsou ikony nejčastěji uloženy. Znázornění těchto složek, obsahující ikony je vidět na obrázku 27.



Obrázek 27 - Drawable a mipmap v adresáři resources

Při vlastní analýze potenciálně nebezpečných aplikací byly tyto soubory postupně prozkoumány. Výsledkem bylo, že se v těchto aplikacích objevují tři druhy podvržených ikon aplikací, a to konkrétně:

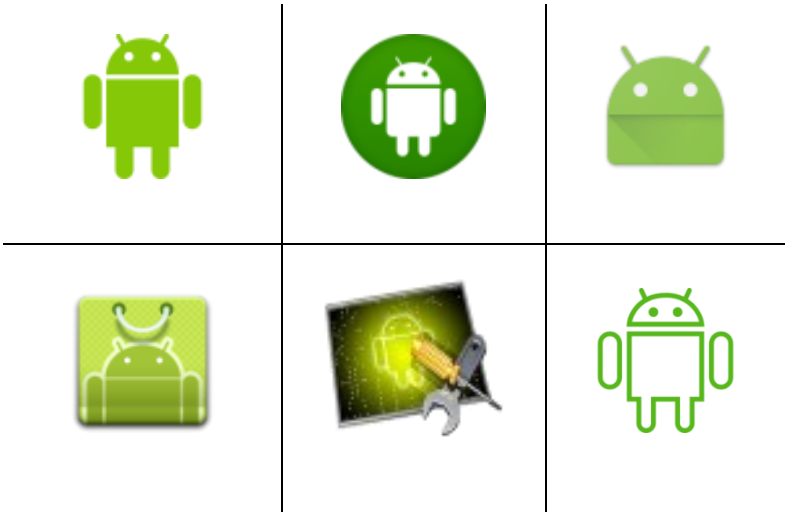
- Modifikované legitimní systémové ikony operačního systému Android

Nejčastěji vyskytující ikony byly právě modifikované legitimní systémové ikony operačního systému Android, které je možné vidět na obrázku 28.

Obrázek 28 – Legitimní systémové ikony aplikací operačního systému Android [26]

Při pohledu na obrázek 29 níže, je možné zpozorovat různé detaily, kterými jsou ikony podvržených Android aplikací změněny. Na první pohled je zcela zřetelná podobnost mezi znázorněnými skupinami (obrázky 28 a 29). Avšak při podrobnějším zkoumání je možné si

všimnout právě malých detailů, kterými jsou ikony změněny. Jedná se především o změnění barvy, zarovnání zakulacených rohů či případné vložení oficiální ikony do vytvořeného pozadí.



Obrázek 29 - Podvržené ikony Android

- Průhledné ikony aplikace

V několika případech došlo při analýze ke zjištění, že některé aplikace autor skrývá za průhlednými ikonami.

- Částečně změněné legitimní ikony aplikace

Analýza také vykázala výsledky, že ke změnám nedochází jen u legitimních systémových ikon operačního systému Android, ale také ikony jiných aplikací jsou pozměňovány. Na obrázku 30 jsou některé konkrétní příklady v porovnání s originální ikonou od vydavatele aplikace znázorněny.

Změněné ikony	Ikony od vydavatele
	
	

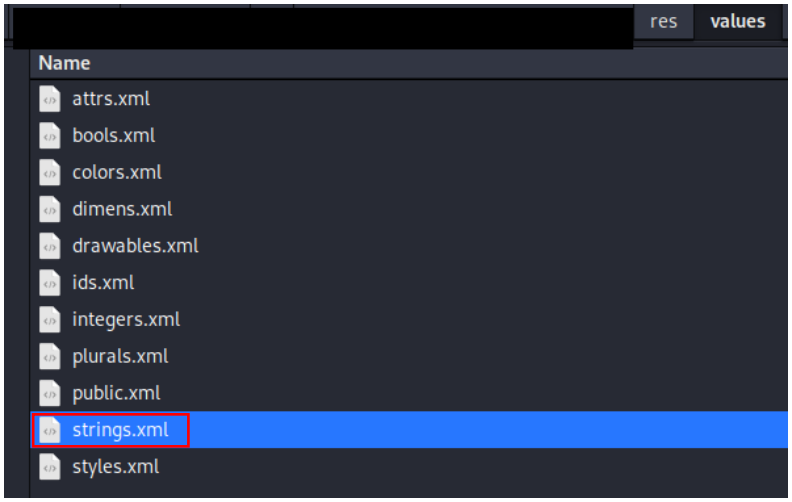
Obrázek 30 - Porovnání změněných ikon vůči originálům

Jako u legitimních systémových ikon operačního systému Android je možné na výše vyobrazených příkladech vidět drobné detailní úpravy ikon, jimiž autor danou aplikaci maskuje.

V tomto kroku došlo ke zjištění, že tvůrce potenciálně škodlivých aplikací využívá aktivní maskování malwaru pomocí podvržených ikon. Díky tomuto kroku dochází pro uživatele k velmi obtížnému určení, že se jedná o infikovaný software, jelikož jeho maskování jej vydává za oficiální.

9.2.3 Analýza textových řetězců – strings.xml

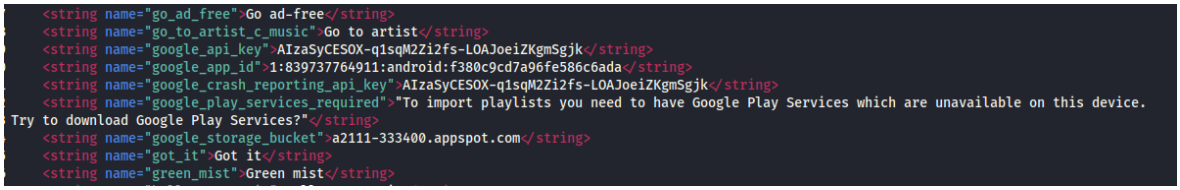
Jedná se o soubor, jež obsahuje některé textové řetězce, které budou využívány v dané aplikaci. Cílem tohoto kroku bylo odhalit právě části, které by vykazovaly nestandartní a potenciálně nebezpečné vlastnosti. Soubor strings.xml se nachází v adresářích res a values, jež je možné vidět na obrázku 31.



Obrázek 31 - Strings.xml

Falešné aktualizace

Při analýze došlo k hledání klíčových slov, jako jsou Google play, update a Google. Výběr těchto slov byl z důvodu, zda v aplikaci vydavatel nevyužívá některé obcházení legitimních aplikací a nedochází k falešným aktualizacím aplikací ad. Při prozkoumání všech potenciálně nebezpečných aplikací bylo zjištěno, že v několika případech se objevují opakující se informace o falešných aktualizacích, které ale nejsou součástí Google Play. Jeden z příkladů je znázorněn na obrázku 32.



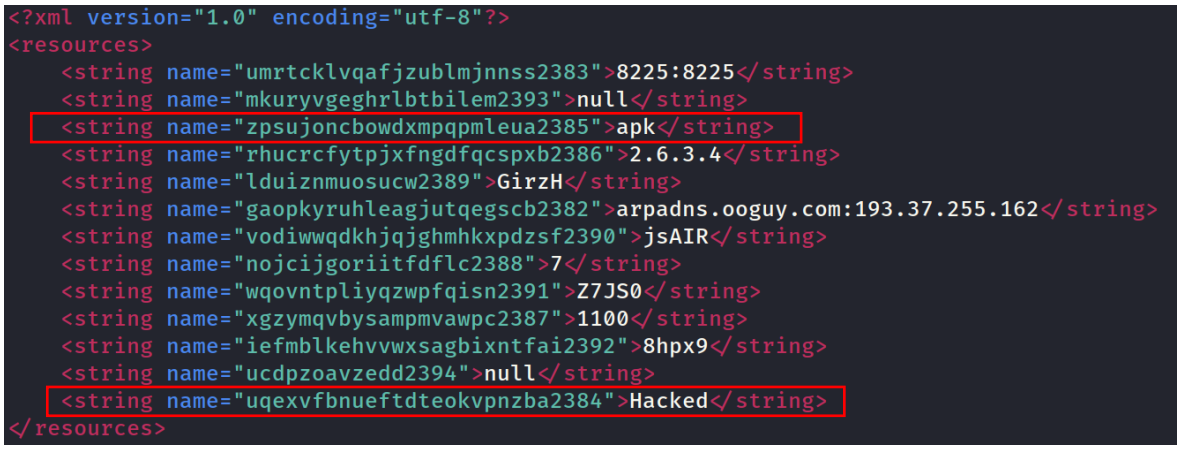
Obrázek 32 - Ukázka stažení falešné aktualizace Google Play Services

Název labelu

Zde dochází ke kontrole názvu labelu v návaznosti na podkapitolu 9.1 - Analýza souboru Androidmanifest.xml. Jelikož v mnoha případech při pojmenovávání labelu dochází k odkazování na strings.xml, tak bylo nutné zkontrolovat jednotlivé názvy. Z dvaceti pěti zkontrolovaných aplikací bylo dvacet jedna jejich názvů totožných s názvem APK balíčku. U tří případů byly labely pojmenovány naprosto za jinou aplikaci, než měly v jejich názvu. Z toho je zřejmé, že se jedná o duplicitní aplikace, jichž útočník poskytuje více, a to pouze pod jiným názvem. Poslední případ vykazoval podezřelé odkazování do strings.xml pomocí řetězce:

android:label="@string/**zpsujoncbowdxmpqpmleua2385"**

Při analýze textového řetězce došlo ke zjištění, že pojmenování labelu je „apk“. Následně zde ale byl na první pohled podezřelý jiný textový řetězec, a to s názvem „Hacked“. Daný příklad je možný vidět na obrázku 33.

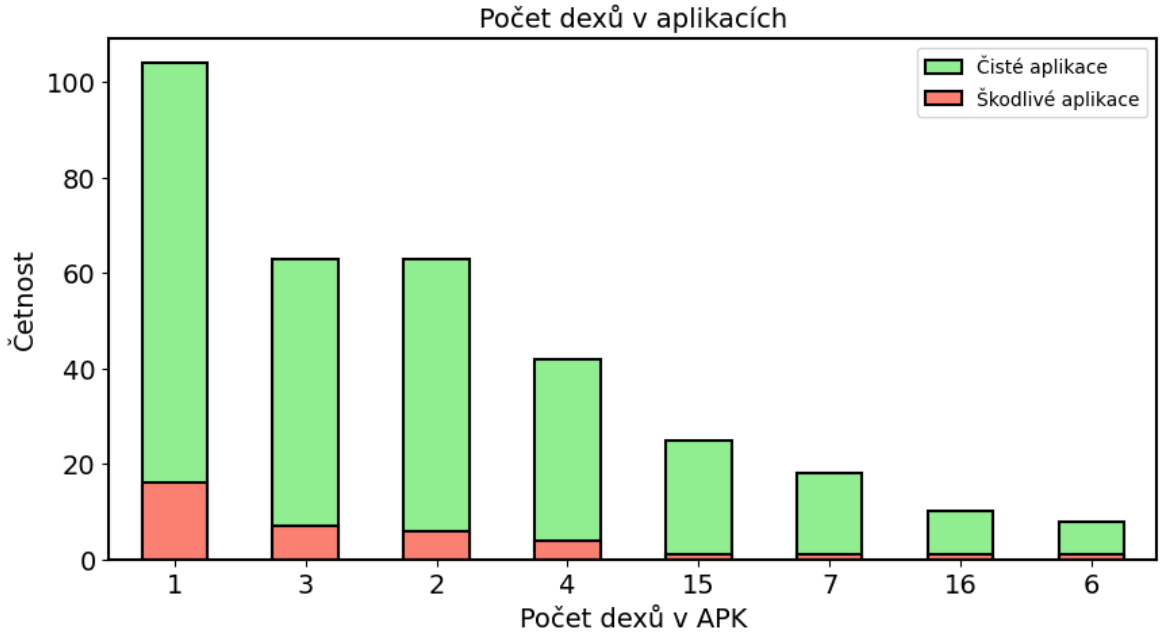


Obrázek 33 - Strings.xml - název labelu

Jak je z výsledků zřejmé, tak pouze v deseti případech, ze všech infikovaných aplikací, se objevují nějakým způsobem podezřelé názvy labelu. Jedná se o sedm případů s existencí názvu spjatého se slovem Android. V dalších 3 případech se jedná o zcela jiný název, než je pojmenována aplikace. Z tohoto důvodu je možné tvrdit, že existuje nějaký detekční potenciál spjatý s názvy labelu.

9.3 Kontrola DEX souborů a úrovní API

V této kapitole dochází ke kontrole anomálií souborů classes.dex v infikovaných aplikacích. K tomuto úkonu bylo nutné mít aplikační balíčky dekompilované. V prvním kroku byl zjištěn počet dex souborů v jednotlivých aplikacích, jenž je vidět na obrázku 34. Tento graf byl tvořen ze dvou souborů dat. Do první kategorie spadají aplikace, které vykazovaly jednu až dvě pozitivní detekce, jež je možné mylně považovat za infikované (Čisté aplikace). Druhou kategorií jsou potenciálně škodlivé aplikace, jejichž detekce byla určena na více jak dvě detekce (Škodlivé aplikace) .



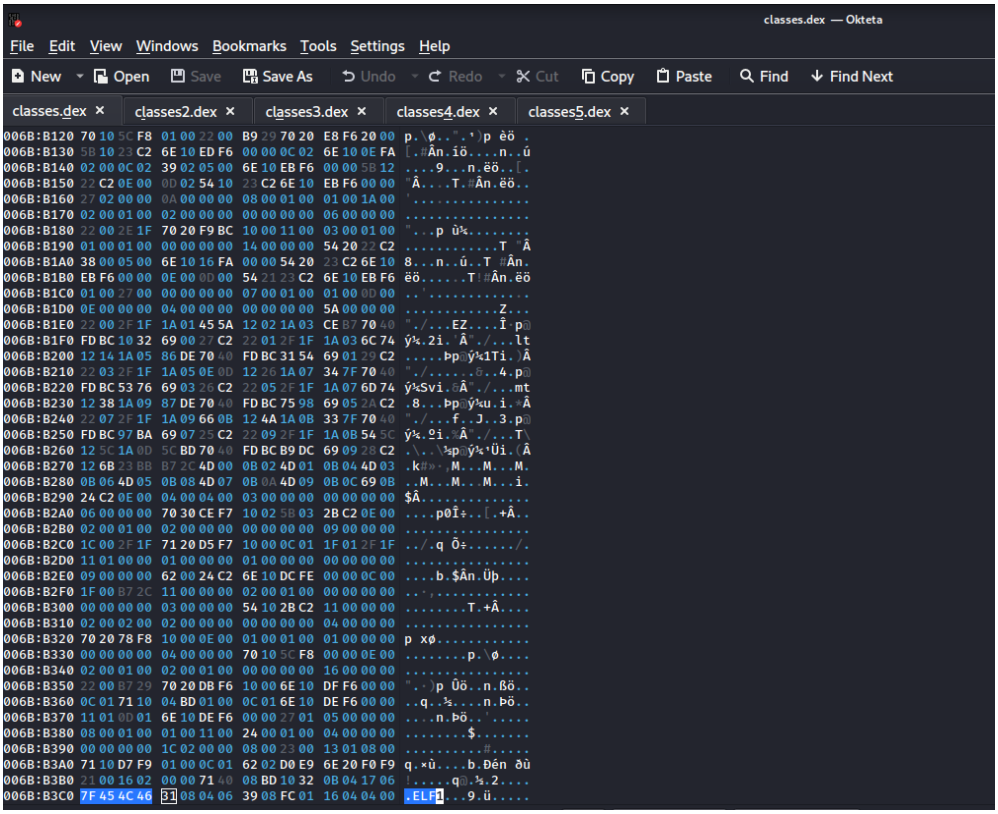
Obrázek 34 - Počet dexů v aplikacích

Jak je z obrázku 34 zřejmé, tak počet dex souborů potenciálně škodlivých aplikací není nikterak závislý na infekčnosti aplikace. To je možné vysvětlit tím, že graf počtu dex souborů ve škodlivých aplikacích přímo kopíruje trend výskytu souborů v jednotlivých potenciálně čistých aplikací. A to pouze s rozdílem jejich počtů, ale výsledek je proporcionální k počtu dat.

Jak bylo vysvětleno v oddíle 2.1.4 – Android Runtime, tak dex soubory obsahují aplikační logiku, která je určena pro běhové prostředí operačního systému Android. Z tohoto důvodu v druhém kroku dochází ke kontrole hlaviček souborů a celé binární struktury dex souborů. Tento krok je vyžadován kvůli kontrole, zda se ve vzorku neobjevuje linuxový formát spustitelných souborů Executable and Linkable Format (ELF) a některý z daných souborů není podvržený za odlišný soubor mimo dex. K tomuto bodu byl využit hexadecimální editor

Okteta. Ten byl vybrán z důvodu jeho jednoduchého grafického uživatelského rozhraní a zároveň kvalitního zpracování souborů, jež je možné vidět na obrázku 35.

Po vyhodnocení všech souborů bylo výsledkem, že v hlavičkách se v žádném z případů neobjevuje ELF a v každém z případů se jedná o soubor dex. Avšak při kontrole celé binární struktury byly nalezeny vzorky, jenž obsahují spustitelný soubor uvnitř kódu. Tento příklad je vidět na obrázku 35.



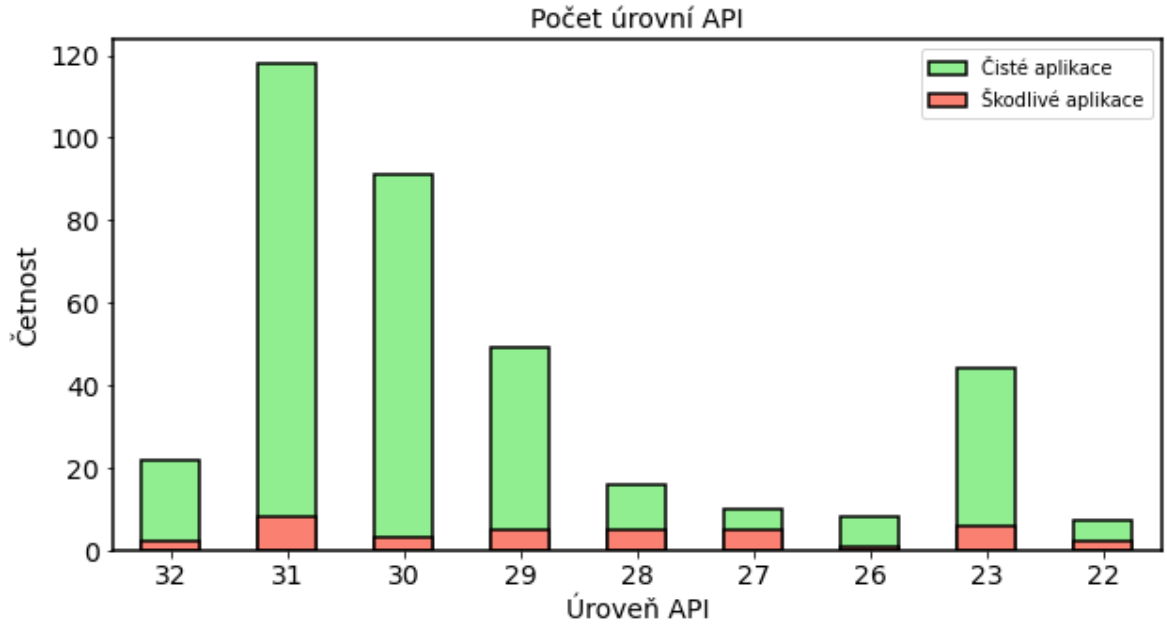
Obrázek 35 – ELF uvnitř dex souboru

Spustitelné soubory uvnitř souboru dex mohou značit jednu z anomálií, kterou je zapotřebí více analyzovat. Z tohoto důvodu by v následujícím výzkumu mohlo dojít k prozkoumání těchto dex souborů, co dané ELF způsobují a zda jsou pro uživatele potenciální hrozbou.

Dalším krokem bylo zjištění úrovně API daných aplikací. Jedná se o celočíselnou hodnotu určující jak daná aplikace bude fungovat v různých verzích systému platformy Android. Porovnání jednotlivých verzí a podrobnější popis je rozebrán v kapitole 2 – Operační systém. Jednotlivé úrovně API daných aplikací jsou uloženy v souboru AndroidManifest.xml. Jsou vyjádřeny v první části daného souboru v atributech v <uses-sdk>, pod jedním z konkrétně zmíněných názvů:

- android:minSDKVersion – určující minimální požadovanou úroveň API na které může aplikace běžet,
- android:targetSDKVersion – určující cílovou úroveň API na které má aplikace běžet,
- android:maxSDKVersion – určující maximální požadovanou úroveň API na které může aplikace běžet.

Konkrétní rozložení úrovní API u jednotlivých aplikací jsou znázorněny na obrázku 36. Jako u předchozího grafu týkající se počtu souborů dex, byl graf tvořen ze dvou souborů dat, kterými jsou potenciálně čisté aplikace a škodlivé aplikace.



Obrázek 36 - Počet úrovní API

Při pohledu na graf je zřejmé, že většina vzorků tvořících datovou sadu potenciálně čistých aplikací mají úrovně API spadající do nejvíce využívaných operačních systému Android. Stejný výsledek je možné zpozorovat i u infikovaných aplikací. Z tohoto důvodu je možné tvrdit, že se jedná o stále uplatnitelné aplikace, které nejsou zastaralé. Toto tvrzení je odůvodněno díky porovnání s tabulkou 2 v kapitole 2 – Operační systém.

10 VYHODNOCENÍ DOSAŽENÝCH VÝSLEDKŮ

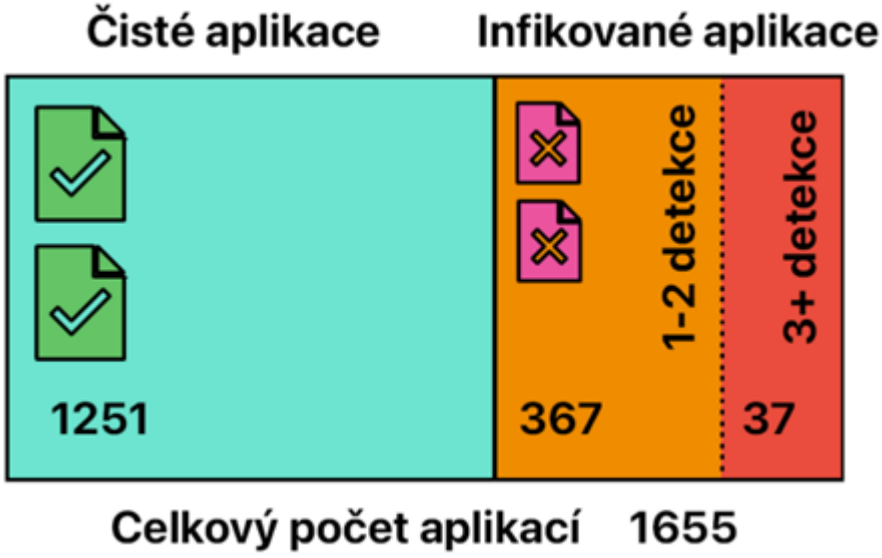
V této kapitole dochází k vyhodnocení a interpretaci dosažených výsledků jednotlivých kroků, kterými se diplomová práce zabývala.

Prvním důležitým úkolem diplomové práce byl sběr dostatečně velké datové sady aplikací pro operační systém Android prostřednictvím torrentů a file share serverů. Pro tento krok byly vybrány 4 torrentové stránky a 3 file share servery. Dohromady ze všech zdrojů bylo získáno 1771 aplikací s nimiž bylo pracováno v navazujících částech práce. V dalším kroku došlo k ověření malignity jednotlivých aplikací. Pro tento úkol byla využita online platforma VirusTotal, která umožňuje uživatelům bezplatně nahrávat a analyzovat soubory za účelem detekce a identifikace škodlivého softwaru. 1251 aplikací z celkového počtu 1771 nevykazovalo žádnou detekovanou hrozbu. Zbylých 520 aplikací bylo detekováno minimálně jedním antivirovým programem.

V následujícím kroku byly mezi získanými vzorky prošetřovány možné duplicity. Nejprve došlo ke kontrole duplicit uvnitř jednotlivých skupin dle zdroje stažení. Zde bylo zjištěno, že ve skupině Fileshare_1 se objevuje z celkových 92 aplikací pouze 12 unikátních vzorků. Poté došlo k prošetření duplicit mezi všemi skupinami. Pro tento úkol bylo využito dvou hashovacích metod. První metoda používala klasické hashovací funkce, konkrétně algoritmus SHA-256 pomocí aplikace HashMyFiles. Při vyhodnocení bylo zjištěno, že 50 aplikací bylo analyzováno jako 100 % duplicitních s jinými aplikacemi z datové sady. Druhá metoda, využívající fuzzy hashovací funkce, se nezaměřovala pouze na úplnou duplicitu. Jejím cílem bylo zjištění podobnosti mezi jednotlivými vzorky. Práh minimální shody byl určen na 30%. Výsledkem bylo 95 podobných aplikací, včetně zmíněných 50 vzorků s úplnou duplicitou. Z těchto výsledků byly brány do následující analýzy pouze unikátní vzorky. Konkrétně 1251 čistých aplikací a 404 potenciálně infikovaných aplikací, které vykazovaly alespoň jednu pozitivní detekci.

Z důvodu, že jedna až dvě detekce mohou vykazovat falešně pozitivní výsledek, tak datová sada unikátních potenciálně infikovaných aplikací byla rozdělena následovně. Vzorky, které vykazovaly jednu až dvě detekce (367 aplikací). Vzorky vykazujících více než dvě pozitivní detekce (37 aplikací). Do dalších kroků analýzy bylo použito právě těchto 37 potenciálně nebezpečných aplikací.

Na obrázku 37 je graficky znázorněno rozložení datové sady na jednotlivé části, které byly popsány v textu výše.



Obrázek 37 - Rozložení datové sady

Dalším úkolem práce byla analýza potenciálně infekčních souborů. Zde došlo ke zkoumání několika částí, přičemž klíčovým prvkem byla analýza aplikačních zdrojů a následné vyhodnocení jejich detekčního potenciálu.

Nejprve byla provedena analýza souboru AndroidManifest.xml. Konkrétně byly zkoumány:

- názvy package,
- požadovaná oprávnění,
- API úroveň aplikace a
- názvy labelu.

Z těchto kroků bylo možné určit, že největší detekční potenciál vykazují oprávnění. Důvodem tohoto tvrzení je, že u více jak 75 % aplikací se při analýze vyskytovalo požadování nebezpečných oprávnění, která by mohla nějakým způsobem ohrozit bezpečnost a soukromí uživatele. Dále je možné tvrdit, že API úroveň zkoumaných aplikací vykazuje velký procentuální podíl ve využívanosti aktuálních verzí operačních systémů Android. Tudíž se nejedná o zastaralé aplikace, které by již neměly využití. Jako poslední krok v této části byla kontrola názvů package a labelu. V obou případech některý z názvů obsahoval detekční potenciál, který spočíval v možném maskování infekční aplikace za některé z oficiálních pojmenování, které se vyskytuje u legitimních aplikací pro operační systém Android.

Během dalšího kroku byla provedena kontrola dex souborů, která proběhla ve třech částech:

- Analýza počtu dex souborů v jednotlivých aplikačních balíčcích,
- kontrola hlaviček a
- kontrola binární struktury.

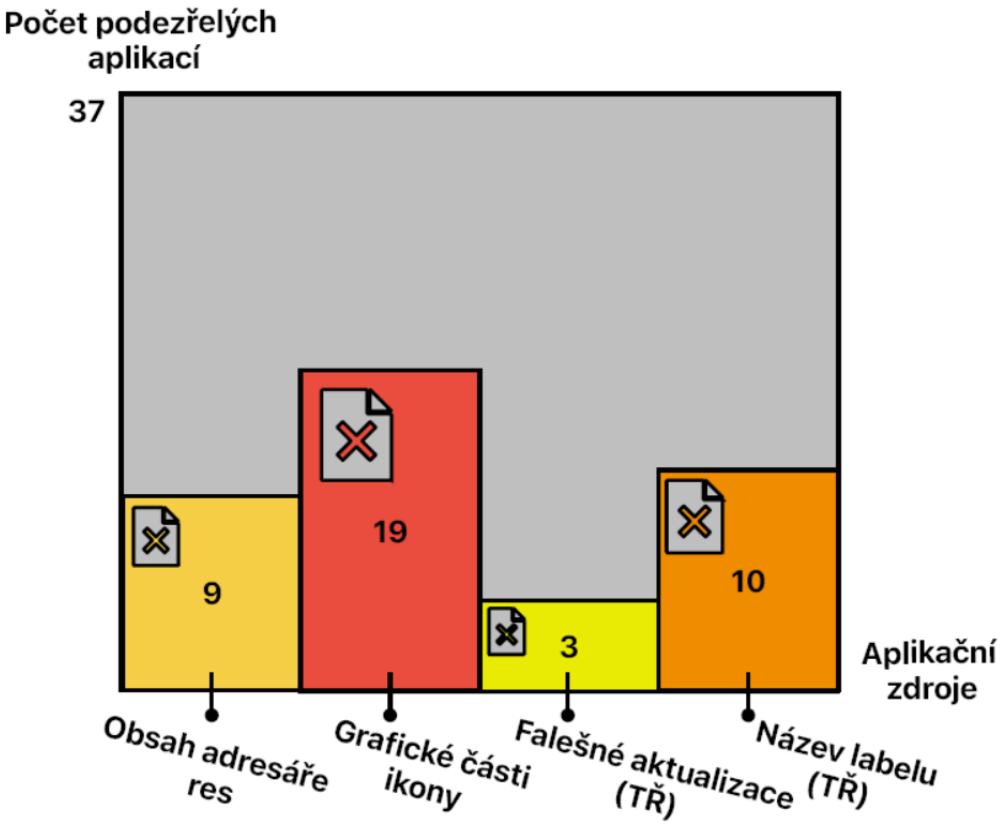
Výsledkem tohoto kroku jsou tři důležité poznatky, konkrétně, že počet dex souborů obsažených v aplikacích není nikterak závislý na infekčnosti aplikace. Tudíž je možné tvrdit, že počet dex souborů nemá detekční potenciál. Dále hlavičky vždy jednoznačně určovaly dex soubor. Naopak při kontrole binární struktury bylo zjištěno, že se v některých případech objevuje spustitelný formát ELF. Tyto případy mohou vykazovat potenciální hrozbu a bylo by vhodné v dalším výzkumu zjistit, z jakého důvodu se tyto spustitelné soubory ve struktuře vyskytují.

Následující krok, který byl pro diplomovou práci klíčový vzhledem k analýze jednotlivých částí, řešil problematiku aplikačních zdrojů. Konkrétně docházelo k hledání možných nestandartních či podezřelých, opakujících se hodnot, jež by mohly prozrazovat detekční potenciál aplikačních zdrojů. Tento krok se skládal z několika úkonů, kterými byly:

- Porovnání adresářů res čistých aplikací s potenciálně infikovanými aplikacemi,
- Zkoumání grafických částí adresáře
    - Ikony.
- Analýza textových řetězců strings.xml
    - Falešné aktualizace a
    - názvy labelu.

V této části byl zjišťován potenciál jednotlivých aplikačních zdrojů. V následujícím kroku byly porovnávány adresáře benigních aplikací vůči maligním aplikacím se stejnou API úrovní. Zde bylo zjištěno, že v devíti případech se objevuje jen část adresářů, přičemž několik klíčových chybí. Následně byly kontrolovány adresáře obsahující grafiku, konkrétně ikony jednotlivých aplikací. U většiny aplikací bylo zjištěno, že tyto aplikační zdroje vykazují detekční potenciál. Konkrétně v devatenácti případech se jednalo o podvržení ikon, kterými útočníci maskují škodlivé aplikace. To může mít za následek obtížné zjištění potenciálně nebezpečné aplikace uživatelem, jelikož se aplikace tváří jako oficiální software. V posledním kroku došlo k analýze textových řetězců (strings.xml). Tato část se skládala ze dvou částí. V prvním úkonu byly hledány možné informace o falešných aktualizacích. Zde

bylo zjištěno, že tyto informace se objevují pouze u tří aplikací. Tudíž je možné říct, že detekční potenciál textových řetězců zabývajících se falešnými aktualizacemi není nikterak velký a tyto případy se objevují zřídka. Druhým krokem této části bylo kontrolování názvů labelu. Z výsledků je možné tvrdit, že názvy labelu vykazují detekční potenciál, jelikož v deseti případech se jednalo o nějakým způsobem podezřelé pojmenování labelu. Konkrétně se v sedmi případech jednalo o názvy, které obsahovaly některý z názvů Android, AndroidClient ad. V dalších třech případech byly názvy labelu zcela odlišné od názvu APK balíčku a vždy odkazovaly na jinou aplikaci. Vyhodnocení detekčních potenciálů jednotlivých aplikačních zdrojů je souhrnně zobrazeno na obrázku 38.



Obrázek 38 - Detekční potenciál aplikačních zdrojů

Podle zobrazeného grafu je patrné, že největší detekční potenciál škodlivých aplikací se nachází v grafických prvcích aplikace. Také obsah adresářů a názvy labelu mají vysoký detekční potenciál. Zatímco detekční potenciál pro textové řetězce (TŘ) týkající se falešných aktualizací je malý, přestože bylo nalezeno několik podezřelých případů. Tato analýza probíhala při omezeném počtu vzorků (37). Pokud by byl zahrnut větší počet vzorků, tak by jednotlivé části mohly ukazovat na větší detekční potenciál.

ZÁVĚR

Prvním krokem práce bylo zjištění současného stavu řešeného problému. Byla provedena literární rešerše, která poskytla přehled o aktuálních řešeních. Výsledkem tohoto kroku bylo zjištění, že v dané problematice existují mezery týkající se tvorby vlastní datové sady a analýzy vzorků stažených prostřednictvím file share serverů a torrentů. Následně se práce zaměřuje na popis operačního systému Android, konkrétně na jeho historii, architekturu, komponenty a balíčky. Dalším krokem byla analýza zdrojů aplikací, z nichž byla datová sada vytvořena. Jsou popsány základní principy torrentů a serverů pro sdílení souborů, včetně jejich statisticky významných zástupců. Ve čtvrté kapitole se práce zabývá virovými skenery, konkrétně službou VirusTotal, která byla v praktické části použita k ověření infekčnosti aplikace. Následující kapitola pojednává o hashovacích funkcích, a to jak o klasických, tak o fuzzy. U obou metod jsou popsány jednotlivé principy fungování, důvody použití a jejich zástupci. V poslední kapitole teoretické části jsou rozebrány jednotlivé analytické metody včetně jejich výhod a nevýhod.

Praktická část začíná popisem vytvořené datové sady, která byla následně podrobena virovému skeneru VirusTotal. Na základě výsledků tohoto kroku byly vzorky rozděleny do dvou skupin, na benigní a maligní aplikace. Následně byla zkontrolována unikátnost jednotlivých vzorků v datové sadě. Pro tento úkol byly v prvním kroku použity klasické hashovací algoritmy, konkrétně SHA-256. Výsledkem tohoto kroku bylo zjištění počtu výskytů duplicitních aplikací mezi vybranými zdroji aplikací. Ve druhém kroku byly použity fuzzy hashovací algoritmy k určení případných podobností mezi jednotlivými vzorky. Po vyhodnocení datové sady pomocí výše uvedených kroků byla provedena dekompilace APK. Ta byla provedena při spolupráci s penetrační laboratoří PTLAB při UTB ve Zlíně. Dekompilované aplikace byly poté podrobeny jednotlivým krokům statické analýzy. Postupně byly zkoumány soubory a adresáře, jako například AndroidManifest.xml, soubory dex a aplikační zdroje. Výsledkem této práce je analýza mapující aktuální stav mobilních aplikací stahovaných prostřednictvím vybraných serverů pro sdílení souborů a torrentů. Díky následnému zkoumání byl také určen detekční potenciál aplikačních zdrojů. Výzkum provedený v rámci této práce bude využit pro navazující výzkum v PTLABu při UTB ve Zlíně.

SEZNAM POUŽITÉ LITERATURY

[1] Android Statistics (2022). Businessofapps [online]. 2022. Dostupné z: https://www.businessofapps.com/data/android-statistics/

[2] Is Google Play Safe? [online]. 24.6.2022. Dostupné z: https://www.lifewire.com/is-google-play-safe-153675

[3] ARP, Daniel, Michael SPREITZENBARTH, Malte HÜBNER, Hugo GASCON a Konrad RIECK. Drebin: Effective and Explainable Detection of Android Malware in Your Pocket. In: Proceedings 2014 Network and Distributed System Security Symposium [online]. Reston, VA: Internet Society, 2014, ISBN 1-891562-35-5. Dostupné z: doi:10.14722/ndss.2014.23247

[4] HAN, Hyoil, SeungJin LIM, Kyoungwon SUH, Seonghyun PARK, Seong-je CHO a Minkyu PARK. Enhanced Android Malware Detection: An SVM-Based Machine Learning Approach. In: 2020 IEEE International Conference on Big Data and Smart Computing (BigComp) [online]. 2020, s. 75-81. ISBN 978-1-7281-6034-4. Dostupné z: doi:10.1109/BigComp48618.2020.00-96

[5] AGRAWAL, Prerna & Trivedi, BHUSNAN. (2020). Automating the process of browsing and downloading APK Files as a prerequisite for the Malware Detection process. International Journal of Emerging Trends & Technology in Computer Science. 9. 13-17, ISSN 2278-6856

[6] MA, Zhuo, Haoran GE, Yang LIU, Meng ZHAO a Jianfeng MA. A Combination Method for Android Malware Detection Based on Control Flow Graphs and Machine Learning Algorithms. IEEE Access [online]. 2019, 7, 21235-21245. ISSN 2169-3536. Dostupné z: doi:10.1109/ACCESS.2019.2896003

[7] CUEVAS, Rubén, Michal KRYCZKA, Roberto GONZÁLEZ, Angel CUEVAS a Arturo AZCORRA. TorrentGuard: Stopping scam and malware distribution in the BitTorrent ecosystem. Computer Networks [online]. 2014, 59, 77-90. ISSN 13891286. Dostupné z: doi:10.1016/j.bjp.2013.12.007

[8] LI, Dongfang, Zhaoguo WANG, Lixin LI, Zhihua WANG, Yucheng WANG a Yibo XUE. FgDetector: Fine-Grained Android Malware Detection. In: 2017 IEEE Second International Conference on Data Science in Cyberspace (DSC) [online]. 2017, s. 311-318. ISBN 978-1-5386-1600-0. Dostupné z: doi:10.1109/DSC.2017.13

[9] LINDORFER, Martina, Matthias NEUGSCHWANDTNER a Christian PLATZER. MARVIN: Efficient and Comprehensive Mobile App Classification through Static and Dynamic Analysis. In: 2015 IEEE 39th Annual Computer Software and Applications Conference [online]. 2015, s. 422-433. ISBN 978-1-4673-6564-2. Dostupné z: doi:10.1109/COMPSAC.2015.103

[10] CAI, Haipeng, Na MENG, Barbara RYDER a Daphne YAO. DroidCat: Effective Android Malware Detection and Categorization via App-Level Profiling. IEEE Transactions on Information Forensics and Security [online]. 2019, 14(6), 1455-1470. ISSN 1556-6013. Dostupné z: doi:10.1109/TIFS.2018.2879302

[11] NARAYANAN, Annamalai, Mahinthan CHANDRAMOHAN, Lihui CHEN a Yang LIU. Context-Aware, Adaptive, and Scalable Android Malware Detection Through Online Learning. IEEE Transactions on Emerging Topics in Computational Intelligence [online]. 2017, 1(3), 157-175. ISSN 2471-285X. Dostupné z: doi:10.1109/TETCI.2017.2699220

[12] DA COSTA, Francisco Handrick, Ismael MEDEIROS, Thales MENEZES, João Victor DA SILVA, Ingrid Lorraine DA SILVA, Rodrigo BONIFÁCIO, Krishna NARASIMHAN a Márcio RIBEIRO. Exploring the use of static and dynamic analysis to improve the performance of the mining sandbox approach for android malware identification. Journal of Systems and Software [online]. 2022, 183. ISSN 01641212. Dostupné z: doi:10.1016/j.jss.2021.111092

[13] ZHANG, Yaocheng, Wei REN, Tianqing ZHU a Yi REN. SaaS: A situational awareness and analysis system for massive android malware detection. Future Generation Computer Systems [online]. 2019, 95, 548-559. ISSN 0167739X. Dostupné z: doi:10.1016/j.future.2018.12.028

[14] KARBAB, ElMouatez Billah, Mourad DEBBABI, Abdelouahid DERHAB a Djedjiga MOUHEB. Android malware detection using machine learning: data-driven fingerprinting and threat intelligence. Cham: Springer, [2021]. Advances in information security. ISBN 978-3030746636.

[15] APK file (Android Package Kit file format). Techtarget [online]. Dostupné z: https://www.techtarget.com/whatis/definition/APK-file-Android-Package-Kit-file-format

[16] GARG, Shivi a Niyati BALIYAN. Data on Vulnerability Detection in Android. Data in Brief [online]. 2019, 22, 1081-1087. ISSN 23523409. Dostupné z: doi:10.1016/j.dib.2018.12.038

[17] ZHANG, Yingmin, Chao FENG, Lianfen HUANG, Chaolin YE a Le WENG. Detection of Android Malicious Family Based on Manifest Information. In: 2020 15th International Conference on Computer Science & Education (ICCSE) [online]. IEEE, 2020, 2020, s. 202-205. ISBN 978-1-7281-7267-5. Dostupné z: doi:10.1109/ICCSE49874.2020.9201835

[18] PUERTA, José, Sanz, BORJA & Igor, SANTOS & Pablo, BRINGAS. (2015). Using Dalvik Opcodes for Malware Detection on Android. Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science). 9121. 416-426. 10.1007/978-3-319-19644-2-35.

[19] FAR, Ali, Hani RAGAB HASSEN, Michael A. LONES a Hind ZANTOUT. An in-depth review of machine learning based Android malware detection. Computers & Security [online]. 2022, 121. ISSN 01674048. Dostupné z: doi:10.1016/j.cose.2022.102833

[20] KABAKUS, Abdullah Talha. DroidMalwareDetector: A novel Android malware detection framework based on convolutional neural network. Expert Systems with Applications [online]. 2022, 206. ISSN 09574174. Dostupné z: doi:10.1016/j.eswa.2022.117833

[21] THANGAVELOO, Rajan, Wong WANG JING, Chiew KANG LENG a Johari ABDULLAH. DATDroid: Dynamic Analysis Technique in Android Malware Detection. International Journal on Advanced Science, Engineering and Information Technology [online]. 2020, 10(2), 536-541. ISSN 2460-6952. Dostupné z: doi:10.18517/ijaseit.10.2.10238

[22] AMOS, Brandon, Hamilton TURNER a Jules WHITE. Applying machine learning classifiers to dynamic Android malware detection at scale. In: 2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC) [online]. IEEE, 2013, 2013, s. 1666-1671. ISBN 978-1-4673-2480-9. Dostupné z: doi:10.1109/IWCMC.2013.6583806

[23] SHYONG, Yung-Ching, Tzung-Han JENG a Yi-Ming CHEN. Combining Static Permissions and Dynamic Packet Analysis to Improve Android Malware Detection. In: 2020 2nd International Conference on Computer Communication and the Internet (ICCCI) [online]. IEEE, 2020, 2020, s. 75-81. ISBN 978-1-7281-5800-6. Dostupné z: doi:10.1109/ICCCI49374.2020.9145994

[24] Android or iOS? This map shows which is the most used operating system in each country in the world. Crast [online]. 10.4.2023. Dostupné z:

https://crast.net/308488/android-or-ios-this-map-shows-which-is-the-most-used-operating-system-in-each-country-in-the-world/

[25] Mobile malware evolution 2021. Securelist [online]. 21.1.2023. Dostupné z: https://securelist.com/mobile-malware-evolution-2021/105876/

[26] About the Android Open Source Project. Source.android [online]. Dostupné z: https://source.android.com/

[27] IOS: What Does iOS Mean?. Techopedia [online]. 28.8.2012. Dostupné z: https://www.techopedia.com/definition/25206/ios

[28] The history of Android: The evolution of the biggest mobile OS in the world. ANDROIDAUTHORITY [online]. John Callaham, 2022. Dostupné z: https://www.androidauthority.com/history-android-os-name-789433/

[29] If you can't build it, buy it: Google's biggest acquisitions mapped. WIRED [online]. Matt Reynolds, 2017. Dostupné z: https://www.wired.co.uk/article/google-acquisitions-data-visualisation-infoporn-waze-youtube-android

[30] The first Android phone was announced 13 years ago today. Androidpolice [online]. Arol Wright, 2021. Dostupné z: https://www.androidpolice.com/2021/09/23/the-first-android-phone-was-announced-13-years-ago-today/

[31] Nelichotivé: Android 9 zatím stále vede nad Androidem 12. Svetandroida [online]. Marek Houser, 2022. Dostupné z: https://www.svetandroida.cz/android-fragmentace-statistika-8-2022/

[32] Understanding Android API levels. Microsoft [online]. 2021. Dostupné z: https://learn.microsoft.com/en-us/xamarin/android/app-fundamentals/android-api-levels?tabs=windows

[33] Platform Architecture. Developer.android [online]. 2021. Dostupné z: https://developer.android.com/guide/platform

[34] Android Automotive and Physical Car Interaction. AndroidAutomotivOSBook [online]. Moritz Iseke, 2020. Dostupné z: https://www.androidautomotivebook.com/android-automotive-and-physical-car-interaction/

[35] Why do we use the Linux Kernel in Android ?. Emteria [online]. Teresa Reidt, 2022. Dostupné z: https://emteria.com/learn/linux-kernel

[36] Overview of Linux Kernel Security Features. Linux [online]. The Linux Foundation, 2013. Dostupné z: https://www.linux.com/training-tutorials/overview-linux-kernel-security-features/

[37] What is SELinux?. RedHat [online]. 2019. Dostupné z: https://www.redhat.com/en/topics/linux/what-is-selinux

[38] Technologies for container isolation: A comparison of AppArmor and SELinux. RedHat [online]. Lukas Vrabec, 2020. Dostupné z: https://www.redhat.com/sysadmin/apparmor-selinux-isolation

[39] NIU, Yingjiao, Yuewu WANG, Shijie JIA, Quan ZHOU, Lingguang LEI, Qionglu ZHANG a Xinyi ZHAO. Enhancing the Security of Mobile Device Management with Seccomp. Journal of Physics: Conference Series [online]. 2020, 1646(1). ISSN 1742-6588. Dostupné z: doi:10.1088/1742-6596/1646/1/012138

[40] How to use the Linux kernel's Integrity Measurement Architecture. RedHat [online]. Huzaifa Sidhpurwala, 2020. Dostupné z: https://www.redhat.com/en/blog/how-use-linux-kernels-integrity-measurement-architecture

[41] Hardware Abstraction Layer (HAL) overview. Source.android [online]. Dostupné z: https://source.android.com/docs/core/architecture/hal

[42] Get started with the NDK. Developer.android [online]. Dostupné z: https://developer.android.com/ndk/guides

[43] Working of Dalvik Virtual Machine in Android. Data-flair [online]. Dostupné z: https://data-flair.training/blogs/android-dalvik-virtual-machine/

[44] Difference Between Dalvik and ART in Android. Geeksforgeeks [online]. Dostupné z: https://www.geeksforgeeks.org/difference-between-dalvik-and-art-in-android/

[45] Android Core: JVM, DVM, ART, JIT, AOT. *Medium* [online]. Shishir, 2020 . Dostupné z: https://medium.com/programming-lite/android-core-jvm-dvm-art-jit-aot-855039a9a8fa

[46] How JVM Works – JVM Architecture?. Geeksforgeeks [online]. 2022. Dostupné z: https://www.geeksforgeeks.org/jvm-works-jvm-architecture/

[47] What is DVM(Dalvik Virtual Machine)?. Geeksforgeeks [online]. 2020. Dostupné z: https://www.geeksforgeeks.org/what-is-dvmdalvik-virtual-machine/

[48] Architecture overview. Source.android [online]. Dostupné z: https://source.android.com/docs/core/architecture

[48] Android Architecture. Geeksforgeeks [online]. 2021. Dostupné z: https://www.geeksforgeeks.org/android-architecture/

[49] KHAN, Jamil a Sara SHAHZAD. Android Architecture and Related Security Risks. Asian Journal of Technology & Management Research. 2015, 5(2). ISSN 2249 –0892.

[50] Application Fundamentals. Developer.android [online]. 2022. Dostupné z: https://developer.android.com/guide/components/fundamentals

[51] Android - Application Components. Tutorialspoint [online]. Dostupné z: https://www.tutorialspoint.com/android/android_application_components.htm

[52] Android Services. Medium [online]. Hüseyin Özkoç, 2021. Dostupné z: https://medium.com/@huseyinozkoc/android-services-tutorial-with-example-fa329e6a5b4b

[53] Services overview. Developer android [online]. 2022. Dostupné z: https://developer.android.com/guide/components/services

[54] Broadcasts overview. Developer.android [online]. Dostupné z: https://developer.android.com/guide/components/broadcasts

[55] Content provider basics. Developer.android [online]. Dostupné z: https://developer.android.com/guide/topics/providers/content-provider-basics

[56] AndroidManifest.xml file in android. *JavaTpoint* [online]. Dostupné z: https://www.javatpoint.com/AndroidManifest-xml-file-in-android

[57] Difference between apk (.apk) and app bundle (.aab). Stackoverflow [online]. 2018. Dostupné z: https://stackoverflow.com/questions/52059339/difference-between-apk-apk-and-app-bundle-aab

[58] ELLIOTT, Dom. What a new publishing format means for the future of Android. Medium [online]. 2018. Dostupné z: https://medium.com/googleplaydev/what-a-new-publishing-format-means-for-the-future-of-android-2e34981793a

[59] X. JIANG. a Y. ZHOU. Android malware [online]. Springer, 2013. ISBN 978-1461473930.

[60] LYNCH, Jamie. The Dex File Format. Bugsnag [online]. 2018. Dostupné z: https://www.bugsnag.com/blog/dex-and-d8

[61] ARSC. Documentation.fileformat [online]. Dostupné z: https://docs.fileformat.com/programming/arsc/

[62] What Are The Purposes Of Files In Meta-inf Folder Of An Apk File? With Solutions. Folkstalk [online]. Dostupné z: https://www.folkstalk.com/tech/what-are-the-purposes-of-files-in-meta-inf-folder-of-an-apk-file-with-solutions/

[63] APK Signature Scheme v2. Source.android [online]. Dostupné z: https://source.android.com/docs/security/features/apksigning/v2

[64] Android | res/values folder. Geeksforgeeks [online]. 2022. Dostupné z: https://www.geeksforgeeks.org/android-res-values-folder/

[65] R. MATELES, D. REJABEK, O. Margalit a R. MOSKOVITCH. Decompiled APK based malicious code classification. Future Generation Computer Systems: Volume 110 [online]. 2020, 135-147. ISSN 0167-739X. Dostupné z: doi: https://doi.org/10.1016/j.future.2020.03.052

[66] Torrenting: What is It and How does It Work?. Techslang [online]. 2022. Dostupné z: https://www.techslang.com/torrenting-what-is-it-and-how-does-it-work/

[67] BISCHOFF, Paul. What is Torrenting? Is it Safe? Is it illegal? Are you likely to be caught?. Comparitech [online]. 2022. Dostupné z: https://www.comparitech.com/blog/vpn-privacy/is-torrenting-safe-illegal-will-you-be-caught/

[68] HARDYN, Michal. P2p-networks – kopie. Times [online]. 2017. Dostupné z: https://www.times.cz/kryptomena/p2p-networks-kopie/

[69] What Is BitTorrent and Is It Safe?. Kaspersky [online]. Dostupné z: https://www.kaspersky.com/resource-center/definitions/bittorrent

[70] Torrent file. Wikipedia [online]. Dostupné z: https://en.wikipedia.org/wiki/Torrent_file

[71] 12 nejlepších torrentových stránek (2022) | Bezpečné a funkční. Vpnmentor [online]. 2022. Dostupné z: https://cs.vpnmentor.com/blog/10-nejlepsich-torrent-stranek/

[72] The Pirate Bay Trial: The Official Verdict – Guilty. Torrentfreak [online]. 2009. Dostupné z: https://torrentfreak.com/the-pirate-bay-trial-the-verdict-090417/

[73] Pirate Bay is The King of Torrents Once Again. Torrentfreak [online]. 2016 . Dostupné z: https://torrentfreak.com/pirate-bay-king-torrents-160814/

[74] About Limettorents. Limetorrents [online]. 2018. Dostupné z: https://www.limetorrents.online/about-limetorrents/

[75] All You Need To Know About "Zooqle" – Torrent's Best Search Engine. Hackzub [online]. 2019. Dostupné z: https://www.hackzhub.com/zooqle/

[76] OLUWATOSIN, Haroon Shakirat. Client-Server Model. IOSR Journal of Computer Engineering [online]. 2014, 16(1), 57-71. ISSN 22788727. Dostupné z: doi:10.9790/0661-16195771

[77] Client and Server model. JavaTpoint [online]. Dostupné z: https://www.javatpoint.com/computer-network-client-and-server-model

[78] 9 nejlepších českých webů, kde lze stáhnout filmy a seriály. Zive [online]. 2016. Dostupné z: https://www.zive.cz/clanky/9-nejlepsich-ceskych-webu-kde-lze-stahnout-filmy-a-serialy/sc-3-a-184164/default.aspx#part=1

[79] Datoid.cz. Mydebrid [online]. Dostupné z: https://mydebrid.com/datoid-cz

[80] Sdílej.cz. Sdílej.cz [online]. Dostupné z: https://www.sdilej.cz/

[81] Virus total - How it works. VirusTotal [online]. Dostupné z: https://support.virustotal.com/hc/en-us/articles/115002126889-How-it-works

[82] 7 nejlepších online antivirových scannerů – aktualizace 2022. Safetydetectives [online]. 10.02.2022. Dostupné z: https://cs.safetydetectives.com/blog/nejlepsich-online-antivirovych-scanneru/#safetydetective

[83] OŠŤÁDAL, Radim. Teoretický základ a přehled kryptografických hashovacích funkcí [online]. Brno, 03.2012n. l. Dostupné také z: https://is.muni.cz/www/ostadal/hash_overview.pdf. Semestrální práce. Masarykova univerzita.

[84] MD5 Třída. Microsoft [online]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/api/system.security.cryptography.md5?view=net-6.0

[85] SHA-2, MD5 či BLAKE3: víte, jaké hashovací algoritmy jsou bezpečné? MasterDC [online]. 15.12.2021. Dostupné z: https://www.master.cz/blog/sha-2-md5-blake3-jake-hash-algoritmy-jsou-bezpecne/

[86] SHA-2, MD5 či BLAKE3: víte, jaké hashovací algoritmy jsou bezpečné?. Master [online]. Dostupné z: https://www.master.cz/blog/sha-2-md5-blake3-jake-hash-algoritmy-jsou-bezpecne/

[87] Y. LI, S. C. SUNDARAMURTHY, A. G. BARDAS, X. OU, D. CARAGEA, X. HU a J. JANG. Experimental Study of Fuzzy Hashing in Malware Clustering Analysis [online]. Washington, D.C.: Conference: 8th Workshop on Cyber Security Experimentation and Test (CSET 15), 2015.

[88] Content Examination Definitions: Using Fuzzy Hashing. Community.mcafee [online]. 2019. Dostupné z: https://community.mimecast.com/s/article/Content-Examination-Definitions-Using-Fuzzy-Hashing-809049828

[89] GROOTEN, Martijn. Optimizing ssDeep for use at scale. Virusbulletin [online]. Cylance,USA, 2015. Dostupné z: https://www.virusbulletin.com/virusbulletin/2015/11/optimizing-ssdeep-use-scale

[90] Gayoso, M. J. a Artés-Rodríguez, A. Improved ssdeep signature generation for effective similarity detection. Mathematics, [online] 2020, 8(4), 503 [cit. 2023-05-17]. DOI: 10.3390/math8040503. Dostupné z: https://doi.org/10.3390/math8040503.

[91] D. MUGISHA, ANDROID APPLICATION MALWARE ANALYSIS. International Journal of Mobile Learning and Organisation 12 (Android Malware) [online]. 2019. ISSN 1746-7268.

[92] A. A. ALI a A. S. H. ABDUL-QAWY. Static Analysis of Malware in Android-based Platforms: A Progress Study. IJCDS Journal [online]. 2020, 1-10. Dostupné z: doi:10.12785/ijcds/100132

[93] LI, Li, Tegawendé F. BISSYANDÉ, Mike PAPADAKIS, Siegfried RASTHOFER, Alexandre BARTEL, Damien OCTEAU, Jacques KLEIN a Le TRAON. Static analysis of android apps: A systematic literature review. Information and Software Technology [online]. 2017, 88, 67-95. ISSN 09505849. Dostupné z: doi:10.1016/j.infsof.2017.04.001

[94] SHATNAWI, Ahmed S., Qussai YASSEN a Abdulrahman YATEEM. An Android Malware Detection Approach Based on Static Feature Analysis Using Machine Learning Algorithms. Procedia Computer Science [online]. 2022, 201, 653-658. ISSN 18770509. Dostupné z: doi:10.1016/j.procs.2022.03.086

[95] SCHMEELK, Suzanna, Junfeng YANG a Alfred AHO. Android Malware Static Analysis Techniques. In: Proceedings of the 10th Annual Cyber and Information Security Research Conference [online]. New York, NY, USA: ACM, 2015, 2015-04-07, s. 1-8. ISBN 9781450333450. Dostupné z: doi:10.1145/2746266.2746271

[96] KAPRATWAR, Ankita. Static and Dynamic Analysis for Android Malware Detection. San Jose, 2016. Master´s project. San Jose State University. Vedoucí práce Mark Stamp.

[97] Trojan:Win32/AgentTesla!ml (AgentTesla ml). Howtofix.guide [online]. Dostupné z: https://howtofix.guide/trojanwin32-agentteslaml/

[98] Manifest.permission. Developer.android [online]. Dostupné z: https://developer.android.com/reference/android/Manifest.permission

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

APK	Android Application Package
LLC	Limited liability company
Inc.	Incorporated
GUI	Graphic User Interface
UTB	Univerzita Tomáše Bati
iOS	iPhone Operating System
HAL	Hardware Abstraction Layer
ART	Android Runtime
DVM	Dalvik Virtual Machine
AOT	Ahead-Of-Time
JIT	Just-In-Time
VM	Virtual Machine
JVM	Java Virtual Machine
AAB	Android Application Bundle
P2P	Peer-to-peer
API	Application Programming Interface
NIST	National Institute of Standards and Technology
ELF	Executable and Linkable Format
TŘ	Textový řetězec

SEZNAM OBRÁZKŮ

SEZNAM TABULEK