Penetračné testovanie súčasných mobilných aplikácií

Michal Kubíček

Bakalárska práca
2022

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav automatizace a řídicí techniky

Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Michal Kubíček**
Osobní číslo:	**A18165**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Informační a řídicí technologie**
Forma studia:	**Prezenční**
Téma práce:	**Penetrační testování současných mobilních aplikací**
Téma práce anglicky:	**Penetration Testing of Contemporary Mobile Applications**

Zásady pro vypracování

1. Vypracujte literární rešerši na dané téma.
2. Proveďte analýzu současných softwarových nástrojů, kterém lze využít na statické testování mobilních aplikací běžících pod operačním systémem Android.
3. Proveďte analýzu současných softwarových nástrojů, kterém lze využít na dynamické testování mobilních aplikací běžících pod operačním systémem Android.
4. Proveďte praktickou demonstraci současných softwarových nástrojů statické analýzy.
5. Proveďte praktickou demonstraci současných softwarových nástrojů dynamické analýzy.
6. Navrhněte vhodnou metodiku penetračního testování mobilních aplikací běžící pod operačním systémem Android.

Forma zpracování bakalářské práce: **tištěná/elektronická**
Jazyk zpracování: **Slovenština**

Seznam doporučené literatury:

1. GUPTA, Adytia. Learning Pentesting for Android Devices. 1st ed. Birmingham: Packt Publishing, 2014. ISBN 1190314.
2. BALOCH, Rafay. Ethical Hacking and Penetration Testing Guide. 1st ed. Boca Raton: Taylor & Francis Group, 2015. ISBN 1482231611.
3. HUNTLEY, Samuel. Android Application Pentesting Handbook.1st ed. California: CreateSpace Independent Publishing Platform, 2016. ISBN 1530176972.
4. CHELL, Dominic, Tyrone ERASMUS a Shaun COLLEY. The Mobile Application Hacker's Handbook. 1st ed. Indianapolis: John Willey, 2015. ISBN 1118958500.
5. GUNASEKERA, Sheran. Android Apps Security: Mitigate Hacking Attacks and Security Breaches. 2nd ed. Singapore: Apress Media, 2020. ISBN 1484216814.
6. WONG, Reginald. Mastering Reverse Engineering: Re-engineer your ethical hacking skills. 1st ed. Birmingham: Packt Publishing, 2018. ISBN 178883884X.

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

Datum zadání bakalářské práce: **15. ledna 2022**
Termín odevzdání bakalářské práce: **20. května 2022**

doc. Mgr. Milan Adámek, Ph.D. v.r.
děkan

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

prof. Ing. Vladimír Vašek, CSc. v.r.
ředitel ústavu

Ve Zlíně dne 15. ledna 2022

Michal Kubíček

Penetračné testovanie súčasných mobilných aplikácií

Prehlašuji, že

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

Prehlašuji,

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

Ve Zlíně, dne 9.5.2022

Michal Kubíček v.r.
Podpis studenta

ABSTRAKT

Táto bakalárska práca sa zaoberá penetračným testovaním mobilných aplikácií bežiacich na platforme Android. Cieľom práce bolo analyzovať a popísať nástroje využívané na statickú aj dynamickú analýzu a následne tieto nástroje aj použiť na praktické ukážky využívania nájdených zraniteľností v aplikáciách. Výsledkom práce sú teda ukážky rôznych útokov a navrhnutie metodiky penetračného testovania aplikácií.

Kľúčové slova: Penetračné testovanie, Android, Aplikácia, Zraniteľnosť, Útok, Analýza, Data, Metodika

ABSTRACT

This Bachelor's thesis deals with penetration testing of mobile applications running on the Android platform. The objective was to analyze and describe the tools used for static and dynamic analysis, then use them for practical examples of exploiting the vulnerabilities found in applications. The result of the thesis are demonstrations of various attacks and suggestion of a methodology for penetration testing of applications.

Keywords: Penetration testing, Android, Application, Vulnerability, Attack, Analysis, Data, Methodology

POĎAKOVANIE

Chcel by som poďakovať pánovi Ing. Milanovi Oulehlovi Ph.D. za cenné rady, odborné vedenie a čas, ktorý mi v priebehu celého vypracovania práce poskytoval.

OBSAH

ÚVOD

Mobilné aplikácie sa postupom času stávajú čoraz väčšou súčasťou ľudských životov, veľa ľudí si na tieto aplikácie zvyklo a bez rozmýšľania s nimi zdieľajú informácie zo súkromia ich životov. Sú však natoľko bezpečné, aby sme sa na ne mohli spoliehať s úplnou dôverou?

Práve na zistenie bezpečnosti a potenciálnych zraniteľností slúži penetračné testovanie, ktoré umožňuje developerom zistiť, na koľko je ich aplikácia bezpečná voči zneužitiu.

V súčasnej dobe prevláda medzi mobilnými operačnými systémami Android, ktorý používa približne 70% populácie.[1] Preto je táto Bakalárska práca zameraná na penetračné testovanie aplikácií bežiacich pod systémom Android.

V teoretickej časti bude práca zameraná na moderné nástroje slúžiace na statické či dynamické penetračné testovanie. Tieto nástroje budú podrobne analyzované a rozobrané na ich jednotlivé súčasti, taktiež budú popísané zraniteľnosti, ktorými môžu byť aplikácie napadnuté či zneužité.

Práca sa bude zaoberať aj samotným zložením a fungovaním aplikácií pre systém Android. Tieto prvky by mali umožniť dostatočný prehľad v problematike na to, aby mohli byť tieto nástroje aj prakticky vyskúšané a aby bolo možné pracovať s nimi.

V tejto práci budú prakticky vyskúšané jednotlivé nástroje, ich funkčnosť, obsiahlosť a schopnosti preniknúť do aplikácie, s cielom nájsť a následne využiť nejaké jej zraniteľnosti. Výsledkom práce budú praktické ukážky práce s týmito nástrojmi, a návrh vhodnej metodiky, ktorá by mala byť používaná pri penetračných testoch aplikácií bežiacich pod systémom Android.

I. TEORETICKÁ ČASŤ

1 LITERÁRNA REŠERŠ NA DANÚ TÉMU

Problematiku penetračného testovania rieši množstvo prameňov. Z akademických prameňov je to napríklad Scopus. Článok *Penetration Frameworks and Development Issues in Secure Mobile Application Development: A Systematic Literature Review[2]* okrem iného popisuje aj základnú štruktúru aplikácie pre operačný systém Android. Tieto informácie sú pre túto prácu veľmi prínosné, pretože tieto aplikácie budú testované a budú v nich hľadané zraniteľnosti. Preto je zásadné poznať a dobre sa orientovať v tejto štruktúre. Ďalej tento článok pojednáva o hrozbách a zraniteľnostiach systému Android, takže poskytne dobrý úvod do tejto problematiky.

Ďalším zaujímavým článkom je *A Suggested Model for Mobile Application Penetration Test Framework[3]*. Tento článok ponúka informácie o rôznych typoch testovania, ako aj tipy na nástroje, ktoré je možné na tieto úkony použiť. Niektoré z týchto nástrojov budú v práci aj použité. V tomto článku je možné nájsť prehľadné vývojové diagramy o možnostiach penetračného testovania.

V článku *Mobile Application Security Penetration Testing Based on OWASP[4]*, jeho autori rozoberajú jednotlivé zraniteľnosti mobilných aplikácií, podľa zoznamu zostaveného OWASPom. Niektorými z týchto zraniteľností sa bude zaoberať aj táto práca, preto je tento článok dobrým obohatením vedomostí.

Veľmi dobré podanie informácií o MITM útokoch, proxy a odchytávaní komunikácie poskytuje článok *A MITM Based Penetration Test Efficiency Improvement Approach for Traffic-Encrypted Mobile Applications of Power Industry [5]*. Je v ňom popísaný aj nástroj Burp Suite, ktorý je v tejto práci použitý na ukážku odchytávania komunkácie. Taktiež sa zaoberá jednotlivými typami proxy a dáva tak možnosť hlbšieho pohľadu na funkčnosť a architektúru MITM útoku pomocou proxy.

V rámci rešerše boli spracované aj ďalšie zdroje ako sú napr. OWASP, developer.android a podobne. OWASP je v rámci cybersecurity veľmi dôveryhodný a uznávaný zdroj, ktorý je rešpektovaný napr. Microsoftom.

1.1 Mobilná aplikácia

Mobilná aplikácia je typ softvéru, ktorý je navrhnutý tak, aby bežal na mobilnom zariadení (Smartfóny, Tablety). Často takéto aplikácie poskytujú užívateľom podobne zamerané služby, ako aplikácie na počítačoch. Tieto aplikácie bývajú zväčša malé individuálne softwarové balíčky s jednoúčelovo zameranou funkčnosťou. Tzn. väčšinou sú vyvíjané s cieľom splniť určitý cieľ. Aplikácie môžu byť najzákladnejšie rozdelené podľa toho, na akom mobilnom operačnom systéme bežia, pričom najbežnejšie sú aplikácie bežiace na operačnom systéme Android a iOS. Veľké množstvo aplikácií využíva princíp klient-server, kde práve klientská časť je tá, ktorá beží na mobilnom zariadení. Aplikácie môžu byť v mobilnom zariadení už predinštalované. Sú to aplikácie ako kontakty, správy, fotoaparát a ďalšie iné. Dodatočné aplikácie si potom užívateľ môže dosťahovať z oficiálnych obchodov, ako je v prípade Androidu Google Play od firmy Google. Aplikácie sa však dajú stiahnuť aj z rôznych internetových stránok, čo však už nesie väčšie riziko. Aplikácia, ktorá nie je dostatočne zabezpečená, môže byť modifikovaná útočníkom, ktorý čaká dokiaľ si potenciálna obeť aplikáciu stiahne. Aby programátor, ktorý vytvorí aplikáciu zistil, či je bezpečná, môže nechať aplikáciu prejsť penetračným testovaním, ktoré má za cieľ nájsť bezpečnostné chyby a diery v aplikácii a upozorniť na ne programátora.

1.2 Systém Android

Android je mobilný operačný systém vytvorený v roku v 2007, pod zastrešením firmy Google, ktorá odkúpila pôvodnú firmu Android i.n.c. Stala sa tak dcérskou spoločnosťou tejto firmy. Aj napriek vlastníctu je však tento systém vo forme Open source, ktorá ho robí voľne dosiahnuteľný pre kohokoľvek, vrátane komerčných použití.[6]

Z hľadiska histórie počas celej doby existencie a vývoja prešiel tento systém niekoľkými zásadnými zmenami. Po vývoji systému Android pre mobilné telefóny, prišlo s uvedením verzie 3.0 (HoneyComb) zameranej na tablety a zariadenia s väčším displejom. Hlavnou myšlienkou bolo ponúknutie funkcií, ktoré menšie displeje nezvládli.[7]

V ďalšej verzii, ktorou bola 4.0 sa vývojári rozhodli spojiť funkcie z verzie 3.0 pre tablety, s verziou 2.3.(Gingerbread), orientovanou na smartfóny. V tejto verzii sa po prvýkrát

objavila aj funkcia odomykania zariadenia pomocou odfotenia tváre, jednalo sa tak o prvé použitie biometriky v histórii Androidu. [7]

Vo verzii 4.4 (KitKat) prišlo obmedzenie, že aplikácia nemôže čítať dáta na externom úložisku, keď nemá povolenie *READ_EXTERNAL_STORAGE*. Ak však aplikácia pristupuje k jej špecifickým adresárom, nepotrebuje toto povolenie. Ďalej prišli vylepšenia umožnujúce NFC, teda funkciu platobného klienta simulujúceho platobnú kartu. [8]

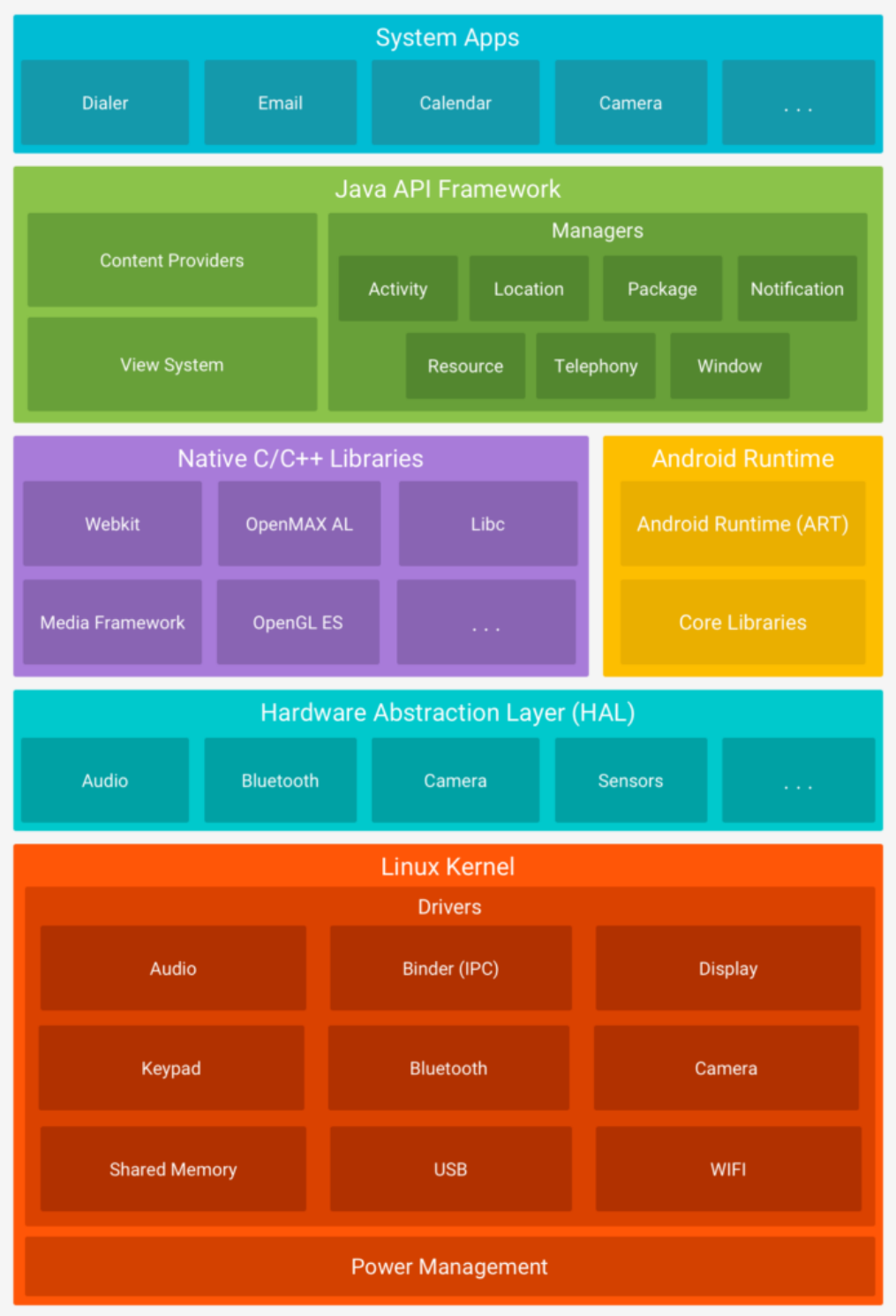
Príchodom Androidu 5.0(Lollipop) prišla jedna z najväčších zmien, a to úplný prechod na *runtime ART* z Dalvik Virtual Machine, už vo verzii 4.4. bol koncept ART ako alternatíva k Dalvik Virtual Machine, avšak vo verzii 5.0. už bol inkludovaný len ART runtime. [9]

Android 6.0.(Marshmallow) priniesol taktiež markantnú zmenu, a to zmenu runtime povolení, teda tzv. *permission modelu*. Táto verzia predstavila nový model povolení, v ktorom môžu používatelia priamo spravovať povolenia aplikácií za behu. Tento model poskytuje lepšiu kontrolu nad povoleniami a zjednodušuje inštaláciu a aktualizácie vývojárom aplikácií.[10]

Verzia 8.0.(Oreo) priniesla zlepšenie výdrže batérie vďaka tomu, že po prechode aplikácie do „cached“ stavu, teda stavu bez aktívnych komponentov, systém uvoľní prebudenia aplikácie. Ďalej majú aplikácie na pozadí obmedzený prístup k službám na pozadí. [11]

Systém je vyvíjaný ďalej do aktuálne najnovšej verzie Android 12, ktorá priniesla množstvo vylepšení hlavne po stránke užívateľského pôžitku, napríklad navigáciou pomocou gestikulácie, efektov aplikácií, alebo viac oknovým módom.

Architektúra tohoto operačného systému obsahuje viacero komponentov pre podporu potrieb rôznych zariadení.



Obrázok 1. Štruktúra operačného systému Android[12]

1.2.1 Linux Kernel

Základom platformy Android je Linux Kernel, v preklade Linuxové jadro. Je zodpovedné za vytvárenie vlákien, správu pamäte a pod. Stručne povedané, jadro je kľúčovou súčasťou operačného systému Android, ktorá spravuje prostriedky CPU. Použitie Linuxového jadra umožňuje vyvíjať výrobcom ovládače pre známe jadro a taktiež umožňuje používať základne bezpečnostné prvky ako napr. Sandboxovanie aplikácií. [13]

1.2.2 Hardware Abstraction Layer (HAL)

HAL definuje štandardné rozhranie, ktoré určuje kompatibilitu hardvéru s vyššou vrstvou Java API Framework. HAL obsahuje viacero knižníc, ktoré implementujú rozhranie pre špecifický typ hardvéru, ako napríklad snímač odtlačkov prstov alebo kamera. Ak teda dôjde k volaniu od API Frameworku, ktoré sa dožaduje prístupu k hardvéru, systém Android načíta konkrétnu knižnicu tohoto komponentu. [14]

1.2.3 Android Runtime

Android runtime (prostredie) je jednou z najdôležitejších častí systému Android. Pri použití ART má každá aplikácia svoj vlastný proces s vlastnou inštanciou ART. Jeho predchodca je Dalvik runtime, ktorý bol používaný do verzie Android 5.0. ART je zostavený tak, že spúšťa viacero virtuálnych strojov na zariadeniach s malou pamäťou spustením súborov DEX, ktoré sú bajtkódový formát optimalizovaný a navrhnutý pre minimálne nároky na pamäť. ART má základné funkcie ako sú kompilácia AOT (Ahead-of-Time), kompilácia JIT (Just-in-time) a Garbagge collection. [15]

1.2.4 Knižnice Native C/C++

Komponenty systému Android ako sú napríklad ART alebo HAL, sú založené na natívnom kóde, ktorý vyžaduje knižnice v jazyku C a C++. Rozhrania API Framework sú poskytnuté Androidom s cieľom sprístupnenia funkcionality niektorých týchto natívnych knižníc aplikáciám.[16]

1.2.5 Aplikačný framework

Vrstva aplikačného frameworku poskytuje vývojárom potrebné API (Application Programming Interface), teda rozhranie na programovanie aplikácií. Veľa funkcií je tak dostupných vďaka volaniu rozhraní API napísaných v jazyku Java. Základnými stavebnými blokmi pri vytváraní aplikácií sú práve tieto rozhrania. Pri jazyku Java tak túto vrstvu môžeme volať *Java API Framework*. [17]

1.2.6 Systémové aplikácie

Aplikácie v zariadení Android je možné rozdeliť do dvoch skupín, buď to môžu byť systémové alebo nesystémové aplikácie.

Systémové aplikácie sú predinštalované aplikácie nachádzajúce sa v systémovej oblasti. Teda v priečinku */system/app,* tento priečinok je len na čítanie. To znamená, že používateľ priamo do neho nemôže inštalovať alebo z neho odinštalovávať aplikácie. Týmito aplikáciami môžu byť napríklad fotoaparát, správy, kalkulačka, a iné. Vo všeobecnosti sa tieto aplikácie nedajú odinštalovať z dôvodu možného ovplyvnenia správnej funkcionality zariadenia. Ak je potrebné takúto aplikáciu odinštalovať, musí byť zariadenie *rootované*.

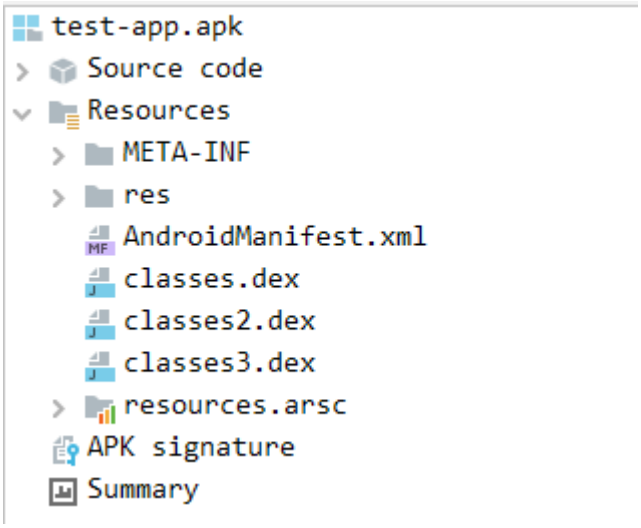
Nesystémové alebo používateľské aplikácie sú aplikácie, ktoré vyvinula tretia strana. Do zariadenia sa môžu nainštalovať cez Google Play alebo cez súbor APK. Tieto aplikácie sú uložené v priečinku */data/app* a tento priečinok má oprávnenie na čítanie aj zápis, takže tieto aplikácie môžu byť ľubovolne odinštalovené. [11]

1.3 APK a jeho štruktúra

Aplikácie systému Android sú väčšinou distribuované vo formáte APK (Android Package Kit). APK obsahuje všetky informácie a elementy, ktoré sú nutné ku správnej funkcionalite aplikácie. Súbor APK má určitú podobnosť s typom súboru JAR. Je to ZIP archív s preddefinovanou špecifickou štruktúrou, napríklad obsahuje súbor AndroidManifest.xml a ďalšie súbory typické pre Android. Preto sa v niektorých prípadoch môže stať, že je súbor APK zobrazený ako súbor typu ZIP [18]. V ďalšej časti bude teda priblížený obsah samotného APK súboru.

Obrázok 2. Zloženie APK súboru [19]

Podľa obrázku je viditeľné, že APK súbor sa skladá z niekoľkých základných komponentov. Aby bol však priblížený detailejší obsah takéhoto súboru, nižšie je zobrazený obsah dekompilovaného APK súboru. K samotnému obsahu sa však treba dostať pomocou nástrojov, ktoré budú detailnejšie rozobrané v inej časti tejto práce. Pre tento prípad je použitý nástroj nazývaný JADX. Pomocou tohoto nástroja je možné dekompilovať APK súbor a nahliadnuť tak do jeho vnútra.



Obrázok 3. Obsah APK súboru [zdroj vlastný]

1.3.1 Zdrojový kód (Source code)

Obsahuje zdrojové kódy aplikácie. Keďže každá aplikácia má svoj vlastný zdrojový kód, v tejto časti mu nebude venovaná pozornosť, nakoľko sa tento zdrojový kód líši v každej aplikácii. Pozornosť bude venovaná komponentom, ktoré aplikácie zdieľajú.

1.3.2 META-INF

V tomto súbore sa nachádza predovšetkým súbor *MANIFEST.FM,* ktorý obsahuje meno a SHA1-Digest (Secure Hash Algorithm) všetkých súborov v balíku APK. Tieto HASHe sa používajú na overenie každého súboru, to znamená, že akákoľvek zmena súboru v balíku zmení príslušný HASH, ktorý už nebude sedieť s pôvodným.
Ďalšími dôležitými súbormi sú *CERT.SF,* ktoré obsahujú zoznam všetkých súborov spolu s ich SHA1. A *CERT.RSA,* ktorý obsahuje podpísaný obsah *CERT.SF* spolu s certifikátom verejného kľúča použitého na podpísanie obsahu. [19]

1.3.3 Zdroje (Res)

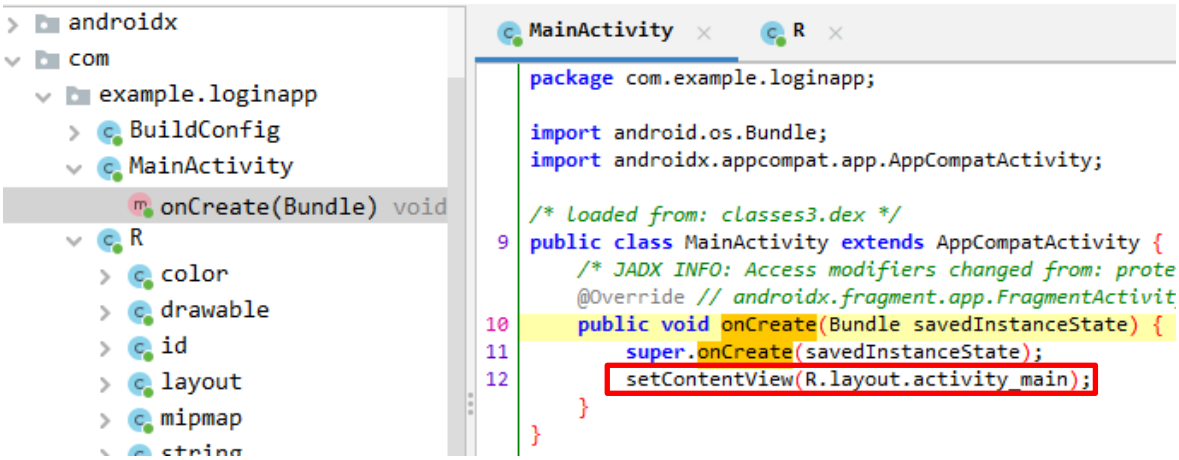
Zložka *res* obsahuje väčšinu súborov .xml, .png, .jpeg, a ďalších podobných súborov. Tieto súbory sú rozdelené do zložiek podľa ich využitia. Môžu sa tu vyskytovať zložky ako napr. /*drawable,* kde sú uložené súbory na vykreslenie, */layout,* kde sú súbory .xml, ktoré majú na starosti rozloženie alebo */values,* ktoré obsahujú informácie o hodnotách. [19]

1.3.4 AndroidManifest.xml

Každá aplikácia musí mať súbor *AndroidManifest.xml* v koreňovom adresári. Tento súbor obsahuje nevyhnutné informácie o aplikácii, ktoré sú použité na „opísanie" aplikácie operačnému systému, obchodu Google Play, atď. Okrem iného, každý súbor *AndroidManifest.xml* musí mať zadeklarované *App Components,* zahŕňajúce všetky aktivity a služby. *Permissions,* teda povolenia, ktoré aplikácia potrebuje pri prístupe do chránených častí systému alebo inej aplikácie a taktiež hardvérové a softvérové funkcie, ktoré aplikácia vyžaduje, teda *kompatibilita*. [20]

Nahliadnutie do tohoto súboru je tak vo väčšine prípadov prvým krokom k hľadaniu zraniteľností. V tomto súbore sa nachádza veľa možných nastavení a povolení, ktoré môžu útočníkovi pomôcť s hľadaním zraniteľností. Z praktického hľadiska je toto hľadanie vykonávané prechádzaním elementov <*activity*>, kde sú vyhľadávané rôzne časti kódu.

ACTION_MAIN je považovaný za tzv. *Entry point*, teda vstupný bod. Zvyčajne je skombinovaný s *CATEGORY_LAUNCHER* práve v <*intent-filter*> značke. Pokiaľ je v aplikácii nájdený *entry point,* je potrebné sa v MainActivity pozrieť na obsah metódy *onCreate,* konkrétne na *setContentView*, kde je ako parameter názov layoutu. [21]



Obrázok 4. setContentView v metóde onCreate [zdroj vlastný]

Značka *<intent-filter>*, označujúca *intent* alebo zámer, môže byť použitá na spustenie aktivity, posielanie komponentom prijímačov alebo komunikáciu s procesmi na pozadí. Obsah týchto *intents* by mal byť skontrolovaný, aby neobsahoval žiadne informácie, ktoré môžu byť zachytené. [22]

Medzi ďalšie atribúty XML elementu *application*, ktoré môžu byť využité, patria napríklad *<application android:debuggable="true" </application>*
Táto značka označuje, že aplikácia je v režime ladenia. Toto ladenie či debugging môže útočníkovi poskytnúť veľa informácií. Aplikácie, ktoré sú v stave vývoja majú tento atribút nastavenú na hodnotu *true*, pretože debugging využívajú vývojári. Ak je však aplikácia vydaná pre verejnosť, mala by mať tento atribút nastavenú hodnotu na *false*.

Nasledujúcou takouto hrozbou môže byť atribút *<application android:allowBackup="true" </application>*. Tento atribút určuje možnosť zálohy aplikácie, preto pri aplikáciách uchovávajúcich citlivé informácie môže byť veľmi ľahko zneužitý, takže by mal byť nastavený na hodnotu *false*, aby sa uniknutiu údajov touto metódou zabránilo.

Ďalšou potenciálne nebezpečnou značkou môže byť *<receiver>*, táto značka označuje prijímač prenosu ako jeden z komponentov aplikácie. Tieto prijímače umožňujú

aplikáciám získať zámery vysielané systémom alebo aplikáciami. Pri tejto značke je zaujímavý hlavne atribút *android:exported="true",* kde hodnota *true* znamená, že prijímač môže prijímať správy zo zdrojov mimo svojej aplikácie. [23]

1.3.4.1 AndroidManifest App Components

Komponenty aplikácie sú základnými stavebnými kameňmi aplikácie pre Android. Každý komponent je bodom, cez ktorý môže systém alebo používateľ vstúpiť do aplikácie. Pre každý vytvorený komponent musí byť zadeklarovaný zodpovedajúci prvok XML v súbore AndroidManifest. Základnými komponentami sú <*activity*>, <*service*>, <*receiver*>, <*provider*> ak je použitý ktorýkoľvek z týchto komponentov bez deklarovania, systém ho nemôže spustiť. [20]

1.3.4.2 AndroidManifest Permissions

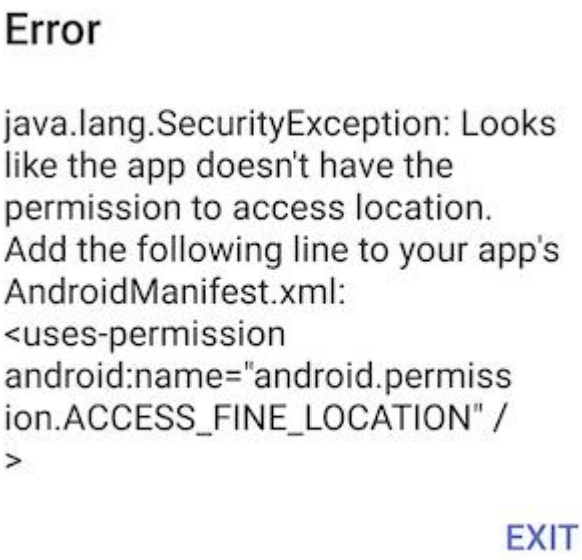
Aplikácia si musí vyžiadať povolenie k prístupu do citlivých dát úžívateľa ako sú SMS, galéria alebo k systémovým službám (prístup na internet, GPS).

Od Androidu 6.0 a vyššie si môže užívateľ určiť niektoré povolenia, avšak odhliadnuc od verzie Androidu, všetky povolenia musia byť zadeklarované pomocou <*uses-permission*> elementu v súbore AndroidManifest.

Aplikácia môže tiež chrániť jej vlastné súčasti pomocou povolení. Nové povolenie sa deklaruje pomocou prvku <*permission*>.[20]

Ak by aplikácia chcela pristupovať k nejakej časti chránenej povolením, ale toto povolenie nemá, systém jej prístup nemôže povoliť a nastane tzv. *SecurityException.*

Ide o výnimku, ktorá je rozšírením *RuntimeExceptio*n, a je vyvolaná správcom bezpečnosti z dôvodu porušenia nejakého bezpečnostého prvku, v tomto prípade nesprávne použitie povolení, alebo neudelenie povolenia. [24]



Obrázok 5. Chyba vyvolaná Security Exception [zdroj vlastný]

1.3.4.3 AndroidManifest kompatibilita

Súbor Manifest je tiež miesto, kde sa deklarujú, ktoré typy hardvéru a ktoré softvérové služby aplikácia vyžaduje, takže prakticky, s ktorými zariadeniami je aplikácia kompatibilná. Google Play obchod nedovolí aplikácii nainštalovať sa na zariadenie, ktoré tieto služby alebo systémové požiadavky nespĺňajú.

Tieto požiadavky sa deklarujú v elemente <*uses-feature*>. Príklad funkcionality môže byť napríklad aplikácia navigácie, ak vaše mobilné zariadenie neobsahuje GPS, alebo iné lokalizačné služby, prípadne hardvérové nástroje k tomu potrebné, nemalo by byť možné takúto aplikáciu nainštalovať. [20]

1.3.5 Súbor Classes.dex

Kód aplikácie je väčšinou napísaný v jazyku Java, potom je skompilovaný do CLASS súborov, ktoré sú následne konvertované do DEX (Dalvik Executable) súboru, tento je kompatibilný s Dalvikom. Formát DEX je navrhnutý tak, aby bol vhodný pre systémy obmedzené pamäťou a rýchlosťou procesora. V dnešnej dobe je už bežné, že aplikácie

obsahujú viacero *classes.dex* súborov, ktoré sa potom označujú *classes.dex, classes2.dex,...* [19]

1.3.5.1 Jazyk Smali

Aplikácie pre Android bežia v Dalvik VM alebo novšie v ART a tieto súbory sú spúšťané vo formáte DEX (Dalvik Executable). Keď je skompilovaný kód aplikácie, súbor APK obsahuje súbory .DEX, ktoré obsahujú bajtový kód Dalvik. Toto je formát, ktorému platforma rozumie. Pre človeka nie je však ľahké čítať alebo upravovať binárny kód, takže existujú nástroje na konverziu do a z ľudsky čitateľnej reprezentácie. Najbežnejší ľudsky čitateľný formát je známy ako *Smali*. Takže prakticky ide o reprezentáciu pre DEX formát.

Najbežnejšie využitie jazyka Smali je, keď je potrebné upraviť kód aplikácie, napríklad s cieľom vkladania blokov kódu alebo upravenie už existujúceho kódu. Príkladom tohoto využitia môže byť odstránenie pripnutia certifikátu z aplikácie, aby mohol byť vykonaný MITM útok alebo pridanie, či upravenie kódu.

Príklad štruktúry jazyka: na definovanie jednotlivých návratových typov metód v jazyku java sa používajú jednoznakové identifikátory

V - void
Z - Boolean
B - byte
S - short
C - char
F - float
I - int
J - long
D - double
[- array
L - class definition

Výsledná implementácia metódy *MyMethod* s návratovým typom *void* teda vyzerá takto:

Java

private static void myMethod()

smali

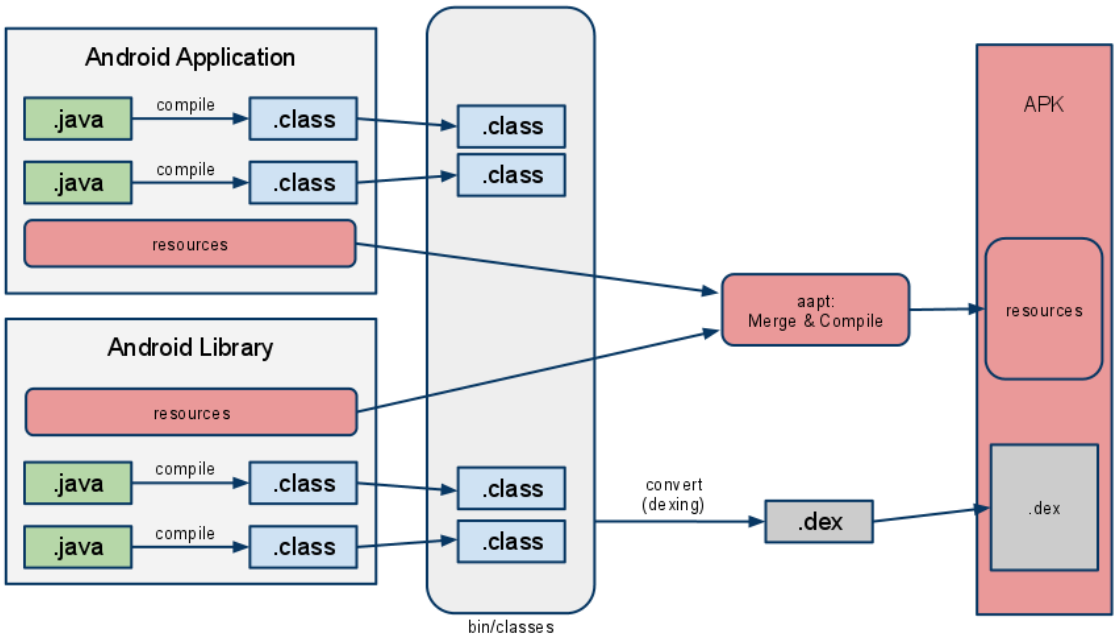
.method private static myMethod() [25]

1.3.6 Súbor Resources.arsc

Súbor *.ARSC* je tabuľka zdrojov aplikácie používaných aplikáciou.

Resources.arsc obsahuje všetky skompilované zdroje a ich identifikátory. Pri spustení aplikácia použije jej *.ARSC* súbor na rýchle lokalizovanie a prístup ku zdrojom. [19]

1.4 Proces vytvárania APK



Obrázok 6. Grafické znázornenie vytvárania APK. [26]

Celý proces začína zdrojovým kódom v jazyku Java alebo Kotlin. Keď je kód napísaný, skompiluje sa do súborov *.class*. Tieto jednotlivé *.class* súbory sú potom umiesnené do zložky *Classes.* V ďalšom kroku sa zložka classes zabalí do jedného alebo viacerých súborov s príponou *DEX*, tým sa vytvoria súbory *classes.dex*. Zdroje aplikácie sú zostavené pomocou nástroja Android Asset Packaging Tool. V tomto kroku sú zoskupené

a skompilované všetky potrebné zdroje, s ktorými aplikácia pracuje. Výstupom tohoto nástroja je súbor *.APK* obsahujúci všetky zdroje okrem kódu, teda už obsahuje *assets, res, resources.arsc* atď. Kompilovaný kód aplikácie so všetkými súbormi *class.dex* a natívnymi knižnicami sú potom tiež zabalené do súboru APK.

Súbor APK sa potom optimalizuje pomocou ZipAlign a tým sa zaistí, že všetky nekompilované údaje sú zarovnané štvorbajtovou hranicou, aby sa zlepšil výkon pri prístupe k veľkým binárnym súborom, ako sú napr. obrázky.

Nakoniec je súbor APK digitálne podpísaný, aby ho bolo možné overiť a nainštalovať na zariadenia so systémom Android. [26]

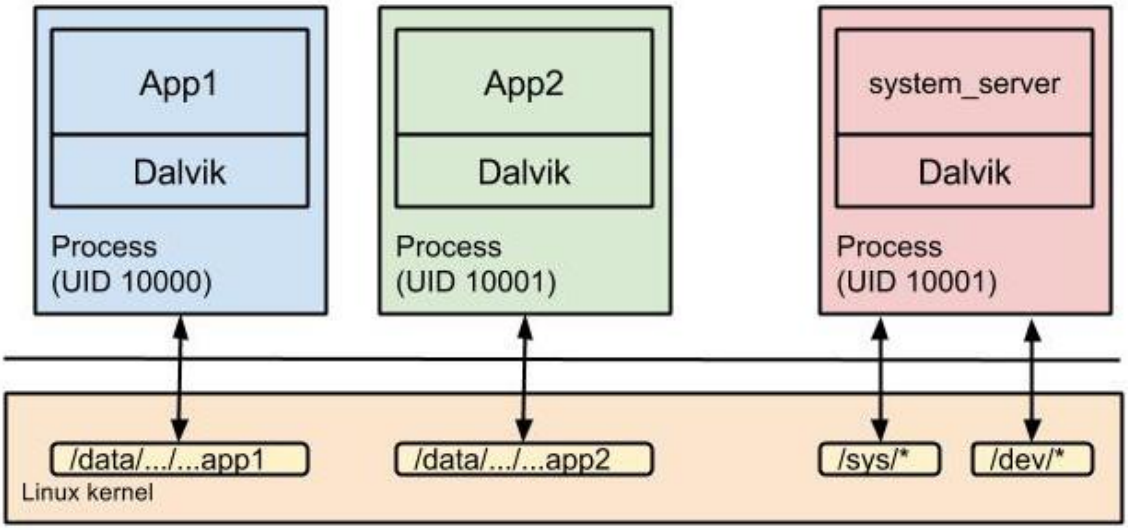
1.5 Android Security Features

Ako každý operačný systém, má aj Android svoje vlastnosti a postupy, ktoré majú za úlohu spraviť zariadenia čo najbezpečnejšie. Preto budú v tejto časti bližšie predstavené niektoré najdôležitejšie vlastnosti.

1.5.1 Aplikačný Sandbox

Android využíva Linuxovú user-based ochranu na identifikáciu a izolovanie zdrojov aplikácií. Táto izolácia aplikácií chráni ostatné aplikácie a taktiež samotný systém pred škodlivými aplikáciami. Izolácia funguje na báze pridelenia unikátneho identifikátoru(UID) každej aplikácii, ktorá je spustená vo svojom vlastnom procese. Android používa tieto UID na nastavenie izolácie aplikácií na úrovni jadra. Jadro uplatňuje bezpečnosť medzi aplikáciami a systémom na úrovni procesu prostredníctvom štandardných súčastí Linuxu, ako sú ID užívateľa a skupín, ktoré sú priradené aplikáciám. V predvolenom nastavení aplikácie nemôžu navzájom interagovať a majú obmedzený prístup k operačnému systému. Ak sa aplikácia pokúsi urobiť niečo nebezpečné alebo potenciálne škodlivé, napr. čítanie údajov od inej aplikácie bez potrebného povolenia, je jej v tom zabránené. Pretože izolácia je zabezpečená už na úrovni jadra, všetok softvér nad ním ako sú knižnice, aplikačný framework, aplikačný runtime sú tiež izolované. [27]

Výnimka však nastáva pri aplikáciách, ktoré zdieľajú rovnaké vývojárske certifikáty. Tieto môžu zdieľať aj rovnaké ID. Jedná sa o takzvané *sharedUserId.* Aplikácie, ktoré zdieľajú rovnaké ID môžu navzájom pristupovať ku svojím údajom, a ak je to potrebné, aj bežať v rovnakom procese. [28]



Obrázok 7. Grafické znázornenie Sandboxovania. [29]

Bezpečnosť sandboxingu však môže byť ohrozená. Podobne ako pri iných bezpečnostných funkciách, samostatné ochrany zabezpečujúce izoláciu aplikácií niesú nezraniteľné. Preto je dôležitá hĺbková ochrana, aby sa predišlo tomu, že jednotlivé zraniteľnosti vedú ku kompromitácii operačného systému alebo iných aplikácií. Android sa pri izolácii spolieha na množstvo ochrán, ktoré boli uvedené v priebehu času, aby posilnili pôvodnú izoláciu diskrečnej kontroly prístupu (DAC) založenej na UID.

V systéme Android 5.0. poskytoval SELinux (Security Enhanced Linux) oddelenie Mandatory Access Controll (MAC), teda povinného riadenia prístupu, medzi systémom a aplikáciami. Všetky aplikácie tretích strán však bežali v rovnakom kontexte SELinux, takže izoláciu medzi týmito aplikáciami zaisťoval UID DAC.

Android 6.0 poskytoval rozšírenie izolácie tak, aby SELinux izoloval aplikácie cez *per-physical-user* hranicu, teda oddelenie na základe používateľov. Taktiež boli nastavené bezpečnejšie predvolené hodnoty pre dáta aplikácie, napr. pre aplikácie s *targetSdkVersion >=24*, boli predvolené hodnoty DAC povolenia v domovskom priečinku aplikácie zmenené z 751 (vlastník má plný prístup, skupina vlastníka môže čítať a otvárať adresár, ostatní môžu len otvárať adresár) na 700 (vlastník má plnú kontrolu, ostatní nemôžu robiť nič), čo poskytlo bezpečnejšie predvolené hodnoty pre súkromné dáta aplikácie.

Pri zavedení Android 8.0 boli všetky aplikácie spúšťané s *seccomp-bpf* filtrom, ktorý obmedzuje systémové volania, ktoré mohli aplikácie používať a tým sa posilnila hranica medzi aplikáciami a jadrom.

V systéme Android 9.0 musia všetky neprivilegované aplikácie s *targetSdkVersion >= 28* bežať v samostatných izoláciách SELinux, ktoré poskytujú Mandatory Access Controll (MAC) pre každú aplikáciu. [27]

Ak je zariadenie rootované, princíp sandboxingu je porušený. To znamená, že je možné dostať sa k jej chráneným častiam, a zároveň je možné dostať sa do sandboxu inej aplikácie. Napríklad útočník môže vidieť obsah zložky */data/data*. Toto je privátny dátový priestor pre aplikácie a bežný používateľ do neho nemá prístup. Ak je však na zariadení v režime root príkazom *ls –lah* (kde parameter *l* znamená dlhý výpis zo všetkými detailami, parameter *a* znamená výpis všetkých položiek vrátane skrytých, a parameter *h* znamená zobrazenie v tzv. *Human readable* režime) v ADB Shell zobrazený obsah zložky */data/data,* je možné vidieť zoznam aplikácií, z ktorých každá tu uchováva svoje privátne dáta.

```
drwxr-x--x  7 u0_a44    u0_a44    4.0K 2022-04-10 03:33 jakhar.aseem.diva
```

Obrázok 8. Údaje o aplikácii v /data/data vypísané príkazom ls -lah. [zdroj vlastný]

Pri pohľade na konkrétnu aplikáciu, je vidno v prvom stĺpci oprávnenia. Tento zápis je *Unixový* a skladá sa z 10 segmentov, z ktorých každý reprezentuje určitý význam následovne. Prvé písmeno *d* značí že daný obsah je *directory,* teda zložka resp. adresár, ak by sa jednalo o *súbor*, označenie na prvom mieste by bolo -. Ďalej treba postupovať vždy

po trojiciach znakov, prvá trojica od miesta 2 po miesto 4 značí povolenia vlastníka súboru písmenami *r,w,x,* kde *r* znamená *read,* teda čítať obsah tejto zložky, *w,* teda *write* znamená, že môže zapisovať do tejto zložky. A nakoniec *x,* teda *execute* môže otvárať obsah zložky. Ďalšie dve trojice fungujú na rovnakom princípe s tým, že druhá označuje práva skupiny vlastníka a posledná práva všetkých ostatných. V zhrnutí sa dá povedať že daný obsah je adresár a jej vlastník môže zapisovať, čítať a otvárať, skupina vlastníka len otvárať a všetci ostatný len otvárať.

V ďalšom stĺpci je vidno číslo 7, toto číslo hovorí o počte odkazov na tento súbor. Tretí stĺpec označuje vlastníka súboru alebo adresára. Štvrtý stĺpec hovorí o tom, kto je vlastníkom skupiny. V piatom stĺpci je vidno veľkosť súboru, tu sa prejavil parameter *h,* kde je badateľné, že miesto čísla bajtov, ktoré by sa zobrazili defaultne, sa zobrazil údaj veľkosti v kilobajtoch, čo je prospešné hlavne pri väčších súboroch. Šiesty stĺpec hovorí o dátume poslednej úpravy a posledný, teda siedmy stĺpec zobrazuje názov adresára. [30]

```
1|vbox86p:/data/data # cd jakhar.aseem.diva/
vbox86p:/data/data/jakhar.aseem.diva # ls -lah
total 28K
drwxr-x--x  7 u0_a44 u0_a44 4.0K 2022-04-10 03:33 .
drwxrwx--x 87 system system 4.0K 2022-04-05 07:25 ..
drwxrwxrwx  3 u0_a44 u0_a44 4.0K 2022-03-24 13:04 Inspeckage
drwxrwx--x  2 u0_a44 u0_a44 4.0K 2022-03-24 11:56 cache
drwxrwx--x  2 u0_a44 u0_a44 4.0K 2022-03-24 11:56 code_cache
drwxrwx--x  2 u0_a44 u0_a44 4.0K 2022-03-24 11:56 databases
lrwxrwxrwx  1 root   root     37 2022-04-10 03:33 lib -> /data/app/jakhar.aseem.diva-1/lib/x86
drwxrwx--x  2 u0_a44 u0_a44 4.0K 2022-03-24 13:31 shared_prefs
vbox86p:/data/data/jakhar.aseem.diva #
```

Obrázok 9. Obsah aplikácie v jej privátnom dátovom priestore. [zdroj vlastný]

Po otvorení aplikácie je viditeľný celý obsah tohoto adresára a ak by sa v ňom nachádzali nejaké citlivé informácie, útočník sa k nim môže dostať a ukradnúť ich.

1.5.2 Podpisovanie aplikácií

Podpisovanie aplikácií umožňuje identifikovať autora aplikácie a tiež updatovať ich aplikáciu bez potreby vytvárať komplikované rozhrania a povolenia. Každá aplikácia bežiaca na platforme Android musí byť podpísaná jej developerom. Ak sa aplikácia bez

podpisu pokúsi o nainštalovanie, bude zamietnutá buď obchodom Google Play alebo samotným inštalátorom na Android zariadení.

Keď sa APK súbor nainštaluje na Android zariadenie, Package manager overí, či je aplikácia správne podpísaná a obsahuje certifikát. Certifikát obsahuje verejný kľúč a ak tento kľúč súhlasí s kľúčom, ktorý bol použitý na podpísanie aplikácie na zariadení, nová aplikácia má možnosť špecifikovať to vo svojom súbore Manifest, ktorý zdieľa UID s ostatnými podpísanými aplikáciami.

Aplikácia môže byť podpísaná tiež treťou stranou alebo vlastnoručne. Android poskytuje podpisovanie kódu použitím vlastnoručne podpísaných certifikátov, ktoré môže developer vygenerovať s externou pomocou alebo povolením. Aplikácie sú tiež schopné deklarovať bezpečnostné povolenia na úrovni ochrany podpisu, obmedzujúce prístup iba aplikáciám podpísaných rovnakým kľúčom, zatiaľ čo si zachovajú odlišné UID a aplikačnú izoláciu.

V prípade že sú dve alebo viac aplikácií podpísané rovnakým vývojárskym kľúčom, operačný systém prídelí všetkým týmto aplikáciám rovnaké UID. Pri tomto zdieľanom UID si potom môžu aplikácie navzájom pristupovať do svojich zdrojov.

Android poskytuje tri základné schémy podpisov. Schému v1, schému v2 a schému v3. [33]

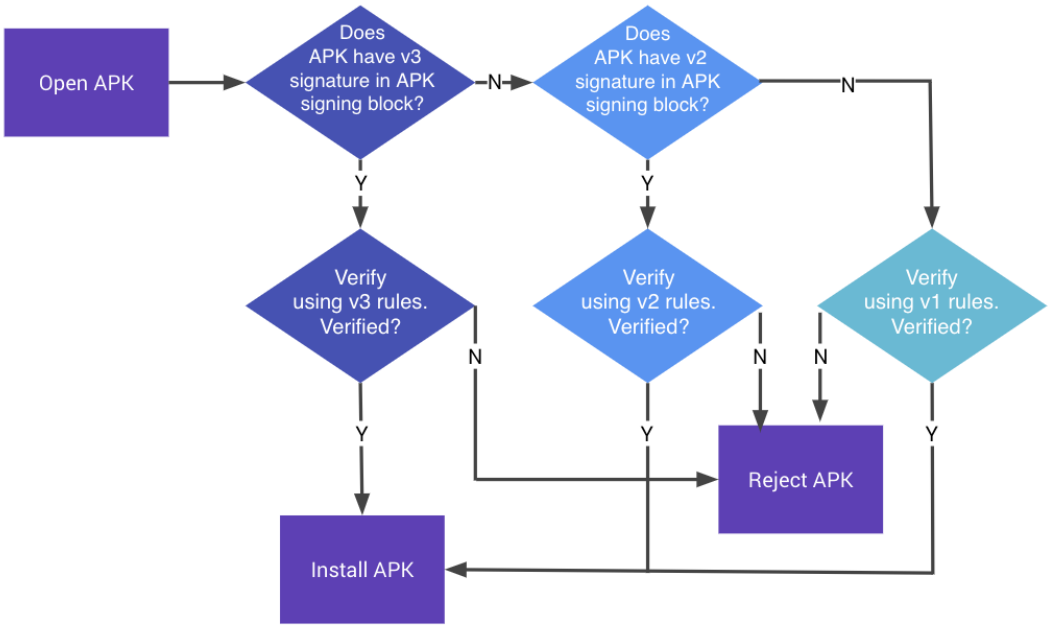
1.5.2.1 Schéma v1 (JAR signing)

Od počiatku Androidu bolo podpisovanie jeho súčasťou. Je založené na podpísaných JAR súboroch (Úspešné overenie JAR súboru prebehne vtedy, ak je podpis platný a žiadne súbory, ktoré sú v JAR neboli zmenené). V1 podpisy však nedokážu ochrániť niektoré časti APK, ako napríklad metadáta ZIPu. Takže overovač APK musí spracovať veľa neoverených dát a potom zmazať tie, ktoré niesú podpísané a to ponúka veľký priestor pre potenciálne útoky. Okrem toho musí overovač dekomprimovať všetky komprimované záznamy, čo zaberie veľa času a pamäte. [33]

1.5.2.2 Schémy v2, v3(v2+)

Od systému Android 7.0 a vyššie, je podporovaná podpisová schéma v2. Táto schéma funguje na princípe hashov. Obsah súboru je zahashovaný a podpísaný, následne sa výsledný blok na podpis súboru APK vloží do súboru APK.

Schéma v3 označovaná aj ako v2+ pracuje so súborom APK ako s blokom a vykonáva kontrolu podpisu v celom súbore. Akákoľvek úprava súboru APK vrátane metadát ZIPu znehodnotí podpis. Táto schéma je podstatne rýchlejšia a umožňuje detekciu rôznych neoprávnených úprav. Tento formát podpisu je spätne kompatibilný, takže APK podpísané novým formátom je možné nainštalovať aj na staršie Android zariadenia. [33]



Obrázok 10.Vývojový diagram overovania podpisu APK [33]

1.5.3 Autentifikácia

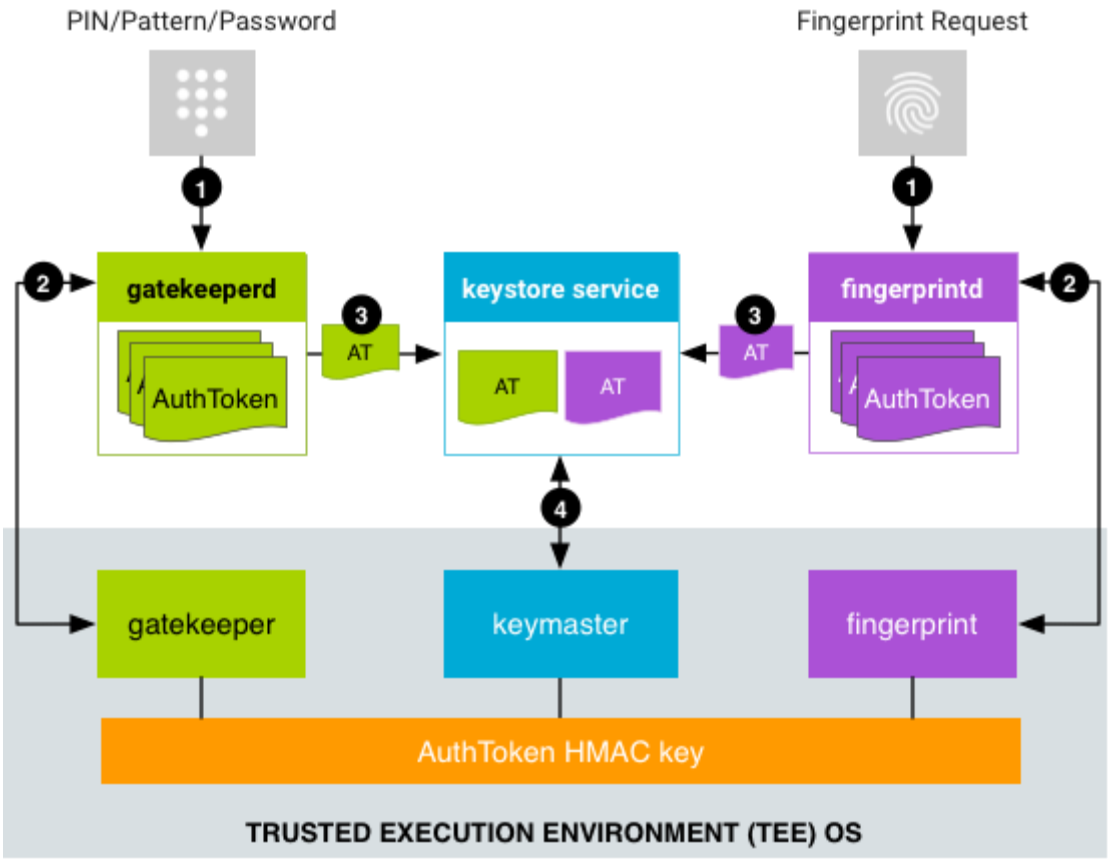
Android používa koncepciu kryptografických kľúčov s overením používateľa, ktorá vyžaduje *Cryptographic key storage and service provider* a *User authenticators*. Na zariadeniach disponujúcich snímačom odtlačkov prstov, môže užívateľ nastaviť jeden

alebo viacero odtlačkov, ktoré odomknú zariadenie. Podsystém *Gatekeeper* vykoná overenie vstupu (hesla, vzoru,...) v prostredí *Trusted Execution Environment (TEE).*

Cryptographic key storage and service provider ukladá kryptografické kľúče a poskytuje štandardné kryptografické služby. Android podporuje *Hardware-Backed Keystore,* čo je dostupnosť TEE prostredia na čipe (System on chip). Toto poskytuje Androidovým zariadeniam vykonávať hardvérovo podporované, silné bezpečnostné služby pre systém Android, platformové služby a dokonca aj aplikácie tretích strán. Taktiež Android podporuje aj *Keymaster* pre kryptografické služby, vrátane hardvérovo podporovanej kryprografie pre uloženie kľúčov, ktoré môžu zahŕňať TEE alebo Secure Element(SE), ako napríklad *StrongBox.*

User authenticators potvrdzujú prítomnosť užívateľa a úspešnú autentifikáciu. Gatekeeper overuje heslá pomocou HMAC(keyed-Hash Message Authentication Code) s hardvérovo podporovaným tajným kľúčom. Gatekeeper je teda používaný pre overenie PINu, vzoru alebo hesla. Na overenie odtlačku prsta a podobné biometrické vstupy sa používajú iné metódy, ktoré budú podrobnejšie opísané v odstavci Biometrika.

Keď používateľ nastaví heslo a dostane SID, môže spustiť autentifikáciu, ktorá sa začne zadaním PINu, hesla, vzoru alebo biometrického vstupu. Všetky komponenty TEE zdieľajú HMAC kľúč, ktorý používajú na vzájomné overenie. [32]



Obrázok 11. Proces overovania vstupov [32]

1.5.3.1 Postup pri zápise autentikácie

Po prvom spustení (alebo po uvedení zariadenia do továrenských nastavení) sú všetky autentifikátory pripravené na prijatie vstupu od užívateľa. Najskôr však musí byť vložený PIN, vzor alebo heslo do Gatekeepera. Tento počiatočný zápis vygeneruje 64-bitový Secure Identifier (SID), ktorý slúži ako identifikátor a zároveň väzobný token pre kryptografický materiál užívateľa. Toto SID je viazané na heslo, ktoré používateľ zadal. Výsledkom úspešnej autentifikácie do Gatekeepera sú tzv. *AuthTokeny*, ktoré obsahujú SID používateľa.

Ak chce používateľ zmeniť heslo, musí predložiť existujúce heslo. Ak je teda existujúce heslo úspešne overené, potom sa používateľove SID, asociované s existujúcim heslom prenesie na nové heslo, umožňujúc tak prístup úžívateľa k jeho dátam po zmene hesla. Ak

užívateľ nezadá existujúce heslo, nové heslo sa zapíše s náhodným používateľským SID. Toto rezultuje do situácie, kedy síce užívateľ má prístup k zariadeniu, ale stratí všetky kľúče vytvorené pod starým SID. Tento zápis je známy ako *untrusted enroll (nedôveryhodný zápis).* Za normálnych okolností však framework Androidu nedovolí nedôveryhodný zápis, takže väčšina užívateľov neuvidí túto funkciu, avšak môže byť vynútená tvrdým resetom hesla buď zo strany administrátora alebo útočníka. [32]

1.5.3.2 Biometrika

Biometrika ponúka pohodlnejšiu ale potenciálne menej bezpečnú cestu overenia identity. Vo viacúrovňovom modeli autentifikácie poskytuje primárna autentifikácia (PIN, heslo, vzor) najvyššiu úroveň zabezpečenia. Biometrika je na sekundárnej úrovni autentifikácie a ponúka rovnováhu medzi bezpečnosťou a pohodlím.

Biometrické senzory môžu byť rozdelené do troch tried a to trieda 3 (silná), trieda 2 (slabá), trieda 1 (pohodlná), na základe miery ich akceptácie hrozieb a na základe bezpečnosti biometrického pipeline. Táto klasifikácia určuje schopnosti, ktoré má biometrický senzor na prepojenie s platformou a aplikáciami tretích strán. Štandardne sú biometrické senzory klasifikované ako trieda 1, ak chcú byť klasifikované do vyššej triedy, musia spĺňať ďalšie požiadavky. Všetky tri triedy sa môžu integrovať so zamknutou obrazovkou ale iba silné a slabé (triedy 2 a 3) autentifikátory môžu integrovať s *android.hardware.biometrics* API. Trieda 2 a 3 môžu integrovať s BiometricPrompt, a trieda 3 má navyše aj možnosť ukladať časové kľúče a kľúče založené na operácii. Aby developer zabezpečil, že používatelia budú môcť bez problémov používať biometriku, musia integrovať svoj biometric stack s *BiometricPrompt, BiometricManager* a *ACTION_BIOMETRIC_ENROLL*. [33]

1.5.4 Šifrovanie (Encryption)

Šifrovanie je prostriedok na zabezpečenie údajov používateľa na zariadení s operačným systémom Android, pomocou symetrických šifrovacích kľúčov. Po zašifrovaní zariadenia sa všetky údaje, ktoré vytvoril používateľ zašifrujú predtým, ako sa odovzdajú do úložiska a pri čítaní sa znova automaticky dešifrujú ešte pred ich vrátením do procesu volania. Šifrované údaje zaistia, že v prípade zisku údajov neoprávnenou osobou, nebude môcť

tieto informácie prečítať. Android má dva základné typy šifrovania a to *file-based encryption* a *full-disk encryption.*

file-based encryption je k dispozícii od uvedenia Android 7.0, toto šifrovanie súborov umožňuje rôznym súborom byť zašifrované rôznymi kľúčmi, ktoré môžu byť nezávisle odomknuté. Zariadenia, ktoré podporujú tento štýl šifrovania môžu tiež podporovať *Direct Boot,* ktorý dáva šifrovaným zariadeniam možnosť priameho bootu až na uzamknutú obrazovku, takže je možný rýchly prístup k dôležitým funkciám.

full-disk encryption Android 5.0 až Android 9 podporujú šifrovanie celého úložiska. V tomto type šifrovania je použitý jeden kľúč chránený heslom v užívateľovom zariadení. Takže aby sa docielilo úplnej ochrane používateľových dát v zariadení, pri bootovaní musí užívateľ zadať heslo, aby sa sprístupnila akákoľvek časť úložiska. Tento spôsob je veľmi efektívny z hladiska bezpečnosti, pretože väčšina funkcionality jadra zariadenia nie je okamžite dostupná, keď nie je zariadenie plne spustené. Keďže prístup ku všetkým dátam je chránený heslom užívateľa, nie je možné fungovanie ani základných funkcií ako budík alebo prijímanie hovorov. Šifrovanie celého úložiska je založené na *dm-crypt,* teda funkcii jadra, ktorá funguje na block device vrstve, vďaka tomu full-disk šifrovanie funguje aj s Embedded MultiMediaCard(eMMC) a podobnými flash zariadeniami. Šifrovací algoritmus využívaný pri tomto type šifrovania je 128-bitový AES s block-cipher reťazením (CBC) a ESSIV:SHA256 (Encrypted Salt-Sector Initialization Vector s dosadením 256-bitového Secure Hash Algoritmu).

Metadata encryption tento spôsob bol predstavený uvedením Androidu 9, kde má tiež podporu potrebného hardvéru. Tento spôsob šifrovania doplňuje šifrovanie súborov. Pri šifrovaní metadát je jeden kľúč prítomný pri štarte. Tento kľúč je chránený Keymasterom, ktorý je naopak chránený funkciou *Verified Boot,* ktorá bude popísaná neskôr. Šifrovanie metadát je možné nastaviť len pri prvom formátovaní oddielu, to znamená, že táto funkcia je len pre nové zariadenia. Toto šifrovanie vyžaduje povolenie modulu *dm-default-key* v jadre. [34]

1.5.5 SELinux

Súčasťou modelu zabezpečenia Androidu je aj Security-Enhanced Linux(SELinux). Je používaný na vynútenie Mandatory Access Controll (MAC) vo všetkých procesoch, dokonca aj v procesoch spustených pomocou root/superuser oprávnení. Vďaka SELinuxu môže systém lepšie chrániť systémové služby, kontrolovať prístup k systémovým protokolom a údajom aplikácií. SELinux má dva globálne režimy, a funguje na princípe predvoleného odmietania, tzn. Všetko čo nieje povolené, je odmietnuté.

Permisívny režim je režim, kde sa odmietnutia povolení zaznamenávajú ale nevynucujú. *Režim vynucovania* je režim, v ktorom sa odmietnutia povolení zaznamenávajú aj vynucujú. V režime vynucovania taktiež obsahuje bezpečnostnú politiku, ktorá štandardne funguje v rámci AOSP(Android Open Source Project). V režime vynucovania sú odmietnutia zakázané a všetky porušenia sú zaznamenané jadrom v *dmesg* a *logcat.* SELinux tiež podporuje *per-domain permissive mode,* čo je doménový permisívny mód, kde môžu byť špecifikované povolené domény, pričom zbytok systému sa umiestni do režimu globálneho vynucovania. Doména je označenie identifikujúce procesy v bezpečnostnej politike, kde sa so všetkými procesmi označenými rovnakou doménou prostupuje rovnako. Tento režim tak umožňuje inkrementálnu aplikáciu SELinuxu na čoraz väčšiu časť systémov. [35]

1.5.6 Technológia Trusty TEE

Trusty je bezpečný operačný systém, ktorý poskytuje Trusted Exexution Environment (TEE), teda prostredie pre dôveryhodné spustenie pre Android. Trusty OS beží na rovnakom procesore ako Android OS, avšak Trusty je izolovaný od zvyšku systému hardvérovo aj softvérovo. Tieto dva operačné systémy fungujú paralelne, takže Trusty má prístup k plnému výkonu procesora a pamäte aj keď je úplne izolovaný. Trusty je chránený touto izoláciou pred škodlivými aplikáciami, ktoré môže užívateľ nainštalovať a tiež proti potenciálnym zraniteľnostiam systému Android. Trusty je kompatibilný s procesormi ARM a Intel. Aplikácia Trusty je definovaná ako zbierka binárnych súborov, binárny manifest a kryptografický podpis. Počas behu Trusty, aplikácie bežia ako izolované procesy v privilegovanom móde pod Trusty kernelom.

Pri zariadeniach, ktoré majú implementované TEE, sa často hlavný procesor označuje ako nedôveryhodný, takže nemá prístup k určitým oblastiam pamäte RAM, hardvérovým registrom a write-once „poistkám“, ktoré obsahujú tajné údaje od výrobcu, ako napr. kryptografické kľúče špecifické pre zariadenie. Ako príklad fungovania môže byť uvedený rámec DRM. Softvér, ktorý beží na procesore TEE má prístup ku špecifickým kľúčom zariadenia, ktoré sú potrebné na dešifrovanie chráneného obsahu. Hlavný procesor vidí len tento zašifrovaný obsah a tak je zabezpečená vysoká úroveň ochrany pred softvérovými útokmi. [36]

1.5.7 Overený Boot (Verified Boot)

Cieľom tejto funkcie je zabezpečiť, aby všetok spustiteľný kód bol od overeného zdroja a nie od útočníka. Verified boot vytvára úplný reťazec dôvery, od hardvérovo chráneného koreňa až po zavádzač, zavádzacie oddiely a iné overené oddiely ako napr. *system, vendor, oem* atď. Pri spúštaní zariadenia každá fáza overuje integritu ďalšej fázy pred odovzdaním vykonania.

Overený boot vyžaduje kryptografické overenie všetkého spustiteľného kódu a dát, ktoré sú súčasťou verzie systému Android, ktorá sa zavádza. Obsahuje jadro načítané z *boot* partície, strom zariadení načítaných z *dtbo* partície, systémovú partíciu, atď. Malé oddiely akými sú *boot* alebo *dtbo* sa čítajú len raz a zvyčajne sa overujú načítaním celého obsahu do pamäte, kde sa potom vypočíta hash. Tento hash sa potom porovná s očakávaným hashom, a pokiaľ sa tieto nezhodujú, Android sa nenačíta. Pri väčších oddieloch, ktoré sa nezmestia celé do pamäte, sa môže použiť hashovací strom, kde je overovanie nepretržité, a prebieha pri načítaní údajov do pamäte. V tomto prípade sa koreňový hash stromu vypočíta počas behu a porovná sa s očakávanou hodnotou koreňového hashu. Android obsahuje ovládač *dm-verity*, ktorý slúži na overenie väčších partícií. Ak sa koreňový hash v určitom bode nezhoduje s očakávaným, údaje sa nepoužijú a systém prejde do chybového stavu.

Očakávané hodnoty hashov, s ktorými sa pri verified boote pracuje, sú väčšinou uložené buď na konci alebo na začiatku každej overenej partície alebo vo vyhradenej partícii (prípadne v oboch zároveň). Rozhodujúci je priamy alebo nepriamy podpis týchto hashov od *root of trust*, čo je koreň dôvery.

Aj s kompletne zabezpečeným procesom aktualizácie je možné, že neperzistentný exploit jadra Androidu manuálne nainštaluje staršiu a zraniteľnejšiu verziu Androidu alebo sa reštartuje do zraniteľnej verzie a potom ju použije na inštaláciu trvalého exploitu, odkiaľ má útočník kontrolu nad zariadením a môže s ním robiť prakticky čokoľvek. Ochrana proti tomuto typu útoku sa volá *Rollback Protection* a zvyčajne je implementovaná tak, že na zaznamenanie najnovšej verzie systému a odmietnutie nainštalovania nižšej verzie, používa *tamper-evident* úložisko. Toto úložisko je navrhnuté tak, aby vedelo detekovať neautorizovaný prístup do zariadenia. [37]

1.6 Penetračné testovanie

Penetračné testovanie je spôsob, ako vyhodnotiť bezpečnosť aplikácie pomocou využívania akýchkoľvek bezpečnostných zraniteľností alebo dier v systéme. Toto testovanie môže byť prevedené automaticky alebo manuálne. Tento proces zahŕňa zbieranie informácií o cieli ešte pred testom, identifikovanie možných entry pointov, teda vstupných bodov alebo rôznych iných príznakov a následne pokus o využitie nájdených poznatkov s cieľom dostať sa ku zraniteľným častiam aplikácie. Penetračné testovanie je komplexné testovanie zabezpečenia aplikácie a má určitú definovanú štruktúru. Prvým bodom je príprava. V tejto príprave by mal byť definovaný rozsah testovania, ciele testovania a podobne. Ďalším bodom je tzv. Intelligence gathering, ktorý môže predstavovať analýzu architektonického kontextu aplikácie, aby tester porozumel jej kontextu a funkčnosti. Následne by mala byť aplikácia zmapovaná a na základe informácií z predchádzajúcich fáz spojených s informáciami, ktoré boli získané statickou alebo dynamickou analýzou, môže byť dôkladne pochopená aplikácia a získaný prehľad o možných zraniteľnostiach. Následne prichádza fáza zvaná zneužitie. V tejto fáze sa tester pokúša preniknúť do aplikácie pomocou zraniteľností, ktoré boli nájdené. Posledná fáza je hlásenie alebo report. V tejto fáze by mal tester zostaviť výslednú správu o aplikácii, nájdených zraniteľnostiach a údajoch, ku ktorým sa dostal neoprávnene. Každá aplikácia môže mať svoje vlastné definície citlivých dát. Vo všeobecnosti sa však za najdôležitejšie považujú: overovacie údaje (heslá, PIN kódy), osobne identifikovateľné informácie (čísla platobných kariet, zdravotné údaje, a pod.), identifikátory zariadení, citlivé údaje, ktorých

únik by mohol viesť k poškodeniu mena spoločnosti, ktorá aplikáciu vyvíja a údaje, ktoré sú chránené zákonnou povinnosťou. [38]

1.6.1 Typy penetračného testovania

Penetračné testovanie sa najzákladnejšie delí podľa poznatkov, ktoré má útočník (tester), k dispozícii.

Black-Box Testing je testovanie, pri ktorom tester nemá žiadne informácie o aplikácii, ktorú testuje. Tento proces sa nazýva aj „zero-knowledge testing“, teda testovanie bez vedomostí. Hlavným cieľom tohoto testu je nechať testera, aby napodobnil správanie reálneho útočníka, ktorý si môže nájsť verejne dohladateľné informácie a následne ich použiť.

White-Box Testing pri tomto type testovania je použitý presný opak black-box testovania a teda tester má k dispozícii všetky potrebné informácie, zdrojové kódy, dokumentáciu, atď. Tento spôsob testovania môže byť oveľa rýchlejší vďaka transparentnosti a vedomostiam, s ktorými môže tester vybudovať sofistikovanejšie testy.

Gray-Box Testing je testovanie, ktoré nespadá do žiadnej z týchto skupín a je tak na pomedzí. To znamená, že testerovi sú poskytnuté niektoré informácie (väčšinou len prístupové údaje) a ostatné informácie musia byť objavené. Toto testovanie je teda kompromisom z hľadiska ceny, rýchlosti, rozsahu a počtu testov, ktoré musia byť vykonané. Preto je tento typ testovania najpoužívanejši v oblasti bezpečnosti. [39]

1.6.2 Statická a Dynamická analýza

Analýza zraniteľností je proces, pri ktorom sa hľadajú zraniteľnosti aplikácie, môže byť vykonaná manuálne, avšak na identifikáciu najzávažnejších zraniteľností sa používajú automatické skenery.

Statická analýza alebo Static Aplication Security Testing (SAST), zahŕňa skúmanie komponentov a zdrojového kódu aplikácie bez spustenia a môže byť buď automatická alebo manuálna. Zdrojový kód je kontrolovaný kvôli zabezpečeniu správnej implementácie bezpečnostných kontrol a vo vačšine sa používa hybridný prístup, teda z časti manuálny a z časti automatický. Nemusí sa však jednať len o zdrojový kód, testovanie môže zahŕňať

zdroje (resources) aplikácie a podobne. V prípade analýzy kódu sa tester dostane ku zdrojovému kódu ale aj ku kódu v jazyku Smali, ktorý je možné modifikovať, ako bude demonštrované v jednej z praktických ukážok.

Pri manuálnej kontrole kódu tester manuálne analyzuje zdrojový kód mobilnej aplikácie a snaží sa nájsť chyby zabezpečenia. Môže tak vykonať rôznymi spôsobmi, či už vyhľadávania kľúčových slov alebo skúmaním každého riadku kódu. Bežný prístup k manuálnej analýze zahŕňa nájdenie identifikátorov zraniteľností vyhľadávaním určitých rozhraní API, kľúčových slov alebo volania metód súvisiacich s databázou ako napr. *executeQuery*. Kód obsahujúci tieto reťazce je potom dobrým štartom pre manuálnu analýzu. Manuálna analýza dokáže odhaliť aj zraniteľnosti pri ktorých je kód síce technicky bezpečný, avšak môže obsahovať rôzne porušenia noriem, obchodnej logiky a podobne, ktoré automatické nástroje nedokážu odhaliť. Na druhej strane manuálna kontrola vyžaduje odborníka ovládajúceho programovací jazyk, štruktúru a ďalšie špecifické informácie používané mobilnými aplikáciami, preto môže byť manuálna kontrola veľmi časovo náročná a únavná úloha.

Automatická analýza pomocou nástrojov urýchluje tento proces SAST. Nástroje kontrolujú kód na základe preddefinovaných pravidiel alebo osvedčených postupov. Po skončení analýzy tento nástroj zobrazí zoznam chýb, varovaní a informácií o zistených porušeniach. Niektoré nástroje na statickú analýzu bežia iba pri aplikácii, ktorá je skompilovaná, niektoré musia mať k dispozícii zdrojový kód a iné fungujú len ako moduly pre analýzu vo vývojovom prostredí. Nástroje statickej analýzy obsahujú veľa informácií o pravidlách potrebných pre analýzu mobilných aplikácií, preto môžu produkovať veľa falošných poplachov, obzvlášť ak nie sú nakonfigurované pre cieľové prostredie.

Dynamická analýza Dynamic Aplication Security Testing (DAST) je testovanie aplikácií uprostred ich behu v reálnom čase. Hlavným cieľom dynamickej analýzy je nájsť zraniteľné miesta v programe počas jeho spustenia. V tomto type analýzy je hlavným bodom pozorovania správanie aplikácie, samotný zdrojový kód sa neanalyzuje. Dynamická analýza je vykonávaná aj proti backendovým službám a API, kde môžu byť analyzované odozvy. Taktiež sa táto analýza vykonáva v kontrolovanom prostredí. Tým môžu byť napríklad emulátory, buď dostupné zdarma alebo zákazkové, ktoré sú však veľmi

nákladné. Zlatou strednou cestou v tomto prípade sú platené emulátory, ktoré poskytujú vyšší štandard oproti bezplatným. V niektorých prípadoch sa tieto emulátory dajú nájsť vo forme skúšobných verzií alebo bezplatných community verziách pre osobné použitie, týmto je aj emulátor používaný na praktické úkážky v tejto práci, zvaný Genymotion.

DAST je vzyčajne používaná na skontrolovanie bezpečnostných mechanizmov, ktoré poskytujú spolahlivú ochranu proti najbežnejším typom útokov, ako napr. chyba konfigurácie servera alebo zverejnenie údajov počas prenosu.

Nedostatočná citlivosť automatických nástrojov na kontext aplikácie je problém, pri ktorom môže nastať identifikovanie potenciálnych problémov, ktoré su irelevantné. Takéto výsledky sa nazývajú *false positives.* Bežne sú hlásené zreniteľnosti, ktoré sú zneužiteľné vo webovom prehliadači, ale v mobilnej aplikácii nepredstavujú hrozbu, toto môže vznikať pretože automatické nástroje používané na sken backendovej služby sú založené na bežných webových prehliadačových aplikáciách. Problémy ako *Cross-Site Request Forgery* (CSRF) alebo *Cross-Site Scripting* (XSS), sú podľa tohoto aj hlásené. [40]

1.7 Zraniteľnosti (Vulnerabilities)

Mobilné aplikácie ako každý iný softvér nie sú dokonalé, vždy sa nájdu nejaké zraniteľné miesta, ktoré môžu byť zneužité útočníkom, ktorý sa snaží získať citlivé dáta užívateľov, alebo inak ohroziť fungovanie danej aplikácie. Niektoré typy zraniteľností sú však častejšie a nebezpečnejšie ako iné a práve na tomto zoradení pracovala významná nezisková organizácia OWASP, ktorá zostavila rebríček najzávažnejších zraniteľností. [41]

1.7.1 Zranitelnosť Improper platform usage

Nesprávne použitie platformy prakticky znamená zneužitie nejakej vlastnosti alebo ovládacích prvkov danej platformy. V súčasnej dobe sú hlavnými platformami, resp. mobilnými operačnými systémami Android a iOS. Keďže vývojári sa snažia vyvinúť aplikáciu tak, aby ju mohlo používať čo najviac ľudí, snažia sa ju sprístupniť na čo najširšie spektrum zariadení. Toto však so sebou prináša problém, pretože každá z platforiem vyžaduje špecifickú implementáciu. Preto by teoreticky museli vývojári

programovať aplikáciu pre každú platformu zvlášť. Druhým spôsobom je vývoj takzvanej *cross-platform* aplikácie, teda aplikácie fungujúcej na viacerých typoch platforiem súčasne, ktorá má za cieľ pokryť čo najvačší počet koncových zariadení. Pri oboch týchto typoch však môže nastať táto zraniteľnosť, ak aplikácia nesprávne implementuje bezpečnostné funkcie. Akákoľvek odhalená služba alebo volanie API môžu byť zneužité. Prostredníctvom týchto vstupov potom môže útočník posielať škodlivý kód alebo neočakávané sekvencie udalostí do zraniteľného koncového bodu a tým tak docieliť získanie informácií alebo porušiť správne fungovanie zariadenia. [42]

1.7.2 Zraniteľnosť Insecure data storage

Pri tomto type zraniteľnosti je kľúčovým bodom priamy prístup k zariadeniu, to môže byť docielené získaním zariadenia fyzicky, buď krádežou alebo nájdením zariadenia. Ďalšou možnosťou je malvér alebo falošná aplikácia konajúca podľa cieľov útočníka. Ak útočník získa kontrolu nad zariadením, môže ho pripojiť k počítaču a pomocou špecializovaných nástrojov zobraziť jeho obsah, zahŕňajúci adresáre aplikácií alebo osobné informácie užívateľa nachádzajúce sa v zariadení. Táto zraniteľnosť sa môže vyskytnúť, ak vývojári predpokladajú, že nikto nebude mať prístup k súborovému systému zariadenia. Ale práve toto by mali predpokladať, pretože pri získaní zariadenia je veľká pravdepodobnosť, že útočník sa bude chcieť dostať k informáciám práve prehliadaním úložiska. Ďalším spôsobom je *Rootovanie* (Android) alebo *Jailbreak* (iOS), pri ktorom sa obchádzajú zabudované ochrany. Ak údaje nie sú správne chránené, dajú sa zobraziť pomocou špecializovaných nástrojov.

Táto zraniteľnosť môže viesť k strate dát jedného alebo v horšom prípade množstva užívateľov, tieto informácie môžu byť potom zneužité napríklad na krádež identity alebo podvod. Dobrým spôsobom ako sa tejto zraniteľnosti vyhnúť je ukladanie dát do databáz a cloudov, ktorým je zabránené ukladaniu dát lokálne v zariadení, tým pádom aj pri získaní zariadenia sa v súborovom systéme nenachádzajú žiadne dôležité informácie. Navyše ak je zariadenie odcudzené, stále sa užívateľ môže k dátam dostať, pretože sa nachádzajú na externom úložisku do ktorého má stále prístup. [43]

1.7.3 Nezabezpečená komunikácia (Insecure communication)

Pri bežnom spôsobe fungovania aplikácie sú dáta vymieňané medzi serverom a klientom. Pri tejto výmene musia teda dáta prejsť od mobilného zariadenia cez lokálnu sieť, alebo sieť operátora a internet. Táto cesta je väčšinou zabezpečená, avšak útočník môže využiť slabé miesta a zachytiť určité dáta už počas prenosu. Slabé miesta môžu byť v tomto prípade zdieľané pripojenie, sieťové zariadenia alebo malvér, ktorý sa dostane do zariadenia. V niektorých prípadoch sa môže stať, že mobilné aplikácie nechránia presuny dát po sieti, niektoré používajú SSL/TLS pri autentifikácii ale iné nemusia, tak isto nemusí mať aplikácia bezpečnostné prenosové prvky implementované správne, čo môže obmedziť alebo znemožniť ich správnu funkčnosť. Táto zraniteľnosť môže viesť napríklad ku krádeži užívateľského konta alebo v prípade uniknutia dát administrátora ku krádeži celého webu. Zneužívanie tejto zraniteľnosti je väčšinou realizované *Man In The Middle* útokom. [44]

1.7.4 Nezabezpečené autentifikácia

Autentifikácia je proces overovania alebo rozpoznania identity nejakého subjektu. Bližšie je rozobraná v kapitole o bezpečnostných vlastnostiach operačného systému Android. Pri tejto zraniteľnosti útočník využíva bezpečnostné diery v procese autentifikácie, ktoré sa zvyčajne vykonávajú pomocou automatizovaných útokov. Princíp zneužitia spočíva v pochopení fungovania autentifikačnej schémy. Potom pomocou zraniteľných častí útočník dokáže sfalšovať alebo obísť autentifikáciu odoslaním servisných požiadaviek na server, ktorý používa mobilná aplikácia pre autentifikáciu. Týmto je teda obídená priama interakcia servera s aplikáciou.

Ak je schéma autentifikácie slabá, umožnuje útočníkovi anonymne vykonávať funkcie v rámci mobilnej aplikácie. Táto slabá autentifikácia je v prostredí mobilných zariadení celkom bežná v dôsledku vstupného faktora. Tento vstupný faktor podporuje nedostatočné, často 4 ciferné PIN kódy. Narozdiel od webovej aplikácie, pri mobilnej aplikácii sa neočakáva, že používateľ bude stále online. Preto sú mobilné pripojenia menej spoľahlivé ako webové. Niektoré mobilné aplikácie môžu mať požiadavku na offline overenie, a toto môže mať veľký dopad na implementáciu mobilnej autentifikácie. Na nájdenie slabých autentifikačných schém sa používajú binárne útoky na aplikáciu, ktorá je offline. Týmto

tak môže útočník docieliť obídenie offline autentifikácie a spustiť tak aktivitu, ktorá by mala za normálnych podmienok vyžadovať autentifikáciu. [45]

1.7.5 Zraniteľnosť Insufficient cryptography

Kryptografia je súbor techník bezpečnej komunikácie, umožnujúci zobraziť obsah tejto komunikácie len odosielateľovi a príjemcovi. Zaoberá sa šifrovaním, čo je proces modifikácie informácie za cieľom zamaskovať ju tak, aby aj pri získaní tejto informácie bola bez potrebného kľúča nečitateľná a nezneužiteľná. Na šifrovanie sa používajú rôzne typy šifier, ktoré sa však so zvyšujúcim výpočtovým výkonom stávajú postupne nedostatočné a dajú sa prelomiť. Práve týmto faktom je spôsobená táto zraniteľnosť. Vývojári sa spoliehajú pri programovaní na šifry, ktoré sa už v dnešnej dobe dajú prelomiť alebo nedostatočne chránia kľúče slúžiace na „odomknutie“ dát. Nedostatočná kryptografia môže byť využitá buď fyzicky útočníkom, alebo malvérom, ktorý sa dostane do zariadenia. Cieľom útočníka je vrátiť zašifrovanú informáciu do jej pôvodnej podoby, tak aby ju mohol zneužiť vo svoj prospech.

Nedostatočná kryptografia môže byť zneužitá, ak aplikácia používa chybný spôsob šifrovania alebo dešifovania. Táto chyba tak môže viesť k dešifovaniu citlivých údajov útočníkom. Tak isto môže byť v aplikácii implementovaný slabý šifovací algoritmus, ktorý dokáže útočník priamo prelomiť.

Aj tie najlepšie šifrovacie algoritmy môžu byť prelomené, ak sa útočník dostane ku kľúču. Mnoho developerov používa korektný šifrovací algoritmus ale implementujú ho svojim vlastným spôsobom, ktorý nie je správny a tak sa útočník môže dostať ku jeho kľúču. Najčastejšími chybami sú: ukladanie kľúča v tom istom priečinku ako šifrovaný text, nechávanie kľúčov v kóde alebo iné sprístupnenie útočníkovi. Ďalším nesprávnym spôsobom je vytváranie vlastných šifrovacích algoritmov. Odporúča sa vždy používať overené moderné algoritmy, ktoré poskytujú vysokú mieru bezpečnosti. Mnoho kryptografických algoritmov, ktoré boli dlho považované za bezpečné sa v posledných rokoch stali slabými a nedostatočnými, keďže majú značné nedostatky alebo inak nepostačujú moderným bezpečnostným požiadavkám. Tieto obsahujú: RC2, MD4, MD5, SHA1. [46]

2 ANALÝZA SÚČASNÝCH SOFTWAROVÝCH NÁSTROJOV PRE STATICKÉ TESTOVANIE

V tejto kapitole budú podrobnejšie rozobrané nástroje, ktoré sa v súčasnosti používajú na rôzne úkony týkajúce sa statickej analýzy. Tieto nástroje budú obsahovať popis vlatností, využitia a prípadne aj proces inštalácie.

2.1 APKTool

APKTool je konzolový nástroj slúžiaci na dekompiláciu a kompiláciu APK balíčkov. Dokáže dekompilovať zdroje do takmer pôvodnej podoby a prebudovať ho po vykonaní modifikácií, taktiež je pomocou neho možné debugovanie jazyka Smali.
Na inštaláciu a správnu funkčnosť vyžaduje prítomnosť Java 1.8 a novšej na zariadení. Prvým krokom inštalácie na platforme *Windows* je stiahnutie wrapper scriptu, ktorý je následne potrebné uložiť ako *apktool.bat* do cieľovej zložky. Po tomto kroku je treba vykonať stiahnutie súboru JAR, ktorý by mal byť premenovaný na *apktool.jar* a presunutý do zložky, kde sa nachádza súbor BAT. Následne je potrebné sa v príkazovom riadku presunúť do priečinku, kde sa nachádzajú tieto súbory a potom môžu byť vykonávané operácie, ktoré tento nástroj poskytuje.
Pri inštaláciu na platformu *Linux* a *macOS* sa postupuje veľmi podobne, avšak pri týchto operačných systémoch je potrebný prístup ako root. [47]

Inštalácia wrapper scriptu *apktool.bat* nieje nevyhnutná, stačí súbor *apktool.jar*, ktorý je spustený v príkazovom riadku cez príkaz *java –jar apktool* a ďalej sa s ním dá pracovať ako v prípade súboru *.bat*.

Apktool disponuje štruktúrou príkazov v príkazovom riadku podobným tým bežným, s rozdielom pri udávaní parametrov, teda na dekompiláciu aplikácie *application.apk* je použitý príkaz *apktool d application.apk*, kde je kľúčovým slovom *apktool* zavolaný tento nástroj, parametrom *d* mu je určené, že bude zdroj dekompilovať a na záver je poskytnutý nástroju názov aplikácie, ktorú bude dekompilovať.

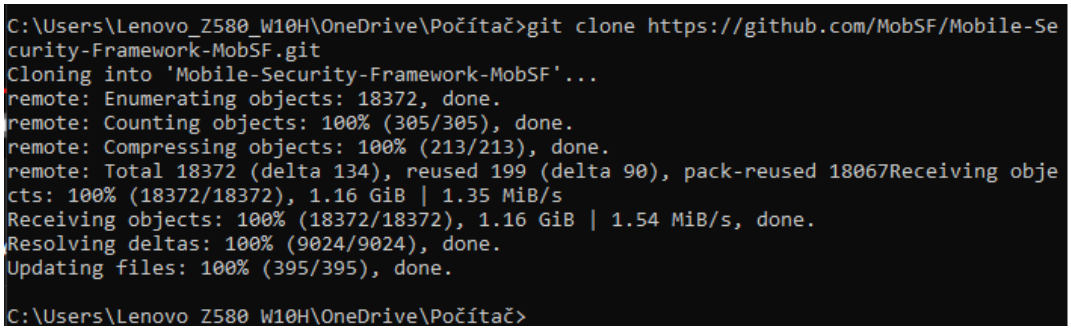
Pri kompilácii, respektíve builde, bude tento príkaz vyzerať rovnako s tým, že parametrom bude *b* značiace proces buildu (kompilácie), cieľového priečinku, výsledný príkaz teda je *apktool b application*.

Ďalej tento nástroj umožnuje inštaláciu frameworkov pomocou parametra *if*. Pri inštalovaní frameworku *framework-res.apk*, by príkaz znel *apktool if framework-res.apk*.

2.2 Mobile Security Framework (MobSF)

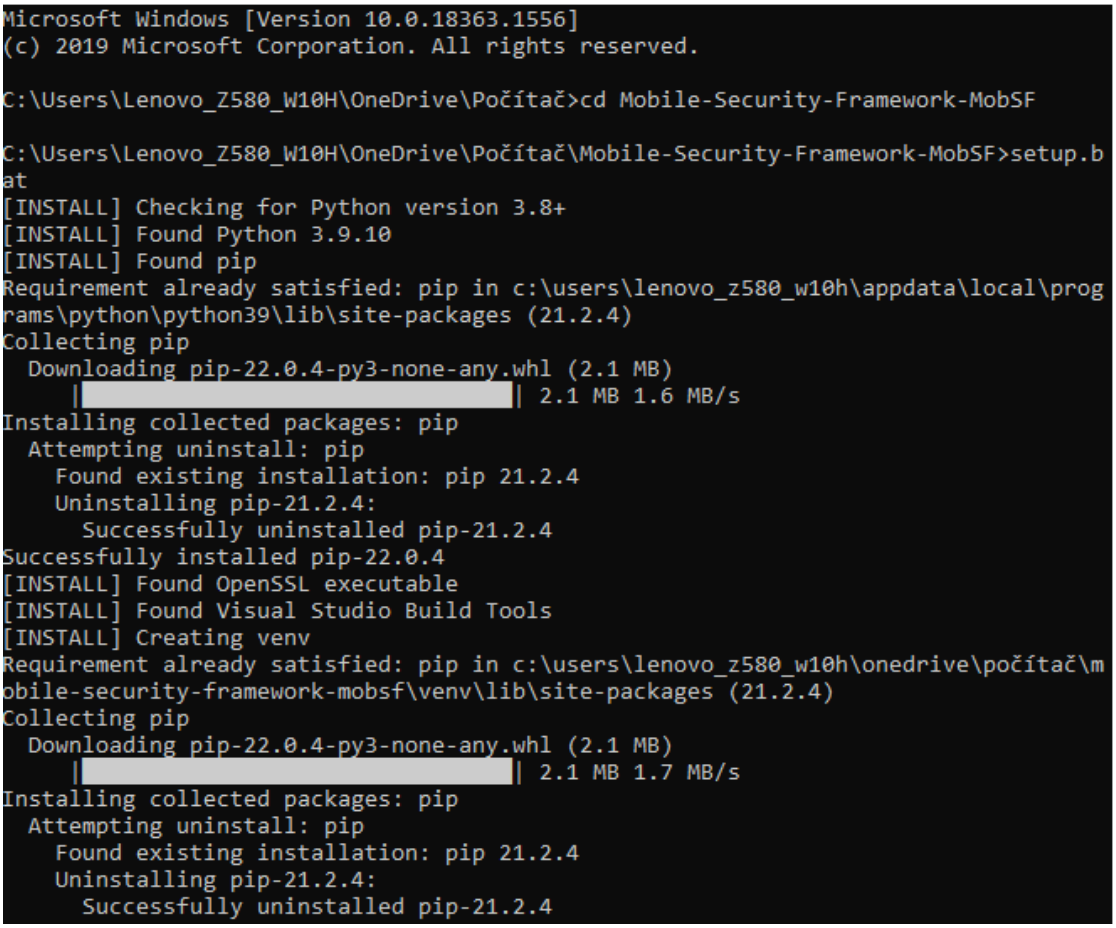
Tento nástroj poskytuje nielen statickú a dynamickú analýzu mobilných aplikácií pre Android a iOS, ale svojím rozsiahlym reportom môže poskytnúť veľa užitočných informácií pre celkové penetračné testovanie.
Tvorcovia MobSF ho definujú ako „automatizovanú komplexnú mobilnú aplikáciu pre Android/iOS/Windows penetračné testovanie, analýzu malvéru a rámec hodnotenia bezpečnosti, ktorý je schopný vykonávať statickú a dynamickú analýzu“ [48]. Toto je celkom presný popis, avšak netreba výstup z tohoto nástroja brať ako stopercentný a očakávať, že všetky problémy sa v tejto správe zobrazia. Optimálne je preto tento nástroj použiť ako začiatočný bod, ktorý testera potom nasmeruje správnym smerom, aby mohli byť podrobnejšie preskúmané dané oblasti aplikácie.

Na správne fungovanie tohoto nástroja na Windows OS je potrebné predom nainštalovať Java JDK, Python 3.8 a vyšší, Visual Studio Build Tools, Git for Windows, Open SSL a WKHTMLtoPDF.
Keď zariadenie obsahuje všetky tieto potrebné komponenty, môže byť spustené klonovanie adresára git.



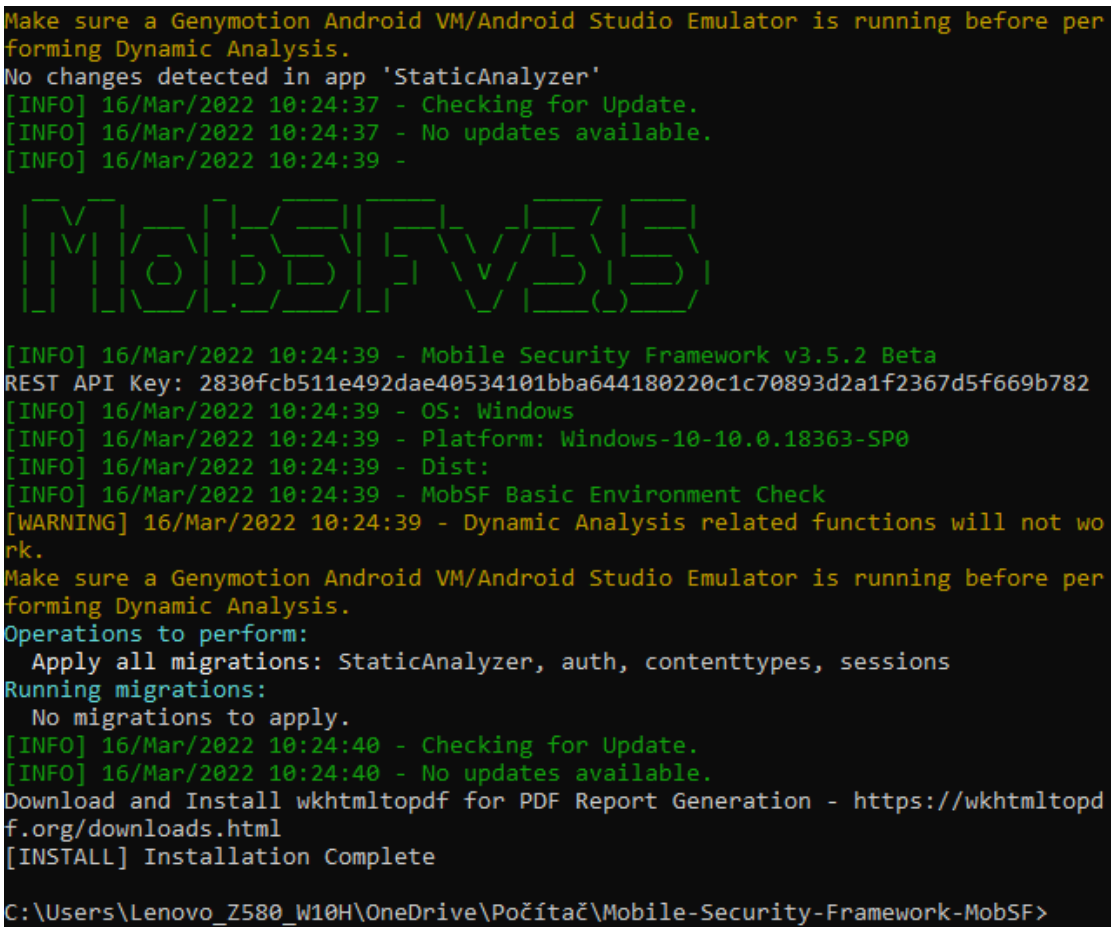
Obrázok 12. klonovanie git adresára MobSF [zdroj vlastný]

Po naklonovaní adresára je potrebné sa doň dostať a to v príkazovom riadku zadaním príkazu *cd* a názvu adresára. Následne stačí už len spustiť inštaláciu pomocou súboru *setup.bat*.



Obrázok 13. Spustenie inštalácie MobSF [zdroj vlastný]

Po spustení tohoto súboru sa začnú inštalovať jednotlivé súčasti tohoto nástroja, tento proces zaberie len pár minút a všetko sa už spraví automaticky, takže do chodu nemusí byť zasahované.



Obrázok 14. Stránka s informáciami po nainštalovaní MobSF [zdroj vlastný]

Keď je MobSF nainštalovaný, je možné spustiť ho pomocou súboru *run.bat* v príkazovom riadku. Po spustení program vypíše, kde beží (defaultne je to localhost s portom 8000). Následne môže byť v prehliadači otvorená táto adresa. Po zadaní adresy je zobrazené grafické rozhranie tohoto nástroja. Ak je MobSF spustený v režime statickej analýzy, je možné nahrať akýkoľvek súbor mobilnej aplikácie (najčastejšie súbory s príponami .apk alebo .ipa), ktorý je následne zanalyzovaný. Po úspešnej analýze je zobrazená obsiahla správa zobrazujúca všetky úskalia, ktoré by mali byť skontrolované a môžu byť potenciálne nebezpečné. Ak sa testuje alebo analyzuje viacero súborov naraz, vygenerované prehľady je možné kedykoľvek znova navštíviť prostredníctvom karty

„Nedávne kontroly“, ktorá zobrazí všetky súbory nahrané do MobSF. Tieto správy možno tiež exportovať vo formáte PDF na kontrolu mimo aplikácie alebo zdieľať.

Tieto správy budú obsahovať informácie o všetkom, od toho, či je súbor bezpečne podpísaný a ako to bolo urobené, až po funkcie, ktoré bude aplikácia využívať po inštalácii do zariadenia. Výsledná správa od MobSF sa môže zdať neprehľadná kvôli veľkosti a obrovskému množstvu informácií, ktoré generuje. Preto je kľúčové sústrediť sa na dôležité aspekty, ktoré rozšíria akékoľvek vykonávané testovanie a izolujú potenciálne problémy a riziká v rámci aplikácie.

2.3 JADX

JADX je nástroj, ktorý slúži primárne na dekompiláciu APK súborov. Taktiež dokáže dekompilovať Dalvik bajtkód na kód Java, zobraziť a dekódovať obsah súboru *AndroidManifest.xml* alebo zdroje z balíku *resources.arsc*. Tento nástroj sa dá získať v dvoch podobách, ako klasický konzolový nástroj podobný nástroju Apktool alebo vo forme GUI (Graphic User Interface), pod názvom *Jadx-gui.* Táto verzia nástroja je veľmi prehľadná s intuitívnym a pohodlným ovládaním vrátane drag&drop funkcie na pridanie súboru, ktorý sa bude dekompilovať. Následne poskytuje rozhranie pre zobrazenie kódu, v ktorom je možné vyhľadávať kľúčové slová alebo zobraziť celú štruktúru súboru. JADX nástroj umožňuje taktiež aj debugovanie jazyka Smali. Tento debugger obsahuje funkcie ako krokovanie, nastavovanie breakpointov, modifikovanie hodnôt, atď. [49]

Pri tomto nástroji nie je inštalácia nutná, stačí stiahnuť balíček, ktorý je potrebné rozbaliť a potom už len pomocou spustenia *jadx.bat* alebo *jadx-gui.bat* cez príkazový riadok vybrať či bude použitá konzolová alebo grafická verzia. Ak je však nástroj nainštalovaný, poskytuje možnosť spustenia *.exe* súboru, ktorý je spustiteľný aj bez potreby príkazového riadku.

2.4 Dex2jar

Dex2Jar je nástroj na prácu s *.dex* a Java *.class* súbormi. Súbory dex sú súbory kompilovaného kódu pre Android. Hlavnou funkciou nástroja Dex2jar je konvertovanie súborov typu *dex* v balíku APK, na súbory typu *jar* a naopak. Následne je možné zobraziť zdrojový kód aplikácie pomocou ľubovolného dekompilátora Java. Výstupom sú však súbory typu *class* a nie priamo kód ktorý napísal vývojár. [50]

Inštalácia v prípade tohoto nástroja nieje potrebná, stačí nástroj stiahnuť a potom cez príkazový riadok z adresára, kde sa tento nástroj nachádza môžu byť vykonávané jednotlivé operácie.

2.5 JD-GUI

JD-GUI je grafický nástroj, ktorý umožňuje zobraziť zdrojový kód súborov *class*, ktoré sa nachádzajú v súbore *jar*. Okrem súborov *jar* tento nástroj dokáže zobraziť aj obsah súborov ako sú *aar, java, jmod,* a podobne. Následne je možné v tomto nástroji prechádzať zrekonštruovaný kód. Tento kód je v čitateľnom formáte a je jednoduché sa v ňom orientovať. [51]

Získanie tohoto nástroja je veľmi jednoduché, stačí stiahnuť príslušnú verziu pre daný systém, nakoľko je tento nástroj dostupný pre Linux, Mac aj Windows. Následne už len treba rozbaliť stiahnutý ZIP archív a umiesniť ho na ľubovolné miesto. Potom spustením *.exe* súboru s názvom *jd-gui.exe* spustiť tento nástroj a buď pomocou menu vo vrchnej časti alebo metódou drag&drop vložiť požadovaný súbor, ktorý bude prezeraný.

3 ANALÝZA SÚČASNÝCH SOFTWAROVÝCH NÁSTROJOV PRE DYNAMICKÉ TESTOVANIE

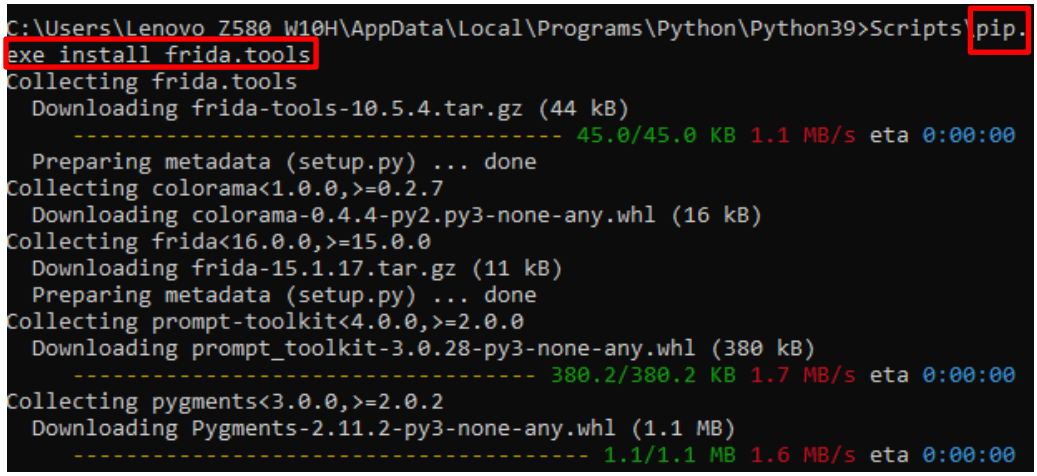
Podobne ako v predošlej kapitole, aj v tejto sú popísané nástroje, avšak pri tejto kapitole sa jedná o nástroje dynamickej analýzy.

3.1 Frida

Frida je nástrojová sada pre dynamickú inštrumentáciu kódu. Dovoľuje užívateľovi „injectovať“ kusy JavaScriptu alebo vlastnej knižnice do natívnych aplikácií platforiem Windows, macOS, Linux, iOS a Android. Frida taktiež poskytuje niektoré jednoduché nástroje vstavané vo Frida API. Tieto nástroje môžu byť použité podľa potrieb používateľa.

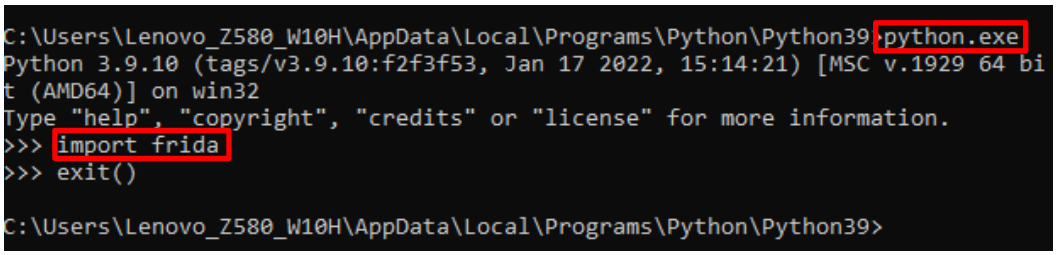
Kód nástroja Frida je napísaný v jazyku C a injectuje QuickJS do cieľových procesov, kde je vložený JavaScript vykonaný s plným prístupom do pamäti. Zachytáva alebo dokonca volá natívne funkcie vo vnútri procesu. Obsahuje obojsmerný komunikačný kanál, ktorý je používaný na komunikáciu medzi užívateľovou aplikáciou a JavaScriptom, bežiacim vovnútri procesu.
Použitie Pythonu a JavaScriptu umožnuje rýchly vývoj pomocou bezrizikového API. Frida môže pomôcť jednoducho zachytiť chyby v JavaScripte a vytvoriť výnimku namiesto spadnutia programu.
Fridu je možné použiť aj priamo z jazyka C bez potreby písania kódu v Pythone, okrem jadra v C obsahuje aj väzby viacerých jazykov ako napr. Node.Js, Python, Swift,.NET, atď. Taktiež je veľmi jednoduché vytvoriť dodatočné väzby pre iné jazyky a prostredia.[52]

Inštalácia tohoto nástroja vyžaduje Python vo verzii 3 a vyššej. Pomocou príkazového riadku a skriptu *pip.exe* je možné nainštalovať tento nástroj bez potreby dodatočného klonovania adresára.

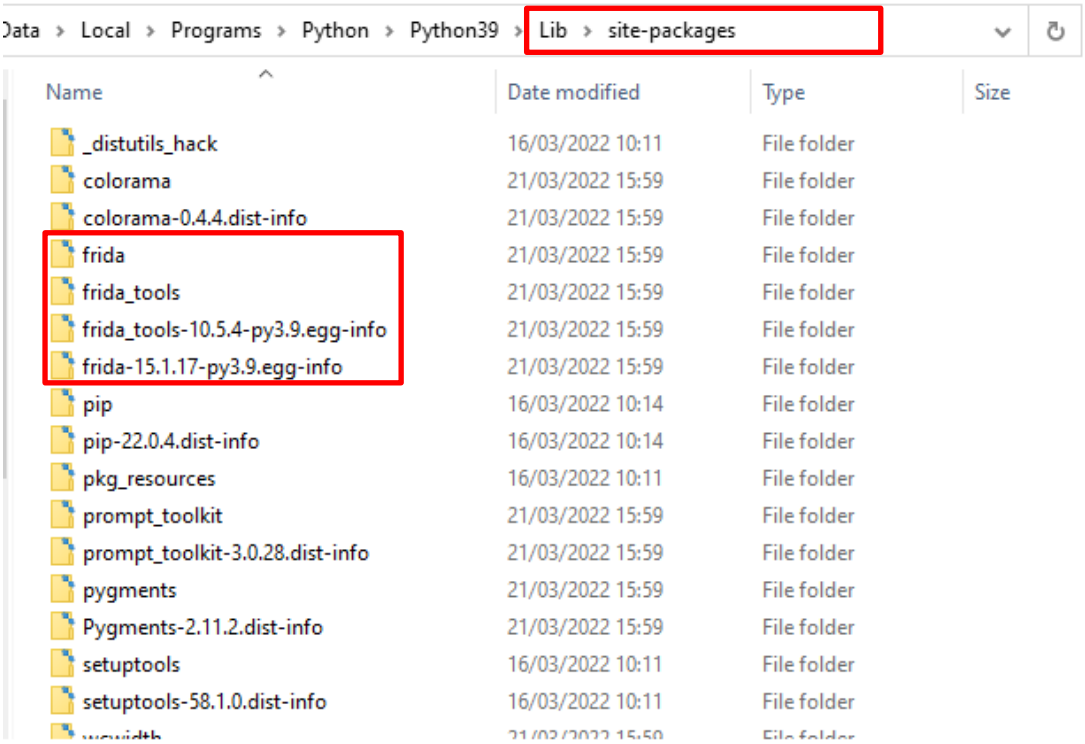


Obrázok 15. Inštalácia nástroja Frida pomocou pip.exe [zdroj vlastný]

Po zadaní tohoto príkazu sa automaticky stiahne nástroj Frida a všetky jeho komponenty. Následne ešte pomocou Python Shell treba skontrolovať, či je nástroj nainštalovaný. Tento shell sa spúšťa pomocou súboru *python.exe*, v shelle je toto overenie vykonané príkazom *import frida* a ak sa nezobrazí chybová hláška, nástroj by mal byť správne nainštalovaný.



Obrázok 16. Overenie inštalácie nástroja Frida pomocou Python shell [zdroj vlastný]



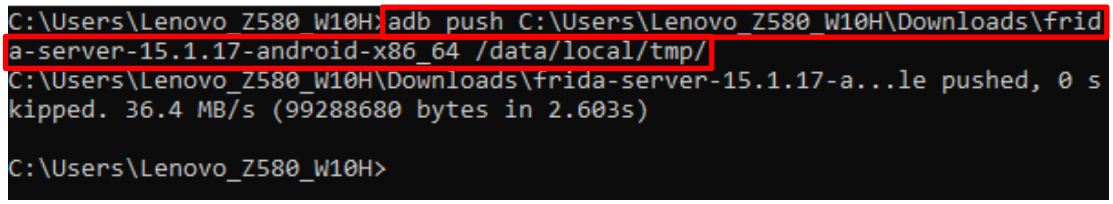
Obrázok 17. Adresár obsahujúci komponenty nástroja Frida [zdroj vlastný]

Okrem toho je vhodné skontrolovať súborovú štruktúru. V adresári *Python*, zložke *Lib* a v nej *site-packages,* by mali byť viditeľné všetky súčasti nástroja Frida. Následne pre potreby testovania je nainštalované ešte jedno rozšírenie. Je to nástroj poháňaný Fridou na skúmanie aplikácie za behu, umožňuje zobrazovať a sledovať spustené funkcie v rámci aktivít alebo modifikovať boolean návratové hodnoty. Toto rozšírenie sa volá *Objection* a inštaluje sa príkazom *pip3 install objection* v príkazovom riadku.

```
C:\Users\Lenovo_Z580_W10H>pip3 install objection
Collecting objection
  Downloading objection-1.11.0.tar.gz (327 kB)
     ---------------------------------------- 327.2/327.2 KB 195.1 kB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: frida>=14.0.0 in c:\users\lenovo_z580_w10h\appdata\local\prog
rams\python\python39\lib\site-packages (from objection) (15.1.17)
Requirement already satisfied: frida-tools>=6.0.0 in c:\users\lenovo_z580_w10h\appdata\local
\programs\python\python39\lib\site-packages (from objection) (10.5.4)
Requirement already satisfied: prompt_toolkit<4.0.0,>=3.0.3 in c:\users\lenovo_z580_w10h\app
data\local\programs\python\python39\lib\site-packages (from objection) (3.0.28)
Collecting click
  Downloading click-8.0.4-py3-none-any.whl (97 kB)
     ---------------------------------------- 97.5/97.5 KB 349.0 kB/s eta 0:00:00
```

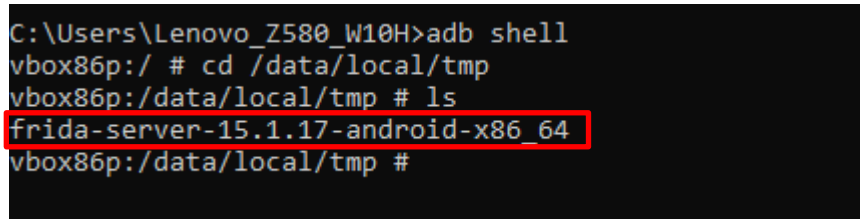
Obrázok 18. Inštalácia rozšírenia Objection [zdroj vlastný]

Ďalším krokom bude nainštalovanie Frida servera na zariadenie Android, prípadne teda emulátor so systémom Android. Príkazom *frida –version* v príkazovom riadku je zistená verzia nástroja Frida, následne je potrebné z Githubu stiahnuť prislúchajúcu verziu Frida Servera, ktorý sa stiahne v súbore typu *XZ,* ktorý treba rozbaliť. Pomocou nástoja ADB je potom tento súbor presunutý do zariadenia emulátora, príkaz bude vyzerať *adb push <cesta k súboru ktorý bol stiahnutý> <priečinok v emulátore kam bude umiestnený>*



Obrázok 19. Umiestnenie Frida servera na emulátor [zdroj vlastný]

Keď je súbor prenesený, môže byť jeho prítomnosť skontrolovaná cez ADB shell, vypísaním obsahu cieľového priečinka.



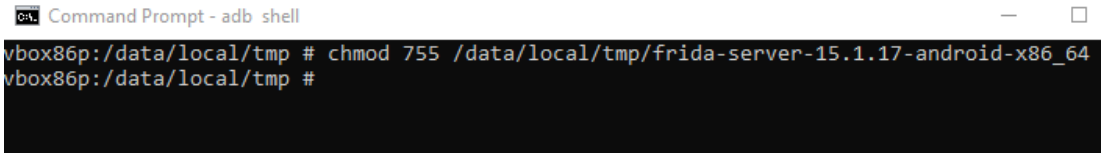
Obrázok 20. Zobrazenie súboru serveru na emulátore [zdroj vlastný]

Keď je súbor úspešne priemiestnený, ešte je potrebné nastaviť jednotlivé povolenia príkazom *chmod* a kde parameter 755 znamená:

7 – nastavenie povolenia čítať, písať a spúšťať pre vlastníka používateľa

5 – nastavenie povolenia čítať a spúšťať pre vlastníka skupiny

5 – nastavenie povolenia čítať a spúšťať pre ostatných



Obrázok 21. Nastavenie povolení pre súbor serveru [zdroj vlastný]

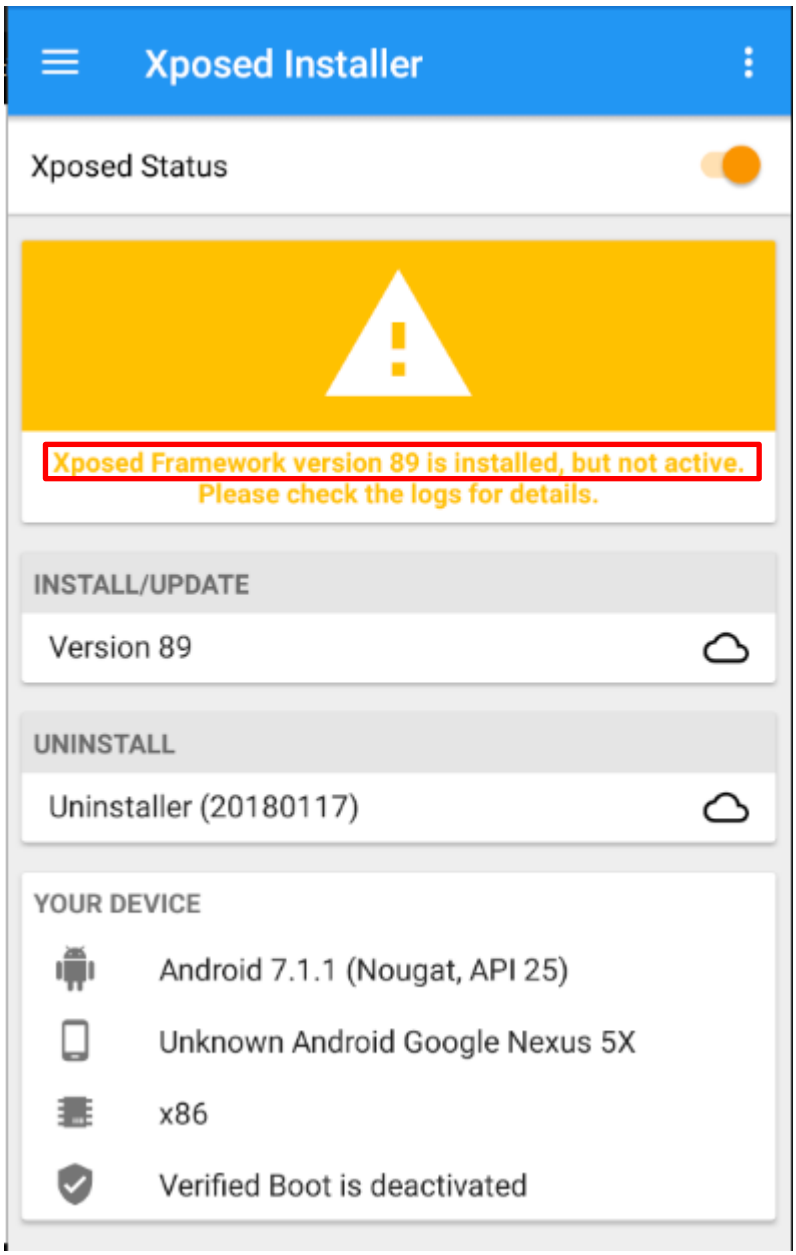
Už stačí len tento server spustiť pomocou príkazu *./<názov servera> &*. Týmto je inštalácia dokončená a nástroj je pripravený na použitie.

3.2 Xposed Framework

Xposed je framework pre zariadenia Android s prístupom root. To samo o sebe sprístupní veľa možností, avšak tento framework umožňuje dosťahovať a pridávať rôzne vylepšenia a ďalšie nástroje zvané „moduly“, ktoré môžu byť využité. [53]

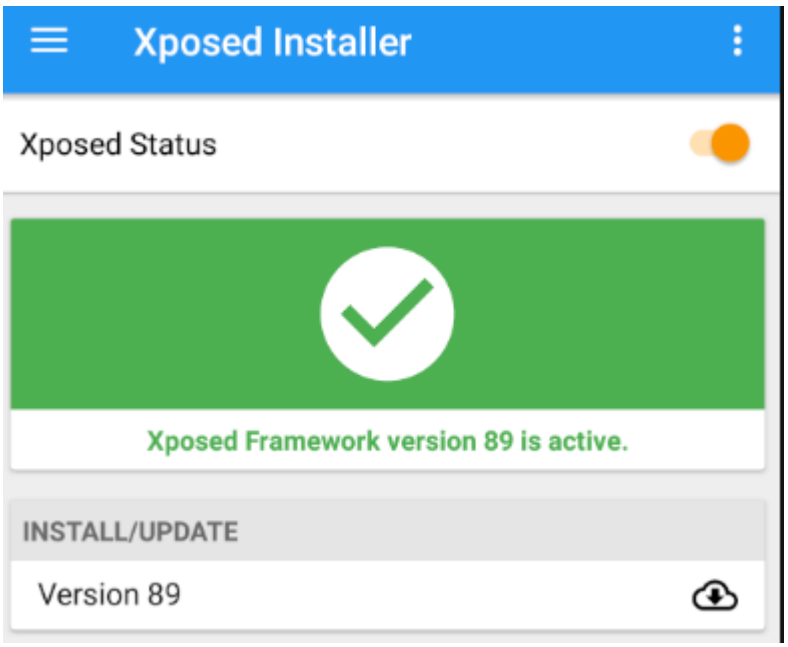
Inštalácia tohoto frameworku je jednoduchá, stačí stiahnuť súbor APK a tento potom nainštalovať do zariadenia. Ako už je vyššie spomenuté, inštalácia tohoto frameworku vyžaduje root. Štandardne je v zariadení zablokované inštalovanie aplikácie z neznámych zdrojov, teda prakticky je povolené len sťahovanie z oficiálneho obchodu, ktorým je Google Play. Toto nastavenie však môže byť veľmi jednoducho vypnuté. Stačí prejsť do nastavení a v sekcii zabezpečenie povoliť tzv. neznáme zdroje.

Po spustení Xposed sa zobrazí obrazovka, ktorá hovorí že tento framework je nainštalovaný, avšak nie je aktivovaný.



Obrázok 22. Neaktivovaný Xposed Framework [zdroj vlastný]

Na aktivovanie je potrebné kliknúť na „Version 89“, kde je zvolená možnosť „install“ a zariadenie už dokončí všetko potrebné. Na záver sa zobrazí hláška informujúca o tom, že zmeny sa prejavia až po reštarte zariadenia, preto môže byť zariadenie reštartované cez príkazový riadok pomocou príkazu *adb reboot*.

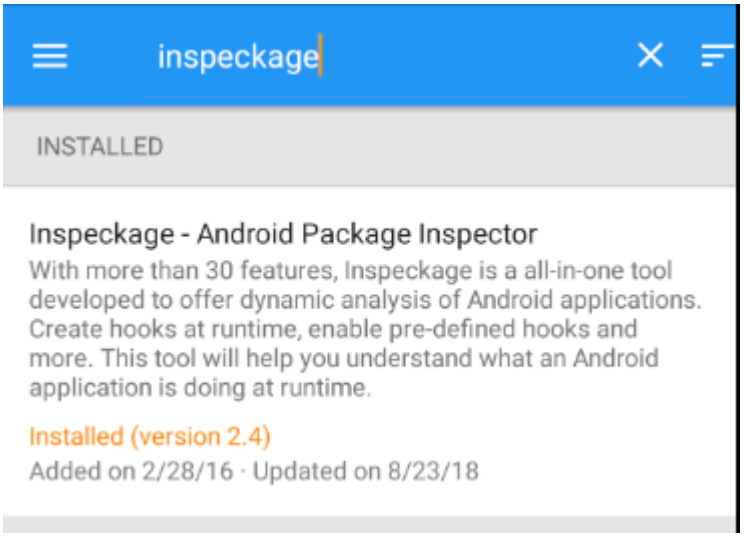


Obrázok 23. Xposed Framework po aktivovaní [zdroj vlastný]

Po reštarte zariadenia a spustení aplikácie Xposed je vidieť, že Framework je aktívny.

Tento nástroj sám o sebe však až tak užitočný nieje, čo ho robí užitočným a všestranným sú práve spomínané moduly, ktoré môžu byť do neho nainštalovené. Preto je v ďalšom odseku priblížená inštalácia jedného z týchto modulov.

Inspeckage je modul, ktorý umožňuje sledovať celý beh aplikácie a zachytávať skoro všetky úkony, ktoré aplikácia vykoná. Na inštaláciu tohoto modulu je potreba v aplikácii Xposed v záložke „downloads“, ktorá slúži na stiahnutie modulov zadať kľúčové slovo „inspeckage“. Následne aplikácia vyhľadá tento modul a ten môže byť nainštalovaný kliknutím naň.



Obrázok 24. Inštalácia modulu Inspeckage [zdroj vlastný]

Následne musí byť modul aktivovaný cez záložku „Modules“, kliknutím na konkrétny modul. Po aktivácii musí byť zariadenie reštartované, aby sa vykonali zmeny. Nakoniec je potrebné len vybrať aplikáciu ktorá bude odchytávaná.

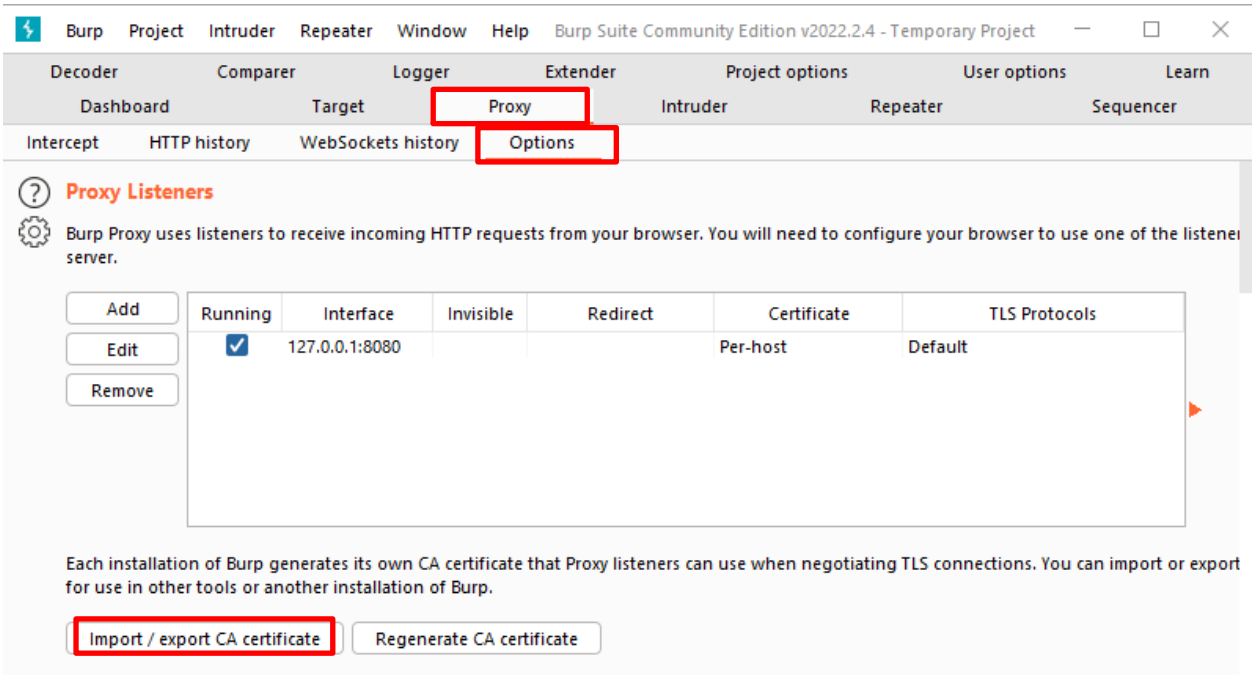
3.3 Burp Suite

Burp Suite je sada nástrojov, ktoré sa používajú primárne na testovanie webových aplikácií. Vzhľadom nato, že veľká časť mobilných aplikácií využíva internetové pripojenie a funguje na báze modelu klient-server, je tento nástroj vhodný práve na testovanie takýchto aplikácií. Jadnou z hlavných predností tohoto nástroja je schopnosť zachytávať HTTP požiadavky pomocou nastavenej proxy.[54]

V bežnej praxi tieto HTTP požiadavky idú z aplikácie priamo na webový server a potom sa odpoveď vráti do aplikácie. Pomocou Burp Suite Proxy však môže byť vykonané niečo ako MITM(Man-In-The-Middle) útok, pretože postavením tohoto nástroja medzi aplikáciu a webový server môže byť odchytávaný tento prenos. Následne môže byť rozhodnuté či bude daná požiadavka poslaná ďalej (forward) alebo zahodená (drop). Taktiež tento nástroj

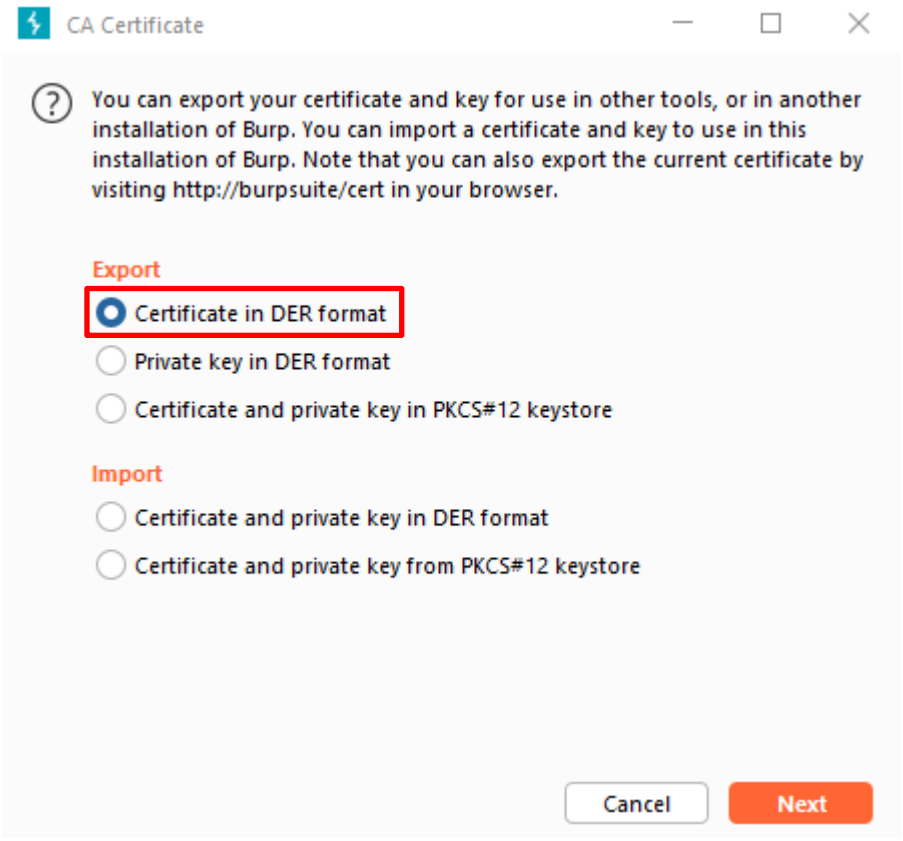
umožňuje vidieť a analyzovať obsah danej požiadavky a skontrolovať tak zraniteľnosť alebo cistlivosť posielaných dát.

Inštalácia tohoto nástroja pozostáva zo stiahnutia a nainštalovania community edition, ktorá je dostupná zdarma, po zadaní emailu sa stiahne spustiteľný súbor pomocou ktorého je vykonaná inštalácia. Po spustení nainštalovaného nástroja je potrebné vybrať *Temporary project*, teda dočasný projekt, ostatné možnosti nie sú v tejto verzii dostupné. Po kliknutí na tlačítko *next* je ponúknutá možnosť vybrať konfiguračný súbor, tu je zvolená možnosť *use Burp defaults*, takže sa použije prednastavená konfigurácia. Následne môže byť nástroj Burp Suite spustený. V tejto ukážke bude použitá proxy, takže v ďalšom kroku je nutné prejsť do podzložky *Proxy* v menu a zvoliť záložku *options*.



Obrázok 25. Burp Suite Proxy [zdroj vlastný]

Následne je potrebné vytvoriť certifikát, ktorý bude nainštalovaný na mobilné zariadenie, aby bolo pripojenie dôveryhodné a prenos dát mohol bez problémov fungovať. To je vykonané pomocou kliknutia na *Import/export CA certificate*.

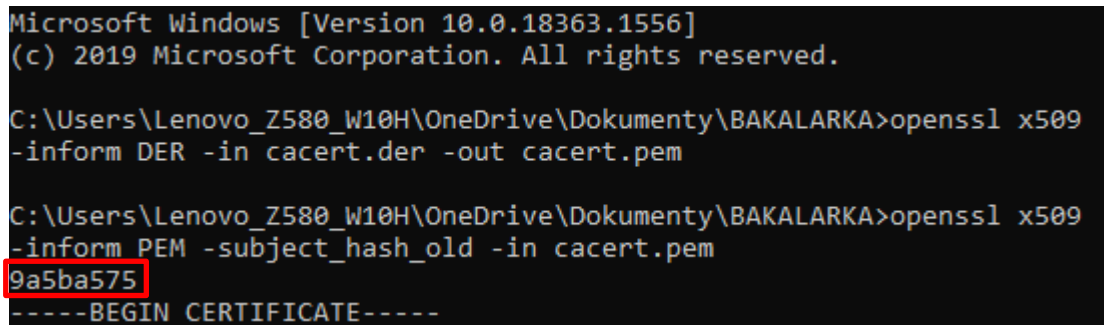


Obrázok 26. Výber certifikátu vo formáte DER [zdroj vlastný]

Následne je zvolená možnosť CA certifikát vo formáte *DER*. Tento formát značí digitálny certifikát v binárnom formáte. Následne je treba zvoliť miesto pre uloženie a vytvoriť certifikát.

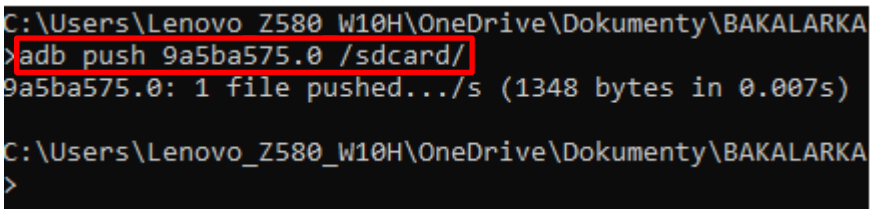
Po viacerých neúspešných pokusoch o inštaláciu v tomto formáte bolo zistené, že Android potrebuje tento certifikát vo formáte *PEM* a názov musí byť v *hash* formáte s príponou *.0*. Podľa stránky *developer.android.com* sa pripínanie certifikátu vykonáva hashom verejného kľúča certifikátu X.509. Reťazec certifikátov je potom platný len vtedy, ak tento reťazec obsahuje aspoň jeden z pripnutých verejných kľúčov[55]. Na túto operáciu je použitý nástroj OpenSSL. V príkazovom riadku je pomocou príkazu *openssl x509 –inform DER –in <názov_certifikátu.der> -out <názov_nového_certifikátu.pem>* vykonaná konverzia na formát PEM. Následne treba ďalším príkazom vytvoriť hash daného *PEM* certifikátu. A to

príkazom *openssl x509 –inform PEM –subject_hash_old –in <název_nového_certifikátu.pem>*.



Obrázok 27. Získanie Hashu certifikátu pomocou OpenSSL [zdroj vlastný]

Potom treba premenovať certifikát s príponou *DER* na *<hash>.0*. Teda v tomto prípade *9a5ba575.0*. Ďalším krokom bude prekopírovanie tohoto certifikátu do zariadenia. Na túto operáciu môže byť použitý *ADB* a pomocou príkazu *adb push <cesta_k_certifikátu> <cieľové_miesto_na_zariadení>* tento certifikát prenesený na zariadenie.

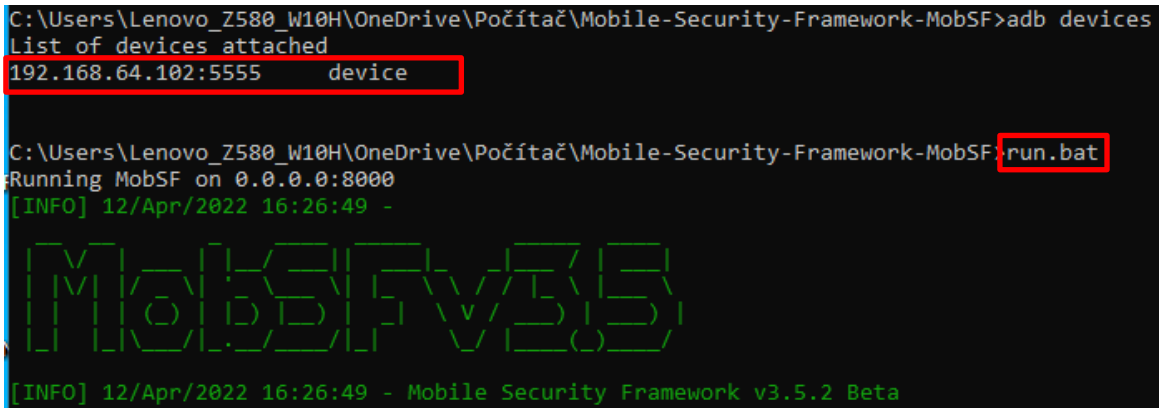


Obrázok 28. Presunutie certifikátu na emulátor [zdroj vlastný]

Nakoniec už zostáva len presunúť tento certifikát do zložky */system/etc/sceurity/cacerts/*, kde by mali byť uložené aj iné certifikáty. Po reštarte zariadenia by malo byť už všetko funkčné a môže byť pristúpené k nastavovaniu proxy. To bude ukázané už pri konkrétnom prípade v časti praktickej demonštrácie tohoto nástroja.

3.4 MobSF Dynamic Analyzer

Tento nástroj bol už popísaný v sekcii nástrojov statickej analýzy, keďže je tento nástroj veľmi komplexný, umožňuje aj dynamickú analýzu. Preto bude vykonaná aj praktická demonštrácia tejto časti. Na spustenie tohoto nástroja v režime *Dynamic Analyzer* je potrebné mať dopredu spustené testovacie prostredie, teda emulátor Genymotion. Keď je zariadenie spustené, môže byť pomocou príkazu *run.bat* tento nástroj spustený.



Obrázok 29. Spustenie MobSF pomocou súboru run.bat [zdroj vlastný]

V závere už stačí vybrať zo zoznamu aplikácií na zariadení aplikáciu, ktorá bude testovaná.

II. PRAKTICKÁ ČASŤ

4 PRAKTICKÁ DEMONŠTRÁCIA SÚČASNÝCH SOFTVÉROVÝCH NÁSTROJOV PRE STATICKÚ ANALÝZU

Pri práci s aplikáciami bude potrebné aplikácie aj reálne spúšťať, existujú teda dve možnosti. Buď pracovať s reálnym zariadením s operačným systémom Android alebo použiť tzv. emulátor. Emulátor je zariadenie, ktoré vytvára virtuálne zariadenie. Toto zabezpečuje odstránenie rizika poruchy alebo nebezpečenstva pre zariadenie Android, keďže existuje len vo virtuálnom prostredí. Výhodou je tiež možnosť výberu konkrétneho zariadenia bez potreby nákupu fyzického. Môže byť teda k dispozícii ľubovolné zariadenie a prakticky aj ľubovolná verzia systému Android, na ktorej budú ukážky vykonávané.

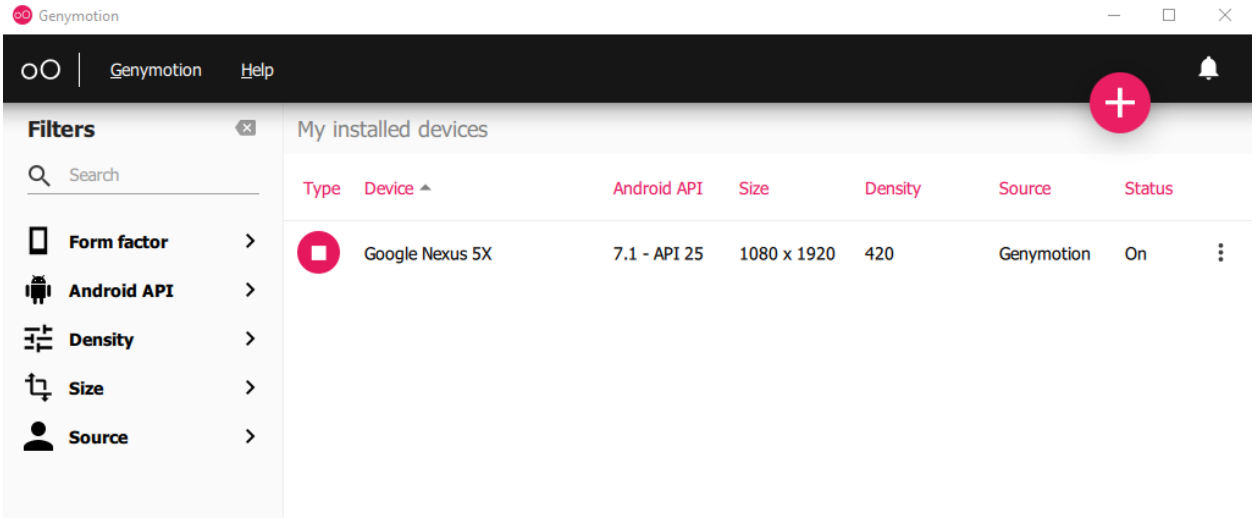
4.1 Príprava prostredia

4.1.1 Genymotion

Genymotion je emulátor, ktorý poskytuje kompletný set nástrojov a vlastností systému pri interakcii s virtuálnym prostredím, vo verzii community pre vlastné použitie je dostupný zdarma. Umožňuje testovať mobilné aplikácie na veľkom množstve virtuálnych zariadení. Emulátor Genymotion je dostupný pre systémy Windows, Linux, aj macOS.

Inštalácia zahŕňa stiahnutie inštalačného súboru, kde je ponúknuté na výber stiahnutie súboru, ktorý obsahuje aj inštaláciu programu VirtualBox. Toto je program, ktorý sa stará o virtualizovanie zariadenia a jeho spúšťanie. Po nainštalovaní je obdržaný kompletne funkčný model obsahujúci programy Genymotion a VirtualBox.

Po spustení Genymotion je potrebné zvoliť zariadenie, ktoré bude virtualizované. Pre túto prácu je zvolené zariadenie *Google Nexus 5X* so systémom *Android 7.1*. Po výbere zariadenia sa stiahnu súbory potrebné na virtualizáciu tohoto zariadenia, ktoré následne môžu byť spustené.

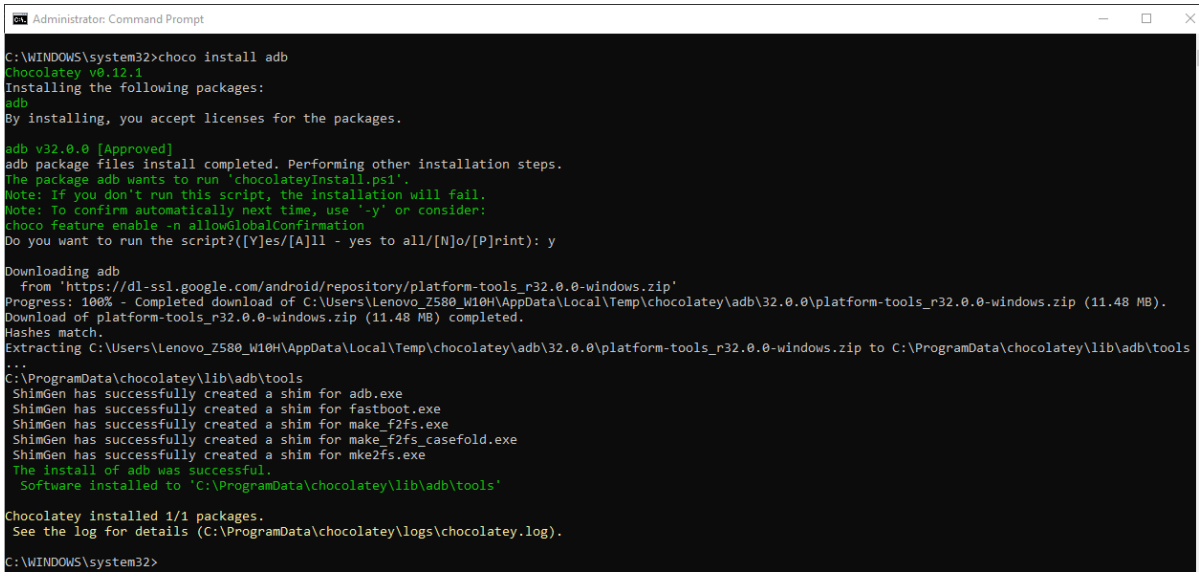


Obrázok 30. Prostredie Genymotion [zdroj vlastný]

4.1.2 Android Debug Bridge (ADB)

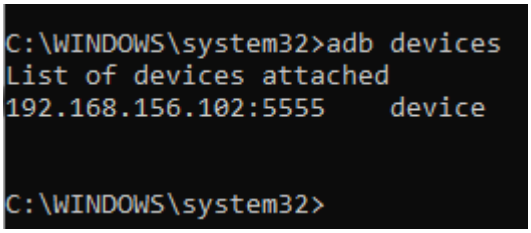
ADB je všestranný nástroj pre príkazový riadok, ktorý umožňuje komunikáciu s virtualizovaným zariadením (alebo fyzickým zariadením pripojeným cez USB). Nástroj umožňuje vykonávať operácie ako inštaláciu, zobrazenie informácií o aplikácii na zariadení, ukladanie dát do pamäte zariadenia a podobne.

Na inštaláciu tohoto nástroja je použitý inštalačný program zvaný „chocolatey“, ktorého úlohou je zjednodušiť inštaláciu nástrojov a programov v systéme Windows. Na nainštalovanie nástroja *ADB* je použitý príkaz *choco install adb* v príkazovom riadku. Inštalačný program sa potom postará o celú inštaláciu vrátane stiahnutia súborov.



Obrázok 31. Inštalácia Android Debugg Bridge [zdroj vlastný]

Po nainštalovaní ADB je ešte potrebné pripojiť virtuálne zariadenie k tomuto nástroju. V prípade Genymotion sa zariadenie pripojí automaticky, ak by sa tak ale nestalo, stačí zadať príkaz *adb connect <ip:port>*. Pripojenie potom môže byť skontrolované príkazom *adb devices,* ktorý vypíše zoznam pripojených zariadení.

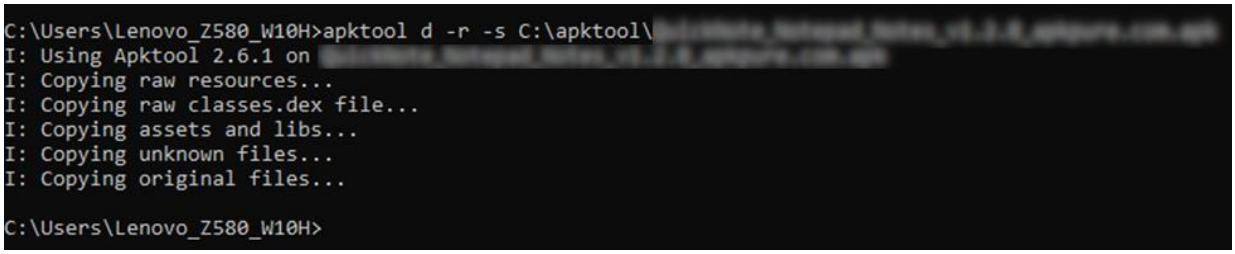


Obrázok 32. Vypísanie pripojených zariadení cez ADB [zdroj vlastný]

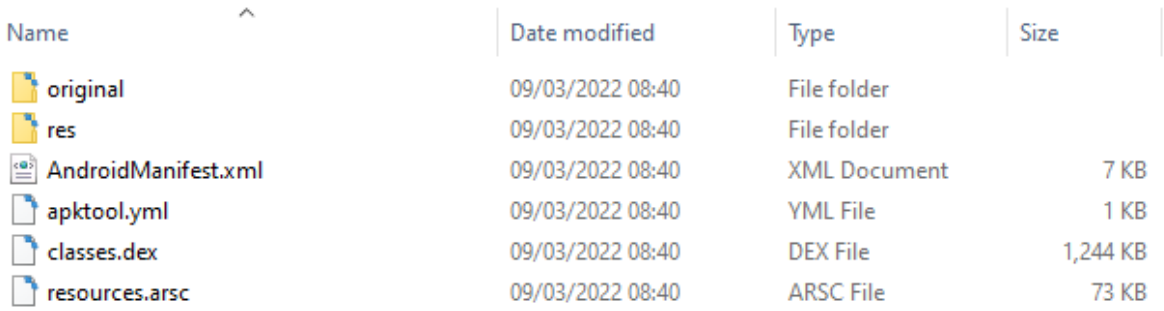
4.2 Práca s nástrojmi Statickej analýzy

4.2.1 APKTool

Hlavným použitím nástroja APKTool je dekompilácia a kompilácia súborov. V tejto ukážke je pomocou tohoto nástroja dekompilovaný súbor APK. Príkazom *apktool d <cesta k aplikácii>*, je súbor dekompilovaný, následne sa dekompilovaný súbor uloží do priečinka, z ktorého je spúšťaný tento nástroj. Po jeho rozbalení je viditeľný obsah súboru.

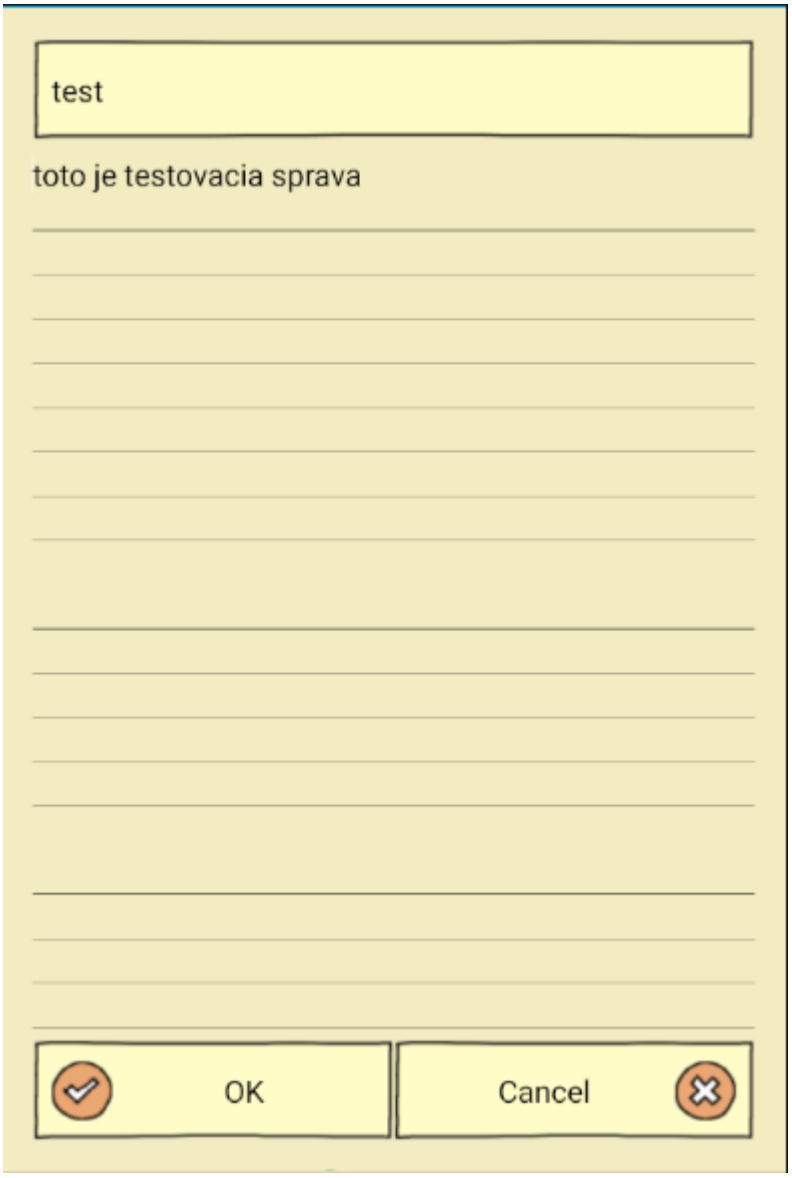


Obrázok 33. Dekompilácia aplikácie pomocou ApkTool [zdroj vlastný]



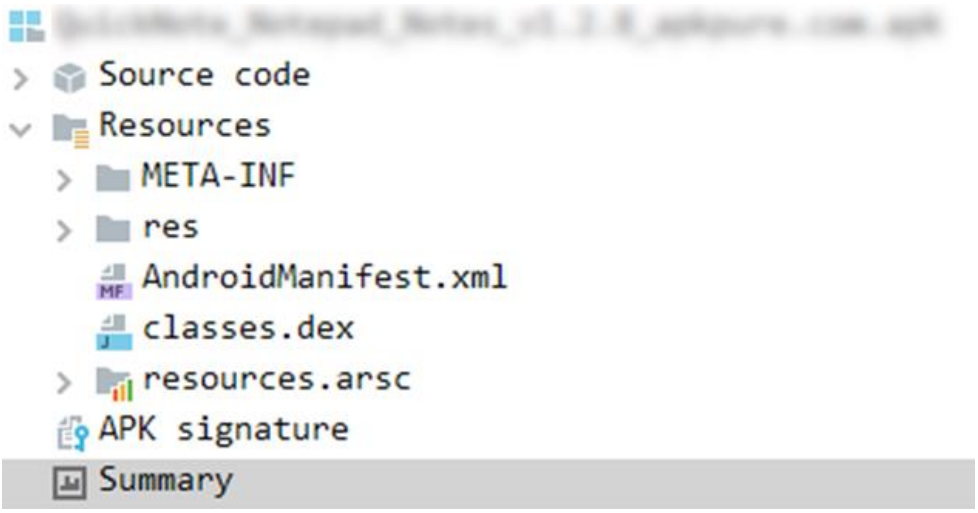
Obrázok 34. Obsah zložky aplikácie po dekompilácii nástrojom ApkTool [zdroj vlastný]

Do virtuálneho zariadenia je nainštalovaná aplikácia, ktorá slúži ako poznámkový blok, do ktorého je možné zapisovať jednotlivé poznámky.

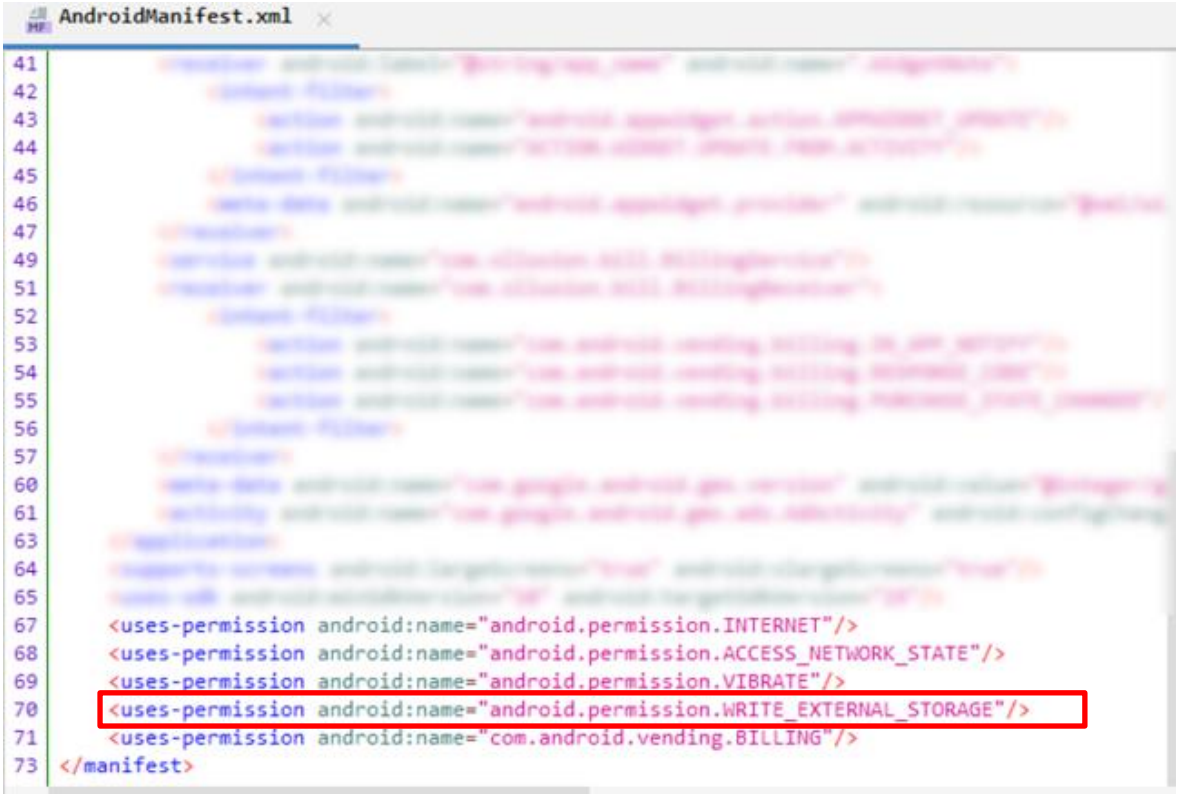


Obrázok 35. Poznámka vložená v aplikácii [zdroj vlastný]

V tejto aplikácii bude vytvorená jedna poznámka, v ktorej bol názov „test“ a obsah „toto je testovacia sprava“. Na prezeranie štruktúry dekompilovaného súboru je použitý nástroj JADX, kvôli jeho prehľadnému zobrazovaniu zdrojového kódu.



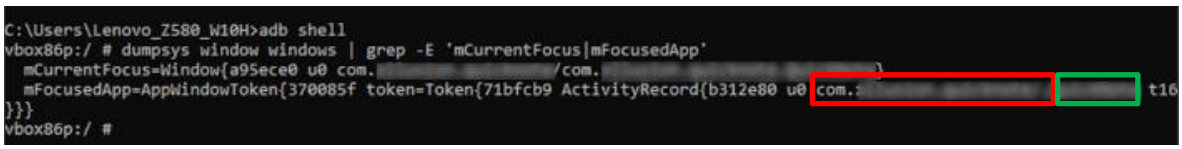
Obrázok 36. Obsah aplikácie otvorený v nástroji JADX [zdroj vlastný]



Obrázok 37. Povolenia použité v súbore AndroidManifest.xml [zdroj vlastný]

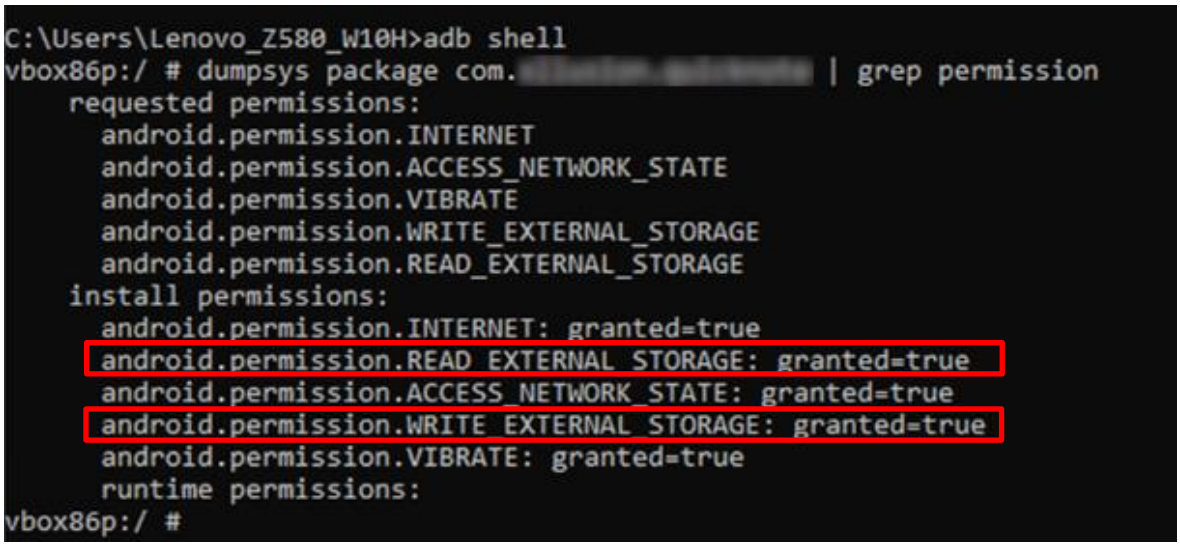
Pri prezeraní súboru *AndroidManifest.xml* je zaujímavé pridelenie povolenia, ktoré umožňuje zápis do externej pamäte. Toto povolenie môže byť nebezpečné, nakoľko môže viesť k tzv. Man-In-The-Disk útoku, ktorý bude podrobnejšie rozobraný neskôr.

Prvým krokom statickej analýzy bude prezrenie aplikácie pomocou ADB shell príkazového riadku a príkazu *dumpsys window windows | grep -E 'mCurrentFocus|mFocusedApp',* ktorý ukáže názov balíčka a aktuálnu aktivitu spustenej aplikácie.



Obrázok 38. Zobrazenie názvu balíka a spustenej aktivity [zdroj vlastný]

Po získaní informácií o balíčku je možné sa dostať k zobrazeniu povolení, a overenie či má táto aplikácia naozaj prístup k tomuto povoleniu. Na zobrazenie povolení je použitý príkaz *dumpsys package <balíček> | grep permission*



Obrázok 39. Zobrazenie povolení ktoré používa aplikácia [zdroj vlastný]

Ako je viditeľné po zadaní príkazu, ktorý zobrazí povolenia pre daný aplikačný balík, táto aplikácia má udelené povolenia pre čítanie a zápis do externého úložiska. Neopatrné používanie externého úložiska aplikáciami môže otvoriť dvere útoku vedúcemu k ľubovoľnému množstvu nežiaducich výsledkov, ako je tichá inštalácia nevyžiadaných, potenciálne škodlivých aplikácií do telefónu používateľa, alebo zmena dát nachádzajúcich sa na tomto úložisku. Útoky typu *Man-in-the-Disk* sú možné, keď aplikácie nedbajú na správne používanie externého úložiska, teda zdroja, ktorý je zdieľaný medzi všetkými aplikáciami a nepoužíva Android Sandbox. Ak sa nepoužijú bezpečnostné opatrenia, aplikácie sú zraniteľné voči rizikám škodlivej manipulácie s údajmi. Toto teda môže slúžiť ako príklad zraniteľnosti *Improper Platform Usage*, teda nesprávne použitie platformy a v ďalšej časti bude ukázané, ako sa dá takáto zraniteľnosť zneužiť.

V súbore Manifest sa nachádzajú zoznamy všetkých aktivít, ktoré aplikácia využíva, z týchto aktivít bude bližšie rozobraná aktivita s názvom „.EditNote“, ktorá slúži na editovanie poznámok v tejto aplikácii.

Obrázok 40. Zraniteľná aktivita v aplikácii [zdroj vlastný]

Podľa názvu bola v zdrojovom kóde dohľadaná trieda „EditNote“, ktorá obsahuje kód, ktorý sa vykonáva pri editovaní záznamov v tejto aplikácii. Je viditeľné, že vstupné reťazce sú vkladané pomocou *intent.extra*, ktoré môžu byť zmenené a tým dosadené vlastné informácie.

AndroidManifest.xml | EditNote

```
            }
69          for (int i2 = lineCount; i2 < lineCount + 20; i2++) {
71              int lineBounds2 = (getLineBounds(0, rect) * (i2 - lineCount)) + getLineBounds(lineCoun
73              canvas.drawLine(rect.left, lineBounds2, rect.right, lineBounds2, paint);
            }
76          super.onDraw(canvas);
        }
    }

    @Override // android.app.Activity
81  public void onCreate(Bundle bundle) {
82      super.onCreate(bundle);
83      getWindow().setSoftInputMode(3);
84      setContentView(C0001R.layout.edit_note);
85      a();
86      this.b = new c(this);
88      Bundle extras = getIntent().getExtras();
90      if (extras != null) {
93          String string = extras.getString("android.intent.extra.SUBJECT");
94          String string2 = extras.getString("android.intent.extra.TEXT");
96          if (string != null) {
                this.e.setText(string);
            }
97          if (string2 != null) {
                this.f.setText(string2);
```

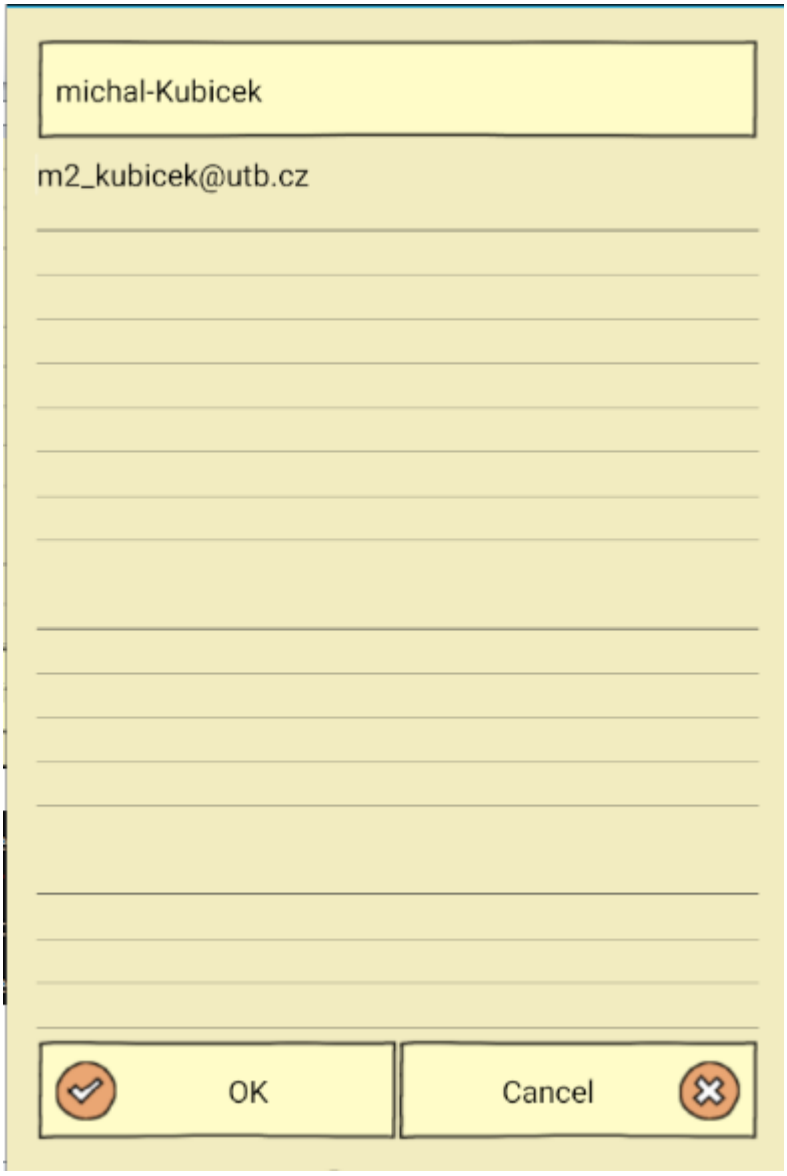
Obrázok 41. Zdrojový kód zraniteľnej aktivity [zdroj vlastný]

Pomocou príkazu *adb shell am start –n <balíček>/.<Aktivita> -e <android.intent.extra.SUBJECT (zvolený text) > –e <android.intent.extra.TEXT (zvolený text)>*“. Je možné do týchto premenných dosadiť ľubovolné dáta.

```
C:\WINDOWS\system32> adb shell am start -n com.                    /.EditNote -e
android.intent.extra.SUBJECT michal-Kubicek -e android.intent.extra.TEXT m2_k
ubicek@utb.cz
Starting: Intent { cmp=com.                    /.EditNote (has extras) }

C:\WINDOWS\system32>
```

Obrázok 42. Vkladanie vlastných údajov cez intent.extra [zdroj vlastný]

Po prevedení príkazu je vidieť, že sa zadané vstupy naozaj zobrazia v tejto aplikácii na miestach, kde to bolo predpokladané.



Obrázok 43. Výsledná zmena poznámky [zdroj vlastný]

4.2.2 JADX

Tento nástroj je obľúbený kvôli jeho GUI, teda užívateľskému rozhraniu, ktoré je veľmi prívetivé a intuitívne. Navyše obsahuje zvýraznenie kľúčových slov a syntaxu, takže manuálna statická analýza v ňom je veľmi prehľadná . Na ukážku použitia tohoto nástroja bude využitá často sa objavujúca zraniteľnosť súvisiaca s nesprávnym uložením dát,

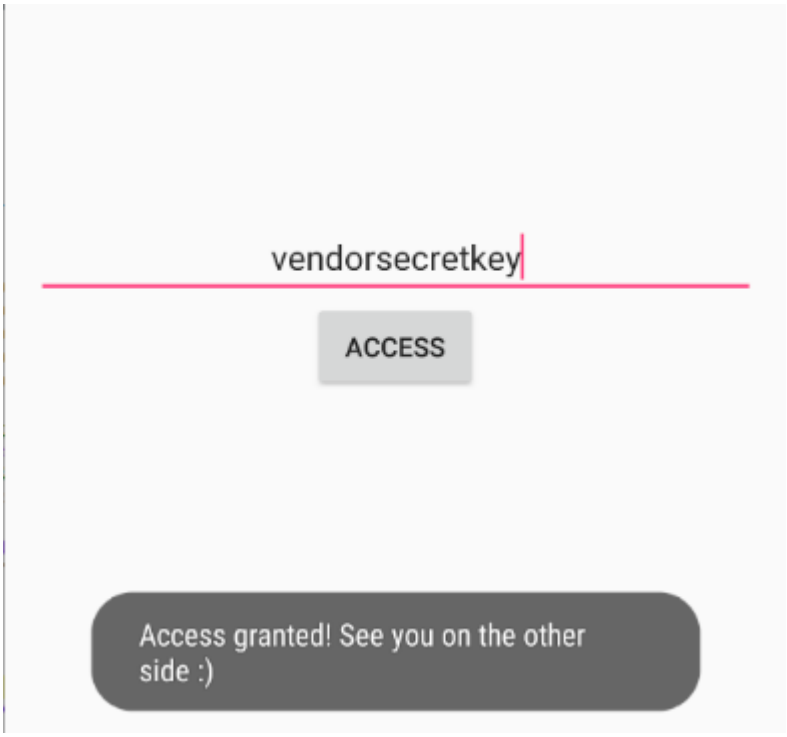
nazývaná aj *hard-coded credentials*. Jedná sa o dôležité a citlivé informácie nechané v kóde. Napríklad heslá, prihlasovacie údaje a podobne.

V ukážke je použitá aplikácia DIVA, ktorá má simulovať zraniteľnú aplikáciu, v ktorej sa nachádzajú chyby. Táto aplikácia je vyvinutá za cieľom edukácie v oblasti penetračného testovania a reverzného inžinierstva. Obsahuje viacero zraniteľností a cieľom je naučiť testerov ako tieto chyby a zraniteľnosti odhaliť a zneužiť.

Pri prezeraní kódu v aplikácii bola zachytená metóda *access*, ktorá slúži na odomknutie prístupu do ďalšej časti. Náchádza sa tu však vyššie spomenutá chyba, kedy je kľúčový reťazec v nezašifrovanej forme použitý v kóde s cieľom porovnania s reťazcom, ktorý zadá používateľ. Týmto je teda odhalené heslo potrebné pre vstup do ďalšej úrovne. Zanechanie citlivých údajov v kóde môže byť zaradené do zraniteľnosti *Improper data storage*, teda nesprávne ukladanie dát.

```
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

/* loaded from: classes.dex */
10 public class HardcodeActivity extends AppCompatActivity {
    /* JADX INFO: Access modifiers changed from: protected */
    @Override // android.support.v7.app.AppCompatActivity, android.support.v4.app.FragmentActivity, andro
11  public void onCreate(Bundle savedInstanceState) {
12      super.onCreate(savedInstanceState);
13      setContentView(R.layout.activity_hardcode);
    }

16  public void access(View view) {
19      if (((EditText) findViewById(R.id.hcKey)).getText().toString().equals("vendorsecretkey")) {
17          Toast.makeText(this, "Access granted! See you on the other side :)", 0).show();
        } else {
17          Toast.makeText(this, "Access denied! See you in hell :D", 0).show();
        }
    }
}
```

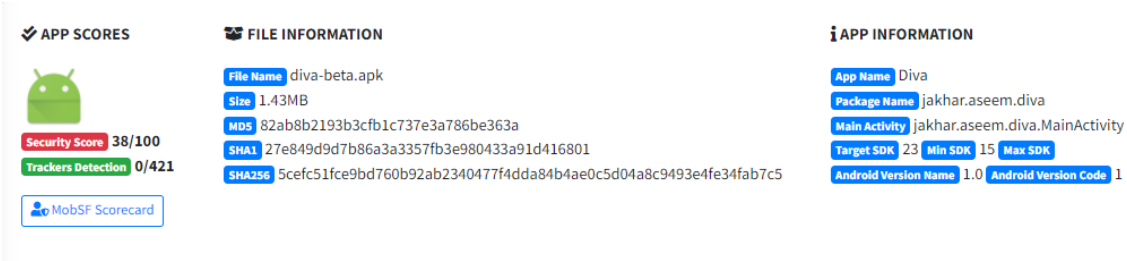
Obrázok 44. Nájdené heslo zanechané v kóde [zdroj vlastný]



Obrázok 45. Získanie prístupu pomocou nájdeného hesla [zdroj vlastný]

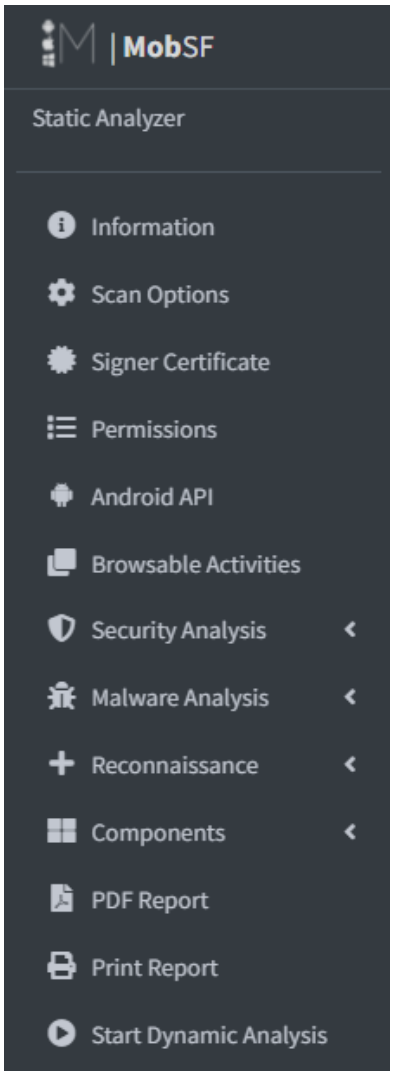
4.2.3 MobSF

Po vložení APK súboru do nástroja MobSF začne prebiehať analýza. Keď je analýza hotová, nástroj zobrazí report, v ktorom sa dajú nájsť všetky potrebné informácie.



Obrázok 46. Detaily o aplikácii zobrazené pomocou MobSF [zdroj vlastný]

Ako prvé sú zobrazené detaily o aplikácii vrátane mien, veľkosti, SHA hashov, a podobne.



Obrázok 47. Panel nástrojov dostupných v MobSF [zdroj vlastný]

Ďalším hlavným prvkom je navigačný panel na boku obrazovky, ktorý obsahuje kapitoly celého reportu, takže vďaka nemu môžu byť rozkliknuté a zobrazené ktorékoľvek súčasti.

</> CODE ANALYSIS

Search:

NO	ISSUE	SEVERITY	STANDARDS	FILES
1	The App logs information. Sensitive information should never be logged.	info	**CWE:** CWE-532: Insertion of Sensitive Information into Log File **OWASP MASVS:** MSTG-STORAGE-3	jakhar/aseem/diva/InsecureDataStorage3Activity.java jakhar/aseem/diva/AccessControl1Activity.java jakhar/aseem/diva/InsecureDataStorage4Activity.java jakhar/aseem/diva/AccessControl2Activity.java jakhar/aseem/diva/SQLInjectionActivity.java jakhar/aseem/diva/InsecureDataStorage2Activity.java jakhar/aseem/diva/LogActivity.java

Obrázok 48. Chyba nájdená v aplikácii pomocou statickej analýzy nástrojom MobSF [zdroj vlastný]

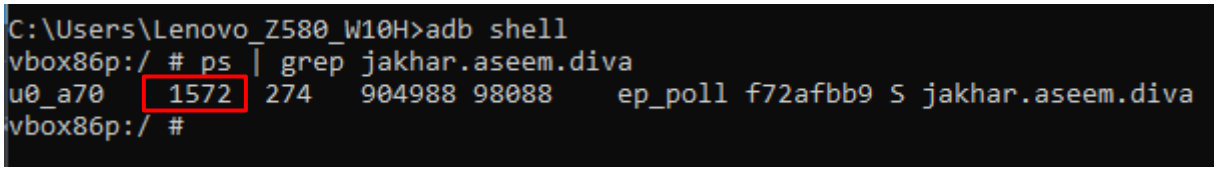
Pri tejto aplikácii analýza kódu ukázala, že aplikácia nebezpečne loguje citlivé dáta. Po rozkliknutí súboru typu *.java* je viditeľný vyznačený riadok kódu, ktorý tejto zraniteľnosti zodpovedá.

LogActivity.java

```
1.  package jakhar.aseem.diva;
2.
3.  import android.os.Bundle;
4.  import android.support.v7.app.AppCompatActivity;
5.  import android.util.Log;
6.  import android.view.View;
7.  import android.widget.EditText;
8.  import android.widget.Toast;
9.  /* loaded from: classes.dex */
10. public class LogActivity extends AppCompatActivity {
11.     /* JADX INFO: Access modifiers changed from: protected */
12.     @Override // android.support.v7.app.AppCompatActivity, android.support.v4.app.FragmentActivity, android.support.v4
13.     public void onCreate(Bundle savedInstanceState) {
14.         super.onCreate(savedInstanceState);
15.         setContentView(R.layout.activity_log);
16.     }
17.
18.     public void checkout(View view) {
19.         EditText cctxt = (EditText) findViewById(R.id.ccText);
20.         try {
21.             processCC(cctxt.getText().toString());
22.         } catch (RuntimeException e) {
23.             Log.e("diva-log", "Error while processing transaction with credit card: " + cctxt.getText().toString());
24.             Toast.makeText(this, "An error occured. Please try again later", 0).show();
25.         }
26.     }
27.
28.     private void processCC(String ccstr) {
29.         throw new RuntimeException();
30.     }
31. }
```

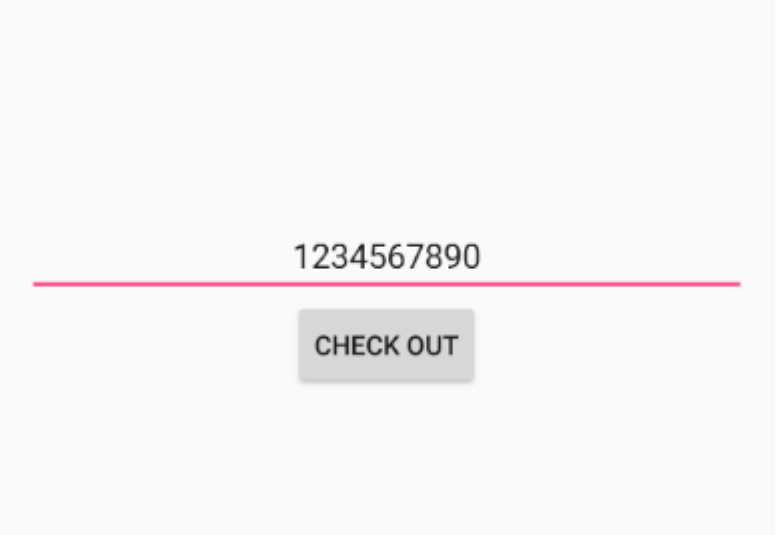
Obrázok 49. Kód s nebezpečným logovaním informácií [zdroj vlastný]

Aby mohla byť táto zraniteľnosť využitá, musí byť nadobudnutý prístup k bežiacej aplikácii v emulátori, každá aplikácia má priradené PID, ktoré je možné zistiť. Na túto operáciu je použitý Android Debug Bridge Shell, kde je pomocou príkazu *ps | grep <package-name>* zobrazený status procesov aplikácie. Názov balíka, ktorý je v tomto prípade potrebný zobrazí nástroj MobSF hneď na začiatku reportu. V prípade tejto aplikácie je názov balíka *jakhar.aseem.diva*.

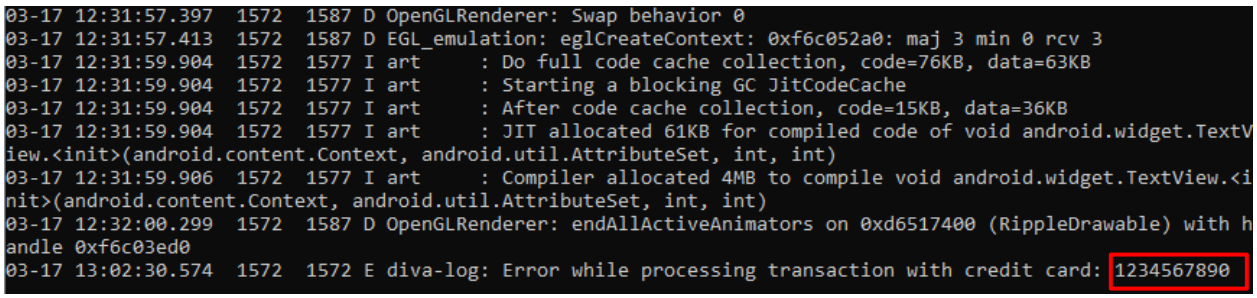


Obrázok 50. Zistenie PID aplikácie pomocou ADB shell [zdroj vlastný]

V zobrazených informáciách je na druhom mieste PID aplikácie, ktoré bude potrebné v ďalšom priebehu. Následne je vypísaný log aplikácie s týmto PID pomocou príkazu *logcat | grep <PID>*, kde je vidno, že po zadaní hodnoty do aplikácie, sa táto hodnota zobrazí v logu.



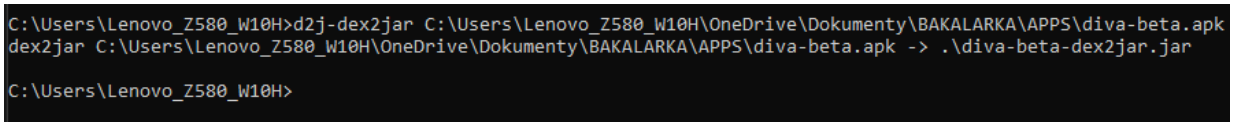
Obrázok 51. Zadanie údajov do aplikácie [zdroj vlastný]



Obrázok 52. Zobrazenie zadanej informácie v logu [zdroj vlastný]

4.2.4 Dex2jar a JD-GUI

Pri ukážke práce s týmito nástrojmi bude opäť použitá DIVA aplikácia. Ukážka *dex2jar* a *JD-GUI* nástrojov bude demonštrovaná spolu, nakoľko sa dopĺňajú a lepší prínos majú ak sa využívajú spolu. Pomocou príkazu *d2j-dex2jar <cesta_k_apk>* je z aplikácie spravený súbor *jar*, čo je vlastne archív súborov Java.

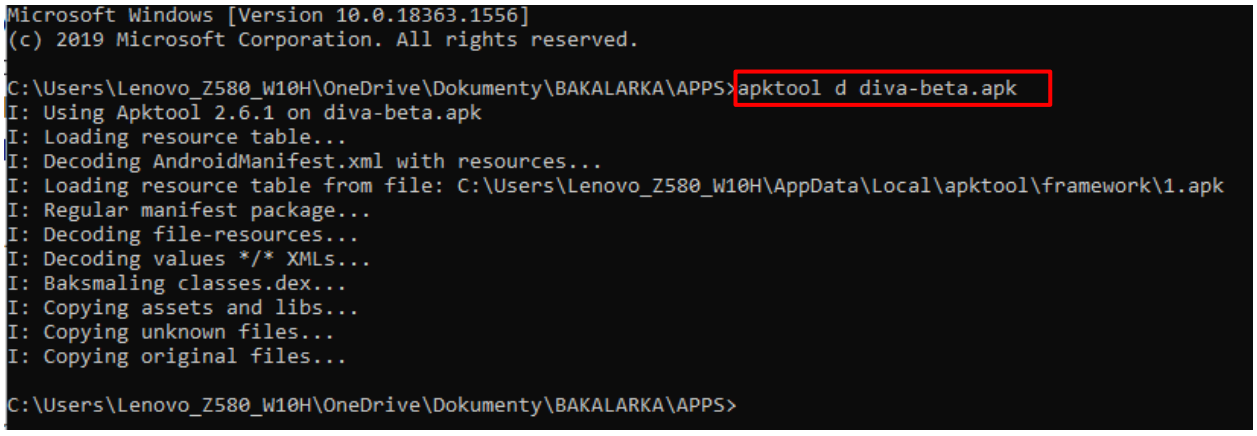


Obrázok 53. Dex2Jar dekompilácia aplikácie [zdroj vlastný]

Výstupom teda bude súbor *diva-beta-dex2jar.jar*, ktorý je výstupom z programu dex2jar. Ďalej môže byť otvorený nástroj JD-GUI a do neho potom vložený výstupný súbor *jar*. Následne nástroj zobrazí komponenty tohoto súboru.

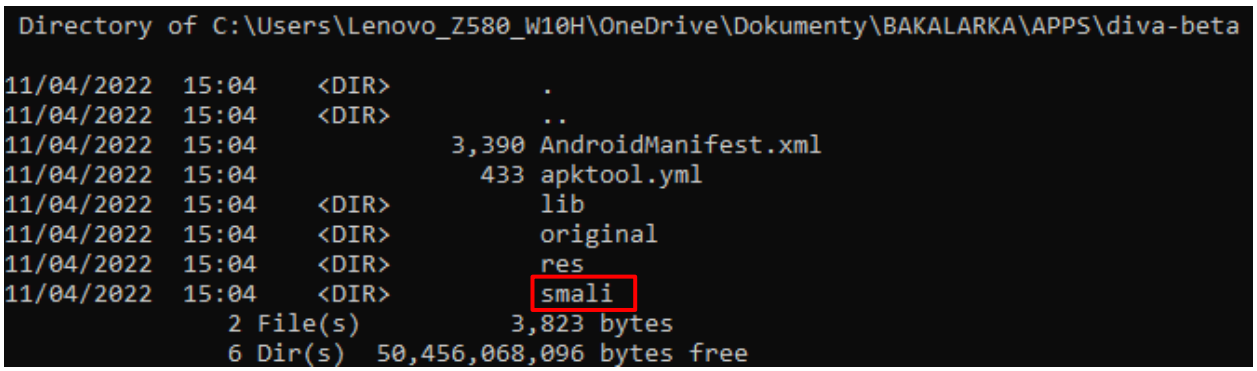
Obrázok 54. Zobrazenie hesla zanechaného v kóde pomocou JD-GUI [zdroj vlastný]

V tomto nástroji je viditeľné pekné zobrazenie štruktúry súborov *class* a následne aj kódu, ktorý sa v nich nachádza. Ako už bolo pri jednej predchádzajúcich ukážiek zistené, v jednom z týchto súborov je heslo vo formáte textu zanechané v zdrojovom kóde. Preto bude v tejto ukážke cieľom zmeniť ho pomocou modifikácie *smali*. Aby bol získaný prístup k súborom obsahujúcim kód smali, je potrebné túto aplikáciu dekompilovať pomocou nástroja APKtool.



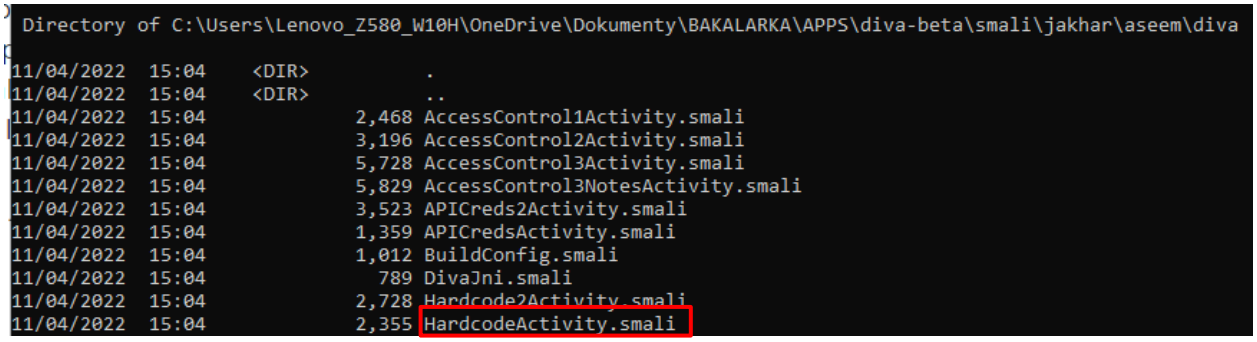
Obrázok 55. Dekompilácia aplikácie nástrojom APKTool s cieľom získania Smali [zdroj vlastný]

Po dekompilovaní je možné nahliadnuť do obsahu tejto aplikácie a lokalizovať zložku, v ktorej sa nachádzajú súbory smali.



Obrázok 56. Zobrazenie dekompilovanej aplikácie so zložkou Smali [zdroj vlastný]

V adresárovej štruktúre je vidno, že priamo v dekompilovanej aplikácii je adresár *smali*. Táto zložka obsahuje viacero podzložiek. Po hľadaní boli úspešne nájdené súbory typu *smali*, ktoré presne zodpovedajú súborom typu *class*, ktoré boli zobrazené a v nich nájdené heslo.



Obrázok 57. Cieľový súbor obsahujúci kód v jazyku Smali [zdroj vlastný]

Potom môže byť tento súbor otvorený v ľubovolnom textovom editore, napríklad aj v bežnom notepade. Toto je vykonané príkazom *notepad <názov_súboru>*

```
C:\Users\Lenovo_Z580_W10H\OneDrive\Dokumenty\BAKALARKA\APPS\diva-beta\smali\jakhar\aseem\diva>
notepad HardcodeActivity.smali

C:\Users\Lenovo_Z580_W10H\OneDrive\Dokumenty\BAKALARKA\APPS\diva-beta\smali\jakhar\aseem\diva>
```

Obrázok 58. Otvorenie súboru Smali v poznámkovom bloku [zdroj vlastný]

Jazyk Smali je transformovaný dex formát, ktorý je používaný Dalvikom, takže jeho štruktúra môže byť neprehľadná. Ak je však cieľ hľadania známy, nemal by byť problém sa v tomto súbore zorientovať.

```
.line 19
.local v0, "hckey":Landroid/widget/EditText;
invoke-virtual {v0}, Landroid/widget/EditText;->getText()Landroid/text/Editable;

move-result-object v1

invoke-virtual {v1}, Ljava/lang/Object;->toString()Ljava/lang/String;

move-result-object v1

const-string v2, "vendorsecretkey"

invoke-virtual {v1, v2}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

move-result v1

if-eqz v1, :cond_0
```

Obrázok 59. Nájdenie hesla v zdrojovom kóde súboru typu Smali [zdroj vlastný]

Po prezretí súboru bol nájdený reťazec, ktorý bol hľadaný. V ďalšom kroku bol string modifikovaný a upravená tak funkčnosť aplikácie.

```
const-string v2, "michalkubicek"
```

Obrázok 60. Zmena hesla v kóde súboru typu Smali [zdroj vlastný]

Pre túto ukážku bolo zmenené heslo, ktorým je možné sa dostať do chránenej časti aplikácie. Tu začína druhá fáza, v ktorej bude vykonané spätné zostavenie a podpísanie súboru tak, aby mohol byť nainštalovaný na zariadenie.

```
C:\Users\Lenovo_Z580_W10H\OneDrive\Dokumenty\BAKALARKA\APPS>apktool b diva-beta
I: Using Apktool 2.6.1
I: Checking whether sources has changed...
I: Smaling smali folder into classes.dex...
I: Checking whether resources has changed...
I: Building resources...
I: Copying libs... (/lib)
I: Building apk file...
I: Copying unknown files/dir...
I: Built apk...

C:\Users\Lenovo_Z580_W10H\OneDrive\Dokumenty\BAKALARKA\APPS>
```

Obrázok 61. Kompilácia modifikovanej aplikácie na súbor APK [zdroj vlastný]

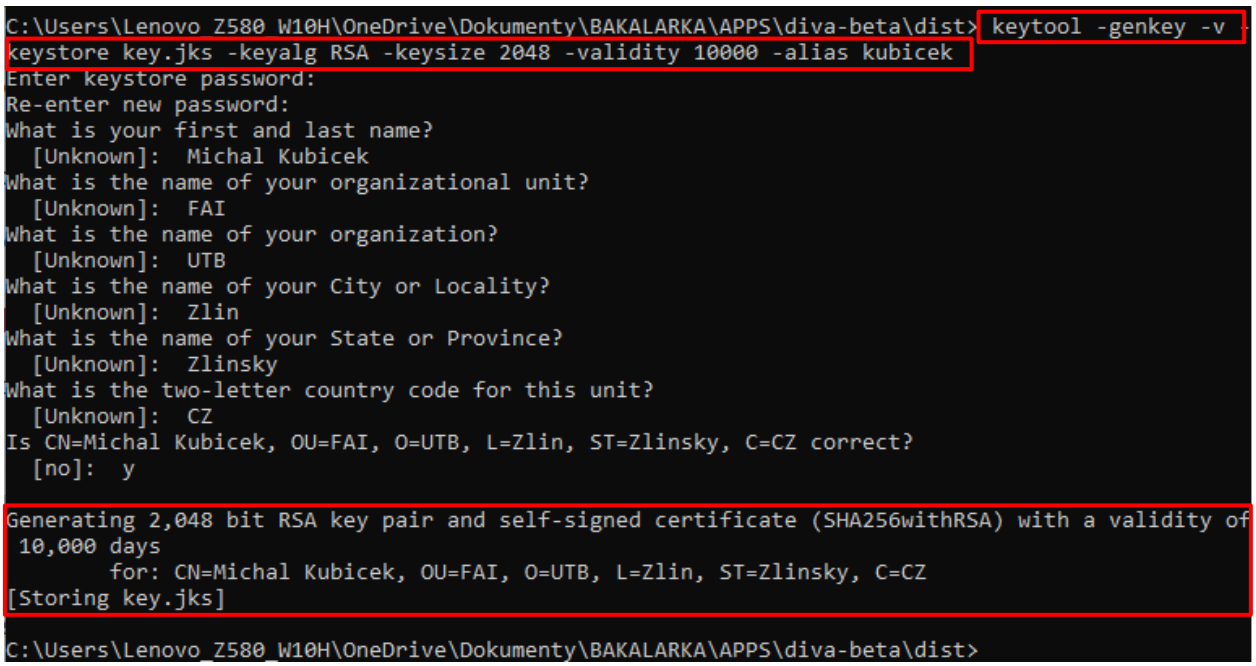
Pomocou nástroja APKtool je zostavená táto aplikácia aj s modifikovaným súborom. Príkaz *apktool b <adresár_s_rozbalenou_aplikáciou>* zostaví túto aplikáciu a umiestni ju do zložky *dist* v tomto adresári.

```
 Directory of C:\Users\Lenovo_Z580_W10H\OneDrive\Dokumenty\BAKALARKA\APPS\diva-beta\dist

11/04/2022  15:21    <DIR>          .
11/04/2022  15:21    <DIR>          ..
11/04/2022  15:21         1,473,818 diva-beta.apk
               1 File(s)      1,473,818 bytes
               2 Dir(s)  50,022,825,984 bytes free

C:\Users\Lenovo_Z580_W10H\OneDrive\Dokumenty\BAKALARKA\APPS\diva-beta\dist>
```

Obrázok 62. Lokalizovanie kompilovaného súboru [zdroj vlastný]

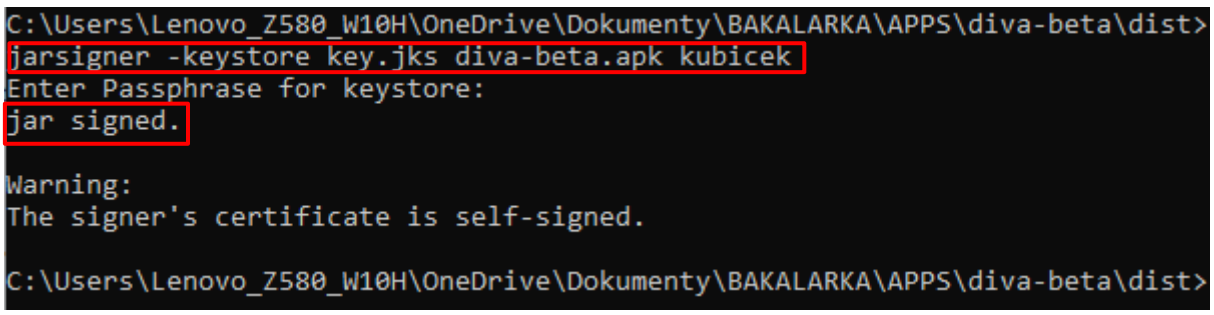
Následne už len zostáva túto aplikáciu podpísať. Na toto je použitý nástroj, ktorý obsahuje Java, nazýva sa *keytool*. Je to nástroj na správu kľúčov a certifikátov a umožňuje používateľom spravovať vlastné verejné a súkromné kľúče a certifikáty pri vlastnej autentizácii.

Pomocou príkazu *keytool –genkey –v –keystore key.jks –keyalg RSA –keysize 2048 –validity 10000 –alias kubicek,* kde je zvolený názov keystore, algoritmus, ktorý bude použitý, veľkosť kľúča, dĺžku platnosti a alias, bude vytvorený náhodne vygenerovaný kľúč. Po zadaní tohoto príkazu je ešte potrebné zadať heslo a vyplniť požadované informácie.



Obrázok 63. Vygenerovanie kľúča na podpis [zdroj vlastný]

Týmto úkonom je vygenerovaný certifikát a už stačí aplikáciu len podpísať. Na toto taktiež bol využitý nástroj, ktorý ponúka sama Java. Nazýva sa *jarsigner*. Tento nástroj slúži na podpisovanie a overenie Java archívov, teda *jar* súborov.



Obrázok 64. Podpísanie aplikácie pomocou nástroja Jarsigner [zdroj vlastný]

Po zadaní príkazu *jarsigner –keystore <názov_keystore> <cesta_k_apk> <alias>* sa zobrazí výzva na zadanie hesla. Po správnom zadaní hesla k certifikátu je *jar* súbor podpísaný.

Následne zostáva už len overiť, či je kód zmenený a vyskušať túto modifikovanú aplikáciu na zariadení. Tu je zopakovaný rovnaký proces, pomocou nástroja dex2jar je z aplikácie spravený súbor *jar* a ten následne otvorený v nástroji JD-GUI. Po zobrazení súboru *class*, v ktorom sa nachádza heslo je viditeľné, že toto heslo je úspešne zmenené.

HardcodeActivity.class

```
package jakhar.aseem.diva;

import android.content.Context;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class HardcodeActivity extends AppCompatActivity {
  public void access(View paramView) {
    if (((EditText)findViewById(2131492987)).getText().toString().equals("michalkubicek")) {
      Toast.makeText((Context)this, "Access granted! See you on the other side :)", 0).show();
      return;
    }
    Toast.makeText((Context)this, "Access denied! See you in hell :D", 0).show();
  }

  protected void onCreate(Bundle paramBundle) {
    super.onCreate(paramBundle);
    setContentView(2130968607);
  }
}
```

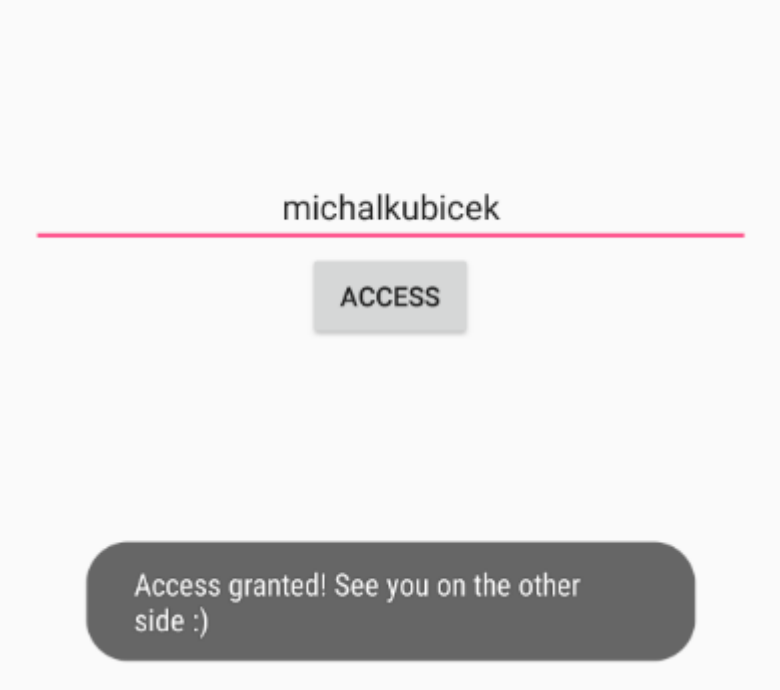
Obrázok 65. Zobrazenie zmeneného hesla v kóde pomocou JD-GUI [zdroj vlastný]

Už zostáva aplikáciu len nainštalovať na zariadenie a vyskúšať túto zmenu v praxi. Nainštalovať túto aplikáciu je možné pomocou nástroja *ADB* a príkazom *install*.

```
C:\Users\Lenovo_Z580_W10H\OneDrive\Dokumenty\BAKALARKA\APPS\diva-beta\dist>adb install diva-beta.apk
Performing Streamed Install
Success

C:\Users\Lenovo_Z580_W10H\OneDrive\Dokumenty\BAKALARKA\APPS\diva-beta\dist>
```

Obrázok 66. Nainštalovanie modifikovanej aplikácie [zdroj vlastný]

Po spustení aplikácie a zadaní nového hesla je viditeľné, že overenie prebehlo úspešne, takže zmena aplikácie pomocou modifikácie smali kódu bola úspešná.



Obrázok 67. Úspešná skúška prihlásenia novým heslom [zdroj vlastný]

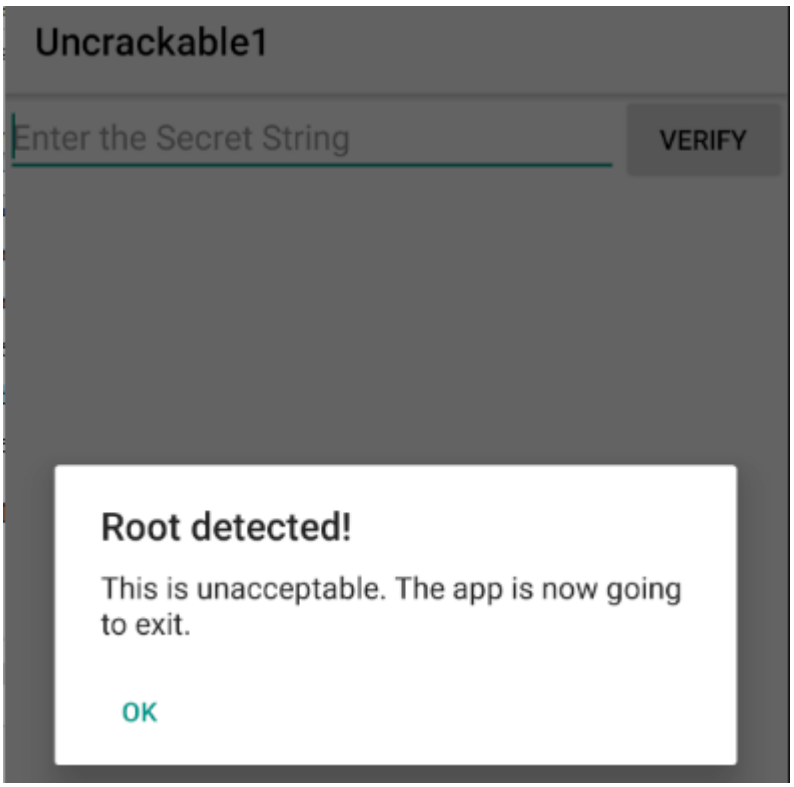
5 PRAKTICKÁ DEMONŠTRÁCIA SÚČASNÝCH SOFTVÉROVÝCH NÁSTROJOV PRE DYNAMICKÚ ANALÝZU

5.1 Frida

Praktické použitie tohoto nástroja bude ukázané na bypasse root detekcie. Rootovanie je proces, pri ktorom je odomknuté Android zariadenie tak, aby bol nadobudnutý prístup k privilegovaným prvkom, ktoré sú pre bežného úžívateľa nedostupné. Zjednodušene to môže byť popísané ako získanie administrátorského prístupu do systému. Keďže pri tomto špeciálnom prístupe je možnosť vidieť a robiť úkony, ktoré klasický úžívateľ nemôže, predstavuje to pre tvorcov aplikácie potenciálne riziko. Preto sú niektoré aplikácie vybavené takzvanou detekciou Rootu, to znamená, že vedia odhaliť či je zariadenie v tomto režime alebo nie. Útočníci často používajú rootovanie na lepší prehľad o tom, ako sa aplikcia správa počas behu, alebo na úpravu aplikácie počas spustenia.

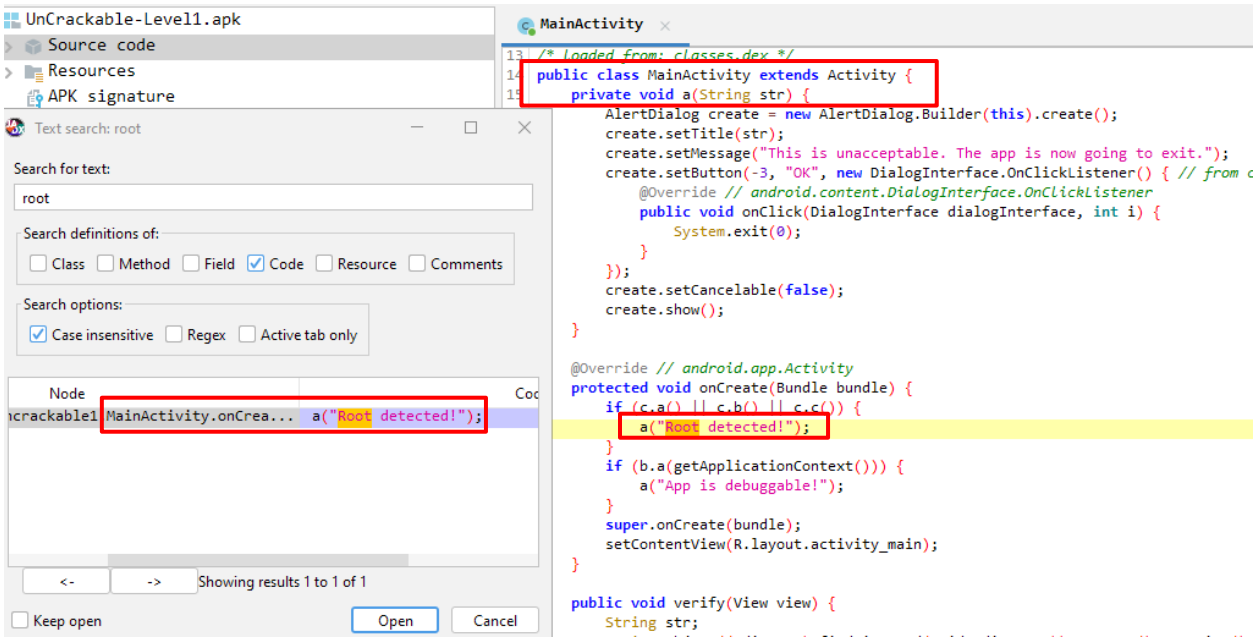
Keďže rootované zariadenie je oveľa viac ohrozené, je dôležité implementovať nástroje ktoré dokážu root odhaliť. Toto odhalenie je nevyhnutné pre zabezpečenie toho, aby sa aplikácie otvárali len na mieste, kde je to bezpečné. Automatická detekcia rootu môže ľahko odhaliť príznaky zariadenia v režime root a následne vypnúť aplikáciu. Rôzne flexibilné nástroje potom môžu ovplyvniť ako bude reakcia na prítomnosť rootu reagovať. Môžu napríklad obmedziť niektoré funkcie miesto toho, aby zastavili celú aplikáciu. Ak bude potrebné do aplikácie zasahovať alebo mať do nej plný prístup aj v režime root, je nutné túto detekciu obísť.

Najskôr je spustená aplikácia na zariadení s rootom aby bolo zistené či má túto detekciu.



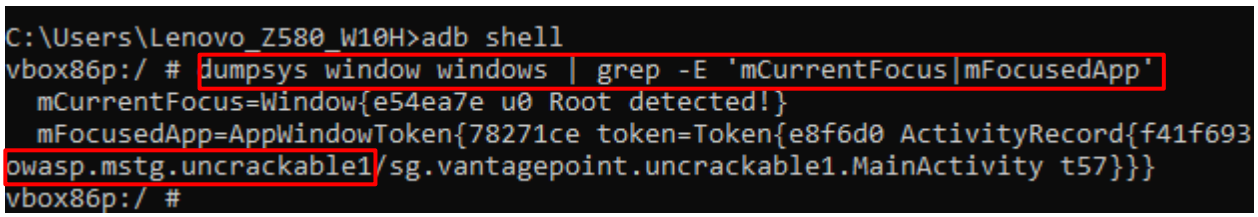
Obrázok 68. Aplikácia s detekciou Rootu [zdroj vlastný]

Ako je viditeľné, po spustení sa ihneď zobrazila správa, že je detekovaný root a aplikácia sa zavrie. Preto bolo ďalším krokom prezrenie kódu tejto aplikácie. Po rozbalení a spustení fukncie vyhľadať a vyhľadaní kľúčového slovo *root*, nakoľko sa zobrazuje aj v samotnej správe, je nájdený potrebný kód.



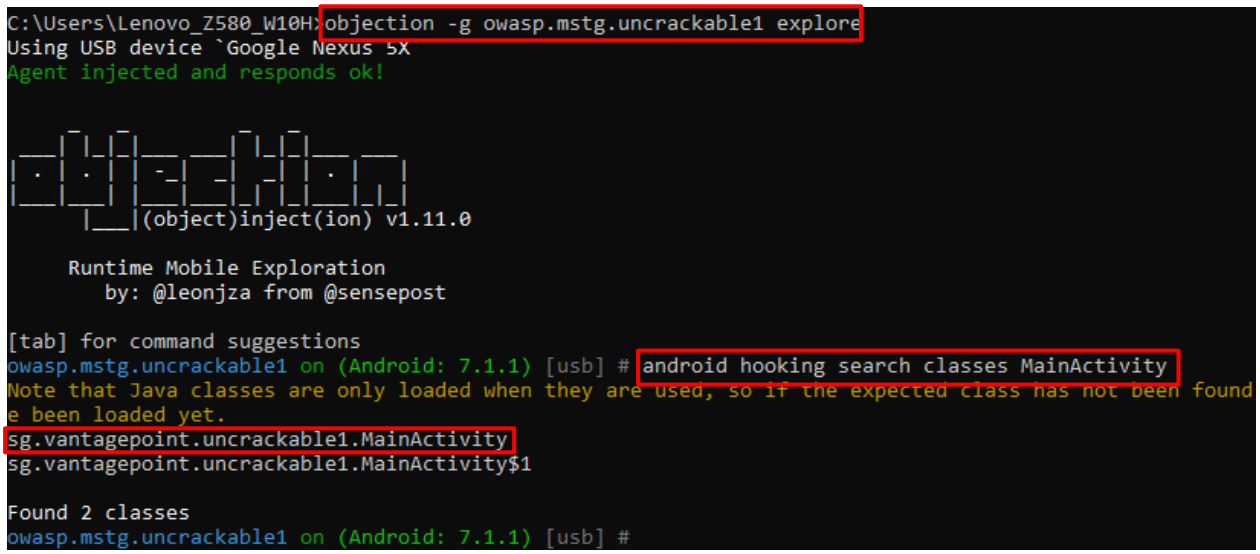
Obrázok 69. Vyhľadanie cieľovej aktivity zodpovednej za detekciu Rootu [zdroj vlastný]

Po vyhľadaní je badateľné, že v aktivite *MainActivity* bola nájdená metóda, ktorá obsahuje toto kľúčové slovo. Následne pomocou príkazu *dumpsys window windows | grep -E 'mCurrentFocus|mFocusedApp'* v *adb shell* je vypísaný názov balíčka spustenej aplikácie.



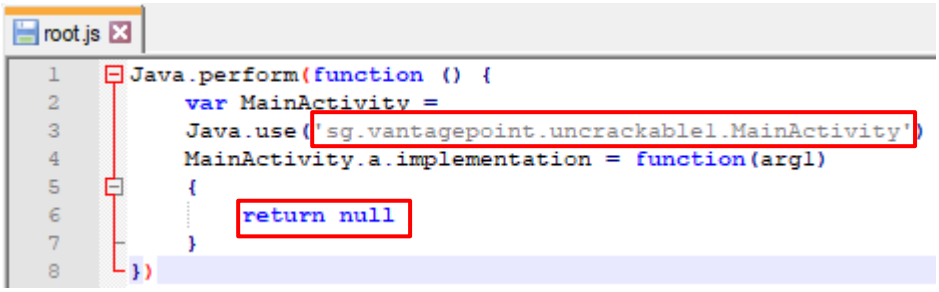
Obrázok 70. Vypísanie názvu balíčka cez ADB shell

Následne môže byť vyskúšané vytvorenie skriptu, ktorý bude vložený do aplikácie. Keďže z kódu je známe, že o detekciu rootu sa stará metóda *a* v triede *MainActivity*, daný skript bude zameraný na túto metódu a miesto vrátenia hodnoty bude vracať *null*. Na zistenie presného názvu triedy môže použité rozšírenie *objection*, kde pomocou príkazu *objection-g <package> explore* je otvorená aplikácia a následne do nej môže byť zasahované pomocou tzv. *Android hooking*. Týmto sú vyhľadané aktívne *classes*.



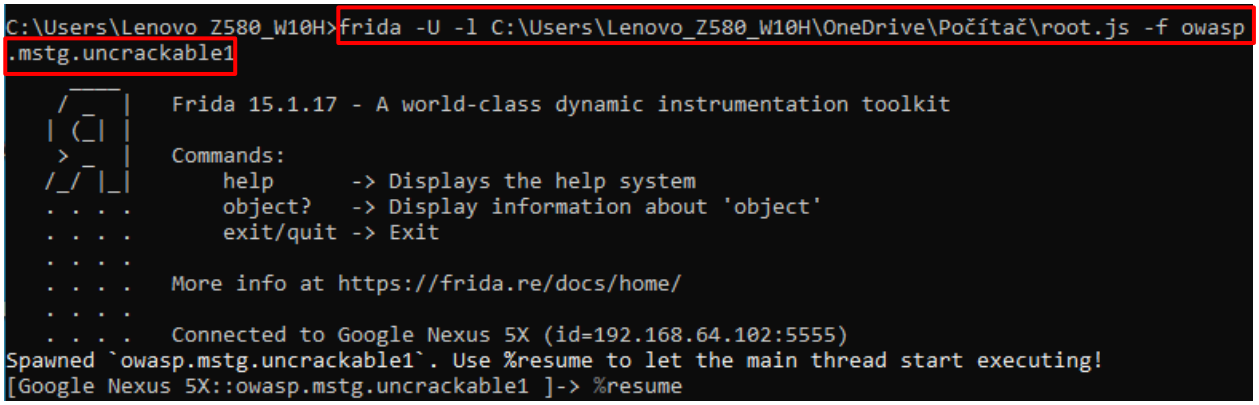
Obrázok 71. Zobrazenie názvu aktivity pomocou rozšírenia Objection [zdroj vlastný]

Keď je nájdený presný názov class, je možné vytvoriť script, v ktorom bude v *MainActivity* použitá spomínaná trieda a vrátený *null*.



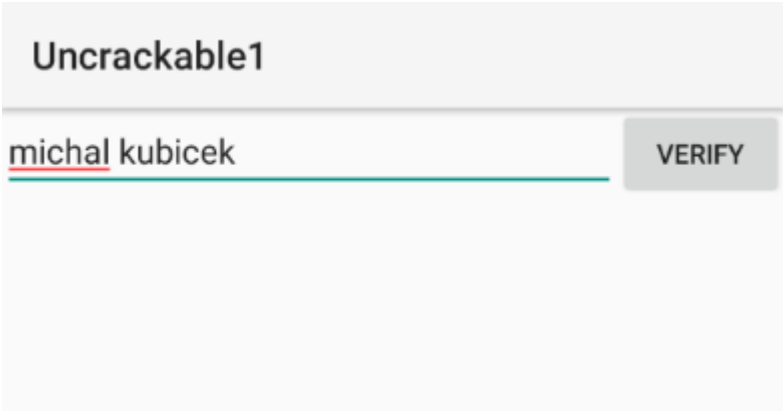
Obrázok 72. Vytvorenie vlastného scriptu na obídenie detekcie Rootu [zdroj vlastný]

Následne tento script môže byť vložený do aplikácie pomocou nástroja Frida. V príkazovom riadku sa zadaním príkazu *frida –U –l <cesta k skriptu ktorý je vkladaný> -f <package cieľovej aplikácie>* vloží skript.



Obrázok 73. Injection scriptu do aplikácie [zdroj vlastný]

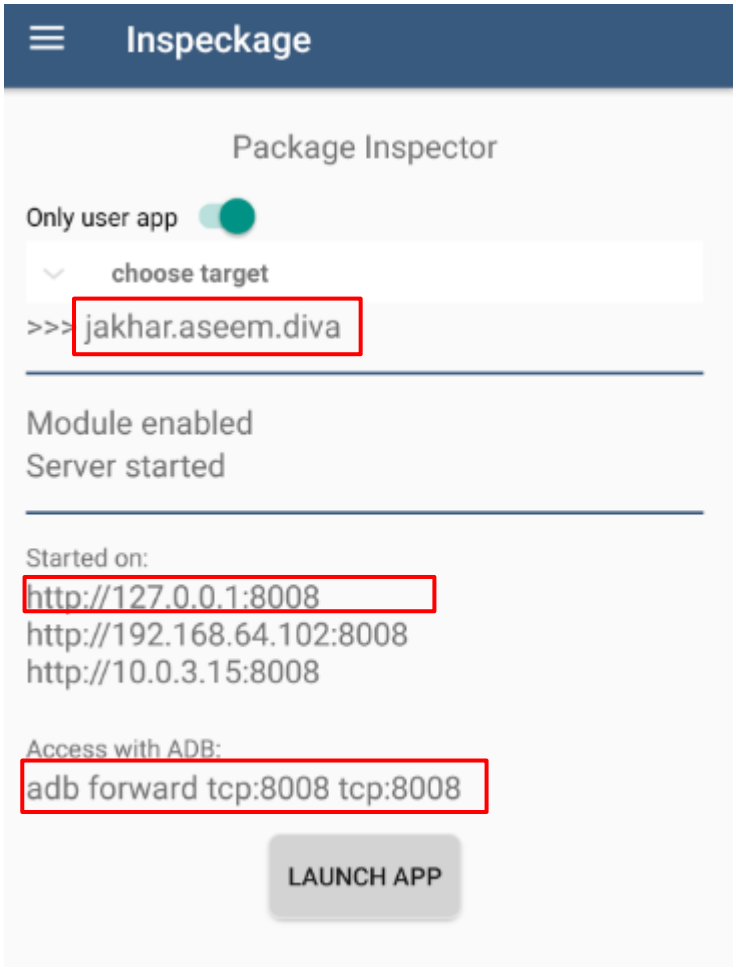
To vo výsledku vedie k “oklamaniu” aplikácie a tak nie je root detekovaný. Následne je možné bez problémov pristupovať do aplikácie.



Obrázok 74. Úspešné obídenie Rootu a získanie prístupu do aplikácie [zdroj vlastný]

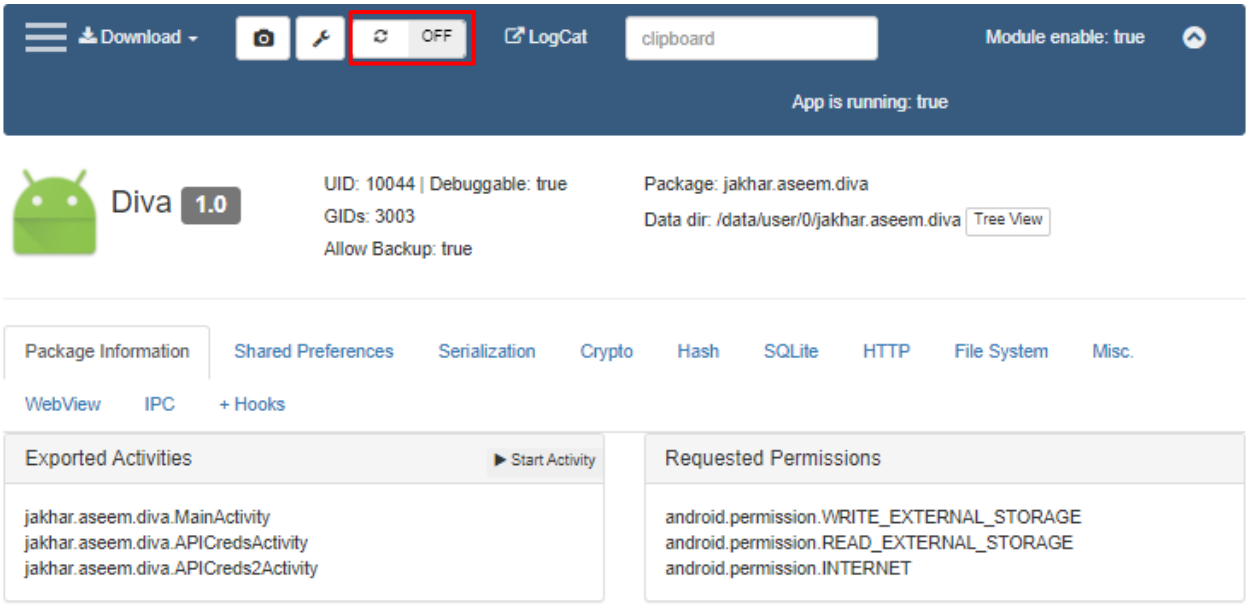
5.2 Xposed Framework – Inspeckage Module

Na vyskúšanie tohoto nástroja bude opäť použitá DIVA aplikácia, tentokrát však so zameraním na problém s ukladaním dát. Po otvorení Inspeckage je v tomto prípade zvolené odchytávanie aplikácie DIVA.



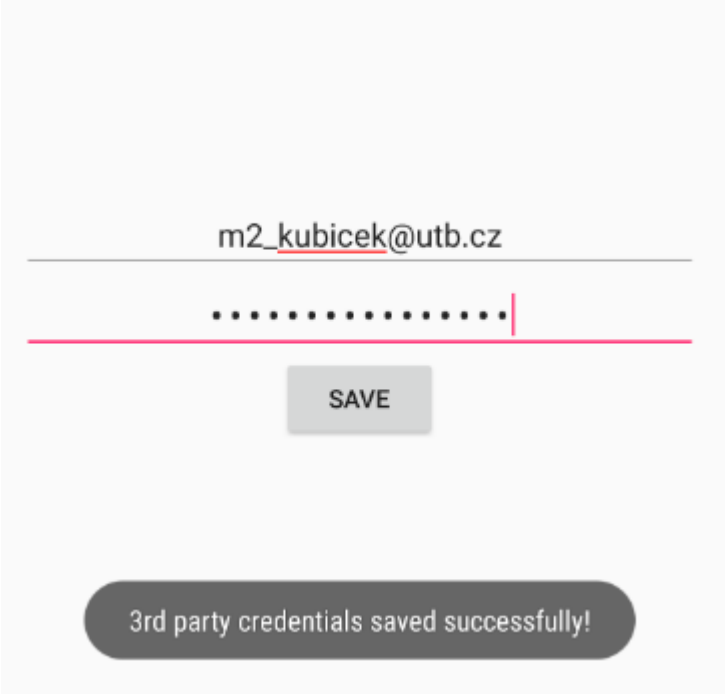
Obrázok 75. Detaily aplikácie v module Inspeckage [zdroj vlastný]

Následne je pomocou príkazu *adb forward tcp:8008 tcp:8008* spustený tento port, a potom je možné v prehliadači zadať adresu *127.0.0.1:8008*, ktorá zobrazí podrobnosti o aplikácii. V ďalšom kroku je stlačením vyznačeného tlačítka spustená automatická aktualizácia zobrazených dát v momente, keď aplikácia vykoná nejakú aktivitu.



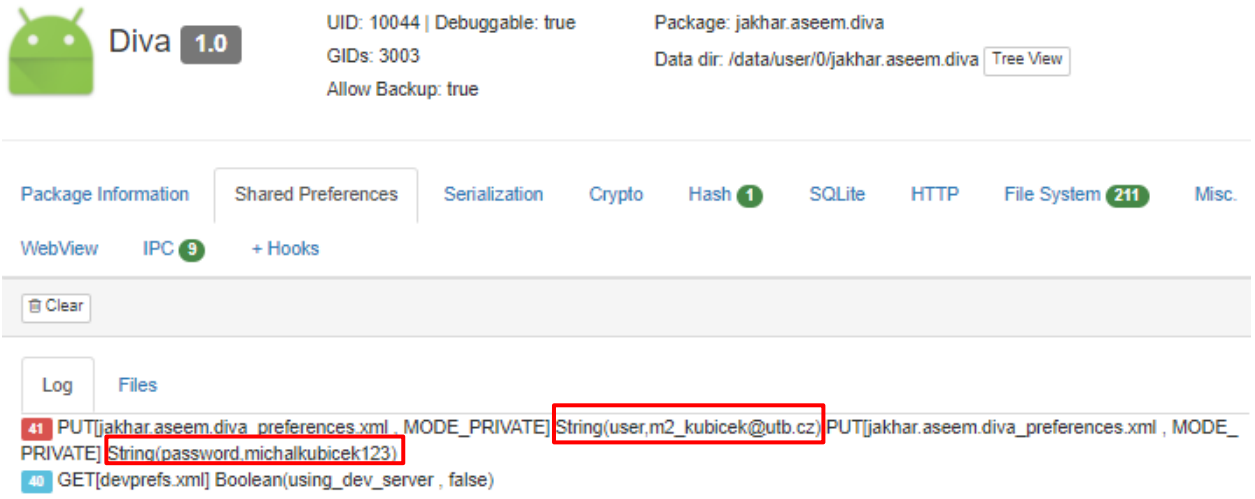
Obrázok 76. Prostredie Inspeckage s vloženou aplikáciou [zdroj vlastný]

Keď je spustená aplikácia, je hneď viditeľné, že nastali nejaké zmeny. Podľa zelených čísiel je badateľné v akom odvetví a koľko akcií sa vykonalo. V aplikácii na Android zariadení je spustená daná aktivita a zadané ľubovolné údaje.



Obrázok 77. Zadanie osobných údajov do aplikácie [zdroj vlastný]

Po zadaní údajov a kniknutí na *save* je vypísaná hláška, že tieto údaje boli úspešne uložené. V ďalšom kroku bude prejdené do rozhrania Inspeckage.

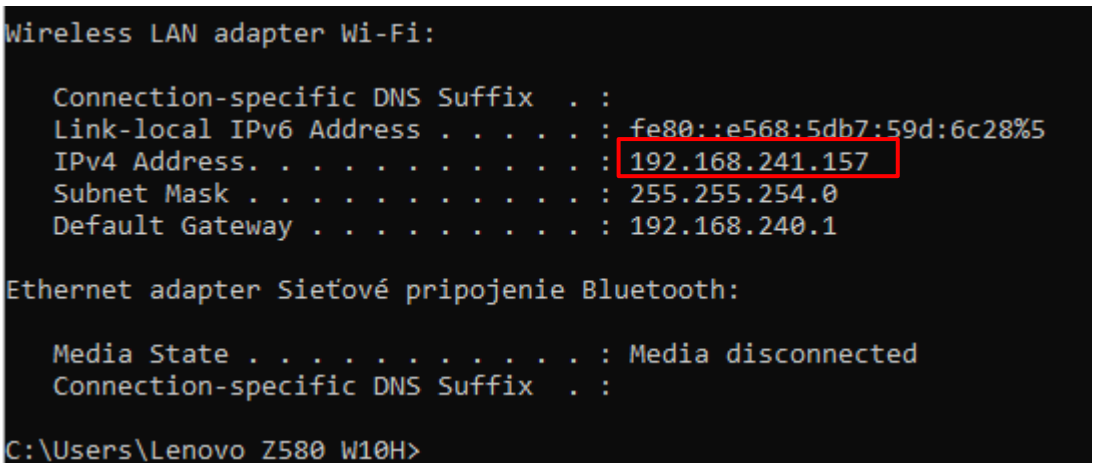


Obrázok 78. Zachytenie citlivých údajov pomocou Inspeckage [zdroj vlastný]

Po zobrazení je vidno, že v záložke *Shared Preferences* sa podarilo odchytiť nešifrované údaje, ktoré boli zadané v aplikácii.

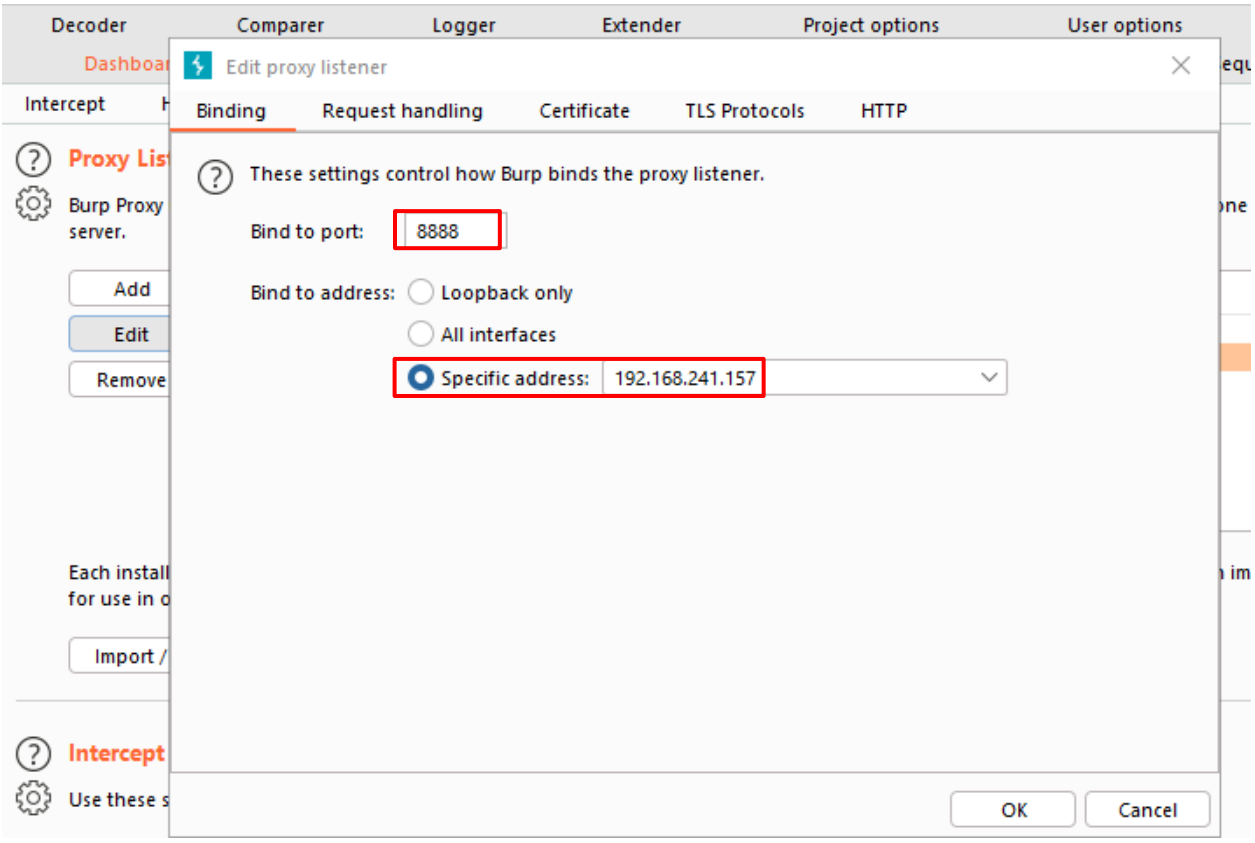
5.3 Burp Suite

V praktickej ukážke práce s týmto nástrojom bude hlavnou časťou Proxy, preto bude ukázané ako ju nastaviť. Pre nastavenie proxy je potrebná IP adresa a port. Využitá bude IP adresa hlavného zariadenia, teda počítača. Táto adresa môže byť zistená pomocou príkazu *ipconfig* v príkazovom riadku.



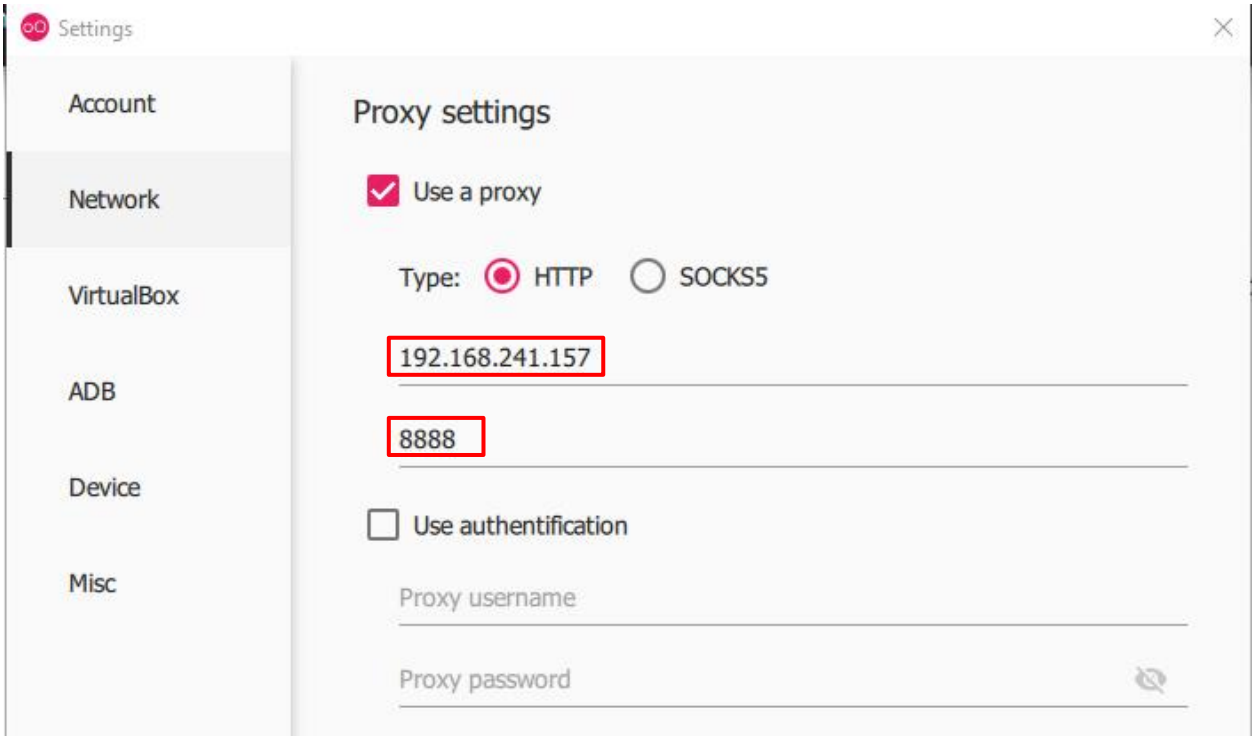
Obrázok 79. Zistenie aktuálnej IP adresy zariadenia [zdroj vlastný]

V tomto prípade je IP adresa počítača v lokálnej sieti *192.168.241.157*. Následne je táto adresa zadaná do nástroja Burp Suite pri vytváraní novej Proxy.



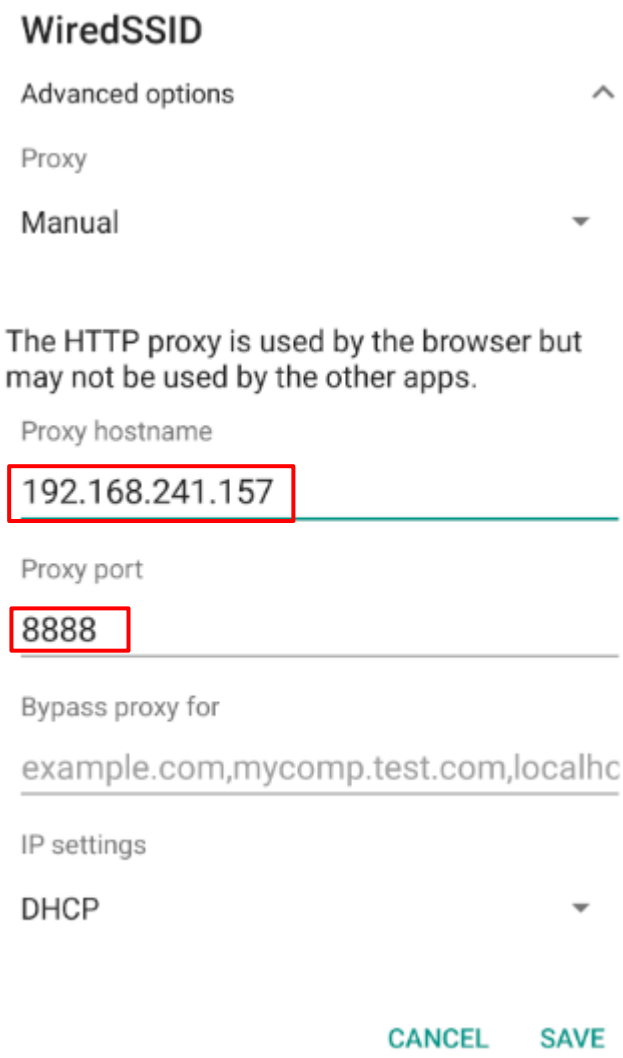
Obrázok 80. Zadanie IP do Proxy nástroja Burp Suite [zdroj vlastný]

Pre port môže byť teoreticky zvolené akékoľvek číslo z rozsahu 0 – 65535, štandardne sa však volí port 8080, pre prípad, že by tento port bol už obsadený, je zvolený port 8888, ktorý sa dobre pamätá. Následne, keď je proxy spustená, môže byť nastavená v emulátore. Cez nastavenia je zvolená záložka *Network*, kde sú zadané do údajov o proxy zvolené údaje.



Obrázok 81. Zadanie IP a nastavenie pripojenia na proxy v prostredí Genymotion [zdroj vlastný]

Týmto je proxy sprístupnená pre zariadenia v emulátore. Ostáva teda už len pripojiť samostatné zariadenie. To je vykonané v nastaveniach, keď je vybraná možnosť *modify network* na sieťovom pripojení.



Obrázok 82. Pripojenie virtuálneho smartfónu na Proxy [zdroj vlastný]

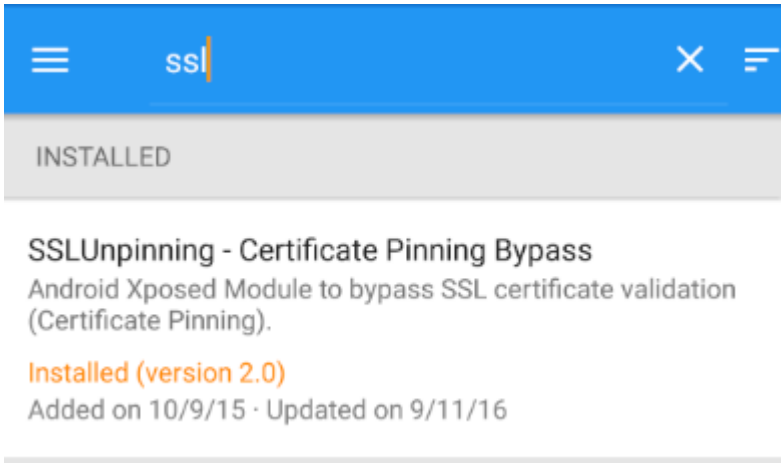
Týmto je zariadenie pripojené cez proxy a môžu byť tak odchytávané jeho žiadosti na webové servery.

Ako praktická ukážka práce s týmto nástrojom bude nástroj použitý na odchytenie údajov pri bypasse SSL Pinningu.

SSL, teda Secure Socket Layer vytvára základ dôveryhodného spojenia. Toto spojenie zaisťuje, že všetky prenášané údaje medzi webovým serverom a aplikáciou zostanú utajené resp. súkromné. SSL certifikáty majú pár kľúčov, jeden je verejný a druhý súkromný, tieto

kľúče sa využívajú na vytvorenie šifovaného spojenia. SSL Pinning, alebo pripnutie certifikátu SSL je proces priradenia hostoteľa k jeho certifikátu alebo verejnému kľúču. Keď je známy certifikát alebo verejný kľúč, je pripojený k hostiteľovi. Týmto sa teda eliminuje môžnosť MITM útoku, keďže všetky pokusy o pripojenie k iným certifikátom ako je ten pripnutý sú odmietnuté.[58]

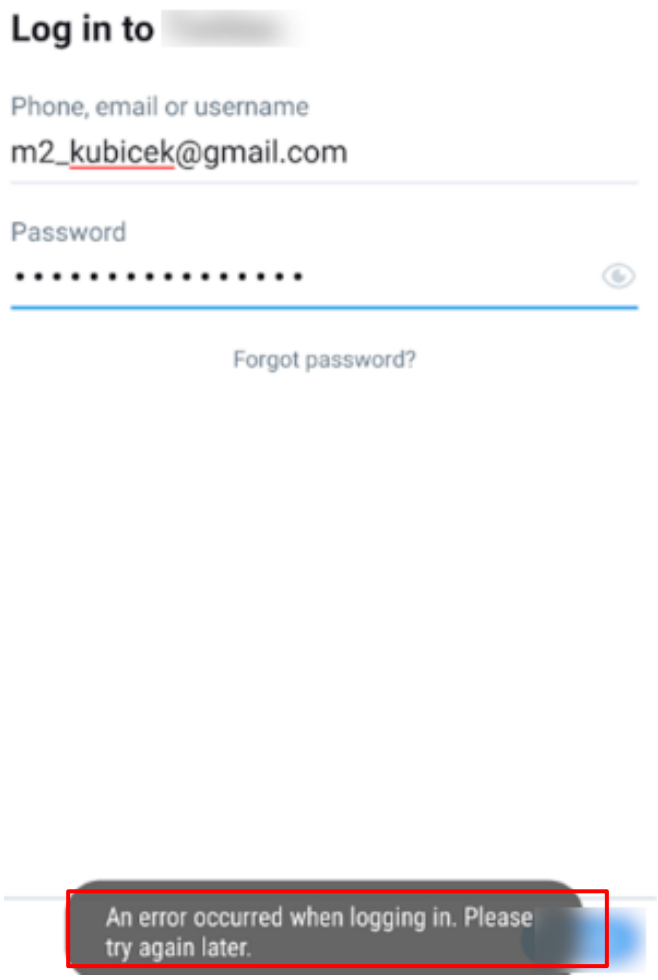
V skratke teda na to aby mohli byť odchytávané HTTP žiadosti aplikácie, ktorá má ochranu pomocou SSL certifikátov, musí byť táto ochrana obídená. Na toto bude využitý ďalší modul z Frameworku Xposed, ktorý umožňuje odpojiť túto aplikáciu od SSL.



Obrázok 83. Inštalácia SSLUnpinning modulu [zdroj vlastný]

Začaté bude teda stiahnutím daného modulu, do záložky *download* zadaním kľúčového slova „ssl“ a kliknutím na modul *SSLUnpinning – Certificate Pinning Bypass*. Po nainštalovaní sa zobrazí medzi stiahnutými aplikáciami alebo v aplikácii Xposed v záložke *Modules*.

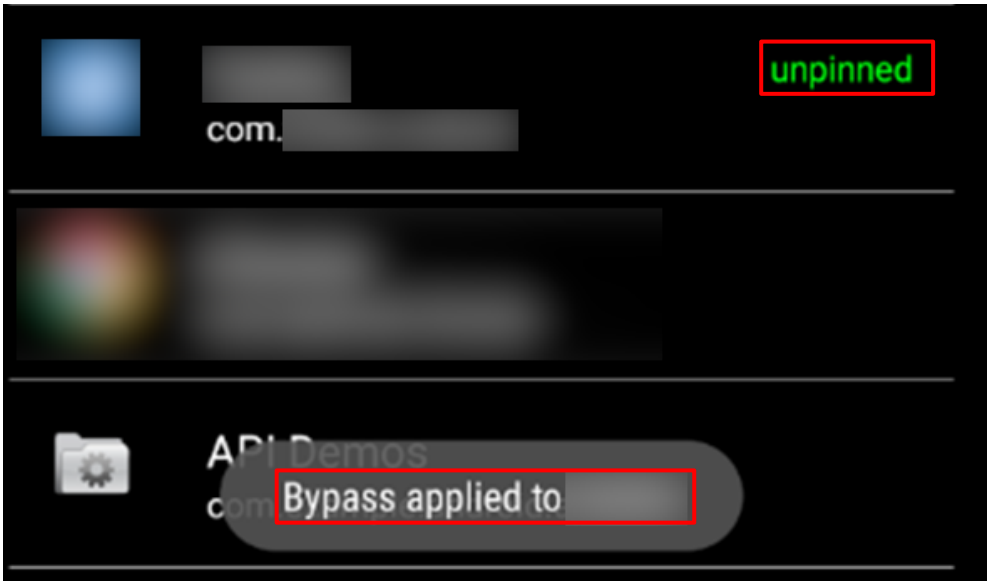
Pri prvom pokuse o prihlásenia bez zasahovania do aplikácie je viditeľné, čo sa stane ak je vykonané odchytávanie HTTP požiadaviek v aplikácii s aktívnym SSL Pinningom.



Obrázok 84. Chyba pri prihlasovaní do aplikácie [zdroj vlastný]

Po zadaní údajov a pokuse o prihlásenie je viditeľné, že aplikácia zobrazí chybovú hlášku. Certifikát sa nezhodoval s kľúčom a tak nebolo umožnené aplikácii odoslať požiadavku na prihlásenie. V BurpSuite sa taktiež nezobrazila žiadna požiadavka.

Ak je však použitý modul, ktorý bol v predošlej časti stiahnutý, tak umožní odpojiť aplikáciu od SSL a v dôsledku toho by malo byť možné odchytávať citlivé dáta.

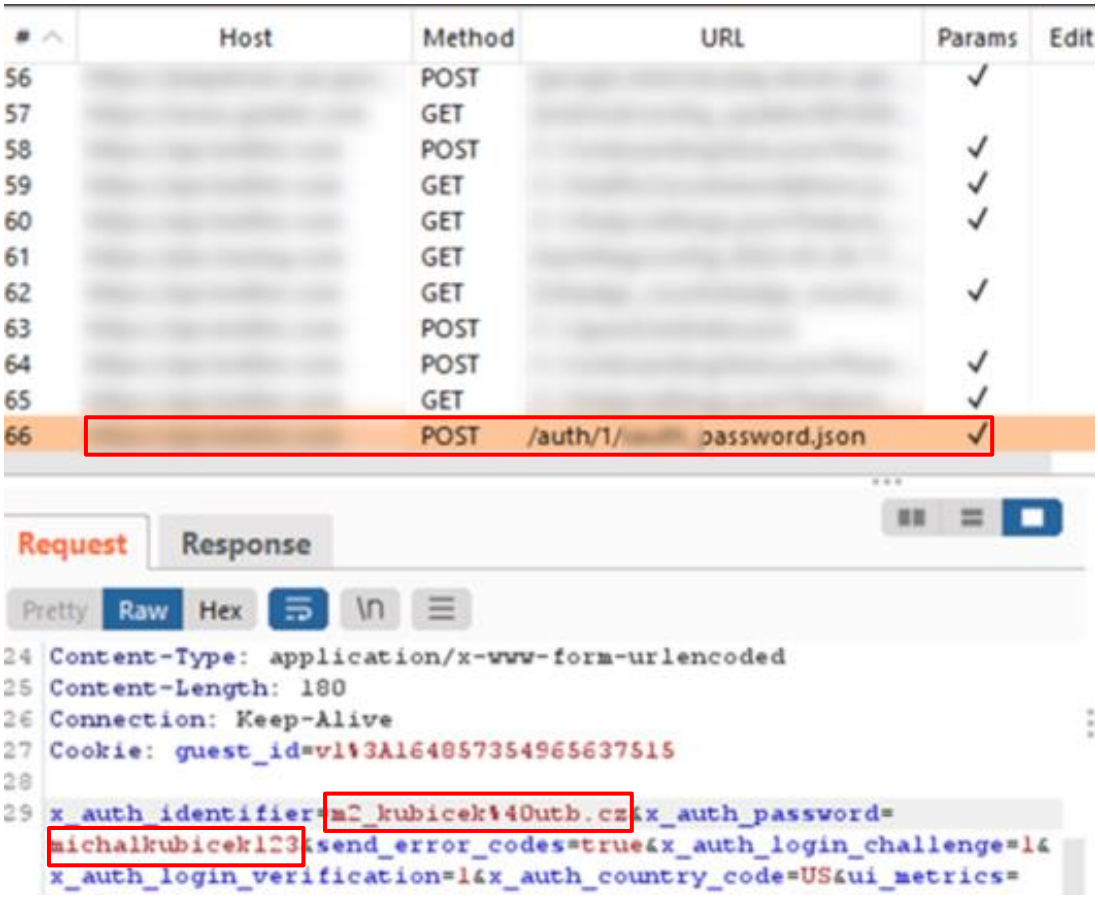


Obrázok 85. Aplikovanie bypassu na cieľovú aplikáciu [zdroj vlastný]

Po kliknutí na zvolenú aplikáciu sa objaví hláška o aplikovaní tzv. Bypassu a zelený nápis značiaci odopnutie. Po zadaní údajov požiadavka na prihlásenie prebehne úspešne a je viditeľné, že Burp Suite zachytil požiadavky tejto aplikácie.

Obrázok 86. Vloženie prihlasovacích údajov do aplikácie [zdroj vlastný]

Po rozkliknutí požiadavky typu *POST* je vidieť, že táto požiadavka obsahuje citlivé údaje ako sú zadané ID a heslo používateľa.



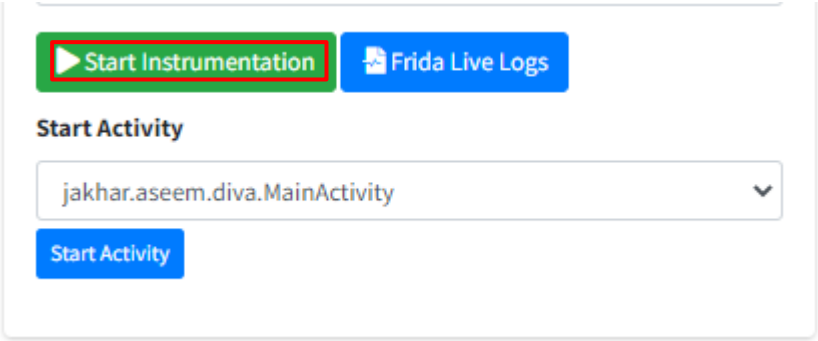
Obrázok 87. Odchytenie prihlasovacích údajov v požiadavke na server [zdroj vlastný]

Pomocou frameworku Xposed a modulu SSL Unpinning bolo vykonané úspešné odpojenie aplikácie od ochrany SSL pinningom a následne bola pomocou MITM útoku cez nástroj Burp Suite zachytená požiadavka s citlivými údajmi.

5.4 MobSF Dynamic Analyzer

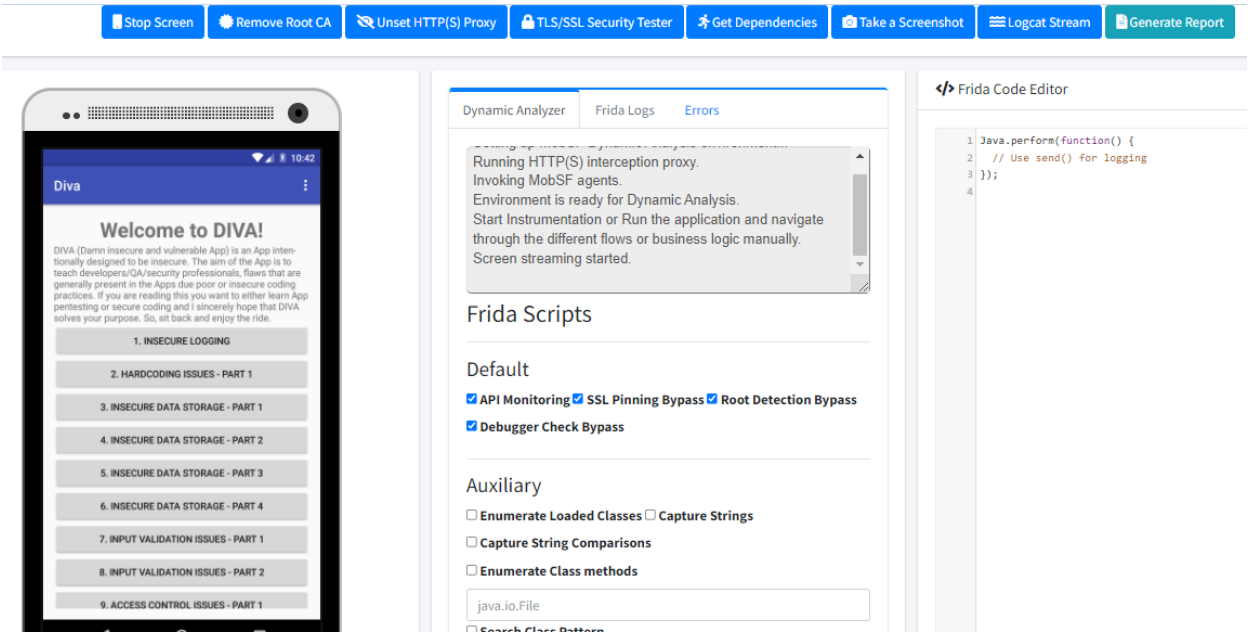
V tejto ukážke bude ukázané praktické použitie nástroja MobSF Dynamic Analyzer, prakticky bude priblížená zraniteľnosť zvaná *insufficient cryptography*, v preklade, nedostatočná kryptografia, ktorá je detailnejšie popísaná v kapitole Vulnerabilities.

Na ukážku bude použitá testovacia aplikáciu DIVA, v zozname aplikácií na zariadení, ktoré ukáže tento nástroj je vybraná spomínaná aplikácia. V ďalšom kroku sa zobrazí testovacia obrazovka.



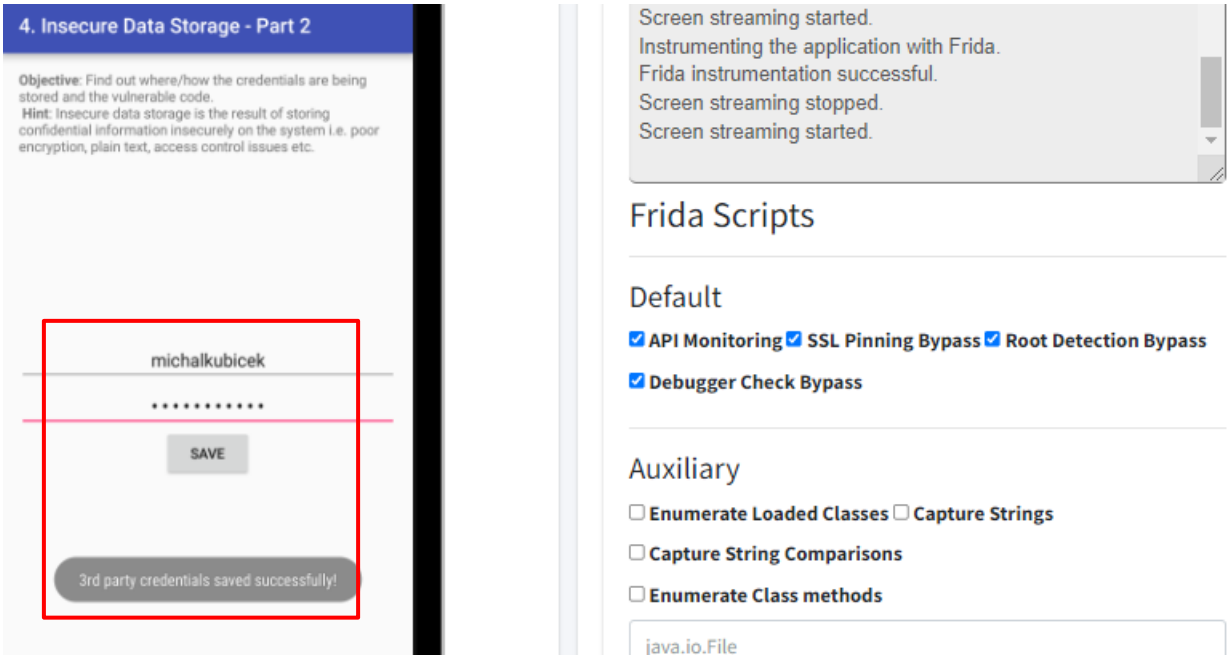
Obrázok 88. Spustenie dynamickej analýzy aplikácie v nástroji MobSF DA [zdroj vlastný]

Po kliknutí na tlačítko *Start Instrumentation* sa aplikácia spustí. Následne je nutné odkliknúť že bude požadované zobraziť obrazovku zariadenia v tomto nástroji, aby bolo možné ju v ňom ovládať. Tento krok nie je nevyhnutný, dá sa používať aj priamo obrazovka emulátora.



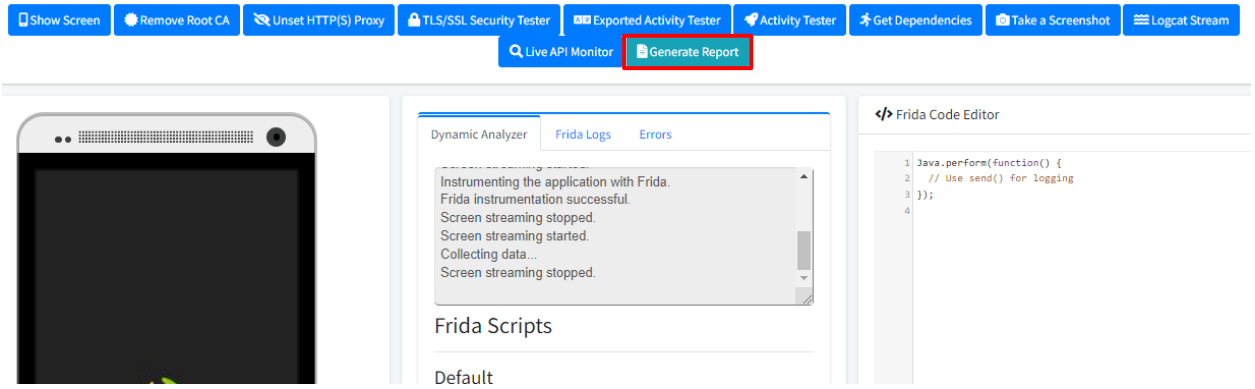
Obrázok 89. Prostredie MobSF Dynamic analyzer [zdroj vlastný]

Po vykonaní týchto úkonov je možné prechádzať cez aplikáciu a môžu v nej byť zadávané dáta do vstupov.



Obrázok 90. Vloženie osobných údajov do aplikácie [zdroj vlastný]

Po zadaní dát a kliknutí na *save* je zobrazená hláška o uložení dát. Potom je možné pokračovať v ďalších úkonoch, avšak pre túto ukážku to stačí. Keď sú zmeny vykonané, stačí kliknúť na tlačítko *Generate report*. Týmto sa začne vytvárať report o celom behu aplikácie.



Obrázok 91. Generovanie reportu [zdroj vlastný]

Následne je možné z reportu vybrať rôzne odvetvia v závislosti na záujme testera. Tento report je veľmi rozsiahly, takže obsahuje veľké množstvo informácií o správaní tejto aplikácie.

V podzložke *databases*, ku ktorej bolo prejdené pri prezeraní reportu, je nájdený zápis do databázy.

Obrázok 92. Zachytenie citlivých údajov v nezašifrovanej forme [zdroj vlastný]

V tomto zápise sú viditeľné úplne odhalené prihlasovacie údaje, ktoré nie sú nijakou formou zašifrované. Je teda možné ich veľmi ľahko zneužiť.

6 NÁVRH VHODNEJ METODIKY PENETRAČNÉHO TESTOVANIA APLIKÁCIÍ BEŽIACICH POD OPERAČNÝM SYSTÉMOM ANDROID

Zabezpečiť mobilné aplikácie je náročné. Aplikácie čelia veľkému množstvu hrozieb v dôsledku prostredia, v ktorom bežia. Penetračné testovanie je vhodným spôsobom ako tieto hrozby a zraniteľnosti odhaliť ešte pred tým, ako by ich mohol zneužiť útočník. Metodika penetračného testovania je postup, ktorý by mal byť dodržaný pri výkonávaní penetračného testovania. V tejto časti práce bude odkazované na postupy, ktoré boli detailnejšie rozobrané v praktických ukážkach.

6.1 Príprava

Hlavným zámerom tohoto bodu je získanie informácie o aplikácii. To môže zahŕňať vyhľadanie uniknutých dát alebo zdrojového kódu aplikácie z rôznych verejných zdrojov. To možu byť internetové stránky, vývojárske fóra a prípadne zdroje poskytnuté klientom.

Pochopenie platformy je v rámci prípravy kľúčovým prvkom penetračného testovania. Ak je pred testovaním pochopená funkčnosť a princípy fungovania platformy, je potom ľahšie pochopiť aj fungovanie samotnej testovanej aplikácie. Následne je potom jednoduchšie vytvoriť model hrozby pre aplikáciu. V tejto práci je platforma Android rozobraná v kapitole *1.2 Systém Android.*

Pochopenie typu aplikácie pri tomto bode ide hlavne o rozlíšenie, čí je daná aplikácia natívna, hybridná alebo webová. Toto zistenie potom umožní dodatočné správne vybranie nástrojov, ktoré budú použité. Napríklad ak je zrejmé, že aplikácia je natívna bez potreby prístupu na internet, nie je nutné používať nástroje určené na odchytávanie sieťovej komunikácie.

Príprava nástrojov a prostredia je posledným bodom prípravy. V tejto fáze už je známe aká aplikácia bude testovaná a sú známe aj informácie alebo jej dáta. Na základe týchto zistení je potom možné zvoliť nástroje, ktorými bude aplikácia testovaná. Penetračné testovanie by malo prebiehať v kotrolovanom prostredí, teda emulátore. V tejto práci je

postup a príprava emulátora rozobraná v kapitole *4.1 Príprava prostredia*. Následne môže prebehnúť testovanie aj na fyzickom zariadení.

6.2 Analýza

6.2.1 Statická analýza

Statická analýza sa vykonáva bez spustenia aplikácie na získanom, poskytnutom alebo dekompilovanom zdrojovom kóde a súboroch, ktoré k aplikácii patria. Prvým krokom statickej analýzy môže byť vykonanie tejto analýzy pomocou automatizovaného nástroja, akým je napríklad MobSF. Informácie, funkcie a proces inštalácie tohoto nástroja je rozobraný v kapitole *2.2 Mobile Security Framework (MobSF)*. Následná ukážka funkčnosti a práce s týmto nástrojom je popísaná v časti *4.2.3 MobSF*. Tento nástroj dokáže odhaliť prvky, ktoré môžu byť zneužité. Napríklad nebezpečné logovanie a podobne.

6.2.2 Analýza lokálnych súborov

Keď je aplikácia nainštalovaná, je jej pridelený vlastný adresár v súborovom systéme. Pri používaní tejto aplikácie bude zapisovať a čítať práve z tohoto adresára. Súbory, ku ktorým aplikácia pristupuje by mali byť tiež analyzované. Následne ak je to možné, je dobré sa pozrieť aj do privátneho dátového priestoru aplikácie, kde sa nachádza celý obsah vrátane zložiek ako *Shared_Prefs* alebo *databases*. Prístup ku tejto časti aplikácie je popísaný v kapitole *1.5.1 Aplikačný Sandbox*.

6.2.3 Analýza súboru AndroidManifest.xml

Súbor AndroidManifest.xml je súbor obsahujúci všetky informácie o balíku aplikácie, vrátane komponentov ako sú aktivity, služby, receivers alebo intents. Taktiež obsahuje informácie o povoleniach aplikácie. Toto všetko môžu byť veľmi dôležité informácie nakoľko je možné vyčítať z nich určité príznaky, ktoré môžu byť zneužité. Informácie a možné zneužitia týchto komponentov ako aj hľadanie *entry pointov,* je popísané v kapitole *1.3.4 AndroidManifest.xml*.

6.2.4 Reverzné inžinierstvo

Reverzné inžinierstvo spočíva v pokuse o dekompilovanie aplikácie do zdrojového kódu, ktorý je možné čítať a orientovať sa v ňom. Ak je táto dekompilácia dosiahnutá a je získaný prístup ku zdrojovému kódu aplikácie, mala by byť vykonaná manuálna kontrola kódu s cieľom pochopiť internú funkčnosť aplikácie a nájsť ďalšie zraniteľnosti, ktoré automatická statická analýza nenašla. To môžu byť rôzne chyby aplikačnej logiky prípadne heslá zabudnuté v kóde. Príklad reverzného inžinierstva a dekompilácie aplikácie je ukázaný v kapitole *4.2.2 JADX,* kde je pomocou nástroja JADX dekompilovaná aplikácia a v zdrojovom kóde nájdené zanechané heslo.

6.2.5 Dynamická analýza

Dynamická analýza je vykonávaná počas spustenia aplikácie a jej cieľom je získať prehľad o správaní aplikácie za behu. To zahŕňa napríklad medziprocesovú komunikáciu aplikácie a pod. Vhodným prvým krokom dynamickej analýzy môže byť napr. Xposed Framework. Tento nástroj dokáže zaznamenávať zmeny v aplikácii v reálnom čase a následne umožňuje prehľad vo vykonaných zmenách a presune dát v rámci aplikácie. Tento nástroj ako aj jeho funkcie, inštalácia a aktivácia sú popísané v časti *3.2 Xposed Framework*. Následne je praktická ukážka práce s týmto nástrojom a výsledky dynamickej analýzy pomocou tohoto nástroja ukázané v kapitole *5.2 Xposed Framework – Inspeckage Module.*

Ďalším prvkom dynamickej analýzy môže byť pokus o obídenie logiky aplikácie pomocou injectovania scriptu. Ak je pri manuálnej kontrole kódu objavená časť kódu, ktorá zodpovedá za určitú časť funkčnosti, ktorá môže byť obídená, môže byť použitý nástroj Frida, ktorý práve takéto injectovanie kódu umožňuje. Tento nástroj je popísaný v kapitole *3.1. Frida* a praktická ukážka práce s ním, kde je práve ukázaná aj možnosť injectovania kódu do aplikácie za behu. Je v kapitole *5.1.*

6.2.6 Analýza sieťovej prevádzky

Analýza sieťovej prevádzky je vykonávaná nakonfigurovaním zariadenia tak, aby smerovalo propojenie k serveru pomocou proxy, ktorá je kotrolovaná testerom. Toto umožňuje zachytávať požiadavky a upravovať tak túto prevádzku. Prípadne je možné jednotlivé žiadosti na server preskúmať a zistiť, či neobsahujú nejaké citlivé dáta. V tejto

práci je na odchytávanie sieťovej komunikácie použitý nástroj Burp Suite. Popis, inštalácia a nastavenie proxy potrebnej na tento druh analýzy je popísaná v teoretickej časti tejto práce, konkrétne v kapitole *3.3 Burp Suite.* Praktická ukážka práce s týmto nástrojom je spojená s odopnutím SSL certifikátu, ktorý zabraňoval posielaniu dát v nedôveryhodnom pripojení cez proxy. Celá ukážka aj s popisom je v kapitole *5.3. Burp Suite.*

6.3 Zneužitie

V tejto časti penetračného testovania je vykonávané zneužívanie nájdených zraniteľností. Toto môže predstavovať snahu o získanie citlivých informácií prípadne vykonanie nejakej škodlivej činnosti. V práci je praktická časť koncipovaná tak, že v ukážkach je po nájdení nejakej zraniteľnosti v zápätí vykonaný aj pokus o zneužitie. Buď sú teda ukázané získané citlivé dáta, prípadne obídená funkčnosť aplikácie a podobne.

Ďalšou časťou, ktorá sem môže byť zaradená je tzv. eskalácia privilégií. To znamená, že sú vykonané kroky k tomu, aby boli získané vyššie privilégiá a nadobudnutý prístup ako super user, teda root.

6.4 Správa

Koncovým výstupom penetračného testovania by mala byť správa, ktorá uvádza zistené zraniteľnosti a slabé miesta. Zraniteľnosti by mali byť ohodnotené z hľadiska rizika pre používateľa alebo spoločnosť. V ďalšej časti práce bude teda zostavený zoznam zraniteľností nájdených v aplikácii DIVA, keďže je táto aplikácia používaná pri najväčšom počte praktických ukážok.

Zraniteľnosť 1: Hard-coded credentials

Vplyv zraniteľnosti: vysoký

Pravdepodobnosť výskytu: malá

Hodnotenie rizika: 6/10

Popis: Pri manuálnom prechádzaní kódu bolo zistené, že metóda vykonávajúca overenie prístupu do chránenej časti vykonáva toto overenie pomocou formy

porovnávania reťazcov, takže odhaluje celé heslo, ktoré umožňuje prístup do chránenej časti.

Náprava: Namiesto pevného kódu pre overenie hesla by mal byť vytvorený režim prvého prihlásenia, ktorý bude vyžadovať od používateľa zadanie jedinečného hesla.

Zraniteľnosť 2: Insecure logging (nebezpečné logovanie)

Vplyv zraniteľnosti: stredný

Pravdepodobnosť výskytu: malá

Hodnotenie rizika: 4/10

Popis: Automatickou statickou analýzou nástrojom MobSF bolo odhalené nebezpečné logovanie citlivých informácií. Po prístupe k logu je tak možné zachytiť tieto informácie.

Náprava: Logovacie riešenie by malo byť vytvorené tak, aby bolo bezpečné. Návrh bezpečného logovania by mal teda pozostávať z následovných prvkov:

- Zakódovať a overiť všetky nebezpečné znaky
- Nelogovať citlivé informácie (ID, heslá, údaje o kartách,...)
- Chrániť integritu logu a zvážiť povolenia log súborov.

Zraniteľnosť 3: Nesprávne ukladanie dát

Vplyv zraniteľnosti: veľký

Pravdepodobnosť výskytu: stredná

Hodnotenie rizika: 7/10

Popis: Pomocou dynamickej analýzy nástrojom Xposed Framework, konkrétne modulu Inspeckage. Bolo odhalené ukladanie prihlasovacích údajov v zložke

Shared_Prefs, do ktorej je možné sa pomocou rootovaného zariadenia dostať a údaje odcudziť.

Náprava: Aby sa útočník mohol dostať ku zložke *Shared_Prefs,* musí mať prístup root. Preto je prvým krokom ako zabrániť tejto zraniteľnosti implementácia detekcie rootu. Toto však nemusí byť úplná ochrana, nakoľko detekcia rootu sa dá obísť, ako je ukázané aj v jednej z ukážok. Ďalšou možnosťou je ukladanie dát namiesto Shared Preferences do Jetpack DataStore. Toto je riešenie, ktoré umožňuje ukladať dáta asynchrónne, konzistentne a transakčne.

Zraniteľnosť 4: Nedostatočná kryptografia

Vplyv zraniteľnosti: veľký

Pravdepodobnosť výskytu: stredná

Hodnotenie rizika: 7/10

Popis: Dynamickou analýzou pomocou MobSF Dynamic Analyzer, bolo odhalené zapisovanie prihlasovacích údajov do databázy. Tieto prihlasovacie údaje neboli nijakým spôsobom zašifrované a preto mohli byť veľmi ľahko ukradnuté.

Náprava: Ak je potrebné prenášať alebo ukladať citlivé údaje, je potrebné použiť šifrovanie týchto údajov tak silné, aby použitý algoritmus zodpovedal aktuálnym kryptografickým štandardom. Po zašifrovaní týchto údajov je potrebné starostlivo chrániť kryptografické kľúče, aby nedošlo ku krádeži, pretože potom je sila daného algoritmu irelevantná.

ZÁVER

Cieľom tejto práce bolo priblížiť a poukázať na penetračné testovania mobilných aplikácií bežiacich pod systémom Android. V teoretickej práci bol popísaný systém Android, jeho súčasti ako aj bezpečnostné funkcie a technológie, ktoré využíva. Taktiež bol popísaný súbor APK a jeho súčasti, nakoľko sa jedná o súbor, v ktorom sú aplikácie distribuované. V teoretickej časti boli zároveň priblížené a popísané niektoré zraniteľnosti vyskytujúce sa v mobilných aplikáciách.

Ďalším bodom teoretickej časti bola analýza a popis nástrojov, ktoré sa využívajú pre statickú analýzu ako aj dynamickú analýzu. Popis týchto nástrojov zahŕňal ich funkcie, možnosti ako s nimi pracovať a taktiež aj návod ako ich nainštalovať a správne nastaviť, aby bolo možné s nimi pracovať. Taktiež bolo priblížené aj nastavenie virtuálneho zariadenia spusteného v kontrolovanom prostredí, teda emulátore. Následne bolo vytvorené virtuálne zariadenie a nainštalovaný nástroj ADB, ktorý slúži na komunikáciu s virtuálnym zariadením.

V praktickej časti boli jednotlivé nástroje využité na praktické ukážky práce s nimi. Každý z týchto nástrojov bol použitý na demonštráciu nejakej zraniteľnosti, ktorá môže byť v aplikáciách objavená. Následne boli ukázané aj možné útoky a vysvetlené postupy ako sa dajú takéto zraniteľnosti zneužiť. V práci boli ukázané zraniteľnosti ako napríklad nebezpečné ukladanie dát vo forme hard-coded credentials, nebezpečné logovanie, ktoré bolo odhalené pomocou nástroja MobSF. V tomto prípade boli z logu odchytené citlivé dáta zadané na vstupe aplikácie. Ďalej bolo predvedené obídenie root detekcie pomocou injectovania scriptu do bežiacej aplikácie, alebo odchytávanie HTTP požiadaviek doplnené o bypass SSL pinningu, ktorý umožňil citlivé dáta odchytávať. Taktiež bola v tejto práci vytvorená ukážka modifikácie kódu v jazyku Smali, spojená so spätnou kompiláciou, generovaním certifikátu a podpísaním upravenej aplikácie. Následne bola modifikovaná aplikácia aj úspešne nainštalovaná na zariadenie a vyskúšaná.

V poslednej časti práce bol vytvorený postup, podľa ktorého by malo byť vykonávané penetračné testovanie a zostavený zoznam pravidiel metodiky, ktoré odkazovaním na praktické príklady z tejto práce vysvetľujú dané postupy. Nakoniec bol vytvorený report o zraniteľnostiach nájdených v aplikácii DIVA.

ZOZNAM POUŽITEJ LITERATÚRY

[1] Mobile Operating System Market Share Worldwide | Statcounter Global Stats. *Statcounter Global Stats - Browser, OS, Search Engine including Mobile Usage Share* [online]. Copyright © StatCounter 1999 [cit. 03.04.2022]. Dostupné z: https://gs.statcounter.com/os-market-share/mobile/worldwide

[2] HAQ, Ikram Ul a Tamim Ahmed KHAN. *Penetration Frameworks and Development Issues in Secure Mobile Application Development: A Systematic Literature Review.* IEEE Access [online]. 2021, 9, 87806-87825 [cit. 2022-05-14]. ISSN 2169-3536. Dostupné z: doi:10.1109/ACCESS.2021.3088229

[3] OZGUR, Berkecan, Ibrahim Alper DOGRU, Goksel UCTU a Mustafa ALKAN. *A Suggested Model for Mobile Application Penetration Test Framework.* In: 2021 International Conference on Information Security and Cryptology (ISCTURKEY) [online]. IEEE, 2021, 2021-12-2, s. 18-21 [cit. 2022-05-14]. ISBN 978-1-6654-0776-2. Dostupné z: doi:10.1109/ISCTURKEY53027.2021.9654417

[4] ALANDA, Aide, Deni SATRIA, H.A MOODUTO a Bobby KURNIAWAN. *Mobile Application Security Penetration Testing Based on OWASP.* IOP Conference Series: Materials Science and Engineering [online]. 2020, 846(1) [cit. 2022-05-14]. ISSN 1757-8981. Dostupné z: doi:10.1088/1757-899X/846/1/012036

[5] ZHANG, Lei, Binbin WANG, Quanjiang SHEN, Yan SONG, Naiwang GUO a Liangjun XIE. *A MITM Based Penetration Test Efficiency Improvement Approach for Traffic-Encrypted Mobile Applications of Power Industry.* In: 2021 IEEE 6th International Conference on Computer and Communication Systems (ICCCS) [online]. IEEE, 2021, 2021-4-23, s. 743-747 [cit. 2022-05-14]. ISBN 978-1-6654-1256-8. Dostupné z: doi:10.1109/ICCCS52626.2021.9449241

[6] What is Android? Everything you need to know about Google's OS. *Android Authority: Tech Reviews, News, Buyer's Guides, Deals, How-To* [online]. Copyright © 2022 Authority Media. All rights reserved. [cit. 03.04.2022]. Dostupné z: https://www.androidauthority.com/what-is-android-328076/

[7] Android history: The evolution of the biggest mobile OS in the world. *Android Authority: Tech Reviews, News, Buyer's Guides, Deals, How-To* [online]. Copyright © 2022 Authority Media. All rights reserved. [cit.

03.04.2022]. Dostupné z: https://www.androidauthority.com/history-android-os-name-789433/

[8] Android 4.4 APIs | Android Developers. *Android Developers* [online]. [cit. 03.04.2022]. Dostupné z: https://developer.android.com/about/versions/kitkat/android-4.4

[9] Android Lollipop | Android Developers. *Android Developers* [online]. [cit. 03.04.2022]. Dostupné z: https://developer.android.com/about/versions/lollipop

[10] Android 6.0 Changes | Android Developers. *Android Developers* [online]. [cit. 03.04.2022]. Dostupné z: https://developer.android.com/about/versions/marshmallow/android-6.0-changes

[11] What are System Apps? - Hexnode Help Center. *Unified Device Management | Hexnode UEM* [online]. Copyright © 2022 Mitsogo Inc. All Rights Reserved. [cit. 03.04.2022]. Dostupné z: https://www.hexnode.com/mobile-device-management/help/what-are-system-apps/

[12] Platform Architecture | Android Developers. *Android Developers* [online]. [cit. 15.04.2022]. Dostupné z: https://developer.android.com/guide/platform

[13] Platform Architecture | Android Developers. *Android Developers* [online]. [cit. 15.04.2022]. Dostupné z: https://developer.android.com/guide/platform#linux-kernel

[14] Platform Architecture | Android Developers. *Android Developers* [online]. [cit. 15.04.2022]. Dostupné z: https://developer.android.com/guide/platform#hal

[15] Platform Architecture | Android Developers. *Android Developers* [online]. [cit. 15.04.2022]. Dostupné z: https://developer.android.com/guide/platform#art

[16] Platform Architecture | Android Developers. *Android Developers* [online]. [cit. 15.04.2022]. Dostupné z: https://developer.android.com/guide/platform#native-libs

[17] Platform Architecture | Android Developers. *Android Developers* [online]. [cit. 15.04.2022]. Dostupné z: https://developer.android.com/guide/platform#api-framework

[18] What Is an APK File and What Does It Do? Explained *Technology, Simplified* [online]. [cit. 15.04.2022]. Dostupné z: https://www.makeuseof.com/tag/what-is-apk-file/

[19] ANDROID PENTESTING SERIES PART 4 : APK File Structure -. - *AppSec, Network Security, Penetration Testing, Bug Bounties, Coding and more..* [online]. [cit. 15.04.2022]. Dostupné z: https://sec-art.net/2022/01/21/android-pentesting-series-part-2-apk-file-structure/

[20] App Manifest Overview | Android Developers. *Android Developers* [online]. [cit. 15.04.2022]. Dostupné z: https://developer.android.com/guide/topics/manifest/manifest-intro

[21] Intent | Android Developers. *Android Developers* [online]. [cit. 15.04.2022]. Dostupné z: https://developer.android.com/reference/android/content/Intent.html

[22] Security Guidelines for Android Manifest Files – Penetration Testing Lab. *Penetration Testing Lab – Offensive Techniques & Methodologies* [online]. [cit. 15.04.2022]. Dostupné z: https://pentestlab.blog/2017/01/24/security-guidelines-for-android-manifest-files/

[23] <receiver> | Android Developers. *Android Developers* [online]. [cit. 23.04.2022]. Dostupné z: https://developer.android.com/guide/topics/manifest/receiver-element.html?hl=en

[24] How to Resolve the SecurityException in Java | Rollbar. *Rollbar - Error Tracking Software for Continuous Code Improvement* [online]. Copyright © 2012 [cit. 23.04.2022]. Dostupné z: https://rollbar.com/blog/java-securityexception/

[25] Smali: Assembler for Android's VM | Medium. *The Mobile Security Guys – Medium* [online]. [cit. 23.04.2022]. Dostupné z: https://mobsecguys.medium.com/smali-assembler-for-dalvik-e37c8eed22f9

[26] ANDROID PENTESTING SERIES PART 4 : APK File Structure -. - *AppSec, Network Security, Penetration Testing, Bug Bounties, Coding and more..* [online]. [cit. 23.04.2022]. Dostupné z: https://sec-art.net/2022/01/21/android-pentesting-series-part-2-apk-file-structure/

[27] Application Sandbox | Android Open Source Project. *Android Open Source Project* [online]. [cit. 23.04.2022]. Dostupné z: https://source.android.com/security/app-sandbox

[28] <manifest> | Android Developers. *Android Developers* [online]. [cit. 23.04.2022]. Dostupné z: https://developer.android.com/guide/topics/manifest/manifest-element#uid

[29] HiQES Android Security Part 1: App Basics You Are Missing. *HiQES Provides application development and custom driver development* [online]. [cit. 23.04.2022]. Dostupné z: http://hiqes.com/android-security-part-1/

[30] ls Command in Linux for Listing Files. *Linoxide - Linux Tutorials & Tips* [online]. Copyright © 2022 BTreme. All rights reserved [cit. 23.04.2022]. Dostupné z: https://linoxide.com/20-ls-command-linux/

[31] Application Signing | Android Open Source Project. *Android Open Source Project* [online]. [cit. 23.04.2022]. Dostupné z: https://source.android.com/security/apksigning

[32] Authentication | Android Open Source Project. *Android Open Source Project* [online]. [cit. 23.04.2022]. Dostupné z: https://source.android.com/security/authentication

[33] Biometrics | Android Open Source Project. *Android Open Source Project* [online]. [cit. 23.04.2022]. Dostupné z: https://source.android.com/security/biometric

[34] Encryption | Android Open Source Project. *Android Open Source Project* [online]. [cit. 03.05.2022]. Dostupné z: https://source.android.com/security/encryption

[35] Security-Enhanced Linux in Android | Android Open Source Project. *Android Open Source Project* [online]. [cit. 23.04.2022]. Dostupné z: https://source.android.com/security/selinux

[36] Trusty TEE | Android Open Source Project. *Android Open Source Project* [online]. [cit. 29.04.2022]. Dostupné z: https://source.android.com/security/trusty

[37] Verified Boot | Android Open Source Project. *Android Open Source Project* [online]. [cit. 29.04.2022]. Dostupné z: https://source.android.com/security/verifiedboot

[38] 5 Steps Approach to Penetration Testing Methodology. *Astra Security - Comprehensive Suite Making Security Simple* [online]. Copyright © 2022 [cit. 29.04.2022]. Dostupné z: https://www.getastra.com/blog/security-audit/penetration-testing-methodology/

[39] Penetration Testing: What, Why and How [A Complete Guide]. *Astra Security - Comprehensive Suite Making Security Simple* [online]. Copyright © 2022 [cit. 29.04.2022]. Dostupné z: https://www.getastra.com/blog/security-audit/penetration-testing/

[40] Mobile App Security Testing - OWASP Mobile Security Testing Guide. [online]. [cit. 29.04.2022]. Dostupné z: https://mobile-security.gitbook.io/mobile-security-testing-guide/overview/0x04b-mobile-app-security-testing

[41] OWASP Mobile Top 10 | OWASP Foundation. *OWASP Foundation, the Open Source Foundation for Application Security* | *OWASP Foundation* [online]. [cit. 29.04.2022]. Dostupné z: https://owasp.org/www-project-mobile-top-10/#

[42] M1: Improper Platform Usage | OWASP Foundation. *OWASP Foundation, the Open Source Foundation for Application Security* | *OWASP Foundation* [online]. [cit. 29.04.2022]. Dostupné z: https://owasp.org/www-project-mobile-top-10/2016-risks/m1-improper-platform-usage

[43] M2: Insecure Data Storage | OWASP Foundation. *OWASP Foundation, the Open Source Foundation for Application Security* | *OWASP Foundation* [online]. [cit. 29.04.2022]. Dostupné z: https://owasp.org/www-project-mobile-top-10/2016-risks/m2-insecure-data-storage

[44] M3: Insecure Communication | OWASP Foundation. *OWASP Foundation, the Open Source Foundation for Application Security* | *OWASP Foundation* [online]. [cit. 29.04.2022]. Dostupné z: https://owasp.org/www-project-mobile-top-10/2016-risks/m3-insecure-communication

[45] M4: Insecure Authentication | OWASP Foundation. *OWASP Foundation, the Open Source Foundation for Application Security* | *OWASP Foundation* [online]. [cit. 29.04.2022]. Dostupné z: https://owasp.org/www-project-mobile-top-10/2016-risks/m4-insecure-authentication

[46] M5: Insufficient Cryptography | OWASP Foundation. *OWASP Foundation, the Open Source Foundation for Application Security* | *OWASP Foundation* [online]. [cit. 29.04.2022]. Dostupné z: https://owasp.org/www-project-mobile-top-10/2016-risks/m5-insufficient-cryptography

[47] GitHub - iBotPeaches/Apktool: A tool for reverse engineering Android apk files. *GitHub: Where the world builds software · GitHub* [online]. Copyright © 2022 GitHub, Inc. [cit. 29.04.2022]. Dostupné z: https://github.com/iBotPeaches/Apktool

[48] GitHub - MobSF/Mobile-Security-Framework-MobSF: Mobile Security Framework (MobSF) is an automated, all-in-one mobile application (Android/iOS/Windows) pen-testing, malware analysis and security assessment framework capable of performing static and dynamic analysis.. *GitHub: Where the world builds software · GitHub* [online]. Copyright © 2022 GitHub, Inc. [cit. 29.04.2022]. Dostupné z: https://github.com/MobSF/Mobile-Security-Framework-MobSF

[49] GitHub - skylot/jadx: Dex to Java decompiler. *GitHub: Where the world builds software · GitHub* [online]. Copyright © 2022 GitHub, Inc. [cit. 03.05.2022]. Dostupné z: https://github.com/skylot/jadx

[50] GitHub - pxb1988/dex2jar: Tools to work with android .dex and java .class files. *GitHub: Where the world builds software · GitHub* [online]. Copyright © 2022 GitHub, Inc. [cit. 03.05.2022]. Dostupné z: https://github.com/pxb1988/dex2jar

[51] GitHub - java-decompiler/jd-gui: A standalone Java Decompiler GUI. *GitHub: Where the world builds software · GitHub* [online]. Copyright © 2022 GitHub, Inc. [cit. 03.05.2022]. Dostupné z: https://github.com/java-decompiler/jd-gui

[52] Welcome | Frida • A world-class dynamic instrumentation framework. *Frida • A world-class dynamic instrumentation framework | Inject JavaScript to explore native apps on Windows, macOS, GNU/Linux, iOS, Android, and QNX* [online]. [cit. 03.05.2022]. Dostupné z: https://frida.re/docs/home/

[53] Welcome to the Xposed Module Repository! | Xposed Module Repository. *Welcome to the Xposed Module Repository! | Xposed Module Repository* [online]. [cit. 03.05.2022]. Dostupné z: https://repo.xposed.info/

[54] Burp Suite - Application Security Testing Software - PortSwigger. *Web Application Security, Testing, & Scanning - PortSwigger* [online]. Copyright © 2022 PortSwigger Ltd. [cit. 03.05.2022]. Dostupné z: https://portswigger.net/burp

[55] Network security configuration | Android Developers. *Android Developers* [online]. [cit. 03.05.2022]. Dostupné z: https://developer.android.com/training/articles/security-config

[56] What Is SSL (Secure Sockets Layer)? | What is an SSL Certificate? | DigiCert. *SSL Digital Certificate Authority - Encryption & Authentication* [online]. Copyright © 2022 DigiCert, Inc. All rights reserved. [cit. 03.05.2022]. Dostupné z: https://www.digicert.com/what-is-an-ssl-certificate

ZOZNAM POUŽITÝCH SYMBOLOV A SKRATIEK

ADB	Android Debug Bridge – nástroj umožňujúci komunikovať so zariadením
API	Application Programming Interface – Rozhranie pre programovanie aplikácií
APK	Android Package – typ súboru pre Androidové aplikácie
ARM	Advanced RISC Machine – architektúra CPU
ART	Android RunTime – aplikačné runtime prostredie používané systémom Android
CA	Certificate Authority – je subjekt, ktorý vydáva digitálne certifikáty
CBC	Cipher Block Chaining – algoritmus používajúci blokovú šifru
CPU	Central Processing Unit – Centrálna riadiaca jednotka
DAC	Discretionary Access Controll – diskrečná kontrola vstupu (pri Sandboxovaní)
DAST	Dynamic Application Security Testing – dynamické testovanie bezp. aplikácie
DIVA	Damn Insecure and Vulnerable App – aplikácia, v ktorej sú zámerné chyby
GPS	Global Positioning System – systém umožňujúci získať polohu zariadenia
GUI	Graphic User Interface – Grafické rozhranie pre používateľa
HAL	Hardware Application Layer – definuje štandardné rozhrania pre hardvér
HMAC	Hash-based message authentication code – kód na auntentifikáciu správ
HTTP	Hyper Text Transfer Protocol – protokol používaný vo World Wide Webe
iOS	iPhone Operating System – operačný systém pre zariadenia iPhone
IP	Internet Protocol (address) – séria čísiel identifikujúca zariadenia v sieti
JAR	Java Archive – typ súboru obsahujúci zdrojový kód v jazyku Java
JVM	Java Virtual Machine – virtuálny stroj ktorý umožňuje spúštanie Java súborov
MAC	Mandatory Access Controll – typ riadenia prístupu
MITM	Man In The Midle – typ útoku na internetovú komunikáciu
MobSF	Mobile Security Framework – Nástroj na statickú aj dynamickú analýzu

NFC	Near Field Communication – technológia umožňujúca bezkontaktné platby
PDF	Portable Document Format – formát dokumentov
PIN	Personal Identification Number – identifikačné číslo, slúžiace ako forma hesla
RAM	Random-Access Memory – pamäť z náhodným prístupom
SAST	Static Application Security Testing – statické testovanie bezpečnosti aplikácie
SEL	Security Enhanced Linux – definuje riadenie prístupu pre aplikácie, procesy
SHA	Secure Hash Algorithm – skupina populárnych kryptografických algoritmov
SMS	Short Message Service – služba umožňujúca posielať krátke textové správy
SID	Secure Identifier – používané na reprezentáciu používateľa v TEE
SSL	Secure Socket Layer – technológia pre bezpečné internetové pripojenie
TEE	Trusted Execute Environment – bezpečné prostredie pre spúšťanie kódu
UID	Unique Identifier – unikátny identifikátor
VM	Virtual Machine – virtuálne zariadenie

ZOZNAM OBRÁZKOV