# Bezpečnostné mechanizmy mobilných aplikácií bežiacich pod operačným systémom Android

Martin Kubíček

Bakalárska práca
2021



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

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

Akademický rok: 2020/2021

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

| | |
|---|---|
| Jméno a příjmení: | **Martin Kubíček** |
| Osobní číslo: | **A18164** |
| Studijní program: | **B3902 Inženýrská informatika** |
| Studijní obor: | **Informační a řídicí technologie** |
| Forma studia: | **Prezenční** |
| Téma práce: | **Bezpečnostní mechanismy mobilních aplikací běžících pod operačním systémem Android** |
| Téma práce anglicky: | **Security mechanisms of mobile applications for Android operating system** |

## Zásady pro vypracování

1. Vypracujte literární rešerši na dané téma.
2. Proveďte analýzu současných bezpečnostních standardů a doporučení, které jsou aplikovatelné na bezpečný vývoj mobilních aplikací, například OWASP, Google Security for Android Developers apod.
3. Implementujte vybrané korektní bezpečnostní postupy formou praktické demonstrační aplikace.
4. Proveďte shrnutí poznatků zjištěných během implementačního procesu.
5. Vypracujte soubor pravidel určených pro vývojáře, které napomohou bezpečnému vývoji mobilních aplikací určených pro Android OS.

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

Seznam doporučené literatury:


1. Android Apps Security: Mitigate Hacking Attacks and Security Breaches. 2020. 2nd Edition, Kindle Edition. Singapore: Apress. ISBN 978-1484216811.
2. ELENKOV, Nikolay. 2014. Android Security Internals: An In-Depth Guide to Android's Security Architecture. USA: No Starch Press. ISBN 978-1593275815.
3. KIM, Peter. 2018. The Hacker Playbook 3: Practical Guide To Penetration Testing. USA: Secure Planet. ISBN 978-1980901754.
4. WHITEHOUSE, Ollie. [2015]. The Mobile application hacker's handbook. Indianapolis: John Wiley. ISBN 978-1118958506.
5. DRAKE, Joshua J. 2014. Android hacker's handbook. Indianapolis: Wiley. ISBN 978-1118608647.
6. Hacking Android: Ethical Hacking,Android hacker, Phone hacking,Learn all about Android to modify and protect your device Against security threats. 2020. Kindle Edition. unknown: independently published. ISBN 979-8688866513.


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

Datum zadání bakalářské práce: **15. ledna 2021**
Termín odevzdání bakalářské práce: **17. května 2021**

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

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

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

Ve Zlíně dne 15. ledna 2021

**Martin Kubíček**

**Bezpečnostné mechanizmy mobilných aplikácií bežiacich pod operačným systémom Android**

**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 14.5.2021

Martin Kubíček v.r.
Podpis studenta

## ABSTRAKT

Táto bakalárska práca sa zaoberá zabezpečením mobilných aplikácií pre Android OS. Cieľom bolo analyzovať súčasné bezpečnostné odporúčania a mechanizmy, a následne navrhnúť sadu pravidiel, ktorých použitím pri vývoji bude možné vytvoriť komplexné zabezpečenie a ochranu proti zraniteľnostiam. Výsledkami práce sú implementácia navrhnutých pravidiel v zabezpečenej aplikácii, ukážka rôznych útokov prostredníctvom popisovaných zraniteľností a následné testovanie útokov na zabezpečenú aplikáciu.


Kľúčové slova: Zraniteľnosť, Android, Útok, Data, Root, Zabezpečenie, Šifrovanie, Odporúčanie, SQL Injection


## ABSTRACT

This Bachelor's thesis deals with Android mobile applications security. The objective was to analyze current security recommendations and mechanisms and thereafter suggest a set of rules. Application of those rules in development would help to create complex security and protection against vulnerabilities. The results of the thesis are implementation of suggested rules in secure application, demonstration of various attack through described vulnerabilities and subsequent testing of the attacks to secure application.


Keywords: Vulnerability, Android, Attack, Data, Root, Security, Encryption, Recommendations, SQL Injection

# POĎAKOVANIE

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

# OBSAH

# ÚVOD

Informačná bezpečnosť– čoraz viac diskutovaná téma, ktorej dôležitosť si veľa ľudí neuvedomuje. Takmer každý z nás používa určitú formu zariadení s prístupom na internet, pričom môžeme povedať, že medzi týmito zariadeniami sú v najväčšom množstve zastúpené práve chytré mobilné telefóny a tablety, kde dominuje v počte zariadení operačný systém Android, na ktorom beží až takmer 72 % zariadení [1].

Nepochybne sú tieto chytré zariadenia veľkým pomocníkom, avšak vystavujú nás rizikám, o ktorých veľa užívateľov ani len netuší. Často krát tieto zariadenia, alebo priamo užívateľ poskytujú citlivé údaje osobám či firmám, o ktorých vôbec nič nevedia a tým sa vystavujú nevedomému riziku. Často sú týmto rizikom útočníci a tvorcovia mobilného malwaru. Tí sa snažia pomocou rôznych útokov na aplikácie dáta získať, či už pre vlastný prospech alebo ich ďalej predávajú na čiernom trhu. Na takéto aktivity skúsení útočníci využívajú zraniteľnosti aplikácií, čo sú v podstate chyby v zabezpečení.

Práve zraniteľnosti a hlavne ochrana voči nim je predmetom tejto bakalárskej práce. Vzhľadom na počet rôznych známych útokov a fakt, že nové metódy útokov sú vynachádzané každý deň, je dôležitá nie len znalosť útokov a zraniteľných miest v aplikáciách, ale aj účinné metódy obrany a rýchla reakcia na novoobjavené útoky.

Práca je zameraná na popis zraniteľností mobilných aplikácií, ktoré môžu byť využité, na už spomínané útoky. Ďalšou dôležitou témou sú bezpečnostné štandardy využívané spoločnosťami, ale aj jednotlivcami, ktorý sú odborníkmi v danej problematike. Tie by mali byť akýmsi východzím bodom pri vývoji akejkoľvek aplikácie, pretože bez základných bezpečnostných opatrení, môže byť aplikácia veľmi ľahkým cieľom útokov.

V praktickej časti tejto práce, popisuje implementáciu daných bezpečnostných postupov a štandardov v zabezpečenej aplikácii. Popisuje tiež útoky na zraniteľnú aj zabezpečenú aplikáciu, čo umožnilo overiť kvalitu navrhovaných metód. Výsledkom práce je súbor pravidiel, určený pre bezpečný vývoj aplikácií a implementácia týchto pravidiel.

# I. TEORETICKÁ ČASŤ

# 1 LITERÁRNA REŠERŠ NA DANÚ TÉMU

## 1.1 Aplikácia

Aplikácia je spustiteľný program, ktorý vykonáva nejakú vopred vytvorenú funkcionalitu. Veľké množstvo aplikácií využíva architektúru klient-server. Klientská časť beží na mobilnom telefóne/tablete používateľa a posiela žiadosti na serverovú časť. Klient, teda aplikácia môže byť nainštalovaná v podstate dvoma spôsobmi. Za prvé je to stiahnutím z neoficiálnych zdrojov, ako sú rôzne webové stránky, torrenty, atď., čo sa neodporúča práve kvôli tomu, že takéto aplikácie môžu predstavovať veľké bezpečnostné riziko. Za druhé, môže byť aplikácia stiahnutá z oficiálnych distribútorských platforiem ako sú Google Play pre Android OS alebo AppStore pre iOS. Sťahovanie z takýchto overených zdrojov je nepochybne bezpečnejšie ale aj napriek tomu nie je vylúčené, že aplikácie v sebe skrývajú zraniteľnosti. Serverová časť aplikácie slúži na ukladanie, získavanie dát z databáz, a reagovanie na požiadavky
od klientov. Samozrejme, existujú aj aplikácie, ktoré fungujú čisto off-line, bez prístupu na internet tzn. bežia, a ukladajú dáta iba na zariadenie používateľa ako napríklad kalkulátor, textové editory a tak podobne. Vzhľadom na to, že prevažujú klient-server aplikácie, budeme sa venovať hlavne tomuto typu.

## 1.2 Príklad fungovania aplikácie

Fungovanie aplikácie môže byť demonštrované na aplikácii slúžiacej k zobrazovaniu počasia. Používateľ otvorí aplikáciu a nastaví, že chce zobraziť, aké bude zajtra počasie. Požiadavka je pri otvorení aplikácie odoslaná cez internet na server spoločnosti, ktorá prevádzkuje aplikáciu. Po príchode požiadavky ju server vyhodnotí, a začne vyhľadávať meteorologické dáta pre oblasť, v ktorej sa užívateľ nachádza. Po nájdení potrebných dát ich server odošle naspäť a používateľovi sa zobrazia na mobilnom zariadení. Aplikácia však pre správne určenie počasia, pochopiteľne, potrebuje aj polohu používateľa. Poloha je teda posielaná na server spolu s ostatnými požiadavkami. Tu môže ale vzniknúť problém a to napríklad v prípade, že by bol daný prenos sledovaný útočníkom. V prípade, že by bola odosielaná len požiadavka na to, aké bude zajtra počasie, je pre útočníka táto informácia nezaujímavá. Ak je však spolu s ňou posielaná aj spomínaná poloha, tá môže byť s určitosťou považovaná za citlivú informáciu a zneužitá útočníkom.

## 1.3 Zraniteľnosti aplikácií

V tejto kapitole sa pozrieme na zraniteľnosti, ktoré môžu útočníci využiť na získanie citlivých dát. Podľa nadácie OWASP, ktorej cieľom je zlepšenie zabezpečenia softwaru.

The OWASP Foundation (Open Web Application Security Project) je nezisková organizácia, ktorá nepatrí žiadnej technologickej spoločnosti a nie je orientovaná na tvorbu zisku. OWASP je teda nezávislá organizácia poskytujúca nestranné informácie a ide o spoločenstvo expertov, ktorý sa snažia vylepšiť bezpečnostné postupy pri vývoji softwaru. Výsledky výskumov vytvorených touto organizáciou sú všeobecne prijímané odbornou verejnosťou. Veľká väčšina penetračných laboratórií spoločností zaoberajúcich sa bezpečným vývojom mobilných aplikácií zoznam Mobile TOP 10 dlhodobo sleduje a reaguje na publikované zmeny. Popisované zraniteľnosti sú citáciou tohto zoznamu [2].

**1. Improper Platform Usage**

**2. Insecure Data Storage**

**3. Insecure Communication**

**4. Insecure Authentication**

**5. Insufficient Cryptography**

**6. Insecure Authorization**

**7. Client Code Quality**

**8. Code Tampering**

**9. Reverse Engineering**

**10. Extraneous Functionality**

### 1.3.1 Improper Platform Usage (Nesprávne použitie platformy)

Kapitola vychádza z informácií popísaných v dokumente OWASP [3].

Ako už bolo spomenuté v úvode, na trhu s mobilnými zariadeniami dominuje operačný systém Android. Práve pre to je práca na tento operačný systém zameraná.

V súčasnej dobe existujú 2 hlavné operačné systémy používané v mobilných zariadeniach: Android a iOS. Ostatné mobilné platformy sú v dobe písania bakalárskej práce zanedbateľné.

Navyše pre Android OS výrobcovia vytvárajú rôzne nadstavby ako je Samsung One UI, MIUI od spoločnosti XIAOMI alebo napríklad EMUI od Huawei. Nadstavby umožňujú dostať do telefónu nový vzhľad či funkcie. Ale môžu negatívne ovplyvňovať stabilitu aplikácií alebo rýchlosť reakcie na zistenie bezpečnostného problému [44].

Toto je však dôvodom vzniku problémov pri vývoji aplikácií, pretože je veľká fragmentácia naprieč verziami, čo komplikuje ladenie programu. Nehľadiac na platformu, vývojári chcú aby ich aplikácia bola dostupná čo najväčšiemu počtu užívateľov. Majú teda v podstate 2 možnosti. Buď budú rovnakú aplikáciu programovať natívne pre každý systém zvlášť, alebo si zvolia „cross-platform“ prístup.

Podľa konzultačnej softwarovej firmy Brainspire [4] je cross-platform jasnou voľbou väčšiny firiem - „Vývoj mobilných aplikácií pre rôzne platformy umožňuje urýchlenie procesu vývoja a zníženie nákladov. Vďaka týmto dvom hlavným výhodám, je táto oblasť mimoriadne populárna a zaisťuje jej neustály rast v posledných rokoch. Cieľová skupina je totiž dostupná na viacerých platformách naraz a nie všetky spoločnosti sú schopné alokovať svoje zdroje na vytvorenie natívnych aplikácií pre každú z nich zvlášť. Platformové vývojové nástroje umožňujú vytvárať aplikácie pre viac mobilných platforiem naraz. Pri písaní kódu sa používajú programovacie jazyky tretích strán a nakoniec sa kód skompiluje do natívnej aplikácie pre každý OS.“

Operačné systémy alebo platformy používané mobilnými aplikáciami a zariadeniami poskytujú širokú škálu funkcií vrátane bezpečnostných funkcií. K nesprávnemu použitiu platformy dochádza, keď aplikácia tieto funkcie neimplementuje správne, čo umožňuje využiť ich ako cestu k útoku.

### 1.3.2 Insecure Data Storage (Nezabezpečené dátové úložisko)

Rada spoločností sa spolieha, že ukladanie dát na mobilných zariadeniach je bezpečné, čo nie je pravda. Závisí to ale aj na tom, kde sú dáta ukladané . Ak je totiž využívaný verejný dátový priestor, dáta sú viditeľné. Môže sa teda javiť ako bezpečné ukladať dáta do privátneho dátového priestoru. Toto však taktiež nie je úplne bezpečné, pretože napríklad pri roote zariadenia je porušená zásada sandboxingu aplikácií, a dáta môžu byť viditeľné. Útočník taktiež môže byť vytvorený a vložený škodlivý software alebo môže byť vykonaná modifikácia legitímnej aplikácie tak, aby bolo možné odcudziť uložené informácie. Je taktiež možné aj extrahovať dáta prostredníctvom USB.

K zraniteľným miestam v oblasti zabezpečenia dát dochádza, keď vývojové tímy predpokladajú, že používatelia alebo malware nebudú mať prístup k súborovému systému mobilného zariadenia a následným citlivým informáciám v úložisku dát. Súborové systémy sú ale za určitých okolností prístupné. Vývojárske spoločnosti a programátori by mali očakávať, že útočník alebo škodlivý software bude chcieť napadnúť úložisko s citlivými údajmi.

Root alebo jailbreaking mobilného zariadenia obchádza radu zabezpečení, týkajúcich sa súborového systému.

To môže viesť ku kompromitácii dát v najlepšom prípade pre jedného používateľa a v najhoršom pre mnohých používateľov. Zneužitie citlivých údajov môže mať za následok krádež identity, porušenie súkromia, podvody, poškodenie povesti atď. [5].

#### *1.3.2.1 Root zariadenia*

Nové mobilné zariadenia ako sú chytré mobilné telefóny a tablety sú dodávané s predinštalovaným operačným systémom a niektorými aplikáciami. Takéto zariadenia sú vybavené technickými zámkami (softwarom), ktoré obmedzujú možnosti, ktoré sa dajú so zariadením vykonávať. Napríklad prístup a modifikácie určitých častí systému, súborov, adresárov a prostriedkov mobilných zariadení. Taktiež nie je povolené napríklad pridávať vlastné motívy a upravovať ich. Aby tieto zmeny mohol užívateľ vykonať, potrebuje práva super užívateľa tzv. root.

V operačnom systéme s linuxovým jadrom, ako je Android, je root používateľom s najvyššou úrovňou oprávnení a prístupu. Pri používaní mobilného telefónu sa užívatelia do jeho operačného systému prihlasujú ako neprivilegovaní používatelia. Neprivilegovaný používateľ nemá nulové oprávnenie, ako by sa mohlo zdať z názvu. Je to obyčajný užívateľ, ktorý môže vykonávať len to, na čo má oprávnenie. Naopak root je ekvivalent administrátora, teda super užívateľa, ktorý má oveľa väčšie oprávnenia [6].

Aby sme to zhrnuli, root je teda najvyšší užívateľ, ktorý má v podstate neobmedzený prístup k súborovému systému telefónu. Tu ale vzniká spomínaný problém, kde výrobcovia spoliehajú na to, že k určitým súborom nemá užívateľ prístup, čo je pri normálnom používaní zariadenia pravda. S použitím rootu však už nie.

### 1.3.3 Insecure Communication (Nezabezpečená komunikácia)

Ako už názov napovedá, nezabezpečená komunikácia sa týka zraniteľnosti mobilných aplikácií, keď sú citlivé údaje zachytávané počas prenosu. Tento typ zraniteľnosti mobilných aplikácií sa môže vyskytovať u aplikácií, ktoré si vymieňajú údaje spôsobom klient-server, príklad o fungovaní aplikácie (viď. Kapitola 1.2). Po odoslaní dáta prechádzajú sieťou operátora mobilného zariadenia, prípadne cez Wi-Fi poskytovateľa a internetom až na server.

Posudzovanie náročnosti zneužitia tejto zraniteľnosti sa líši. Je napríklad jednoduchšie sledovať používateľov nezabezpečeného internetového pripojovacieho bodu v kaviarni alebo na vlakovej stanici, a naopak, je ťažšie sledovať cieľových používateľov v ich domácej zabezpečenej sieti, prípadne firmy v ich sieti.

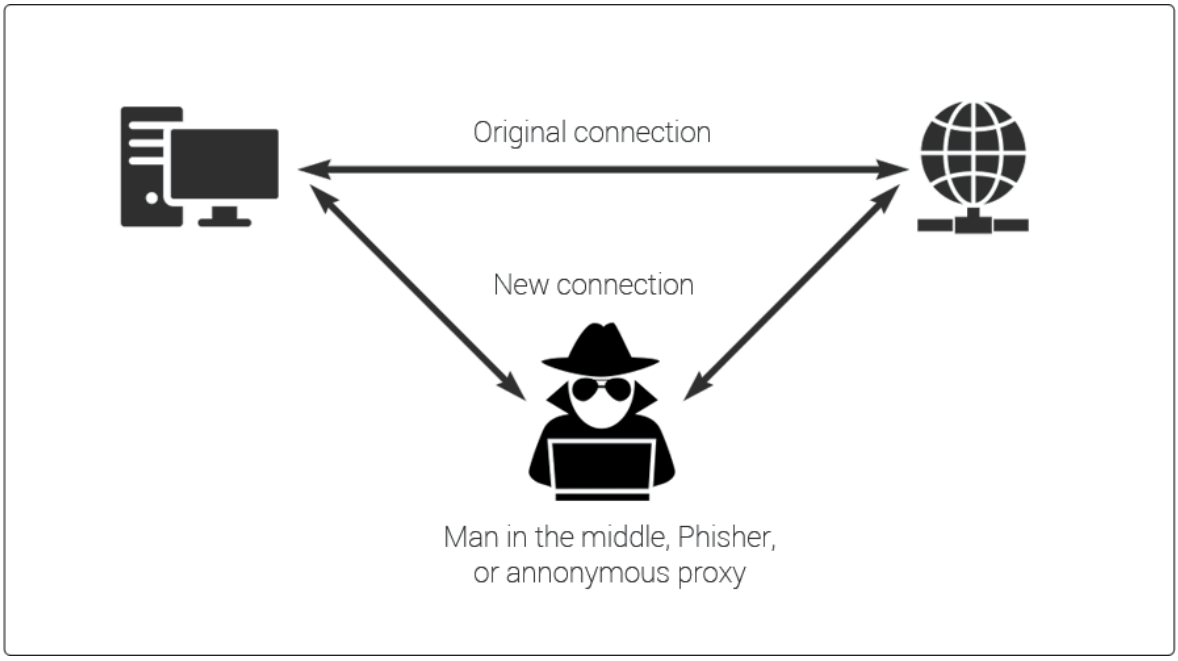
Útočník môže prevádzať útok prostredníctvom nezabezpečenej komunikácie v nasledujúcich scenároch:

1. Útočník s obeťou zdieľa miestnu sieť (napadnutá alebo monitorovaná sieť Wi-Fi, verejné siete)
2. Útočník úspešne vloží malware do mobilného zariadenia obete.

Mobilné aplikácie väčšinou používajú zabezpečenie prenosu, to však neznamená že je vždy použité a implementované správne. Niektoré zastarané aplikácie a servery dokonca môžu používať nešifrovaný prenos pomocou protokolu HTTP.

Ako náhle útočník využitím tejto chyby získa kontrolu nad komunikáciou obete, môže odchytiť niektoré citlivé informácie alebo údaje, či pozmeniť posielané dáta, čo nakoniec môže viesť až ku krádeži účtov, čísiel kreditných kariet a pod. [7].

Takéto útočné metódy sú často realizované pomocou Man In The Middle (MITM) útoku.




Obrázok 1. Man in the middle útok [37]

### *1.3.3.1 MITM*

Popis MITM útoku vychádza z informácií popísaných na webe getastra.com [8].

Úspešné vykonanie MITM útoku má dve odlišné fázy: odpočúvanie a dešifrovanie. V prípade, že užívateľ so serverom komunikuje prostredníctvom protokolu HTTP, krok dešifrovania nie je nutný. Takýto postup je ukázaný v kapitole 4.2.1.

V prvom kroku je zachytený prenos používateľa cez sieť útočníka skôr, ako sa dostane do určeného cieľa. Najbežnejším (a najjednoduchším) spôsobom, je pasívny útok, pri ktorom útočník sprístupní verejnosti škodlivé pripojovacie body Wi-Fi. Spravidla sú pomenované spôsobom, ktorý zodpovedá ich umiestneniu a nie sú chránené heslom. Ako náhle sa obeť pripojí k takémuto prístupovému bodu, získa útočník plný prístup k výmene nezašifrovaných údajov.

Útočníci, ktorí sa chcú aktívnejšie postaviť k odpočúvaniu, môžu podniknúť jeden z nasledujúcich útokov:

1. **IP spoofing** - "Pri IP spoofingu sú používané nástroje na úpravu zdrojovej adresy v hlavičke paketu, aby si prijímajúci počítačový systém myslel, že paket je z dôveryhodného zdroja, napríklad z iného mobilného zariadenia v legitímnej sieti, a prijal

ho. Pretože k tomu dochádza na úrovni siete, neexistujú žiadne vonkajšie príznaky neoprávnenej manipulácie" - kaspersky.com [45].

2. **ARP spoofing** – Podvrh odpovede na ARP dotaz. ARP dotaz slúži k prekladu IP adries príjemcov na ich MAC adresy. Týmto útokom je možné presmerovať pakety smerujúce na MAC adresu obete na svoju vlastnú MAC adresu a odchytávať tak informácie [54].

Existujú aj metódy, vďaka ktorým je možné vidieť aj zašifrované údaje. Po zachytení šifrovanej komunikácie je potrebné dešifrovať akýkoľvek obojsmerný prenos SSL bez upozornenia používateľa alebo aplikácie. Existuje niekoľko metód, ako to dosiahnuť:

1. Únos SSL, ku ktorému dochádza, keď útočník odovzdá sfalšované autentifikačné kľúče používateľovi aj aplikácii počas nadviazania spojenia TCP. Týmto sa nastaví to, čo sa javí ako bezpečné spojenie, aj keď v skutočnosti človek v strede ovláda celú reláciu. Dnes už je však viac používané TLS, čo je vylepšená a viac zabezpečená verzia SSL.

2. Odstránenie protokolu SSL degraduje pripojenie HTTPS na HTTP zachytením autentifikácie TLS odoslanej z aplikácie používateľovi. Útočník odošle používateľovi nezašifrovanú verziu webu aplikácie, pričom zachová zabezpečenú reláciu s aplikáciou. Útočník tak vidí celú komunikáciu používateľa ako nešifrovanú.

Medzi citlivé údaje, ktoré môže útočník takýmto spôsobom získať patria šifrovacie kľúče, heslá, informácie o súkromí používateľov, podrobnosti o účte, tokeny relácií, dokumenty atď. Tieto údaje môžu prichádzať do aplikácie zo serveru, z aplikácie na server alebo medzi zariadením a iným zariadením (napr. Terminál NFC alebo karta NFC).

### 1.3.4 Insecure Authentication (Nezabezpečená autentizácia)

#### *1.3.4.1 Autentifikácia*

Autentifikácia je proces zisťovania, či je niekto alebo niečo v skutočnosti tým za koho sa vydáva. Technológia autentifikácie poskytuje kontrolu prístupu do systémov. Overuje, či sú údaje používateľa v zhode s údajmi v databáze autorizovaných používateľov.

Používatelia sú zvyčajne identifikovaní pomocou určitého ID (email, užívateľské meno, číslo, kód...) a autentifikácia sa vykonáva, keď používateľ pristupuje do aplikácie alebo stránky a poskytne údaj patriaci k ID. Najviac sa používa autentifikácia, kedy užívateľ zadá svoje ID a heslo. Oba tieto údaje by mal poznať iba používateľ. Existujú však aj ďalšie metódy autentifikácie a ich použitie pri dvojfaktorovej alebo viacfaktorovej autentifikácii (MFA) [10].

#### *1.3.4.2 Nezabezpečená autentifikácia*

Takáto autentifikácia môže byť oklamaná alebo obídená. Útok môže prebiehať tak, že sú zadávané požiadavky na prihlásenie napríklad pomocou brute force útoku, kde v niektorých prípadoch je možné obísť obmezenie limitu pokusov o prihlásenie vymazaním privátnej pamäti aplikácie, na ktorú je útočené. Útok je vykonávaný prostredníctvom mobilného malwaru v rámci zariadenia alebo útokom pomocou botnetov. Botnety môžu byť taktiež využité
na útoky DDoS na back-end server mobilnej aplikácie.

„Botnety sú siete unesených počítačových alebo mobilných zariadení používaných na vykonávanie rôznych podvodov a kybernetických útokov. Termín „botnet“ je tvorený slovom „robot“ a „sieť“. Roboti slúžia ako nástroj na automatizáciu hromadných útokov, ako je krádež dát, distribúcia škodlivého softwaru a DDoS útoky“ – usa.kaspersky.com [11].

Používateľ ani nevie, že jeho zariadenie je na takéto účely využívané a spozorovať, že niečo nie je v poriadku sa dá až napríklad spontánnym spomalením mobilného zariadenia. Zle implementované schémy overovania umožňujú potenciálnemu útočníkovi anonymne vykonávať funkcie v rámci mobilnej aplikácie alebo servera typu back-end. Slabšia autentifikácia v mobilných zariadeniach môže vyplynúť z toho, ako sú implementované autentizačné formuláre. Požiadavky na autentifikáciu pre mobilné aplikácie sa môžu úplne líšiť od tradičných schém autentifikácie webu kvôli požiadavkám na dostupnosť. Prístupové formuláre pre mobilné zariadenia môžu podporovať krátke heslá, ktoré sú založené výlučne na štvormiestnych kódoch PIN, prípadne je uložený prístupový token v telefóne, čo má za ciel minimalizovať nutné prihlásenia [10].

### 1.3.5 Insufficient Cryptography (Nedostatočná kryptografia)

Kryptografia sa zaoberá šifrovaním informácií a je základným princípom zabezpečenia informácií. V podstate uniknutá alebo neoprávnene získaná informácia, ktorá je nečitateľná,

nemá žiadnu výpovednú hodnotu a nemôže byť zneužitá. Šifrovacie metódy čelia celej rade problémov, z ktorých je najvýznamnejší neustále sa zvyšujúci výpočtový výkon. So zvyšujúcim sa výpočtovým výkonom počítačov sa podarilo prelomiť šifry, ktoré boli skôr považované za bezpečné. Chyby spojené implementáciou šifrovacích algoritmov môžu byť nasledovné [12].

#### *1.3.5.1 Nebezpečné a neodporúčané algoritmy*

Základnou chybou pri vývoji aplikácií je to, že vývojári na „zabezpečenie“ používajú šifry, ktoré už nie sú odporúčané ako napríklad TDEA [46], alebo sú dokonca považované za nebezpečné DES [47]. Tieto šifry môžu byť využívané z dôvodu, že majú jednoduchú implementáciu a sú rýchle.

#### *1.3.5.2 Nebezpečné uloženie kľúča*

Rovnako ako slabé šifry, môže bezpečnosť dát ohroziť aj zlá manipulácia s kľúčom alebo jeho nebezpečné uloženie. Kľúč by sa nemal nachádzať vo verejnej časti perzistentnej pamäte mobilného zariadenia, SD kartách a celkovo by nemal byť v nešifrovanej podobe umiestnený v žiadnej časti súborového systému mobilného zariadenia, ktorá je prístupná malwaru alebo útočníkovi.

#### *1.3.5.3 Implementácia vlastných algoritmov*

Veľmi zásadnou chybou je vytváranie vlastných šifrovacích algoritmov. Proces tvorby šifry je veľmi náročný a šifra musí prejsť celým procesom publikácie, musí stáť na matematických základoch a prejsť dôkladnou kryptoanalýzou a oponentúrou. Pokiaľ by tieto podmienky neboli splnené, šifra nemôže byť považovaná za bezpečnú a nemôže byť používaná. Pri tvorbe komerčných aplikácií je teda tento postup z hľadiska časovej náročnosti neprípustný.

#### *1.3.5.4 Bezpečná šifra, slabý kľúč*

Bezpečnosť každej šifry je nejakým spôsobom závislá na kľúči. Napríklad šifra AES je považovaná za bezpečnú a odporúčanú. Avšak pokiaľ by bola dĺžka kľúča 128 bitov, je bezpečnosť tejto šifry spochybniteľná. Je teda odporúčané používať kľúč o dĺžke 256 bitov, prípadne viac. Pokiaľ je používaná pri implementácii nejaká šifra, vždy je potrebné overiť, aká dĺžka kľúča je bezpečná pre danú šifru.

#### *1.3.5.5 Implementácia vlastných kryptografických primitív*

Podobne ako u problému s implementáciou vlastných šifrovacích algoritmov, pri tvorbe aplikácie pre komerčné účely nemôže jeden vývojár dostatočne overiť a preukázať bezpečnosť kryptografickej primitívy. Ďalším dôvodom, prečo sa vlastnej implementácii vyhnúť je nie len časová náročnosť počas implementačného procesu, ale aj prípadná nutnosť udržovať navrhnuté primitívy aktuálne a bezpečné. V opačnom prípade by ich použitie mohlo predstavovať riziko.

### 1.3.6 Insecure Authorization (Nezabezpečená autorizácia)

#### *1.3.6.1 Autorizácia*

„Autorizácia je proces, ktorý niekomu umožňuje prístup k prostriedku. Samozrejme, táto definícia môže znieť nejasne, ale veľa situácií v reálnom živote nám môže pomôcť ilustrovať, význam tohto pojmu.

Dobrým príkladom je vlastníctvo domu. Vlastník má úplné prístupové práva k nehnuteľnosti (zdroju), môže však poskytnúť iným osobám právo na prístup k nej. Hovoríme teda, že vlastník oprávňuje ľudí na prístup. Tento jednoduchý príklad nám umožňuje zaviesť niekoľko konceptov v kontexte autorizácie. Napríklad prístup do domu je povolenie, tzn. akcia, ktorú môžete vykonať. Ďalším povolením, ktoré môžeme mať, je jeho zariaďovanie, čistenie, opravy atď. Povolenie sa stáva privilégiom (alebo právom), keď je niekomu pridelené. Takže ak pridelíte povolenie zariadiť dom vášmu interiérovému dizajnérovi, udeľujete mu privilégium“ - Auth0.com [14].

#### *1.3.6.2 Nezabezpečená autorizácia*

Útočníci, ktorí využívajú chyby zabezpečenia, zvyčajne tieto útoky realizujú prostredníctvom automatizovaných útokov, ktoré využívajú dostupné alebo na mieru vytvorené softwarové nástroje. Keď útočník pochopí, ako schéma autorizácie funguje a kde je zraniteľná, prihlási sa do aplikácie ako legitímny užívateľ a úspešne prejde kontrolou autorizácie. Následne prehľadáva aplikáciu aby našiel zraniteľný bod, na ktorom sa vykonali administratívne funkcie, a ktorý by mohol byť využitý na vedenie útoku.

Slabé alebo chýbajúce autorizačné schémy umožňujú útočníkovi vykonávať funkcie, na ktoré by nemal mať oprávnenie overený používateľ mobilnej aplikácie s nižšími

oprávneniami. Požiadavky na autorizáciu sú zraniteľnejšie pri autorizačnom rozhodovaní v mobilnom zariadení ako pri rozhodovaní prostredníctvom vzdialeného servera. To môže byť z dôvodu mobilných požiadaviek off-line použiteľnosti.

Technický dopad zlej autorizácie je podobný ako technický dopad zlej autentifikácie. Môže mať rôznu povahu a závisieť od funkčnosti, ktorá sa vykonáva s vyššími oprávneniami ako by mala. Napríklad vykonávanie neoprávnenej správcovskej funkcionality môže mať za následok prístup k citlivým informáciám [13].

### 1.3.7 Client Code Quality (Kvalita kódu klienta)

Tieto typy problémov nemusia byť nevyhnutne samy o sebe bezpečnostnými problémami, ale môžu viesť k iným slabým miestam zabezpečenia, ktoré môžu byť zneužité útočníkom.

Útočník zvyčajne zneužíva zraniteľné miesta poskytovaním starostlivo vytvorených vstupov svojej obeti. Tieto vstupy sa prenášajú do kódu, ktorý sa nachádza v mobilnom zariadení, kde dochádza k zneužitiu. Takéto útoky väčšinou využívajú úniky pamäte tzv. "memory leaks" a pretečenia vyrovnávacej pamäte, prípadne chyby ošetrenia vstupov.

Problémy s kvalitou kódu sa môžu vyskytovať vo veľkom množstve mobilných kódov. Dobrou správou je, že väčšina problémov s kvalitou kódu je pomerne neškodná a vzniká kvôli zlému postupu pri programovaní. Prísť na takéto problémy prostredníctvom manuálnej kontroly kódu je zvyčajne časovo náročná a je vyžadovaná vysoká odbornosť analytikov.

Útoky, ktoré spadajú do tejto kategórie, môžu mať za následok vykonanie cudzieho kódu alebo znemožnenie služby na koncových bodoch vzdialeného servera (a nie samotnom mobilnom zariadení) [15].

#### *1.3.7.1 Code Injection*

Code injection je nebezpečný útok, ktorý využíva chybu spôsobenú spracovaním neplatných údajov a je používaná útočníkmi na vloženie kódu do zraniteľného programu.

K chybám dochádza, keď aplikácia spracuje nedôveryhodné údaje. Injekčné chyby sú veľmi rozšírené, najmä v staršom kóde alebo kóde, ktorý písali vývojári s neznalosťou takýchto útokov. Často sa nachádzajú v dotazoch SQL, LDAP, Xpath alebo NoSQL, Analyzátoroch XML, hlavičkách SMTP, argumentoch atď. Chyby pri vkladaní kódu sú ľahko odhaliteľné pri skúmaní kódu, ale často je ťažké ich odhaliť prostredníctvom testovania.

Ďalšie možné dopady sú:

1. Upravenie hodnôt v databáze alebo ich získanie pomocou útok SQL Injection.

2. Inštalácia malwaru alebo vykonanie škodlivého kódu na serveri vložením skriptovacieho kódu servera (napríklad PHP alebo ASP).

3. Zvýšenie oprávnení na oprávnenie root pomocou zneužitia zraniteľnosti Shell Injection v binárnom prostredí root setuid v systéme UNIX alebo lokálnom systéme využitím služby v systéme Windows [17].

#### *1.3.7.2 SQL Injection*

SQL injection je zjavne najznámejší z injection útokov. Je to chyba zabezpečenia, ktorá je spôsobená neošetrovaním zadávaných vstupov. To umožňuje útočníkovi zadávať reťazce znakov, ktoré útočníkovi umožňujú zobraziť údaje, ktoré bežne nedokáže načítať. Môžu to byť údaje patriace iným používateľom alebo akékoľvek iné údaje, ku ktorým má samotná aplikácia prístup. V mnohých prípadoch môže útočník tieto údaje upraviť alebo vymazať, čo spôsobí trvalé zmeny v obsahu alebo správaní aplikácie.

V niektorých situáciách môže útočník eskalovať útok SQL Injection, aby ohrozil podkladový server alebo inú back-endovú infraštruktúru.

Úspešný útok môže mať za následok neoprávnený prístup k citlivým údajom, ako sú heslá, údaje o kreditných kartách alebo osobné informácie o používateľovi. Mnoho významných únikov citlivých údajov v posledných rokoch bolo výsledkom útokov SQL Injection, čo viedlo k poškodeniu reputácie a regulačným pokutám. V niektorých prípadoch môže útočník získať pretrvávajúce zadné vrátka do systémov organizácie a čerpať z nich dáta, čo môže zostať dlhý čas bez povšimnutia.

Niektoré bežné príklady SQL injection zahŕňajú:

1. Načítanie skrytých dát, kde môže útočník upraviť dotaz SQL, aby sa vrátil ďalšie výsledky, ku ktorým by normálne nemal mať prístup.

2. Oklamanie prihlasovacej logiky, kde je možné vykonať SQL injection na prihlasovacie údaje a získať tak prístup do aplikácie (viď kapitola 4.5.1).

3. Útoky UNION, kde môže načítať údaje z rôznych databázových tabuliek.

4. Preskúmanie databázy, kde môže extrahovať informácie o verzii a štruktúre databázy, čím získa informácie pomocou ktorých môže prevádzať ďalšie útoky.

5. Slepé vkladanie SQL, kde útočník zisťuje ako aplikácia odpovedá a akým spôsobom by bolo možné ju napadnúť [18].

### 1.3.8 Code Tampering (Neoprávnená manipulácia s kódom)

Útočník si stiahne aplikáciu, ktorú následne rozbalí a vloží do nej škodlivú funkcionalitu. Zároveň je odstránená softwarová ochrana. Následne je aplikácia znova zabalená a podpísaná. A útočník môže oklamať používateľa, aby si nainštaloval infikovanú aplikáciu prostredníctvom phishingových útokov, prípadne webových stránok a torrentov. To môže útočníkovi poskytnúť priamu cestu k narušeniu zamýšľaného použitia softwaru na osobný alebo peňažný zisk.

Upravené formy aplikácií sú prekvapivo bežnejšie, ako sa môže zdať. Existuje celý bezpečnostný priemysel založený na detekcii a odstraňovaní neautorizovaných verzií mobilných aplikácií v obchodoch s aplikáciami. V závislosti na prístupe organizácií k riešeniu problému s detekciou úpravy kódu, môžu mať obmedzené až veľmi úspešné spôsoby detekcie neautorizovaných verzií kódu vo vyskytujúcich sa aplikáciách.

Dopad tejto zraniteľnosti môže byť rôzny, v závislosti od samotnej úpravy kódu. Medzi dopady tak môžu patriť napríklad neoprávnená nová funkcionalita, krádež identity alebo rôzne podvody. Takáto zraniteľnosť je náchylná aj na code injection útoky [16].

Rovnako tu môže zohrávať úlohu aj už spomínaný root zariadenia (viď kapitola 1.3.2)

### 1.3.9 Reverse Engineering (Reverzné inžinierstvo)

Útočník si zvyčajne stiahne zacielenú aplikáciu z obchodu s aplikáciami a analyzuje ju pomocou nástrojov ako Dex2Jar, JADX, APKTOOL, JEB Decompiler atď.

Útočník musí vykonať analýzu aplikačnej logiky, aby získal, zdrojový kód SMALI, ktorý následne analyzuje aby mu bolo umožnené zistiť, knižnice, algoritmy a zdroje vložené do aplikácie.

Všeobecne je všetok Androidový kód zraniteľnejší reverzným inžinierstvom. Aj ostatné spustiteľné súbory môžu byť reverzne analyzované, napríklad nástrojom GDB, ktorý podporuje jazyky ako Objective C, Fortran, Rust, C, C++ atď [48]. Niektoré aplikácie sú náchylnejšie ako iné. Kód napísaný v jazykoch/frameworkoch, ktoré umožňujú dynamickú introspekciu za behu (Java, .NET, Objective C, Swift), je obzvlášť ohrozený.

Pri Androide, kde je logika uložená v classes.dex, classes2.dex a pod. je nemožné schovať kód pred útočníkom. Je teda možné prevádzať analýzu, avšak vykonať nejaký útok prostredníctvom reverzného inžinierstva nie je až tak jednoduché.

Útočník môže využiť reverzné inžinierstvo napríklad na odhalenie informácií o back-endových serveroch, odhalenie kryptografických mechanizmov a šifier, krádež duševného vlastníctva alebo získať znalosti potrebné na vykonanie úpravy kódu a tým reverzné inžinierstvo využiť na príklad na vloženie cudzieho kódu (viď Kapitola 1.3.8) alebo vykonanie irelevantnej funkcionality (viď kapitola 1.3.10), či iných útokov [19].

#### *1.3.9.1 APK Balíček*

APK (Android Application Package) je inštalačný súbor pre zariadenia, ktoré bežia pod operačným systémom Android. Nejedná sa o špeciálny formát ale o ZIP archív. Je možné ho dekomprimovať. Môže byť prirovnaný k súborom setup.exe pre windows.

#### *1.3.9.2 Možné útoky pomocou reverzného inžinierstva*

**Analýza tabuľky reťazcov:**

Výsledkom analýzy tabuľky reťazcov je, že útočník môže v kóde objaviť „hardcoded“ reťazec pripojenia, ktorý obsahuje autentifikačné údaje do back-endovej databázy. Útočník použije tieto pripojenia na získanie prístupu do databázy, kde má možnosť ukradnúť či zneužiť veľké množstvo citlivých údajov o používateľoch aplikácie.

**Analýza zdrojového kódu:**

Pomocou prevodníka Dex2Jar môže útočník ľahko konvertovať apk súbor na súbor jar. Väčšina zdrojov však nie je čitateľná, preto je potrebné v ďalšom kroku použiť Java Decompiler (napríklad JDGUI) sa útočník môže pohodlne dostať ku kódu. Existujú však aj rôzne iné programy s rovnakou funkcionalitou ako napríklad JADX, Apktool a pod. kde je možné zobrazenie súborov jedným krokom.

### 1.3.10 Extraneous Functionality (Irelevantná funkcionalita)

Útočník sa snaží porozumieť fungovaniu mobilnej aplikácie, aby našiel skryté funkcie v back-endových systémoch, ktoré by mohol nejakým spôsobom zneužiť. Útočník zvyčajne využije nadbytočné funkcie priamo zo svojho vlastného zariadenia bez akejkoľvek účasti iných koncových používateľov.

Útočník si stiahne a preskúma mobilnú aplikáciu. Preskúma používané protokoly a dôležité súbory ako manifest.xml, použité layouty a dekompilovaný kód, s cieľom nájsť akékoľvek skryté podmienky, funkcie alebo testovací kód, ktorý vývojári v kóde zanechali. Práve tieto skryté funkcie v back-endovom systéme využijú na vykonanie útoku. Modifikácie kódu však musia byť vykonané v nižšom jazyku, čo môže predstavovať problém.

Existuje veľká pravdepodobnosť, že akákoľvek mobilná aplikácia obsahuje nadbytočné funkcie, ktoré nie sú používateľovi priamo vystavené prostredníctvom rozhrania. Väčšina tohto dodatočného kódu však útočníkovi neposkytne žiadny ďalší prehľad o možnostiach back-endu. Niektoré funkcie však môžu byť pre útočníka veľmi užitočné, a to napríklad funkcie, ktoré odhaľujú informácie týkajúce sa back-end testovacích, ukážkových alebo UAT prostredí, či hardcoded credentials, teda „natvrdo“ zadané prihlasovacie údaje v kóde. Takéto veci by s určitosťou nemali byť súčasťou produkčného kódu. Na kontrolu kódu môžeme použiť automatizované statické a dynamické analytické nástroje, ktoré dokážu zachytiť niektoré jednoduchšie a nápadnejšie funkcionality, avšak niektoré je ťažké objaviť automatizovaným spôsobom [20].

#### *1.3.10.1 Príklad útoku na túto zraniteľnosť*

“V rámci testovania mobilných koncových bodov vývojári zahrnuli do mobilnej aplikácie skryté rozhranie, ktoré by zobrazovalo administratívny informačný panel. Tento informačný panel pristupoval k informáciám o správcovi prostredníctvom back-endového servera API. Do produkčnej verzie kódu vývojári nezahrnuli kód, ktorý by informačný panel zobrazoval kedykoľvek. Zahrnuli však kód, ktorý mal prístup k rozhraniu API administrátora. Útočník vykonal analýzu binárneho súboru s reťazcovou tabuľkou a objavil pevne zakódovanú adresu URL do administratívneho koncového bodu REST a tým získal prístup k informačnému panelu” [20].

# 2 ANALÝZA SÚČASNÝCH BEZPEČNOSTNÝCH MECHANIZMOV

V predchádzajúcej kapitole sme si popísali zraniteľnosti, ktoré najviac ohrozujú bezpečnosť používateľov a ich dáta. Poďme sa teda pozrieť na to, ako na takéto zraniteľnosti reagujú veľké firmy a profesionáli v tejto oblasti a ako sa im brániť.

## 2.1 Insecure Data Storage (Nezabezpečené dátové úložisko)

### 2.1.1 OWASP

Táto kapitola je citovaná z odporúčaní OWASP [5].

Zásadným pravidlom mobilných aplikácií je neukladať údaje, pokiaľ to nie je nevyhnutne potrebné. Vývojári musia predpokladať, že dáta môžu byť napadnuté hneď, ako sa dostanú do telefónu. Taktiež je potreba vziať do úvahy dôsledky straty dát mobilných používateľov. Dôležité je zistiť, aké údaje sa ukladajú a primerane ich chrániť.

Podľa organizácie OWASP by pri vytváraní aplikácií malo byť použitých týchto 5 pravidiel:

1. Pre lokálne úložisko možno pomocou rozhrania API pre správu zariadení Android vynútiť šifrovanie do miestnych úložísk súborov pomocou príkazu „setStorageEncryption“.

2. U úložiska SD kariet možno dosiahnuť určitú bezpečnosť prostredníctvom knižnice „javax.crypto“. Existuje niekoľko možností, ale najjednoduchšia je proste zašifrovať všetky údaje v obyčajnom texte pomocou hlavného hesla a AES 128 (v dnešnej dobe je doporučované použiť AES 256 [49]).

3. Je nutné zaistiť, aby vlastnosti zdieľaných preferencií neboli MODEWORLDREADABLE, pokiaľ to nie je výslovne nevyhnutné na zdieľanie informácií medzi aplikáciami.

4. Pri ukladaní prostriedkov citlivých informácií nie je možné spoliehať sa na „hardcoded“ šifrovacie alebo dešifrovacie kľúče.

5. Je dobré zvážiť poskytnutie ďalšej vrstvy šifrovania nad rámec predvolených mechanizmov šifrovania poskytovaných operačným systémom.

### 2.1.2 Android developers

Android poskytuje už preddefinovaný stupeň ochrany aplikácií a to pomocou funkcionalít ako sandbox aplikácií pre Android, ktorý izoluje jednotlivé aplikácie od ostatných aplikácií, či aplikačný framework s robustnými implementáciami bežných bezpečnostných funkcií, ako je kryptografia, oprávnenia a zabezpečené IPC (Interprocess Communication).

Na oficiálnych stránkach Android OS pre vývojárov sú odporúčania, ktoré poukazujú na potrebu využívať ďalšie funkcionality, na zabezpečenie aplikácií:

1. Šifrovaný súborový systém, ktorý je možné povoliť na ochranu údajov na stratených alebo odcudzených zariadeniach.

2. Užívateľom udelené povolenia na obmedzenie prístupu k funkciám systému a údajom používateľa.

Rovnako ako u OWASPu je aj tu odporúčané vyhnúť sa MODEWORLDREADABLE a používať šifrovanie. Odporúčania ohľadom šifrovania sú rozobrané v kapitole 2.4. [21].

## 2.2 Insecure Communication (Nezabezpečená komunikácia)

### 2.2.1 OWASP - Všeobecné osvedčené postupy

Kapitola vychádza z informácií popísaných v zozname odporúčaní OWASP [7].

Platí predpoklad, že sieťová vrstva nie je bezpečná a je náchylná na odpočúvanie.

1. Je odporúčané použiť protokol SSL/TLS na prenosové kanály, ktoré mobilná aplikácia používa na prenos citlivých informácií, tokenov relácií alebo iných citlivých údajov do rozhrania API alebo webovej služby typu back-end.

2. Nie je bezpečné používať zmiešané relácie SSL, pretože by mohli prezradiť ID relácie používateľa.

3. Je nutné použiť silné štandardné šifrovacie sady s príslušnou dĺžkou kľúča a certifikáty podpísané dôveryhodným poskytovateľom CA.

4. Nikdy by nemalo byť povolené používať certifikáty s vlastným podpisom a malo by sa zvážiť pripnutie certifikátu pre aplikácie, ktoré vyžadujú vyššiu bezpečnosť.

5. Vždy je nutné vyžadovať overenie reťazca SSL.

6. Zabezpečené pripojenie by malo byť vytvorené až po overení totožnosti servera koncového bodu pomocou dôveryhodných certifikátov v reťazci kľúčov.

7. Používatelia aby mali byť varovaní prostredníctvom používateľského rozhrania, ak mobilná aplikácia zistí neplatný certifikát.

8. Citlivé dáta by nemali byť odosielané cez alternatívne kanály (napr. SMS, MMS alebo oznámenia).

9. Pokiaľ je to možné, mala by byť aplikovaná samostatná vrstva šifrovania na všetky citlivé údaje pred ich poskytnutím kanálu SSL. V prípade, že sú v implementácii SSL zistené chyby, zašifrované údaje poskytnú sekundárnu ochranu proti narušeniu dôvernosti.

Novšie hrozby umožňujú útočníkovi odpočúvať komunikáciu tak, že zachytia prenos v mobilnom zariadení tesne predtým, ako ich knižnica SSL mobilného zariadenia zašifruje a prenesie sieťový prenos na cieľový server.

### 2.2.2 OWASP - Špecifické odporúčania pre Android

"Po vývojovom cykle by mal byť odstránený kód, ktorý aplikácii umožní, aby prijímala všetky certifikáty, ako napríklad *org.apache.http.conn.ssl.AllowAllHostnameVerifier* alebo *SSLSocketFactory*. *ALLOW_ALL_HOSTNAME_VERIFIER*. Všetky sú rovnocenné s dôverou vo všetky certifikáty.

Ak je používaná trieda, ktorá rozširuje *SSLSocketFactory*, je potrebné sa uistiť, že je správne implementovaná metóda *checkServerTrusted*, aby bol certifikát serveru správne skontrolovaný" [7].

### 2.2.3 Android - Zabezpečenie načúvacích portov

Spravidla by nemali byť na zariadeniach otvorené žiadne porty na odpočúvanie, pretože poskytujú pre vzdialeného útočníka možnosť na získanie prístupu k zariadeniu. Zariadenia so systémom Android by mali minimalizovať počet portov na počúvanie internetu, ktoré vystavujú najmä pri štarte, alebo v predvolenom nastavení. Pri štarte by štandardne nemal

počúvať žiadny port. Koreňové procesy a procesy vlastnené jedinečným identifikátorom systému (UID) by nemali vystavovať žiadne porty.

Doporučenie, ktoré bolo publikované v [22] : "For local IPC-using sockets, apps must use a UNIX domain socket with access limited to a group. Create a file descriptor for the IPC and make it +RW for a specific UNIX group. Any client apps must be within that UNIX group."

## 2.3 Insecure Authentication (Nezabezpečená autentizácia)

Ak je webová aplikácia prenášaná na jej mobilný ekvivalent, požiadavky na autentifikáciu mobilných aplikácií by sa mali zhodovať s požiadavkami komponentu webovej aplikácie. Preto by nemalo byť možné autentifikovať sa pomocou slabších autentifikačných faktorov ako vo webovom prehliadači.

Lokálna autentifikácia používateľa môže viesť k zraniteľnostiam na strane klienta. Ak aplikácia ukladá údaje lokálne, rutinu autentifikácie je možné na rootovaných zariadeniach obísť pomocou manipulácie s programom za behu alebo úpravy binárneho súboru.

Ak je to možné, je potrebné zabezpečiť, aby sa všetky požiadavky na overenie totožnosti vykonávali na strane servera. Po úspešnej autentifikácii sa údaje aplikácie načítajú do mobilného zariadenia. To zabezpečí, že údaje aplikácie budú k dispozícii až po úspešnej autentifikácii.

Ak je vyžadované ukladanie údajov na strane klienta, údaje budú musieť byť šifrované pomocou šifrovacieho kľúča, ktorý je bezpečne odvodený z prihlasovacích údajov používateľa. To zabezpečí, že uložené údaje aplikácie budú prístupné až po úspešnom zadaní správnych prístupových údajov.

Funkcia trvalého overovania (zapamätanie používateľa) implementovaná v mobilných aplikáciách by nikdy nemala ukladať heslo používateľa do zariadenia.

V ideálnom prípade by v mobilných aplikáciách mal byť využívaný autentifikačný token špecifický pre zariadenie, ktorý môže používateľ v mobilnej aplikácii odvolať. Takto je možné zabezpečiť, že aplikácia dokáže zmierniť neoprávnený prístup z odcudzeného/strateného zariadenia.

Na autentifikáciu používateľa by nemali byť použité žiadne sfalšovateľné hodnoty ako napríklad identifikátory zariadení alebo geografické umiestnenie. Trvalé overovanie

v mobilných aplikáciách by malo byť implementované ako prihlásenie a predvolene by nemalo byť povolené.

Pokiaľ je to možné, používateľom by nemalo byť dovolené používať štvormiestne čísla PIN na overenie hesla. Je potrebné predpokladať, že útočníci môžu obísť všetky kontroly autorizácie a autentifikácie na strane klienta. Vždy, keď je to možné, musia byť na strane servera opätovne vynútené kontroly autorizácie a autentifikácie.

Z dôvodu požiadaviek na off-line použitie sa môže od mobilných aplikácií vyžadovať, aby v rámci kódu mobilnej aplikácie vykonávali miestne kontroly autentifikácie alebo autorizácie. Ak je to tak, by mali byť vykonávané aj kontroly lokálnej integrity v rámci svojho kódu, aby sa zistili akékoľvek neoprávnené zmeny kódu.

Dobrou metódou autentizácie je použiť čoraz viac sa rozširujúce biometrické metódy odtlačkov prstov a podobne [9].

## 2.4 Insufficient Cryptography (Nedostatočná kryptografia)

Kapitola vychádza z informácií popísaných v dokumente “best practices for encryption in android™” [23].

### 2.4.1 Zariadenie nie je bezpečné

Je potreba predpokladať, že čokoľvek uložené v zariadení je citlivé na čítanie malwarom alebo útočníkom. Najdôležitejšie je nikdy neukladať tajný kľúč vo forme holého textu na zariadenie. Možnosť, ako zabrániť ukladaniu kľúča na zariadený je ho vygenerovať, kedykoľvek to bude potrebné a mal by byť odvodený od hesla, poskytnutého používateľom.

Aj keď šifrovanie nezabráni útočníkovi v načítaní zašifrovaných bitov zo súboru, bráni mu v zobrazení dešifrovaných údajov. Niektoré Mobile Device Management konzoly majú zavedenú funkcionalitu „remote wipe“ z dôvodu predpokladu, že čokoľvek uložené v zariadení môže čítať niekto, kto má zariadenie (bez ohľadu na oprávnenie súboru, ID používateľa atď.).

### 2.4.2 Java String

Triedu Java String by nemala byť používaná. Všetko, čo sa týka držania textov, šifrovania, dešifrovania, „salts“, inicializačných vektorov, seedov, hesiel atď. by sa malo vykonať pomocou char polí alebo bajtových polí.

Objekty typu String sú nemenné (static), to znamená, že nie sú definované žiadne metódy, ktoré by vám umožňovali zmenu (prepísať) alebo vynulovanie obsahu reťazca po použití. Táto vlastnosť robí objekty String nevhodnými na ukladanie citlivých bezpečnostných informácií, napríklad hesiel používateľov. Namiesto toho by sama mali citlivé informácie o bezpečnosti zbierať a ukladať do poľa typu char.

Z praktického hľadiska má prvok užívateľského rozhrania EditText v systéme Android priradené reťazce v rôznych bodoch implementácie. Nemôže sa teda postupovať podľa tohto osvedčeného postupu a používať komponent Android EditText na zbieranie hesiel alebo holého textu. Reťazec, ktorý je obdržaný z EditTextu, môže zostať viditeľný pre proces.

Tento problém je v Jave riešený knižnicou Swing GUI ktorá, má metódu *JPasswordField*, ktorá vracia pole typu char, nie String.

### 2.4.3 Nesnažte sa vytvárať vlastné šifry a kryptografické primitívy

Ako už bolo spomenuté, neexistuje jednoduchší spôsob ako urobiť chybu v šifrovaní, než implementáciou vlastných šifier a algoritmov.

Je nutné používať šifry, ktoré už sú vytvorené, a ktorých algoritmy sú uznávané ako nezávadné a dostatočne bezpečné. Samozrejme je treba zvážiť použitie danej šifry v závislosti na citlivosti dát, s ktorými aplikácia pracuje. Rovnako je treba zvážiť použitie symetrickej alebo asymetrickej kryptografie. Ďalším dôležitým parametrom je dĺžka kľúča, pretože aj šifra, s bezpečným algoritmom môže byť zlomená ak používa nedostatočne dlhý kľúč. Ako bezpečné šifry sa dnes odporúča použiť **AES-256, RSA-3072**, či **Diffie-Hellmanovu** výmenu kľúčov využívajúcu eliptické krivky s použitím krivky **P-384**. Prípadne hashovací algoritmus **SHA-384** [38].

### 2.4.4 Nepoužívajte žiadny kľúč správy donekonečna

Je nutné evidovať počet správ odoslaných pomocou určitého kľúča a vymeňte každý kľúč, hneď ako sa priblíži k bezpečnému limitu. Po odoslaní $2^{48}$ blokov AES s CBC sa pravdepodobnosť dešifrovania príliš posunie v prospech útočníka. V tomto okamihu je nutné zmeniť kľúč.

DES a 3DES vyžadujú kľúčovú zmenu už po $2^{16}$ blokoch. To je jeden z dôvodov, prečo AES má väčšiu veľkosť bloku ako staršie algoritmy DES alebo 3DES. AES umožňuje pred zmenou kľúča zašifrovať podstatne viac údajov.

## 2.5 Insecure Authorization (Nezabezpečená autorizácia)

Prísnosť autorizácie závisí na koncepte aplikácie. Napríklad aplikácie s bankovníctvom musia byť chránené oveľa viac ako napríklad aplikácie s počasím. Základná úroveň bezpečnosti však musí byť zachovaná a preto je nutné sa riadiť určitými pravidlami.

Je potreba overiť role a povolenia autentifikovaného používateľa iba pomocou informácií obsiahnutých v back-endových systémoch. Nie je možné sa spoliehať na žiadne role ani informácie o povolení, ktoré pochádzajú zo samotného mobilného zariadenia.

Back-endový kód by mal nezávisle overovať, či sa všetky prichádzajúce identifikátory spojené s požiadavkou, ktoré prichádzajú spolu s identifikáciou, zhodujú a patria k prichádzajúcej identite. Odporúčané je tiež využiť frameworky ako Oauth, prípadne viacfázovú autorizáciu či autentizáciu [13].

## 2.6 Client Code Quality (Kvalita kódu klienta)

Kvalita kódu môže byť rozhodujúca vec, hlavne ak na vývoji spolupracuje viacero ľudí. Zle napísaný, "ťažkopádny" kód môže zvyšovať riziko chyby a tým viesť k vzniku iných zraniteľností.

1. Je potrebné udržiavať konzistentné vzory kódovania, na ktorých sa všetci v organizácii, prípadne všetci zapojení vývojári zhodnú. Väčšinou bývajú vytvorené tzv. "coding policies", v ktorých bývajú udané akési štandardy, ktorými by sa vývojári mali riadiť. Popísané sú väčšinou názvy premenných, funkcií atď., aby výsledný kód mal ucelenú a jednotnú formu.

2. Kód, by mal byť čitateľný a dobre zdokumentovaný, samozrejme v prípade, že je našim cieľom obfuskácia kódu, snažíme sa o opak (viď kapitola 2.8.1).

3. Pomocou automatizácie je potrebné identifikovať pretečenia vyrovnávacej pamäte a úniky pamäte pomocou nástrojov statickej analýzy tretích strán ako Dmalloc a pod. [50].

4. Pri manuálnej statickej analýze by mal byť použitý tzv. „four eyes principle", to znamená, že kód by mal kontrolovať iný človek ako ten, ktorý ho písal aby sa predišlo chybám, ktoré si daný človek nevšimol.

5. Uprednostnenie riešení pretečenia medzipamäte a úniku pamäte pred ostatnými problémami s „kvalitou kódu“, nakoľko tento problém je náchylnejší na zneužitie a môže mať väčší bezpečnostný dopad [15].

### 2.6.1 Čomu sa pri písaní kódu vyhnúť?

#### *2.6.1.1 Strings.xml*

Základnou chybou kvality kódu je neukladanie všetkých reťazcov, ktoré sa majú zobraziť, v súbore strings.xml. Je však nutné poznamenať, že po dekompilácii sú .xml súbory čitateľné, preto sa neodporúča ukladať sem akékoľvek citlivé informácie, či informácie, ktoré by mohli byť nejakým spôsobom zneužité.
Neukladanie reťazcov do spoločného súboru spôsobuje problémy pri zdieľaní kódu, pretože vývojár je nútený navrhnúť vlastné spôsoby zobrazovania správnej verzie správy, podľa lokálneho nastavenia používateľa. Ak sú správy v súbore strings.xml, dajú sa ľahko preložiť a integrovať do aplikácie. Android OS potom bez problémov vie, ktorý zdroj reťazca sa má použiť, na základe lokálneho nastavenia, ktoré si používateľ nastavil na svojom zariadení.
Odporúča sa teda všetky reťazce používané v kóde ukladať do string.xml a následne sa v kóde na tento súbor odkazovať [24].

#### *2.6.1.2 Nezohľadnenie životného cyklu aktivity*

Akýkoľvek typ zmeny konfigurácie spôsobí zničenie a nové vytvorenie aktuálnej aktivity. Aby bolo zaistené, že prechod bude pre používateľa bezproblémový, je nutné uložiť stav, v ktorom sa nachádzala aplikácia, tesne pred zmenou konfigurácie. Po zničení môže byť znovu vytvorená tak, ako používateľ očakáva [24].
Príklad fungovania je na [55] popísaný nasledovne:
Keď je v metóde onCreate metóde pridaný parameter BundlesavedInstanceState automaticky je pri vytváraní stav uložený v savedInstanceState. Pokiaľ tento state niečo obsahuje tak je vytvorený string savedInstanceState.getString(Game_State_Key). Následne je z layoutu aplikácie získaný TextView a pomocou metódy setText môže byť nastavený stav aplikácie.

## 2.7 Code Tampering (Neoprávnená manipulácia s kódom)

Mobilná aplikácia musí byť schopná za behu zistiť, či bol kód pridaný alebo zmenený z toho, čo vie o svojej integrite, v čase kompilácie. Aplikácia musí byť schopná za behu aplikácie primerane reagovať na porušenie integrity kódu.

Aplikácia môže mať implementovanú funkciu, ktorá pri detekcii neoprávnenej manipulácie v tichosti vymaže používateľské údaje, kľúče alebo iné dôležité údaje tak, aby neupozornila útočníka, ktorý by tak pokračoval vo svojej snahe zbytočne. Aplikácie, ktoré zistili manipuláciu, môžu tiež upozorniť správcu.

V systéme Android je možné verejný kľúč použitý na podpísanie aplikácie prečítať z certifikátu aplikácie a použiť na overenie, či bola aplikácia podpísaná súkromným kľúčom vývojára. Pomocou triedy PackageManager je možné získať podpisy našej aplikácie a potom ich porovnať so správnou hodnotou. Ak niekto neoprávnene manipuloval s aplikáciou alebo ju znova podpísal, porovnanie zlyhá, čo má za následok zistenie neoprávnenej manipulácie s aplikáciou [16].

### 2.7.1 Android Root Detection

Zvyčajne sa upravovaná aplikácia spustí v rootovanom prostredí. Preto je rozumné pokúsiť sa tieto typy narušených prostredí zistiť za behu a zodpovedajúcim spôsobom na ne reagovať (správa na server alebo vypnutie). Existuje niekoľko bežných spôsobov, ako zistiť rootované zariadenie s Androidom:

1. Skontrolujte prítomnosť testovacích kľúčov (viď kapitola 4.1.1)

2. Skontrolujte, či build.prop obsahuje riadok ro.build.tags = testovacie kľúče označujúce zostavenie vývojára alebo neoficiálnu ROM

3. Skontrolujte certifikáty OTA tým, že overíte, či súbor /etc/security/otacerts.zip existuje

4. Skontrolujte niekoľko známych root packages, ktoré vzniknú po nainštalovaní APK súboru

    - com.noshufou.android.su
    - com.thirdparty.superuser
    - eu.chainfire.supersu
    - com.koushikdutta.superuser

5. Skontrolujte binárne súbory SU
    - /system/bin/su
    - /system/xbin/su
    - /sbin/su
    - /system/su
    - /system/bin/.ext/.su

6. Pokúste sa spustiť príkaz su, ak bude úspešne vykonaný, používateľ je root.

## 2.8 Reverse Engineering (Reverzné inžinierstvo)

Kapitola vychádza z odporúčaní na [19] a [25].

Na začiatok je vhodné povedať, že neexistuje žiadny 100% účinný spôsob ako zabrániť reverznému inžinierstvu. Existujú však metódy, ktoré môžu celý proces poriadne skomplikovať a odradiť tak potenciálneho útočníka.

### 2.8.1 Zásady obrany proti reverznému inžinierstvu

Firmy sa samozrejme snažia zabrániť prezradeniu ich know-how, prípadne prezradeniu iných citlivých informácií pomocou rôznych techník.

#### *2.8.1.1 Nástroje na obfuskáciu.*

**Obfuskácia:**

„Cieľom obfuskácie je zabrániť alebo aspoň čo najviac sťažiť analýzu zdrojového kódu a to tak, že sa v zdrojovom kóde vykonajú úpravy, ktoré spôsobia, že pre človeka sa takýto kód stane značne nečitateľným a to aj pre samotného autora“ [56].

Na trhu existuje rada takýchto bezplatných i komerčných nástrojov, napríklad Proguard. Rovnako však na trhu existuje aj rada rôznych deobfuskátorov, teda nástrojov, ktoré sú schopné schovaný či akýmkoľvek spôsobom “zahmlený” kód zobraziť. Ak chcete zmerať účinnosť ktoréhokoľvek nástroja na obfuskáciu, ktorý si vyberiete, skúste deobfuskovať kód pomocou nástrojov ako IDA Pro a Hopper.

#### *2.8.1.2 Umiestnenie aplikačnej logiky na server*

Ako už bolo viackrát povedané, všetko čo je umiestnené v mobilnom zariadení môže byť napadnuté. Preto je dobrou metódou umiestnenie dôležitého kódu na server a v prípade nutnosti môže byť daná funkcia zavolaná. To síce môže predstavovať nevýhodu pri veľkom počte používateľov, stále však platí, že je treba zvážiť, do akej miery je pre nás bezpečnosť kľúčová.

#### *2.8.1.3 Dôležité časti kódu píšte v jazyku C/C++*

Java je značne jednoduchšia na dekompiláciu v porovnaní s jazykmi C/C++. Taktiež jazyk Kotlin je veľmi presadzovaný zo strany Googlu. Časti kódu napísané týmito jazykmi môžu byť pridané do kompilovanej knižnice, kde kód potom môže byť rozložený do assembly kódu. Reverzné inžinierstvo veľkej knižnice z assembly kódu je extrémne časovo náročné, čo môže odradiť útočníka.

#### *2.8.1.4 Súbory .so*

Použitím NDK, kde môžete súbory natívne zapisovať ako .so sa znižuje pravdepodobnosť, že budú dekompilované na rozdiel od APK. Existujú síce nástroje schopné dekompilácie aj NPK, avšak útočník by potreboval dobré znalosti ARM procesorov a jazyku assembler, aby nejakým spôsobom zneužil získané dáta.

#### *2.8.1.5 Viacúrovňové zabezpečenie*

Je veľmi dôležité používať viac úrovní zabezpečenia. Napríklad zabrániť vstupu do zariadenia pamätaním si informácie o zariadení, prípadne identifikáciou pokusov prihlásenia z neznámeho zariadenia atď. Je však potrebné ošetrenie voči mazaniu pamäte, nakoľko týmto spôsobom je možné resetovať pokusy o prihlásenie. Čím viac úrovní zabezpečenia bude použité, tým bude aplikácia ťažším cieľom pre útočníka. Je však nutné vziať do úvahy aj použiteľnosť pre bežného používateľa, ktorého by mohla príliš veľká bezpečnosť a neustále overovanie údajov odradiť od používania (viď kapitola 5.4).

## 2.9 Extraneous Functionality (Nadbytočná funkcionalita)

Najlepším spôsobom, ako zabrániť tejto chybe, je vykonať manuálnu kontrolu kódu pomocou odborníkov na zabezpečenie alebo odborníkov na danú tému, ktorí majú s týmto problémom skúsenosti. Mali by robiť nasledovné:

1. Preskúmať konfiguračné nastavenia aplikácie
2. Overiť, či vo finálnej produkčnej zostave aplikácie nie je zahrnutý testovací kód.
3. Preskúmať všetky koncové body API, ku ktorým pristupuje mobilná aplikácia, a overiť, či sú tieto koncové body dobre zdokumentované a zabezpečené.
4. Preskúmať všetky hlásenia, ktoré aplikácia môže poskytnúť, aby ste sa uistili, že sa v nich nevyskytujú žiadne potenciálne zneužiteľné informácie.

Všetka funkcionalita, ktorá slúžila na testovanie, funkcionalita, ktorá nie je využívaná či obsahuje bezpečnostné vady, by mala byť odstránená, prípadne nahradená bezpečnou formou [20].

Rovnako je tiež treba ošetriť Log Leakage. Detailnejší popis je v kapitolách 3.4.3 a 4.4.2.

# II. PRAKTICKÁ ČASŤ

# 3 SÚBOR PRAVIDIEL PRE BEZPEČNÚ IMPLEMENTÁCIU

V tejto kapitole sa nachádza popis pravidiel, ktoré sú najdôležitejšie pre všeobecnú bezpečnosť aplikácií. Pravidlá vyplynuli buď zo skúseností z implementácie bezpečnostných postupov, znalostí získaných pri zaoberaní sa témou bezpečného vývoja Androidových aplikácií, alebo boli po analýze zraniteľností vybrané z pravidiel získaných v rámci rešeršnej prípravy.

## 3.1 Implementovať detekciu rootu

Ako bolo spomenuté v teoretickej časti, telefón, ktorý má privilégiá najvyššieho používateľa, teda rootu môže spôsobovať problémy vo viacerých odvetviach a poskytuje radu možností pre útoky na aplikáciu. Taktiež môže pomôcť odhaľovať radu ďalších zraniteľností. Práve to je dôvod, prečo by mala mať každá aplikácia, ktorá nejakým spôsobom spracováva alebo ukladá citlivé dáta na lokálne úložisko telefónu implementovanú ochranu proti rootu. Existuje mnoho nástrojov, pomocou ktorých je možné získať root prístup do telefónu. Preto je veľmi dôležité nie len implementovať správnu detekciu ale neustále sa informovať o nových možnostiach a aktualizovať kód.

## 3.2 HTTP je nebezpečné

Pri implementácii internetového spojenia z akýmkoľvek účelom je nesmierne dôležité používať šifrované spojenie. V kapitole 4.2.1 je ukázané, ako jednoducho je možné odchytávať komunikáciu a získať tak prístup k citlivým dátam posielaným cez HTTP. Nie je pravda, že použitím HTTPS je vytvorená absolútna bezpečnosť spojenia, avšak použitie tohto štandardu je základom bezpečnosti každej implementácie internetového spojenia, pretože východzie spojenie v Androide je realizované pomocou HTTP. Je však potrebné dbať na správne ošetrenie a korektnú implementáciu a to aj z dôvodu, že ak útočník zistí, že aplikácia používa nešifrované spojenie HTTP okamžite pochopí, že aplikácia nie je bezpečná a nebola vyvíjaná dostatočne skúseným vývojárom a začne v nej hľadať ďalšie zraniteľnosti. HTTP by teda v dnešnej dobe už nemalo byť vôbec implementované a používané.

## 3.3 Správne šifrovanie je kľúčom k bezpečnosti dát

Všetky citlivé dáta, ku ktorým by sa nejakým spôsobom mohol útočník dostať je potrebné zabezpečiť šifrovaním a znemožniť tak odcudzenie alebo zneužitie informácií v prípade, že by sa k nim nejaký útočník dostal.

Samozrejme s nástupom kvantových počítačov bude väčšina moderných šifier nebezpečná, takže bude treba hľadať také šifrovacie systémy, ktoré by takýmto strojom odolali. Ako nádej sa v túto chvíľu javí Lattice-based cryptography.

Pri implementácii šifrovania je potrebné zistiť, ktoré šifry sú bezpečné a to v závislosti na citlivosti dát, s ktorými aplikácia pracuje. Všeobecne sa však odporúča použiť jeden z najrozšírenejších algoritmov AES s dĺžkou kľúča 256 bitov, ktorý je schopný bezpečne šifrovať aj informácie až do úrovne TOP SECRET [38].

Zároveň je treba rozlišovať, ktoré algoritmy je vhodné použiť na šifrovanie konkrétneho typu dát. Napríklad na veľké dátové súbory by mala byť použitá symetrická kryptografia. Naopak pri šifrovaní súborov so symetrickými kľúčmi alebo na zabezpečenie komunikácie by sa mala používať asymetrická kryptografia.

Rovnako dôležité je však aj správne generovanie kľúča a práca s ním. Pri manipulácii s kľúčom a jeho ukladaní je nutné dodržiavať pravidlá a nevystavovať sa riziku prezradenia. Ako náhle je prezradený symetrický kľúč (symetrická kryptografia) alebo privátny kľúč (asymetrická kryptografia) tak sú všetky šifrované dáta kompromitované.

V neposlednom rade je veľmi dôležité nepokúšať sa vymýšľať a implementovať vlastné šifry a kryptografické primitívy, pretože nebudú dostatočne otestované a môžu sa v nich nachádzať zraniteľnosti či slabé miesta. Z hľadiska času aj bezpečnosti je doporučené používať existujúce a overené kryptografické metódy.

## 3.4 Statická a dynamická analýza kódu

### 3.4.1 Statická analýza

Statická analýza kódu sa nezaoberá chovaním testovanej aplikácie. To znamená, že aplikácia nie je pri testovaní spúšťaná. Naopak sa zaoberá dvoma hlavnými oblasťami, ktorými sú zdrojové kódy (aplikačná logika) a neprogramovými zdrojmi, ako sú napríklad reťazce (user interface strings), grafiku a XML layouty. Hlavnou úlohou statickej analýzy je zabezpečiť,

aby sa v kóde nenachádzali citlivé informácie, nadbytočné, či irelevantné funkcionality alebo bezpečnostné chyby.

### 3.4.2 Dynamická analýza

Dynamická analýza sa naopak nezaoberá kódom, ale chovaním aplikácie. Aplikácia je spustená v kontrolovanom prostredí (napr. zákazkové emulátory mobilných zariadení). Testuje sa napríklad reakcia na rôzne vstupy, odpovede aplikácii, funkcionality, odolnosť voči útokom/ nebezpečným vstupom a pod.

### 3.4.3 Použitie v praxi

Je dôležité klásť dôraz na správny kód a jeho požadované správanie pri behu. V teoretickej časti sú popisované 2 zraniteľnosti, ktoré vychádzajú z chybného kódu a práve tomu je možné zabrániť pozornou kontrolou a testovaním kódu. Pri manuálnom kontrolovaní kódu statickou analýzou je odporúčané používať tzv. 4-eyes-principle, teda praktiku, pri ktorej by si mal kód sám po sebe skontrolovať vývojár a následne ešte niekto iný, aby sa zabránilo tomu, že v kóde by sa vyskytovala chyba, ktorú si vývojár nevšimol, alebo by tam bola chyba, spôsobená neznalosťou vývojára v určitej oblasti. V oboch prípadoch by sa použitím tejto metódy mohlo predísť chybám či nedostatkom.

Ďalšou odporúčanou metódou je používať existujúce nástroje statickej analýzy, ktoré môžu pomôcť detegovať chyby v kóde, ktoré si vývojár nevšimol. Je možné napríklad vyhľadávať slová, ktoré sa vyskytujú pri manipulácii s citlivými údajmi (viď kapitola 4.4.1.3) alebo regulárne výrazy.

Správnou dynamickou analýzou by sa dá zase vyhnúť chybám ako napríklad pretečeniam pamätí či iným problémom pri behu aplikácie, ktoré by mohli viesť k vzniku zraniteľností alebo poskytovať útočníkom priestor pre vykonávanie útokov a činnosť mobilných malwarov. Rovnako je treba klásť dôležitosť na hlásenia, pretože sa môže stať, že v hláseniach budú prezradené informácie, ktoré môžu napovedať útočníkovi a poskytnúť mu informácie o fungovaní aplikačnej logiky. To môže byť vysvetlené na príklade, kedy sa útočník pokúša prihlásiť do aplikácie. Ak mu pri zadaní náhodných údajov aplikácia poskytne chybové hlásenie “Heslo je nesprávne!”, vie, že zadané meno je správne. Hlásenia by teda mali byť neutrálneho charakteru a nemali by prezrádzať detaily. Napríklad “Užívateľské údaje sú nesprávne!”. Toto by však znova malo podliehať kontrole kódu a testovaniu.

Podobný problém ako hlásenia môžu byť spôsobovať aj logy. Logovanie je metóda ladenia aplikácie, kedy si vývojár vypisuje do určitého súboru informácie, a zisťuje tak, či aplikácia funguje správne. V prípade, že sú tieto logy nechané v produkčnej verzii aplikácie ktorá je prístupná koncovým používateľom, môžu spôsobiť únik citlivých informácií tzv. log leakage. Je preto veľmi dôležité, aby logy boli odstránené pred produkčným nasadením aplikácie a to vždy, pokiaľ nie je vyložene nejaký závažný dôvod na to, aby boli logy súčasťou aplikácie.

## 3.5 Ošetrenie vstupov (SQL Injection)

Veľká časť útokov je vedená práve cez užívateľské vstupy. Hlavne injection útoky, proti ktorým Android OS neposkytuje žiadnu preddefinovanú ochranu (viď kapitola 5.2). Je teda potrebné tieto vstupy ošetrovať naozaj dôkladne, avšak toto závisí len na vývojárovi. Je to však nutné aby bolo znemožnené útočníkovi akýmkoľvek spôsobom vkladať nebezpečné vstupy alebo získať prístup k údajom takýmto spôsobom.

Ak je napríklad od používateľa požadované vloženie emailu, textové pole musí akceptovať iba písmená bez diakritiky, čísla, znak bodky a zavináč. Všetky ostatné znaky by mali byť označené za nevyhovujúce a predanie zadaného reťazca ďalej do PHP skriptu alebo SQL databázy znemožnený. Existuje viacero metód, ktoré je možné použiť na filtráciu vstupov. Vždy sa však odporúča filtrovanie vstupov už priamo na klientskej aplikácii, nakoľko môže byť použitá lokálna databáza a bez potrebného filtrovania by sa mohol útočník dostať k dátam, alebo by mohol útokom pozmeniť chovanie aplikácie. Samozrejme, pokiaľ je možné použiť viacfázové ošetrenie, napríklad v klientskej aplikácii a potom na serveri, určite je treba ho implementovať a nespoliehať sa len na jednu z týchto overovacích metód.

# 4 IMPLEMENTÁCIA KOREKTNÝCH BEZPEČNOSTNÝCH POSTUPOV

V tejto kapitole je popisovaná praktická implementácia korektných postupov. Pre overenie získaných znalostí bolo vhodné, vytvoriť aj zraniteľnú aplikáciu, na ktorú môžu byť simulované útoky a následne rovnaké útoky testované aj na zabezpečenej aplikácii. Výsledky útokov potom môžu byť porovnané s tým je možné overiť funkčnosť navrhnutých bezpečnostných pravidiel.

**Zraniteľná aplikácia** – Pomôže lepšie pochopiť zraniteľnosti a útoky, ktoré sú smerované na tieto zraniteľnosti. Zároveň zlepší prehľad o tom, ako je možné aplikácie napádať. Lepšie pochopenie zraniteľností pomôže budovať lepšie ochranné mechanizmy.

**Zabezpečená Aplikácia** – Obsahuje korektné bezpečnostné mechanizmy, ktorých funkčnosť je možné prakticky overiť. Výsledky útokov prevádzaných na obe aplikácie je možné v niektorých prípadoch porovnať, a získať tak prehľad o tom, či je daná bezpečnostná metóda účinná alebo nie.

## 4.1 Detekcia Rootu

Nebezpečnosť rootu je popisovaná už v teoretickej časti práce. Ďalej bude vysvetlené, ako je môže užívateľská aplikácia detegovať root mobilného zariadenia, na ktorom je spúšťaná. Existuje celá rada webových stránok popisujúcich root mobilných zariadení, pričom je možné povedať, že s každou novšou verziou Android OS je root telefónu komplikovanejší, pretože samotní vývojári operačného systému chcú tejto zraniteľnosti zabrániť. Existujú 2 základné postupy a to, že je potreba stiahnuť špeciálny APK súbor, pomocou ktorého je možné získať rolu „super užívateľa“, alebo je možné telefón „rootnuť“ prostredníctvom programu z počítača cez USB rozhranie (USB debugging).

Na simuláciu rootovania a vývoj však nie je treba ani vlastniť telefón s operačným systémom Android. Dnešné technológie sú natoľko pokročilé, že je možné takýto prístup simulovať. Spočiatku sa teda ako jasný krok javilo získať prístup root na Emulátore, ktorý je súčasťou vývojového prostredia Android studio. Praktickými experimentami, ktoré boli prevedené v bakalárskej práci však došlo k zisteniu, že nie je možné získať úplný root prístup k emulátoru, nakoľko sú možnosti značne obmedzené a tak sa táto možnosť ukázala ako nesprávna. Po rešerši ďalších možností, sa však podarilo získať informácie o softwari s názvom **Nox App player** [26], ktorý je taktiež emulátorom, ktorý môže bežať na osobnom počítači

a dokáže simulovať telefón s operačným systémom Android. Navyše tento softwar obsahuje funkcionalitu, ktorá umožňuje spustiť emulátor s prístupom root jedným kliknutím a je umožnené tento emulátor prepojiť s vývojovým prostredím Android studio a inštalovať naň vyvíjanú aplikáciu. Preto sa táto možnosť veľmi osvedčila.

Pri tvorbe kontrolného kódu je postupované podľa doporučených postupov. Prvým krokom je skontrolovanie build tags na výskyt testovacích kľúčov.

### 4.1.1 Building Tags

"Kontrola building tagu je dôležitá z toho dôvodu, že jadro operačného systému je pri kompilácii podpísané. Release-keys znamená, že bolo podpísané oficiálnym kľúčom od oficiálneho vývojára. Test-keys znamenajú, že bolo podpísané kľúčom patriacim vývojárom tretej strany. Ak je teda nájdený výraz test-key znamená to, že s jadrom bolo manipulované" [27].

```
private boolean detectTestKeys() {
    String buildTags = android.os.Build.TAGS;
    return buildTags != null && buildTags.contains("test-keys");
}
```

Obrázok 2. Kontrola building tags

### 4.1.2 OTA certificate

OTA znamená Over-the-Air, a je to certifikát, ktorý umožňuje aktualizácie prostredníctvom bezdrôtového pripojenia. Android je predvolene aktualizovaný prostredníctvom Wi-Fi pomocou verejných certifikátov od spoločnosti Google. Ak teda tieto certifikáty nie sú prítomné, zvyčajne to znamená, že je nainštalovaná upravená ROM, ktorá sa aktualizuje inými spôsobmi. Preto je potrebné overiť, či tieto certifikáty existujú. Zvyčajne sú umiestnené v zložke /etc/security [28].

```
public static boolean checkOtaCert() {
    try {
        File file = new File( pathname: "/etc/security/otacerts.zip");
        if (file.exists()) {
            return true;
        }
    } catch (Throwable e1) {
        e1.printStackTrace();
    }
    return false;
}
```

Obrázok 3. Kontrola OTA certifikátov

### 4.1.3 Známe root Apk súbory

V úvode praktickej časti bolo spomenuté, že root telefónu je väčšinou realizovaný pomocou špecializovaného softwaru, ktorý si užívateľ stiahne ako .apk súbor. Preto bolo nutné vykonať prieskum najpoužívanejších .apk rootovacích súborov a otestovať, či sa v súborovom systéme zariadenia nenachádzajú. Medzi najpoužívanejšie apk súbory patria:

1. Magdisk (com.topjohnwu.magisk)
2. Dr.Fone – Root (com.wondershare.drfone)
3. KingoRoot (kingoroot.supersu)
4. OneClickRoot (bitbot.root.click.one.em)
5. RescueRoot (com.hexamob.howtoroot)
6. KingRoot (com.kingroot.kinguser)
7. SuperSU Root (eu.chainfire.supersu)
8. Framaroot (com.mgyun.shua.su)
9. Root Checker (com.joeykrim.rootcheck) [29]

Prehľadávanie je vykonávané v adresároch internej aj externej pamäti emulátoru, hlavným cieľom je však adresár /data/data, ktorý sa schválne prehľadáva ako prvý. V týchto adresároch sú hľadané názvy adresárov, s cieľom zistenia výskytu týchto “packages”, ktoré vzniknú po nainštalovaní .apk súboru. Taktiež niektoré z týchto programov sú platené, preto boli odskúšané len tie verzie, ktoré sú zadarmo. V obrázkoch nižšie je ukážka kódu, kde je cyklus, v ktorom sa vyhľadávajú spomínané známe .apk súbory (viď Obrázok 4). Na ďalšom obrázku je zoznam všetkých ciest, ktoré sú prehľadávané (viď Obrázok 5).

```
private boolean checkKnownApk() {
    for (String filename : rootApks) {
        for (String path : binaryPaths) {
            File f = new File(path, filename);
            boolean fileExists = f.exists();
            if (fileExists) {
                Log.v( tag: "UTBZLIN", msg: "APK"+ f.getAbsolutePath());
                return true;
            }
        }
    }
    return false;
}
private String[] rootApks = {
        "com.topjohnwu.magisk", //magisk
        "kingoroot.supersu",    //kingoroot
        "com.wondershare.drfone", //dr.fone root
        "com.kingoapp.apk",
        "com.yellowes.su",
        "com.mgyun.shua.su", //iRoot
        "eu.chainfire.supersu", //supersu
        "com.koushikdutta.superuser", //Superuser
        "bitbot.root.click.one.em",   //oneclickroot
        "com.kingroot.kinguser", //kingroot
        "com.joeykrim.rootcheck", //rootchecker
        "com.hexamob.howtoroot"  // RootAllDevices (rescueroot)
};
```

Obrázok 4. Kontrola adresárov nainštalovaných APK súborov

```
private String[] binaryPaths = {
        "/data/data",
        "/data/user/0/",
        "/storage/emulated/0/Android/data/",
        "/sdcard/Android/data/",
        "/data/local/",
        "/data/local/bin/",
        "/data/local/xbin/",
        "/sbin/",
        "/su/bin/",
        "/system/bin/",
        "/system/bin/.ext/",
        "/system/bin/failsafe/",
        "/system/sd/xbin/",
        "/system/usr/we-need-root/",
        "/system/xbin/",
        "/cache",
        "/data",
        "/dev"

};
```

Obrázok 5. Prehľadávané cesty

### 4.1.4 Kontrola binárnych súborov SU a Busyboxu

#### *4.1.4.1 Binárny súbor SU*

Binárny súbor SU znamená, že sa v zariadení nachádza Super User. Tento súbor sa inštaluje pri roote telefónu a je teda potrebné otestovať, či sa niekde v telefóne tento súbor nachádza. Väčšinou sa tento súbor nachádza v týchto adresároch, avšak pre istotu sú kontrolované všetky spomenuté adresáre v poli *binaryPaths* (viď. Obrázok 5.)

- /system/bin/su
- /system/xbin/su
- /sbin/su
- /system/su
- /system/bin/.ext/.su

Vytvorená funkcia pomocou cyklu prechádza všetky spomenuté cesty, a v prípade, že je nájdená aspoň jedna zhodu, nepokračuje sa ďalej, ale návratová hodnota je nastavená na true a neskôr vyhodnotená. V prípade, že funkcia nájde takýto súbor, je telefón s najväčšou pravdepodobnosťou skutočne „rootnutý“, preto je jej pridelená veľká závažnosť (viď kapitola 4.1.7).

#### *4.1.4.2 Busybox*

Busybox programom alebo užívateľom umožňuje vykonávať v telefóne akcie pomocou príkazov OS Linux. Busybox poskytuje telefónu funkčnosť, ktorú bez neho operačný systém Android nemá. Mnoho programov, najmä rootovacích, vyžaduje na vykonávanie funkcií programu busybox. Bez nainštalovaného busyboxu telefón poskytuje obmedzenú funkcionalitu. Práve preto je nutné kontrolovať aj prítomnosť busyboxu [30].

```java
private boolean checkForBusyBoxBinary() {
    return checkForBinary( filename: "busybox");
}

private boolean checkForSuBinary() {
    return checkForBinary( filename: "su");
}

/**
 * @param filename - check for this existence of this
 * @return true if exists
 */
private boolean checkForBinary(String filename) {
    for (String path : binaryPaths) {
        File f = new File(path, filename);
        boolean fileExists = f.exists();
        if (fileExists) {
            return true;
        }
    }
    return false;
}
```

Obrázok 6. Kontrola výskytu su a busybox súboru

### 4.1.5 Kontrola USB debugging

Niektoré rootovacie programy umožňujú rootovať zariadenie prostredníctvom USB. Aby však bolo možné vykonávať podobné úkony, je nutné v developerských nastaveniach mobilného telefónu povoliť USB debugging. Samozrejme detekcia tohto nastavenia vôbec nemusí znamenať, že telefón je naozaj rootnutý. Detekcia slúži iba na upozornenie, pretože bežný užívateľ nemá potrebu akokoľvek zasahovať do zariadenia týmto spôsobom. Z toho vyplýva, že ak je USB debugging povolený, užívateľ vykonáva s telefónom určité špecializovanejšie úkony ako bežné používanie a je oveľa vyššia pravdepodobnosť, že práve takéto zariadenie môže byť rootované.

Pri aplikáciách extrémne citlivých na zabezpečenie ako sú napríklad bankové aplikácie, je odporúčané zabrániť spusteniu aplikácie aj pri samostatnej detekcii USB debuggingu, nakoľko aj bez rootu zariadenia je možné pomocou USB odchytávať citlivé informácie.

```java
private boolean UsbDebugging(){
    return Settings.Secure.getInt(this.getContentResolver(), Settings.Global.ADB_ENABLED,  def: 0) == 1;
}
```

Obrázok 7. Kontrola USB debugging

### 4.1.6 Skúška vykonávania príkazov

Ďalšou metódou overenia rootu je skúška vykonávania príkazov, ktoré môže vykonať iba root. Ak teda je v kóde zadaný príkaz na vykonanie „su“ príkazu a telefón nie je rootovaný, príkaz sa nepodarí vykonať. Naopak ak je telefón rootovaný, príkaz sa vykoná. Túto funkcionalitu je potrebná zabaliť do try-catch bloku, čo zabráni prípadnému pádu aplikácie v prípade, že sa príkaz nepodarí vykonať. V závislosti na úspešnosti vykonania príkazu je potom vrátená hodnota true alebo false, s ktorou sa pracuje pri vyhodnocovaní.

```
protected boolean ExecuteCommands() {
    try{
        Process su = Runtime.getRuntime().exec( command: "su");
        su.waitFor();
        return true;
    }catch(IOException | InterruptedException e){
         return false;
    }
}
```

Obrázok 8. Ukážka funkcie testujúcej vykonávanie príkazov

Tento príkaz je možné skúsiť vykonať na zariadení aj prostredníctvom konzoly. U prvého zadania bol zapnutý root, a je vidno, že sa príkaz vykonal. Naopak pri druhom príkaze je root vypnutý a po zadaní príkazu je zobrazené chybové hlásenie: „su: not found“.

```
C:\Users\Lenovo_T430_W10P\AppData\Local\Android\Sdk\platfo
rm-tools>adb shell "su"

C:\Users\Lenovo_T430_W10P\AppData\Local\Android\Sdk\platfo
rm-tools>adb shell "su"
/system/bin/sh: su: not found
```

Obrázok 9. Test rootu vykonaním príkazu „su“

### 4.1.7 Vyhodnocovacia logika

Aby bolo možné určiť, či sa jedná o rootované zariadenie, je potrebné spracovať výsledky popisovaných funkcií a vyhodnotiť ich. V aplikácii je pre to vytvorený bodovací systém, kde je prideľovaný určitý počet bodov každej funkcii, ktorá zachytí nejaké podozrenie. Body sú prideľované v závislosti na tom, ako veľmi je detegovaná skutočnosť závažná.

```
protected int GetSuspiciousScore(int suspiciousScore){
    if(CheckKnownApk()) {
        suspiciousScore += 5;
        if (suspiciousScore >= 5) return suspiciousScore;
    }
    if(CheckForBusyBoxBinary()){
        suspiciousScore+=5;
        if (suspiciousScore >= 5) return suspiciousScore;
    }
    if(CheckForSuBinary()){...}
    if(ExecuteCommands()){
        suspiciousScore+=5;
        if (suspiciousScore >= 5) return suspiciousScore;
    }
    if(UsbDebugging()){
        suspiciousScore+=2;
        if (suspiciousScore >= 5) return suspiciousScore;
    }
    if(!CheckOtaCert()){
        suspiciousScore+=3;
        if (suspiciousScore >= 5) return suspiciousScore;
    }
    if(DetectTestKeys()){
        suspiciousScore+=3;
        if (suspiciousScore >= 5) return suspiciousScore;
    }
    return suspiciousScore;
```

Obrázok 10. Prideľovanie bodov za nálezy

V prípade, že je dosiahnutý určitý počet bodov, je zariadenie identifikované ako rootované a nemusí sa ani pokračovať v dokončovaní ostatných funkcií. Tieto bodové hodnoty sú nastavené tak, aby aj jedna závažnejšia detekcia signalizovala root, zatiaľ čo u menej závažných sú potreba aspoň dve detekcie, prípadne kombinácia menej závažnej a závažnej detekcie. Ako už bolo spomenuté, hodnoty je potreba prispôsobiť podľa požiadaviek na zabezpečenie aplikácií, napr. USB debugging je neprípustný u bankových aplikácií.

Po pridelení hodnôt funkciám je výsledok funkcie porovnaný s povolenou hodnotou a pokiaľ ju prekračuje, je zobrazené chybové hlásenie a aplikácia je zatvorená. Použitím tejto detekcie hneď pri zapnutí aplikácie v metóde onCreate, je spôsobené, že aplikáciu nie je možné spustiť na rootovanom telefóne (viď obr. 11, System.exit(0);).

```
protected void RootTest(){
    int suspiciousScore = 0;
    suspiciousScore = GetSuspiciousScore(suspiciousScore);

    if(suspiciousScore >=5) {
        new AlertDialog.Builder( context: Login.this)
                .setTitle("Rooted Device! ")
                .setMessage("It looks your device is rooted!" +
                "\nThis app can't run on such a device! Contact your official device" +
                " service to get more information.")
                .setCancelable(false)
                .setPositiveButton( text: "OK", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        finish();
                        System.exit( status: 0);
                    }
                }).show();
    }
}
```

Obrázok 11. Zobrazenie chybového hlásenia o roote zariadenia

## 4.2 HTTP je nebezpečné

V teoretickej časti bol spomenutý Man-In-The-Middle (MITM) útok. Teoretickým základom je, že sa útočník dostane medzi odosielateľa a prijímateľa dátového toku a je tak schopný vidieť celú komunikáciu, ba dokonca aj pozmeniť odoslanú informáciu a bez vedomia odosielateľa ju preposlať ďalej.

Na prenos informácií z klienta na servera sa používajú rôzne komunikačné protokoly. Najskôr sa používal hlavne HTTP. Tento protokol prenáša dáta vo forme tzv. plain textu, teda textu, ktorý nie je ničím šifrovaný a požiadavka je posielaná prostredníctvom siete internet na server. Časom sa však ukázalo, že takáto komunikácia je extrémne nebezpečná, pretože s potrebným softwarom je jednoduché vidieť celú komunikáciu medzi serverom a klientom. Bohužiaľ, aj v dnešnej dobe ešte stále existuje veľa aplikácií či stránok, využívajúcich takýto typ komunikácie.

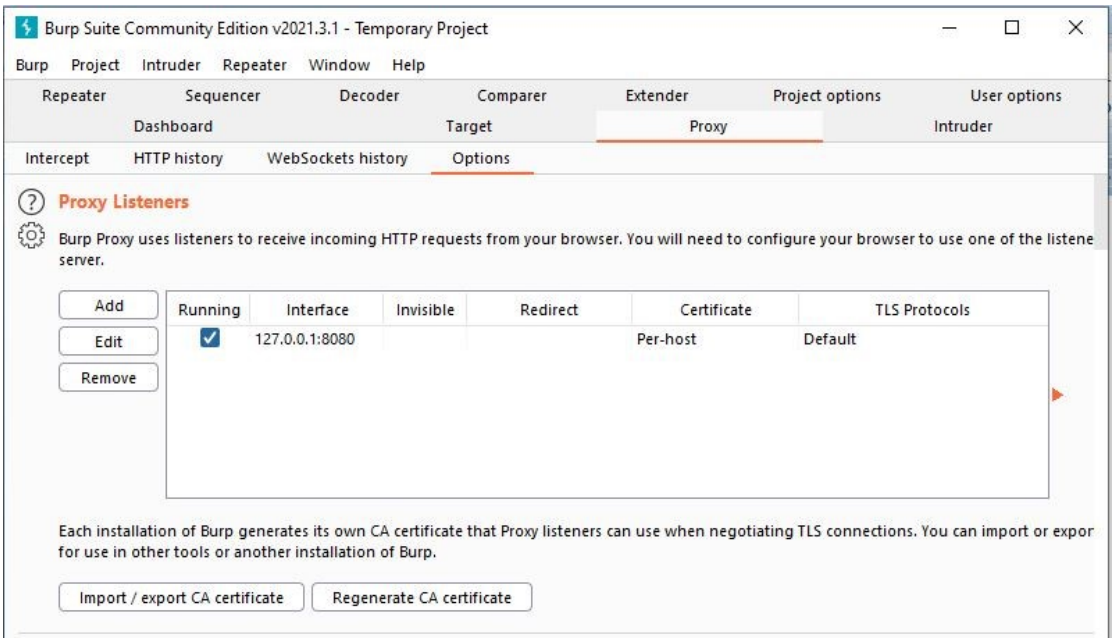
### 4.2.1 Odchytávanie HTTP komunikácie

Na simuláciu útoku na takúto komunikáciu je použitý software Burp Suite, v ktorom je nastavený proxy server [31].

**Proxy server** – Zohráva v internetovej komunikácii úlohu „prostredníka“ teda prijíma žiadosti a preposiela ich ďalej na server a rovnako tiež prijíma odpovede serveru a preposiela ich smerom k užívateľovi. Proxy servery taktiež bývajú využívané ako určitá forma ochrany, nakoľko pravá IP adresa je „skrytá“ za IP adresou proxy serveru. Prípadne na prístup k obsahu, ktorý nedovoľuje určitá krajina. V prípade, že obeť používa nezabezpečenú komunikáciu, útočník má prístup ku všetkým citlivým údajom.

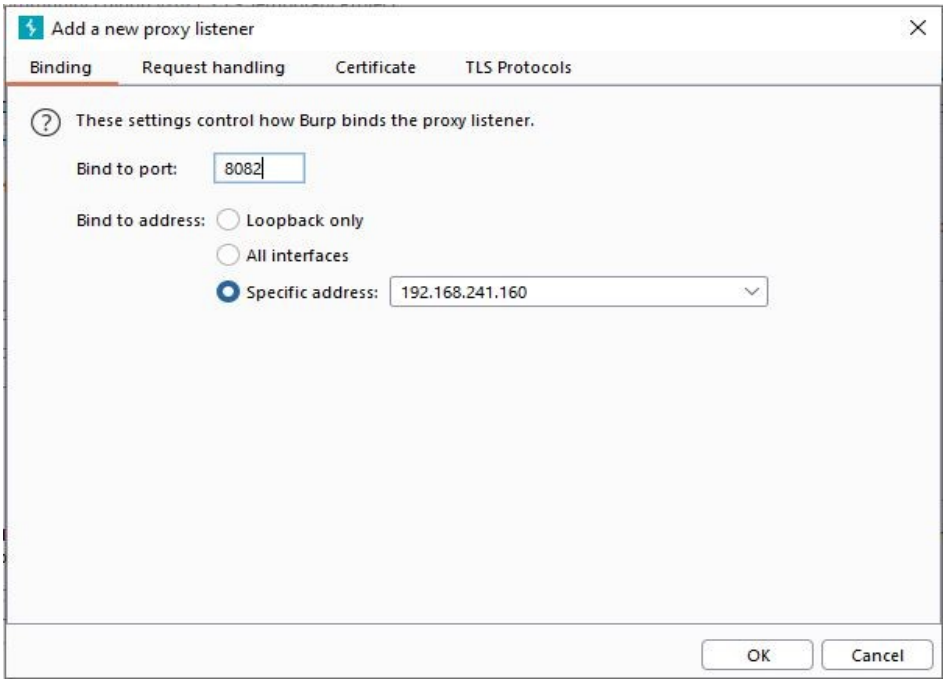
Proxy server je nastavený v programe Burp Suite následovne:

1. Po spustení programu je potreba vyhľadať záložku Proxy a v nej vybrať Options



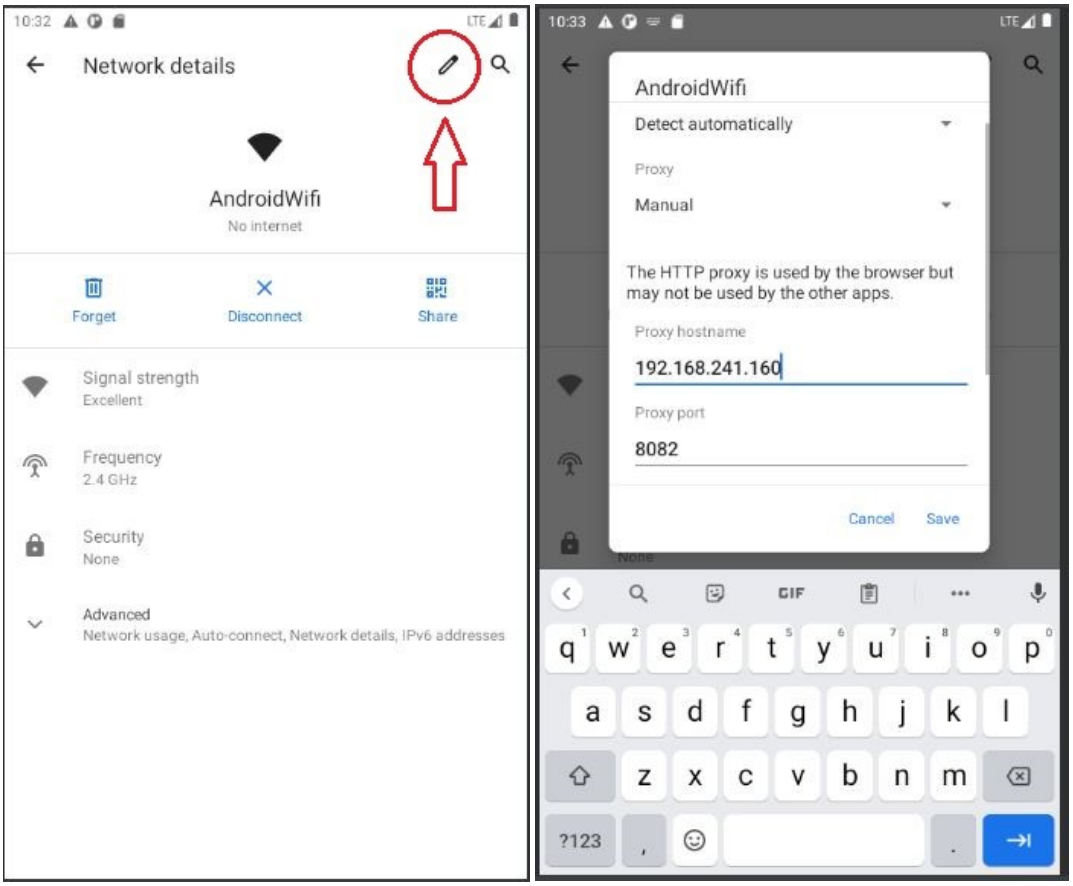

Obrázok 12. BurpSuite>Proxy>Options

2. Následne pomocou tlačidla *Add* môže byť pridaná IP adresu PC v lokálnej sieti, pretože v tomto prípade sa PC chová ako proxy server. Ďalej je potreba nastaviť odchytávanie premávky na určitom porte. Na stránkach spoločnosti firmy Portswigger (tvorca Burp Suite) [31] je odporúčané použiť port 8082, nakoľko nie je veľmi využívaný v normálnej internetovej komunikácii a preto by na ňom mala byť odchytávaná len požadovaná komunikácia posielaná cez tento port. Môže však byť využitý hociktorý iný port.



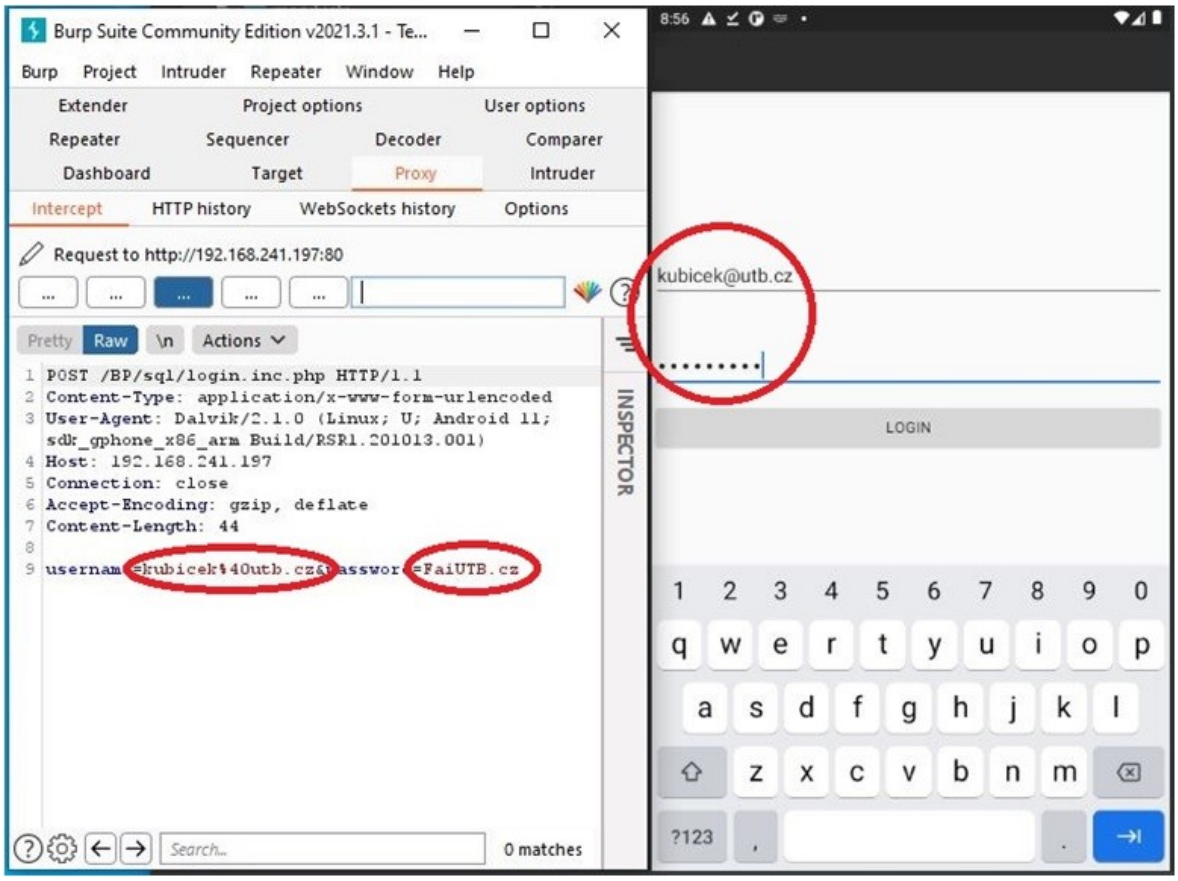

Obrázok 13. Nastavenie IP adresy a portu proxy serveru

3. Následne je potreba urobiť simuláciu útoku, ktorým je pozmenené posielanie dát nie priamo na stránku alebo server, ktorú obeť zadala, ale na proxy server. Proxy server je nastavený na zariadení Android. Pri práci boli použité emulátory dostupné priamo vo vývojovom prostredí Android Studio. V emulátore je možné sa dostať pomocou nastavení do nastavení Wi-Fi, kde po rozkliknutí siete, na ktorú je zariadenie pripojené je vpravo hore možnosť úpravy (ikonka ceruzky na Android 11 – viď obrázok 14). Po kliknutí sa je zobrazené menu, v ktorom je nutné vyplniť IP adresu proxy serveru (teda znova rovnakú adresu ako v programe Burp Suite) a port.




Obrázok 14. a 15. Nastavenie proxy serveru na Android telefóne

4. Po úspešnom nastavení a uložení je spojenie s proxy serverom hotové. Funkčnosť je možné otestovať napríklad v prehliadači. Po napísaní prihlasovacích údajov a prihlásení sa do aplikácie je žiadosť s prihlasovacími údajmi posielaná na server, kde je vidno, že program Burp Suite poskytuje informácie o tom, o aký typ metódy sa jedná, aký je typ obsahu tejto žiadosti, informácie o telefóne a aj to najdôležitejšie, prihlasovacie údaje. Tie môžeme dokonca pred preposlaním na server zmeniť.




Obrázok 16. Odchytenie prihlasovacích údajov

### 4.2.2 Nesprávna implementácia

Popri implementácii zraniteľnej aplikácie bol vytvorený aj lokálny server pomocou programu XAMPP [43], ktorý obsahuje SQL databázu a nie je súčasťou mobilnej aplikácie. Do tejto databázy boli vložené údaje používateľa, pomocou ktorých sa môže prihlásiť. Spojenie je realizované pomocou triedy *HttpUrlConnection*. Volaním *URL.openConnection()* sa potom vytvorí spojenie, v ktorom sú dáta posielané metódou POST.

Manipulácia s dátami a ich predanie serveru je zaistené PHP skriptom, ktorý vytvára dotaz na databázu a overuje, či boli prihlasovacie údaje správne. Táto schéma funguje bez problémov, avšak ako bolo demonštrované, problém je v bezpečnosti. Táto implementácia je čo sa týka bezpečnosti chybná.

### 4.2.3 Bezpečné prevedenie

“V prípade, že by sme realizovali spojenie rovnako, je potrebné miesto triedy *HttpUrlConnection* použiť bezpečný HTTPS variant a vytvoriť komunikáciu takýmto spôsobom“ [32].

```
//bezpecna varianta je pouzitie SSL spojenia
  URL url = new URL("https://www.fai.utb.cz");
  HttpsURLConnection urlConnection = (HttpsURLConnection) url.openConnection();
  urlConnection.connect();
  InputStream in = urlConnection.getInputStream();
```

Obrázok 17. Bezpečné spojenie pomocou HTTPS

Bezpečná aplikácia je ale implementovaná úplne odlišným spôsobom a na vytvorenie databázy je použitá Firebase od spoločnosti Google. Je to bezpečný variant, nakoľko nie je potrebné realizovať vlastné spojenie prostredníctvom HTTPS protokolu a riskovať tak vznik zraniteľností a zároveň Firebase obsahuje triedy, ktoré je možné použiť pri práci s databázou, či na uľahčiť procesu vývoja a hlavne zabezpečenie, že výsledná aplikácia splňuje bezpečnostné požiadavky.

Ďalším odporúčaním, ktoré uvádza priamo stránka developer.android.com je: “Unless you intend to send data from your app to a different app that you don't own, you should explicitly disallow other developers' apps from accessing the ContentProvider objects that your app contains. This setting is particularly important if your app can be installed on devices running Android 4.1.1 (API level 16) or lower, as the android:exported attribute of the <provider> element is true by default on those versions of Android” [40].

```
<provider
    android:name="com.google.firebase.provider.FirebaseInitProvider"
    android:authorities="com.example.securecodes"
    android:exported="false" />
```

Obrázok 18. Zakázanie prístupu k objektom ContentProvider

#### *4.2.3.1 FireBase*

*FirebaseAuth.getInstance();*

*FirebaseAuth* je trieda, kde volanie metódy getInstance vráti inštanciu tejto triedy zodpovedajúcu predvolenej inštancii FirebaseApp. Tým sa realizuje spojenie s cloudovou databázou.

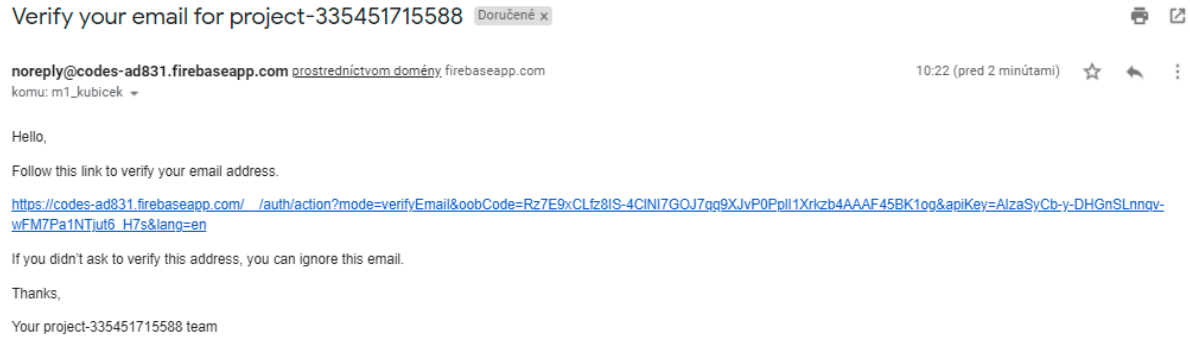
Autentizácia je vyriešená prostredníctvom emailu a hesla, ktoré sú predané metóde *signInWithEmailAndPassword*, ktorá je taktiež súčasťou triedy *FirebaseAuth*.

```
mAuth.signInWithEmailAndPassword(email,password).addOnCompleteListener(new OnCompleteListener<AuthResult>() {
    @Override
    public void onComplete(@NonNull Task<AuthResult> task) {
        if(task.isSuccessful()){
            FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();
            if(user.isEmailVerified()){
                startActivity(new Intent( packageContext: Login.this,UserProfile.class));
                progressBar.setVisibility(View.GONE);
            }else{
                user.sendEmailVerification();
                Toast.makeText( context: Login.this, text: "Please verify your e-mail. Verification email has been sent.",
                progressBar.setVisibility(View.GONE);
            }
        }else{
            Toast.makeText( context: Login.this, text: "Username or password is wrong!",Toast.LENGTH_LONG).show();
            progressBar.setVisibility(View.GONE);
        }
        editTextPassword.getText().clear();
        editTextEmail.getText().clear();
    }
});
```

Obrázok 19. Prihlásenie užívateľa Firebase

Odporúčaným spôsobom, ako získať aktuálneho používateľa, je volanie metódy *getCurrentUser*. Ak nie je prihlásený žiadny užívateľ, getCurrentUser vráti null.

V tejto funkcii je taktiež využité overovanie emailových adries pomocou metód *isEmailVerified*. Pri prvom prihlásení server Firebase pošle overovací email, vďaka metóde *sendEmailVerification*. Prostredníctvom hypertextového odkazu v doručenom emaile sa overí emailová adresa a pri ďalšom prihlásení už bude užívateľ identifikovaný ako overený.




Obrázok 20. Overovací email z databázového serveru

Po registrácii je v databáze užívateľovi pridelené špeciálne vygenerované UID, a zároveň kvôli bezpečnosti nie je v databáze uložené heslo. Pod UID sú potom uložené údaje zadané pri registrácii.

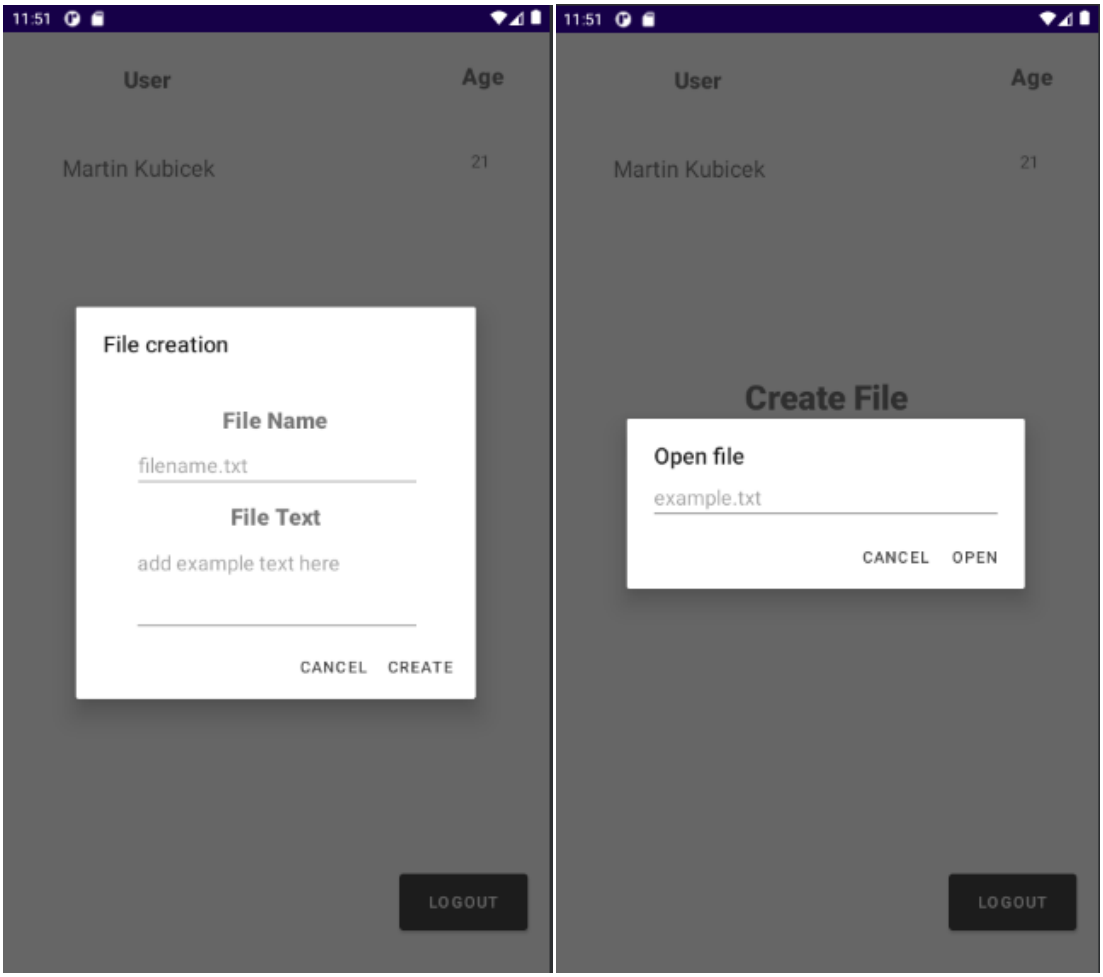

Obrázok 21. Ukážka uloženia dát v databáze Firebase

Pre kontrolu bezpečnosti bolo ešte vykonané overenie bezpečnosti stránky realtime databázy od Firebase, aby bolo isté, že sú údaje bezpečne uložené. Neboli však nájdené žiadne zraniteľnosti alebo údaje, ktoré by mohli byť zneužité. Navyše pre prístup k databáze je potrebné sa prihlásiť.

## 4.3 Šifrovanie dát

Ak aplikácia ukladá nejaké citlivé dáta je potreba správne šifrovanie. Na ukážku správnej implementácie bolo postupované podľa odporúčaní priamo na oficiálnych stránkach pre vývojárov developer.android.com [33].

V zabezpečenej aplikácii je vytvorený príklad manipulácie so súbormi, v ktorých sú uložené dáta a to vytváranie a čítanie. Pri vytvorení súboru je potrebné vkladané dáta šifrovať a zároveň pri otváraní súboru je nutné ich správne dešifrovať aby bolo programu umožnené ďalej s nimi pracovať.




Obrázok 22. a 23. Ukážka rozhraní na tvorbu a čítanie súborov

Po prihlásení sa do aplikácie používateľa uvíta obrazovka s možnosťami na otvorenie a vytvorenie súboru.

### 4.3.1 Vytvorenie súboru

Po kliknutí na „Create file“ sa objaví okno, v ktorom je umožnené vložiť názov súboru a text, ktorý sa má do súboru vložiť. Po kliknutí na „CREATE“ sa dané reťazce predajú metóde na vytvorenie súboru a šifrovanie dát. V metóde EncryptFile sa najprv vygeneruje kľúč, ktorý bude potrebný na šifrovanie pomocou AES256.

Následne sa získa cesta k súborom aplikácie, v ktorých sa za pomoci builderu vytvorí súbor. Ako parametre sa mu predajú názov súboru, kľúč a schéma šifry.

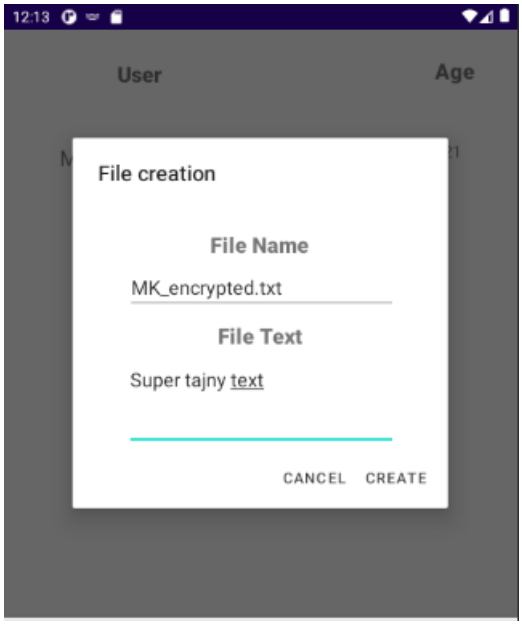
```
public boolean EncryptFile(String name, String text) {
    // Although you can define your own key generation parameter specification,
    // it's recommended that you use the value specified here.
    Context context = getApplicationContext();
    MasterKey mainKey = null;
    try {
        mainKey = new MasterKey.Builder(context)
                .setKeyScheme(MasterKey.KeyScheme.AES256_GCM)
                .build();
    } catch (GeneralSecurityException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    File appFiles = context.getFilesDir();
    EncryptedFile encryptedFile = null;
    try {
        encryptedFile = new EncryptedFile.Builder(context,
                new File(appFiles, name),
                mainKey,
                EncryptedFile.FileEncryptionScheme.AES256_GCM_HKDF_4KB
        ).build();
```

Obrázok 24. Generovanie kľúča a vytvorenie šifrovaného súboru

Zadaný text súboru je prevedený na pole bytov, a potom pomocou metódy *write()* triedy *OutputStream* zapísaný do šifrovaného súboru.

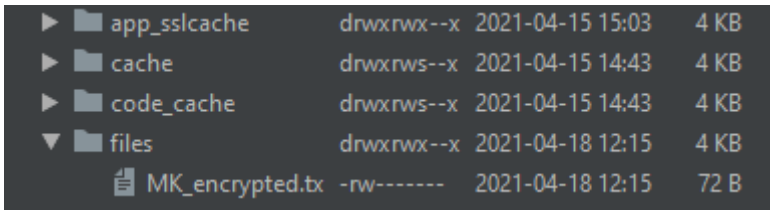
#### *4.3.1.1 Ukážka šifrovania*

Na otestovanie šifrovacieho postupu je potrebné vytvoriť súbor. Do elementu užívateľského rozhrania pre zadávanie mena súboru je zadaný názov súboru spolu s koncovkou .txt aby bol vytvorený textový súbor. Do elementu pre text je zadaný text, ktorý bude vložený do súboru.




Obrázok 25. Tvorba súboru v aplikácii

Po kliknutí na tlačidlo create sa vytvorí zašifrovaný súbor, ktorý je vidieť v zložke files pod adresárom vytvorenej aplikácie.




Obrázok 26. Súbor vytvorený v adresári aplikácie

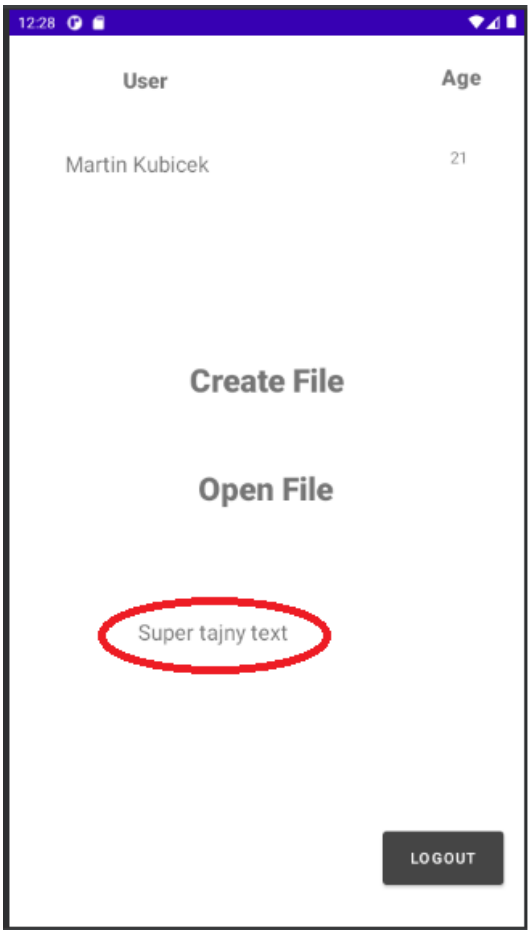
Po manuálnom otvorení súboru bez dešifrovania sa zobrazí postupnosť znakov, ktoré ako je možné vidieť utajujú zmysel správy, ktorý nie je možné prečítať. Je teda jasné, že pôvodný text je zašifrovaný.


Obrázok 27. Otvorenie súboru bez dešifrovania

### 4.3.2 Otvorenie súboru

Pre otvorenie súboru v aplikácii a následné dešifrovanie užívateľ zadá do vstupného poľa názov súboru, ktorý sa potom predá metóde. Jej úlohou je tento súbor vyhľadať v adresári aplikácie a v prípade, že je nájdený pokračuje sa dešifrovaním.

Dáta sú zo súboru čítané prostredníctvom metódy *read()* triedy *InputStream*. Následne sú skonvertované na byty a jeden po druhom vkladané do poľa bytov. Ako náhle je toto pole kompletné, je prevedené na reťazec, ktorý je vrátený metódou a následne zobrazený v aplikácii.




Obrázok 28. Zobrazenie dešifrovaného textu zo súboru

### 4.3.3 Key Management

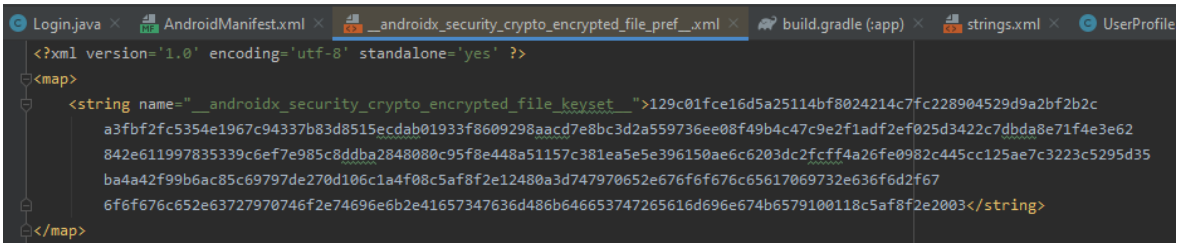
Ako bolo spomenuté v teoretickej časti tak key management je veľmi dôležitý, pretože ako náhle je kľúč od šifry prezradený, šifrovaný obsah je kompromitovaný. V tejto kapitole je popísané, ako je s kľúčom zaobchádzané pri použití odporúčaných metód zo stránky developer.android.com [40].

Knižnica Security používa na správu kľúčov dvojdielny systém:

- Keyset, ktorý obsahuje jeden alebo viac kľúčov používaných na šifrovanie súboru alebo údajov zdieľaných preferencií. Samotný keyset je uložený v priečinku Shared-Preferences.

- Master Key (primárny kľúč) je kľúč, ktorý šifruje všetky keysety. Tento kľúč je uložený pomocou Android keystore system.

Zašifrovaný keyset je teda uložený v súbore, ktorý je nazvaný podľa knižnice a to: „__androidx_security_crypto_encrypted_file_keyset__.xml“. Samotný keyset má po zašifrovaní až 520 znakov.




Obrázok 29. Zašifrovaný keyset

#### *4.3.3.1 Android keystore system*

“Systém Android Keystore umožňuje ukladať kryptografické kľúče do kontajnera, aby bolo ťažšie ich extrahovať zo zariadenia. Hneď ako sa kľúče nachádzajú v keystore, je možné ich použiť na kryptografické operácie, pričom kľúčový materiál zostáva neexportovateľný.

Okrem toho ponúka možnosti na obmedzenie toho, kedy a ako sa dajú kľúče použiť, napríklad požiadavka na autentifikáciu používateľa na použitie kľúča alebo obmedzenie použitia kľúčov, iba v určitých kryptografických režimoch. Systém Keystore používa rozhranie KeyChain API zavedené v systéme Android 4.0” – developer.android.com [40].

Android keystore taktiež obsahuje rôzne možnosti pridávania, generovania, či práce s kľúčmi.

**Ochrana proti extrahovaniu kľúčov**

Táto kapitola vychádza z poznatkov na stránke proandroiddev.com [42].

Kľúčový materiál v Android Keystore je chránený pred extrakciou pomocou dvoch bezpečnostných opatrení:

1. Kľúčový materiál nikdy nevstúpi do aplikačného procesu. Keď aplikácia vykonáva kryptografické operácie pomocou kľúča v Android Keystore, na pozadí sú zašifrovaný text, ktorý sa má dešifrovať, je privádzaný do systémového procesu, ktorý vykonáva kryptografické operácie. Ak je proces aplikácie narušený, útočník môže byť schopný použiť kľúče aplikácie, ale nemôže extrahovať ich kľúčový materiál z daného zariadenia.

2. Kľúčový materiál môže byť viazaný na zabezpečený hardware (napr. Trusted Execution Environment (TEE), Secure Element (SE)) zariadenia Android. Toto sa nazýva Hardware security module. Keď je táto funkcionalita pre kľúč povolená, kľúč nikdy nebude vystavený mimo zabezpečeného hardwaru.

Vzhľadom na to, že na vývoj je použitý emulátor, čo nie je fyzické zariadenie, hardware security modul nemôže byť použitý. Je však dôležité poznať ako funguje a preto je vhodné zaradiť jeho popis. Skratky TEE a SE sú dôležité pre popis hardware security modulu. Prítomnosť tejto funkcionality je možné overiť pomocou funkcie *isInsideSecureHardware()*.

TEE – "je technika zabezpečenia obsahu v zariadeniach so systémom Android prostredníctvom zabezpečenia oblasti hlavného procesora, aby boli chránené citlivé informácie. Ponúka izolované prostredie integrované do operačného systému. Aplikácie bežiace v TEE sa nazývajú dôveryhodné aplikácie" [51].

SE – je ešte o krok ďalej ako TEE. Je to oddelený mikročip, ktorý má vlastné CPU, úložisko, RAM atď., ktorý je špeciálne navrhnutý na riešenie bezpečnostných úkonov. SE je teda najvyššia úroveň dnešnej bezpečnosti zariadení s Android OS.

StrongBox - "je implementácia Keymaster HAL, ktorá sa nachádza v hardware security module. Ide o dôležité vylepšenie zabezpečenia pre zariadenia Android" [52].

Pokiaľ nie je možné použiť StrongBox a funkcia *isInsideSecureHardware()* vráti ako odpoveď true, tak sú kľúče chránené aspoň pomocou TEE.

Bola vykonaná kontrola, či sa kľúč nenachádza niekde v aplikácii, alebo v jej súboroch pomocou dekompilácie apk súboru. Kľúče, ani žiaden citlivý obsah aplikácie neboli nájdené.

## 4.4 Dôležitosť statickej analýzy

Na ukážku dôležitosti statickej analýzy, sú simulované útoky na dve vyskytujúce sa chyby vývojárov. Prvým je útok reverzným inžinierstvom na zraniteľnú aplikáciu, v ktorej sa nachádzajú tzv. hardcoded credentials. Tým druhým zase využitie logov na získanie citlivých informácií.

### 4.4.1 Hardcoded credentials

Hardcoded credentials by sa do slovenčiny dalo preložiť ako „natvrdo zadané prihlasovacie údaje“. Túto chybu v podstate popisuje jej názov a jej výskyt je spôsobený nasledovne.

Pri vývoji často krát dochádza k tomu, že vývojár sa špecializuje na konkrétnu oblasť, napríklad, sa môže jednať o funkcionalitu pre prihláseného používateľa. Aby sa vývojár nemusel dokola prihlasovať pri ladení kódu, vytvorí si v určitú skratku tým, že v kóde vytvorí podmienku, že ak zadá *meno=admin* a *heslo=admin,* nemusí napríklad vykonávať dvojfázové overenie a systém ho ihneď pustí ďalej, čo mu uľahčí a zrýchli prácu.

To pri vývoji samozrejme nie je nijak nebezpečné. Problém nastáva až vtedy, keď vývojár na tento kód zabudne a práve vďaka tomu, že neprebehla správna statická analýza, tento kód ostane v aplikácii až do produkčnej verzie. Keď sa potom náhodný útočník pokúsi pomocou reverzného inžinierstva zobraziť kód aplikácie, narazí tam na spomenutú skratku a tým zistí, aké meno a heslo použiť aby sa prihlásil do aplikácie.

#### *4.4.1.1 Implementácia*

Takáto situácia je implementovaná v zraniteľnej aplikácii. Je pridaná podmienka, v ktorej sa udáva, že ak je zadané meno používateľa „*admin@utb.cz*“ a heslo „*FaiUTB.cz*“, je priamo spustená aktivita zobrazujúca úspešné prihlásenie bez toho, aby bola vôbec realizovaná žiadosť na server.

```
final String email = etEmail.getText().toString();
final String password = etPassword.getText().toString();
if(email.equals("admin@utb.cz") && password.equals("FaiUTB.cz")) {
    Intent intent = new Intent( packageContext: MainActivity.this, SuccessActivity.class);
    startActivity(intent);
    MainActivity.this.finish();
}
```

Obrázok 30. Hardcoded credentials v kóde aplikácie

#### *4.4.1.2 Dekompilácia .apk súboru*

Jeden z možných útokov reverzným inžinierstvom je dekompilácia .apk súboru. Ak si užívateľ stiahne aplikáciu, stiahne si .apk súbor, v ktorom nie je možné vidieť kód. Existuje však nástroje, ktorými je možné .apk súbor dekompilovať na kód jazyka JAVA a útočník tak môže získať informácie o fungovaní aplikácie, ale aj nájsť chyby, ktoré mohli vývojári v kóde zanechať.

V simulácii bol na dekompiláciu .apk súboru využitý software Jadx [34]. Tento software umožňuje pomocou príkazového riadku získať prístup ku kódu .apk súboru jedným príkazom.

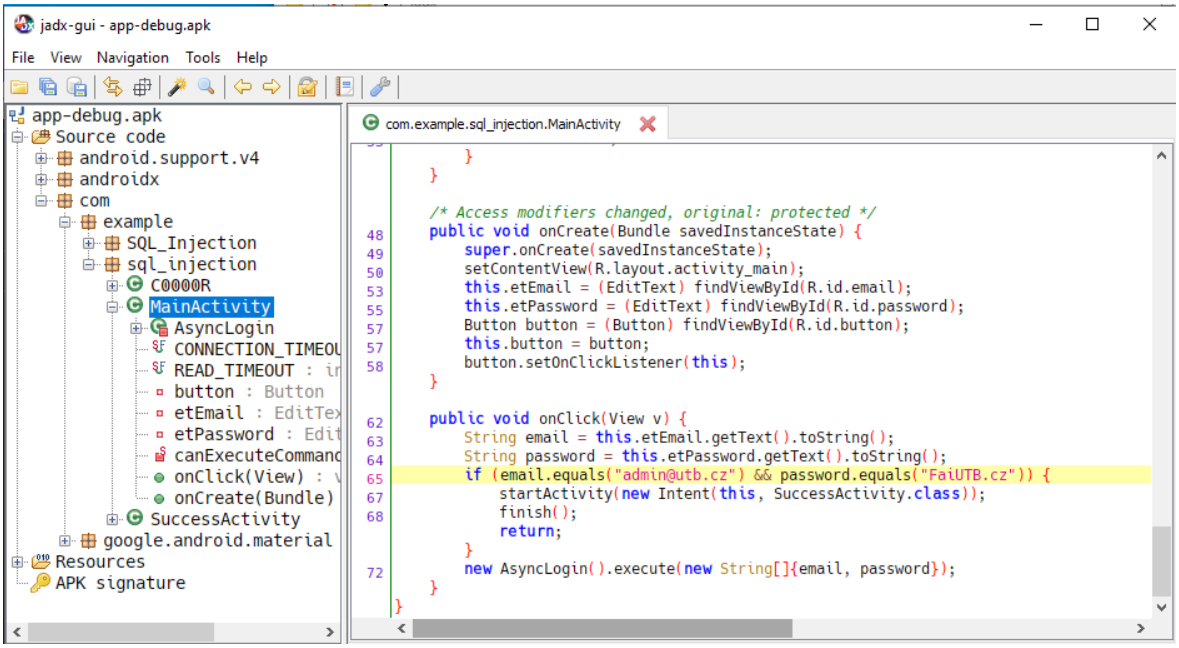
Najprv je nastavená umiestnenie na adresár, kde sa nachádza .bat súbor programu Jadx pomocou príkazu „cd“ a adresy cesty. Následne je program Jadx spustený v jeho grafickej forme príkazom „jadx-gui“ a ako parameter je predaná cestu k .apk súboru, ktorý bude dekompilovaný.

```
C:\Users\Lenovo_T430_W10P\Desktop\Bakalarka\PROGRAMY\jadx\jadx-0.9.0\bin>jadx-gui
"C:\Users\Lenovo_T430_W10P\AndroidStudioProjects\SQL_Injection\app\build\outputs\a
pk\debug\app-debug.apk"
```

Obrázok 31. Príkaz na dekompiláciu súboru

Po vykonaní tohto príkazu sa zobrazí okno, v ktorom sú zobrazené JARové balíčky, v ktorých je všetok kód, spolu so súborom AndroidManifest.xml a mnohými ďalšími súbormi. Preskúmaním obsahu balíčkov bolo zistené, že dekompiláciou bol získaný kompletný kód jednotlivých aktivít. V nich je vidno, že na riadku 65 (Obrázok. 32), sa nachádza podmienka, v ktorej sú uvedené spomínané hardcoded credentials. Taktiež je možné si všimnúť, že v kóde sú malé odlišnosti, avšak čo sa týka funkcionality alebo spomínaných údajov je

všetko dostatočne dobre viditeľné na to, aby útočník mohol zneužiť poskytnuté údaje alebo získať prehľad o fungovaní kódu a tak prípadný prístup k ďalším zraniteľnostiam.




Obrázok 32. Dekompilovaný APK súbor

#### *4.4.1.3 Regulárne výrazy*

Odporúčanou metódou statickej analýzy je vyhľadávanie regulárnych výrazov v texte, napríklad na overenie spomínaných hardcoded credentials. Samozrejme, vyhľadávanie by malo byť komplexné, aby pokrylo celý kód a chyby v ňom. V tomto prípade je však hlavným problémom meno a heslo. Pre kontrolu by mali byť vyhľadávané slová, ktoré súvisia práve s prihlasovaním. To môžu byť napríklad názvy funkcií ako „*login*“, „*register*“, „*credentials*“, „checkcredentials“ alebo názvy premenných „*email*“, „*pass*“, „*passwd*“ a podobné reťazce, či regulárne výrazy. Ďalším odporúčaním, čo sa týka regulárnych výrazov v oblasti prihlasovania je vyhľadávať znak „@“. Tento znak je totiž prítomný v emaile, a v kóde sa bežne nenachádza. Ak je teda daný znak v kóde nájdený, je veľmi pravdepodobné, že je v kóde natvrdo zadaný email, pri ktorom sa s veľkou pravdepodobnosťou nachádza aj heslo. Regulárne výrazy v kóde je možné vyhľadávať pomocou rôznych online nástrojov ako napríklad *regex101.com* [39].

### 4.4.2 Log Leakage (Únik informácií prostredníctvom logov)

Tento problém bol teoreticky popísaný už v kapitole 3.4.3. Praktická ukážka môže vyzerať nasledovne.

Vývojári môžu testovať aplikáciu a vypisovať si prihlasovacie heslo, ktoré prichádza do metódy na spracovanie, aby získali informáciu o tom, či je prenos dát správny. Použitá môže byť metóda *v()* triedy *Log*, ktorá ako parameter prijíma TAG a správu, ktorú chce vývojár vypísať. TAG sa používa na identifikáciu správy, nakoľko v logoch je množstvo ďalších informácií. Zvykne sa deklarovať ako globálna premenná v takomto tvare:

*private static final String TAG = MainActivity.class.getSimpleName(),*

kde MainActivity je preddefinovaný názov hlavnej aktivity, ktorá sa ako prvá vytvorí hneď po vytvorení projektu. Premenná s názvom TAG je potom zavolaná do logu, kde je vypísané získané heslo.

```
final String password = etPassword.getText().toString();
Log.v(TAG, msg: "Heslo vo funkcii: " + password);
```

Obrázok 33. Výpis do logu

Tieto logy môžu byť nebezpečné, pretože je možné ich odchytávať. Pokiaľ je aplikácia spustená, je možné pomocou príkazu *logcat* zobraziť logy aplikácie. Na filtrovanie podľa tagu je možné použiť príkaz s parametrom –s a v úvodzovkách je názov tagu. Výsledný príkaz je teda nasledovný: *adb logcat –s "MainActivity"* Po zadaní tohto príkazu sa vypíšu všetky logy, ktoré majú zhodný TAG. Na nasledujúcim obrázku je vidieť, že heslo bolo úspešne odchytené.

```
C:\Users\Lenovo_T430_W10P\AppData\Local\Android\Sdk\platform-tools>adb logcat -s "MainActivity"
--------- beginning of kernel
--------- beginning of main
--------- beginning of system
04-30 16:01:28.759  4594  4594 V MainActivity: Heslo vo funkcii: FaiUTB.cz
```

Obrázok 34. Odchytenie logu prostredníctom logcatu

## 4.5 Ošetrenie vstupov

Existuje rada útokov, ktoré smerujú útoky na nezabezpečené vstupy do programu s cieľom využiť zraniteľností. Je veľmi dôležité, aby daný vstup prijímal len znaky, ktoré naozaj majú zmysel. Napríklad pole, do ktorého sa očakáva zadané meno používateľa, bude akceptovať

len veľké a malé písmená a samozrejme, v prípade, že by pripadala v úvahu celosvetová prevádzka aplikácie, tak aj znaky rôznych iných abecied ako napríklad azbuka, čínske znaky atď. Rozhodne by teda nemal akceptovať rôzne špeciálne znaky a tzv. escape characters ako sú spätné lomítka, prípadne úvodzovky a rôzne ďalšie.

Ako príklad útoku, ktorý môže vznikať vďaka neošetreným vstupom je popísaná SQL Injection. Tento útok je simulovaný na zraniteľnú aj zabezpečenú aplikáciu. Ako už bolo spomenuté, zraniteľnosť, ktorá umožňuje vykonať SQL injection je veľmi nebezpečná, pretože dokáže ovplyvniť databázu a tým môže dôjsť k zmazaniu alebo odcudzeniu dát.

### 4.5.1 SQL injection

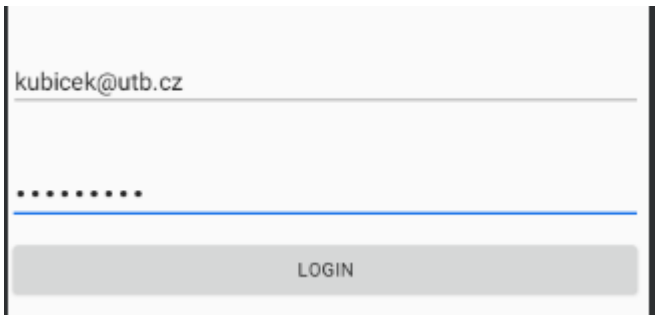
Pre vykonanie SQL injection je dobré mať prehľad o tom, ako daný systém funguje. Takéto informácie je možné získať pomocou reverzného inžinierstva. Prípadne ak nemáme prehľad o tom ako systém funguje a akým spôsobom sa pracuje s databázou, môžeme vykonať tzv. „slepý útok“, kedy skúšame do vstupov vkladať náhodné útočné reťazce a skúmame odpovede, z ktorých môžeme získať viac informácií o fungovaní systému [35].

V prípade vyvíjaných aplikácií a simulovanom útoku je však presne známe, ako daný systém funguje, preto je možné SQL injection vykonať cielene.

#### *4.5.1.1 Aplikácia pre prístup k dátam*

Pre účely práce bola vytvorená aplikácia, v ktorej je potrebné meno a heslo na prístup k dátam. Nejedná sa teda o personalizované účty, ale čisto len o autorizáciu pre prístup k dátam. Takéto fungovanie je implementované aj v zraniteľnej aplikácii a vyzeralo by nasledovne:

1. Užívateľ zadá meno a heslo do aplikačného rozhrania.




Obrázok 35. Prihlásenie do aplikácie

2. Meno a heslo je potom odoslané z aplikácie a dosadené do dotazu na databázu ako metóda POST, kde vyznačená časť sa vykoná priamo na serverovej časti v SQL databáze.

```php
$query = ("SELECT * FROM tbl_login WHERE email LIKE '$username' AND password LIKE '$password'");
$result = mysqli_query($conn, $query);
// po dosadeni vyzera dotaz takto
//$result = mysqli_query($conn, "SELECT * FROM tbl_login WHERE email LIKE 'kubicek@utb.cz' AND password LIKE 'FaiUTB.cz'");
```

Obrázok 36. PHP skript s dotazom na databázu

3. Po vykonaní dotazu sa vráti jeden alebo viac vyhovujúcich účtov späť do skriptu, kde sa testuje, či databáza našla aspoň jeden účet so zhodným menom a heslom.

```
SELECT * FROM tbl_login WHERE email LIKE 'kubicek@utb.cz' AND password LIKE 'FaiUTB.cz'
```

Profilovanie

Zobraziť všetko | Počet riadkov: 25 Filtrovať riadky: Hľadať v tabu

+ Nastavenia

| email | password |
|---|---|
| kubicek@utb.cz | FaiUTB.cz |

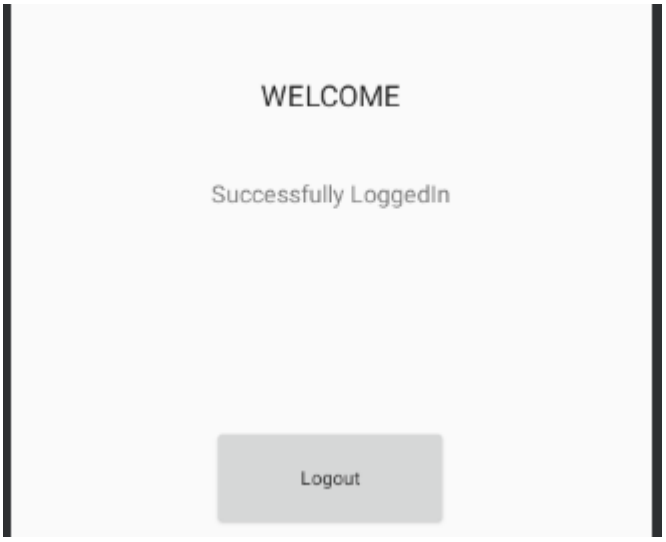
Obrázok 37. Dotaz v jazyku SQL s vráteným výsledkom

```php
if(!$result) { echo "Error: " . mysqli_error($conn); }

if ($result->num_rows > 0) {
    echo "true"; //ak sa nachzdza aspon 1 zaznam v databaze,
}               //vrati true a uzivatel ziska pristup
else{
    echo "false";//inak vrati false a pristup je zamietnuty
}
```

Obrázok 38. Spracovanie výsledku dotazu

4. V prípade že áno, program povolí užívateľovi prístup k dátam

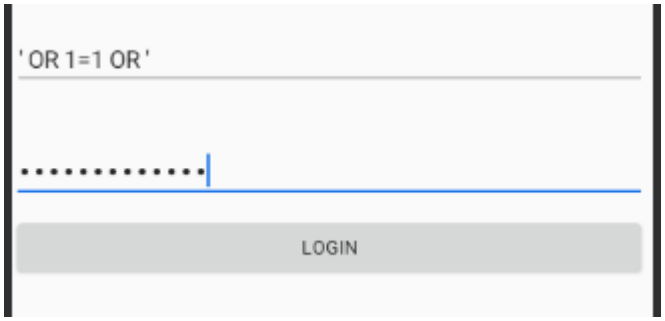



Obrázok 39. Úspešné prihlásenie používateľa

5. V prípade, že nie, vypíše sa chybové hlásenie oznamujúca chybné údaje a prístup je odopretý.

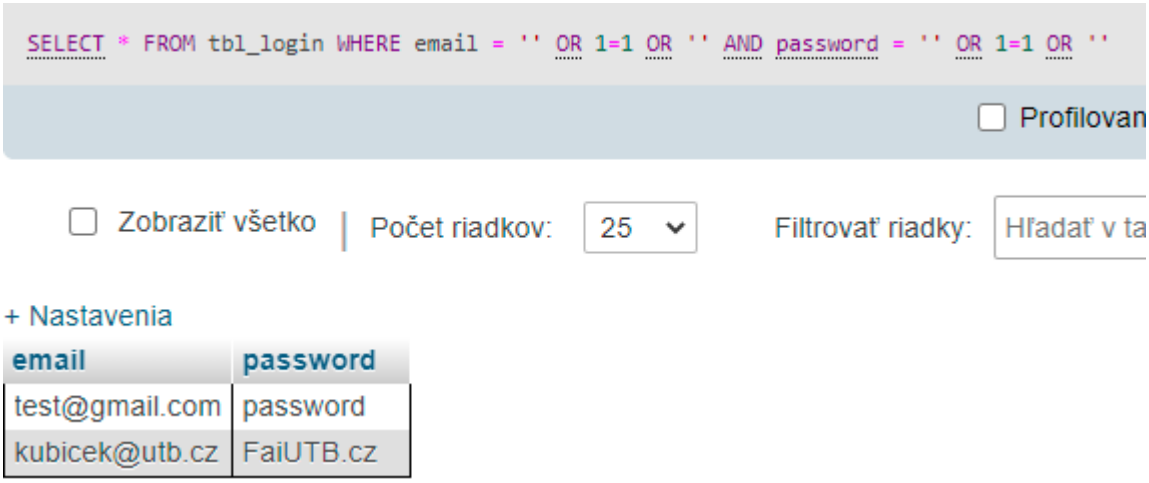
Samotná štruktúra jazyka PHP, v ktorom je riešená serverová logika sa nachádza mimo mobilnej aplikácie a údaje sú do tohto jazyka posielané metódou POST. PHP však už v syntaxe obsahuje základnú ochranu voči SQL Injection, kedy sa názov parametru píše do jednoduchých úvodzoviek, čo zabraňuje použitiu lomítok (//), prípadne pomlčiek (--) na okomentovanie zbytku výrazu. Existujú však aj spôsoby, ktorými je toto možné obísť.

V zadefinovanom type aplikácie je možné použiť reťazec SQL Injection v mene aj hesle, takže nie je potrebné ukončovať riadok.



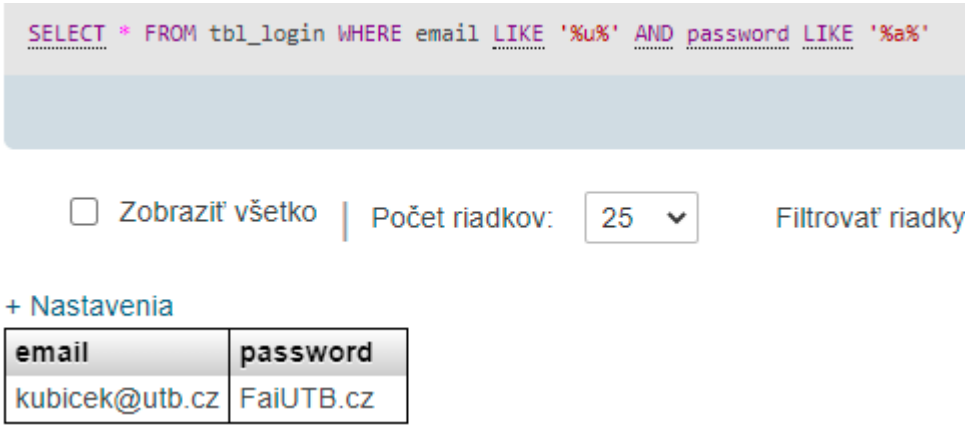

Obrázok 40. Vloženie reťazcov SQL Injection

Po zadaní reťazcu ‘ OR 1=1 OR ‘ do mena aj hesla dôjde k bezproblémovému predaniu PHP skriptu a následne databáze, v ktorej bude dotaz vyzerať nasledovne. Je zrejmé, že injection je v tomto prípade vykonaná vo vnútri výrazov, ktoré boli zadané bez akéhokoľvek vonkajšieho zásahu do dotazu.




Obrázok 41. SQL Injection v databáze

Samozrejme je možné použiť aj tzv. “*wildcard characters*“ [36], ktoré umožňujú hľadať podľa obsahujúcich písmen.




Obrázok 42. SQL Injection s použitím wildcard characters

#### *4.5.1.2 Aplikácia s personalizovanými účtami*

Prvý príklad aplikácie bol na útok značne jednoduchší. Ako ale útok môže vyzerať v prípade, že sa jedná o oveľa rozšírenejší typ aplikácií a to, že každý užívateľ má personalizovaný

účet. Takáto aplikácia sa určite neuspokojí s odpoveďou true alebo false zo serveru. Užívateľ teda musí zadať konkrétne prihlasovacie meno a správne heslo a následne mu budú z databáze ako odpoveď poskytnuté údaje.

V takomto prípade už útok prvým spôsobom nebude veľmi účinný, pretože by útočník musel zadať taký útočný reťazec, aby odpoveďou serveru bol jeden konkrétny účet. To v prípade, že sa jedná o veľmi používanú aplikáciu bude pomerne zložité.

Jednoduchším spôsobom útoku teda bude eliminácia nutnosti hľadania hesla a sústredenie sa iba na užívateľské meno.

V prípade, že je dotaz na databázu konštruovaný rovnako ako v prípade prvého typu aplikácie (Obrázok. 36), bolo by možné docieliť to okomentovaním zbytku riadku. Vďaka tomu, že je zadaný výraz v úvodzovkách nie je možné použiť escape characters v PHP ani v SQL.

Samozrejme nie všetko funguje na všetkých verziách. V tomto prípade bola na localhost serveri verzia databáze: **10.4.17-MariaDB.** Dôležité je ale to, že všetky z týchto útokov by fungovali v prvom príklade aplikácie.

Možné spôsoby získania prístupu, ktoré fungovali v aplikácii s použitím okomentovania zbytku riadku sú:

- %u%' # - použitie wildcard characters
- ' or 1=1# - vráti všetky záznamy.

Nakoľko je SQL injection veľmi rozšírená a diskutovaná téma, existuje celá rada spôsobov a možností útoku v závislosti na štruktúre a funkcionalite aplikácií, či webov.

### 4.5.2 Ochrana proti SQL Injection

V prípade, kedy by bolo realizované spojenie s databázou rovnako ako v zraniteľnej aplikácii, je hneď viacero možností ako správne pracovať so zadanými reťazcami.

Prvou, už spomínanou možnosťou je ošetrenie vstupov tzn. zabránenie vkladaniu znakov, ktoré nepatria do daného textového vstupu. To je možné realizovať pomocou *Regular Expressions,* skrátene regexu alebo pomocou triedy Patterns. Samozrejme je treba chrániť všetky vstupné polia, takže je potrebné implementovať vzory aj pre heslá, mená a pod.

```
//funkcia na osetrenie vstupu by mala byt pouzita pri spravnej implementacii
if(!isValidEmail(email)){
    Toast.makeText(getApplicationContext(), text: "Invalid email address",Toast.LENGTH_SHORT).show();
}
//dalsia moznost osetrenia je pouzitie regular expression
String emailPattern = "[a-zA-Z0-9._-]+@[a-z]+\\.+[a-z]+";
if (!email.matches(emailPattern)) {
    Toast.makeText(getApplicationContext(), text: "Invalid email address", Toast.LENGTH_SHORT).show();
}
```

Obrázok 43. Ošetrenie emailového vstupu do aplikácie

```
public static boolean isValidEmail(CharSequence target) {
    return (!TextUtils.isEmpty(target) && Patterns.EMAIL_ADDRESS.matcher(target).matches());
}
```

Obrázok 44. Funkcia na overenie emailu

Nakoľko predtým používaná trieda Pattern neponúka žiadne funkcie pre ošetrovanie hesiel, bolo nutné toto pole ošetriť „manuálne“. Zároveň je pri registrácii v aplikácii vyžadované komplexné heslo, čo je taktiež kontrolované. V prvej podmienke prebieha kontrola prítomnosti neželaných znakov v hesle. V prípade, že je nejaký nájdený funkcia vráti false a označí heslo ako nevyhovujúce a neposiela ho ďalej do kódu. Ak je heslo vyhovujúce, kontroluje sa v ďalšej podmienke jeho komplexnosť.

```
public boolean isValidPassword(final String password) {
    Pattern special = Pattern.compile ("[#/'\"~;-]");
    Matcher hasSpecial = special.matcher(password);
    if (hasSpecial.find()) {
        return false;
    }
    Pattern pattern;
    Matcher matcher;
    final String PASSWORD_PATTERN = "^(?=.*[0-9])(?=.*[a-z])(?=.*[A-Z])(?=.*[@$%^&+=*!])(?=\\S+$).{4,}$";
    pattern = Pattern.compile(PASSWORD_PATTERN);
    matcher = pattern.matcher(password);

    return matcher.matches();
}
```

Obrázok 45. Kontrola hesla

Samotné PHP môže byť považované za menej bezpečný jazyk, pretože pri implementácii kódu neskúseným vývojárom môže spôsobiť bezpečnostné problémy. Odporúča sa teda používať frameworky ako napríklad Laravel [53], ktoré sú určitou nadstavbou samotného PHP

a podporujú bezpečnosť, alebo je odporúčané používať bezpečné techniky pre prácu z parametrami, ako napríklad predávanie parametrov databáze pomocou *PHP Data Objectu* (PDO) alebo ošetrením parametrov pri načítaní z POST metódy použitím *msqli_real_escape_string()*. Pri vytváraní dotazov by malo byť vždy použité „*bindParam*“, namiesto priameho priradenia premennej do dotazu.

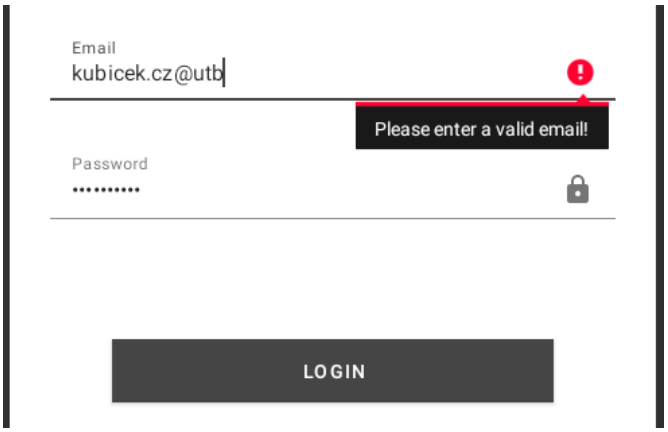
```php
//spravne pouzitie
$username = $mysqli -> real_escape_string ($_POST['username']);
$password = $mysqli -> real_escape_string ($_POST['password']);

$sql = 'SELECT * FROM tbl_login WHERE email = :username AND password = :password';
$stmt = $conn->prepare($sql);
$stmt->bindParam(':username', $username, PDO::PARAM_STR);
$stmt->bindParam(':password', $password, PDO::PARAM_STR);
$stmt->execute();
```

Obrázok 46. Funkcia na overenie emailu

### 4.5.3 Ochrana proti SQL Injection v zabezpečenej aplikácii

V kapitole 4.2.3 bolo popisované použitie databázy Firebase v implementovanej aplikácii. Veľká výhoda, čo sa týka tejto zraniteľnosti je to, že Firebase Realtime Database je databáza typu NoSQL, čo znamená, že nie je náchylná na SQL Injection útoky. Dáta sú miesto toho, aby boli formované do dotazov SQL spracovávané API a práve preto nehrozí SQL Injection. Samozrejme aj napriek tomu, že nie je možné prevádzať SQL injection, aplikácia obsahuje rovnaké patterny a metódy na overenie emailových adries a ostatných polí rovnako, ako bolo popisované pri bežnej implementácii SQL databáz v predchádzajúcej kapitole (viď Obrázok 39). Jedinou zmenou pre užívateľa je tu upravenie grafickej stránky aplikácie a okamžité upozornenie používateľa na chybne zadaný email, prípadne heslo [35].




Obrázok 47. Oznámenie o chybnom emaile

# 5 ZHRNUTIE POZNATKOV ZISTENÝCH BEHOM IMPLEMENTAČNÉHO PROCESU

Vývoj aplikácií, študovanie materiálov a navrhovanie či už bezpečných, alebo nebezpečných ukážkových postupov prinieslo množstvo skúseností. V rámci tejto práce bola vykonaná rešeršná príprava, kde boli študované a citované odborné popisy zraniteľností a odporúčané postupy. Táto kapitola popisuje poznatky, ktoré boli zistené behom implementácie a tvorby aplikácií. Aplikácie boli vyvíjané vo vývojovom prostredí Android Studio 4.1. v jazyku JAVA.

## 5.1 Bezpečnosť komunikácie je závislá len na vývojárovi

Vývojové prostredie Android Studio je veľmi praktické a užívateľsky prívetivé. Ponúka všetky funkcionality, ktoré vývojár potrebuje, aby mohol pohodlne vytvárať mobilné aplikácie. Čo sa však týka používania bezpečnej implementácie internetovej komunikácie, obsahuje len základné ochranné prvky. Spočiatku sa zdalo, že vývojové prostredie neobsahuje žiadne takéto odporúčania bezpečnosti, avšak pri implementácii názornej ukážky prenosu pomocou protokolu HTTP, došlo k chybe, ktorú sa podarilo identifikovať až pomocou krokovania programu. Bolo zistené, že kód porušuje prednastavené bezpečnostné opatrenia.

```
    conn = (HttpURLConnection)url.openConnection();  url: "http://192.168.241.197/BP/http/login.inc.php"
    conn.setReadTimeout(READ_TIMEOUT);
    conn.setConnectTimeout(CONNECTION_TIMEOUT);
    conn.setRequestMethod("POST");

    // setDoInput and setDoOutput method depict handling of both send and receive
    conn.setDoInput(true);
    conn.setDoOutput(true);

    // Append parameters to URL
    Uri.Builder builder = new Uri.Builder()
            .appendQueryParameter("username", params[0])
            .appendQueryParameter("password", params[1]);  params: {"kubicek@utb.cz..., "FaiUTB.cz"}
    String query = builder.build().getEncodedQuery();

    // Open connection for sending data
    OutputStream os = conn.getOutputStream();
    BufferedWriter writer = new BufferedWriter(
            new OutputStreamWriter(os,  charsetName: "UTF-8"));
    writer.write(query);
    writer.flush();
    writer.close();
    os.close();
    conn.connect();  conn: "com.android.okhttp.internal.huc.HttpURLConnectionImpl:http://192.168.241.197/BP/http/login.
} catch (IOException e1) {  e1: "java.io.IOException: Cleartext HTTP traffic to 192.168.241.197 not permitted"
```

Obrázok 48. Hlásenie o nepodporovaní HTTP prenosu

"*Cleartext HTTP traffic to (IP adresa serveru) not permitted*".

Toto hlásenie znamená, že Android Studio nepovoľuje nešifrovanú komunikáciu, pretože pochopiteľne je veľmi nebezpečná a neodporúčaná. Je však jednoduchšie obísť tieto „odporúčania" ako meniť celý postup implementácie, čo môže byť nebezpečné hlavne ak vývojár nie je oboznámený s bezpečnými postupmi. To platí taktiež pokiaľ postupuje podľa neaktuálnych dokumentácií. Pri implementácii však bolo požiadavkou poukázať na nebezpečnosť HTTP, preto bolo pokračované v implementácii týmto spôsobom, aj napriek varovaniu, ktoré sa podarilo obísť veľmi jednoduchým spôsobom.


Obrázok 49. Spôsob obídenia hlásenia

Použitím *android:usesCleartextTraffic = "true"* je umožnené povoliť posielanie nezašifrovaných dát a prenos tak funguje úplne normálne. Z toho vyplýva, že vývojové prostredie bráni vývojárovi v používaní extrémne nebezpečných a dobre známych chybných postupov, avšak nedá sa povedať, že by vyslovene viedlo k používaniu bezpečných štandardov. Výsledná aplikácia teda stále môže byť náchylná na útoky a môže obsahovať veľa zraniteľností.

## 5.2 Neexistujúca ochrana proti SQL injection

Pri realizácii práce bola jednou z hlavných tém realizácia SQL Injection a ochrana proti nej. To viedlo k zisteniu, že pri implementácii dovolí IDE Android Studio bez akéhokoľvek upozornenia vývojárovi implementovať postupy, ktoré sú veľmi náchylné na vznik zraniteľností, prostredníctvom ktorých je možné realizovať SQL injection. Hlavne, pri práci s databázami, kedy sú informácie získavané zo vstupov, neexistuje žiadna ochrana či ošetrenie vstupov voči zadaniu escape characters.

Rovnako ako implementácia použitých vzorov pre kontrolu napríklad emailu. Bez znalosti tohto problému a použitia štandardného čítania vstupov, by ako email prešiel reťazec „ *' OR 1=1");//* ", ktorý nemá s emailom nič spoločné, takže okrem toho, že by pri ukladaní mohol spôsobiť chyby v databáze, by mohol pri prihlasovaní spôsobiť okomentovanie žiadaného dotazu na databázu v PHP skripte a následnú SQL injection v databáze.

Je teda veľmi dôležité ošetrovať každý vstup do aplikácie nehľadiac na to, či môže predstavovať hrozbu SQL injection prípadne akúkoľvek inú. Ak je teda od užívateľa požadované, aby zadal email, je nutné striktne obmedziť použiteľné znaky a ich postupnosť tak, aby mohol byť zadaný len existujúci email. V odbornej literatúre je však našťastie možné dohľadať množstvo vzorov, ktoré je sú použiteľné na filtrovanie jednotlivých typov vstupných polí.

## 5.3 Neexistuje dlhodobo bezpečná implementácia

Pri vývoji aplikácií je veľmi dôležité myslieť na fakt, že neexistuje dokonalá a hlavne trvalá ochrana. Výpočtový výkon zariadení stále stúpa, útočné techniky sa neustále zlepšujú a reagujú na najnovšie bezpečnostné restrikcie. S príchodom nových verzií operačných systémov prichádzajú nové funkcionality, v ktorých sa môžu ukrývať nové bezpečnostné chyby. S príchodom nových verzií operačných systémov sa môže stať, že aktuálne používané bezpečnostné postupy už viac nebudú aplikovateľné.

Je preto potreba neustále sa informovať a držať si prehľad o aktuálnych útokoch a zraniteľnostiach a aplikáciu aktualizovať ihneď po tom, čo je zistené nové slabé miesto. Pokiaľ je aplikácia používaná globálne, je toto ešte dôležitejšie, pretože čím viac ľudí aplikáciu používa, tým bude pre útočníkov lákavejším cieľom.

## 5.4 Rovnováha medzi zabezpečením a použiteľnosťou

Pri implementácii aplikácie patrila podstatná časť navrhovaniu rôznych zabezpečení a postupov, čo viedlo k zisteniu, že čím je aplikácia bezpečnejšia, tým horšie sa používa a navyše sú zvýšené náklady, kvôli potrebe vývoja bezpečnostných mechanizmov. To môže byť vysvetlené na nasledujúcom príklade autentizácie. V najbezpečnejšom možnom prípade, si aplikácia pýta od užívateľa heslo pred každým vstupom do aplikácie, alebo aj pred vykonaním dôležitých zmien či iných úkonov. Zadané heslo by malo byť komplexné a zložité.

Uvedený problém má niekoľko príčin. Prvou je to, že ľudia nie sú stroje, a majú problém si dostatočne zložité heslá pamätať. Druhým problémom je frekvencia. Ak by aplikácia vyžadovala vkladanie hesla pri každom prihlásení, bolo by to nepochybne určitým spôsobom otravné a zdĺhavé. To by viedlo užívateľov k tomu, že by prestali používať aplikáciu. Aby bola aplikácia tzv. „user-friendly“, je potreba počty prihlásení značne zredukovať, prípadne umožniť aplikácii pamätanie si hesla. To má však zase za následok zníženie bezpečnosti a vznik rizík spojených s odcudzením, alebo prezradením hesla.

Takéto problémy existujú v množstve iných odvetví vývoja aplikácií, a ako je možné si všimnúť, tento efekt môže byť interpretovaný ako bod na priamke, kde na jednej strane je bezpečnosť a na druhej použiteľnosť. Je teda dôležité určiť si cieľ aplikácie a nastaviť pomyselný bod tak, aby bola docielená požadovaná úroveň zabezpečenia, ale zároveň, aby aplikácia bola dostatočne dobre použiteľná. V dnešnej dobe už nám s týmto problémom čiastočne pomáhajú rôzne biometrické funkcie ako odtlačok prsta, skenovanie tváre a pod.

# ZÁVER

Cieľom tejto práce bolo ozrejmiť a poukázať na vyskytujúce sa zraniteľnosti androidových aplikácií. Napriek tomu, že sa na bezpečnosť kladie väčší dôraz ako v minulosti, stále existuje množstvo stávajúcich zraniteľností v aplikáciách, ku ktorým sa pridávajú nové. V teoretickej časti bol kladený dôraz na zoznámenie sa s najbežnejšími zraniteľnosťami, vyskytujúcimi sa v mobilných aplikáciách. Bolo popísané, ako tieto zraniteľnosti vznikajú, ako môžu byť zneužité, a aké môžu mať dopady úspešné útoky, ktoré využívajú prítomnosť zraniteľností. Ďalším krokom bolo popísanie bezpečnostných postupov, ktoré môže vývojár urobiť, aby predišiel vzniku popisovaných zraniteľností. Bolo potrebné urobiť dôkladný prieskum, a vybrať správne a aktuálne postupy, ktoré môžu pomôcť zabezpečiť aplikáciu.

Po spracovaní týchto odporúčaní bolo možné navrhnúť súbor pravidiel, ktoré by mali byť použité pri vývoji aplikácií. Po zamyslení sa nad všetkými zraniteľnosťami bolo vytvorených 5 pravidiel, ktoré pri správnej implementácii zachytávajú väčšinu zraniteľností. Zameriavajú sa na šifrovanie, detekciu rootu, upozorňujú na nebezpečnosť HTTP, či pomáhajú eliminovať SQL Injection alebo iný nebezpečný kód.

Pri implementácii bola vytvorená aj zraniteľná aplikáciu, aby bolo možné lepšie porozumieť daným zraniteľnostiam a možnosti ako ich využiť. Následnou implementáciou navrhnutých pravidiel z ktorých boli niektoré porovnávané práve s nezabezpečenou aplikáciou. Tým bolo možné otestovať ich bezpečnosť a funkčnosť navrhnutých pravidiel.

Prvým pravidlom bola detekciu rootu, ktorá bola otestovaná na viacerých emulátoroch s funkčným výsledkom detekcie a znemožnením použitia aplikácie na takomto zariadení. Naopak zraniteľná aplikácia bez akejkoľvek detekcie sa pochopiteľne spustila bez problémov.

HTTP prenos v zraniteľnej aplikácii bol prostredníctvom programu Burp Suite a proxy serveru úspešne odchytený, čím sa podarilo preukázať, že dáta sú posielané vo forme otvoreného textu a prenos je veľmi nebezpečný.

Šifrovanie súborov bolo taktiež implementované v zabezpečenej aplikácii. Bol použitý odporúčaný mechanizmus využívajúci šifru AES256. Úspešnosť šifrovania bola testovaná otvorením zašifrovaného súboru bez využitia dešifrovacieho algoritmu. To potvrdilo, že text v súbore je zašifrovaný. Následne bol súbor otvorený práve s použitím dešifrovacieho

algoritmu implementovaného v aplikácii, aby bolo možné potvrdiť správnosť a obojsmernosť použitého algoritmu.

Na dôležitosť statickej a dynamickej analýzy bolo poukázané pomocou reverzného inžinierstva a to konkrétne dekompiláciou APK súboru, kde sa v zraniteľnej aplikácii podarilo objaviť natvrdo zadané prihlasovacie údaje, ktoré potom mohli byť zneužité na prihlásenie do aplikácie.

Poslednou témou bola veľmi známa SQL Injection, ktorá bola demonštrovaná na zraniteľnej aplikácii. V dôsledku viacerých porušení bezpečných postupov bolo možné úspešne využiť útok SQL Injection na prihlásenie sa do systému. V zabezpečenej aplikácii sa okrem použitia doporučeného ošetrovania vstupov použila aj NoSQL databáza, ktorá zaručuje ochranu proti SQL útokom.

# ZOZNAM POUŽITEJ LITERATÚRY

[1] Mobile OS market share 2021 | Statista. • *Statista - The Statistics Portal for Market Data, Market Research and Market Studies* [online]. Copyright © Statista 2021 [cit. 26.03.2021]. Dostupné z: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/

[2] OWASP Mobile Top 10. *OWASP Foundation, the Open Source Foundation for Application Security* [online]. [cit. 26.03.2021]. Dostupné z: https://owasp.org/www-project-mobile-top-10/

[3] M1: Improper Platform Usage | OWASP. *OWASP Foundation, the Open Source Foundation for Application Security* [online]. [cit. 26.03.2021]. Dostupné z: https://owasp.org/www-project-mobile-top-10/2016-risks/m1-improper-platform-usage

[4] The 7 Best Cross-Platform Mobile Development Tools. *Custom Software Development Denver - Custom Software Consulting* [online]. Copyright © [cit. 26.03.2021]. Dostupné z: https://www.brainspire.com/blog/the-7-best-cross-platform-mobile-development-tools

[5] M2: Insecure Data Storage | OWASP. *OWASP Foundation, the Open Source Foundation for Application Security* [online]. [cit. 27.03.2021]. Dostupné z: https://owasp.org/www-project-mobile-top-10/2016-risks/m2-insecure-data-storage

[6] Rooting your Android phone: Security risks & disadvantages. *Comparitech - Tech researched, compared and rated* [online]. Copyright © 2021 Comparitech Limited. All rights reserved. [cit. 28.03.2021]. Dostupné z: https://www.comparitech.com/blog/information-security/rooting-android-security/

[7] All You Need to Know About Android App Vulnerability: Insecure Communication - Astra Security Blog. *ASTRA Security - Complete Website Security Suite* [online]. Copyright © 2020 [cit. 28.03.2021]. Dostupné z: https://www.getastra.com/blog/app-security/all-you-need-to-know-about-android-app-vulnerability-insecure-communication/

[8] What is MITM (Man in the Middle) Attack | Imperva. *Cyber Security Leader | Imperva, Inc.* [online]. Copyright © 2021 Imperva. All rights reserved [cit. 28.03.2021]. Dostupné z: https://www.imperva.com/learn/application-security/man-in-the-middle-attack-mitm/

[9] M4: Insecure Authentication | OWASP. *OWASP Foundation, the Open Source Foundation for Application Security* [online]. [cit. 28.03.2021]. Dostupné z: https://owasp.org/www-project-mobile-top-10/2016-risks/m4-insecure-authentication

[10] What is Authentication? - Definition from WhatIs.com. *Information Security information, news and tips - SearchSecurity* [online]. [cit. 28.03.2021]. Dostupné z: https://searchsecurity.techtarget.com/definition/authentication

[11] What is a Botnet? | Kaspersky. *Kaspersky Cyber Security Solutions for Home & Business | Kaspersky* [online]. Copyright © [cit. 28.03.2021]. Dostupné z: https://usa.kaspersky.com/resource-center/threats/botnet-attacks

[12] M5: Insufficient Cryptography | OWASP. *OWASP Foundation, the Open Source Foundation for Application Security* [online]. [cit. 28.03.2021]. Dostupné z: https://owasp.org/www-project-mobile-top-10/2016-risks/m5-insufficient-cryptography

[13] M6: Insecure Authorization | OWASP. *OWASP Foundation, the Open Source Foundation for Application Security* [online]. [cit. 28.03.2021]. Dostupné z: https://owasp.org/www-project-mobile-top-10/2016-risks/m6-insecure-authorization

[14] What is Authorization? - Examples and definition - Auth0. *Auth0: Secure access for everyone. But not just anyone.* [online]. Copyright © 2013 [cit. 28.03.2021]. Dostupné z: https://auth0.com/intro-to-iam/what-is-authorization/

[15] M7: Poor Code Quality | OWASP. *OWASP Foundation, the Open Source Foundation for Application Security* [on-line]. [cit. 29.03.2021]. Dostupné z: https://owasp.org/www-project-mobile-top-10/2016-risks/m7-client-code-quality

[16] M8: Code Tampering | OWASP. *OWASP Foundation, the Open Source Foundation for Application Security* [online]. [cit. 29.03.2021]. Dostupné z: https://owasp.org/www-project-mobile-top-10/2016-risks/m8-code-tampering

[17] Code Injection - an overview | ScienceDirect Topics. *ScienceDirect.com | Science, health and medical journals, full text articles and books.* [online]. Copyright © [cit. 01.04.2021]. Dostupné z: https://www.sciencedirect.com/topics/computer-science/code-injection

[18] What is SQL Injection? Tutorial & Examples | Web Security Academy. *Web Application Security, Testing, & Scanning - PortSwigger* [online]. Copyright © 2021 PortSwigger Ltd. [cit. 01.04.2021]. Dostupné z: https://portswigger.net/web-security/sql-injection

[19] M9: Reverse Engineering | OWASP. *OWASP Foundation, the Open Source Foundation for Application Security* [online]. [cit. 01.04.2021]. Dostupné z: https://owasp.org/www-project-mobile-top-10/2016-risks/m9-reverse-engineering

[20] M10: Extraneous Functionality | OWASP. *OWASP Foundation, the Open Source Foundation for Application Security* [online]. [cit. 01.04.2021] Dostupné z: https://owasp.org/www-project-mobile-top-10/2016-risks/m10-extraneous-functionality

[21] Security tips | Android Developers. *Android Developers* [online]. [cit. 04.04.2021]. Dostupné z: https://developer.android.com/training/articles/security-tips#StoringData

[22] Network Security Best Practices | Android Open Source Project. *Android Open Source Project* [online]. [cit. 04.04.2021]. Dostupné z: https://source.android.com/security/best-practices/network

[23] Best Practices For Encryption In Android| Motodev for enterprise [online]. [cit. 08.04.2021]. Dostupné z: http://www.afu.com/MFE_Encryption_Whitepaper.pdf

[24] Critical mistakes to avoid in Android development. *Learn to Code — For Free — Coding Courses for Busy People* [online]. [cit. 08.04.2021]. Dostupné z: https://www.freecodecamp.org/news/first-5-mistakes-to-avoid-in-android-development-51177007a4f6/

[25] Avoid reverse engineering of your android app. *Software Development Company in USA | Simform* [online]. Copyright © 2021 Simform. All Rights Reserved. [cit. 05.04.2021]. Dostupné z: https://www.simform.com/how-to-avoid-reverse-engineering-of-your-android-app/

[26] NoxPlayer - Free Android Emulator on PC and Mac. *NoxPlayer - Free Android Emulator on PC and Mac* [online]. Copyright © [cit. 11.04.2021]. Dostupné z: https://www.bignox.com/

[27] Android root detection using build tags? - Stack Overflow. *Stack Overflow - Where Developers Learn, Share, & Build Careers* [online]. [cit. 10.04.2021]. Dostupné z: https://stackoverflow.com/questions/18808705/android-root-detection-using-build-tags

[28] Signing Builds for Release | Android Open Source Project. *Android Open Source Project* [online]. [cit. 11.04.2021]. Dostupné z: https://source.android.com/devices/tech/ota/sign_builds

[29] 10+ BEST Root Apps for Android (2021 Update). *Meet Guru99 - Free Training Tutorials & Video for IT Courses* [online]. Copyright © Copyright [cit. 11.04.2021]. Dostupné z: https://www.guru99.com/best-root-apps-android.html

[30] What is BusyBox? (in English) | XDA Developers Forums. *XDA Developers Forums* [online]. [cit. 15.04.2021]. Dostupné z: https://forum.xda-developers.com/t/what-is-busybox-in-english.1261946/

[31] Configuring an Android Device to Work With Burp - PortSwigger. *Web Application Security, Testing, & Scanning - PortSwigger* [online]. [cit. 18.04.2021]. Copyright © 2021 PortSwigger Ltd. [cit. 18.04.2021]. Dostupné z: https://portswigger.net/support/configuring-an-android-device-to-work-with-burp

[32] Firebase Realtime Database | Store and sync data in real time. *Firebase* [online]. [cit. 18.04.2021]. Dostupné z: https://firebase.google.com/products/realtime-database

[33] Android Developers. *Android Developers* [online]. [cit. 18.04.2021]. Dostupné z: https://developer.android.com/topic/security/data

[34] GitHub - skylot/jadx: Dex to Java decompiler. *GitHub: Where the world builds software · GitHub* [online]. Copyright © 2021 GitHub, Inc. [cit. 19.04.2021]. Dostupné z: https://github.com/skylot/jadx

[35] SQL Injection Cheat Sheet | Netsparker. *Netsparker | Web Application Security For Enterprise* [online]. Copyright © Netsparker 2021, by Invicti [cit. 22.04.2021]. Dostupné z: https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/

[36] SQL Wildcard Characters. *W3Schools Online Web Tutorials* [online]. [cit. 25.04.2021]. Dostupné z: https://www.w3schools.com/sql/sql_wildcards.asp

[37] Man In The Middle Attack - Man In The Middle, HD Png Download - kindpng. *Millions Of High Quality PNG Images For Free Download - kindpng* [online]. Copyright © Shenzhen BestAI Internet Co., Ltd . 2019 All Rights Reserved [cit. 22.04.2021]. Dostupné z: https://www.kindpng.com/imgv/hRixwhw_man-in-the-middle-attack-man-in-the/

[38] Commercial National Security Algorithm Suite. *National Security Agency - Applications* [online]. Copyright © 1998 [cit. 28.04.2021]. Dostupné z: https://apps.nsa.gov/iaarchive/programs/iad-initiatives/cnsa-suite.cfm

[39] regex101: build, test, and debug regex. *regex101: build, test, and debug regex* [online]. [cit. 30.04.2021]. Dostupné z: https://regex101.com/

[40] App security best practices | Android Developers. *Android Developers* [online]. [cit. 28.04.2021]. Dostupné z: https://developer.android.com/topic/security/best-practices

[41] Android keystore system | Android Developers. *Android Developers* [online]. [cit. 28.04.2021] Dostupné z: https://developer.android.com/training/articles/keystore#GeneratingANewPrivateKey

[42] ProAndroidDev.com [online]. [cit. 28.04.2021] Dostupné z: https://proandroiddev.com/android-keystore-what-is-the-difference-between-strongbox-and-hardware-backed-keys-4c276ea78fd0

[43] XAMPP Installers and Downloads for Apache Friends. [online]. [cit. 01.04.2021] Dostupné z: https://www.apachefriends.org/index.html

[44] Nástavby operačního systému Android | Alza.cz. *Alza.cz – nakupujte bezpečně z pohodlí domova* | Alza.cz [online]. [cit. 11.05.2021]. Dostupné z: https://www.alza.cz/nastavby-android

[45] What is IP Spoofing and How to Prevent It | Kaspersky. *Kaspersky Cyber Security Solutions for Home & Business* | Kaspersky [online]. Copyright © [cit. 11.05.2021]. Dostupné z: https://www.kaspersky.com/resource-center/threats/ip-spoofing

[46] Update to Current Use and Deprecation of TDEA | CSRC. *NIST Computer Security Resource Center* | CSRC [online]. [cit. 11.05.2021] Dostupné z: https://csrc.nist.gov/news/2017/update-to-current-use-and-deprecation-of-tdea

[47] NIST Withdraws Outdated Data Encryption Standard | NIST. *National Institute of Standards and Technology* | NIST [online]. [cit. 11.05.2021] Dostupné z: https://www.nist.gov/news-events/news/2005/06/nist-withdraws-outdated-data-encryption-standard

[48] GDB: The GNU Project Debugger. *The GNU Operating System and the Free Software Movement* [online]. [cit. 11.05.2021] Dostupné z: https://www.gnu.org/software/gdb/

[49] Cryptography | Android Developers. *Android Developers* [online]. [cit. 11.05.2021]. Dostupné z: https://developer.android.com/guide/topics/security/cryptography

[50] Explore Memory and Resource Leak Detection Tools | ICS. *ICS - Integrated Computer Solutions* [online]. Copyright © 1999 [cit. 11.05.2021]. Dostupné z: https://www.ics.com/blog/explore-memory-and-resource-leak-detection-tools

[51] TEE in Android Development - DZone Mobile. *DZone* [online]. [cit. 11.05.2021]. Dostupné z: https://dzone.com/articles/overview-tee-in-android

[52] Google Security Blog [online]. [cit. 11.05.2021]. Dostupné z: https://security.googleblog.com/2021/03/announcing-android-ready-se-alliance.html

[53] Laravel - The PHP Framework For Web Artisans. Laravel - *The PHP Framework For Web Artisans* [online]. Copyright © 2011 [cit. 11.05.2021]. Dostupné z: https://laravel.com/

[54] ARP Spoofing | Veracode. *Confidently secure your applications with Veracode* [online]. Copyright © 2021 VERACODE, All Rights Reserved 65 Network Drive, Burlington MA 01803 [cit. 12.05.2021]. Dostupné z: https://www.veracode.com/security/arp-spoofing

[55] Understand the Activity Lifecycle | Android Developers. *Android Developers* [online]. [cit. 12.05.2021]. Dostupné z: https://developer.android.com/guide/components/activities/activity-lifecycle

[56] Obfuskace a základní obfuskační techniky - CleverAndSmart Management Consulting. *CleverAndSmart Management Consulting* [online]. [cit. 12.05.2021]. Copyright © 2008 [cit. 12.05.2021]. Dostupné z: https://www.cleverandsmart.cz/obfuskace-a-zakladni-obfuskacni-techniky/

# ZOZNAM POUŽITÝCH SYMBOLOV A SKRATIEK

| | |
|---|---|
| iOS | iPhone Operation System – operačný systém zariadení iPhone |
| OS | Operation System – všeobecne operačný systém |
| MITM | Man In The Midle – typ útoku na internetovú komunikáciu |
| IP | Internet Protocol – sada pravidiel internetovej komunikácie |
| URL | Uniform Resource Locator – lokácia informácií umiestnených na internete |
| MAC | Media Access Control – unikátny identifikátor priradený sieťovému adaptéru |
| ARP | Address Resolution Protocol – komunikačný protokol |
| DNS | Domain Name System – systém prideľovania doménových mien |
| SQL | Structured Query Language – dotazovací jazyk na tvorbu databáz |
| TCP | Transmission Control Protocol – protokol uľahčujúci výmenu informácií |
| SSL | Secure Socket Layer – protokol na šifrovanie komunikácie |
| HTTP | Hypertext Transfer Protocol – protokol slúžiaci na prenos dát internetom |
| HTTP | Hypertext Transfer Protocol Secure – kombinácia HTTP a SSL |
| TLS | Transport Layer Security – vylepšená verzia SSL |
| NFC | Near Field Communication – technológia umožňujúca bezkontaktné platby |
| MFA | Multi Factor Authentication – viac faktorová autentikácia |
| DoS | Denial of Service - internetový útok, ktorého cieľom je znemožnenie prevádzky |
| DDoS | Distributed DoS – Distibuovaný DoS útok s využitím viacerých zariadení |
| PIN | Personal Identification Number – identifikačné číslo, slúžiace ako forma hesla |
| XML | Extensible Markup Language – jazyk popisujúci text v digitálnom dokumente |
| SMTP | Simple Mail Transfer Protocol – protokol riadiaci príchod a odchod emailov |
| API | Aplication Programming Interface – prostredie, ktorým komunikujú aplikácie |
| UAT | User Acceptance Testing – proces testovania samotným používateľom |
| REST | Representational State Transfer – softwarový architektonický štýl |

| | |
|---|---|
| IPC | Inter Process Communication – komunikácia procesov v nejakom celku |
| ASLR | Age,Sex,Location,Race – základné identifikačné informácie používateľa |
| CA | Certification Authority – entita vystavujúca digitálne certifikáty |
| UID | Unique identifier – unikátny identifikátor entity, odlišujúci ju od ostatných |
| OTA | Over-The-Air – forma bezdrôtovej komunikácie |
| ROM | Read Only Memory – pamäť uspôsobená len na čítanie |
| USB | Universal Serial Bus – rozhranie na prenos dát |
| PC | Personal Computer – osobný počítač, laptop, pracovná stanica |
| TEE | Trusted Execution Environment – časť hardwaru obsahujúca vlatný systém |
| SE | Secure Element – vyšší level TEE |
| PDO | PHP Data Object – slúži na komunikáciu s dababázou |

# ZOZNAM OBRÁZKOV

# ZOZNAM PRÍLOH

Zoznam príloh na CD.

# PRÍLOHA P I: CD

CD obsahuje hlavný adresár s názvom “Bakalárska_Práca_Martin_Kubíček” v ktorej sú ďalšie 2 podadresáre. Prvý má názov “Aplikácie” a sú v ňom uložené 2 .apk súbory aplikácií s názvami “Zraniteľná_aplikácia.apk” a “Zabezpečená_aplikácia.apk”. Druhý podadresár má názov “Skripty” a sú v ňom prítomné 2 .php súbory s názvami “config.inc.php” a “login.inc.php”