VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

DETEKCE MOBILNÍCH APLIKACÍ POMOCÍ PROFILŮ KOMUNIKACE

DETECTION OF MOBILE APPLICATIONS USING TRAFFIC PROFILING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **RADOVAN BABIC**
AUTHOR

VEDOUCÍ PRÁCE **Ing. PETR MATOUŠEK, Ph.D., M.A.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Babic Radovan**
Program: Informační technologie
Název: **Detekce mobilních aplikací pomocí profilů komunikace**
Detection of Mobile Applications Using Traffic Profiling
Kategorie: Počítačové sítě

Zadání:

1. Seznamte se s nástroji pro emulaci mobilních aplikací, např. Android Studio, ADB, apod.
2. Navrhněte a ověřte způsob pro automatizovanou emulaci mobilních aplikací a zachycení provozu aplikací pro účely profilování.
3. Vytvořte datovou sadu obsahující komunikaci vybraných aplikací a zjistěte, které položky z hlaviček komunikačních protokolů (features) jsou vhodné pro vytvoření profilu.
4. Navrhněte a implementujte nástroj pro extrakci otisků ze síťové komunikace a vytváření profilů komunikace zvolených aplikací.
5. Ukažte, jak lze pomocí databáze profilů aplikací detekovat přítomnost aplikací v síťové komunikaci. Posuďte přesnost detekce.
6. Diskutujte možné využití a další směry vývoje.

Literatura:

- Matoušek P., Burgetová I., Ryšavý O., Victor M., "On Reliability of JA3 Hashes for Fingerprinting Mobile Applications", in Proceedings of ICDF2C 2020.
- Anderson, B., McGrew, D.: TLS Beyond the Browser: Combining End Host and Network Data to Understand Application Behavior. In: Proceedings of the Internet Measurement Conference. pp. 379-392, 2019.
- Kotzias, P., Razaghpanah, A., Amann, J., Paterson, K.G., Vallina-Rodriguez, N., Caballero, J.: Coming of age: A longitudinal study of TLS deployment. In: Proceedings of the Internet Measurement Conference 2018. pp. 415-428, 2018.
- Razaghpanah, A., Niaki, A.A., Vallina-Rodriguez, N., Sundaresan, S., Amann, J., Gill, P.: Studying TLS Usage in Android Apps. In: Proc. of the 13th Conf. on Emerging Networking Experiments and Technologies. p. 350-362. New York, 2017.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Matoušek Petr, Ing., Ph.D., M.A.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 22. října 2020

Abstrakt

Táto bakalárska práca sa zaoberá JA3 a JA3S metódami digitálneho vytvárania profilov komunikácie mobilných aplikácií na základe TLS handshake medzi klientom a serverom. V práci je popísaná využitá metóda emulácie mobilných zariadení využívajúcich operačný systém Android, inštalácia aplikácií, generovanie a odchytávanie prevádzky potrebnej pre vytváranie databázy profilov. Ďalej je v práci popísaná metóda, ktorú som implementoval do nástroja na automatizované vytváranie databázy digitálnych profilov aplikácií, ich následnú klasifikáciu a rozpoznávanie pomocou dát získavaných z internetovej prevádzky v sieti.

Abstract

This bachelor thesis deals with JA3 and JA3S methods of digital profiling of mobile applications based on TLS handshake between client and server. The thesis describes the used method of emulation of mobile devices using the Android operating system, installation of applications, generation and capture of traffic needed to create a database of profiles. Furthermore, the work describes the method that I implemented in the tool for automated creation of a database of digital profiles of applications and their subsequent classification and recognition using data obtained from internet traffic in the network.

Kľúčové slová

JA3, JA3S, Odtlačok, SNI, TLS, Android, Python, SQLite

Keywords

JA3, JA3S, Fingerprint, SNI, TLS, Android, Python, SQLite

Citácia

BABIC, Radovan. *Detekce mobilních aplikací pomocí profilů komunikace.* Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Petr Matoušek, Ph.D., M.A.

Detekce mobilních aplikací pomocí profilů komunikace

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Petra Matouška, Ph.D., M.A. Uviedol som všetky literárne pramene, publikácie a dalšie zdroje, z ktorých som čerpal.

.....
Radovan Babic
11. mája 2021

Poďakovanie

Touto cestou by som sa chcel poďakovať pánovi Ing. Petrovi Matouškovi, Ph.D., M.A. za výborné vedenie mojej práce, cenné rady, poznatky a za trpezlivosť, ktorá bola určite veľakrát potrebná.

Obsah

1 Úvod **3**

2 Emulácia aplikácií a odchytávanie prevádzky **4**
2.1 Android Studio 4
2.1.1 Emulátor 4
2.2 Rozhranie Android Debug Bridge 5
2.2.1 Spustenie ADB 6
2.2.2 Inštalácia aplikácií 7
2.2.3 Simulátor monkey 7
2.3 Sieťový analyzátor Wireshark 8
2.3.1 PyShark 10
2.4 Skript pre inštaláciu aplikácií a emuláciu prevádzky 11
2.4.1 Spustenie nástroja 11
2.4.2 Modul EmulatorHandler 12
2.4.3 Modul AppInstaller 13
2.4.4 Modul PacketHandler 14
2.5 Zhrnutie 15

3 Metódy tvorenia digitálneho TLS odtlačku JA3 a JA3S **16**
3.1 Nadviazanie TLS komunikácie 16
3.2 Digitálny odtlačok klienta JA3 18
3.3 Digitálny odtlačok servera JA3S 19
3.4 Server name indication 20
3.4.1 Levenshteinova vzdialenosť 21
3.5 Skript pre vytváranie databázy 22
3.5.1 Vytvorenie databázového súboru 22
3.5.2 Extrakcia dát a ich následné ukladanie do databázy 23
3.6 Zhrnutie 25

4 Detekcia aplikácií **26**
4.1 Princíp detekcie aplikácií 26
4.2 Testovanie 27
4.2.1 Testovanie nad známymi dátami 28
4.2.2 Testovanie na nevidených dátach 28
4.3 Zhrnutie 31

5 Experimenty **32**
5.1 Experimenty so súbormi pcap 32

5.2 Experimenty s digitálnymi odtlačkami 33
5.2.1 JA3 experimenty 34
5.2.2 JA3S experimenty 35
5.3 Experimenty s SNI 36
5.3.1 Získavanie SNI pomocou názvu aplikácie 36
5.3.2 Získavanie SNI pomocou kľúčových slov 36
5.4 Zhrnutie 37

6 Záver **38**

Literatúra **39**

A Opis datasetu **40**
A.1 Používané aplikácie 40
A.2 Analýza datasetu 40
A.3 Metriky spúšťania skriptu na extrakciu záujmových dát 41

Kapitola 1

Úvod

Cieľom tejto bakalárskej práce je vytvoriť nástroj na automatizované skenovanie internetovej prevádzky na sieti, vytváranie databázy digitálnych odtlačkov [3] a následné rozpoznávanie aplikácií na základe takto vytvorenej databázy. Táto databáza by následne mohla byť využitá napríklad pre štatistiku používaných aplikácií v sieti a identifikáciu konkrétneho používateľa pomocou jeho digitálneho profilu používaných aplikácií (napríklad osoby páchajúce trestnú činnosť). Pre úspešné vytvorenie odtlačku je potrebné oboznámiť sa s možnosťami emulácie mobilných aplikácií, princípom fungovania šifrovanej komunikácie pomocou protokolu TLS a metódami pre vytváranie digitálneho odtlačku JA3 a JA3S[1].

Kapitola 2 je venovaná nástrojom pre emuláciu mobilných zariadení s operačným systémom Android, simuláciu prevádzky a odchytávanie potrebných paketov. Konkrétne ide o nástroje Android Studio, ADB (Android Debug Bridge) a Wireshark (v mojej práci využívam jeho knižničnú verziu pre jazyk Python - PyShark). Pomocou týchto nástrojov a s využitím programovacieho jazyka Python je možné plne automatizovať inštaláciu, emuláciu aplikácií a následný zber dát pre vytváranie odtlačku aplikácií.

V kapitole 3 sa čitateľ oboznámi s metódami digitálneho vytvárania profilu JA3 (klient) a JA3S (server). Je tu opísaný princíp získavania potrebných dát, ako aj samotná informácia o tom, ktoré dáta a prečo sú potrebné a správne pre vytváranie fingerprintu. Ďalej sa čitateľ dozvie o spôsoboch hashovania týchto informácií a ich následnej distribúcii a využití. Pre potrebu plného porozumenia týmto metódam je v kapitole priblížený princíp fungovania protokolu TLS.

V priebehu práce bude vysvetlený postup a konkrétna implementácia opísaného nástroja. Nástroj sa skladá z modulov pre inštaláciu a emuláciu aplikácií, zber paketov a ich následné spracovanie a finálne vytváranie databázy. V kapitole 4 bude opísaný modul pre detekciu aplikácií z dát internetovej prevádzky. Kapitola obsahuje vysvetlenie princípu detekcie aplikácií, spôsoby testovania detekcie na známych a neznámych dátach a ich výsledky. Nástroj je implementovaný v jazyku Python3.

Výstupom tejto práce sú dátové sady rôznych mobilných aplikácií pre platformu Android[2], uložené ako SQLite databáza[3], ktoré bude možné následne použiť pri detekcií aplikácií. Súčasťou práce je taktiež nástroj na extrakciu týchto dát z paketov internetovej komunikácie, ako aj výsledky mojich experimentov s týmito dátami a nástroj na detekciu prítomnosti aplikácií v internetovej prevádzke pomocou týchto dátových sád.

[1] https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967 [3.2.2021]
[2] https://www.android.com/ [8.4.2021]
[3] https://www.sqlite.org/index.html [8.4.2021]

Kapitola 2

Emulácia aplikácií a odchytávanie prevádzky

V tejto kapitole sú opísané nástroje pre emuláciu aplikácií a odchytávanie prevádzky. Fungovanie týchto nástrojov však nebude rozobraté dopodrobna, keďže to nie je hlavná časť mojej práce, no po prečítaní tejto kapitoly by mal čitateľ porozumieť princípom používania daných nástrojov.

Pre potreby mojej práce využívam mobilný operačný systém Android od spoločnosti Google[1]. Na emuláciu zariadenia s týmto operačným systémom som preto zvolil integrované vývojové prostredie Android Studio[2], ktoré popri mnohých iných funkciách poskytuje možnosť vytvorenia virtuálneho stroja. Pre komunikáciu s emulátorom priamo zo zdrojového kódu využívam Android Debug Bridge, hlavne jeho nástroj monkey, opísaný v časti 2.2.3. Následný zber paketov zaobstaráva software Wireshark, v mojej implementácií využívam knižnicu PyShark.

2.1 Android Studio

Android Studio je oficiálne integrované vývojové prostredie od firmy Google pre vývoj mobilných aplikácií pre operačný systém Android. Je postavené na prostredí pre programovací jazyk Java - IntelliJ, ktorého autorom je spoločnosť JetBrains[3]. Preferovaný jazyk je Kotlin, no podporované sú aj Java a C++. Súčasťou tohto prostredia je mnoho užitočných nástrojov, ako napríklad profilovanie v reálnom čase, ktoré ponúka vývojárovi aktuálny prehľad spotreby pamäte, využitia CPU a siete v jeho zariadení, flexibilný systém zostavenia aplikácií, či rýchly a spoľahlivý emulátor, ktorý je spomenutý v podkapitole 2.1.1. Toto vývojové prostredie je dostupné na platformách Windows, macOS a Linux.

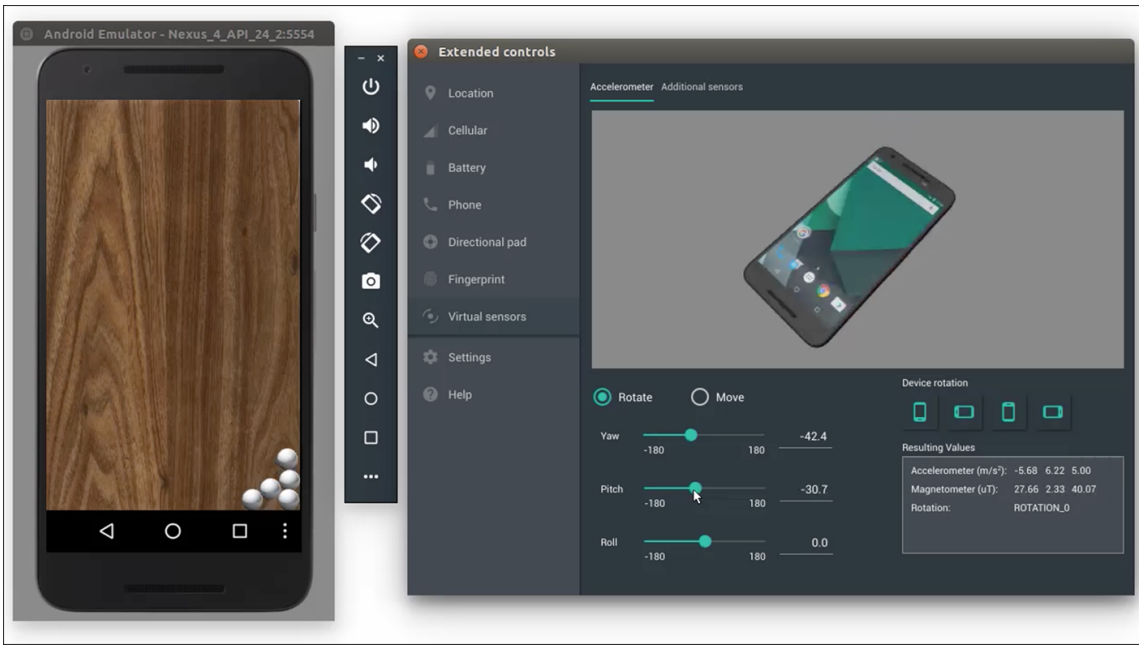
2.1.1 Emulátor

Jednou z mnohých možností Android Studia je emulácia virtuálneho zariadenia. Emulátor poskytuje mnoho funkcií reálneho fyzického zariadenia. Je k nemu možné pristupovať pomocou grafického užívateľského rozhrania alebo programovo pomocou konzoly, či pomocou Android Debud Bridge, ktorý je spomenutý v sekcii 2.2. Emulátor obsahuje implicitné konfigurácie pre Android phone, tablet, Wear OS a Android TV. Konfiguráciu je však možné

[1] https://www.android.com/ [8.4.2021]
[2] https://developer.android.com/studio [8.4.2021]
[3] https://www.jetbrains.com/ [8.4.2021]

ľubovoľne upravovať, nastaviteľná je veľkosť úložiska a pamäte RAM, užívateľ môže zmeniť Android API a taktiež je tu možnosť prednej a zadnej kamery, ktorú je možné buď úplne emulovať, alebo ju napojiť na kameru počítača.

Používanie je veľmi jednoduché a intuitívne, užívateľ je schopný vybrať si spomedzi viacerých typov zariadení (samozrejme len tie, ktoré sú vyrábané spoločnosťou Google), taktiež má na výber verziu operačného systému, ako aj jeho architektúru. Emulované prostredie sa teda dá prispôsobiť užívateľovým preferenciám. Ja som si pre účely tejto práce zvolil dva mobilné telefóny a dva tablety. Zvolil som aj rozličné verzie operačného systému a veľkosti obrazovky. Z mobilných telefónov som zvolil Google Pixel 2 s operačným systémom Android 7.0 a Google Pixel 3XL s operačným systémom Android 10.0. Pre emuláciu tabletov využívam Google Pixel C s Android 8.1 a Google Nexus 10 s Android 6.0.



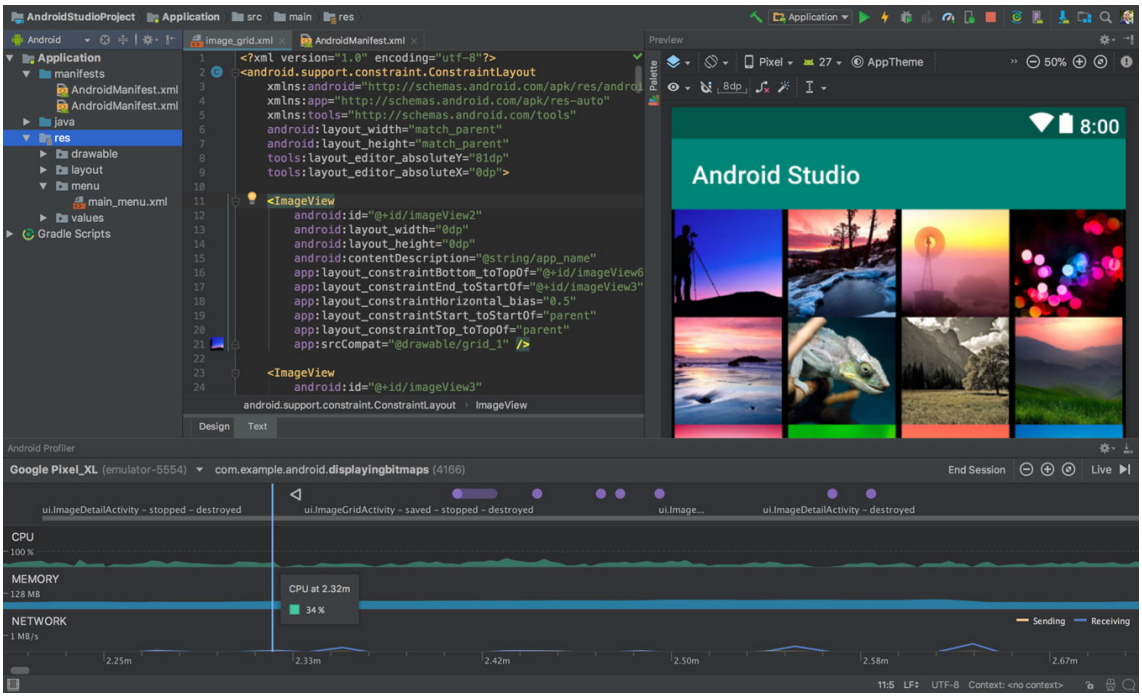
Obr. 2.1: Android Studio emulátor

2.2 Rozhranie Android Debug Bridge

Android Debug Bridge (ďalej ako ADB) je všestranný nástroj typu klient-server pre komunikáciu so zariadením operačného systému Android. Poskytuje viacero príkazov, napríklad pre inštaláciu či debug aplikácií, listovanie a správu všetkých nainštalovaných balíčkov, prístup k logovacím správam a pod. ADB taktiež poskytuje unixový shell, ktorý umožňuje spúšťanie rôznych príkazov na zariadení. Tento nástroj je možné získať ako súčasť balíčku *Android SDK Platform-Tools*. Fungovanie ADB a jeho komunikáciu s mobilným zariadením zabezpečujú jeho tri súčasti:

- *Démon (adbd)* - proces, ktorý beží na pozadí každého pripojeného zariadenia. Démon má pridelený svoj port a je zodpovedný za spúšťanie príkazov na danom zariadení.
- *Klient* - rozhranie, ktoré spúšťa príkazy na cieľovom zariadení. Klient beží na riadiacom zariadení a je možné ho spustiť z príkazového riadku pomocou príkazu *adb*.

- *Server* - proces, ktorý beží na pozadí riadiaceho zariadenia a zabezpečuje komunikáciu medzi démonom a klientom.



Obr. 2.2: Integrované vývojové prostredie Android Studio

2.2.1 Spustenie ADB

Pe spustenie tohto nástroja je potrebné nainštalovať si *Android SDK Platform-Tools*, ktorý je stiahnuteľný pomocou nástroja SDK Manager[4] a následne zadať do konzoly príkaz *adb*, čím si na svojej mašine spustíme ADB klienta.

Akonáhle je klient spustený, skontroluje, či na danom zariadení beží proces serveru. Pokiaľ žiadny proces nie je aktívny, ADB klient ho spustí. Každý klient posiela príkazy serveru na *lokálny TCP port 5037*. Server sa preto okamžite po štarte na tento port naviaže a očakáva príkazy od klienta.

Po naviazaní na port začne server prehľadávať vývojové zariadenie pre pripojené Android zariadenia. Prehľadávanie prebieha na nepárnych portoch v rozsahu 5555 - 5585. Nepárne porty sa používajú preto, lebo každé zariadenie používa nepárny port pre pripojenie k ADB, zatiaľ čo používa párny port pre pripojenie ku konzole. Akonáhle nájde aktívne zariadenie - aktívneho démona, vytvorí si s ním spojenie. Keď server úspešne vytvorí spojenie pre všetky zariadenia, môžeme začať využívať ADB príkazy. Keďže správu pripojení a vykonávanie príkazov má na starosti jeden server, môžeme ku ktorémukoľvek zariadeniu pristúpiť z ktoréhokoľvek klienta alebo skriptu.

[4]https://developer.android.com/studio/releases/platform-tools [8.4.2021]

2.2.2 Inštalácia aplikácií

Jedna z funkcií, ktorú ADB ponúka, je inštalácia mobilných aplikácií zo súboru typu *.apk*. Pri zadávaní akýchkoľvek príkazov pre jedno pripojené zariadenie nie je potrebné špecifikovať jeho názov. Pokiaľ je však zariadení ku počítaču pripojených viac, je nutné zistiť si jeho identifikátor. Zobraziť všetky pripojené zariadenia môžeme pomocou príkazu

```
$ adb devices
```

Tento príkaz nám poskytne nasledujúce informácie:

- *Sériové číslo* - unikátny identifikátor, väčšinou zvolený na základe portu, na ktorom je zariadenie pripojené.
- *Stav pripojenia zariadenia*
  - *device* - zariadenie je pripojené a funkčné
  - *offline* - zariadenie je odpojené alebo neodpovedá
  - *no device* - žiadne zariadenie nie je pripojené
- *Popis* - popis zariadenia, zobrazuje sa len pri použití prepínača *-l*.

Sériové číslo zariadenia môžeme použiť na jeho identifikáciu v ďalších príkazoch, konkrétne pomocou prepínača *-s*. Pokiaľ však plánujeme zadávať viac príkazov jednému zariadeniu, nemusíme explicitne uvádzať sériové číslo, ale môžeme ho nahrať do systémovej premennej `$ANDROID_SERIAL`. Pokiaľ používame prepínač a aj systémovú premennú, prepínač potláča premennú. Pre inštaláciu aplikácie použijeme príkaz *install*. Výsledný príkaz bude teda vyzerať takto:

```
$ adb -s seriove_cislo install nazov_appky.apk
```

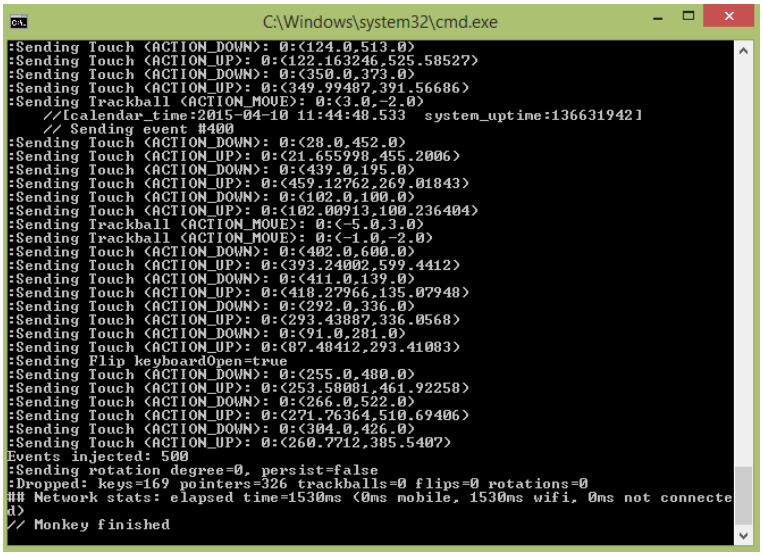
Ako príklad použijem emulátor so sériovým číslom 'emulator-5554' a aplikáciu 'google-chrome'. Pre inštaláciu tejto aplikácie na emulovanom zariadení by príkaz vyzeral nasledovne:

```
$ adb -s emulator-5554 install google-chrome.apk
```

2.2.3 Simulátor monkey

Po úspešnej inštalácii je nutné aplikáciu používať, aby generovala nejakú prevádzku na sieti, ktorú je možné zachytiť a ďalej spracovávať. Nástroj známy ako *monkeyrunner*[5], ktorý funguje pod ADB, je schopný generovať na fyzickom či emulovanom Android zariadení pseudonáhodné udalosti na obrazovke, čím vie spoľahlivo vytvoriť dojem, že je aplikácia používaná reálnym koncovým užívateľom. Monkey dokáže generovať dotyky, kliknutia, gestá, ako aj niekoľko udalostí systémovej úrovne, viď obrázok 2.3.

[5] https://developer.android.com/studio/test/monkeyrunner



Obr. 2.3: Ukážka zasielania toku udalostí do zariadenia pomocou monkey

Monkey je konzolová aplikácia, vytvorená za účelom záťažových testov Android aplikácií. Jej princíp fungovania spočíva v posielaní pseudonáhodných tokov udalostí do testovaného zariadenia. Prístup k tejto aplikácii je možný cez konzolu na riadiacom zariadení alebo pomocou skriptu. Pretože je monkey spustený v prostredí zariadenia/emulátoru, je potrebné spúšťať ho priamo z jeho shellu, alebo za pomoci príkazu *adb shell*. Základná syntax využívania monkey potom vypadá nasledovne:

```
$ adb shell monkey [prepinac] -v pocet_udalosti
```

V tejto práci nebudem opisovať všetky prepínače, ktoré je možné používať. Dôležitá informácia však je, že pokiaľ žiadny z prepínačov nevyberieme, monkey sa spustí v takzvanom non-verbose alebo tichom režime a začne posielať toky udalostí do všetkých balíčkov nainštalovaných na zariadení. Preto je omnoho bežnejší spôsob spustenia spolu s prepínačom *-p*, ktorý špecifikuje balíček, do ktorého sa má tok udalostí odoslať. Syntax príkazu vyzerá nasledovne:

```
$ adb shell monkey -p nazov_balicku -v pocet_udalosti
```

Uvažujme balíček aplikácie Google Chrome s názvom 'com.android.google-chrome'. Pokiaľ by sme chceli, aby monkey vygeneroval 200 pseudonáhodných udalostí pre tento balíček, vyššie spomínaný príkaz použijeme takýmto spôsobom:

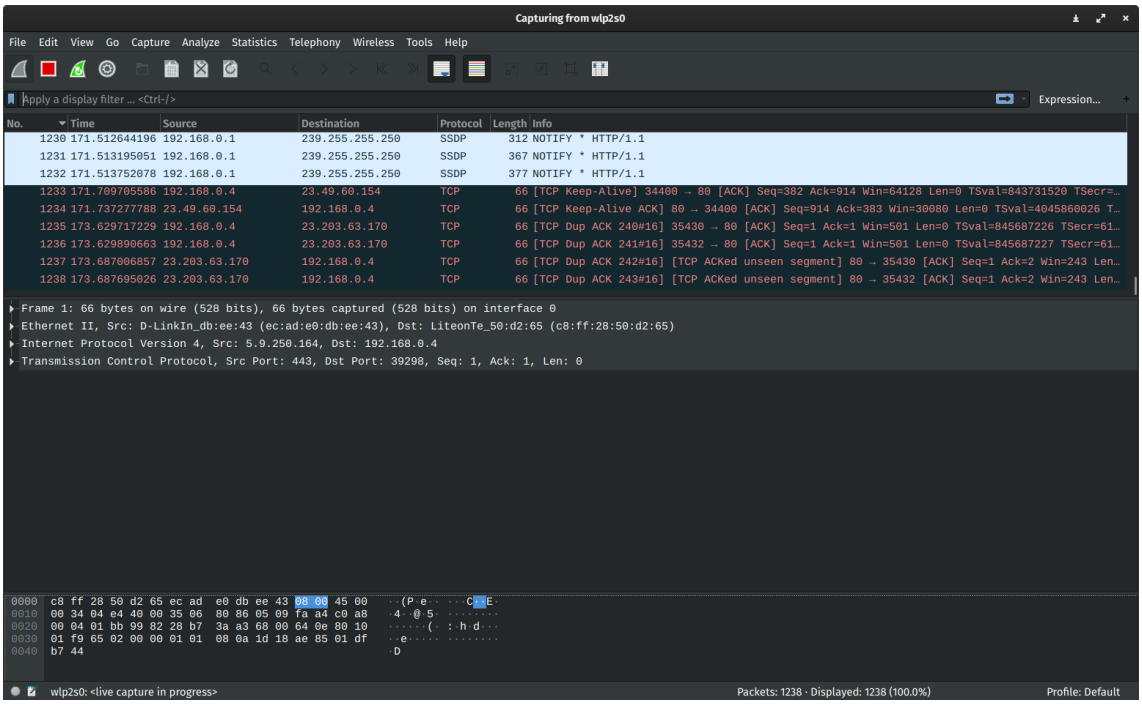
```
$ adb shell monkey -p com.android.google-chrome -v 200
```

2.3 Sieťový analyzátor Wireshark

Wireshark[6] je multiplatformná počítačová aplikácia pre monitorovanie a analýzu internetovej prevádzky na základe paketov prichádzajúcich a odchádzajúcich v sieti. Poskytuje

[6] https://www.wireshark.org/ [8.4.2021]

podobné funkcie ako známy nástroj *tcpdump*[7], ale na rozdiel od neho má grafické užívateľské rozhranie postavené na *Qt widget toolkit*[8], disponuje oveľa väčším množstvom analyzérov komunikačných protokolov a má mnoho možností filtrovania či zobrazovania informácií. Pre zber paketov využíva programovacie aplikačné rozhranie *pcap*[9].



Obr. 2.4: Odchytávanie internetovej prevádzky pomocou Wireshark

Wireshark dokáže odposlúchavať prevádzku buď na konkrétnom sieťovom rozhraní zvolenom užívateľom, alebo na všetkých naraz. Jednou z možností je, pokiaľ to dané rozhranie podporuje, prepnúť ho do tzv. *promiscuous mode*, kedy je možné zachytiť všetku komunikáciu na sieti, lebo kontrolér tohto rozhrania nebude na CPU posielať len tie dáta, ktoré je naprogramovaný prijímať, ale bude preposielať celú komunikáciu. Podľa IEEE 208[10] každý rámec obsahuje cieľovú MAC adresu. Pokiaľ je teda rozhranie v *non-promiscuous mode*, kontrolér automaticky zahadzuje každý paket, ktorý mu nepatrí alebo nie je multicast či broadcast. Táto možnosť teda sprístupňuje skenovanie aj tej prevádzky, ktorá je určená iným zariadeniam v sieti.

Wireshark vykonáva dobrú prácu v oblasti spracovania a vyobrazenia štruktúry rozličných sieťových protokolov. Dokáže analyzovať a zobraziť všetky polia spolu s ich významom tak, ako to dané protokoly definujú. Formát jeho vstupných či výstupných súborov je libpcap, čo znamená, že si dokáže vymieňať dáta s ostatnými libpcap aplikáciami, ako napríklad spomínaný tcpdump. Taktiež je schopný prečítať súbory iných analyzátorov, ako napríklad snoop[11] či Microsoft Network Monitor[12].

[7] https://www.tcpdump.org/ [8.4.2021]
[8] https://www.qt.io/ [8.4.2021]
[9] https://www.tcpdump.org/manpages/pcap.3pcap.html [8.4.2021]
[10] https://standards.ieee.org/standard/208-1995.html [8.4.2021]
[11] http://www.softpanorama.org/Net/Sniffers/snoop.shtml [8.4.2021]
[12] https://docs.microsoft.com/en-us/windows/win32/netmon2/network-monitor [8.4.2021]

2.3.1 PyShark

Okrem desktopovej aplikácie Wireshark obsahuje aj konzolové riešenie *thsark*[13]. Keďže implementačný jazyk mojej práce je Python, mnou využívaná varianta je Python wrapper pre tshark zvaný PyShark[14].

Existuje viacero možností, ako pracovať s paketmi v jazyku Python. Ja som však zvolil PyShark práve preto, že používa analyzátory, ktoré sú nainštalované spolu s Wiresharkom. Poskytuje možnosť aktívneho skenovania prevádzky na sieti, ako aj čítanie z offline súboru.

Čítanie z offline súboru je vykonávane funkciou *FileCapture()*. Príklad použitia tejto funkcie je v nasledujúcej ukážke:

```
import pyshark
cap = pyshark.FileCapture('/tmp/mycapture.cap')

print cap[0]
Packet (Length: 698)
Layer ETH:
       Destination: aa:bb:cc:dd:ee:ff
       Source: 00:de:ad:be:ef:00
       Type: IP (0x0800)
Layer SSL:
       TLSv1 Record Layer: Handshake Protocol: Client Hello
       Content Type: Handshake (22)
       Version: TLS 1.0 (0x0301)
       Length: 205
       Handshake Protocol: Client Hello
       Handshake Type: Client Hello (1)
       Length: 201
       Version: TLS 1.2 (0x0303)
```

Monitorovanie prevádzky zabezpečuje funkcia *LiveCapture()*, ktorá ako parameter očakáva rozhranie, na ktorom chceme prevádzku monitorovať. Pokiaľ žiadny nedostane, monitorovanie prebieha na všetkých dostupných rozhraniach. Spracovanie jednotlivých paketov je možné urobiť viacerými spôsobmi. Jedna z možností je použitie funkcie *sniff()*, ktorá zbiera zadané množstvo paketov (alebo po zadaný čas) a ukladá ich ako objekt, ktorý je následne možné prechádzať ako list alebo je možné definovať funkciu callback(), ktorá sa aplikuje na každý prichádzajúci paket.

```
import pyshark
capture = pyshark.LiveCapture(interface='eth0')

#pouzitie funkcie sniff
capture.sniff(timeout=50)
capture
>>> <LiveCapture (5 packets)>

#defincia callback funkcie
```

[13] https://www.wireshark.org/docs/man-pages/tshark.html [8.4.2021]
[14] https://github.com/KimiNewt/pyshark/ [8.4.2021]

```
def callback(pkt):
    #spracovanie paketu
capture.apply_on_packets(callback, timeout=5)
```

2.4 Skript pre inštaláciu aplikácií a emuláciu prevádzky

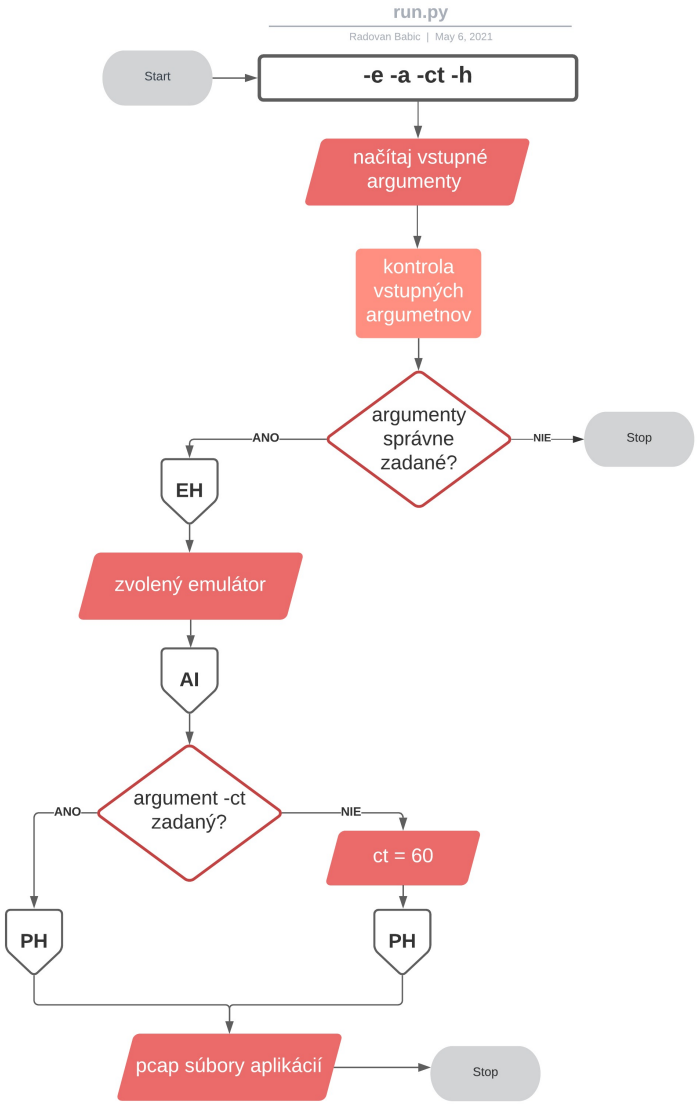
Všetky nástroje spomenuté v tejto kapitole sú nevyhnutné pre vytvorenie databázy odtlačkov aplikácií. Pre potreby automatizovanej inštalácie aplikácií a následnému generovaniu a odchytávaniu prevádzky som implementoval moduly *AppInstaller*, *EmulatorHandler* a *PacketHandler*, ktoré tieto nástroje využívajú pri tvorení dátových setov s použitím emulovaného zariadenia.

2.4.1 Spustenie nástroja

Hlavné spustenie nástroja, a teda spustenie modulov opísaných nižšie, má na starosti skript *run.py*. Tento skript na svojom vstupe prijíma tieto argumenty:

- -e : názov emulátoru, nepovinný argument
- -a : inštalačný súbor/zložka s inštalačnými súbormi pre aplikácie, povinný argument
- -ct : capture timeout, t.j. čas, po ktorom sa zastaví skenovanie prevádzky, nepovinný argument
- -h : vypíše nápovedu pre spustenie skriptu, nepovinný argument

Pokiaľ boli zadané všetky povinné argumenty, skript začína svoju prácu spustením modulu *EmulatorHandler*.



Obr. 2.5: Vývojový diagram skriptu run.py

2.4.2 Modul EmulatorHandler

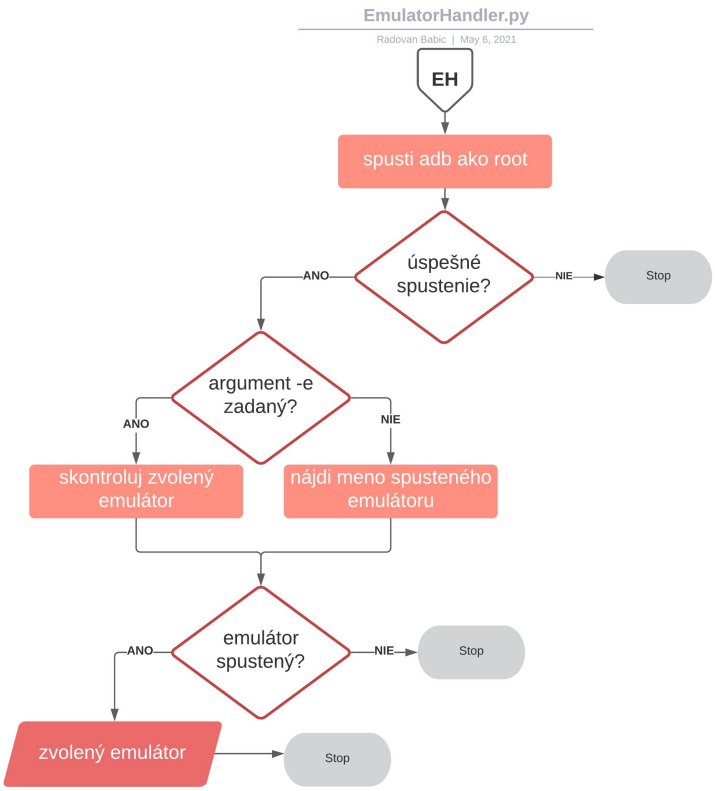
Ako prvý na rad prichádza modul zodpovedný za ovládanie emulátora, keďže bez neho nie je možné žiadne dáta generovať. Pre používanie môjho nástroja je však ako prerekvizita nutné mať emulátor zapnutý predom. Je síce možné emulátor naštartovať z prostredia terminálu, ale zautomatizovať tento proces by bolo podľa môjho názoru nepraktické. Pre spustenie emulátora je potrebné vytvoriť AVD (Android Virtual Device), čo bolo spomenuté v podkapitole 2.1.1. Ten sa dá vytvoriť jedine z prostredia Android Studio pomocou sprievodcu, v ktorom si užívateľ zvolí typ zariadenia a verziu operačného systému.

Pre spustenie emulátoru z prostredia terminálu je nutné poznať názov vytvoreného AVD, ktoré je dostupné len v Android Studiu a pre jeho modifikáciu či pridanie je opäť nutný

prístup do tohto prostredia. Taktiež pre identifikáciu možných chýb v správaní aplikácií poskytuje grafické užívateľské rozhranie viac informácií.

Aby bolo možné odchytávať prevádzku z emulátora, je potrebné spustiť ADB ako správca (root). Pokiaľ je emulátor naštartovaný, *EmulatorHandler* po jeho spustení vyskúša vykonať príkaz, ktorý ADB prepne do root režimu. Ak sa počas spúšťania vyskytne chyba, skript končí a vypíše príslušné chybové hlásenie.

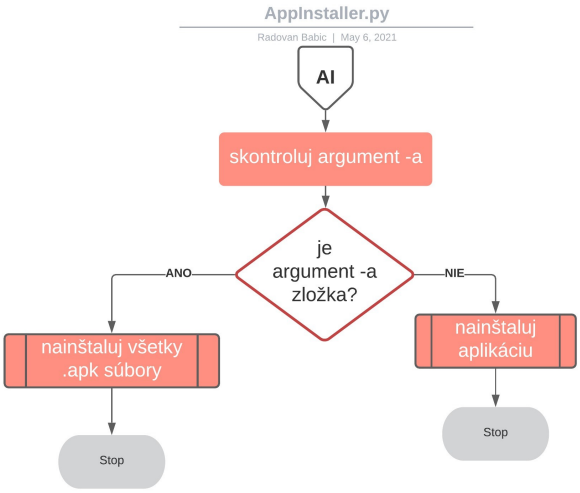
Názov emulátora môže byť zadaný manuálne pomocou argumentu *-e*. Pokiaľ je tento argument pri štarte vynechaný, skript sa pokúsi automaticky vyhľadať dostupné emulátory a použije prvý z nich. V opačnom prípade skript skončí s príslušným chybovým hlásením. Za predpokladu, že prepnutie ADB do režimu root prebehlo úspešne a je zvolený jeden z emulátorov, *EmulatorHandler* vráti túto informáciu modulu *AppInstaller*.



Obr. 2.6: Vývojový diagram skriptu EmulatorHandler.py

2.4.3 Modul AppInstaller

Modul *AppInstaller* je zodpovedný za inštaláciu aplikácií na emulované Android zariadenie. Ako prvá prebieha kontrola cesty zadaného argumentu *-a*. V prípade, že bola ako tento argument predaná cesta k jednému inštalačnému súboru, skript ho nainštaluje. Pokiaľ užívateľ zadal cestu k priečinku, skript sa pokúsi nainštalovať všetky súbory typu *.apk* na spustený emulátor. Ak sa nejaký súbor nepodarilo nainštalovať, skript nekončí, ale vypíše príslušné chybové hlásenie a názov súboru, pri ktorom problém nastal. Akonáhle tento modul dokončil inštaláciu aplikácií, nasleduje generovanie a odchytávanie prevádzky.



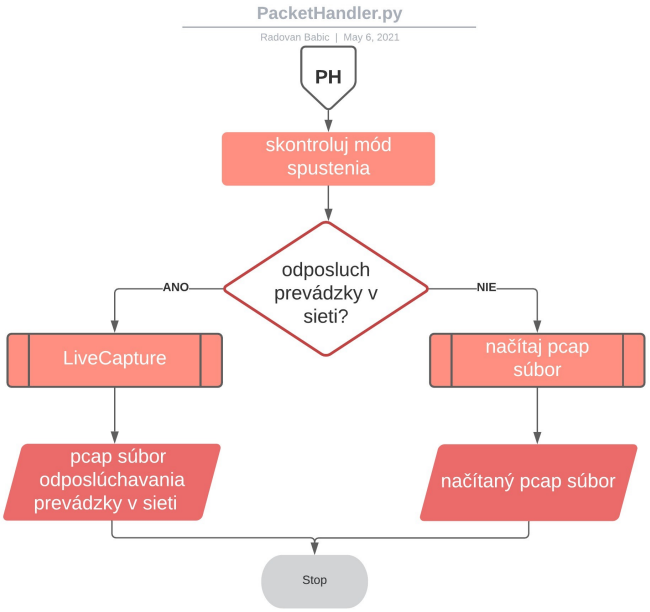
Obr. 2.7: Vývojový diagram skriptu AppInstaller.py

2.4.4 Modul PacketHandler

Modul *PacketHandler* má na starosti generovanie a zachytávanie prevádzky, ktorá bude následne využitá pre vytváranie digitálnych odtlačkov aplikácií. Využíva vyššie spomínaný nástroj Monkey (sekcia 2.2.3) a Wireshark (sekcia 2.3). Modul zoberie zaradom všetky aplikácie nainštalované v počas behu skriptu, postupne generuje pseudonáhodné udalosti a následne zbiera sieťové dáta generované aplikáciou.

Zber dát prebieha odposlúchaním internetovej prevádzky na takzvanom *extcap* rozhraní. Rozhranie extcap je univerzálne doplnkové rozhranie, ktoré umožňuje externým binárnym súborom pôsobiť ako rozhranie pre snímanie internetovej prevádzky priamo vo Wiresharku. Používa sa v scenároch, keď spôsob snímania nie je jeden z bežne používaných modelov (snímanie z klasického rozhrania v reálnom čase, z rúry (pipe), zo súboru atď.). Typickým príkladom je pripojenie emulovaného hardvéru istého druhu k hlavnej aplikácii Wireshark.

Po pripojení na toto rozhranie začne modul generovať a odchytávať prevádzku. Pokiaľ nebol čas stanovený argumentom *-ct*, odposlúchanie sa zastaví po šesťdesiatich sekundách a modul pakety uloží ako .pcap súbor do zložky */pcap/nazovAplikacie*. Tento modul však nie je spustený samostatne, slúži len ako pomocný modul pre skript *run.py* a pre nástroj opísaný v kapitole 3.5.2.



Obr. 2.8: Vývojový diagram skriptu PacketHandler.py

2.5 Zhrnutie

V tejto kapitole boli zhrnuté aplikácie, ktoré som využil pre emuláciu zariadení a pre generovanie a odchytávanie prevádzky. Keďže je mojím cieľom vytvoriť plne automatizovaný nástroj, bolo nevyhnutné ich použitie. Objasnený bol princíp, akým funguje Android Studio emulátor, akým spôsobom dokáže užívateľ používať virtuálne mobilné zariadenie, ako aj nástroj ADB pre následnú komunikáciu so zariadením pomocou konzoly alebo skriptov. Ďalšou podstatnou časťou mojej práce je zber a analýza dát, na čo je ideálny opísaný nástroj Wireshark. Tento nástroj poskytuje veľa možností spracovania dát, no hlavne konzolovú variáciu TShark. V kapitole boli opísané moduly mnou implementovaného nástroja pre emuláciu zariadenia, inštaláciu aplikácií a následného generovania a zberu dát určených pre vytváranie digitálnych odtlačkov. Čitateľ bol oboznámený so spôsobom fungovania každého modulu, ako aj s ich spustením pomocou hlavného riadiaceho skriptu.

Kapitola 3

Metódy tvorenia digitálneho TLS odtlačku JA3 a JA3S

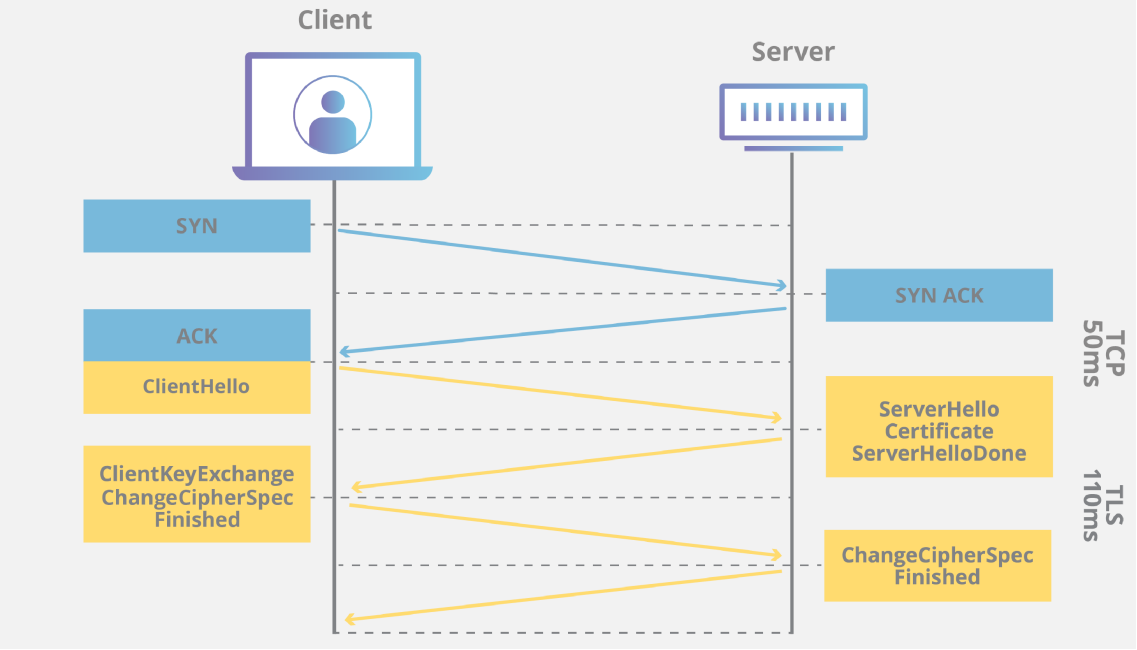
Keďže sa technologický priemysel rozširuje veľkou rýchlosťou a technológie používa čím ďalej tým viac ľudí, narastá preto aj potreba zvyšovania bezpečnosti, hlavne pokiaľ ide o internetovú prevádzku. V dnešnej dobe je už väčšina prevádzky šifrovaná a z tej nešifrovanej časti je už veľmi ťažké získať dostatočné informácie na to, aby sme boli schopní s adekvátnou úspešnosťou identifikovať aplikácie na sieti. V tejto kapitole sa čitateľ oboznámi s metódami vytvárania digitálneho profilu na základe nadviazania šifrovanej komunikácie. Konkrétne ide o metódy JA3 pre klienta a JA3S pre server. Pre dostatočné pochopenie týchto metód bude v tejto kapitole popísaný základný princíp nadväzovania zabezpečenej komunikácie pomocou protokolu TLS, ako aj informácia o tom, ktoré z jej vlastností používam pre potreby vytvorenia digitálneho odtlačku aplikácie.

3.1 Nadviazanie TLS komunikácie

Transport Layer Security (TLS) [2] je skupina kryptografických protokolov, ktoré poskytujú možnosť zabezpečenej šifrovanej komunikácie pre širokú škálu internetových služieb a dátových prenosov, čím výrazne znižujú riziko odposlúchania či falšovania správ. Princíp fungovania tohto protokolu je možné zhrnúť do troch bodov:

- Vzájomná dohoda zúčastnených strán na podporovaných algoritmoch.
- Asymetrická výmena kľúča pre symetrické šifrovanie a autentizácia na základe certifikátov.
- Symetrické šifrovanie internetovej prevádzky.

Takéto nadviazanie komunikácie pomocou tohto protokolu opísané v prvých dvoch bodoch sa nazýva TLS Handshake. Skladá sa z niekoľkých správ, no pre potreby vytvorenia digitálneho odtlačku aplikácie sú dôležité správy ClientHello (pre metódu JA3) a ServerHello (pre metódu JA3S). Z týchto správ by malo byť možné extrahovať informácie, ktoré môžu byť následne použité na identifikáciu konkrétnej aplikácie.



Obr. 3.1: Nadviazanie šifrovanej komunikácie - TLS Handshake

Správa ClientHello poskytuje informácie o tom, akú verziu protokolu TLS klient používa, aké všetky algoritmy podporuje a zoznam rozšírení, ktoré sa však už od klienta ku klientovi môžu líšiť počtom aj typom. Správa ClientHello je podľa RFC 8446 [2] definovaná nasledovne:

```
uint16 ProtocolVersion;
opaque Random[32];

uint8 CipherSuite[2]; /* Cryptographic suite selector */

struct {
    ProtocolVersion legacy_version = 0x0303; /* TLS v1.2 */
    Random random;
    opaque legacy_session_id<0..32>;
    CipherSuite cipher_suites<2..2^16-2>;
    opaque legacy_compression_methods<1..2^8-1>;
    Extension extensions<8..2^16-1>;
} ClientHello;
```

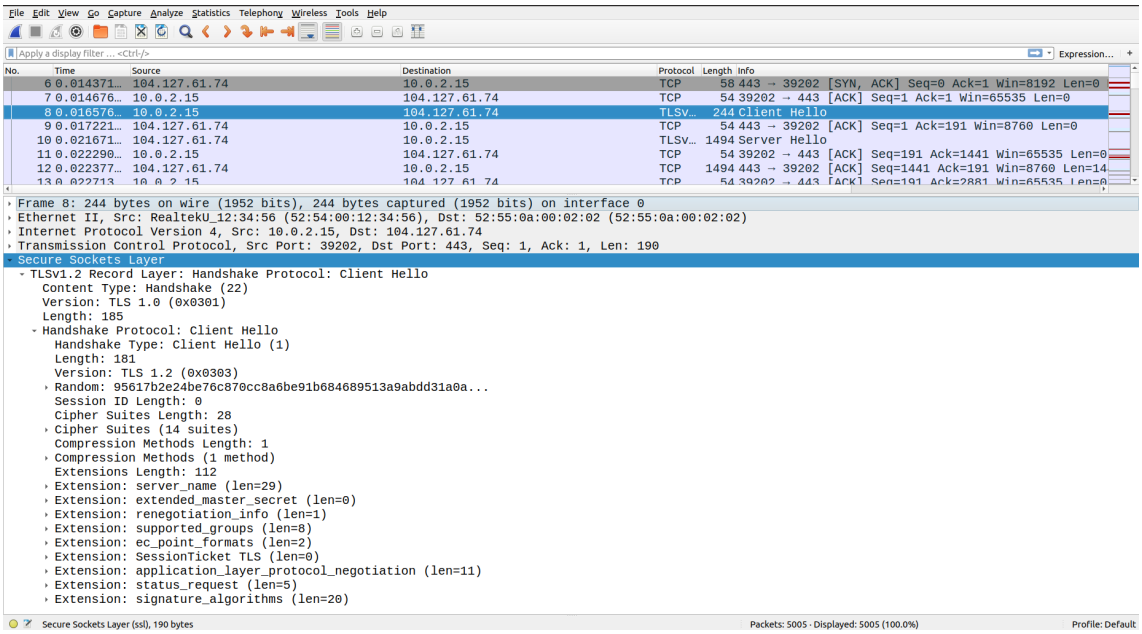
V správe ServerHello server presne špecifikuje, ktorý z klientom poskytnutých algoritmov bude použitý v rámci tejto komunikácie a taktiež obsahuje zoznam rôznych rozšírení. Z týchto informácií je následne možné použitím metód JA3 a JA3S vytvoriť digitálny odtlačok aplikácie, ktorý následne použijeme na jej klasifikáciu a rozpoznanie.

Správa ServerHello je v RFC 8446 definovaná nasledovne:

```
struct {
    ProtocolVersion legacy_version = 0x0303; /* TLS v1.2 */
    Random random;
    opaque legacy_session_id_echo<0..32>;
    CipherSuite cipher_suite;
    uint8 legacy_compression_method = 0;
    Extension extensions<6..2^16-1>;
} ServerHello;
```

3.2 Digitálny odtlačok klienta JA3

JA3 je metóda vytvárania digitálnych odtlačkov zo šifrovanej TLS komunikácie pre klienta pomocou dát získaných zo správ ClientHello. Bola vytvorená spoločnosťou Salesforce[1] za účelom identifikácie škodlivého softvéru na sieti. Metóda funguje na princípe extrakcie záujmových dát z hlavičiek zabezpečenej komunikácie - SSL hlavičiek. Ide o klasické odchytávanie internetovej prevázdky, napríklad pomocou vyššie spomínaného nástroja Wireshark, opísaného v kapitole 2.3, kedy nás vždy zaujímajú len pakety, ktoré majú SSL hlavičku a sú v tomto prípade typu ClientHello.



Obr. 3.2: Príklad paketu so SSL hlavičkou zachyteného pomocou nástroja Wireshark

Podľa metódy JA3 sa odtlačok počíta z nasledujúcich SSL záznamov:

- Verzia protokolu TLS.
- Veľkosť zoznamu podporovaných šifier.
- Zoznam klientom podporovaných šifier.

[1] https://www.salesforce.com/eu/ [8.4.2021]

- Zoznam rozšírení (konkrétne číselných hodnôt označujúcich ich typ).
- Rozšírenie elliptic_curves.
- Rozšírenie ec_point_formats.

Tieto dáta poskytujú dostatočné množstvo informácií[2] na to, aby bolo možné na sieti vypozorovať nezvyčajnú komunikáciu a odhaliť tak škodlivý softvér. Následne sú tieto dáta uložené ako predloha, podľa ktorej daný malvér komunikuje na internetovej sieti - jeho digitálny odtlačok. Autori taktiež tvrdia, že počet rôznych variácií a odtlačkov je konečný a rozumne vysoký. Pri použití tejto metódy by teda malo byť možné jednoznačne identifikovať konkrétnu aplikáciu.

Môj prieskum a praktické odskúšanie prevedenia tejto metódy však v správaní mobilných aplikácií odhalili menšie rozdiely, kvôli ktorým som musel metódu JA3 upraviť, aby som bol schopný vytvárať digitálne odtlačky. Po spracovaní dát, ktoré je opísané v podkapitole 3.5, som zistil, že v mnou odchytenej komunikácii sa nenachádzalo rozšírenie *elliptic_curves*, aj keď by sa na základe JA3 malo. Tým som prišiel o značnú časť dôležitých informácií. Pri pozorovaní paketov som si však všimol, že sa v nich nachádza rozšírenie *signature_algorithms*, ktoré obsahuje zoznam podporovaných šifrovacích algoritmov. Rozhodol som sa toto rozšírenie použiť namiesto chýbajúceho, keďže by mi mohlo, na základe môjho predpokladu, poskytnúť informácie potrebné na klasifikáciu aplikácie. Taktiež ako prvý element Salesforce využíva verziu TLS. V komunikácii sa však vyskytujú verzie dve: verzia záznamu a verzia handshake, viď kapitola 3.1. Preto do odtlačku pridávam obe tieto verzie.

Všetky tieto záujmové dáta sú potom zlúčené do jedného reťazca znakov na základe konkrétnych pravidiel. Každá položka (napr. verzia, či rozšírenie) je oddelená čiarkou a v rámci každej položky sú jej elementy (napr. konkrétne rozšírenie zo zoznamu) oddelené pomlčkami. Výsledný reťazec môže potom vyzerať napríklad takto:

769,771,28,49195-49196-52393-49199-49172-156-157-47-53,1027-2052-2054-1537-513,0

Pre jednoduchšiu prácu s týmto reťazcom je z neho vypočítaný matematický hash pomocou algoritmu MD5 [3] a tento hash je následne považovaný za digitálny odtlačok danej aplikácie - tzv. fingerprint. Výsledný odtlačok zo spomínaného reťazca by vyzeral nasledovne:

aa5012c845c5daa5181fee9ac6d11bb4

Odtlačok v tejto podobe je následne uložený do databázy. Spôsob tohto ukladania je takisto opísaný v podkapitole 3.5.

3.3 Digitálny odtlačok servera JA3S

Metódy JA3 a JA3S sú vo svojej podstate rovnaké. Obe metódy totiž pracujú so záujmovými dátami z hlavičky šifrovanej SSL komunikácie. V tomto prípade však ide o správu ServerHello a tým pádom táto metóda využíva rozdielne záujmové dáta. Implementácia metódy definuje používanie týchto položiek:

- Verzia protokolu TLS.

[2] https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967 [8.4.2021]

- Serverom zvolená šifra.
- Zoznam rozšírení (opäť v podobe čísiel vyjadrujúcich typ).

Tieto dáta sa v mojej komunikácii nachádzajú, preto som bol schopný vytvoriť odtlačok servera bez problémov. Podobne ako u JA3 aj tu sa verzie TLS nachádzali dve a rovnako ich teda používam obe. Vytváranie reťazca a jeho hashovanie je rovnaké ako pri JA3. Príklad reťazca odtlačku servera:

771,771,49195,23-65281-11-35-16

Príklad jeho hashu pomocou MD5:

ca6c8c36eef8c4aa318fd134d96c110a

Jedno moje zistenie však vyvracia tvrdenie vývojárov zo Salesforce. Podľa ich článku klient so serverom môže komunikovať rôznymi spôsobmi, preto sa jeho odtlačkov v komunikácii vyskytuje niekoľko. Avšak tvrdia, že server s klientom vždy komunikuje len jedným spôsobom, a preto bude mať len jeden odtlačok. Počas môjho prieskumu som však zistil, že serverových odtlačkov sa v komunikácii nachádzalo vždy niekoľko, vo väčšine prípadov dokonca výrazne viac ako tých klientských. Viac v kapitole 5.

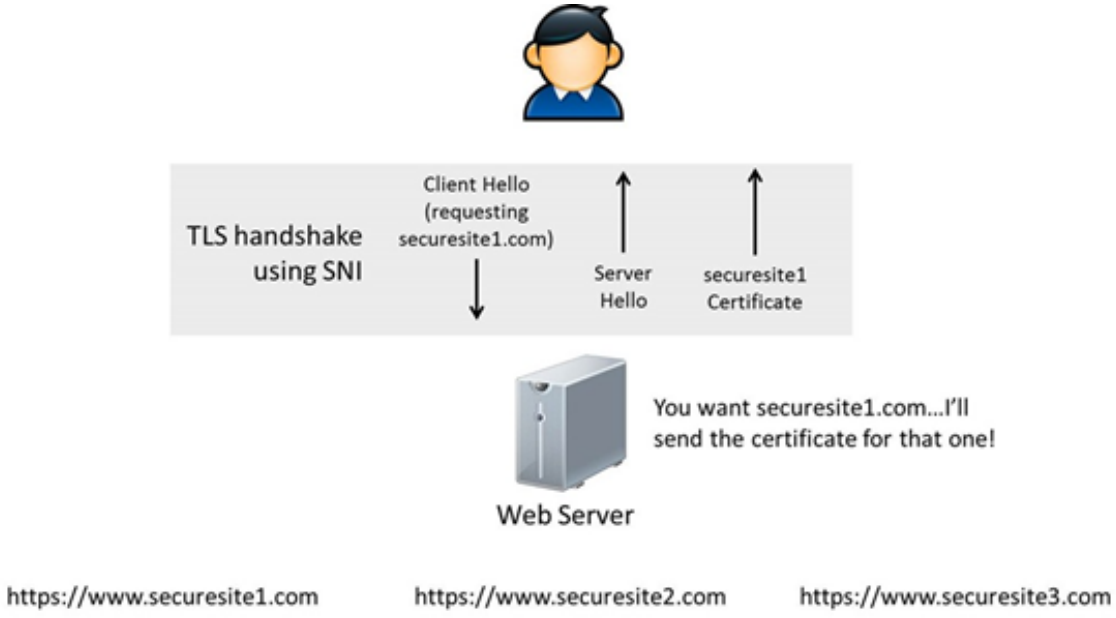
3.4 Server name indication

Počas mojich experimentov s digitálnymi odtlačkami som zistil, že detekcia len pomocou metód JA3 a JA3S nie je presná, pretože sa odtlačky vo veľmi vysokej miere medzi aplikáciami prekrývajú (viac v kapitole 5). Pre zvýšenie presnosti detekcie aplikácií som na podnet vedúceho tejto práce preskúmal možnosť brať do úvahy ešte jednu položku TLS komunikácie, a to SNI (server name indication).

SNI je jedno z rozšírení sieťového protokolu TLS, pomocou ktorého klient na začiatku procesu nadviazania spojenia indikuje, ku ktorému hostiteľovi sa pokúša pripojiť (poskytne jeho doménové meno). Požadované doménové meno nie je šifrované, takže útočník môže vidieť, na ktorý server klient posiela svoju požiadavku, čo v mojej práci využívam aj ja.

Ako už bolo v práci spomenuté, pri vytváraní TLS spojenia si klient vyžiada digitálny certifikát z webového servera. Klient ho následne preskúma a porovná meno, ku ktorému sa pokúšal pripojiť, s menami zahrnutými v certifikáte. Ak dôjde k zhode, pripojenie bude pokračovať ako obvykle. Ak sa nenájde zhoda, môže byť používateľ upozornený na nezrovnalosť a pripojenie sa môže prerušiť, pretože nesúlad môže naznačovať pokus o útok typu man-in-the-middle.

Je veľmi ťažké až nemožné, obzvlášť z dôvodu nedostatku úplného zoznamu všetkých mien vopred, získať jediný certifikát, ktorý by pokrýval všetky mená, za ktoré bude server zodpovedný. Server, ktorý je zodpovedný za viac mien hostiteľov, by v tomto prípade musel predložiť iný certifikát pre každé meno (alebo malú skupinu mien), čo v praxi znamená, že každý webový server, ktorý chce využívať HTTPS, by musel mať svoju vlastnú IP adresu.



Obr. 3.3: Princíp fungovania SNI

Riešenie poskytlo SNI, pomocou ktorého je možné pri prístupe cez HTTPS vytvárať virtuálne webové servery - t.j. viac rôznych doménových mien umiestnených na jednej IP adrese. Pomocou SNI môže klient pred výmenou šifrovacích kľúčov najprv oznámiť meno požadovaného webového servera a server tak môže vybrať príslušný šifrovací kľúč požadovaného virtuálneho servera.

3.4.1 Levenshteinova vzdialenosť

Počas mojich experimentov opísaných v kapitole 5 som zistil, že vo väčšine aplikácií sa nachádza SNI podobný názvu aplikácie alebo jej balíčku, čím je možné výrazne zvýšiť úspešnosť ich detekcie z prevádzky na sieti. SNI však treba s názvom porovnať pre zistenie percentuálnej zhody a následnú klasifikáciu aplikácie. Pre potreby tohto porovnávania som zvolil metódu počítania Levenshteinovej vzdialenosti.

Levenshteinova vzdialenosť, taktiež známa ako editačná vzdialenosť [1], je algoritmus z roku 1965, ktorý bol navrhnutý ruským matematikom Vladimírom Iosifovich Levenshteinom pre potreby merania editačnej vzdialenosti v textových reťazcoch. Levenshteinova vzdialenosť dvoch textových reťazcov určuje počet operácií potrebných pre prevod jedného reťazca na druhý. Prípustné operácie sú tri jednoduché editačné operácie, a to pridanie ľubovoľného znaku, odstránenie ľubovoľného znaku alebo zámenu ľubovoľného znaku za iný znak. Najmenší počet týchto operácií nám potom udáva editačnú vzdialenosť medzi nimi.

Pomocou tejto metódy je možné porovnávať dva reťazce. Levenshteinova vzdialenosť nám však neurčuje percentuálnu zhodu medzi dvoma reťazcami, v tomto prípade medzi dvoma SNI, preto som pre potreby počítania zhody medzi SNI použil jednoduchý matematický princíp. Pokiaľ si zoberieme dĺžku väčšieho reťazca, odčítame od nej Levenshteinovu vzdialenosť a následne výslednú hodnotu podelíme opäť dĺžkou väčšieho reťazca, dostaneme ako výsledok percentuálnu zhodu medzi dvoma reťazcami. Keď si dĺžku väčšieho reťazca označíme ako *maxLen* a Levenshteinovu vzdialenosť ako *levenshteinLen*, výsledný vzorec pre vypočítanie zhody potom vyzerá nasledovne:

percentuálna zhoda = (maxLen - levenshteinLen) / maxLen

Ostáva už len určiť hranicu, tzv. threshold, ktorej presiahnutie indikuje zhodu medzi SNI v databáze a sieťovej prevádzke. Bližším skúmaním dát som zistil, že ideálna hranica je 70%, keďže SNI obsahuje okrem názvu serveru aj jeho doménu a iné informácie (výsledky týchto experimentov nájdeme v kapitole 5).

3.5 Skript pre vytváranie databázy

Dáta potrebné na získavanie informácií som nazbieral spôsobom opísaným v kapitole 2. Stručne zhrnuté, použil som Android Studio (sekcia 2.1) pre emuláciu mobilného zariadenia, nástroj Monkey (sekcia 2.2.3) na generovanie simulovaného používania aplikácie a následne nástroj Wireshark (sekcia 2.3) na odchytávanie paketov komunikácie do súboru typu PCAP.

Jednou z častí nástroja, ktorý som implementoval ako súčasť tejto práce, je aj skript pre spracovanie dát z PCAP súborov. Jeho implementácia je uložená v súbore *fingerprinter.py*. Pokiaľ je modul spustený samostatne, ako prvé prebieha spracovanie argumentov. Program prijíma tieto argumenty:

- -h : vypíše návod na používanie skriptu, nepovinný argument
- -a : názov aplikácie, povinný argument (názov je voliteľný užívateľom, podľa neho sa vytvorí príslušný databázový záznam pre aplikáciu)
- -p : cesta ku súboru/zložke so súbormi PCAP, povinný argument

Na poradí týchto argumentov nezáleží, je však potrebné zadať všetky povinné argumenty, v opačnom prípade skript skončí s chybovým hlásením. Konkrétne spustenie programu potom môže vyzerať napríklad takto:

```
$ python3 fingerprinter.py -a GMail -p /home/user/Documents/gmail/pcapfiles
```

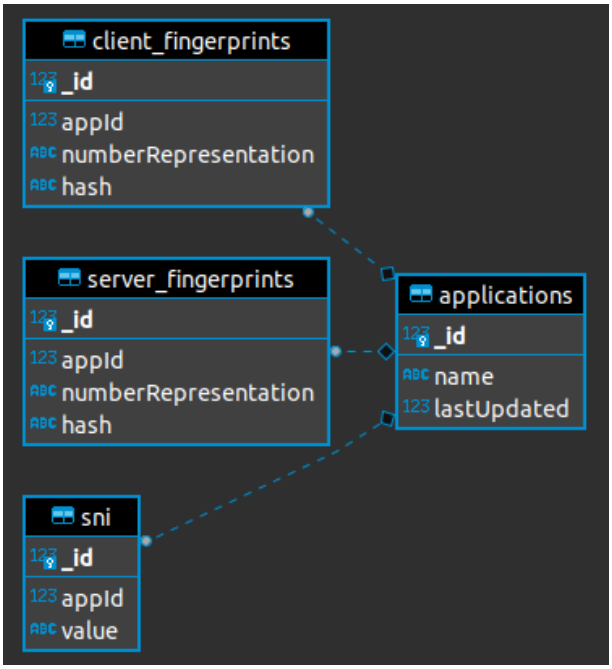
3.5.1 Vytvorenie databázového súboru

Pre potreby mojej práce využívam SQL databázu, konkrétne dialekt SQLite. Tento dialekt som zvolil preto, že v rámci mojej práce nebude nutné vykonávať zložité úkony s databázou, len jednoduhé vkladanie a vyberanie dát a na to je SQLite ideálne.

Vytvorenie databázového súboru má na starosti skript *dbCreator.py*, ktorý po spustení skontroluje prítomnosť databázového súboru v zložke */capturer/database*. Pokiaľ sa tu databázový súbor nenachádza, skript ho vytvorí. Ak bol už databázový súbor vytvorený, skript pokračuje a skúša pridať všetky definované tabuľky. Po skončení skriptu je databáza plne pripravená na používanie.

Schému databázy môžeme vidieť na obrázku 3.4. Hlavná tabuľka *applications* obsahuje všetky známe aplikácie, pre ktoré už bol vytvorený záznam. Tabuľky *client_fingerprints* a *server_fingerprints* obsahujú všetky digitálne odtlačky, jednak v textovej podobe a jednak v podobe zahashovanej, ako bolo spomenuté v kapitolách 3.2 a 3.3. V tabuľke *sni* sa nachádzajú všetky SNI, ktoré splnili podmienky pridania do databázy (kapitola 3.4.1). Každá tabuľka obsahuje odkaz na tabuľku aplikácií, aby bolo možné následne daný odtlačok priradiť konkrétnej aplikácii.

Digitálne odtlačky a SNI však nechceme do databázy ukladať duplikovane, stačí nám vždy jeden jedinečný záznam. Hodnoty odtlačkov a SNI sú preto definované pomocou kľúčového slova `UNIQUE`, čo nám zaistí, že sa v tabuľkách budú jednotlivé záznamy vyskytovať



Obr. 3.4: Databázová schéma v podobe ER diagramu

len raz. Aby mal používateľ informáciu o tom aké staré záznamy sa v tabuľke pre jednotlivé aplikácie nachádzajú, obsahuje tabuľka aplikácií taktiež informáciu o posledom dátume aktualizácie. Táto hodnota je zapísaná ako Epoch Unix timestamp[3], ktorá nám udáva počet sekúnd od začiatku počítania unixového času, t.j. od 1.1.1970 00:00:00.

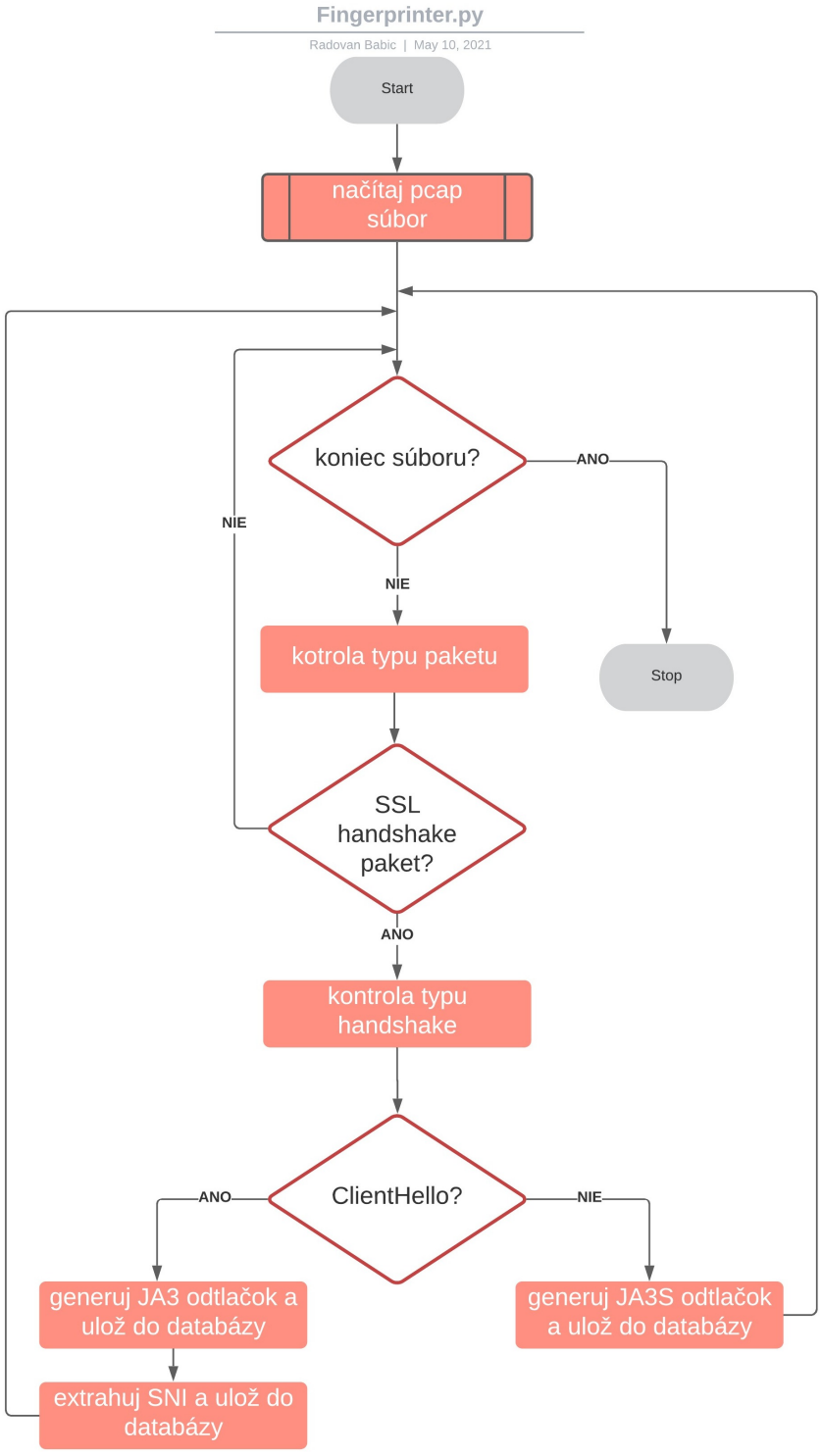
3.5.2 Extrakcia dát a ich následné ukladanie do databázy

Pokiaľ boli vstupné argumenty zadané správne, skript začína svoju prácu vytvorením databázy. Akonáhle je databáza vytvorená a pripravená pre používanie, skript pokračuje otvorením PCAP súboru alebo súborov a hľadá v nich pakety, ktoré obsahujú SSL hlavičku. Ostatné pakety totiž nepatria do šifrovanej komunikácie, preto ich do analýzy nezapájam. Z komunikácie sú taktiež odstránené GREASE hodnoty, ako aj komunikácia s reklamnými servermi. Otvorenie PCAP súboru má na starosti modul *PacketHandler* opísaný v sekcii 2.4.

Akonáhle program narazí na SSL paket, je potrebné ho klasifikovať. Nie každý paket je potrebný, hľadáme len pakety, ktoré nadväzujú šifrovanú komunikáciu. Podľa toho, či je správa typu ClientHello alebo ServerHello, sú z nej extrahované záujmové dáta, z ktorých je následne podľa postupu opísaného v kapitole 3 vytvorený digitálny odtlačok aplikácie.

Počas tohto procesu je každý nový odtlačok pridávaný do databázy, za čo zodpovedajú funkcie definované v skripte *dbHandler.py*. Tento skript priamo komunikuje s databázou, pridáva do nej dáta a kontroluje, či neboli porušené integritné obmedzenia databázy. To by za normálnych okolností spôsobilo pád aplikácie s chybovým hlásením. Takto je zaručené, že všetky dáta, vkladané do databázy, sú správne a jedinečné.

[3]https://www.unixtimestamp.com/ [25.4.2021]



Obr. 3.5: Vývojový diagram skriptu fingerprinter.py

3.6 Zhrnutie

Táto kapitola bola venovaná princípu fungovania metód na vytváranie digitálnych odtlačkov softvéru. Čitateľ sa oboznámil s princípom nadviazania šifrovanej komunikácie, ktoré je kľúčové pre vytvorenie odtlačku. Dve najdôležitejšie súčasti takejto komunikácie sú správy ServerHello a ClientHello. Na základe metód JA3 a JA3S, navrhnutými vývojármi zo Salesforce a s malými úpravami pre môj konkrétny problém, som za pomoci emulovaných dát tieto správy odchytil a extrahoval z nich záujmové dáta potrebné pre identifikáciu konkrétnej aplikácie. Z takto extrahovaných dát som podľa postupu opísaného v tejto kapitole vytvoril MD5 hash, ktorý budem ďalej používať ako jednoznačný identifikátor aplikácie - jej digitálny odtlačok. Ďalej bol čitateľ oboznámený s využívaním SNI, ako aj s dôvodom, prečo túto položku taktiež zapájam do procesu klasifikácie a následnej detekcie aplikácií. V kapitole bol opísaný spôsob porovnávania SNI pomocou počítania Levenshteinovej vzdialenosti, spôsob vytvárania a používania databázy typu SQLite, ako aj následné ukladanie a triedenie dát.

Kapitola 4

Detekcia aplikácií

Táto kapitola je venovaná detekcii aplikácií. Opísaný je v nej princíp tejto detekcie, ako aj popis mnou implementovaného nástroja. V kapitole sú opísané datasety, ktoré som pre potreby vývoja a ladenia nástroja využíval, ako aj výsledky testovania na známych a neznámych dátach. Čitateľ bude oboznámený so spôsobom zisťovania zhody, ako aj s problémami, ktoré pri detekcií aplikácií môžu nastať.

4.1 Princíp detekcie aplikácií

Detekcia aplikácií prebieha pomocou nástroja *AppDetector*. Pri spustení nástroj načítava vstupné argumenty. Možné hodnoty sú tieto dve:

- -h : vypíše nápovedu pre obsluhu nástroja, nepovinný argument.
- -p : cesta k súboru / zložke so súbormi pcap, povinný argument.

Po načítaní vstupných argumentov začína nástroj otvorením pcap súboru. Jednotlivé pakety sú následne prechádzané a pokiaľ nástroj narazil na ClientHello alebo ServerHello správu, extrahuje z tejto komunikácie záujmové dáta potrebné ku detekcii. Z týchto dát si vytiahne SNI a pomocou nástroja opísaného v kapitole 3.5 vygeneruje digitálnu signatúru.

Dáta získané z paketu komunikácie sú porovnávané s dátami uloženými v databáze. Pre označenie detekcie za úspešnú musia byť splnené tieto tri pravidlá:

- Prítomnosť známeho JA3 odtlačku.
- Prítomnosť známeho JA3S odtlačku.
- Prítomnosť známeho SNI.

Pokiaľ nie je splnená aspoň jedna z vyššie uvedených podmienok, nástroj aplikáciu neoznačí ako rozpoznanú a pokračuje v detekcii ďalej. Dôvodom pre tento spôsob rozpoznávania aplikácií je fakt opísaný v kapitole 5.2. Odtlačky JA3 a JA3S sa totižto v pomerne vysokej miere prelínajú medzi rôznymi aplikáciami, preto je potrebné pri detekcii zahrnúť aj kontrolu rozšírenia SNI.

4.2 Testovanie

Testovanie detekcie prebiehalo najskôr na mnou vytvorenom datasete a po vyhodnotení týchto výsledkov som detekciu testoval na nevidených dátach. Pre interpretáciu výsledkov použijem metódu vytvorenia matice zámen (confusion matrix) a spočítanie presnosti (accuracy), precíznosti (precision) a pravdepodobnosti detekcie (recall).

Pre tento výpočet potrebujeme zadefinovať 4 hodnoty:

- True positive (TP): aplikácia sa nachádzala v komunikácii a bola detegovaná.
- True negative (TN): aplikácia sa nenachádzala v komunikácii a nebola detegovaná.
- False positive (FP): aplikácia sa nenachádzala v komunikácii a bola detegovaná.
- False negative (FN): aplikácia sa nachádzala v komunikácií a nebola detegovaná.

Presnosť nám udáva ako často bola detekcia úspešná. Je to podiel správne detegovaných aplikácií k počtu detekcií.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Precíznosť určuje mieru správnosti detekcie aplikácie. Vyjadruje počet prípadov, kedy detegovaná aplikácia bola detegovaná správne. Je to pomer správne detegovaných aplikácií ku všetkým aplikáciám, ktoré algoritmus označil ako detegované.

$$Precision = \frac{TP}{TP + FP}$$

Pravdepodobnosť detekcie nám určuje podiel medzi počtom úspešných detekcií a všetkými výskytmi aplikácie v poskytnutých dátach. Udáva spoľahlivosť detekcie.

$$Recall = \frac{TP}{TP + FN}$$

Confusion matrix je špecifický typ tabuľky, ktorá slúži na vizualizáciu výkonnosti algoritmu. Stĺpce tejto tabuľky predstavujú vstupné dáta pre detekciu aplikácie a riadky vyjadrujú reálny počet detekcií. Čísla v zátvorkách označujú predpokladaný počet detekcií. Pre pohodlnejšiu interpretáciu dát použijem v tabuľke miesto názvu aplikácie jej iniciály.

Aplikácia	Názov v tabuľke
AccuWeather	AW
Seznam	S
GMail	G
MujVlak	MV
Viber	V
CestovnePoriadky	CP
Slack	SL
Discord	DC

Tabuľka 4.1: Iniciály aplikácií použité v matici zámen

4.2.1 Testovanie nad známymi dátami

Testovanie nad známymi dátami slúži ako ukazovateľ úspešnosti detekcie na dátach, z ktorých bola vytvorená databáza odtlačkov a SNI. Využíval som ho aj počas vývoja môjho nástroja pre prípadné doladenie nezrovnalostí.

	AW	S	G	MV	V	CP	SL	DC
AW	180 (180)	-	-	-	-	-	-	-
S	-	132 (132)	-	-	-	-	-	-
G	-	-	5 (5)	-	-	-	-	-
MV	-	-	-	12 (12)	-	-	-	-
V	-	-	-	-	1 (1)	-	-	-
CP	-	-	-	-	-	10 (10)	-	-
SL	-	-	-	-	-	-	48 (48)	-
DC	-	-	-	-	-	-	-	21 (21)

Tabuľka 4.2: Matica zámen

Z matice zámen je zrejmé, že počet prípadov kedy by aplikácia nebola úspešne detegovaná je nulový. Výsledky testovania nad známymi dátami ukazujú 100% úspešnosť tejto metódy detekcie aplikácií. Metriky presnosti nájdeme v tabuľke 4.3. Keďže prebehla detekcia vo všetkých prípadoch správne, všetky metriky nadobúdajú hodnotu 100%.

Metrika	Hodnota
Accuracy	100%
Precision	100%
Recall	100%

Tabuľka 4.3: Metriky presnosti detekcie aplikácií pre známe dáta

4.2.2 Testovanie na nevidených dátach

Táto časť testovania mala ukázať úspešnosť detekcie z dát, ktoré nie sú súčasťou mnou vytvoreného datasetu. Skriptu som ako argument predal pcap súbor, ktorý obsahoval pseudonáhodnú aktivitu všetkých aplikácií využívaných v tejto práci.

Testovanie na nevidených dátach prinieslo pozoruhodné výsledky. Niektoré aplikácie sa mi podarilo detegovať vo všetkých prípadoch, iné aplikácie len v niektorých prípadoch a boli aj také, ktoré sa detegovať nepodarilo. Moje datasety obsahovali všetky signatúry pre aplikácie *AccuWeather*, *Slack* a *Discord*, obsahovali časť signatúr pre *MujVlak* a *CestovnePoriadky*, no neobsahovali žiadny odtlačok pre aplikácie *Viber* a *Seznam*. Výsledky testovania sú znázornené v matici zámen 4.4.

	AW	S	G	MV	V	CP	SL	DC
AW	194 (194)	-	-	-	-	-	-	-
S	-	0 (92)	-	-	-	-	-	-
G	-	-	0 (1)	-	-	-	-	-
MV	-	-	-	2 (3)	-	-	-	-
V	-	-	-	-	0 (35)	-	-	-
CP	-	-	-	-	-	2 (6)	-	-
SL	-	-	-	-	-	-	2 (2)	-
DC	-	-	-	-	-	-	-	8 (8)

Tabuľka 4.4: Matica zámen pre nevidené dáta

Databáza odtlačkov bola vytvorená v druhej polovici roku 2019. Dáta potrebné pre tvorbu odtlačkov sa ale časom stávajú nekonzistentné, čo spôsobilo drastickú zmenu úspešnosti detekcie. Pre bližší rozbor použijem aplikáciu *Seznam*.

TLS pakety	Z JA3	NZ JA3	Z JA3S	NZ JA3S	Z SNI	NZ SNI
18515	19	8	0	27	13	14

Tabuľka 4.5: Metriky detekcie aplikácie Seznam na nevidených dátach

Testovanie prebiehalo na emulovanom zariadení Google Pixel 3a s operačným systémom Android 11.0. Tabuľka 4.5 obsahuje podrobné informácie prečo detekcia neprebehla úspešne. Písmeno Z v tabuľke označuje známe dáta, ktoré sa nachádzali v databáze odtlačkov, písmená NZ označujú neznáme dáta, t.j. dáta, ktoré sa v databáze odtlačkov nenachádzali. Stĺpec TLS pakety vyjadruje počet skenovaných paketov nevidených dát.

Z rozboru výsledkov je zrejmé, že správanie aplikácie sa značne zmenilo. V komunikácii sa nachádzali neznáme JA3 odtlačky, ako aj neznáme SNI. Úspešnosť detekcie však klesla na nulu kvôli správaniu servera. Z JA3S odtlačkov nájdených v nevidených dátach sa ani jeden nenachádzal v databáze. Ako som spomínal v kapitole 4.1, tento fakt spôsobil, že aplikácia *Seznam* nemohla byť úspešne detegovaná.

Metrika	Hodnota
TP	208
TN	385
FP	0
FN	133

Tabuľka 4.6: Hodnoty TP, TN, FP a FN pre nevidené dáta

V tabuľke 4.6 vidíme hodnoty potrebné pre výpočet metrík *accuracy*, *precision* a *recall*. Hodnota TP udáva, že sa vyskytlo 208 prípadov, kedy bola aplikácia detegovaná správne. TN vyjadruje počet paketov, ktoré nepatrili žiadnej aplikácii a boli správne vyhodnotené ako neznáme. Hodnota FP sa rovná nule, lebo nenastal ani jeden prípad, kedy by bola detegovaná aplikácia, ktorá nebola súčasťou dát. Nakoniec nám hodnota FN vyjadruje počet aplikácií, ktoré neboli detegované, no boli v komunikácii prítomné.

Metrika	Hodnota
Accuracy	81.7%
Precision	100%
Recall	31%

Tabuľka 4.7: Metriky presnosti detekcie aplikácií pre nevidené dáta

Z tabuľky 4.7 je zrejmé, že detekcia na nevidených dátach bola pomerne presná, precízna, ale veľké zmeny fungovania komunikácie zo strany serveru posunuli pravdepodobnosť detekcie na nízku hodnotu. Je preto potrebné podotknúť, že metóda opísaná v tejto práci, ako aj mnou implementovaný nástroj, sú správne a funkčné, ale je potrebné databázu odtlačkov pravidelne aktualizovať.

Pre podloženie tohto tvrdenia som databázu odtlačkov aktualizoval, vygeneroval novú neznámu prevádzku a testovanie vykonal ešte raz. Výsledky tohto testovania nájdeme v tabuľkách 4.8, 4.9 a 4.10.

	AW	S	G	MV	V	CP	SL	DC
AW	62 (62)	-	-	-	-	-	-	-
S	-	48 (55)	-	-	-	-	-	-
G	-	-	1 (1)	-	-	-	-	-
MV	-	-	-	2 (3)	-	-	-	-
V	-	-	-	-	4 (4)	-	-	-
CP	-	-	-	-	-	4 (8)	-	-
SL	-	-	-	-	-	-	2 (6)	-
DC	-	-	-	-	-	-	-	3 (3)

Tabuľka 4.8: Matica zámen po aktualizácii databázy

Metrika	Hodnota
TP	126
TN	178
FP	0
FN	16

Tabuľka 4.9: Hodnoty TP, TN, FP a FN po aktualizácii databázy

Metrika	Hodnota
Accuracy	94.4%
Precision	100%
Recall	88.7%

Tabuľka 4.10: Metriky presnosti detekcie aplikácií po aktualizácii databázy

Z dát môžeme vidieť, že detekcia po vykonaní aktualizácie databázy bola výrazne úspešnejšia. Hlavný rozdiel spočíval vo vygenerovaní chýbajúcich JA3S odtlačkov.

4.3 Zhrnutie

V tejto kapitole bol opísaný princíp fungovania detekcie aplikácií pomocou mnou implementovanej metódy. Vysvetlený tu bol postup, ako aj pravidlá pre označenie aplikácie ako úspešne detegovanej. Podkapitola 4.2 obsahovala informácie o testovaní. Testovanie bolo prevádzané na dátach známych a neznámych. Testovanie nad známymi dátami malo za úlohu overiť správnosť metódy a na neznámych dátach poskytlo informácie o fungovaní detekcie pri dátach z reálnej prevádzky na sieti. Popísaný bol postup testovania, metriky využité na výpočet presnosti, precíznosti a pravdepodobnosti detekcie aj s ich výsledkami. Výstupom tejto kapitoly je tvrdenie, že metóda a nástroj implementovaný v tejto práci sú korektné, no úspešnosť detekcie sa odvíja od aktuálnosti databázy odtlačkov. V prípade aktualizácie dát metóda vykazovala výrazne lepšie výsledky.

Kapitola 5

Experimenty

Podstatnú časť mojej práce tvorila analýza a následné experimentovanie s dátami, ktoré som pomocou nástroja implementovaného v rámci tejto práce nazbieral. Tento krok bol nutný k správnemu pochopeniu fungovania šifrovanej komunikácie medzi klientom a serverom, z ktorej som bol následne schopný extrahovať dáta potrebné pre klasifikáciu aplikácií. Vďaka týmto experimentom som bol schopný určiť a definovať podstatné vlastnosti nazbieraných dát, na čo som v tejto práci niekoľkokrát odkazoval. V tejto kapitole bude bližšie opísaný spôsob, akým som experimentoval s nazbieranými dátami, čo bolo podstatou jednotlivých experimentov, ako aj sumarizácia výsledkov a ich dôležitosť pre moju prácu. Presný spôsob používania nástrojov potrebných pre tieto experimenty už nebude spomínaný, keďže boli tieto informácie podrobne preberané v predchádzajúcich kapitolách.

5.1 Experimenty so súbormi pcap

Prvým a najpodstatnejším krokom môjho výskumu bola analýza dát nazbieraných skenovaním internetovej prevádzky. Cieľom týchto experimentov bolo zistiť početnosť paketov šifrovanej komunikácie (pakety obsahujúce SSL hlavičku), počet serverových a klientských paketov a následný počet jedinečných odtlačkov a SNI spĺňujúcich moje kritériá. Na základe týchto informácií som následne mohol určiť, aký spôsob bude podľa mojich zistení najlepší pre klasifikáciu a následnú detekciu mobilných aplikácií.

```
        --- SUMMARY ---
Number of PCAP files scanned: 1
Number of ssl packets: 685
Number of ClientHello messages: 44
New ClientHello fingerprints: 0
Number of ServerHello messages: 44
New ServerHello fingerprints: 0
```

Obr. 5.1: Príklad výstupu jedného behu skriptu

Experimenty som prevádzal opakovaným spúšťaním môjho nástroja, do ktorého som pridal aj vypísanie metrík každého chodu programu. Inými slovami, skript mi po každom spustení podal informáciu o tom, koľko súborov otvoril, koľko paketov prešiel, následnú

početnosť ServerHello a ClientHello správ, ako aj počet novonájdených SNI či odtlačkov. Príklad výpisu môjho nástroja môžeme vidieť v obrázku 5.1 a výstupy týchto metrík sú dostupné v prílohe A.2.

Pri skúmaní týchto výstupov sa objavila otázka o počte potrebných dát k nazbieraniu dostatočného množstva všetkých možných kombinácií odtlačkov a všetkých variánt SNI. Pre zistenie tejto informácie som postupne skriptu predával pcap súbory a sledoval počet novo nájdených dát. Ako príklad pre vysvetlenie tejto metódy použijem aplikáciu *AccuWeather*[1] vo verzií *7.5.1-9-google*, spustenú na emulovanom zariadení *Google Pixel 3a* s operačným systémom *Android 11.0*.

Beh	SSL pakety	Unikátne JA3	Unikátne JA3S	Unikátne SNI
1.	1503	1	12	2
2.	1824	2	5	2
3.	2688	0	4	0
4.	685	0	3	0
5.	4127	0	0	0

Tabuľka 5.1: Postupný zber dát pre aplikáciu AccuWeather

V tabuľke 5.1 môžeme vidieť postupné analyzovanie vytvorených pcap súborov. Cieľom tohto experimentu bolo zistiť, koľko dát je minimálne potrebných ku získaniu všetkých dostupných signatúr aplikácie. Je potrebné dodať, že toto skúmanie bolo vykonávané v rokoch 2020 a 2021, čo znamená, že je odporúčané databázu odtlačkov a SNI pravidelne aktualizovať, pretože sa technológie stále vyvíjajú a počet rôznych signatúr sa môže zvyšovať. Experimenty mi však dodali dostatok informácií o momentálnom stave chovaní aplikácií.

V tabuľke vidíme postupné spúšťanie skriptu, kedy ako vstup bol použitý vždy novo vytvorený pcap súbor aplikácie. Ako môžeme vidieť, prvé štyri spustenia našli v paketoch nové signatúry, zatiaľ čo piate spustenie už nehlási nález neznámeho odtlačku či SNI. Taktiež vidíme, že postupným spúšťaním sa počet nových dát výrazne znižoval až klesol na nulu. Tieto metriky boli medzi aplikáciami takmer totožné, niektoré aplikácie požadovali dokonca menej spustení na to, aby počet nových signatúr klesol na nulu.

Výsledok teda hovorí, že priemerný počet spustení skriptu na extrakciu záujmových dát sú 3 spustenia s objemom približne 2675 paketov. Pri predložení takéhoto množstva dát môžeme predpokladať, že sme databázu naplnili všetkými momentálne dostupnými signatúrami danej aplikácie. Toto tvrdenie som vyvodil na základe metrík spúšťania skriptu pre všetky mnou využívané aplikácie, ktoré sú dostupné v prílohe A.3.

5.2 Experimenty s digitálnymi odtlačkami

Po preskúmaní dát a získaní dostatočného množstva informácií, som začal experimentovať s digitálnymi odtlačkami. Cieľom mojich experimentov bolo preskúmať možnosti vytvárania týchto odtlačkov pomocou metód JA3 a JA3S. Bližším skúmaním som však zistil, že pôvodné metódy opisujú extrakciu záujmových dát, z ktorých sa v mnou nazbieraných dátach nevyskytovalo rozšírenie *elliptic_curves*. Pri experimentoch som urobil zistenie, že položky šifrovanej komunikácie, ktoré sa v dátach nenachádzali, je možné nahradiť inými položkami k dosiahnutiu podobného až totožného výsledku. Táto skutočnosť bola opísaná v kapitolách 3.2 a 3.3.

[1] https://www.accuweather.com/

Tieto experimenty taktiež prebiehali s využitím môjho nástroja, pomocou ktorého som mohol dopodrobna skúmať pakety šifrovanej komunikácie. Na základe týchto pokusov som bol následne schopný vytvoriť databázu digitálnych signatúr mobilných aplikácií, ktoré však podľa mojich zistení, ako sa dočítame ďalej, nie sú dostatočne smerodajné pre jednoznačnú klasifikáciu a detekciu aplikácií. Tieto odtlačky medzi jednotlivými aplikáciami kolidovali v miere dosť vysokej na to, aby som usúdil, že pre jednoznačné určenie pôvodu paketov je do procesu nutné zapojiť ďalší faktor. Ako silný kandidát sa v tej dobe javilo SNI, preto som sa následne zameral na preskúmanie možností využitia tohto rozšírenia.

5.2.1 JA3 experimenty

Experimenty s JA3 odtlačkami priniesli znepokojivé výsledky. Ako bolo spomínané v kapitole 3.1, pri nadväzovaní spojenia so serverom klient definuje spôsoby, ktorými je schopný komunikovať a server následne jeden z týchto spôsobov vyberie. Ako sa však v mojich experimentoch ukázalo, spôsoby nadväzovania šifrovanej komunikácie sú v aplikáciách naprogramované takmer totožným spôsobom. Aplikácie používajú bežne dostupné šifrovacie sady a algoritmy, čo spôsobuje, že tieto zoznamy sa medzi aplikáciami takmer vôbec nelíšia a ich digitálne odtlačky tvorené metódou JA3 medzi sebou vo veľkej miere kolidujú.

Beh	Aplikácia	Pcap súbory	ClientHello správy	Unikátne JA3
1.	AccuWeather	5	518	20
2.	Seznam	3	431	0
3.	GMail	4	60	3
4.	MujVlak	3	16	0
5.	Viber	5	250	0
6.	CestovnePoriadky	4	71	0

Tabuľka 5.2: Výsledky tvorenia databázy JA3 odtlačkov

Ako príklad som použil vzorku šiestich aplikácií. V tabuľke 5.2 môžeme vidieť postupné pridávanie odtlačkov do databázy. Pred prvým spustením bola databáza prázdna, preto môžeme vidieť, že predloženie paketov aplikácie *AccuWeather* vygenerovalo 20 jedinečných digitálnych signatúr. Tieto boli vložené do databázy a pokračovalo sa s ďalšou aplikáciou. Z pozorovania je však zrejmé, že predloženie rovnakého objemu dát inej aplikácie, v tomto prípade *Seznam*, už nevygenerovalo žiadny nový odtlačok. Aplikácia sa pri nadväzovaní TLS spojenia správala totožne, preto nebolo možné vytvoriť jedinečnú predlohu jej komunikácie. To však značne znižuje možnosti detekcie, nakoľko je v tomto prípade nemožné tieto dve aplikácie rozlíšiť.

Výsledky spustenia nástroja s dátami obsahujúce komunikáciu aplikácie *GMail* vyzerali nádejnejšie. Ukázalo sa, že táto aplikácia vygenerovala tri jedinečné signatúry, čo by znamenalo možný úspech pri jej detekcii. Následné prechádzanie dát aplikácií *MujVlak*, *Viber* a *CestovnePoriadky* však znovu prinieslo rovnaký výsledok, keďže počet jedinečných odtlačkov aplikácie bol vo všetkých troch prípadoch nulový.

Opakovanými pokusmi vygenerovať jedinečné signatúry som zistil, že korelácia medzi odtlačkami je natoľko vysoká, až je vo veľmi veľkej miere nemožné používať tieto odtlačky pri detekcii. Vďaka znepokojivým výhľadom metódy JA3 som mnou nazbierané dáta skúsil zanalyzovať ešte raz a všimol som si zaujímavé rozšírenie TLS komunikácie, ktoré by mohlo byť použité na zvýšenie presnosti detekcie pomocou JA3 - *supported_groups*.

Podrobné preskúmanie tohto rozšírenia za pomoci RFC 8446 [2] prinieslo zaujímavé zistenie. V kapitole 3.2 som spomínal, že jedno z rozšírení, ktoré bolo použité v originálnej metóde, som v mojich dátach nenašiel. Dôvodom bola zmena názvu zo spomínaného *elliptic_curves* na *supported_groups*. Toto rozšírenie som pridal do procesu tvorby digitálnej signatúry aplikácie a vykonal znovu experiment.

Beh	Aplikácia	Pcap súbory	ClientHello správy	Unikátne JA3
1.	AccuWeather	5	518	3
2.	Seznam	3	431	52
3.	GMail	4	60	0
4.	MujVlak	3	16	1
5.	Viber	5	250	0
6.	CestovnePoriadky	4	71	5

Tabuľka 5.3: Výsledky tvorenia databázy JA3 odtlačkov po pridaní supported_groups

V tabuľke 5.3 môžeme vidieť výsledky vytvárania odtlačkov po pridaní TLS rozšírenia *supported_groups*. Ukázalo sa, že toto rozšírenie výrazne zlepšuje presnosť detekcie pomocou JA3, pretože sa v komunikácií nachádzalo viac signatúr jedinečných pre aplikáciu (v prípade aplikácie Seznam výrazne viac). Niektoré aplikácie však stále nevygenerovali svoj jedinečný odtlačok. Z experimentov teda vyplýva, že detekcia len pomocou metódy JA3 je možná, ale s nižšou presnosťou. Z tohto dôvodu v mojej práci detekciu aplikácií obohacujem aj o kontrolu SNI, opísané v kapitole 4.

5.2.2 JA3S experimenty

Význam experimentov s JA3S odtlačkami bol rovnaký ako u JA3. Cieľom bolo zistiť početnosť kolízie odtlačkov medzi aplikáciami, a tak určiť dôležitosť prítomnosti tejto metódy v procese klasifikácie a detekcie. Ako príklad znovu použijem vzorku šiestich aplikácií. Pred použitím nástroja bola databáza prázdna.

Beh	Aplikácia	Pcap súbory	ServerHello správy	Unikátne JA3S
1.	AccuWeather	5	519	24
2.	Seznam	3	428	21
3.	GMail	4	60	0
4.	MujVlak	3	16	1
5.	Viber	5	248	0
6.	CestovnePoriadky	4	71	7

Tabuľka 5.4: Výsledky tvorenia databázy JA3S odtlačkov

Experimenty s JA3S priniesli pozitívne správy. Z výsledkov vyplynulo, že odtlačok zo správy servera nám poskytne relevantnú informáciu o pôvode komunikácie. Podľa údajov v tabuľke 5.4 štyri zo šiestich aplikácií vygenerovali jedinečnú signatúru servera, avšak v prípade dvoch aplikácií nebolo možné nájsť unikátny odtlačok. Z toho vyplýva, že prieniky medzi aplikáciami nie sú až tak časté, avšak nie sú ani nulové. JA3S metóda preto taktiež nemôže byť samostatne použitá pre detekciu aplikácií, no výrazne ku tejto detekcii prispieva. Túto metódu som preto zaradil do procesu klasifikácie a detekcie aplikácií.

5.3 Experimenty s SNI

Pri mojich experimentoch s metódami JA3 a JA3S som zistil, že pre dosiahnutie vyššej presnosti zisťovania prítomnosti aplikácií na základe internetovej komunikácie musím do procesu detekcie zahrnúť okrem odtlačkov aj iný faktor. Ako vhodný kandidát sa javilo rozšírenie SNI, opísané v kapitole 3.4. Definícia servera, ku ktorému sa klient pokúša pripojiť, by mohla byť cenná informácia určujúca aplikáciu, z ktorej požiadavka pochádza.

Cieľom experimentov bolo zistiť spôsob priradenia SNI ku aplikácii. Ako bolo v práci spomínané, toto rozšírenie definuje doménové meno servera, na ktorý sa chce klient pripojiť. Teoreticky je teda možné, že bude SNI obsahovať názov aplikácie alebo aspoň jeho časť. Pre potvrdenie či vyvrátenie tohto tvrdenia som pomocou môjho nástroja analyzoval SNI v mnou získaných dátach.

5.3.1 Získavanie SNI pomocou názvu aplikácie

Prvotná myšlienka bola extrakcia SNI a následné zistenie zhody pomocou Levenshteinovej vzdialenosti (kapitola 3.4.1). V prípade presiahnutia hranice 70% by bolo SNI uložené do databázy. Táto metóda ale nefungovala, pretože sa toto rozšírenie skladá z niekoľkých reťazcov oddelených zväčša bodkami. Napríklad jedno z SNI aplikácie *AccuWeather* vyzerá nasledovne:

api.accuweather.com

Na základe tohto zistenia som pochopil, že jednoduché porovnanie SNI nebude možné. Definíciu doménového mena preto najskôr rozdelím na podreťazce, pričom ako oddeľovač použijem bodku a následne použijem Levenshteinov algoritmus na zistenie zhody. Pokiaľ je zhoda vyššia ako 70%, považujem SNI za validné a uložím ho do databázy.

Aplikácia	Pcap súbory	SNI spĺňújce podmienky
AccuWeather	5	4
Seznam	3	17
GMail	4	1
MujVlak	3	0
Viber	5	1
CestovnePoriadky	4	0

Tabuľka 5.5: Výsledky generovania SNI pomocou názvu aplikácie

Tabuľka 5.5 nám ukazuje počet SNI, ktoré za použitia podmienok spomenutých vyššie skript označil za validné. Vo väčšine prípadov sa táto metóda ukázala ako úspešná, avšak existujú aj také aplikácie, ktorých server neobsahuje v jeho doménovom mene priamo variáciu názvu danej aplikácie. Experimenty teda ukázali, že metóda získavania SNI pomocou názvu aplikácie je účinná, no nemôžeme zaručene tvrdiť, že dokáže vždy správne priradiť SNI k jeho aplikácii.

5.3.2 Získavanie SNI pomocou kľúčových slov

Pri hlbšej analýze som spozoroval istú nepriamu spojitosť medzi SNI a aplikáciou. Doménové mená serverov nie vždy obsahujú názov aplikácie, ale stále sú pre danú aplikáciu

špecifické, nakoľko komunikácia vždy prebieha len s určitou množinou serverov, ktoré patria vydavateľovi a slúžia práve pre potreby tejto appky. Napríklad aplikácia *MujVlak* sa pri získavaní dát odjazdu vlakov pripájala vždy na tento server:

ipws2.timetable.cz

Ukázalo sa, že každej aplikácií je možné priradiť množinu serverov, s ktorými bude zaručene komunikovať. Proces tvorby databázy SNI som preto obohatil o možnosť definovať zoznam kľúčových slov. Tieto by mali figurovať rovnako ako názov, kedy by sa rozdelené SNI porovnávalo aj so zoznamom týchto slov. Pokiaľ bola nájdená zhoda, SNI bolo označené za validné a uložené do databázy.

Nevýhoda tejto metódy spočíva v nemožnosti automatizácie procesu. Tento postup vyžaduje priamy zásah užívateľa, ktorý definuje zoznam kľúčových slov. Pri definícii tohto zoznamu je potrebné poznať názov servera daného vydavateľa alebo túto informáciu dohľadať, napríklad ako v mojom prípade analýzou paketov komunikácie danej aplikácie. Na druhej strane je veľkou výhodou fakt, že pri takto vytvorenej databáze môžeme s takmer určitou presnosťou klasifikovať a následne detegovať aplikácie v sieti.

Ďalším problémom je odfiltrovanie nežiadúcej komunikácie a správne zvolenie množiny kľúčových slov. Tento zoznam je potrebné vytvoriť experimentálne, kedy užívateľ prechádza vytvorený dataset a pomocou analýzy SNI prítomných v týchto dátach vyberie správne doménové mená a zvolí hodnoty, ktoré sú im najviac podobné a majú častý výskyt. Skúmanie SNI však nie je predmetom mojej práce.

5.4 Zhrnutie

V tejto kapitole boli opísané experimenty s dátami, ktoré boli nevyhnutné k pochopeniu fungovania internetovej prevádzky a následnej efektívnej extrakcii záujmových dát. Čitateľ bol oboznámený so spôsobom určenia minimálneho počtu dát potrebného pre dostatočné naplnenie databázy potrebnými informáciami. Ďalej boli v kapitole opísané spôsoby zisťovania korelácie JA3 a JA3S, pomocou ktorých sa z komunikácie v sieti vytvárajú digitálne signatúry aplikácií. Kapitola obsahuje informácie o spôsobe získavania SNI, jednak podľa mena aplikácie a jednak použitím zoznamu kľúčových slov. V kapitole bola opísaná metóda porovnávania SNI, ako aj informácia o tom, akým spôsobom boli určené kritériá pre výber validného SNI. Výsledky týchto experimentov ukazujú, že pri správnej selekcii záujmových dát je možné metódy JA3 a JA3S používať pri detekcií aplikácií, avšak samotné tieto metódy neposkytujú dostatočne jednoznačné informácie, a tak je pre potreby úspešnej detekcie potrebné zahrnúť aj metódu kontroly SNI.

Kapitola 6

Záver

Cieľom tejto bakalárskej práce bolo navrhnúť a implementovať nástroje umožňujúce tvorbu databázy odtlačkov mobilných aplikácií s využitím metód JA3 a JA3S a ich následnú detekciu z internetovej prevádzky. Výsledné riešenie, nástroj na inštaláciu a emuláciu aplikácií, extrakciu záujmových dát, tvorbu datasetov a následnú detekciu aplikácií, je plne funkčný a spĺňa všetky body formálneho zadania práce.

V prvej časti práce som skúmal možnosti pre automatizovanú inštaláciu a emuláciu aplikácií a následný zber dát zo sieťovej prevádzky. Vďaka tejto časti práce som sa naučil využívať spomínané nástroje pomocou skriptu alebo príkazového riadku a následne som navrhol a implementoval nástroj, ktorý dokáže automatizovane vytvárať datasety potrebné pre klasifikáciu aplikácií.

Následne som preštudoval metódy vytvárania digitálnych odtlačkov zo šifrovanej sieťovej komunikácie na základe metód JA3 a JA3S. Objasnil som spôsob fungovania komunikácie pomocou sieťového protokolu TLS, ako aj záujmové dáta tejto komunikácie, ktoré sú potrebné pre vytvorenie digitálnej signatúry aplikácie. Ďalej som opísal dôvod, prečo do procesu klasifikácie a detekcie aplikácií zapájam aj rozšírenie SNI, spôsob výberu správneho SNI a navrhol a implementoval som nástroj na automatizované vytváranie databázy odtlačkov.

Pomocou mnou implementovaného nástroja som vytvoril dataset signatúr. Opísal som postup experimentov nad týmito dátami, ako aj ich výsledky a dôležitosť pre moju prácu. Navrhol a implementoval som nástroj pre detekciu aplikácií pomocou takto vytvorených datasetov. Testovanie nástroja prebehlo v dvoch fázach. Prvá fáza testovania používala ako vstupné dáta známu komunikáciu. Výsledky ukázali stopercentnú úspešnosť, môžeme teda konštatovať, že navrhnuté riešenie je korektné a funkčné. Testovanie na nevidených dátach ukázalo vplyv starnutia databázy na úspešnosť detekcie. Pri staršej databáze bola úspešnosť detekcie nízka, avšak po jej aktualizácii sa nástroju podarilo detegovať aplikácie s vysokou mierou úspešnosti.

Možné využitia nástroja spočívajú v analýze prevádzky v sieti, napríklad pomocou IPFIX monitorovania. Administrátori siete by tak mohli sledovať toky dát a regulovať aplikácie, ktoré sa v sieti snažia komunikovať. Ďalšie možné rozšírenie predstavuje možnosti využitia v oblasti kriminalistiky. Spôsob využitia by spočíval vo vytvorení digitálneho profilu osoby páchajúcej trestnú činnosť, ktorá by mohla byť po pripojení do siete identifikovaná na základe aplikácií používaných na jej mobilnom zariadení.

Literatúra

[1] Levenshtein, V. I. Binary codes capable of correcting deletions, insertions, and reversals. *Doklady Akademii Nauk SSSR*. 1965, roč. 163, č. 4, s. 845–848.

[2] Rescorla, E. *The Transport Layer Security (TLS) Protocol Version 1.3* [Request for Comments]. RFC 8446. Internet Engineering Task Force (IETF), August 2018. 1-160 s. Dostupné z: https://tools.ietf.org/html/rfc8446.

[3] Rivest, R. L. *The MD5 Message-Digest Algorithm* [Request for Comments]. RFC 1321. Internet Engineering Task Force (IETF), April 1992. 1-21 s. Dostupné z: https://tools.ietf.org/html/rfc1321.

Príloha A

Opis datasetu

A.1 Používané aplikácie

Názov	Verzia
AccuWeather	7.5.1-9-google
CestovnePoriadky	1.4.0
Discord	72.15
GMail	2020.11.29.346182102
MujVlak	1.15.1
Seznam	7.11.1
Slack	21.05.10.0
Viber	14.4.0.9

Tabuľka A.1: Zoznam používaných aplikácií a ich verzie

A.2 Analýza datasetu

Názvy aplikácií sú z dôvodu prehľadnejšieho zobrazenia zadané ako ich iniciály. Táto skutočnosť bola opísaná v tabuľke 4.1.

Aplikácia	SSL pakety	ClientHello	Počet JA3	ServerHello	Počet JA3S	Počet SNI
AW	10827	518	3	518	24	4
CP	4325	71	2	71	13	1
DC	2015	68	3	68	10	2
G	2281	60	1	60	5	1
MV	1310	16	2	16	7	1
S	7248	341	55	339	36	17
SL	1201	89	3	89	10	1
V	340	10	1	10	4	1

Tabuľka A.2: Analýza datasetu

A.3 Metriky spúšťania skriptu na extrakciu záujmových dát

Beh	SSL pakety	Unikátne JA3	Unikátne JA3S	Unikátne SNI
1.	1503	1	12	2
2.	1824	2	5	2
3.	2688	0	4	0
4.	685	0	3	0
5.	4127	0	0	0

Tabuľka A.3: Postupný zber dát pre aplikáciu AccuWeather

Beh	SSL pakety	Unikátne JA3	Unikátne JA3S	Unikátne SNI
1.	560	1	5	1
2.	765	1	6	0
3.	1420	0	2	0
4.	852	0	0	0
5.	1493	0	0	0

Tabuľka A.4: Postupný zber dát pre aplikáciu CestovnePoriadky

Beh	SSL pakety	Unikátne JA3	Unikátne JA3S	Unikátne SNI
1.	1352	2	7	2
2.	458	1	3	0
3.	205	0	0	0

Tabuľka A.5: Postupný zber dát pre aplikáciu Discord

Beh	SSL pakety	Unikátne JA3	Unikátne JA3S	Unikátne SNI
1.	1230	1	3	1
2.	205	0	2	0
3.	68	0	0	0
4.	150	0	0	0
5.	628	0	0	0

Tabuľka A.6: Postupný zber dát pre aplikáciu GMail

Beh	SSL pakety	Unikátne JA3	Unikátne JA3S	Unikátne SNI
1.	365	2	4	1
2.	752	0	3	0
3.	193	0	0	0

Tabuľka A.7: Postupný zber dát pre aplikáciu MujVlak

Beh	SSL pakety	Unikátne JA3	Unikátne JA3S	Unikátne SNI
1.	1963	15	27	14
2.	2746	22	7	2
3.	2410	16	2	1
4.	552	0	0	0
5.	557	0	0	0

Tabuľka A.8: Postupný zber dát pre aplikáciu Seznam

Beh	SSL pakety	Unikátne JA3	Unikátne JA3S	Unikátne SNI
1.	482	2	8	1
2.	520	1	2	0
3.	199	0	0	0

Tabuľka A.9: Postupný zber dát pre aplikáciu Slack

Beh	SSL pakety	Unikátne JA3	Unikátne JA3S	Unikátne SNI
1.	115	1	2	1
2.	80	0	2	0
3.	145	0	0	0

Tabuľka A.10: Postupný zber dát pre aplikáciu Viber