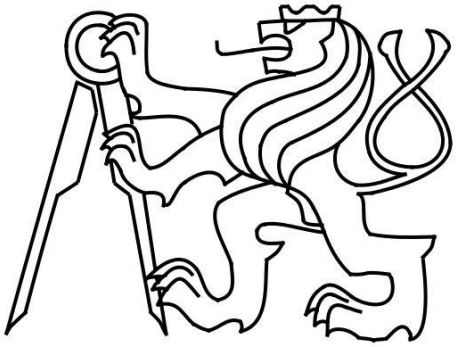
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická
Katedra telekomunikační techniky



Analýza nežádaného provozu mobilního telefonu

Unwanted Mobile Phone Traffic Analysis

Duben 2019

Diplomant: Bc. Jan Moravec

Vedoucí práce: Ing. Pavel Bezpalec, Ph.D

Čestné prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a že jsem v seznamu použité literatury uvedl všechny prameny, z kterých jsem vycházel. Dále prohlašuji, že nemám námitek proti půjčování nebo zveřejňování mé diplomové práce nebo její části se souhlasem katedry.

V Praze dne …….. ……..……

Podpis diplomanta

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Moravec** Jméno: **Jan** Osobní číslo: **440940**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra telekomunikační techniky**

Studijní program: **Elektronika a komunikace**

Studijní obor: **Komunikační systémy a sítě**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Analýza nežádaného provozu mobilního telefonu

Název diplomové práce anglicky:

Unwanted Mobile Phone Traffic Analysis

Pokyny pro vypracování:

Proveďte detailní analýzu uživatelem nevyžádané komunikace operačního systému mobilních telefonů.
Vypracujte metodu pro efektivní odchytávání veškerého datového provozu mobilního telefonu. Vezměte v úvahu všechny možnosti komunikace mobilního telefonu s Internetem (Wi-Fi, mobilní data ...). Zaměřte se zejména na mobilní telefony se systémem Android.

Seznam doporučené literatury:

Android Open Source Project [online]. Dostupné z: https://source.android.com/
Berka, Petr. Dobývání znalostí z databází. Vyd. 1. Praha: Academia, 2003. 366 s. ISBN 80-200-1062-9.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Pavel Bezpalec, Ph.D., katedra telekomunikační techniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **25.02.2019** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce: **19.02.2021**

Ing. Pavel Bezpalec, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

Datum převzetí zadání	Podpis studenta

Poděkování

Děkuji vedoucímu práce Ing. Pavlu Bezpalcovi, Ph.D. za velmi užitečnou metodickou pomoc a cenné rady při zpracování této diplomové práce. Dále bych chtěl poděkovat mým přátelům a mojí rodině, která mě podporovala po celou dobu mého studia i při psaní této diplomové práce.

Anotace

Práce se zabývá komunikační bezpečností mobilních telefonů na operačním systému Android. Byly analyzovány metody záchyty síťové komunikace a popsáno pro jaké případy je vhodně použít danou metodu. Dále byl uveden přehled možností prolomení šifrovaného provozu SSL/TLS, včetně obejití *certificate pinningu* a jejich zhodnocení. V praktické části práce bylo vybráno osm aplikací, u kterých bylo prolomeno šifrování provozu a proveden záchyt komunikace. Závěr práce tvoří vyhodnocení všech výsledků bezpečnostní analýzy síťové komunikace a sumarizace získaných poznatků.

Klíčová slova

mobilní bezpečnost, SSL/TLS, síťový záchyt, osobní data

Summary

The thesis is focusing on mobile security of the Android operation system. Methods of network sniffing were analysed and described in which cases it is usefull to use specific aplication. The methods of network sniffing were analysed and described in which cases it is suitable to use specific one. In the next chapter there are summarized the options to bypass the SSL/TLS communication including a certificate pinning. In the practical part eight aplication were chosen followed by breaking throught their ecryption with communication record.In practical part there was chosen eight aplication. Then breaking throught their encryption with communication record. In the conclusion of this thesis are results of all tests of security analysis of network communication and summarization of reached results.

Index Terms

mobile security, network sniffing, SSL/TLS, personal data

Obsah

1. Úvod

Mobilní zařízení hrají dnes velice důležitou roli, protože jako přenositelná zařízení obsahují velké množství osobních informací. Tyto informace reflektují uživatelské návyky, zvyky, zájmy a vztahy. Proto je nutné věnovat pozornost bezpečnosti těchto zařízení větší měrou než jakémukoli jinému zařízení. Jasným důkazem o důležitosti role mobilních zařízení je objem datového provozu, který za posledních 5 let vzrostl desetkrát více než provoz na fixních přípojkách. [1]

Chytré telefony jsou zařízení podporující datové připojení k internetu a jako operační systém převážně využívají *Android* nebo *iOS*. Ve světě *Androidu* se nacházejí miliony aplikací, které jsou dostupné skrze obchod Google Play, nebo obchody třetích stran. Aplikace, které se nachází na oficiálním úložišti, jsou před jejich zveřejněním kontrolovány automatickým softwarem, zdali neobsahují *malware,* či jiný škodlivý kód. Ale ani tyto kontrolní mechanismy nejsou stoprocentně účinné pro kontrolu nebezpečí, které představuje únik dat. Při stahování těchto aplikací je nutné si uvědomit, že právě ony si žádají od uživatele souhlas o přístup k osobním údajům.

Podle agentury ENISA (Agentura Evropské unie pro bezpečnost sítí a informací) je hrozbou číslo jedna únik dat, ke kterému může dojít různými způsoby. [2] Pokud dojde ke ztrátě nebo odcizení chytrého telefonu a jeho paměť nebo vyměnitelná média nejsou chráněna, může dojít k umožnění přístupu útočníka k datům uživatele. Další problematickou stránkou jsou klienti *cloudových* úložišť. Pokud dojde k odcizení mobilního telefonu, získá tím útočník potencionální přístup k často kritickým souborům, jako jsou již zmíněná soukromá data, ale zde se často nachází také dlouhodobé zálohy uživatele. Většina aplikací používaných ve *smartphonu* navíc vyžaduje, aby uživatel změnil nastavení ochrany osobních údajů, proto aby aplikace mohla přistupovat k citlivým informacím, jako jsou kontakty, fotografie atd. Dalším možným způsobem, při kterém může dojít ke ztrátě dat je, pokud je chytrý telefon připojen k datové síti. Ačkoli mnozí uživatelé mobilních zařízení si jsou vědomi, že aplikace, které používají, mohou sdílet svá osobní data s třetími stranami, mnozí si neuvědomují, jak často se to děje. [3]

Tato práce si dává za cíl popsat, jakým způsobem může docházet k úniků osobních údajů bez vědomí uživatele, když uživatelé sdílejí osobní informace s různými mobilními aplikacemi prostřednictvím datového připojení. Mezi tyto informace patří uživatelská jména, hesla, vyhledávané dotazy a údaje o poloze / geografických souřadnicích. Dále pak prověření, jak tyto aplikace zpracovávají osobní údaje uživatele, a to sledováním typu dat, která mohou sdílet se třetími stranami.

2. Teoretický rozbor

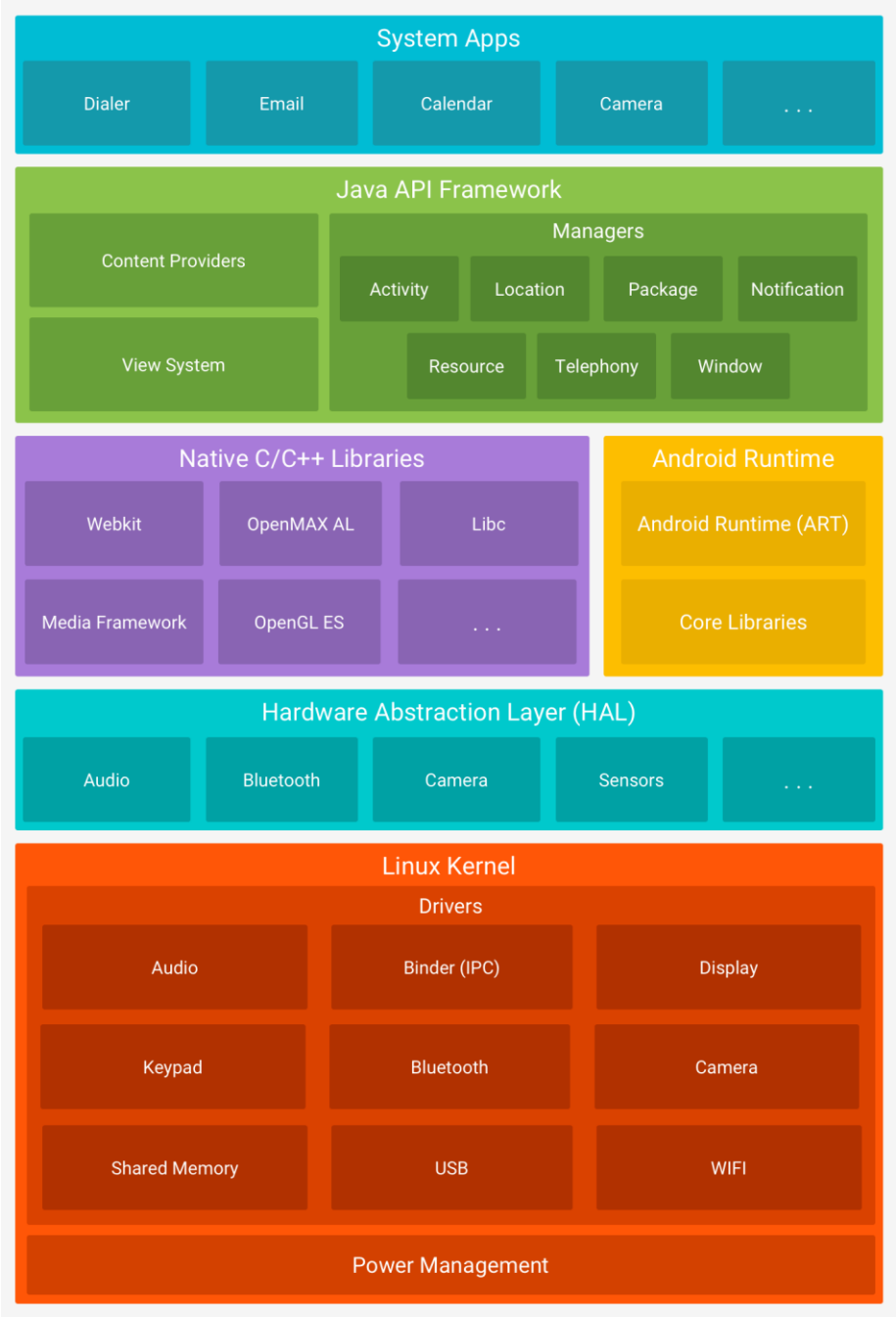
2.1 Android

Historie operačního systému *Android* sahá do roku 2003, kdy byl založen americký *„startup“*, který později skoupila společnost *Google*. Plánem této akvizice bylo vytvoření vlastního mobilního telefonu, který do té doby *Google* neměl, tak aby mohl masivně vstoupit na tento segment trhu. Později v roce 2007 byl zahájen vývoj zcela nového *„open source“* operačního systému, který byl založen na operačním systému *Linux* 2.6. První verze systému byla vydána v roce 2008 a v roce 2009 byla uvolněna SDK verze (*software development kit*), aby se do vývoje systému mohli zapojit další vývojáři a zůstala zachována myšlenka *open source* platformy. Již po čtyřech letech od vydání první verze začal *Android* dominovat oblasti operačních systému pro mobilní telefony s 59% podílem na trhu a zcela nahradil do té doby velice oblíbený *Symbian*. Na konci roku 2018 zabírá *Android* na trhu více jak 88% podílu mezi operačními systémy na mobilních platformách.

Při takovéto dominanci na trhu je více než jasné, že většina dnes dostupných aplikací se primárně vyvíjí pro *Android*, což dokazují i čísla zobrazující počet dostupných aplikací. Momentálně je možné na *Google Play* stáhnout oficiálně okolo 3 800 000 aplikací což je prakticky dvojnásobek, oproti konkurenčnímu *iOS marketu*, kde se jich nachází cca 2 000 000.

Takto velká podpora a oblíbenost platformy samozřejmě přitahuje pozornost také vývojářů *malwaru*, popřípadě jiného škodlivého kódu. V průběhu vývoje jednotlivých verzí, kdy dnes (březen 2019) je poslední vydaná verze 9.0.0, došlo k zjištění mnoha bezpečnostních děr, které umožňovaly otevřít tzv. *„backdoor“* k systému a plně jej ovládnout. Nejzávažnější z nich pochází z roku 2011, kdy skrze nezašifrovaný přihlašovací token k webovým serverům bylo možné *„token“* odchytit a znovu použít k přihlášení k uživatelově účtu. Značnou kritiku sklidil *Android* v průběhu vývoje k možnosti šifrování komunikace, kterou v dřívějších verzích prakticky nepodporoval, dnes je situace mnohem lepší. [4]

2.1.1 Android architektura

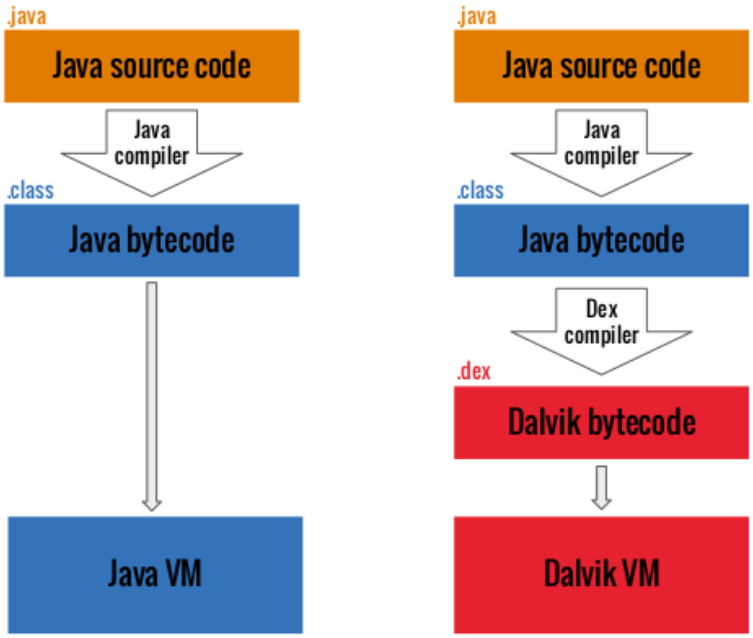


Obrázek 1.1 - Struktura Androidu

Na vrchu architektury operačního systému *Android* se nachází aplikace poskytující základní funkcionality (email, kalendář, fotoaparát). Přímo pod touto úrovní se nachází *Java API framework*, který poskytuje modulární komponenty pro vývojáře aplikací. V *Androidu* se také nachází nativní knihovny psané v *C / C++,* které je možné využít například pro 2D a 3D grafiku. Další komponentou je *Android Runtime*, která slouží více instancím (aplikacím) optimalizovat spotřebu systémových zdrojů, tak, aby byla co nejmenší. *Hardware Abstraction Layer* vrstva obsahující skupinu knihoven, které slouží k poskytovaní hardwarových zdrojů vyšších úrovní, kde je *JAVA API*. Základem celé struktury je poslední vrstva, kde se nachází *Linux Kernel*. Tato vrstva spravuje ovladače pro jednotlivé hardwarové komponenty, správu procesů, nebo také spravuje paměť.

Android aplikace jsou obvykle psány v *Javě* a kompilovány do *„Dalvik bajt“* kódu, který je poněkud odlišný od tradičního *„Java bajt kódu“*. *„Bajt kód Dalvik“* je vytvořen tak, že se nejprve zkompiluje kód *Java* do souborů s příponou *„.class“* a poté se pomocí nástroje *dx* převede *bajt kód JVM* na formát *Dalvik s příponou „.dex“*. Aktuální verze *Androidu* provede tento bajt kód v *Android runtime* (ART). ART je nástupcem původního běhového prostředí *Androidu*, virtuálního stroje *Dalvik*. Klíčovým rozdílem mezi *Dalvik* a ART je způsob, jakým je prováděn bajt kód.

V *Dalvik* je kód přeložen do strojového kódu v době provádění, proces známý jako *„just-in-time“* (JIT) kompilace. Kompilace JIT nepříznivě ovlivňuje výkon: kompilace musí být prováděna při každém spuštění aplikace. Pro zlepšení výkonu byl vytvořen proces *„ahead-of-time“* (AOT), což znamená, že aplikace jsou překompilovány před prvním provedením. Tento předkompilovaný strojový kód se používá pro všechny následné spuštění. AOT zlepšuje výkon a zároveň snižuje spotřebu energie.



Obrázek 1.2 - Kompilace v Androidu

2.1.2 Systém povolování přístupu na mobilních platformách

Vzhledem k možnosti instalace aplikací třetích stran bylo nutné vytvořit systém, který uživateli dává možnost výběru, přidělit aplikaci různé systémové zdroje. Mobilní platformy využívají systém k přidělování přístupu jednotlivým zdrojům, jako například síťovému přístupu, mikrofonu, kalendáři, kontaktům, kameře nebo také přístup k poloze telefonu. [5]

Sandbox

Android přiřazuje každé aplikaci během instalace unikátní *„Linux ID“* a spolu s faktem, že každá aplikace má vlastní instanci virtuálního stroje, to vytváří prostředí nazývané *Sandbox, k*de je garantována isolace každé z nainstalovaných aplikací. Pokud aplikace chce přistoupit ke chráněným zdrojům (paměť, kamera, poloha) nebo k datům jiné aplikace, musí požádat o schválení tohoto přístupu uživatele. Tomuto systému povolování přístupu se říká systém oprávnění. Do verze 6.0 vyžadoval *Android* tyto oprávnění již během instalace aplikace, od této verze však požadavky na oprávnění fungují dynamicky, a to vždy, pokud se vyskytne požadavek v aplikaci na přístup k některým chráněným prostředkům. Tímto modulem je řízen maximální počet systémových prostředků přidělených aplikacím a zabraňuje

vyhrazení příliš mnoha zdrojů. Další nepostradatelnou výhodou je fakt, že aplikace, která havaruje, neovlivní ostatní aplikace spuštěné v zařízení.[6]

2.2.3 Přístup do privilegovaného režimu Androidu

Neboli také obecně používaný název *Root* který popisuje proces, který přepne *Android* do tzv. privilegovaného režimu a otevře tím možnost nahradit systémové aplikace a specifikace, které jsou od výrobce zamčené. *„Rootovaní“* se provádí, pokud je z nějakého důvodu potřeba přistupovat k pokročilejším a samozřejmě také potencionálně nebezpečnějším operacím. Výrobci mobilních telefonů se například v Evropě k takovýmto úpravám staví velice odmítavě, a pokud zjistí takovouto úpravu operačního systému, automaticky ruší záruku, která se k telefonu váže. Ostatní kontinenty to mají trochu jiné, například v USA je tento krok zcela legální a v Austrálii pouze za předpokladu, že budou do telefonu instalovány legální aplikace. Přístup k právům *„root“* uživatele brání výrobci mobilních zařízení především z důvodu ochrany systému a uživatelských dat.

Pro přístup k pokročilejším funkcím systému slouží oprávnění, které přesně definují funkci uživatele nebo aplikace. Oprávnění také rozhodují o tom, kdo může v dané složce vytvářet nové soubory a kdo je může (v případě, že se jedná o spustitelný soubor – typicky aplikaci) spouštět. Někteří uživatelé k danému souboru či složce oprávnění mají, jiní nikoli. Uživateli, který oprávnění nemá, je případný pokus o přístup systémem zablokován.

Aplikace také může požádat o další oprávnění – například k přístupu do adresáře, na paměťovou kartu nebo do knihovny fotografií. Jestliže jsou tyto požadavky schváleny, systém ví, že byla aplikaci s určitým ID, uživatelem udělena povolení přistupovat k daným souborům.

Technicky je možné změnit způsob, jakým telefon přistupuje k souborům, které používá k běhu systému, a přiřadit jím ID uživatele se zvýšenými oprávněními, nebylo by to však bezpečné.

První ochranou proti změně přístupu k souborům je fakt, že binární aplikaci, která se stará o změnu uživatelského ID, nezahrnují do systému, proto uživatel nemůže změnit své ID. Další stupně zabezpečení (typicky soubory v zavaděči systému nebo samotné jádro operačního systému) se pak starají o to, aby zabránily v možnosti změny uživatelské identifikace v rámci *SELinuxu* (*Security-Enhanced Linux*). Někteří výrobci pak přidávají ještě další vrstvu ochrany – příkladem je třeba *Samsung Knox*. Téměř všichni výrobci požadují k pokročilejším operacím takzvané odemknuté *„bootloaderu“*.

Pokud je snaha o to získat práva *„root“* uživatele, je hlavním cílem překonat ochrany které jsou v systému vytvořeny. Lze toho dosáhnout buď odemknutím *„bootloaderu“* prostřednictvím oficiálních prostředku od výrobce, nebo cestou méně oficiální a tj. použitím nějakého bezpečnostního nedostatku (*exploitu*). Tímto způsobem se získá možnost uložit binární soubor *„SubstituteUser“* do složky uložené v proměnné PATH, odkud je možné jej spustit.

Binární soubor *SU* používá příznaky, kterými při běhu říká systému, na jaké uživatelské ID chce přepnout. Na *Androidu* nikdy žádný *root* uživatel neexistoval, proto po zadání *SU* dojde tedy k přepnutí uživatele na *root* s uživatelským *ID 0*. Tímto krokem se získají práva “super uživatele,” který si v systému může dělat prakticky cokoli, například odstranit aplikace předinstalované výrobcem (“*bloatware*”), stejně jako jakékoli systémové soubory uložené v zařízení nebo nastavovat některé parametry hardwaru – například měnit taktovací frekvenci procesoru. [7]

S tímto způsobem úpravy se také samozřejmě pojí jistá rizika, a nejsou to jen rizika bezpečnostní, jak již bylo zmíněno. Neexistuje zde po této úpravě ochrana před smazáním například systémových

souborů, které jsou nutné pro běh systému. Po smazání těchto souborů nemusí *Android* vůbec nastartovat nebo se může chovat nestabilně a nepředvídatelně.

Dalším velice vážným problémem se získáním práv „super uživatele“ je bezpečnost osobních údajů. V aplikacích, které jsou v systému nainstalované, mohou funkce na spuštění příkazu *SU,* po získáních takto vysokých oprávnění mohou například odesílat citlivé osobní údaje na servery třetích stran bez vědomí uživatele. [8]

2.2 Ztráta integrity mobilní komunikace

IMSI (*International Mobile Subscriber Identity*) jedná se o číslo, které přiděluje mobilní operátor SIM kartě v síti GSM nebo UMTS. Číslo se odesílá z mobilního telefonu operátorovi a slouží k dohledání detailních informací v jeho databázi.

„IMSI catcher“ je v České republice znám jako „Agáta“. Jedná se o nástroj sloužící jako náhradní základnová stanice mobilní sítě a může být využit pro lokalizaci mobilního telefonu, či k provedení *„man in the middle“* (MITM) útoku. Systém vysílá stejně jako základnová stanice mobilního operátora, avšak s tím rozdílem, že vysílá vyšším výkonem, proto zařízení v okolí upřednostní právě tuto stanici. Pomocí tohoto systému je například možné dohledat přesnou polohu mobilního telefonu. Stačí znát telefonní číslo, které využívá hledaný telefon a pomocí triangulace (tzn. musí měnit polohu) dojde k přesnému dohledání telefonu. Tento postup využívá také například Policie ČR. [9]

Každý tento systém může prolomit *A5/1* šifrování v reálném čase, toto šifrování se používá pro sítě 2G. Šifrování *A5/3*, které se běžně používá v sítích 3G nebylo zatím prolomeno, alespoň ne veřejně, ale již se objevily teoretické útoky. Šifrování 3G a 4G sítí je aktuálně bráno jako bezpečné, proto se v *IMSI catcheru* využívá pro MITM útok vynucení mobilního telefonu přejít na síť 2G kde je možné prolomit *A5/1* šifrování, nebo jej, popřípadě úplně vypnout. [10] Možnou obranou proti tomuto typu útoku je úplné zakázaní 2G sítí, pokud to daný telefon podporuje. Nesnadnou cestou, která by otevřela možnosti obrany proti takovémuto útoku, se vydal projekt *Android-IMSI-Catcher-Detector*. Tento projekt se zabývá vývojem detektoru takto upravených stanic na základě přijatých dat. [11]

Proti takovýmto hrozbám narušení soukromí a integrity dat, je obrana obtížná. Na druhou stranu, potencionální zneužití například *„IMSI catcheru“* ze strany vlády je v České republice velice malé, avšak existují zde bezpečností hrozby, které jsou daleko závažnější a kterým je většina zařízení vystavena dnes a denně.

3. Metody pro efektivní odchytávaní síťového provozu

Aby bylo možné zjistit, jestli nedochází k únikům nežádoucích dat mobilního zařízení, je nutné provést reverzním inženýrstvím analýzu každé aplikace, která se v systému nachází, včetně systému samotného. Druhou možností je analýza síťové komunikace celého zařízení. Proto jsou v této kapitole popsány technické způsoby, jakými je možné datovou komunikaci odchytit a následně jsou kompletně rozebrány způsoby dešifrování, pokud je komunikace šifrována.

K analýze datového provozu je možné využít převážně dva způsoby, z nichž každý se hodí k jiné činnosti a jiné situaci.

Pasivní analýza

První možná analýza síťového provozu využívá pasivní odposlech síťových paketů a jejich uložení do specifického souboru, například ve formátu *„.pcap“* obecně pojmenovaný jako „záchyt“. Takovýto soubor je možné také vytvořit přímo na *Androidu* pomocí několika aplikací, které budou popsány později.

Aktivní analýza

Druhá možná varianta je aktivní analýza provozu. Jedná se o metodu, kdy se veškerá komunikace přesměruje přes „*proxy server*“. Výhoda tohoto řešení spočívá v možné aktivní modifikaci požadavků a odpovědí již během komunikace, a tudíž možnosti využití různých předem připravených scénářů, pokud by se jednalo například o penetrační testování.

„Proxy server“ funguje jako prostředník mezi klientem a cílovým serverem, překládá klientské požadavky a vůči cílovému počítači vystupuje sám jako klient. Přijatou odpověď následně odesílá zpět na klienta. Může se jednat jak o specializovaný hardware, tak o software provozovaný na běžném počítači. *„Proxy server“* odděluje lokální počítačovou síť od internetu. *„Proxy serverů“* existuje několik druhů.

Proxy server – tento typ *proxy* nijak žádosti od klienta neupravuje, pouze předává cílovému serveru, proto se také nazývá jako „brána“ nebo „tunelovací proxy“

Forward proxy – rozdíl oproti klasickému *proxy* serveru je ten, že v tomto případě se načítají žádosti kdekoliv z internetu, je to internetově orientovaný *proxy*.

Reverzní Proxy – jako v předchozím případě se jedná o internetově orientovaný *proxy*, který se používá k řízení ochrany přístupu v privátních sítích. Může zajišťovat autentizaci, dešifrování, komprese dat, ukládání do mezi paměti anebo vyrovnávat zatížení. [12]

3.1 Wifi

První z možností datového provozu, kterou obsahují všechny mobilní platformy se systém *Android*, je WIFI. Přes toto rozhraní je celosvětově přenášeno největší množství dat a většina verzí systému *Android* dokonce zobrazuje hlášku, která upozorňuje před stahováním souborů, pokud není WIFI zapnuta a jsou využívána mobilní data. WIFI je ve většině případu technologií lokálních sítí a na mobilních zařízeních je provozována ve dvou režimech.

Access Point (AP) **–** přístupový bod řídí komunikaci mezi WIFI zařízeními, která jsou zapojena v infrastrukturním režimu. Přístupové body je možné použít pro poskytování různých služeb pro lokální síť a připojení k internetu.

Ad-hoc – v tomto režimu se navzájem spojují dva klienti, kteří jsou v rovnocenné pozici (*peer-to-peer*). Vzájemná identifikace probíhá pomocí SSID. Obě strany musí být v přímém rádiovém dosahu, což je typické pro malou síť nebo příležitostné spojení, kdy jsou zařízení ve vzdálenosti několika metrů.

3.1.1 Aplikace pro záchyt

Na *Android* existuje početné množství aplikací, které vytvářejí záchyty a fungují na principu *proxy* serveru. To znamená, že veškerý provoz je směřován skrze ně, jedná se tedy o aktivní záchyt. Tyto aplikace lze rozdělit na dvě skupiny. Za prvé ty, co potřebují ke své funkci *„root“* telefonu a za druhé ty, co fungují i bez něho.

Většina aplikací požaduje *„root“* přístup k zachytávaní paketů, důvodem je promiskuitní mód nebo monitoring mód. Pokud je aplikace spuštěna v promiskuitním módu je možné zachytit každý paket, který je transportován přes síť. Pokud není provoz šifrován, může být taky veškerý provoz přečten.

1) Z první kategorie je nejlepší aplikací *TCPdump*. Tato aplikace nabízí vestavěné grafické rozhraní a je možné využít jeho, nebo čistě příkazového řádku. Pracovat s aplikací lze i vzdáleně z připojeného počítače skrze *Android Debug Bridge*. Pro vytváření záchytů se využívá v zařízeních, u kterého není za hodno porušovat záruku kvůli „*rootu*“.

2) Z druhé kategorie aplikací je jako příklad možné uvést *Drony*, *Network Connections*, *Packet Capture* nebo *tPacketCapture* který byl mezi developery velice oblíbený, avšak byl stažen z *Google Play* a plně jej nahradila aplikace *Netcapture*. Všechny tyto aplikace fungují na podobném principu, a to je využití *Android VPN* služby k přerušení datového toku mezi aplikací a cílovým serverem a jeho následné zachycení.

Dalším typem aplikace, která funguje na podobném principu je *Android PCAP*. Tato aplikace ke svému chodu nepotřebuje *„root“*, avšak je k jejímu provozu potřeba externí wifi karta. Aplikace se snaží obejít restrikce, které jsou na interní wifi nastaveny tak, že využije externí wifi kartu připojenou přes USB rozhraní a následně provoz zachytává přímo na USB rozhraní. Tato aplikace, která poskytuje proprietární řešení, ke kterému je nutné využít externí wifi a ze strany vývojářů je vývoj dosti zaostalý, nenabízí ideální řešení pro tuto práci, avšak ukazuje další cestu, kterou je možné se vydat. [13]

Ze záchytů z těchto aplikací je možné vyčíst ID aplikace a URL, ke kterému se snaží přistupovat, a občas status kód, ale nic jiného. Pro zjištění potřebných informací, tj. celý požadavek s odpovědí včetně těla celé zprávy je toto řešení nevhodné. Dalším problémem, který vzniká s použitím podobných aplikací je jejich důvěryhodnost. Například aplikace *tPacketCapture* má na *Google Play* přes 100 000 stažení a z mnoha recenzí je možné se dočíst, že vytvořené *pcap* záchyty jsou odesílány na vzdálenou *proxy* včetně jména uživatele mobilního telefonu, čísla mobilního telefonu a IMEI navíc pomocí nezašifrovaného HTTP.

3.1.2 Wifi hotspot

Jedná se o možnost, kdy se využije síťová karta, například v notebooku, a vytvoří se z ní *hotspot*, ke kterému se následně přípojí mobilní zařízení. Notebook musí být připojen do internetu přes *ethernet* a síťová karta musí být nastavena do monitorovacího módu. Ne každá síťová karta toto však umožňuje, pokud ano, tak je na *linuxu* možné pro toto využít navržený program *airmon-ng*.

Na Windows lze takto vytvořit wifi *hotspot* pomocí příkazu:

```
netsh wlan set hostednetwork mode=allow ssid=XXX key=XXX
```

Tento příkaz vytvoří nové síťové připojení a následující příkaz ho zapne.

```
netsh wlan start hostednetwork
```

Poté stačí přejí k nastavení *ethernetu* a na něm povolit sdílení připojení pro nově vytvořené spojení. Konektivita přes takto vytvořený *hotspot Android* detekuje spolu s bezpečnostní hláškou „Toto připojení může být monitorované“. [14]

3.1.3 Virtualizace Android

Prvně je nutné jednoznačně definovat, jaký je rozdíl mezi virtualizací a emulací. Účel virtuálních strojů je vytvoření izolovaného prostředí od hardwaru, naopak účel emulátoru je co nejdůvěryhodněji reprodukovat chování určitého hardwaru. Oba mají za cíl jistou úroveň nezávislosti na hardwaru, na kterém budou spuštěny, ale virtuální stroj má za cíl simulovat jen tolik hardwaru k tomu, aby mohla být vykonávána práce uživatele.

Konec konců, virtuální stroj nemusí jednat jako jakýkoliv hardware, který opravdu existuje. Emulátor na druhou stranu zkouší přesně reprodukovat chování, včetně různých výstředností nebo chyb, kterých se skutečný hardware dopouští při svém chodu.

Virtualizace či emulace nejsou sice rozhraní skutečného mobilního telefonu, ale je nutné se o nich zmínit. Většina vývojářů aplikací pro Android nepoužívá k testování svých aplikací skutečné zařízení ale právě virtuální stroj či emulátor. Hlavním důvodem, proč je toto výhodnější, než skutečné zařízení je rychlost a efektivnost, kterou přináší pouhé spuštění softwaru, které pracuje jako skutečný Android. Druhým významným důvodem je možnost simulovat určité chování aplikace nebo systému, které na reálném zařízení není možné, například z důvodu bezpečnosti.

Hlavním leaderem, který pracuje na vývoji virtuálního stroje *Androidu* je projekt *Android-x86* [15], na jehož stránkách je možné stáhnout různé verze. Virtuální stroj byl testován na notebooku s procesorem *Intel Core i5-4010*, operační pamětí 12 GB a grafickou kartou *Nvidia 840M*. K virtualizaci byl použit *Oracle VM Virtual Box* a na spuštění stroje byla použita grafická akcelerace *Hypev-V*. Bohužel po otestování několika verzí, od *Androidu* 4.4 až po 8.1., a vyzkoušení několika softwarových úprav na rady vývojářů, bylo shledáno, že virtuální stroj dosahuje vysoké prodlevy a je pro účely testování absolutně nepoužitelný.

3.1.4 Android emulátor

Z předchozí části vyplývá že dalším možným způsobem je využít emulaci operačního systému *Android*, který se spustí na zařízení, na kterém je prováděn záchyt, a v emulátoru se následně nainstalují aplikace, která je cílem analýzy. Tento postup se zdá jako nejjednodušší, ale opak je pravdou. Samotné spuštění emulátoru není jednoduché a provází mnoho dílčích chyb, jak ze strany virtualizačního software, (například *Oracle Virtual Box*), tak od samotného emulátoru. Další podstatnou nevýhodou, která se však v konečném důsledku může ukázat jako výhoda, je samotné prostředí emulátoru. Tyto emulátory byly navrženy pro vývojáře aplikací pro *Android* a jsou k tomu uzpůsobeny. Emulátor slouží ke sledování chování jedné aplikace, ve které dochází k *debuggování*, ne však většího počtu aplikací, které jsou standardně přítomny na telefonu obyčejného uživatele.

Samotná instalace aplikací není na těchto emulátorech jednoduchá. V roce 2014 *Android* změnil licenční podmínky a tím nedovoluje vývojářům dodávat emulátory s předinstalovaným *Google Play*. Vývojáři se těmito pravidly skutečně řídí, komerční emulátory *Genymotion* a *Android SDK*, které byly vyzkoušeny *Google Play* nemají. Komunitou vývojářů však byly vytvořeny *toolkity*, přes které je *Google play* možné do těchto emulátorů doinstalovat. Aplikace se se do emulátoru dají samozřejmě stáhnout i z jiných zdrojů, avšak není zde jistota, že nebyly nějakým způsobem pozměněny. Naproti tomu aplikace v prostředí *Google Play* prochází před samotným zveřejněním kontrolou, která by měla odhalit různé nežádoucí chování aplikace.

Genymotion – jedná se o profesionální virtualizační nástroj pro *Android*, který je možné spouštět jak v *cloudu*, tak v *desktopové* variantě. Nástroj je vytvořen pro vývojáře *Android* aplikací, kteří potřebují maximální realističnost užívaného systému a také přijatelnou rychlost. Díky hardwarové podpoře je možné taky využít GPS nebo multidotykovou obrazovku. Cena tohoto nástroje je pro jednotlivce 99 eur za rok, je zde však možnost stáhnout měsíční zkušební verze. Emulátor nabízí kompletní podporu jednotlivých verzí *Androidu* včetně nejnovější 9.0.1.

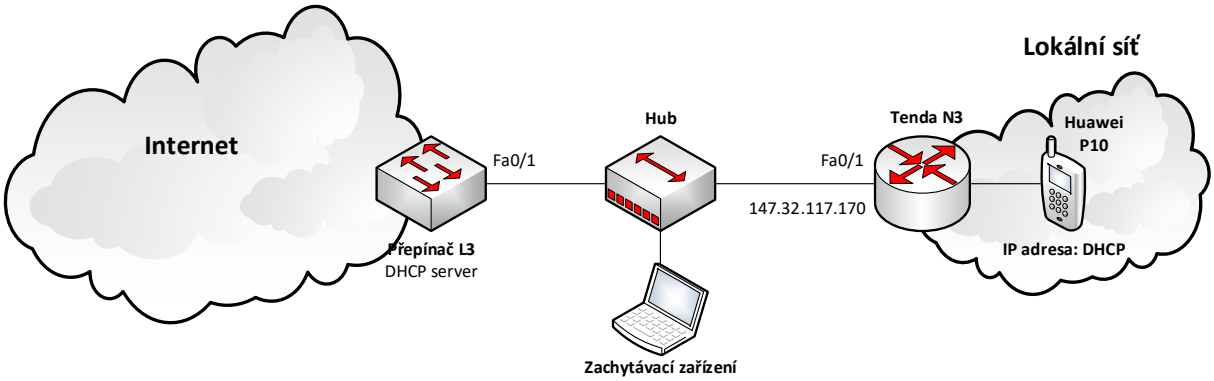
Android Studio Emulator – je IDE navržené přímo pro vývojáře aplikací pro *Android*, ve kterém je možné psát *java* programy a XML pro design. Naproti tomu *Genymotion* je pro testovací účely a simulaci konkrétních virtuálních zařízení, kde je možné spouštět nebo testovat aplikace (soubory v příponou .apk. Je také rychlejší než *Android studio,* a proto se více hodí právě pro testování.

Existují i další emulátory jako *Approov*, která by umožňovala tuto práci vypracovat bez dalších nákladů?

3.1.5 Zařízení zachytávají provoz

Pro zachycení provozu na wifi rozhraní je možnost také využít fyzického odposlechnutí provozu, kdy se mezi mobilní zařízení komunikující skrze wifi k cílovému serveru vloží prvek, které umí provoz odposlechnout.

Z důvodu nedostupnosti přepínače s možností zrcadlení portu bylo využito zařízení hub, které fyzicky rozděluje elektrické signály. Funguje na principu, pokud něco obdrží na jednom rozhraní, automaticky rozesílá na všechny ostatní. K odposlechnutí je možné využít například následující topologii.



Obrázek 3.1 - Fyzické zachycení provozu

V této topologii je mobilní zařízení (*Huawei P10*) součástí LAN sítě a ke komunikaci do internetu využívá přímo připojený směrovač *Tenda N3*, ke kterému je připojen pomocí wifi. Směrovač má na svém WAN portu připojený HUB, který je až následně připojen do switche představujícího výchozí bránu. Následně je k hubu připojeno zachytávací zařízení, které vidí veškerý provoz mezi LAN sítí a internetem. Pokud je však v této topologii na výchozí bráně do sítě aktivován *„whitelist“* MAC adres, který povoluje připojení pouze jedné konkrétní adresy na jedno rozhraní, musí se zamezit zachytávacímu zařízení v aktivní komunikaci. Toho je možné docílit například vytvořením pravidla ve firewallu, které blokuje veškerý odchozí provoz, ale příchozí zůstane neporušen. Z dnešního pohledu však použitý HUB nabízí velice nízkou propustnost, omezenou pouze na rychlost 10Mbit/s. V takto provedené topologii může uživatel jen těžko poznat, že dochází k odposlouchávaní jeho provozu, pokud by se jednalo o *hackerský* útok. Existují i další způsoby, jak fyzicky odposlechnout provoz, například využití přepínače s funkcí „*port mirroring*“, kdy se veškery provoz z jednoho portu zrcadlí na port druhý a topologie bude tedy úplně stejná jako na obrázku 3.1, pouze místo hubu se zvolí onen přepínač.

3.2 Mobilní data

Datová komunikace skrze mobilní data dnes již popularitou dohání wifi, protože limity přenesených dat, které operátoři nabízí, jsou již dostatečné pro každodenní využívání aplikací. Metody pro odchytávaní datového přes toto rozhraní jsou však více omezené než skrze wifi.

První z možností, jak zachytit data která mobilní telefon přenáší je požádat o záchyt mobilního operátora. Operátor však nemusí mít prostředky na odchycení datového provozu z jednoho konkrétního telefonu.

Druhou z možností pro zachycení komunikace je využít softwarové definované rádio (SDR). SDR je rádiový systém, v němž se rozhodující část zpracování signálu realizuje softwarově programovatelnými obvody. Díky tomu lze pouhou změnou softwaru používat různá kmitočtová pásma a různé komunikační protokoly. Změnou protokolů lze simulovat základnovou stanici různých generací sítě.

Třetí a nejefektivnější metodou je využití *proxy* VPN kterou je na *Androidu* možné nastavit na určitou IP adresu. Operační systém Windows nabízí funkce při vytvoření příchozí VPN a následně na tomto rozhraní provést záchyt například programem *Wireshark*. V tomto případě se objevuje problém, pokud nemá zařízení hostující VPN statickou veřejnou IP adresu, kterou ovšem většina zařízení nemá, je nutné využít služeb dynamické DNS. Jednou z nich je například *dyndns.org* která automaticky překládá měnící se IP adresu na jméno hostujícího počítače a toto jméno zůstává beze změny. [16]

Při zachycení komunikace na obou těchto rozhraních ovšem vyvstává jeden vážný problém vzhledem k další analýze. Ve většině případů jsou data, které proudí síťovými rozhraními šifrována, a to z velké části omezuje jejich analýzu.

3.4 Šifrované spojení

Komunikace klienta a serveru (popřípadě klienta a klienta), byla v počátcích internetu nešifrována a kdokoliv, kdo se dostal mezi obě komunikující strany byl schopen komunikaci zachytit (útok typu *Man-in-the-middle).* Tím porušit integritu komunikace a mohl také bez problému číst a upravovat zprávy tak, aby to ani jeden z účastníků nezachytil.

Toto dalo za následek počátku šifrování komunikace mezi oběma komunikujícími stranami. V rámci RM OSI modelu tak vnikla nová podvrstva SSL (*Secure Sockets Layer*), která toto zařizuje. Ačkoliv se zažila zkratka SSL jako obecné označení pro kryptografické protokoly pro ochranu internetové komunikace, SSL byl původně název jednoho konkrétního protokolu. Ten už byl ale ve většině aplikací nahrazen modernějším protokolem TLS (*Transport Layer Security*). Jako obecné označení šifrovacích protokolu se ale stále často používá zkratka SSL.

Protokoly SSL existují ze dvou důvodů. Zaprvé zaručují identifikaci. Díky SSL má klient i server jistotu, že komunikují opravdu spolu navzájem, a ne s někým, kdo se za jednu ze stran vydává. Zadruhé zajišťují šifrování komunikace – zařídí domluvu šifrovacího algoritmu, bezpečnou výměnu klíčů a tvorbu společného tajemství, které pak klient i server používají k šifrování komunikace.

To, že se nějaký z rodiny protokolů SSL v komunikaci se serverem používá, se pozná tak, že v adresním řádku prohlížeče stojí na začátku místo zkratky protokolu HTTP zkratka HTTPS (tedy *Hypertext Transfer Protocol Secure*, někdy také HTTP over SSL). Neznamená to tedy, že je klient při prohlížení v bezpečí – to protokol sám nemůže zaručit – ale znamená to alespoň fakt, že je komunikace šifrovaná, a tedy odolnější vůči odposlouchávání.

SSL zaručuje šifrované spojení. Avšak pokud je spojení šifrované, klientovi nezaručuje ještě to, že server (či jiný klient) se kterým navazuje spojení, je ten, se kterým opravdu spojení chce navázat. Aby toto bylo možné zajistit, vznikly elektronické certifikáty.

Certifikáty hrají v SSL protokolech důležitou roli, certifikační autorita, která za certifikáty stojí, představuje někoho, komu se dá věřit a komu věří obě komunikující strany.

Každý počítač už při instalaci operačního systému dostane seznam některých certifikačních autorit. Díky tomu je pak schopný komunikovat s velkým množstvím legitimních serverů, protože si při návštěvě každého z nich ověří, zda jejich certifikát vydala některá z těchto autorit, jejíž certifikáty má uložené ve svém úložišti.

Certifikáty obsahují bezpečnostní prvky proti falšování, takže každý účastník komunikace by měl být schopen automaticky ověřit, zda je certifikát pravý. Kromě toho v sobě nesou informaci, ke kterému serveru patří. Dále obsahují datum vydání a datum, do kterého bude certifikát platit – certifikáty je potřeba obnovovat, aby mohla být zachována jejich důvěryhodnost. A v neposlední řadě certifikáty obsahují také veřejný klíč vlastníka, který umožňuje zahájení šifrované komunikace.

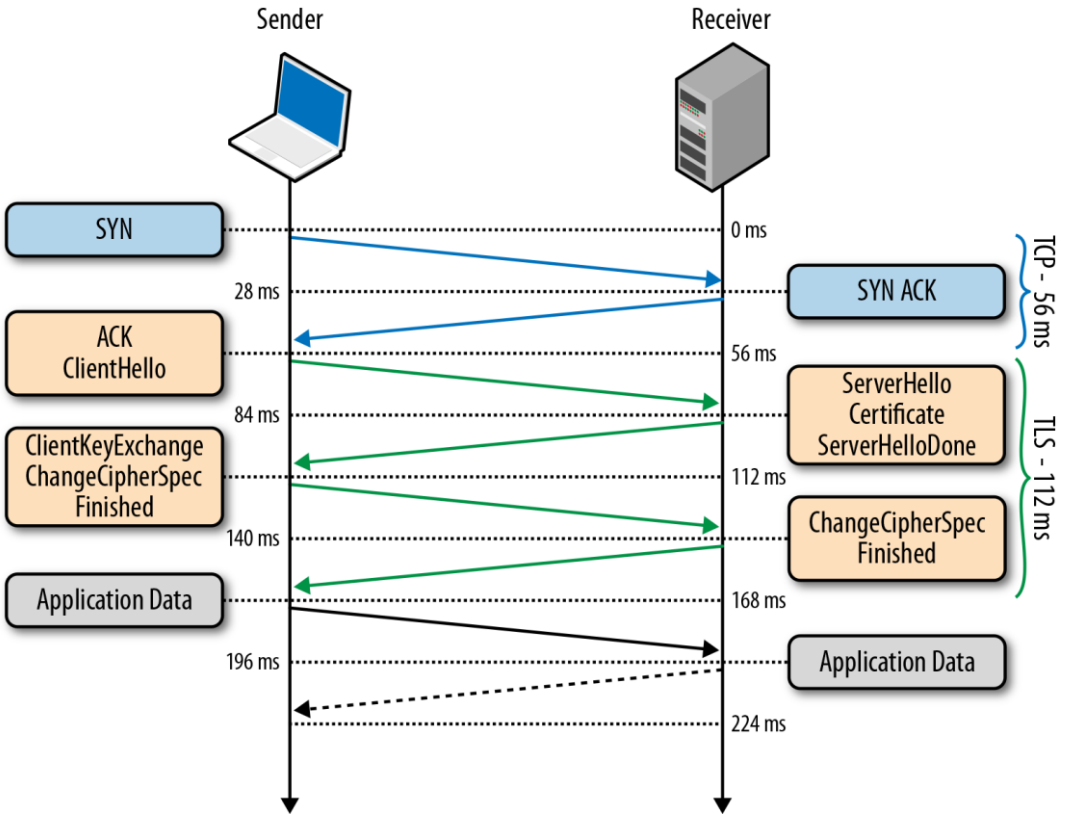
3.4.1 HTTPS

Jedná se o protokol sloužící k zabezpečené komunikaci skrze počítačové sítě. Slouží jako novější a bezpečnější náhrada protokolu HTTP, který je nezabezpečený, a kdokoliv tak může bez větších problémů prohlížet zachycenou komunikaci, číst všechny zprávy, včetně hesel a dalších citlivých údajů. V HTTPS je šifrovaná komunikace zajištěna pomocí SSL nebo TLS protokolu a jako výchozí využívá port 443 namísto portu 80.

Použití HTTPS je povinností každého provozovatele webových stránek, které od uživatele vyžadují jakékoliv citlivé informace. Od července 2018 již prohlížeč Chrome zobrazuje při načtení stránky využívající nezabezpečený HTTP varovnou hlášku „Vaše připojení není zabezpečené“. Princip zabezpečení komunikace funguje na základě asymetrické kryptografie, pomocí které se ověřuje identita webového serveru a popřípadě i klienta. Po ověření identity se následně pro samotnou komunikaci dohodne klíč pro symetrické šifrování, které je v porovnání s asymetrickým znatelně výkonnější. Zvolení klíče pro symetrickou šifru může proběhnout dvěma způsoby. Prvním z nich je, že klíč zvolí sám klient anebo se využije **Diffieho-Helmanova výměna klíče**.

SSL a TLS využívá infrastrukturu veřejného klíče a X.509 certifikáty, které zajištují autentizaci. Samozřejmostí pro úspěšné ověření klienta je důvěra v zaslaný certifikát, kterou zajištují zmíněné certifikační autority. Jejich certifikáty je možné najít v uložišti důvěryhodných certifikátů operačního systému.

Hlavní výhodou protokolu HTTPS je tedy šifrování přenášených dat, a tudíž integrita celého obsahu. Nevýhodou by mohl být pokles rychlosti komunikace u staršího hardware a také potřeba obměňování certifikátu.



Obrázek 3.2 - Navázání HTTPS spojení

Inicializační SSL/TLS *handshake* vyžaduje na začátku spojení dvakrát cestu tam a zpět, oproti spojení v porovnání s klasickým HTTP. Ve výsledku tedy navázání spojení a obdržení prvních dat trvá třikrát déle. Zároveň také dojde k mírnému zvýšení zatížení pásma, z důvodu zvýšení počtu bytů v záhlaví. Celkové zvýšení režie je okolo 6-7 %.

SSL a TLS ve většině případu dnes využívají 128, 192 nebo 256bitové klíče. Tyto klíče jsou pro dnešní použití bezpečné a není nutné se bát jejich prolomení. Pokud budou v budoucnosti využívány kvantové počítače, již tato ochrana nebude muset stačit (Shortův algoritmus by měl provádět faktorizaci v polynomiálním čase). Vývoj kvantových počítačů je opravdovým problémem, který se dodnes nepodařilo smysluplně vyřešit a není jisté, že se to vůbec někdy podaří. Avšak problém s ochranou proti prolomení pomocí dostatečně výkonného kvantového počítaje je z části aktuální již dnes, protože pokud by útočník zachytával komunikaci zabezpečenou HTTPS, mohl by ji pomocí kvantového počítače v budoucnu dešifrovat. Proto se již dnes *Google* a jiné projekty (např. *Open Quantum Safe*) soustředí na vyvíjení post kvantové kryptografie, která by zabezpečila dnešní komunikaci i do budoucna.

3.4.2 Ochrana před podvrženými certifikáty

V roce 2011 se začaly množit útoky na certifikační autority, které měly za následek vydání falešných, ale platných certifikátů. Pokud útok na certifikační autoritu byl úspěšný, mohl útočník vydat nový platný certifikát, a ten využít k *man-in-the-middle* útoku. Další možností zneužití jsou *phisingové* útoky, například podvržení stránek *Facebooku*. Pokud útočník vydá falešný certifikát, prohlížeč nic nezaznamená a bez jakékoliv bezpečnostní hlášky stránku načte. Ochrana před podvrženými certifikáty neboli c*ertificate pinning,* je vlastnost, která umožní serveru internetové služby (např. *Google*) specifikovat, které certifikáty (SSL/TLS) jsou správné. *SSL pinning* je přidaná vrstva bezpečností implementace na straně klienta, která dovolí důvěřovat pouze určitému SSL certifikátu během vytvoření HTTPS spojení.

Ve světe X.509 klienti ověřují platnost serverového certifikátu tím, že verifikují podpisy vydané certifikační autoritou. To znamená, že hlavní výhodou tohoto schématu je její bezestavovost. Server může změnit certifikát každých pár minut a tato verifikace bude stále fungovat. Podpora takto rychlé verifikace je teoreticky bezpečná, ale v praxi těžce implementovatelná, protože server mění certifikát v průměru jednou za rok.

Nyní po aplikaci *pinningu* dochází k tomu, že při první návštěvě stránky s aktivovaným HTTPS a *pinningem,* stránka odeslala skrytou zprávu do prohlížeče klienta, která oznamuje: „Pro navštívenou stránku se bude v následujících X dnech používat certifikát XXX“. Tím přikazuje prohlížeči, že v této době nemá akceptovat žádný jiný certifikát i pokud by byl vydaný certifikační autoritou. Pokud se toto stane, musí prohlížeč informovat server.

Existují dvě varianty *pinningu*, které se liší v tom, co je přednastaveno na klientovi a jak bude porovnán certifikát na serveru, který je obdržen během SSL/TLS „*handshake“*. Oba však mají stejné chování. Za prvé, klient je přednastaven tak, aby věděl, který certifikát má očekávat. A za druhé, pokud se certifikát serveru neshoduje s přednastaveným certifikátem, klient komunikaci přeruší.

Hard Certificate Pinning: V tomto typu má klient přednastavené přesné detaily o certifikátu serveru přímo v systému. Po obdržení certifikátu serveru klient kontroluje, zda se shoduje s přednastaveným. Pokud tomu tak není, nahlásí aplikace chybu a přeruší komunikaci.

CA Pinning: Nemá přednastavené certifikáty serveru, místo toho má nastaven limit pro počet certifikátů, které je připraven akceptovat pro autentifikaci na daném serveru.

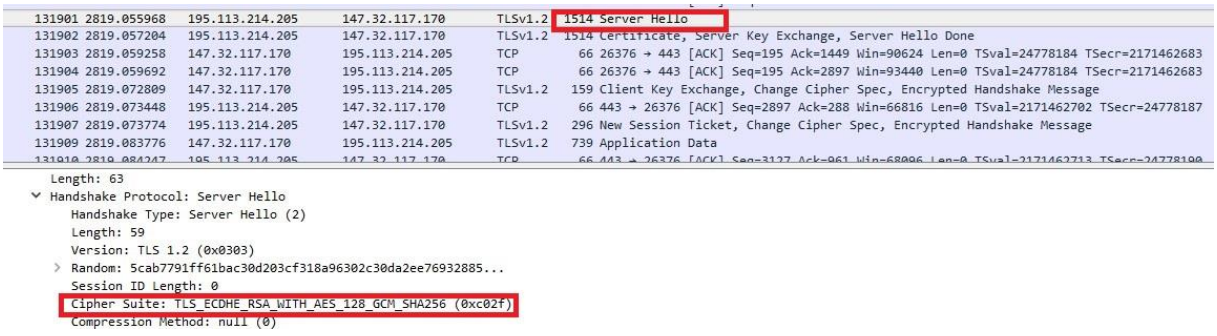
Tato ochrana nevyřeší kompletně slabinu certifikačních autorit a procesu podepisování certifikátů, ale minimalizuje příležitosti pro vytvoření falešného certifikátu. V současné době většina aplikací a prohlížečů *pinning* podporuje.

3.4.3 Dešifrování HTTPS

Vývojáři, testeři, forenzní analytici a další lidé, kteří vyvíjejí nebo zkoumají mobilní aplikace, musí analyzovat jejich datové toky, aby mohli odvodit chování aplikace v síti. A právě šifrování datového toku znesnadňuje analýzu chování aplikací a zjištění toho kdy porušuje soukromí uživatele odesíláním citlivých informací třetím stranám. Existují však metody, které, pokud ne úplně, tak alespoň částečně, dovolují šifrovaný provoz dešifrovat.

1) Použití privátního klíče a certifikátu serveru pro dešifrování

První z možností, jak dešifrovat provoz je v samotném *Wiresharku*. Aby to bylo možné, je potřeba privátní klíč certifikátu serveru, ke kterému se připojuje klient. Tento klíč stačí importovat do *Wiresharku* a šifrované pakety budou *Wiresharkem* dešifrovány. Tento způsob dešifrování však nefunguje vždy, pokud se k šifrování používá *Diffie-Helmanova* šifra. S použitím *Diffie-Helmanovy* šifry je klíč relace přenášen zašifrován dynamicky generovaným párem klíčů, místo použití veřejného klíče z certifikátu. *Wireshark* tedy nebude schopen tento provoz dešifrovat. Zdali je ke komunikaci využita šifra typu *Diffie-Helman*, je možné zjistit ze *Server Hello* zprávy.



Obrázek 3.3 – Příklad použití diffie-helmanovy šifry

Řešením je vypnutí používání *Diffie-Helmana*, toto je možné udělat jak na straně serveru, tak i na straně klienta, kde je to však jednodušší.

Aby bylo možné dešifrovat TLS je potřeba mít k dispozici serverový certifikát i privátní klíč. To v praxi znamená že se musí extrahovat klíč ze serveru se kterým *Android* komunikuje. To však není možné, protože klient není vlastníkem serveru a nemá k němu tudíž přístup. Tímto se tedy vylučuje možnost použít privátní klíč k dešifrování celého provozu.

Výhody

- Může být použito jak ze strany serveru, tak ze strany klienta
- Dešifruje veškeré zprávy v komunikaci

Nevýhody

- Musí být přístup k serveru
- Nepodporuje *Diffie-Helmanovu* šifru

2) Použití *pre-master secret* klíče

Jedna z nejjednodušších metod, jak dešifrovat SSL/TLS pakety ve *Wiresharku* je použití *pre-master secret* klíče. Důvodem, proč je tato metoda nejjednodušší, je že k jejímu použití není potřeba přístup k serveru, aby bylo možné dešifrovat SSL. Tento klíč je generován klientem a použit serverem k odvození *master* klíče, který šifruje provoz celé relace. Relací se v tomto případě myslí ustanovení zabezpečeného tunelu mezi dvěma komunikujícími uzly. Takto vytvořená relace v sobě ještě může obsahovat více spojení. Kryptografický standard je obvykle implementován skrze *Diffie-Helmanovu* metodu.

Pre-master secret klíč se nastavuje pomocí proměnné **SSLKEYLOGFILE**. Internetové prohlížeče tuto proměnou kontrolují po svém startu a pokud je vytvořena, tak do ní zapisují hodnoty použité při generování klíčů TLS relace. Poté je možné nakonfigurovat *Wireshark* tak, aby tyto hodnoty četl a pomocí nich dešifroval SSL/TLS pakety.

Úložiště klíčů v *Androidu* slouží jako prostředí pro uložení kryptografických klíčů, které zabraňuje extrahování klíčů ze zařízení, na kterém systém běží. Klíče, které jsou v úložišti, mohou být použity na kryptografické operace s klíčovým materiálem (převážně uživatelská data), který zůstane neexportovatelný. Úložiště také nabízí možnost vytvořit pravidla kdy a jak mohou být klíče použity. Jako například vyžadovat ověření uživatele pro použití klíče nebo omezit klíče, které mají být použity pouze v určitých kryptografických režimech.

Aby se zabránilo extrakci klíčů používá k tomu úložiště klíčů dvě bezpečnostní opatření.

- Klíčový materiál nikdy nevstoupí do procesu aplikace. Pokud je proces aplikace ohrožen, může být útočník schopen použít klíče aplikace, ale nemůže extrahovat klíčový materiál (například pro použití mimo zařízení *Android*).

- Klíče mohou být vázány na zabezpečený hardware (např. *Trusted Execution Environment* (TEE), *Secure Element* (SE)) zařízení *Android*. Pokud je tato funkce povolena pro klíč, jeho klíčový materiál není nikdy zobrazen mimo zabezpečený hardware. Pokud je operační systém *Android* ohrožen nebo útočník může číst interní úložiště zařízení, může být útočník schopen používat na svém zařízení *Android* klíče z úložiště, ale nesmí je extrahovat ze zařízení. Tato funkce je povolena pouze v případě, že zabezpečený hardware zařízení podporuje konkrétní kombinaci algoritmu klíče a režimů bloků.[17]

Výhody

- Nemusí být přístup k serveru
- Podporuje *Diffie-Helmanovu* šifru

Nevýhody

- Musí být přístup k serveru
- Tuto vlastnost nepodporují mobilní platformy s *Androidem*

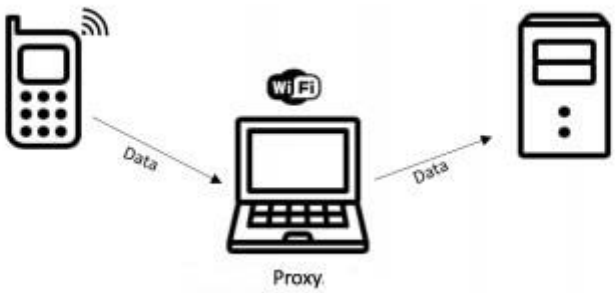
3) Použití proxy

Celý problém s rozšifrování SSL spojení spočívá v tom, že je šifrované *end-to-end* a přerušení tohoto spojení bez vlastnění privátního klíče serveru je bezpředmětné, ledaže by se nepoužil certifikát certifikační autority, ale certifikát který bude vytvořen pro účely záchytu komunikace.

Cílem je tedy přerušení SSL komunikace lokálně tak, aby byl získán přístup k nešifrovaným zprávám a možnost dále je předávat k cílovému serveru nově založeným SSL spojením. Ve výsledku bude tedy celá komunikační relace přerušena a vytvořeny dvě nové nezávislé relace. Tento způsob také otevírá dveře k dalšímu způsobu testování toho, jak se aplikace zachová, pokud ji bude simulováno určité chování serveru. Například chybné zaslání citlivých uživatelských dat. Další z výhod využití *proxy* je potencionální možnost přesměrování spojení přes mobilní datovou síť skrze vlastní *proxy*, díky vytvoření VPN spojení na hosta, kde běží *proxy*. [18]

Nastavení *proxy Android* podporuje a je možné jej nastavit pomocí IP adresy *proxy* serveru a komunikačního portu. Takto vytvořená *proxy* bude fungovat pouze pro HTTP, ale pro HTTPS bude hlásit chybu z důvodu chybějících certifikátů. Pro využití HTTPS je nutné použít vlastní certifikát, který umožní úpravu požadavků zašifrovaného provozu.

V tomto případě je využit *proxy* software, který se nainstaluje na zařízení, na němž bude prováděn záchyt. Existuje pro to několik aplikací. Například *Charles proxy,* který nabízí bezplatnou verzi, avšak s omezením pouze na třicet dní a každých třicet minut se musí aplikace restartovat. Jako alternativa se nabízí velmi oblíbený *Fiddler* který však nabízí omezené množství funkcionalit však není možné ukládat zachycenou komunikaci pro pozdější analýzu. Jako nejlepší alternativa se jeví *Burp Suite professional* která je hojně využívána mezi profesionální penetračními testery a nabízí mnoho funkcionalit.

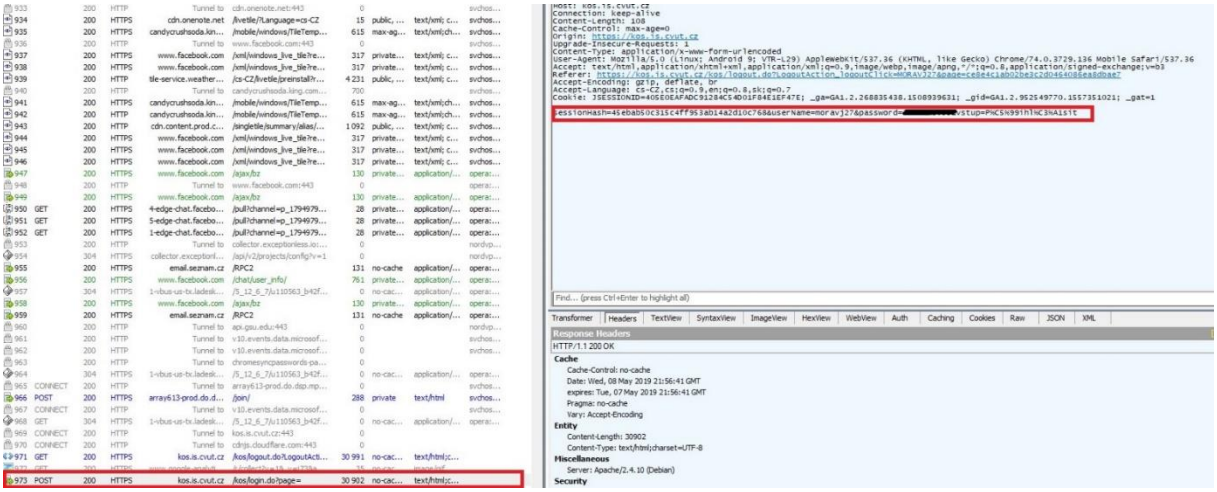


Obrázek 3.4 - Schéma záchytu s proxy serverem

Pro přesměrování provozu přes *proxy* je postup následující. *Burp Suite* vyžaduje, aby byl telefon připojen do stejné lokální sítě jako je záchytové zařízení. Následně pomocí příkazu zjistit IP adresu záchytového počítače a tu následně vložit do konfigurace nové *proxy Androidu*. Je nutné také zvolit port na kterém bude celá *proxy* fungovat, standardně se volí port 8080.

Pro vytvoření šifrovaného tunelu až k *proxy* softwaru je nutné otevřít internetový prohlížeč a navštívit stránku http://burp/. Na této stránce se nachází certifikát aplikace *Burp Suite*, který se stáhne do *Androidu*. Certifikát se však sám nenainstaluje, je potřeba jej nainstalovat manuálně z úložiště.

Pro inicializaci odposlechu *proxy* server zachytí *master key* během *handshake* SSL/TLS podepsaného vlastním certifikátem (protože *proxy* používá vlastní sadu veřejných a soukromých klíčů k dešifrování a opětovnému zašifrování výměny symetrických klíčů).



Obrázek 3.5 - Zobrazení přístupových údajů k zabezpečenému univerzitnímu serveru KOS

V základním nastavení je většina prohlížečů nastavena tak, že pokud zachytí nový certifikát, začne hlásit problém s potenciálním bezpečnostním rizikem. Stačí však tuto informaci potvrdit tím, že byla vyrozuměna a dále prohlížeč nebude vykazovat žádné problémy. Jinak je tomu však u aplikací.

Když se mobilní aplikace rozhodne inicializovat SSL/ TLS spojení, operační systém musí validovat použitý certifikát proto, aby ověřil, že certifikační řetězec je kompletní a kořenový certifikát byl vydán certifikační autoritou. Pokud certifikát nesplňuje validní řetězec důvěryhodnosti většina aplikací ukončí spojení, které je směrované přes potencionálně nebezpečný kanál. Níže uvedené techniky mají za společný cíl přesvědčit mobilní aplikaci, aby důvěřovala certifikátu poskytovaného *proxy* aplikací.

Nejjednodušším způsobem, jak se vyhnout chybám SSL, je mít platný a důvěryhodný certifikát. To je poměrně snadné, pokud je do zařízení možné nainstalovat nové důvěryhodné certifikační autority a důvěřuje certifikátu podepsanému konkrétní certifikační autoritou.

Android má dvě vestavěné certifikační úložiště, které uchovávají informace o tom, které certifikační autority jsou důvěryhodné operačním systémem – *systémové úložiště* (s předinstalovanými certifikačními autoritami) a *úložiště uživatelů* (držení uživatelsky nainstalovaných certifikačních autorit).

Android 7.0 a nižší – Ve výchozím nastavení využívá zabezpečené TLS spojení ze všech aplikací. Ty důvěřují předinstalovaným systémovým certifikačním autoritám a aplikace cílené na systém *Android* 6.0 (úroveň API 23) a nižší také ve výchozím nastavení důvěřují úložišti přidanému uživatelem. Pokud aplikace cílí na verzi *Android* 7.0 nebo nižší, je možné jednoduše přidat nově vytvořenou certifikační autoritu do úložiště certifikátů přidaných uživatelem.

Android 7.0 a vyšší – Pokud aplikace cílí na verze *Android* vyšší než 7.0, je nutné upravit kód aplikace a donutit ji k cílení na *Android* 7.0. Cílená úroveň rozhraní API je uvedena v atributu „*platformBuildVersionCode*“ prvku „*manifest*“ v souboru *AndroidManifest.xml.*

Pokud je certifikát úspěšně nainstalován do uživatelského úložiště certifikátů, aplikace cílí na verzi *Android* 7.0 a certifikát se zobrazuje jako platný, je stále možné, že aplikace bude ukončena s chybnými hláškami. Pravděpodobnou příčinou je to, že vývojáři podnikli další kroky k omezení sady certifikačních autorit důvěryhodných aplikací.

Existují dva způsoby, jak toto obejít.

- Instalace certifikátu *proxy* softwaru jako systémovou CA na zařízení. Jedná se o nejjednodušší řešení, které ale vyžaduje *root* zařízení. Praktická výhoda tohoto řešení je, že se nemusí nastavovat přístupový PIN na zařízení, protože certifikát bude brán jako systémový.

- Manuální úprava manifestu aplikace. Tato varianta se skládá ze tří částí. První z nich je rozbalení aplikace, ve druhé se přidá nový zdroj XML pro definování profilu zabezpečení sítě. Ve třetí části se upraví soubor *AndroidManifest.xml*.

  Tento soubor je umístěn v kořenovém adresáři souboru *APK* aplikace. Soubor manifestu popisuje strukturu aplikace, její součásti (aktivity, služby) a požadovaná oprávnění. Obsahuje také obecná *metadata* aplikací. Například ikonu aplikace, číslo verze a motiv. Soubor může obsahovat další informace, jako jsou kompatibilní rozhraní API (minimální, cílené a maximální verze sady SDK) a typ úložiště, do kterého lze nainstalovat (externí nebo interní).

  V poslední, čtvrté fázi se aplikace opět zabalí jako APK soubor. Jedná se o pracnější variantu, která však nevyžaduje *root* telefonu. [19]

Výhody

- Nemusí být přístup k serveru
- Podporuje *Diffie-Helmanovu* šifru
- Je možné využít i na *Androidu*
- Možnost úpravy dat během relace
- Přes *proxy* je možné směrovat i mobilní data

Nevýhody

- pokud aplikace využívá *certificate pinning*, nebude vůbec komunikovat se serverem

Samotné využití *proxy* funguje velice dobře pokud se nejedná o aplikace, které implementují *certificate pinning*, poté si už s dešifrováním nedokáže poradit.

3.4.4 Obejití Certificate pinning

Všechny přechozí možnosti se věnovaly přerušení SSL provozu pouze tehdy, pokud aplikace nevyužívala *certificate pinning*. Následující řádky se věnují, obejití při aktivovaném *certificate pinningu*.

Když aplikace, která využívá *certificate pinning* provádí HTTPS *handshake* skrze *proxy*, přezkoumá odpověď certifikátu a odmítne odeslat jakýkoli budoucí požadavek, pokud odhalí certifikát nepatřící certifikační autoritě (CA).

Neexistuje obecný postup, jak tento problém vyřešit a jak ho obejít. Pokud je při testování snaha zachytit síťový provoz všech aplikací které se na zařízení nachází a není možné i jedné konkrétní tuto ochranu obejít, je potřeba této aplikaci přiřadit certifikát CA, aby provoz skrze *proxy* procházel nepřerušovaně.

Jednou z prvních možností je řada automatizovaných nástrojů pro deaktivaci validací SSL / TLS. Tyto nástroje zavádějí především nástroje API SSL / TLS k potlačení metod v nativních knihovnách. [20]

1) Xposted

Framework Xposed poskytuje platformu pro vývoj a distribuci těchto automatizovaných nástrojů. *Xposed* pracuje na principech modulů, z nichž každý ovládá určitou funkcionalitu. Hlavní aplikace, ze které se ovládá se nazývá *Xposed Installer,* do kterého je nutné instalovat *framework* podle verze používaného *Androidu*.

Tato aplikace vykazuje často problematické chování s frameworkem které zapříčiňuje její nefunkčnost. Většinu problému však vyřeší nástroj *Xposed Installer Static BusyBox* který opravuje přístup Xposed k superuživateli androidu. Do této aplikace je možné doinstalovat několik nástrojů které se starají o obejití pinningu. [21]

SSL unpinning 2.0 jedná se o první z těchto nástrojů. Tento modul, který se stará o obejití *pinningu* certifikátu se poté dodatečně instaluje přímo do aplikace. Modul nabízí možnost zvolit si konkrétní aplikace, na kterých se aktivuje obcházení *pinningu*.

Tento nástroj fungoval poměrně efektivně, ovšem pouze pro aplikace, které ačkoliv využívaly *pinning*, běžely na starších verzích *Androidu*. *SSL unpinning 2.0* pracuje pouze s knihovnami JSSE a *Apache*. S knihovnami jako *Volley* nebo *LoopJ* neumí pracovat. Poslední vydaná verze tohoto nástroje pochází z roku 2016.

Od tohoto roku přestala fungovat podpora a vývoj dalších verzí a nově se přesunul vývoj k nástroji *Inspeckage – Android Package Inspector*, který pracuje s knihovnami *JSSE, Apache* a *okhttp3*. Ke všem těmto aplikacím je potřeba *rootnuté* zařízení. [22]

JustTrustMe, další nástroj pro *Xposted framework*, funguje na podobném principu jako SSL *unpinning 2.0*, byl však napsán jinými vývojáři.

2) Frida – FRIDA Univerzální *SSL Pinning Bypass Script*

Obvykle je k obejití *certificate pinningu* nutné pozměnit kód aplikace, a tak zasahovat do samotného procesu validace. Frida je ovšem aplikace, která tento problém řeší zcela jinak. Jedná se o výkonný soubor nástrojů – neslouží pouze k překonání *certificate pinningu*, ale má mnoho předností a je ideální pro testování a vyhodnocování nativních aplikací pro Android. Princip funkce spočívá v zachycení dat přijatých a odeslaných aplikacemi a následně vložení vlastních kódů.

Frida běží na operačním systému jako samostatný software zapouzdřený v dynamické knihovně, která je načtena cílovou aplikací za běhu, a následně upravuje chod aplikace. To umožňuje zasahovat do živých procesů, následně testovat, přidávat funkce nebo dokonce ladit aplikace.

Pro použití Frida, je potřeba rozbalit APK, vložit dynamickou knihovnu, upravit kód aplikace tak, aby byla dynamická knihovna první věc, která se bude volat při spuštění aplikace. Pak balíček APK znovu zabalit a nainstalovat. Tento typ obejití je omezen na mobilní zařízení s *rootem*, ale s pomocí *Frida* je nyní možné použít aplikaci pro *Android* a získat přístup k celé řadě funkcí *Frida* bez nutnosti *rootu*. [23]

4) Kill switch

SSL Kill switch je *blackbox* nástroj, který specificky upravuje nízko úrovňové služby SSL / TLS *Androidu* s cílem zakázat ověření kořenového certifikátu systému. Navíc také obchází i jiné druhy certifikátů, jako třeba *certificate pinning*. Tato metoda je však více problematická pro použití na jakémkoli zařízení, protože musí být nainstalována na zařízení s *rootem*, protože provádí úpravy kódu na velice nízké úrovni.

Zatímco *SSL Kill Switch* by mohl být použit k obcházení SSL / TLS validací, takovému typu útoku by však mohlo být snadno zabráněno aplikacemi, které odmítnou iniciovat SSL / TLS spojení, pokud je na komunikačním zařízení detekována přítomnost nástroje.

Zatímco automatizované nástroje jsou užitečné pro usnadnění analýzy mobilních aplikací, tyto nástroje samy o sobě nejsou dostatečné zejména v současné situaci, kdy se vývojáři neustále snaží posilovat existující bezpečnostní mechanismy. Proto je nutné často přistupovat k základnímu přístupu obcházení takovýchto mechanismů a tím jsou manuální úpravy aplikací.

5) Manuální reverzní inženýrství

Jedná se o nejspolehlivější řešení pro obcházení *SSL Pinningu*, který však vyžaduje jisté zkušenosti s programováním a reverzním inženýrstvím. Nakonec je možné, že se vývojáři rozhodli implementovat své vlastní knihovny SSL namísto spoléhání se na systémové knihovny pro ověření certifikátu SSL. V tomto případě je manuální přístup přímo nutný.

1. Pochopit implementaci *SSL Pinningu*. Pro jeho implementaci se využívají služby různých síťových knihoven, jako je *OkHttp*, *Volley*, *Retrofit* atd.

2. Extrahovat APK a převést *Smali* zpět na *Javu*, aby bylo možné hledat kód zodpovědný za zpracování ověření certifikátu. Pro zpracování *javy* je možné využít reverzní software jako třeba JD-GUI.

Smali/*Baksmali* je *assemblér*/*disassemblér* pro *dex* formát používaný implementací Java VM v *Androidu*. Jména "*Smali*" a "*Baksmali*" jsou islandské ekvivalenty "*assembler*" a "*disassembler*". Jakmile je identifikován kód zodpovědný za validaci certifikátu, je možné vybrat, zda ho úplně odstranit, nebo

zavést požadovanou funkci pomocí *Frida*. Aby se předešlo opětovnému sestavení celé aplikace, je obvykle efektivnější spojit funkce odpovědné za ověřování certifikátů.

Když je odpovědná metoda analyzována, identifikována a upravena, může se použít aplikace *APKTool* pro znovu sestavení aplikace a následně může být nainstalována. [23]

Změna pole společného názvu certifikátu (CM pole)

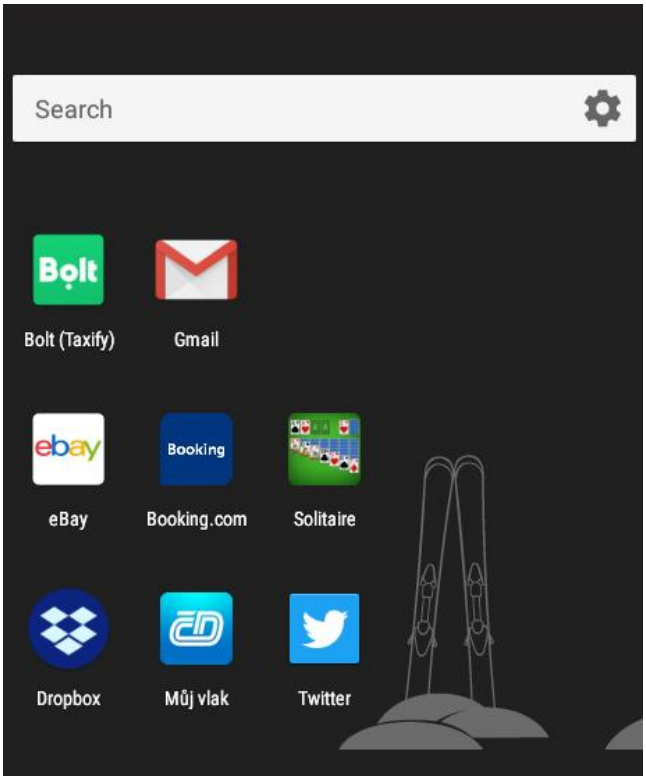
Proxy dynamicky generuje certifikát podepsaný certifikační autoritou, pokud má mobilní aplikace v úmyslu připojit se k určitému serveru. Proxy však použije certifikát CM, který je nutné nastavit v *proxy*. Tato metoda je užitečná, když *proxy* není schopno identifikovat požadavek před vyjednáváním SSL / TLS (např. Pokud klient neposílá požadavek *CONNECT*, ale ověří, zda pole CM certifikátu podepsaného certifikační autoritou má správné a očekávané parametry – název serveru). Tato metoda má výhodu v tom, že nevyžaduje *root* zařízení.

Obecný Postup manuálního obejití *certificate pinningu*

Nalezení správné metody obejití *pinningu* je obvykle obtížné a může trvat poměrně dlouho v závislosti na úrovni implementace. Protože vývojáři standartně využívají existující knihovny, je dobré začít tím, že se vyhledají řetězce a licenčních soubory, které identifikují použitou knihovnu. Jakmile je knihovna identifikována, je možné prohledat zdrojový kód a vyhledejte metody, které jsou vhodné pro obejití *pinningu*.

4. Provedení záchytu provozu

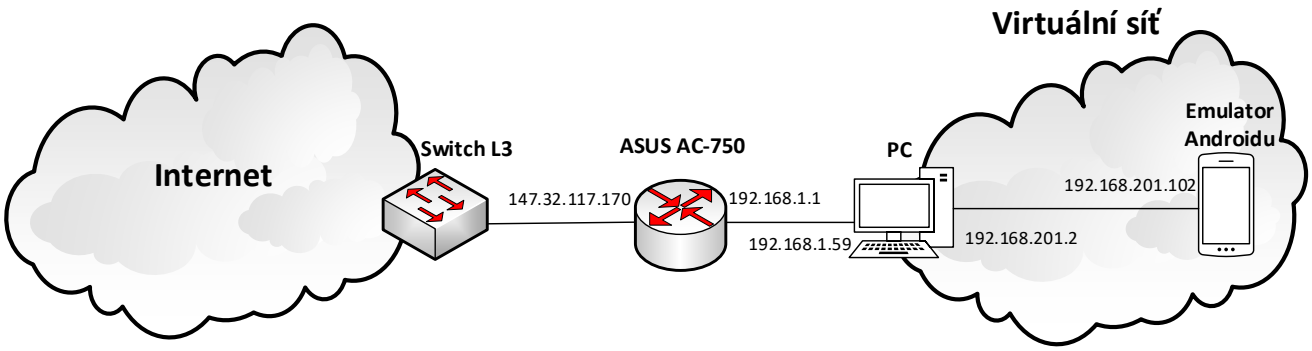
Pro analýzu bylo zvoleno osm aplikací, kterými jsou *Bolt (Taxify), Gmail, Ebay, Booking, Solitare, Dropbox, Můj vlak, Twitter*. Tyto aplikace byli vybrány, protože se jedná o běžné aplikace, které se nacházejí na mnoha telefonech a které většina uživatelů dnes a denně využívá. Některé z těchto aplikací, jako třeba *Gmail, Booking* nebo *Twitter* bývají často i předinstalované na mobilních telefonech a intenzivně pracují s uživatelskými daty.



Obrázek 4.1 - Seznam testovaných aplikací

U každé z těchto aplikací byla vyzkoušena odolnost proti obejití šifrování provozu s využitím *man-in-the-middle* útoku. V první řadě byla vyzkoušeno, jestli aplikace budou komunikovat se serverem při nahrazení původního certifikátu certifikátem *proxy* serveru. A v druhé řadu jsou testovány metody obejití *certificate pinningu* u aplikací, které tuto bezpeční metodu podporují.

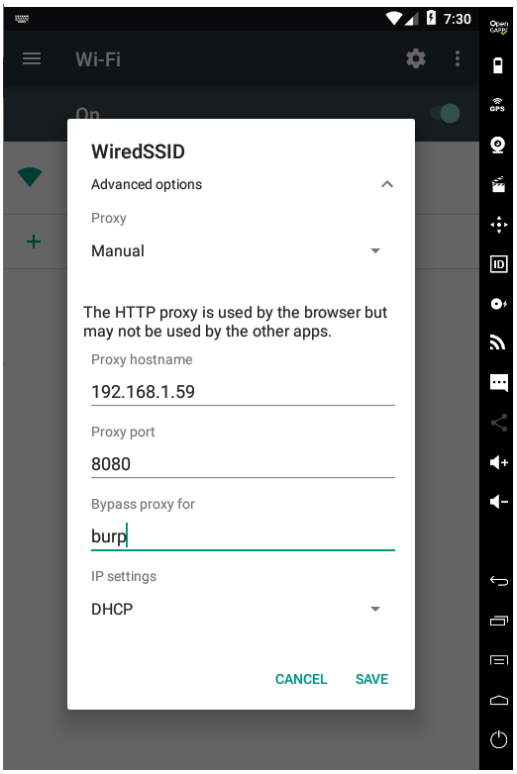
Záchyt síťového provozu je uskutečněn na tomto zapojení. Operační systém *Android* běží v emulátoru na počítači, který je připojen přes router *ASUS AC-750* do Internetu.



Obrázek 4.2 - Schéma síťového zapojení

4.1 Záchyt HTTP/HTTPS provozu

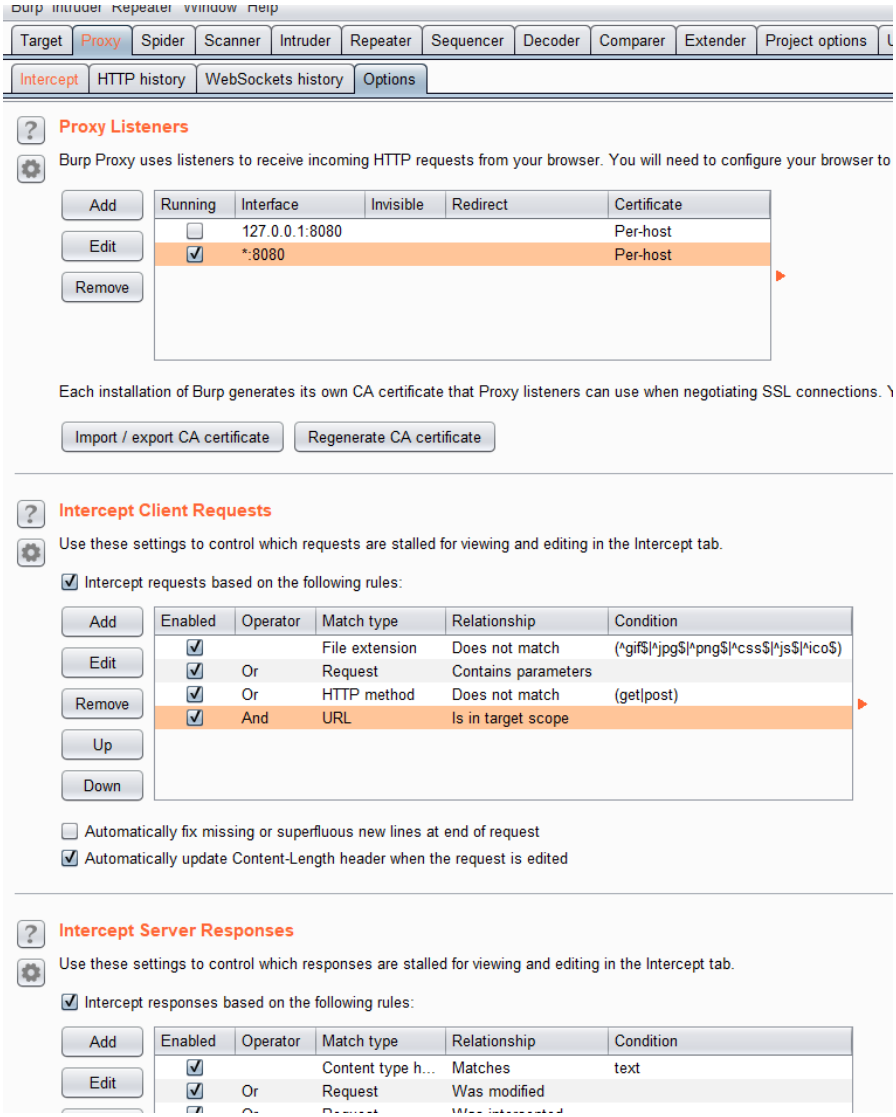
Pro testovací účely byla zvolena emulace *Androidu* 7.1 na telefonu *Samsung 6S*. V této verzi *Androidu* již není možné využívat osobní certifikáty pro komunikaci, a proto je nutné buď modifikovat aplikaci tak, aby přijmula i osobní certifikát, anebo přidat certifikát *proxy* softwaru mezi důvěryhodné. K testování je použit *Android*, ve kterém je nainstalován *root*, proto se využije možnost druhá. V Androidu byla nastavena *proxy* tak, aby byl veškerý provoz směřován přes port 8080 na IP adresu 192.168.1.59, která přísluší hostujícímu počítači.



Obrázek 4.3 - Nastavení proxy v Androidu

K testování byl použit *proxy* software *Burp Suite Profesional*, protože se podařilo získat jeho profesionální verzi, která nabízí mnohá pokročilá nastavení přerušení provozu. Licence *Profesional* byla získána na zkušební dobu jednoho měsíce pouze pro akademické účely. Verze *Comunity* která je zdarma nenabízí možnost ukládat zachycenou komunikaci pro pozdější analýzu. *Burp Suite* je nástroj pro penetrační testování aplikací. Tento nástroj funguje jako *proxy* server mezi serverem a cílovou aplikací a pracuje na aplikační vrstvě (OSI-7). Je také označován jako nástroj *man-in-the-middle*, který pracuje s protokolem HTTP/HTTPS.

Pro nastavení *proxy* je potřeba aktivovat přerušení provozu jak klientských žádostí, tak odpovědí serveru. *Proxy* byla nastavena pro všechny IP adresy na portu 8080, který byl již aktivován v *Androidu*.[24]



Obrázek 4.4 - Nastavení Burp Suite

Certifikát od *Burp Suite* se dá vyexportovat přímo z aplikace a poté přesunout do interního úložiště *Androidu*. Ve výchozím stavu se však certifikát exportuje ve formátu *„der“*, který ale není podporován. Proto je nutné změnit formát certifikátu na *„pem“* a dále změnit název tak, aby byl rovný hodnotě hashovací funkce „`subject_hash_old`“ a na konci `0`. K této konverzi byla využita knihovna *OpenSSL*.

```
root@kali:~/Downloads# openssl x509 -inform DER -in burp.der -out burp.pem
root@kali:~/Downloads# openssl x509 -inform PEM -subject_hash_old -in burp.pem |head -1
9a5ba575
root@kali:~/Downloads# mv burp.pem 9a5ba575.0
```

Obrázek 4.5 - Konverze certifikátu

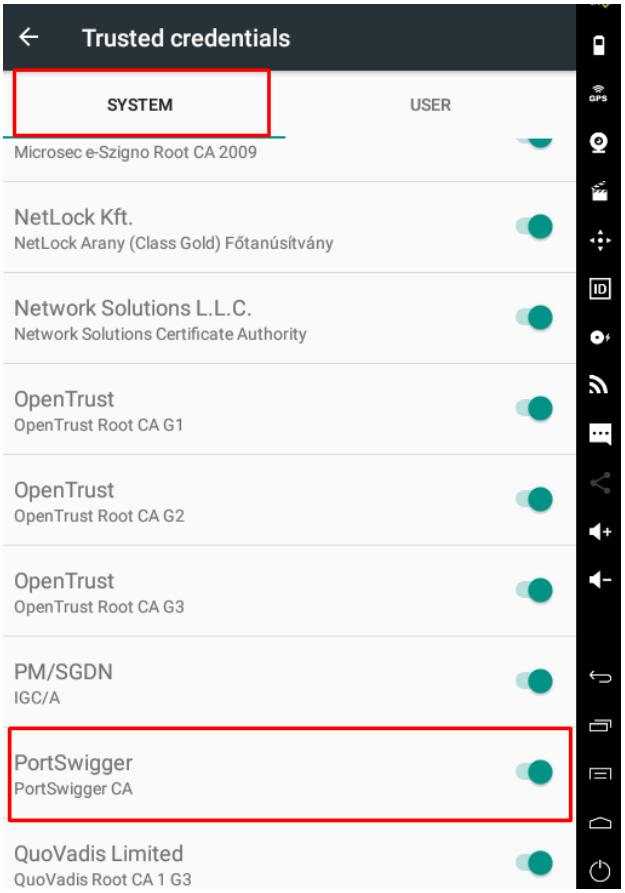
Certifikát musí být přidán do systémového úložiště, které je umístěno v adresáři */system/etc /security /cacerts* a obsahuje soubor pro každý nainstalovaný kořenový certifikát.

Pro zkopírování certifikátu je možné využít *Android Debug Bridge* (adb), který je nástrojem příkazového řádku, umožňujícím komunikovat s emulátorem nebo připojeným zařízením *Android*. Certifikát se překopíruje do systémového úložiště, a nakonec se zařízení musí restartovat, aby se projevily změny.

```
C:\Program Files\Genymobile\Genymotion\tools>adb root

C:\Program Files\Genymobile\Genymotion\tools>adb remount
remount succeeded

C:\Program Files\Genymobile\Genymotion\tools>adb push 9a5ba575.0 /sdcad/
adb: error: failed to copy '9a5ba575.0' to '/sdcad/': remote couldn't create file: Is a directory
9a5ba575.0: 0 files pushed. 0.4 MB/s (1375 bytes in 0.003s)

C:\Program Files\Genymobile\Genymotion\tools>adb push 9a5ba575.0 /sdcard/
9a5ba575.0: 1 file pushed. 0.3 MB/s (1375 bytes in 0.004s)

C:\Program Files\Genymobile\Genymotion\tools>adb shell
vbox86p:/ # mv /sdcard/9a5ba575.0 /system/etc/security/cacerts
vbox86p:/ # chmod 644 /system/etc/security/cacerts/9a5ba575.0
vbox86p:/ # reboot

C:\Program Files\Genymobile\Genymotion\tools>
```

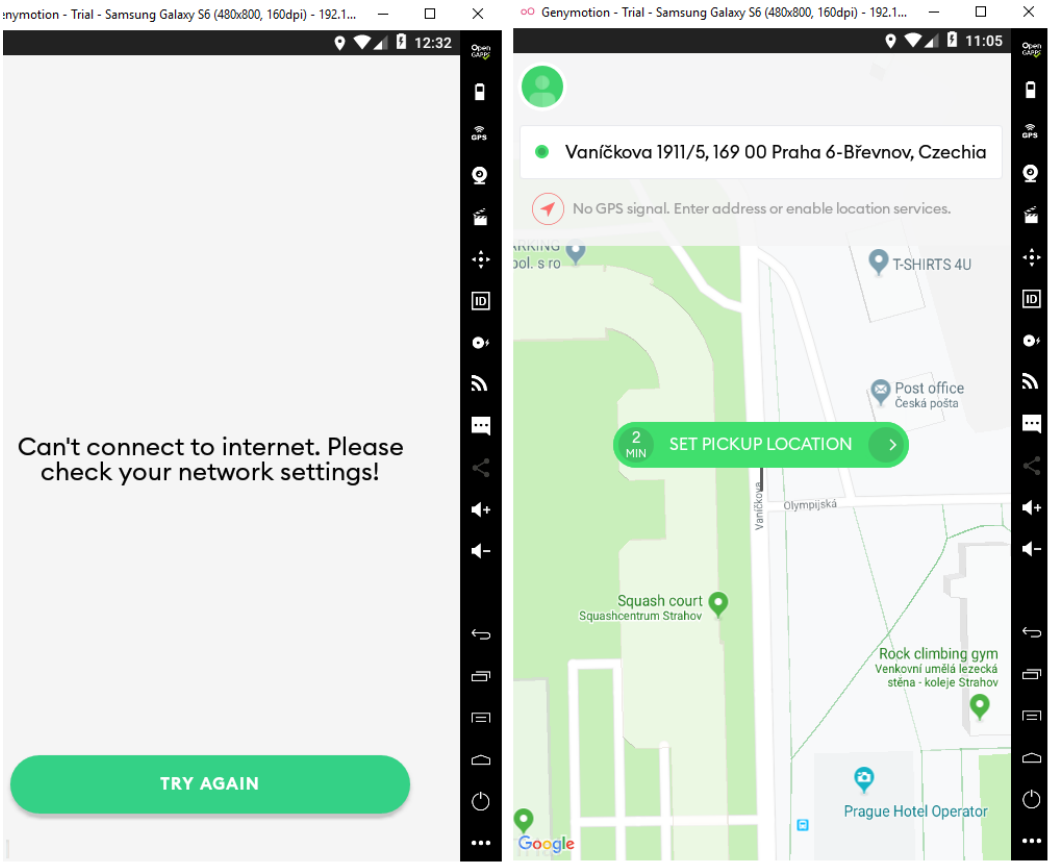
Obrázek 4.6 - Uložení certifikátu mezi důvěryhodné

Na dalším obrázku (4.6) je vidět, že certifikát patřící *PortSwigger* (vývojářská firma *Burp Suite*) je již mezi důvěryhodnými.



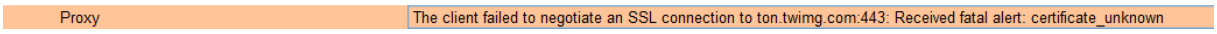
Obrázek 4.7 - Burp Suite certifikát mezi důvěryhodnými certifikáty

Na levé straně obrázku (4.7) je vidět, jak se aplikace *Bolt* chovala před přidáním certifikátu *proxy* serveru mezi důvěryhodné. Aplikace se odmítá připojit k internetu z důvodu nezabezpečeného spojení. na pravé straně je po změně důvěryhodnosti certifikátu vidět, že již aplikace zcela normálně komunikuje.



Obrázek 4.8 - Fungující aplikace *Bolt*

Naproti tomu aplikace *Twitter* zcela odmítá nadále komunikovat, protože využívá *certificate pinning*, jak potvrzuje *Burp Suite* s chybovou hláškou neznámého certifikátu.

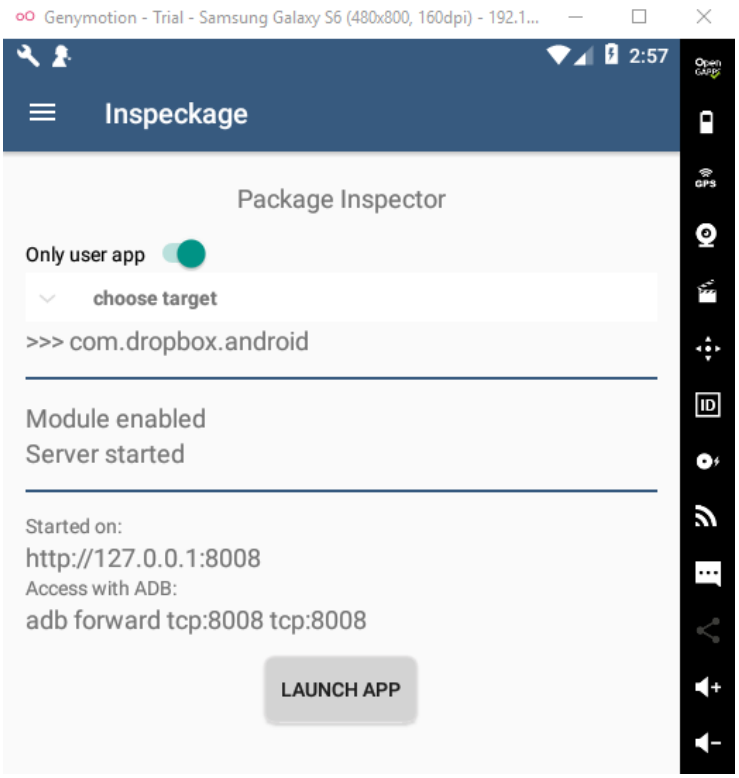


Obrázek 4.9 - Chyba certifikátu *Twitter*

4.1.1 Certificate pinning

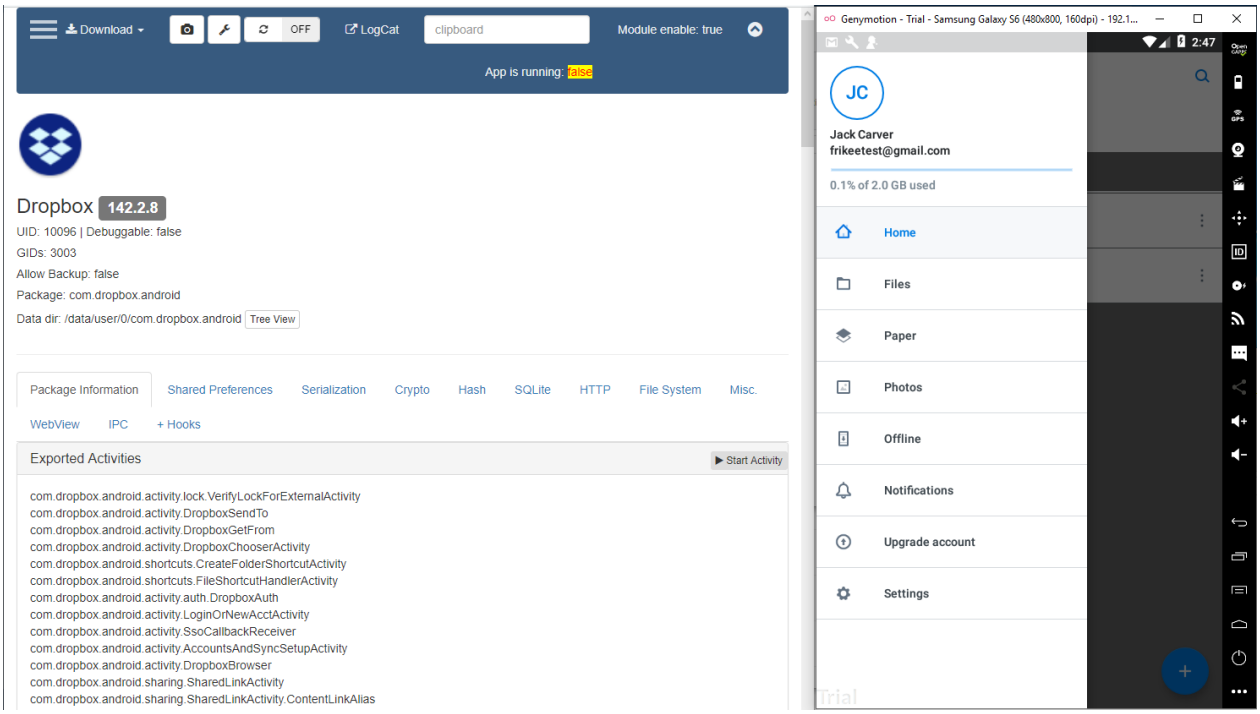
Inspeckage

Pro obejití *certificate pinningu*, kterou zjevně podporuje aplikace *Twitter* byl vyzkoušen automatizovaný program *Inpeckage*. *Inspeckage* je nástroj vyvinutý pro dynamickou analýzu aplikací *Android*. Obsahuje mnohé analytické nástroje pro vývojáře aplikací a mimo to i funkce na obejití *SSL pininngu* ve třech knihovnách, a to konkrétně JSSE, *Apache* a *okhttp3*. Do *Androidu* byl nainstalován jako modul do *frameworku Xposed* a jednou z jeho výhod je, že se dá ovládat přes ADB rozhraní na portu 8008, přímo v prohlížeči hostujícího počítače.



Obrázek 4.10 - Nastavení Inspeckage

Inspackage byl vyzkoušen na obejití *SSL pinningu* v aplikaci *Twitter,* toto však nebylo úspěšné a aplikace při pokusu o připojení k síti náhle spadla. *Inspeckage* byl však úspěšnější u další testované aplikace, a to konkrétně u aplikace *Dropbox*, kde se komletně podařilo obejít *SSL pinning* a tak aplikaci úspěšně spustit. Další aplikace, u které se úspěšně podařilo kontrolu obejít je aplikace *Můj vlak* od Českých drah.



Obrázek 4.11 - Překonání *certificate pinningu* v aplikaci *Dropbox*

Frida

Dosavadními přístupy se nepodařilo obejít šifrování aplikace *Twitter* tak, aby bylo možné provést analýzu aplikace. „Jednu z nejefektivnějších metod“ obejití *certificate pinningu* nabízí aplikace *Frida.* Cílem tohoto procesu je injektovat do aplikace *Java script* (příloha) [25], který poskytuje funkci na obejití *SSL pinningu* pro široké spektrum aplikací. Tento nástroj byl vyzkoušen, na aplikaci *Twitter.*

Pro spuštění aplikace je nejprve nutné stáhnout z *Githubu Frida* server [26], pro *Android* 86x a pomocí příkazu *„adb push“* zkopírovat do dočasných souborů. Aby bylo možné server spustit a mohl běžet na pozadí, musí se upravit pravidla pro čtení a zápis.

```
C:\Program Files\Genymobile\Genymotion\tools>adb push frida /data/local/tmp
frida: 1 file pushed. 39.2 MB/s (26598196 bytes in 0.647s)

C:\Program Files\Genymobile\Genymotion\tools> ./adb shell
'.' is not recognized as an internal or external command,
operable program or batch file.

C:\Program Files\Genymobile\Genymotion\tools>adb shell
vbox86p:/ # su
vbox86p:/ # ls /data/local/tmp/frida
/data/local/tmp/frida
vbox86p:/ # ls -l /data/local/tmp/frida
-rw-rw-rw- 1 root root 26598196 2019-05-18 19:04 /data/local/tmp/frida
vbox86p:/ # chmod 777 /data/local/tmp/frida
vbox86p:/ #  /data/local/tmp/frida &
[1] 3766
vbox86p:/ #
```

Obrázek 4.12 - Instalace *Frida* server

Dalším krokem je instalace *Frida* nástrojů pomocí příkazu *„pip install frida-tools“*. Jedná se o knihovny napsané v *Pythonu* a aby bylo možné je takto nainstalovat, musí se do hostujícího počítače stáhnout a nainstalovat *Python* knihovny, u kterých se musí dopsat cesta do *proměnných prostředí* tak, aby Windows věděl, odkud má spouštět příkazy v příkazové řádce.

Certifikát *Burp Suit* je tomto případě použit ve formátu *„crt“* a je pomocí příkazu: `adb push cert-der.crt` uložen do dočasného úložiště */data/local/tmp.* Následně je nastavena *proxy* stejně, jako v předchozím případě, kdy se ukládal certifikát mezi důvěryhodné.

Následujícím krokem je spuštění *Frida* serveru, který musím být aktivní po celou dobu, kdy aplikace běží.

```
C:\Program Files\Genymobile\Genymotion\tools>adb shell /data/local/tmp/frida &
```

Obrázek 4.13 - Spuštění *Frida* serveru

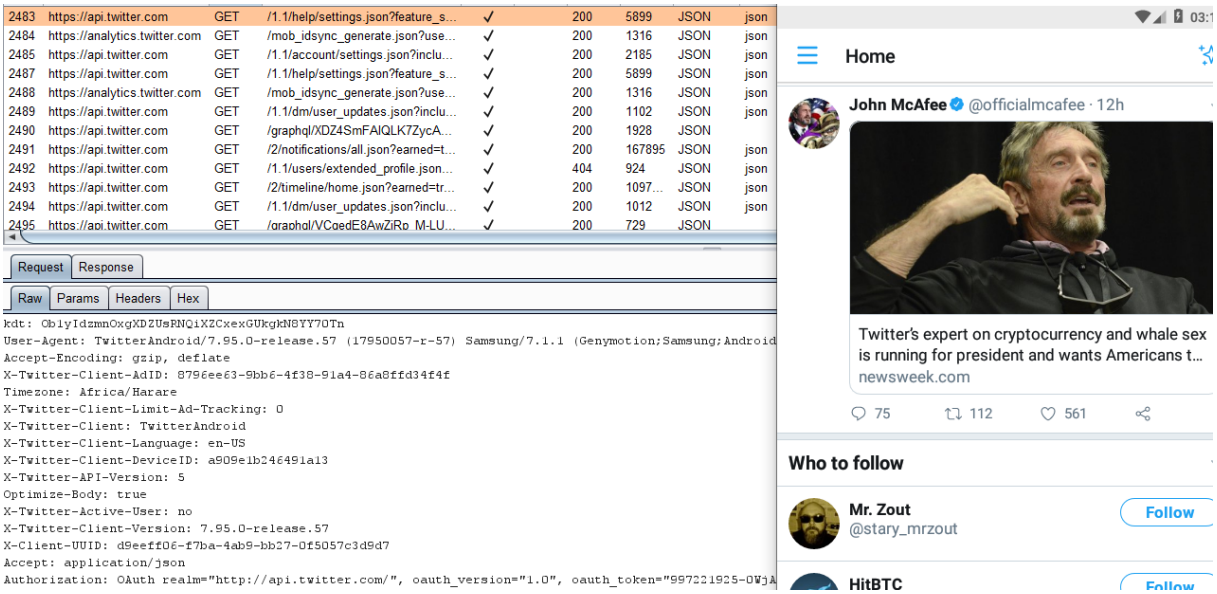
Pro injektování požadované aplikace se využívá název procesu, pod kterým v *Androidu* běží. K zjištění jeho názvu slouží příkaz „`frida-ps`“. Pokud je zařízení připojené přes USB, což emulátor je přidává se ještě parametr *-U*. Pokud by byl použit skutečný telefon, připojený přes wifi, použil by se parametr *-R*. Pro spuštění scriptu slouží příkaz „*`frida -U -f název procesu -l app script.js --no-pause`*“.

```
Thank you for using Frida!

C:\Program Files\Genymobile\Genymotion\tools>frida -U -f com.twitter.android -l script1.js --no-pause
     ____
    / _  |   Frida 12.5.6 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at http://www.frida.re/docs/home/
Spawned `com.twitter.android`. Resuming main thread!
[Genymotion Samsung::com.twitter.android]->
[.] Cert Pinning Bypass/Re-Pinning
[+] Loading our CA...
[o] Our CA Info: CN=PortSwigger CA, OU=PortSwigger CA, O=PortSwigger, L=PortSwigger, ST=PortSwigger, C=PortSwigger
[+] Creating a KeyStore for our CA...
[+] Creating a TrustManager that trusts the CA in our KeyStore...
[+] Our TrustManager is ready...
[+] Hijacking SSLContext methods now...
[-] Waiting for the app to invoke SSLContext.init()...
[o] App invoked javax.net.ssl.SSLContext.init...
[+] SSLContext initialized with our custom TrustManager!
[o] App invoked javax.net.ssl.SSLContext.init...
[+] SSLContext initialized with our custom TrustManager!
[o] App invoked javax.net.ssl.SSLContext.init...
[+] SSLContext initialized with our custom TrustManager!
[o] App invoked javax.net.ssl.SSLContext.init...
[+] SSLContext initialized with our custom TrustManager!
[o] App invoked javax.net.ssl.SSLContext.init...
[+] SSLContext initialized with our custom TrustManager!
[o] App invoked javax.net.ssl.SSLContext.init...
[+] SSLContext initialized with our custom TrustManager!
[o] App invoked javax.net.ssl.SSLContext.init...
[+] SSLContext initialized with our custom TrustManager!
[o] App invoked javax.net.ssl.SSLContext.init...
[+] SSLContext initialized with our custom TrustManager!
```

Obrázek 4.14 - *Frida* spuštění scriptu

Použitý *script* funguje tak, že nahraje *Burp Suite* certifikát na zařízení a dále si vytvoří vlastní *KeyStore* obsahující důvěryhodné certifikáty. Nakonec vytvoří *TrustManager*, který důvěřuje certifikačním autoritám ve vytvořeném *KeyStore.*

Frida s použitým skriptem v tomto případě zafungovala a obešla ověřování certifikátu, které *Twitter* prováděl. Následně došlo k přihlášení a ke stažení obsahu, který *Burp Suite* zaznamenal.

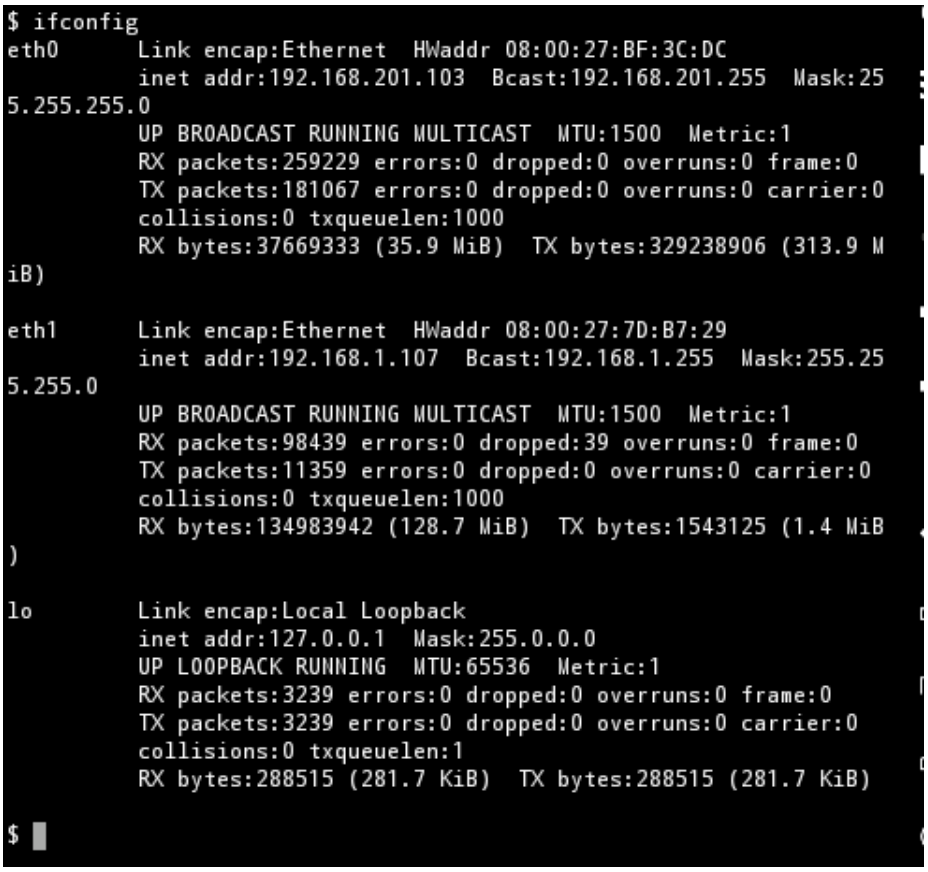


Obrázek 4.15 - Spuštění Twitteru

4.2 Záchyt ne-HTTP/HTTPS provozu

Virtuální stroje *Genymotion* mají v základní konfiguraci *VirtualBoxu* dva síťové adaptéry. Zvolení adaptéru závisí na tom, co se od záchytu očekává. Některé podrobnosti (názvy adaptérů atd.) závisí také na tom, jaký operační systém je použit. Tyto koncepty jsou však identické.

Adaptér 1, obvykle *vboxnet0*, je hostitelský adaptér používaný k podpoře lokálních interakcí mezi hostitelem a VM. V tomto případě je IP adresa VM získaná z vestavěného DHCP serveru *VirtualBoxu* 192.168.201.102 na rozhraní *eth0*. Tento adaptér se používá pro spojení *Android Debug Bridge* (ADB) a vývojových nástrojů k modulu VM, to je, jak Eclipse ADT a Android Studio jsou schopny komunikovat s VM pro instalaci APKs, atd. Pro vytvoření adb shell připojení se využívá také tato síť.



Obrázek 4.16 - Aktivní rozhraní Androidu

Adaptér 2 je nastaven v režimu NAT na síťové připojení hostitele. Toto rozhraní umožňuje přístup k internetu pro VM. Aby bylo možné zjistit, jak emulátor komunikuje s místní sítí nebo internetem, vytvoří se záchyt proti hlavnímu síťovému adaptéru hostitele. Bohužel, když je VM režimu NAT, bude mít VM mít stejnou IP adresu jako hostitel, což ztěžuje filtrování.

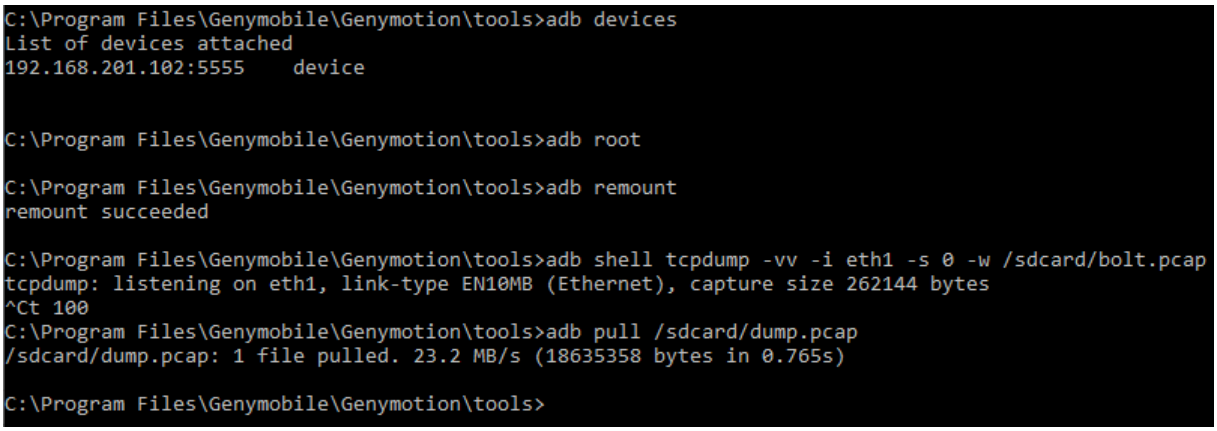
Aby toto nastavení nezpůsobovalo problémy stačí změnit adaptér 2 na "*Bridged Adapter*" a vybrat aktuální aktivní adaptér hostitele v konfiguraci sítě VM. Pokud LAN síť používá protokol DHCP umožní to serveru VM získat vlastní adresu IP. IP adresa pro komunikaci s internetem je přiřazena rozhraní eth1, která je v tom případě 192.168.1.107.

TCPdump

Pro vytvoření záchytu je použit program *TCP dump* který byl stažen do hostujícího počítače a do Androidu nainstalován tímto způsobem

```
adb push ~/Downloads/tcpdump /sdcard/
adb shell
su root
mv /sdcard/tcpdump /data/local/
cd /data/local/
chmod +x tcpdump
```

Záchyt komunikace byl proveden pomocí *TCPdump* programu na portu *eth1* a tento *„pcap“* soubor přenesen do hostujícího počítače na analýzu pomocí *Wiresharku*. [27]



Obrázek 4.17 - Vytvoření záchytu pomocí TCPdump

4.3 Výsledek záchytu

U každé z aplikací se podařilo obejít šifrování síťové provozu různou metodou. U aplikací *Bolt, Gmail, Ebay, Booking* a *Soliter*, není implementován *certificate pinning*. Proto u nich zafungovalo přidání certifikátu *Burp Suite* mezi důvěryhodné. U aplikací *Dropbox a Můj vlak* je implementován *certificate pinning* proto byl nástroj *Inspeckage* se kterým se zabezpečení podařilo obejít. U poslední testované aplikace je taktéž implementován *certificate pinning*, avšak nástroj *Inspeckage* v tomto případě nebyl úspěšný. Prolomení zabezpečení se podařilo až s nástrojem *Frida* a využitím externího *scriptu*.

Aplikace	Využití *SSL pinngu*	Způsob obejití šifrování
Bolt (Taxify)	ne	certifikát mezi důvěryhodnými
Gmail	ne	certifikát mezi důvěryhodnými
Ebay	ne	certifikát mezi důvěryhodnými
Booking	ne	certifikát mezi důvěryhodnými
Solitare	ne	certifikát mezi důvěryhodnými
Dropbox	ano	Inspeckage
Můj vlak	ano	Inspeckage
Twitter	ano	Frida

Obrázek 4.18 - Výsledek obejití šifrování

5. Analýza komunikace

Než je vůbec možné přistoupit k samotné analýze komunikace, vyvstává otázka, co vlastně je uživatelsky nevyžádaný provoz? Na nevyžádaný provoz se lze dívat ze dvou možných pohledů. Tím prvním je pohled, který se týká konzumace dat. Zvlášť na mobilních platformách uživatele zajímá, která aplikace, jenž se nachází na jeho mobilním zařízení, nejvíce spotřebovává data bez toho, aniž by tato spotřeba byla pro uživatele relevantní, protože se nejedná o uživatelská data, ale o data systémová. Tato tématika je důležitá hlavně při používání mobilních dat, kde je ještě dnes zpravidla omezená celková spotřeba dat ze strany operátora. Avšak z kontextu této práce je důležitý pohled druhý, a tím je problematika úniků citlivých uživatelských dat a komunikace, kterou zahrnuje nebezpečný software jako například malware.

Při analýze citlivých uživatelských dat je důležité si definovat, co mezi tato data patří. Obecně mezi tyto údaje jednoznačně patří osobní informace, které je možné dle [28], definovat jako takové informace, které se týkají určité fyzické osoby, a na základě nich lze konkrétního člověka jednoznačně identifikovat. Zvláštní skupinou jsou pak takzvané citlivé údaje (např. zdravotní stav, adresa bydliště).

Google, který spravuje market s aplikacemi Google Play přímo nařizuje vývojářům, jak konkrétně zacházet s údaji uživatele, a to tímto způsobem: „Musíte být transparentní ohledně toho, jak zacházíte s údaji o uživateli (např. s informacemi, které jste o něm a jeho zařízení shromáždili). Uživatele musíte o shromažďování, používání a sdílení dat informovat a data smíte používat jen k účelům, o kterých jste uživatele informovali a získali jeho souhlas.“ Google také zařazuje mezi osobní a citlivé údaje (*personally identifiable information – PII*) mimo jiné také údaje umožňující zjištění totožnosti, finanční a platební údaje, ověřovací údaje, telefonní seznam, údaje související s kontakty, zprávami SMS a hovory, data z mikrofonu a fotoaparátu, či citlivé údaje o zařízení a jeho využití. [29]

Analýza úniků citlivých dat z v datovém provozu je jedna z nejvíce kritických oblastí pro ochranu osobních údajů. Vývojáři však mohou aplikace naprogramovat tak, že se o úniky dat cíleně pokouší. Druhou možností je, že se aplikace může stát terčem hackerského útoku a může tak být zneužita pro únik uživatelských dat. Dále mohou vývojáři v aplikacích špatně implementovat autentifikaci, autentizaci nebo *session managment*. Mnohé aplikace používají pro tyto důležité funkce nezabezpečené protokoly, popřípadě protokoly, u kterých již bylo zabezpečení prolomeno. Dle OWASP MobileTop10 je nezabezpečená transportní vrstva třetí nejhorší zranitelností mobilních zařízení s operačním systémem Android.

Dalším závažným bezpečnostním problémem jsou třetí strany, které mohou platit vývojářům aplikace za přístup k uživatelským datům a sbírat o uživateli informace. Někteří vývojáři přidávají do svých aplikací analyzační knihovny, které sbírají uživatelská data a provedou nad zachycenými daty analýzu, která poskytuje vývojářům informace o tom, jak je jejich aplikace využívána. Dalším možným důvodem vměšování třetí strany může být přímo marketingově cílená reklama. Pokud více aplikací obsahuje podobné knihovny, zachycená data poslouží k celkové analýze uživatelského zařízení, a především uživatele samotného.

5.1.1 Způsoby analýzy

Jedna z možností, jak dostat informace o potenciálních únicích je analýza zdrojového kódu aplikace. Tato možnost je ale zdlouhavá, neefektivní a často není ani úspěšná, protože vývojáři aplikací mohou mít různé osobní metody, jak do aplikace zakomponovat potencionálně nebezpečný kód.

Manuální statická analýza

Manuální testování zabezpečení aplikací je nazýváno také jako *„black-box“* testování, kdy se využívá přístup ke kompilovanému binárnímu kódu, ne však k původnímu zdrojovému kódu. Většinu aplikací lze snadno dekompilovat a reverzním inženýrstvím získat přístup k bajtovému kódu a binárnímu kódu. Výsledek této metody je skoro stejný jako původní zdrojový kód, je však možné že vývojáři aplikace využili různé metody pro zabránění k přístupu k binárnímu kódu, a tudíž tato metoda nemusí být úspěšná.

Automatická statická analýza

Pro zrychlení statické analýzy existují nástroje, které několikanásobně zefektivňují statickou analýzu a umožňují se zaměřit i na složitější aplikace. K dispozici je nepřeberné množství analyzátorů statických kódů, od open source skenerů až po plnohodnotné podnikové skenery. Nejlepší nástroje jsou v tomto případě také ty nejdražší.

Některé statické analyzátory se spoléhají na dostupnost zdrojového kódu, jiné pracují s kompilovanou aplikací. Statické analyzátory nemusí být schopny najít všechny problémy sami, i když mohou pomoci zaměřit se na potenciální problémy.

Dynamická analýza

Na rozdíl od statické analýzy se dynamická analýza provádí při chodu mobilní aplikace. Testováním se zkoumá souborový systém aplikace a monitoruje se chování aplikace v síti.

Pro dynamickou analýzu se využívá přesměrování síťového provozu přes *proxy*, jako například již zmíněný *Burp Suite*. Takto provedené přesměrování je užitečné pro čtení anebo modifikaci všech požadavků aplikací včetně odpovědí koncových bodů, které se používají pro testování autorizace, relací, managementu a dalších zkoumaných funkcí aplikace. [30]

5.2 Testování mobilních aplikací

Při testovaní bylo manuálně simulováno typické chování uživatele při používání aplikace. Během testu byli zkoumány základní funkce aplikace. A to včetně vytvoření uživatelského účtu, vyhledávání pomocí různých klíčových slov, provádění akcí, které vyžadují osobní identifikační údaje a zahrání jednoho herního kola v aplikaci *Solitare* a v aplikaci *Bolt* byla přidána k platebním metodám reálná kreditní karta.

Aby byla zajištěna integrita zachycených dat a aby se zabránilo možnému rušení z jiných aplikací, byla provedena následující opatření: během testování je otevřená pouze testovaná aplikace a žádná jiná. Toho dosáhneme ukončením všech ostatních aplikací a sledováním toho, zda jsou přenášena nějaká data, zatímco nejsou otevřené žádné aplikace. Pro každou aplikaci byla povolena všechna požadovaná oprávnění, například pro sdílení dat o poloze, s výjimkou notifikací. Důvodem, proč byli deaktivovány notifikace, je to, že notifikace odesílají data na pozadí i po zavření aplikace. To by vedlo k zachycení dat nejen z testované aplikace v libovolném čase, ale také z dříve testovaných aplikací.

5.2.2 Kontrola síťových dat na využívání SSL

Pro zjištění, zda některá z aplikací přenáší nezašifrovaná data přes wifi síť, byl pasivně monitorován jejich provoz pomocí programu *Wireshark*.

Pokud mobilní aplikace nepoužívají protokol *SSL / TLS*, data, která jsou přenášena, nejsou šifrována, a proto mohou být jednoduše zachycena prováděním pasivního síťového záchytu na provozním kanálu. Je-li použito SSL nebo TLS, jsou přenášená data zašifrována a žádná třetí strana není schopna odposlouchávat nebo rušit žádnou z přenášených zpráv. [31]

Aby bylo možné navázat spojení SSL, klient a server využívají šifrovací sady. Šifrovací sada se skládá z algoritmu pro výměnu klíčů, algoritmu podpisu, algoritmu blokového šifrování a autentizačního *hashe*, který vypočítává ověřovací klíč. K dispozici je celá řada šifrovacích sad, které poskytují různé úrovně zabezpečení. Výběr šifrovacích sad je důležitý, protože může ohrozit bezpečnost komunikace. To je možné prostřednictvím útoku *TLS Protocol Downgrade* [32] a je to jeden ze způsobů, jak může být spojení SSL / TLS oslabeno.

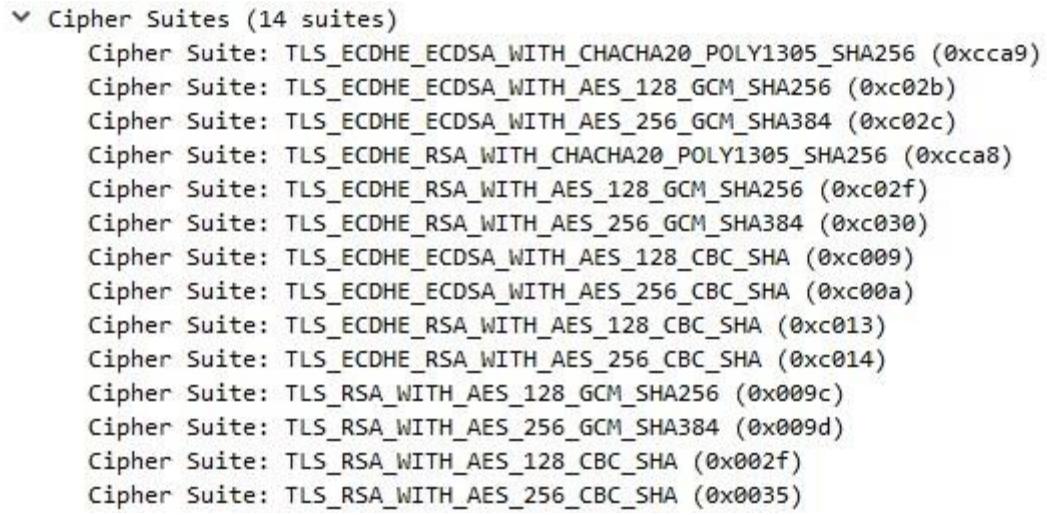
Ve *Wiresharku* je možné zobrazit seznam šifrových sad, které aplikace podporuje pro navázání zabezpečeného spojení se serverem. K posouzení, jak bezpečné tyto šifry jsou je využit nástroj *OSaft* [33], který slouží ke kontrole informací o certifikátech SSL / TLS a testuje spojení podle daného seznamu šifrovacích sad. Skript obsahuje všechny možné kombinace kódových sad následované popisem úrovně bezpečnosti.

Výsledky

Aplikace	Počet šifrovacích sad	Počet slabých šifrovacích sad
Bolt (Taxify)	14	0
Gmail	14	0
Ebay	12	0
Booking	35	0
Solitare	14	0
Dropbox	8	0
Můj vlak	50	0
Twitter	14	0

Obrázek 5.1 – Počet šifrovacích sad

Všechny testované mobilní aplikace používají protokol TLSv1.2 k vytvoření bezpečného kanálu pro komunikaci. Většina aplikací využívá čtrnáct šifrovacích sad. Všechny z těchto čtrnácti šifrovacích sad jsou z pohledu dnešní doby dostatečně odolné proti prolomení a je jedno jaká bude zvolena. Na obrázku (5.2) je zobrazen seznam nejčastěji používaných šifrovacích sad pro testované aplikace.



Obrázek 5.2 - Seznam používaných šifrovacích sad

Pořadí, ve kterém se sady zobrazují ve zprávě *ClientHello*, označuje preferované sady na straně klienta. Pokud se server rozhodne, přijme předvolbu klienta (může také ignorovat požadavek klienta a vybrat si šifrovací sadu, o které si myslí, že je nejlepší). Z pohledu zabezpečení proti odposlechu je síťové šifrování správně nastaveno.

5.2.3 Zkoumání síťových dat po překonání SSL

Pomocí dynamické analýzy je zde zkoumáno, jak různé aplikace přenášejí a zpracovávají uživatelská data na aplikační vrstvě modelu OSI. Na této vrstvě je možné využít aplikační šifrování. Toto šifrování je řešení pro větší datovou bezpečnost, protože šifruje data na nejvyšší vrstvě modelu OSI. Pokud jsou data na této vrstvě šifrována, nemohou být přečtena na žádné nižší vrstvě. Šifrování na aplikační vrstvě tak významnou měrou snižuje počet potencionálních útoků. Další výhodou je možnost zabezpečit citlivá data před jejich uložením do databáze, nebo prostředí cloudu. Analýza je prováděna pomocí nástroje *Burp Suite* na datových tokách HTTP/HTTPS, které jsou zodpovědné za více než 90 % aplikačního provozu [34]

Citlivá uživatelská data byla rozdělena do třech kategorii podle jejich obsahu. První kategorií jsou data, která osahují informace o chování uživatele při používání mobilního telefonu, mezi které patří soukromé zprávy z komunikačních aplikací, SMS nebo historie vyhledávaní. Druhou kategorií jsou osobní údaje, mezi které patří jméno uživatele, email, hesla atd. Třetí kategorií je pak poloha uživatelského telefonu.

Katogorie dat	Typ dat
Chování	Soukromé zprávy
	Histrorie vyhledávání
	Zdravotnické záznamy
Osobní	Jméno
	Věk
	Adresa
	Datum narození
	Udaje o zařízení
	Email
	Pohlaví
	Přihlašovací jméno
	Heslo
	Telefonní číslo
Poloha	Zeměpisná šířka a výška

Obrázek 5.3 – Hledané typy dat

Testovaný emulátor nebyl dlouhodobě využíván jako průměrný uživatelský telefon, proto neobsahuje takové množství osobních údajů jako reálný telefon používaný delší dobu. Základní uživatelská data do emulátoru však byly přidány, jako například telefonní kontakty, testovací hovory nebo osobní schůzky v kalendáři.

Během testování byla zaznamenána konkrétní klíčová slova a osobní uživatelská data, která byla při práci s aplikací použita a tyto klíčová slova pak hledána v zaznamenané komunikaci. Pro každou aplikaci byla povolena všechna požadovaná oprávnění, například sdílení polohy. Pro hledání klíčových slov byl využit *Burp Suite,* který obsahuje nástroj, přímo na to určený.

Chování	Soukromé zprávy	Ahoj toto je test
Chování	Soukromé zprávy	jak se máš
Chování	Soukromé zprávy	ahoj
Chování	Historie vyhledávání	gmail
Chování	Historie vyhledávání	ssl bypass
Chování	Historie vyhledávání	pivo
Chování	Historie vyhledávání	cvut fel
Chování	Historie vyhledávání	léto
Osobní	Jméno	jan
Osobní	Jméno	moravec
Osobní	Věk	24
Osobní	Adresa	česká republika
Osobní	Adresa	czech republic
Osobní	Adresa	praha
Osobní	Adresa	prague
Osobní	Adresa	vaníčkova
Osobní	Adresa	169 00
Osobní	Udaje o zařízení	Android 7.1.1
Osobní	Udaje o zařízení	Samsung 6S
Osobní	Udaje o zařízení	Genymotion
Osobní	Udaje o zařízení	MAC 08:00:27:dc:68:08
Osobní	Udaje o zařízení	IMEI 00000000000
Osobní	Email	frikeeXXXX@gmail.com
Osobní	Email	morXX@email.com
Osobní	Email	jan.mXXXX@email.com
Osobní	Pohlaví	male
Osobní	Přihlašovací jméno	frikee
Osobní	Přihlašovací jméno	frikeetest
Osobní	Heslo	XXXXXXX
Osobní	Heslo	XXXXXXX
Osobní	Heslo	XXXXXXX
Osobní	Heslo	XXXXXXX
Osobní	Heslo	card number
Osobní	Heslo	card expiration
Osobní	Heslo	cvv
Osobní	Telefonní číslo	494668207
Poloha	Zeměpisná šířka a výška	latitude
Poloha	Zeměpisná šířka a výška	longtitude
Poloha	Zeměpisná šířka a výška	50.08
Poloha	Zeměpisná šířka a výška	14.38

Obrázek 5.4 – Hledaná klíčová slova

Aplikace	Nešifrovaná přenesená data	Domény kterým jsou sdíleny data
Bolt (Taxify)	Udaje o zařízení	googleads
		facebook.com
	Poloha	-
	Telefonní číslo	-
	Verifikační kód	-
	Kreditní karta	paymentsos.com
	Jméno	googleapis.com
	Email	-
Gmail	Udaje o zařízení	-
Ebay	Heslo	-
	Email	crashlytics.com
		forter.com
	Udaje o zařízení	crashlytics.com
		forter.com
Booking	Adresa	googleapis.com
	Poloha	googleapis.com
		facebook.com
	Udaje o zařízení	googleanalytics.com
		facebook.com
	Email	googleanalytics.com
		facebook.com
Solitare	Udaje o zařízení	crashlytics.com
		facebook.com
Dropbox	Email	-
	Soukromé zprávy	-
Můj vlak	Heslo	-
	Email	-
	Histrorie vyhledávání	-
Twitter	Histrorie vyhledávání	-
	Heslo	-
	Email	-
	Adresa	-

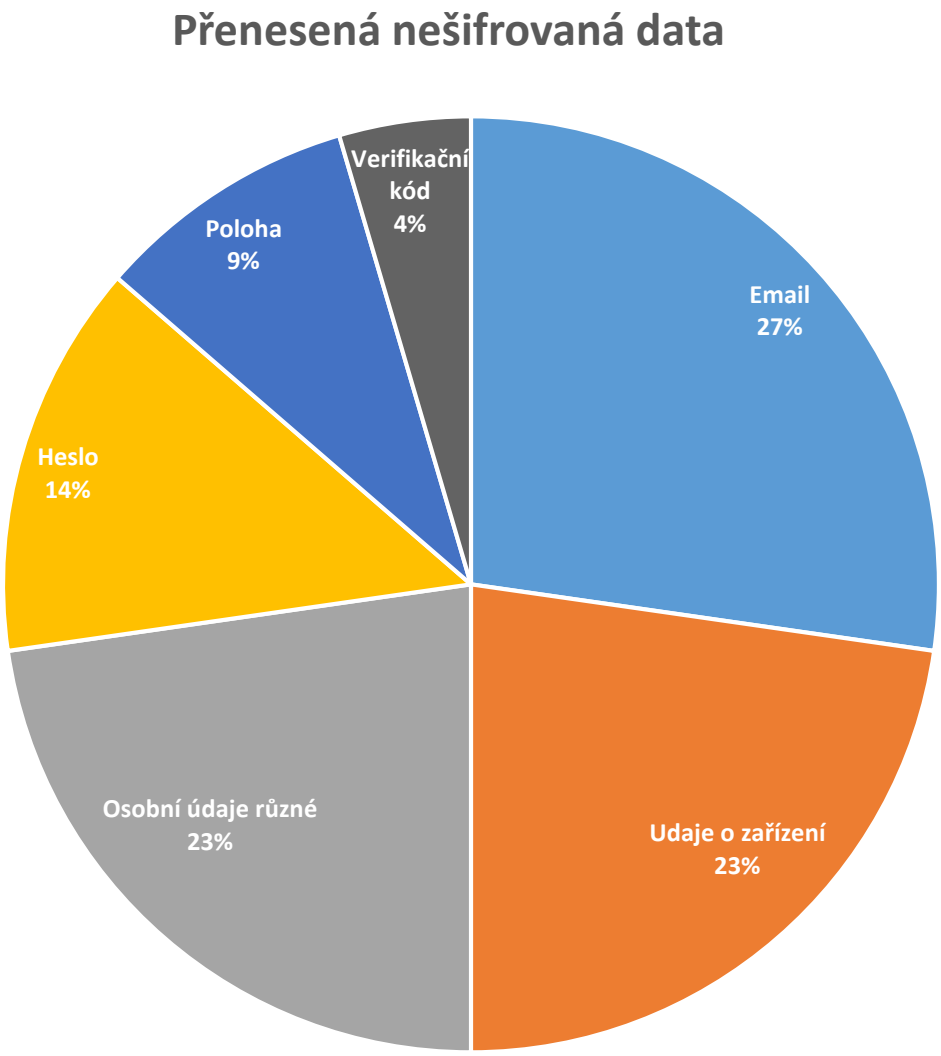
Obrázek 5.5 – Vyhodnocení bezpečnosti a úniku aplikací

Na obrázku 5.5 jsou zobrazena uživatelská data, která aplikace zasílají nešifrovaná a dále domény, kterým aplikace data předávaly.

U každé aplikace je předpokládáno, že toky, ke kterým dochází při testování, jsou pravděpodobně způsobeny aktivitou této aplikace. Jak již bylo zmíněno, byly minimalizovány procesy na pozadí, jako jsou notifikace pro jiné aplikace, aby se snížilo zahlcení komunikace. Nelze však vypnout všechny procesy na pozadí, například procesy související s vlastním operačním systémem telefonu. Pokud tedy Android odeslal během testování data na domény Google, nebo jiné, možná byla tato připojení zaznamenána jako součást konkrétní aplikace, která byla otevřena pro testování.

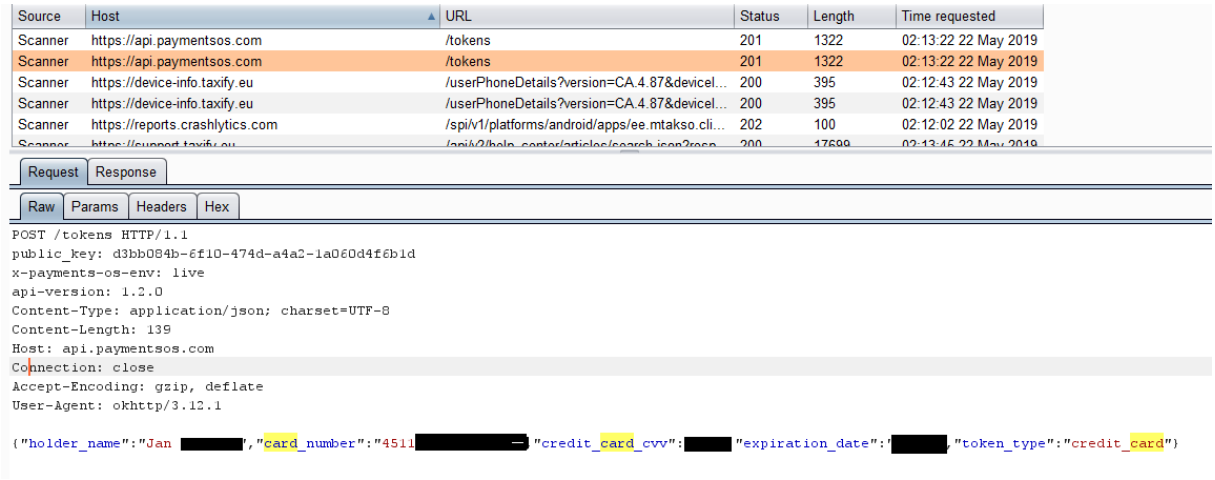
Ve výsledcích testování dominuje email jako nejčastěji nešifrovaná osobní informace, která se vyskytkla mezi všemi nešifrovanými přenesenými daty v 27 % případů. O druhé místo mezi nešifrovanými daty se dělí různé osobní údaje, mezi které patří historie vyhledávaní, soukromé zprávy atd. , a údaje o zařízení . Obě tyto skupiny se podílí na celkovém objemu s 23 %. Minoritnější částí jsou

zde pak také zastoupeny hesla s 14%, poloha s 9% a nakonec velice důležitý verifikační kód se 4%. Tento kód slouží k příhlášení do aplikace a je zasílán skrze SMS zprávu, zastupuje dvoustupňovou autorizaci. Shrnutí poměrů nešifrovaného provozu je zobrazeno na obrázku 5.6.



Obrázek 5.6 – Přenesená nešifrovaná data

K velice znepokojivému zjištění dochází u aplikace *Bolt,* která přenáší údaje o kreditní kartě nešifrované k doméně třetí strany *paymentsos.com,* která zprostředkovává platby pomocí kreditních karet. Údaje o kreditní kartě patří mezi uživatelskými daty, k těm vůbec nejcitlivějším.



Obrázek 5.7 – Nešifrovaná sdílená informace o platební kartě

Z výsledku testování lze dále určit, že ze všech aplikací přenáší nejméně uživatelských dat nešifrovaně aplikace *Gmail.* V této aplikaci jsou šifrovány veškeré údaje o uživateli včetně přijatých a odeslaných emailu. Jediný typ dat, který zde není šifrován jsou údaje o zařízení.

Na druhou stranu, v již zmíněné aplikaci *Bolt* dochází k přenosu největšího množství uživatelských dat nezabezpečeně. Celkem zde bylo detekováno sedm případů, kdy bylo možné číst PII data.

Sdílení dat třetím stranám je nejvíce zastoupeno v aplikaci *Booking,* která se zabývá hotelnictvím, a právě zde dochází k největším únikům dat. Mezi těmito daty se nejvíce objevují údaje o zařízení. Mezi další běžně sdílené typy dat patří emailová adresa, uživatelské jméno a poloha. Třetí stranou, která dostává nejvíce uživatelsky citlivých dat, je *Google*. Důvodem, proč jsou PII data sdílena se třetími stranami, je pravděpodobně proto, že jsou následně použity k cílení reklamy.

Mezi aplikace, které nesdílí žádná data se třetími stranami, patří Dropbox, Můj vlak a Twitter. Jedná se o všechny tři aplikace, které mají aktivovaný *certificate pinning*, a mohlo by se tedy zdát, že patři mezi aplikace, které se snaží s uživatelskými daty zacházet opravdu zodpovědně. Na druhou stranu je ale možné si všimnout, že v *Twitteru* je nešifrována historie vyhledávání, adresa, email, a dokonce i přístupové heslo.

Výsledky testování aplikací

Byl proveden záchyt paketů pro zjištění, zda některá z mobilních aplikací přenáší nezašifrovaná data prostřednictvím sítě wifi. Výsledky ukazují, že všechny otestované aplikace používají protokol SSL k vytvoření bezpečného kanálu pro komunikaci se serverem. Každý, kdo provede bezdrátový záchyt paketů, sice může zachytit provozt, ale nebude ho moci přečíst. SSL může poskytovat soukromí a datovou integritu mezi klientem a serverem, avšak tato bezpečnost může být oslabena pokud se zvolí slabé šifrovací sady. Během testování bylo zjištěno, že všech osm aplikací používá z dnešního pohledu bezpečné testovací sady.

Při testování bezpečnosti přenášených dat na aplikační vrstvě bylo zjištěno, že žádná z aplikací není zcela dokonale zabezpečena. Nejlépe na tom z pohledu šifrování dat a jejich předávání třetím stranám je aplikace *Gmail,* následovaná *Dropboxem,* u kterého se ukázalo, že nešifruje přenášené zprávy, které je možné spolu s daty zasílat dalším uživatelům.

Domény, do kterých aplikace nejčastěji odesílají osobní data, jsou: *googleanalytics.com*, *googleservices.com* a *googleads.com*. To může být způsobeno tím, že společnost Google vlastní různé mobilní reklamní sítě a služby, jako je *AdMob, Google Analytics, Double CLick* a *iAds*.

Závěr

Na začátku této práce byla rozebrána softwarová struktura operačního systému Android včetně toho, jak funguje běh aplikací. Dále je popsán princip zajištění přístupu do privilegovaného režimu operačního systému, jaké tento přístup má výhody a jaké nevýhody.

Ve třetí kapitole jsou kompletně popsány metody, které lze využít pro efektivní odchytávaní síťového provozu na *Androidu.* Z těchto metod je pro další využití vybrána možnost simulace pomocí emulátoru. Dále je rozebráno, jakým způsobem jsou data šifrována a jak je dešifrovat, což je klíčové pro jejich další analýzu.

Po aplikování bezpečnostních mechanismů, jako je neochota aplikací důvěřovat jiným certifikátům než těm, které jsou v uložišti důvěryhodných a aktivovaným *„certificate pinningem“,* je analýza aplikací a celého operačního sytému Android obtížnější než kdykoliv dřív. Hlavním důvodem je nutnost obcházet bezpečnostní mechanismy každé aplikace zvlášť a zcela individuálně tak aby mohl být provoz analyzován.

V praktické části práce bylo zkoumáno, jakým způsobem jsou data uživatele přenášena a zpracovávána různými mobilními aplikacemi. Bylo vybráno osm aplikací, které pracují na operačním systému Android a na nich simulováno typické chování uživatele. U každé aplikace byla otestována komunikace s osobně podepsaným certifikátem. Všechny aplikace však odmítly navázat spojení z důvodu podezření na porušení integrity provozu. Pomocí cetifikátu od *Burp Suite,* který byl přidán mezi důvěryhodné, se podařilo navázat spojení pěti aplikacím z osmi. Pro další tři aplikace které využívají certificate pinning byl vyzkoušen modul *Inspeckage*, který zafungoval na dvou z nich. I když je tato efektivní metoda použita tak, aby se zabránilo útokům *Man-in-the-middle*, podařilo se zabezpečení obejít i u aplikace *Twitter,* která má za sebou velice silný vývojářský tým, který se stará o správnou implementaci bezpečnostních mechanismů.

U každé z aplikací bylo ověřeno, že používá šifrování síťového provozu pomocí protokolu TLSv1.2 a využívá také bezpečné šifrovací sady.

Dále byly aplikace testovány na to, zda šifrují provoz na aplikační vrstě. Ukázalo se, že žádná z aplikací provoz nešifruje stoprocentně. Nejbezpečnější je v tomto směru *Gmail,* u kterého bylo objeveno, že nešifrovaně zasílá pouze informace o mobilním telefonu a žádná uživatelská data s nikým nesdílí. Toto je ale pravděpodobně způsobeno tím, že právě Google, který aplikaci vyvíjí, je hlavním příjemcem uživatelských dat z ostatních aplikací. Naopak u aplikace *Bolt* se ukázalo, že provoz na aplikační vrstvě nešifruje téměř vůbec a dokonce zasílá třetím stranám takové informace, jako údaje o kreditní kartě a to dokonce v nezašifrované podobě. V množsví dat které je sdíleno s třetí stranou se úkazalo že dominuje aplikace *Booking*. Mezi uživatelské informace které nejsou šifrovány patří adresa, email, údaje o zařízení a v neposlední řadě také poloha. Všechny tyto informace jsou také zasílány serverům *Googlu a Facebooku.* V nakupní aplikaci *Ebay* a aplikaci od Českých drah - *Můj vlak*, nejsou šifrovány přístupové údaje k uživatelským účtum a historie vyhledávaní. Aplikace *Můj vlak* však žádná z těchto dat nesdílí se třetí stranou. Naproti tomu *Ebay* sdílí údaje o zařízení a email se společností *Crashlytics,* která je vlastněná *Googlem.* Dále také tyto data zasílá serverům společnosti *Forter,* která se specializuje na odhalování podvodu (*fraudu)* na internetu. Herní aplikace *Solitare* nepracuje s osobními daty, ale informace o telefonu zasílá nezašifrovaně třetím stranám *Crashlytics* a *Facebooku.*

Obecně lze říci, že metody, které byly zvoleny pro hodnocení, jak bezpečně mobilní aplikace přenáší a zpracovávají uživatelská data přes wifi síť, jsou účinná, ale mají svá omezení. Všechny použité metody vyžadují lidský zásah a to výrazně omezuje počet aplikací, které je možné testovat. I když má použitá metodika svá omezení, stále se díky ní daří dospět k významným závěrům o tom, jak bezpečně se přenášejí a zpracovávají uživatelská data různými aplikacemi. Kromě toho, jsou použité metody navrženy tak, aby porušily nebo obcházely základní bezpečnostní mechanismy, které vývojáři používají, jako je například SSL a *certificate pinning*. To je důkazem, že tato bezpečnostní opatření nejsou nezranitelná. V důsledku toho si musí uživatelé plně uvědomit, že jejich osobní údaje nemohou být nikdy 100% bezpečné a jediným způsobem, jak chránit jejich soukromí, je porozumět těmto bezpečnostním rizikům.

6. Literatura

6.1 Seznam použité literatury

[1] Number of smartphone users worldwide from 2014 to 2020 (in billions) [online]. [cit. 2019-05-22]. Dostupné z: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[2] HOGBEN, Giles a Marnix DEKKER. *Smartphones: Information security risks, opportunities and recommendations for users* [online]. 2010 [cit. 2019-03-23]. Dostupné z: https://www.enisa.europa.eu/publications/smartphones-information-security-risks-opportunities-and-recommendations-for-users

[3] SCIENCEDAILY. *Knowledge of location sharing by apps prompts privacy action* [online]. Carnegie Mellon University, 2015 [cit. 2019-03-23]. Dostupné z: https://www.sciencedaily.com/releases/2015/03/150323132846.htm

[4] STATISTA. *Share of Android OS of global smartphone shipments from 1st quarter 2011 to 2nd quarter 2018** [online]. 2018 [cit. 2019-03-24]. Dostupné z: https://www.statista.com/statistics/236027/global-smartphone-os-market-share-of-android/

[5] HERBSTER, Raul, Scott DELLATORRE, Peter DRUSCHEL a Bobby BHATTACHARJEE. *Privacy Capsules: Preventing information leaks by mobile app* [online]. [cit. 2019-03-28]. Dostupné z: https://people.mpi-sws.org/~druschel/publications/pc.pdf?fbclid=IwAR286VsltatSuy8ZmMbS-u1OeGS9ZeXW0h7fKGDOKlNz3uGldCPu1SYCZVM

[6] HAMIDREZA, Aria. UNIVERSITA DEGLI STUDI DE GENOVA. *Android Malware Detection Using Network Behavior Analysis And Machine Learning* [online]. 2017 [cit. 2019-03-30]. Dostupné z: https://people.mpi-sws.org/~druschel/publications/pc.pdf?fbclid=IwAR286VsltatSuy8ZmMbS-u1OeGS9ZeXW0h7fKGDOKlNz3uGldCPu1SYCZVM

[7] KILIÁN, Karel. SVĚT ANDROIDA. *Root: jak funguje, co znamená a k čemu je dobrý?* [online]. 2016 [cit. 2019-04-1]. Dostupné z: https://www.svetandroida.cz/co-je-to-root-jak-funguje/

[8] WIKIPEDIE OTEVŘENÁ ENCYKLOPEDIE. *Rootování Androidu* [online]. 2019 [cit. 2019-04-2]. Dostupné z: https://cs.wikipedia.org/wiki/Rootování_Androidu

[9] GREFER, Lars. GITHUB. Android-IMSI-Catcher-Detector [online]. 2019 [cit. 2019-04-5]. Dostupné z: https://github.com/CellularPrivacy/Android-IMSI-Catcher-Detector/blob/development/README.md

[10] SOOM.CZ. Fake Cell Phone Towers & Stealth SMS [online]. 2014 [cit. 2019-04-6]. Infinity. Dostupné z: https://www.soom.cz/clanky/1163--Fake-Cell-Phone-Towers-Stealth-SMS

[11] WIKIPEDIE OTEVŘENÁ ENCYKLOPEDIE. *IMSI-catcher* [online]. 2019 [cit. 2019-04-8]. Dostupné z: https://en.wikipedia.org/wiki/IMSI-catcher

[12] WIKIPEDIE OTEVŘENÁ ENCYKLOPEDIE. Proxy server [online]. 2019 [cit. 2019-04-8]. Dostupné z: https://cs.wikipedia.org/wiki/Proxy_server

[13] *Android PCAP* [online]. 2019 [cit. 2019-04-15]. Dostupné z: https://www.kismetwireless.net/static/android-pcap/

[14] Brian. *Monitoring Android Traffic with Wireshark* [online]. 2014 [cit. 2019-04-16]. Dostupné z: https://www.linuxjournal.com/content/monitoring-android-traffic-wireshark

[15] *Run Android on your PC: Android-x86* [online]. 2019 [cit. 2019-05-22]. Dostupné z: https://www.android-x86.org

[16] Damian Mehers. *Damian Mehers' Blog* [online]. [cit. 2019-05-20]. Dostupné z: https://damian.fyi/2011/05/26/android-vpn-to-windows7/

[17] *Android Developers* [online]. [cit. 2019-05-20]. Dostupné z: https://developer.android.com/training/articles/keystore

[18] *MyHowTo.org: Intercepting and decrypting SSL communications between Android phone and 3rd party server* [online]. [cit. 2019-05-15]. Dostupné z: http://www.myhowto.org/java/81-intercepting-and-decrypting-ssl-communications-between-android-phone-and-3rd-party-server/

[19] *Ropnop: Configuring Burp Suite with Android Nougat* [online]. [cit. 2019-05-11]. Dostupné z: https://blog.ropnop.com/configuring-burp-suite-with-android-nougat/

[20] J.D'ORAZIOAKIM-KWANG, Christian a Raymond CHOO. *A technique to circumvent SSL/TLS validations on iOS devices* [online]. 2016 [cit. 2019-05-23]. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0167739X16302801?fbclid=IwAR2qmLoXbUsDZovSroJjIF9RUO2HNeorqWA8fK7S2ki9nGPdc4bG_TfwIfU#br000075. University of South Australia.

[21] XdaDevelopers [online]. [cit. 2019-05-23]. Dostupné z: https://forum.xda-developers.com/showthread.php?t=3034811

[22] Github: Inspeckage [online]. [cit. 2019-05-23]. Dostupné z: https://github.com/ac-pm/Inspeckage

[23] ASS, Cody. NetSPI [online]. [cit. 2019-05-23]. Dostupné z: Four Ways to Bypass Android SSL Verification and Certificate Pinning Nvisium: Android Assessments with GenyMotion + Burp [online]. [cit. 2019-05-23]. Dostupné z: https://nvisium.com/blog/2014/01/24/android-assessments-with-genymotion-burp.html

[24] Nvisium: Android Assessments with GenyMotion + Burp [online]. [cit. 2019-05-23]. Dostupné z: https://nvisium.com/blog/2014/01/24/android-assessments-with-genymotion-burp.html

[25] Frida Codeshare: Universal Android SSL Pinning Bypass with Frida [online]. [cit. 2019-05-18]. Dostupné z: https://codeshare.frida.re/@pcipolloni/universal-android-ssl-pinning-bypass-with-frida/

[26] Github: Frida [online]. [cit. 2019-05-21]. Dostupné z: https://github.com/frida/frida/releases

[27] AndreaFortuna: How to install (and run) tcpdump on Android devices [online]. [cit. 2019-05-23]. Dostupné z: https://www.andreafortuna.org/2018/05/28/how-to-install-and-run-tcpdump-on-android-devices/

[28] Management mania [online]. [cit. 2019-05-23]. Dostupné z: https://managementmania.com/cs/osobni-data-personal-data

[29] GooglePlay: Ochrana soukromí, zabezpečení a klamání [online]. [cit. 2019-05-19]. Dostupné z: https://play.google.com/intl/cs/about/privacy-security-deception/personal-sensitive/index.html

[30] Github [online]. [cit. 2019-05-23]. Dostupné z: https://github.com/OWASP/owasp-mstg/blob/master/Document/0x06b-Basic-Security-Testing.md

[31] Github: CheatSheetSeries [online]. [cit. 2019-05-23]. Dostupné z: https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.md

[32] B. Moeller a A. Langley. : TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks [online]. Google. [cit. 2019-05-23]. ISSN 2070-1721. Dostupné z: https://tools.ietf.org/pdf/rfc7507.pdf

[33] OWASP: O-Saft [online]. [cit. 2019-05-23]. Dostupné z: https://www.owasp.org/index.php/O-Saft/Documentation

[34] RAO, Ashwin, Arash MOLAVI KAKHKI, Abbas RAZAGHPANAH, Amy TANG, Shen WANG a Justine SHERRY. Using the Middle to Meddle with Mobile [online]. 2016 [cit. 2019-05-23]. Dostupné z: https://david.choffnes.com/pubs/meddle-main.pdf. Univ. of Washington eStony Brook.

6.2 Seznam obrázků a tabulek

Příloha

Tento skript [25], byl použit v aplikaci *Frida*, k o obejití *certificate pinningu.*

```
setTimeout(function(){
    Java.perform(function (){
        console.log("");
            console.log("[.] Cert Pinning Bypass/Re-Pinning");

            var CertificateFactory = Java.use("java.security.cert.CertificateFactory");
            var FileInputStream = Java.use("java.io.FileInputStream");
            var BufferedInputStream = Java.use("java.io.BufferedInputStream");
            var X509Certificate = Java.use("java.security.cert.X509Certificate");
            var KeyStore = Java.use("java.security.KeyStore");
            var TrustManagerFactory = Java.use("javax.net.ssl.TrustManagerFactory");
            var SSLContext = Java.use("javax.net.ssl.SSLContext");
            // Load CAs from an InputStream
            console.log("[+] Loading our CA...")
            var cf = CertificateFactory.getInstance("X.509");
            try {
                var fileInputStream = FileInputStream.$new("/data/local/tmp/cert-der.crt");
            }
            catch(err) {
                console.log("[o] " + err);
            }
            var bufferedInputStream = BufferedInputStream.$new(fileInputStream);
                var ca = cf.generateCertificate(bufferedInputStream);
            bufferedInputStream.close();

                var certInfo = Java.cast(ca, X509Certificate);
            console.log("[o] Our CA Info: " + certInfo.getSubjectDN());

            // Create a KeyStore containing our trusted CAs
            console.log("[+] Creating a KeyStore for our CA...");
            var keyStoreType = KeyStore.getDefaultType();
            var keyStore = KeyStore.getInstance(keyStoreType);
            keyStore.load(null, null);
            keyStore.setCertificateEntry("ca", ca);
            // Create a TrustManager that trusts the CAs in our KeyStore
            console.log("[+] Creating a TrustManager that trusts the CA in our KeyStore...");
            var tmfAlgorithm = TrustManagerFactory.getDefaultAlgorithm();
            var tmf = TrustManagerFactory.getInstance(tmfAlgorithm);
            tmf.init(keyStore);
            console.log("[+] Our TrustManager is ready...");

            console.log("[+] Hijacking SSLContext methods now...")
            console.log("[-] Waiting for the app to invoke SSLContext.init()...")
                SSLContext.init.overload("[Ljavax.net.ssl.KeyManager;",
"[Ljavax.net.ssl.TrustManager;", "java.security.SecureRandom").implementation =
function(a,b,c) {
                        console.log("[o] App invoked javax.net.ssl.SSLContext.init...");
                SSLContext.init.overload("[Ljavax.net.ssl.KeyManager;",
"[Ljavax.net.ssl.TrustManager;", "java.security.SecureRandom").call(this, a,
tmf.getTrustManagers(), c);
                        console.log("[+] SSLContext initialized with our custom TrustManager!");
                }
    });
},0);
```