Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Bezpečnost Android aplikací z pohledu vývojáře

Bakalářská práce

Autor: Filip Oborník
Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Milan Košťák

Hradec Králové duben 2023

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 21. 4. 2023 Filip Oborník

Poděkování:

Děkuji vedoucímu bakalářské práce Ing. Milanu Košťákovi za metodické vedení práce a veškeré poskytnuté konzultace.

Anotace

Práce se věnuje analýze bezpečnosti mobilních aplikací pro mobilní platformu Android z pohledu vývojáře. Při vývoji pro tuto platformu existuje mnoho potenciálních bezpečnostních chyb, kterých se vývojář může dopustit a které mohou být mnoha způsoby zneužity. Zneužití chyb může například způsobit únik citlivých dat uživatele nebo napadení dalších částí systému. Často je možné chybám, které mohou mít fatální následky, předejít už jen povědomím o jejich existenci, zneužitelnosti a následné znalosti obrany proti nim. Cílem bakalářské práce je jednotlivé potenciální chyby s dopadem na bezpečnost popsat a poukázat na možnosti jejich zneužití. Pro každé potenciální riziko je navrženo řešení, které má za cíl ztížit, případně i znemožnit, jeho zneužití.

Annotation

Title: Security of Android applications from the developer's perspective

The thesis is devoted to the analysis of the security of mobile applications for the Android mobile platform from the developer's point of view. When developing for this platform, there are many potential security vulnerabilities that a developer can commit and that can be exploited in many ways. For example, exploiting vulnerabilities can cause leakage of sensitive user data or compromise other parts of the system. Often, these vulnerabilities, which can have fatal consequences, can be prevented just by being aware of their existence, their exploitability, and then knowing how to defend against them. The aim of the bachelor thesis is to describe each potential vulnerability with security implications and point out the possibilities of their abuse. For each potential risk, a solution is proposed to make it more difficult, or even impossible, to exploit it.

Obsah

1 Úvod *1*

2 Metodika zpracování *2*

3 Bezpečnost systému Android *3*

3.1 Odpovědnost za bezpečnost aplikace 3

3.2 Bezpečnost operačního systému 4

3.2.1 Sandboxing 4

4 Klíčové oblasti bezpečnosti mobilních aplikací *6*

4.1 Ukládání dat 6

4.2 Kryptografie 6

4.3 Autentizace a autorizace 6

4.4 Kvalita kódu a zmírnění zranitelnosti 7

4.5 Obrana proti úpravě a rozklíčování kódu 7

4.6 Síťová komunikace 8

4.7 Interakce s operačním systémem a ostatními aplikacemi 8

5 Root zařízení *9*

5.1 Provedení rootu zařízení 9

6 Ukládání dat aplikace *10*

6.1 Data typu klíč – hodnota 10

6.1.1 Shared preferences 10

6.1.2 Data Store 12

6.1.3 Získání dat ze zařízení s root právy 14

6.1.4 Získání dat ze zařízení bez root práv 16

6.1.5 Obrana – šifrování dat 18

6.2 Databáze 19

6.2.1 SQLite 19

6.2.2 SQLite – zneužití a obrana proti *SQL injection* 19

6.2.3 SQLite – zneužití a obrana proti čtení dat ze souboru 21

6.2.4 Alternativní databáze 23

6.3 Soubory **23**

6.3.1 Privátní soubory aplikace 23

6.3.2 Veřejné soubory aplikace 25

6.3.3 Média 26

6.3.4 Sdílené úložiště 26

7 Autentizace a autorizace ***27***

8 Dekompilace aplikace ***28***

9 Obfuskace zdrojového kódu ***36***

9.1 Rizika při nepoužití obfuskace kódu v Android aplikaci **37**

9.1.1 Zobrazení debug informací 38

9.1.2 Zjištění funkčnosti aplikace a její modifikace 39

10 Bezpečnost síťové komunikace ***40***

10.1 Odposlech nebo úprava síťové komunikace **40**

10.1.1 Android 6 a starší 41

10.1.2 Android 7 a novější 44

10.1.3 Zařízení s provedeným root 45

10.2 Obrana proti útoku Man in the middle (MITM) **46**

10.2.1 Certificate pinning 46

10.2.2 Public key pinning 47

10.2.3 Network security configuration 48

11 Interakce s ostatními aplikacemi ***50***

11.1 Intents **50**

11.2 Intent filtry **51**

11.3 Content providers **52**

12 Závěry a doporučení ***53***

13 Seznam použité literatury ***54***

Seznam obrázků

Obrázek 1 Entity s odpovědností za používání aplikace [2] 4
Obrázek 2 Izolace aplikací v rámci operačního systému [2] 5
Obrázek 3 Diagram možnosti ukládání dat Android aplikace [autor] 10
Obrázek 4 Porovnání zakódovaných a dekódovaných dat pro *Preferences Data Store* [autor] 15
Obrázek 5 Proces kompilace aplikace [15] 28
Obrázek 6 Znázornění obsahu APK souboru po rozbalení programem unzip [autor] 29
Obrázek 7 Nečitelný obsah AndroidManifest.xml souboru po rozbalení APK souboru programem unzip [autor] 29
Obrázek 8 Rozdíl kompilace klasické Java aplikace a Android aplikace [16] 30
Obrázek 9 Znázornění obsahu APK souboru pomocí Android Studia [autor] 31
Obrázek 10 Znázornění obsahu APK souboru pomocí Android Studia [19] 33
Obrázek 11 Porovnání struktury balíčků po dekompilaci mezi obfuskovaným a neobfuskovaným kódem [autor] 37
Obrázek 12 Ukázka dekompilovaných obfuskovaných metod v jazyce Java zobrazených v programu JD-GUI [autor] 37
Obrázek 13 Znázornění útoku typu Man in the middle [22] 40
Obrázek 14 Nastavení portu proxy serveru programu Charles Proxy [autor] 41
Obrázek 15 Zobrazení chyby při pokusu čtení HTTPS komunikace bez validního certifikátu v programu Charles Proxy [autor] 42
Obrázek 16 Nastavení *SSL proxying* pro požadovanou doménu [autor] 43
Obrázek 17 Zobrazení odpovědi z REST API z odposlechnuté HTTPS komunikace [autor] 44

Seznam ukázek kódu

Ukázka kódu 1 Inicializace přístupu do úložiště *Shared Preferences* 11
Ukázka kódu 2 Získání hodnot uložených v *Shared Preferences* pomocí jejich klíče 11
Ukázka kódu 3 Uložení hodnot do *Shared Preferences* pomocí jejich klíče 12
Ukázka kódu 4 Inicializace přístupu do úložiště *Preferences* a *Proto Data Store* .. 13
Ukázka kódu 5 Získání dat z *Preferences* a *Proto Data Store* pomocí jejich klíče .. 13
Ukázka kódu 6 Uložení hodnot do Preferences a Proto Data Store pomocí klíče . 14
Ukázka kódu 7 XML soubor s uloženými daty v úložišti Shared Preferences 14
Ukázka kódu 8 Dekódování soubory z *Protobuf* formátu do čitelné podoby 15
Ukázka kódu 9 Ukázka nastavení atribut allowBackup v Manifestu aplikace 16
Ukázka kódu 10 Ukázka nastavení atribut *allowBackup* a odkazu na konfigurační soubor pro upravení pravidel zálohy v Manifestu aplikace 17
Ukázka kódu 11 Ukázka XML konfiguračního souboru *backup_rules* pro upravení chování zálohy souboru s daty úložiště *Shared Preferences* 17
Ukázka kódu 12 SQL dotaz pro získání detailu uživatele na základě jeho e-mailu a PINu 20
Ukázka kódu 13 Ukázka zranitelného přístupu do databáze sestavením SQL dotazu a manuálním dosazením uživatelského vstupu 20
Ukázka kódu 14 SQL dotaz s vloženými uživatelskými vstupy zneužívající útok *SQL injection* 20
Ukázka kódu 15 Využití PreparedStatement pro dosazení uživatelských vstupů do SQL dotazu 21
Ukázka kódu 16 Ukázka XML konfiguračního souboru *backup_rules* pro upravení chování zálohy souboru s daty SQLite databáze 22
Ukázka kódu 17 Ukázka přístupu do privátního úložiště aplikace 24
Ukázka kódu 18 Ukázka XML konfiguračního souboru *backup_rules* pro upravení chování zálohování privátních souborů aplikace 25
Ukázka kódu 19 Ukázka přístupu do veřejného úložiště aplikace 25
Ukázka kódu 20 Ukázka spuštění příkazu apktool pro dekompilaci aplikace ze souboru *application.apk* do souboru *./output/new-directory* 31

Ukázka kódu 21 Ukázka spuštění příkazu apktool pro kompilaci aplikace ze souboru *application.jar* do souboru APK 32
Ukázka kódu 22 Příkaz na spuštění programu dexToJar pro dekompilaci zdrojových kódů aplikace v souboru app.apk do jazyka Java nebo Kotlin 32
Ukázka kódu 23 Původní zdrojový kód v jazyce Kotlin využitý pro ukázku dekompilace do jazyka Java v následující ukázce kódu 33
Ukázka kódu 24 Část výstupu dekompilace aplikace psané v jazyce Kotlin dekompilované do jazyku Java 34
Ukázka kódu 25 Podmínka pro zápis logu do konzole pouze v případě, že je aplikace spuštěna v debug módu 38
Ukázka kódu 26 Nastavení aplikace pro spuštění v debug módu v Manifestu 38
Ukázka kódu 27 Konfigurační XML soubor pro nastavení důvěry v uživatelsky instalované certifikáty 45
Ukázka kódu 28 Nastavení odkazu na XML konfigurační soubor pro úpravu nastavení zabezpečení síťové komunikace 45
Ukázka kódu 29 Načtení vlastního certifikátu ze zdrojových souborů aplikace typu *raw* 46
Ukázka kódu 30 Nastavení využití vlastního certifikátu pomocí systémové komponenty *TrustManager* 46
Ukázka kódu 31 Způsob realizace HTTPS spojení se serverem, aby došlo k využití instalovaného vlastního certifikátu 47
Ukázka kódu 32 Získání veřejného klíče z lokálně uloženého certifikátu pomocí nástroje *openssl* 47
Ukázka kódu 33 Připnutí veřejného klíče certifikátu pomocí třídy *CertificatePinner* 48
Ukázka kódu 34 Konfigurační XML soubor pro přidání veřejného klíče certifikátu. Klíč je pro účely ukázky upraven na smyšlenou hodnotu. 48
Ukázka kódu 35 Nastavení odkazu na konfigurační XML soubor pro úpravu zabezpečení síťové komunikace v Manifestu aplikace 49
Ukázka kódu 36 Nastavení atributu aktivity *exported*, aby nebylo možné aktivitu spustit ostatními aplikacemi 51

Ukázka kódu 37 Nastavení Aktivity aplikace na možnost zpracování odeslání textových dat pomocí Intent filtru 52

1 Úvod

Android je dnes nejrozšířenějším operačním systémem pro chytré telefony. Jeho zastoupení na trhu činí 72 % k 3. 12. 2022 [1]. Od svého uvedení na trh v roce 2008 společností Google ve verzi Android 1.0 až po současnou verzi Android 13 (rok 2023), lze pozorovat stále větší důraz na jeho bezpečnost. Tento trend je logický, protože dnes telefony neslouží pouze k prohlížení internetu, zasílání SMS a telefonování, ale uživatelé v nich uchovávají citlivá data a ovládají z nich spoustu důležitých služeb pomocí nainstalovaných mobilních aplikací. Příkladem může být mobilní bankovnictví, investiční platformy, e-maily, elektronické pokladny a mnoho dalšího. Proto jak společnost Google, tak i vývojáři aplikací, více a více dbají na bezpečnost. Existují i společnosti, které se zabývají penetračním testováním Android aplikací, kde výstupem jejich práce je protokol s popisem nalezených zranitelností aplikace

Ze strany společnosti Google lze pozorovat důraz na snahu odstínit vývojáře od implementačních detailů a na zavádění vynucených systémových restrikcí, ve snaze co nejvíce zmenšit možnost vzniku potencionálních bezpečnostních chyb, kterých se vývojáři mohou při vývoji dopustit. I přes to, že je dnes platforma Android považována za bezpečnou, je stále určitá část odpovědnosti za bezpečnost aplikace na jejím vývojáři. Proto je nutné, aby vývojáři aplikací měli o problematice bezpečnosti povědomí a při vývoji aplikací se bezpečnostních chyb vyvarovali.

Cílem práce je identifikace bezpečnostních rizik a chyb, kterých se programátor může při tvorbě aplikace pro systém Android dopustit. Práce se opírá jak o oficiální doporučení společnosti Google, vyvíjející tento operační systém, tak i o rady firem a jednotlivců, zabývajících se bezpečností mobilních aplikací a další veřejně dostupné zdroje. Zároveň využívá zkušeností autora z několikaleté praxe vývoje Android aplikací.

2 Metodika zpracování

Práce cílí na vývojáře mobilních aplikací, nikoliv na uživatele nebo vývojáře operačních systémů. Jednotlivá témata jsou popisována z pohledu vývojáře mobilní aplikace. Navrhované řešení vždy bere v potaz omezené možnosti, které vývojář mobilní aplikace má v rámci systému Android a neuvažuje možnost úpravy přístupu na úrovni operačního systému nebo edukaci chování uživatele, které jsou mimo možnosti vývojáře.

Vybrané bezpečnostní chyby a s nimi spojená rizika jsou v práci teoreticky popsána. Jsou také poskytnuty praktické ukázky příkladů realizace útoků, které dané chyby zneužívají. Pro každý útok jsou popsány a rozebrány důsledky, které může na aplikaci a na jejího uživatele mít.

Pro jednotlivé chyby a rizika je navrženo a prakticky popsáno řešení, jakým způsobem může programátor předcházet zneužití daného rizika již při vývoji aplikace. Společně s popisem je poskytnuta i ukázková implementace daného řešení pomocí ukázek částí kódu v programovacích jazycích Kotlin a Java. Ukázky konfiguračních souborů jsou v jazyce XML a databázové dotazy pomocí standardního dotazovacího jazyka SQL.

3 Bezpečnost systému Android

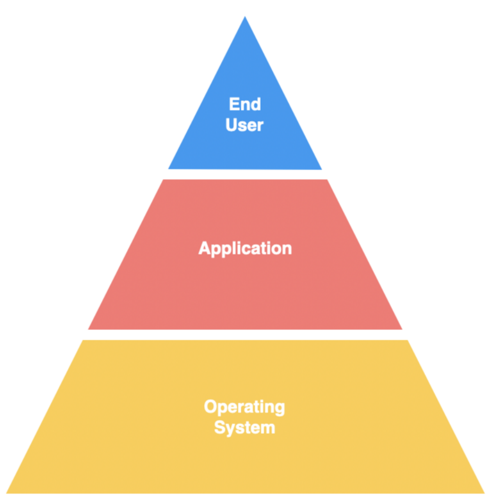
Systém Android pro chytré mobilní telefony vznikl v roce 2008. Od té doby prošel značným vývojem a v roce 2023 je obecně považován za bezpečný operační systém. Společnost Google se s každou novou verzí systému snaží zvyšovat jeho bezpečnost. Popis změn, oprav chyb a nových nebo upravených funkcionalit je možné zjistit ze změnového seznamu (anglicky *changelog*), který je vydáván pro každou novou verzi operačního systému Android.

3.1 Odpovědnost za bezpečnost aplikace

Odpovědnost za bezpečnost při používání mobilní aplikace lze pro zjednodušení rozdělit mezi 3 entity:

- **Operační systém** – jeho odpovědností je poskytnout bezpečné prostředí pro běh aplikace a zároveň poskytnout vývojáři aplikace co nejmenší prostor pro vznik bezpečnostní chyby pří vývoji a běhu aplikace, a to primárně pomocí systémových restrikcí (např. systémová povolení, bezpečný přístup do úložiště a další).
- **Aplikace** – je zodpovědností vývojáře, aby aplikace byla bezpečná – tj. nakládala bezpečně s daty uživatele, komunikovala zabezpečeně s okolím a neumožnila ostatním aplikacím přístup k citlivým datům uživatele nebo spuštění uživatelem neiniciovaných akcí v aplikaci nebo systému.
- **Uživatel** – zodpovědností uživatele je zacházet s aplikací tak, aby nedošlo k odhalení citlivých dat (například přístupových údajů), anebo instalace nebezpečné aplikace z neznámých zdrojů do systému mimo oficiální obchod Google Play.

Vývojář aplikace z povahy systému může primárně ovlivnit pouze bezpečnost aplikační části. Co se týče ostatních entit, uživatele lze edukovat o bezpečném chování přímo v aplikaci. V případě operačního systému lze pouze podávat podněty na zlepšení systému společnosti, která ho vyvíjí – v tomto případě společnosti Google.



Obrázek 1 Entity s odpovědností za používání aplikace [2]

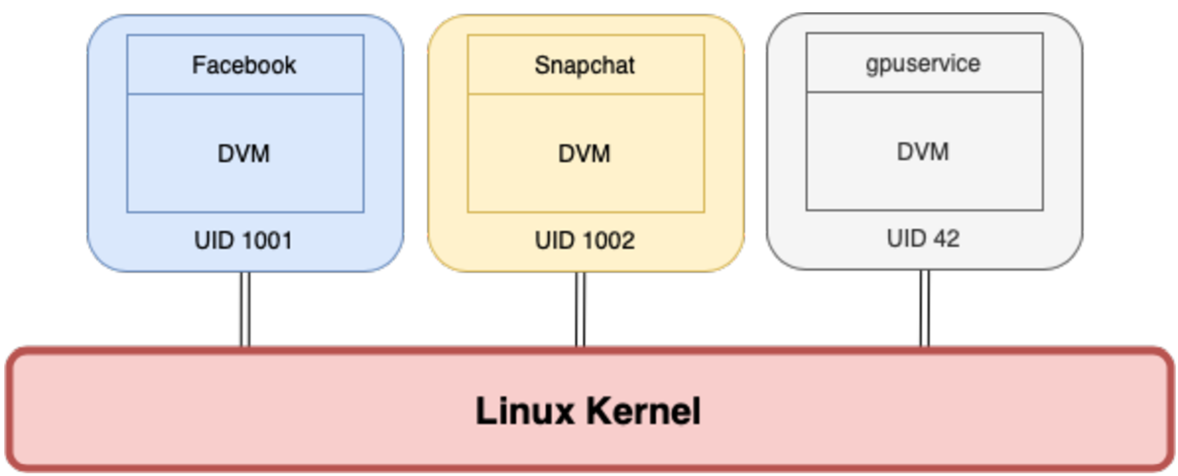
3.2 Bezpečnost operačního systému

Jádro každého operačního systému do velké míry definuje bezpečnost systému jako celku. Operační systém Android je postaven na jádře Linuxu, které definuje klíčové bezpečnostní funkce:

- Izolace procesů
- Model oprávnění
- Meziprocesová komunikace

3.2.1 Sandboxing

Operační systém Android pro běh aplikací využívá principu zvaný *Sandboxing*. Ten slouží k izolaci prostředků aplikace od ostatních aplikací. Cílem je, aby běžící aplikace nemohly mezi sebou samovolně komunikovat. To omezuje přístup aplikace k datům ostatních aplikací, které nejsou explicitně uvedeny jako dostupné pro ostatní aplikace.



Obrázek 2 Izolace aplikací v rámci operačního systému [2]

4 Klíčové oblasti bezpečnosti mobilních aplikací

Společnost The OWASP Foundation, komunita zabývající se bezpečností mobilních a webových aplikací, jmenuje několik klíčových oblastí bezpečnosti a rozebírá je v knize „OWASP Mobile Application Security Testing Guide" [3]. Kapitola je rozdělena na několik podkapitol, z nich se každá zabývá vybranou oblastí, pro kterou jsou vždy obecně nastíněna možná rizika.

4.1 Ukládání dat

Ukládání a obrana citlivých dat je klíčovou součástí bezpečnosti aplikace. Chybné uložení dat může způsobit jejich odhalení ostatním aplikacím, což může vést k jejich krádeži nebo zneužití. Android aplikace mají několik možností, jak a kam data ukládat, a každá z těchto možností slouží pro jiné účely a má jinou míru zabezpečení. Proto je stěžejní, aby programátor data podle jejich účelu a citlivosti ukládal správně.

4.2 Kryptografie

Práce s citlivými daty uživatele v aplikaci je složitá. Při návrhu aplikace by proto měl být kladen důraz, aby citlivá data byla v aplikaci ukládána pouze v případech, kdy je to nezbytné. Ideální situací je, aby drtivá většina citlivých dat zůstala uložena na serveru a pro ověření a nakládání s nimi byly využity alternativní metody, jako například JWT, tokenizace platebních karet a další.

Nicméně ne všechny situace lze řešit alternativním přístupem a někdy je nutné lokálně v zařízení citlivá data uchovat. V tomto případě je správné použití kryptografie stěžejní. Absence kryptografie, použití slabých kryptografických algoritmů nebo chyba v jejich použití může umožnit útočníkovi rozšifrovat citlivá data, která lze dále zneužít. Může se jednat o osobní údaje, hesla, API klíče, údaje platebních karet a mnoho dalších.

4.3 Autentizace a autorizace

Většina aplikací vyžaduje pro jejich používání určitou formu ověření uživatele. Zároveň také uživateli umožňuje přihlášení do aplikace zapamatovat, aby při

každém spuštění nemusel zadávat své přihlašovací údaje. Bezpečnostní problémy mohou vniknout již při samotném procesu přihlašování, například nešifrovanou komunikací se serverem, lokálním uložením nešifrovaných přihlašovacích údajů, nebo jejich výpisem do logu aplikace. Při zapamatování přihlášení uživatele může bezpečnostní hrozbu představovat již zmíněné zapamatování přihlašovacích údajů uživatele, nesprávné uložení autorizačního tokenu uživatele nebo nemožnost zneplatnění tohoto tokenu v případě odcizení zařízení.

V případě aplikací obsahujících citlivá data může být problémem i absence autentizace uživatele při spuštění aplikace. V případě, že uživatel má nastavené slabé heslo telefonu, nebo ho nemá nastavené vůbec, může pak útočník při zcizení telefonu aplikaci se zapamatovaným přihlášením používat a vydávat se tak za uživatele. Proto aplikace pracující s citlivými daty, jako např. mobilní bankovnictví, požadují vždy při spuštění aplikace ověření uživatele. Nikoliv však opětovným zadáním často složitých přihlašovacích údajů, ale alternativními způsoby, jakými může být jednodušší PIN, otisk prstu nebo rozpoznání obličeje.

4.4 Kvalita kódu a zmírnění zranitelnosti

Mobilní aplikace jsou mnohem méně náchylné na útoky typu SQL injection, XSS nebo podobné, protože se často systém nebo systémové knihovny postarají o obranu před jejich zneužitím, nicméně i tak je nelze vyloučit.

4.5 Obrana proti úpravě a rozklíčování kódu

Aplikace jsou do zařízení instalovány pomocí instalačních balíčků APK. Tyto instalační balíčky lze jednoduše získat z Google Play a následně je možné je dekompilovat a získat tak přístup ke zdrojovému kódu a případně ho i upravovat. Této skutečnosti nelze zabránit, nicméně vývojář aplikace může znatelně ztížit a skoro až znemožnit její zneužití. Nejefektivnější metodou je použitím tzv. obfuskace kódu, která zdrojový kód převede do nesrozumitelné podoby a násobně tak ztíží tzv. *reverse engineering* aplikace, česky také jako reverzní inženýrství.

4.6 Síťová komunikace

Komunikace po síti je dnes součástí většiny aplikací. Rizikem je používání nešifrovaného přenosu dat nebo riziko jejich rozšifrování. Jednoduše nelze předejít pokusům o odposlechnutí komunikace mezi serverem a aplikací, nicméně lze zabránit navázání komunikace s neověřeným serverem, minimalizovat přenos citlivých dat a případně přenášená data šifrovat.

4.7 Interakce s operačním systémem a ostatními aplikacemi

Systém Android umožňuje aplikaci interagovat jak s ním samotným, tak i s ostatními aplikacemi v zařízení, např. pomocí systému oprávnění, broadcastů, intent filtrů a dalších systémových konstruktů. Aplikace tak může vystavit svoje služby ostatním aplikacím nebo vysílat do systému informace o události. Nesprávné použití může způsobit, že škodlivá aplikace bude tyto informace zachytávat, případně nutit napadenou aplikaci k vykonání uživatelem neiniciované akce. Obranou je co nejpřísnější restrikce nabízených a přijímaných požadavků a omezení jejich práv na nezbytné minimum pro provedení dané akce.

5 Root zařízení

Společným jmenovatelem spousty způsobů a přístupů k napadení aplikace je tzv. „root“ zařízení, který napadení umožní anebo znatelně zjednoduší. Jedná se o proces přepnutí operačního systému Android do privilegovaného režimu. Provádí se za účelem možnosti překonat omezení výrobců zařízení, jako je například odinstalace předinstalovaných aplikací, spuštění specializovaných aplikací nebo provádění operací, které jsou pro běžného uživatele nedostupné.

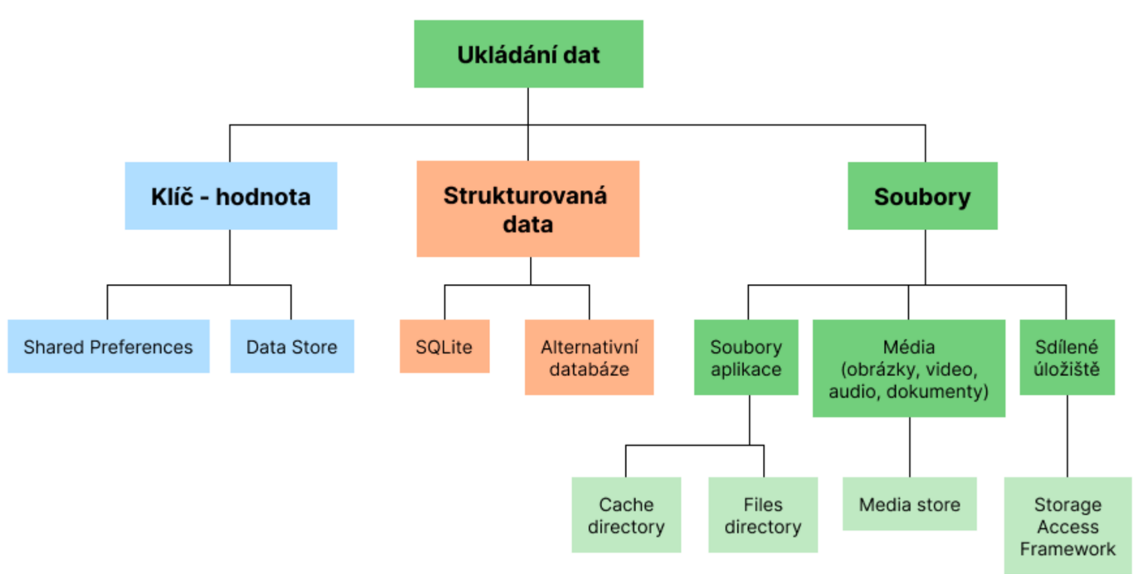
Zároveň s sebou root přístup přináší značná rizika. V případě napadení telefonu může útočník velice jednoduše zneužít tohoto privilegovaného režimu a dostat se například k datům, která by před tím jinak operační systém skryl. Umožňuje také odposlechnout HTTPS komunikaci. Více příkladů je uvedeno v dalších kapitolách práce.

5.1 Provedení rootu zařízení

Provedení rootu zařízení se může lišit pro každého výrobce telefonu a někdy i model zařízení. Návod pro specifické zařízení jde ve většině případů nalézt na stránkach *https://www.xda-developers.com/root/*. [4] Pro zjednodušení rootu zařízení lze využít nástroje Magisk (*https://magiskmanager.com/).*

6 Ukládání dat aplikace

Pro uložení dat v rámci Android aplikace existuje mnoho možností, kdy každá z nich je určená pro různé druhy dat s různou úrovní zabezpečení. Klíčovým krokem programátora při vývoji aplikace je správně určit, o jaká data se jedná a jakou úroveň zabezpečení vyžadují.



Obrázek 3 Diagram možnosti ukládání dat Android aplikace [autor]

6.1 *Data typu klíč – hodnota*

Pro ukládání jednoduchých dat, jako je například uživatelské jméno, informace že uživatel viděl, nebo neviděl, úvodní obrazovku aplikace nebo další jednoduchá metadata, je vhodné využít úložiště typu „klíč – hodnota". Jeho výhodou je, že není nutné dopředu definovat schéma, jak budou daná data ukládána. Zároveň je čtení a zápis do něj rychlé a na práci s ním není vyžadováno tak velké množství systémových prostředků, jako je tomu u ostatních typů ukládání dat. [5]

6.1.1 Shared preferences

Jedná se o dnes již zastaralé, nicméně stále aktivně používané řešení systému Android pro ukládání dat typu klíč – hodnota. Do úložiště se přistupuje pomocí

instance třídy *SharedPreferences*. Tu lze získat pomocí instance třídy *Context* a programátorem zvoleného názvu. Zároveň je nutné nastavit mód přístupnosti. [5]

```
context.getSharedPreferences("fileName", Context.MODE_PRIVATE)
```

Ukázka kódu 1 Inicializace přístupu do úložiště *Shared Preferences*

Mód přístupnosti může nabývat následujících hodnot, které určují, jaké aplikace a procesy se mohou k uloženým datům dostat:

- *MODE_WORLD_WRITABLE* – ostatní aplikace mohou uložená data z úložiště číst a zapisovat do něj.
- *MODE_WORLD_READABLE* – ostatní aplikace mohou uložená data číst.
- *MODE_PRIVATE* – pouze aplikace, která úložiště vytvořila, může data z úložiště číst a zapisovat do něj. Ostatní aplikace nemají informaci, že tato data kdekoliv v úložišti existují.

Z pojmenování a popisu jednotlivých módů je zřejmé, že pro uložení dat, ke kterým by žádná jiná aplikace neměla mít přístup, je nutné použít mód *MODE_PRIVATE*. Od verze Android 7 (rok 2016) jsou dokonce módy *MODE_WORLD_READABLE* a *MODE_WORLD_WRITABLE* považovány za takovou bezpečnostní hrozbu, že v případě jejich použití bude aplikace generovat výjimku *SecurityException*. [5]

I přes použití módu *MODE_PRIVATE* nelze data uložená do tohoto úložiště považovat za zabezpečená. Jsou uložena v nezašifrované formě a určitými technikami se k nim může dostat i někdo jiný než vlastnická aplikace.

Získání dat z úložiště s jakýmkoliv módem se provádí nad instancí třídy *Shared Preferences*. K jednotlivým hodnotám je přistupováno pomocí jejich klíče, který byl zvolen při uložení dat.

```
sharedPrefs.getString("userName")
sharedPrefs.getInt("userId", -1)
```

Ukázka kódu 2 Získání hodnot uložených v *Shared Preferences* pomocí jejich klíče

```
sharedPrefs.edit()
      .putString("userName", "Martin Novák")
      .putInt("userId", 123)
      .apply()
```

Ukázka kódu 3 Uložení hodnot do *Shared Preferences* pomocí jejich klíče

6.1.2 Data Store

Data Store vznikl jako náhrada za klasické *Shared Preferences* a poskytuje oproti nim několik výhod. Jednou z nich je možnost ukládat typované objekty bez nutnosti převádět je na JSON formát a ukládat je jako datový typ *String*. Poskytuje tedy typovou bezpečnost. Další výhodou je optimalizace pro použití *Kotlin coroutines* a *Flows*, což umožňuje efektivnější práci s uloženými hodnotami. [6]

Data Store umožňuje použití ve dvou různých implementacích:

- **Preferences Data Store** – funguje obdobně jako *Shared Preferences* na základě ukládání dat jako typ klíč – hodnota. Data nejsou ukládána podle pevně definovaného schématu a není zajištěna typová bezpečnost.
- **Proto Data Store** – ukládá data jako instance vlastního datového typu. Pracuje s předem pevně definovaným schématem ukládaných dat, což zajišťuje typovou bezpečnost.

Do úložiště se přistupuje pomocí vytvoření rozšiřující proměnné na třídě *Context* přistupující k instanci *DataStore<Preferences>* a volání delegované funkce *preferencesDataStore* nebo přes instanci *DataStore<T>*, kde *T* představuje generický datový typ ukládaný do úložiště, a volání delegované funkce *dataStore()*. Parametry této funkce jsou jméno souboru, kam budou data ukládána, a instance serializéru pro daný datový typ *T*. [6]

```kotlin
// Preferences Data Store
val Context.dataStore: DataStore<Preferences>
  by preferencesDataStore(name = "settings")


// Proto DataStore
val Context.protoDataStore: DataStore<T> by dataStore(
  fileName = "fileName.pb",
  serializer = TSerializer
)
```

Ukázka kódu 4 Inicializace přístupu do úložiště *Preferences* a *Proto Data Store*

Čtení z úložiště se provádí v kontextu instance třídy *Context* pomocí klíče. Data jsou získána jako *Flow*.

```kotlin
// Preferences Data Store
val DATA_KEY = intPreferencesKey("my_data_key")
val exampleCounterFlow: Flow<Int> = context.dataStore.data
  .map { preferences ->
    preferences[DATA_KEY] ?: 0
}



// Proto Data Store
val exampleCounterFlow: Flow<Int> =
  context.settingsDataStore.data
    .map { settings ->
      settings.exampleCounter
    }
```

Ukázka kódu 5 Získání dat z *Preferences* a *Proto Data Store* pomocí jejich klíče

Zápis do Data Store úložiště pomocí funkce *edit* pro Preferences Data Store nebo *updateData* pro Proto DataStore.

```
// Preferences Data Store
context.dataStore.edit { settings ->
    settings[DATA_KEY] = 12
  }

// Proto Data Store
context.settingsDataStore.updateData { currentData ->
    currentData.toBuilder()
      .setData(10)
      .build()
}
```

Ukázka kódu 6 Uložení hodnot do Preferences a Proto Data Store pomocí klíče

Úložiště DataStore sice programátorovi poskytuje příjemnější přístup a zápis do úložiště, nicméně z hlediska bezpečnosti trpí stejným problémem, jako Shared Preferences. Data jsou uložena v čitelné podobě jako soubor v úložišti zařízení.

6.1.3 Získání dat ze zařízení s root právy

Data pro *Shared Preferences* a *DataStore* jsou uložena na disku jako soubor v privátním adresáři aplikace. Za standardních okolností do těchto souborů nemůže uživatel ani jiné aplikace přistupovat. Nicméně pokud má zařízení provedený tzv. root, který je popsaný v předchozí kapitole, lze k souboru s daty přistoupit jako k jakémukoliv veřejnému souboru v úložišti.

Soubory se nachází v cestě */data/data/<application-package-name>/shared_prefs*. Název souboru je pak *<application-package-name_<preferences-file-name>*. XML soubor s uloženými shared preferences pak vypadá následovně.

```
<?xml version='1.0' encoding='utf-8; standalone='yes' ?>
<map>
    <string name="key1">Data 1</string>
    <int name="key1" value="123" />
</map>
```

Ukázka kódu 7 XML soubor s uloženými daty v úložišti Shared Preferences

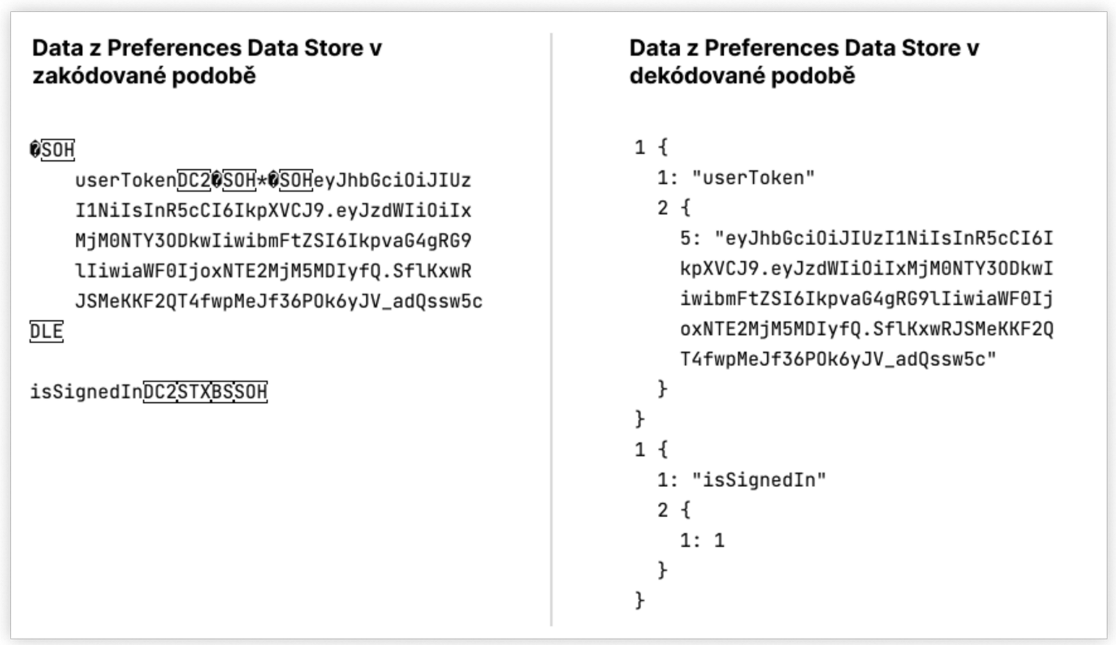
Uložení dat pro *Preferences Data Store* a *Proto Data Store* je trochu odlišné. Data se opět nacházejí v privátním adresáři aplikace, nicméně cesta k souborům se liší – */data/data/<application-package-name>/files/datastore.* I formát souboru je odlišný. Nejedná se totiž o formát XML, ale o vlastní formát *Protobuf* s příponou *preferences_pb*.

Soubor je po otevření v relativně nečitelné podobě, protože je zakódován. K dekódování souboru, aby byl čitelný pro člověka, lze použít program *protoc*. [7]

```
protoc --decode_raw < file_name.preferences_pb
```

Ukázka kódu 8 Dekódování soubory z *Protobuf* formátu do čitelné podoby

Po dekódování je již soubor pro útočníka jednoduše čitelný. Data lze nejen přečíst, ale i modifikovat a po opětovném zakódování je možné je vložit do zařízení a upravit tak chování aplikace.



Obrázek 4 Porovnání zakódovaných a dekódovaných dat pro *Preferences Data Store* [autor]

6.1.4 Získání dat ze zařízení bez root práv

I bez root přístupu do zařízení nejsou data v bezpečí. Lze je získat, pokud má aplikace v manifestu nastavený atribut *allowBackup*. Tento atribut se využívá k nastavení, zda má systém automaticky zálohovat nastavení a soubory aplikace na Google Drive (cloudové úložiště od společnosti Google). Pokud není explicitně stanoveno jinak, je tento atribut nastaven na hodnotu *true* – tedy, že se má automatické zálohování dat provádět. [8]

```
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
...>
     <application>
          android:allowBackup="true"
          ...
     </application>
    ...
</manifest>
```

Ukázka kódu 9 Ukázka nastavení atribut allowBackup v Manifestu aplikace

Potenciální útočník nemá k souborům uloženým na privátním Google Drive úložišti uživatele přístup, nicméně skrze nástroj *adb* může pomocí příkazu *adb backup* vytvořit zálohu celého zařízení nebo jednotlivé aplikace. Soubor, který vznikne je typu *tar*, je tedy možné z tohoto archivu data extrahovat. Extrahovaná data obsahují soubory zařízení a aplikace, včetně souborů obsahujících uložená data pomocí *Shared Preferences* nebo *Data Store*. Pro použití příkazu *adb backup* je nutné znát heslo telefonu používané k jeho odemknutí. [9]

Bránit se proti zneužití zálohy může programátor nastavením atributu *allowBackup* na hodnotu *false*. Tím zálohování úplně vypne a útočník tak při spuštění příkazu *adb backup* data aplikace nezíská. Uživatel nicméně přijde o možnost si data aplikace zálohovat, což může být nežádoucí.

Další možností je vyjmout citlivá data ze zálohy nastavením konfiguračního XML souboru pro atribut *fullBackupContent*. [8]

```
<manifest
  xmlns:android="http://schemas.android.com/apk/res/android">
     <application>
          android:allowBackup="true"
          android:fullBackupContent="@xml/backup_rules"
          ...
     </application>
   ...
</manifest>
```

Ukázka kódu 10 Ukázka nastavení atribut *allowBackup* a odkazu na konfigurační soubor pro upravení pravidel zálohy v Manifestu aplikace

Pomocí souboru *backup_rules.xml* a využití atributů *include* a *exclude* lze specifikovat, které soubory při záloze ukládat a které ne. [8]

```
<full-backup-content>
     <include domain="sharedpref" path="."/>
     <exclude domain="sharedpref" path="user-prefs.xml"/>
</full-backup-content>
```

Ukázka kódu 11 Ukázka XML konfiguračního souboru *backup_rules* pro upravení chování zálohy souboru s daty úložiště *Shared Preferences*

Aplikace pak nicméně musí být schopna tato data po prvním spuštění aplikace po obnovení ze zálohy získat znovu, což může být problematické.

Nejbezpečnější možností, která zároveň nemá vliv na nastavení zálohování, je data uložená pomocí *Shared Preferences* nebo *Data Store* šifrovat, jak je popsáno v následující podkapitole.

Dobrou zprávou také je, že společnost Google si toto riziko uvědomila a od systému Android verze 12 změnila chování příkazu *adb backup*. Při jeho použití dojde k odstranění veškerých privátních dat aplikací, takže útočník nemá možnost se k nim dostat. Programátor aplikace nicméně musí zvolit jako target SDK v API level verzi 31 nebo vyšší. Pokud zvolí starší target SDK, chování příkazu není omezeno a je tedy možné aplikační data získat.

6.1.5 Obrana – šifrování dat

Citlivé hodnoty, jako jsou například access tokeny, API klíče a další, je nutné chránit, a proto je nejlepší a nejbezpečnější možností je před uložením do *Shared Preferences* nebo *DataStore* zašifrovat, aby v případě získání souboru s uloženými daty nebylo možné data přečíst. Šifrování dat přidává při čtení i zápisu další komplexitu, která má za důsledek zvýšený čas vykonání dané operace. Nicméně pro citlivá data je tato nevýhoda bezesporu převážena rizikem jejich možného zneužití.

Pro třídu *SharedPreferences* existuje oficiální implementace pojmenovaná *EncryptedSharedPreferences*, která zajistí automatické šifrování dat při jejich uložení a automatické dešifrování při jejich čtení. Programátor tedy musí pouze využít místo třídy *SharedPreferences* třídu *EncryptedSharedPreferences*. [10]

Pro obě verze Data Store zatím bohužel neexistuje jejich oficiální šifrovaná implementace. Proto je nutné data před uložením do *Data Store* šifrovat manuálně a po přečtení je opět manuálně rozšifrovat. Vhodným šifrovacím algoritmem může být např. *AES (Advanced Encryption Standard)*, nicméně lze použít jakýkoliv jiný, dostatečně silný šifrovací algoritmus.

Potencionální útočník se v obou případech k datům sice dostane, avšak pouze v zašifrované podobě. Aby mohlo dojít k zneužití dat, musel by nejdříve data složitě rozšifrovat, což může být časově velice náročné až nereálné, podle zvoleného šifrovacího algoritmu.

6.2 Databáze

Pro ukládání komplexních mezi sebou propojených dat lze, stejně jako u programů na jiné platformy, využít databází. Pro systém Android existuje mnoho variant. Oficiální variantou je SQLite.

6.2.1 SQLite

Jedná se o databázi založenou na jazyce C a optimalizovanou pro ukládání malého množství dat. Je nejvíce využívaným databázovým enginem na světě. Používá se primárně pro mobilní zařízení, ale i počítače a mnohé další platformy. [11]

Pro dotazování nad SQLite databází se využívá standardní databázový jazyk SQL. Databáze nepodporuje veškeré operace, jako pokročilejší databázové systémy MySQL, Oracle Database a další, nicméně obsahuje většinu důležitých operací pro správu menších databází.

Pro využití databáze SQLite je nutné v aplikaci přidat závislost na knihovně sqlite. Ta poskytuje základní nástroje pro práci s SQLite databází. [11]

6.2.2 SQLite – zneužití a obrana proti *SQL injection*

Jako každá SQL databáze trpí SQLite na útok zvaný *SQL injection*. Jedná se o situaci, kdy útočník do zranitelného SQL dotazu vloží pomocí vstupního atributu svůj SQL dotaz a donutí tak databázi k akci, kterou původní SQL dotaz neměl vykonat. Pomocí tohoto útoku lze z databáze data získat, modifikovat je anebo poškodit obsah databáze. [12]

Pokud programátor dostatečně neošetřil uživatelské vstupy a manuálně modifikuje SQL dotaz přímým vkládáním vstupních atributů, vystavuje aplikaci riziku útoku SQL injection. Níže ukázaný dotaz slouží k získání detailu uživatele pomocí emailu a pinu.

```
SELECT * FROM users WHERE email = 'example@example.com' AND
pin = '1234' LIMIT 1
```

Ukázka kódu 12 SQL dotaz pro získání detailu uživatele na základě jeho e-mailu a PINu

Útočník ale může data o uživateli získat i bez znalosti pinu. Pokud místo pinu vyplní do vstupního pole text, který obsahuje jeho vlastní SQL dotaz (respektive jeho část), získá detail uživatele, aniž by jeho pin znal. Vložený škodlivý SQL kód může být například „''OR 1=1". Po zpracování uživatelského vstupu aplikace daný text vloží jako pin uživatel do svého SQL dotazu.

```
val email = "example@example.com"
val pin = "'' OR 1=1"
val query =  "SELECT * FROM users WHERE email = $email AND  pin
= $pin LIMIT 1")
```

Ukázka kódu 13 Ukázka zranitelného přístupu do databáze sestavením SQL dotazu a manuálním dosazením uživatelského vstupu

Výstupem je pak dotaz, který vypadá následovně:

```
SELECT * FROM users WHERE email = 'example@example.com' AND
pin = '' OR 1=1 LIMIT 1
```

Ukázka kódu 14 SQL dotaz s vloženými uživatelskými vstupy zneužívající útok *SQL injection*

Na dotazu výše je názorně vidět, že zadaný „pin" se do dotazu nevložil pouze jako text, ale dotaz modifikoval o přidání OR podmínky (podmínka nebo). Po spuštění dotazu databáze vyhodnotí, že má vrátit data o uživateli za podmínky, že se shoduje email a zároveň se shoduje pin anebo platí, že 1 = 1, což platí pokaždé. Dojde tedy k získání dat uživatele i přesto, že zadaný pin se neshoduje se správným pinem.

Základní obranou proti *SQL injection* je využívání tzv. *PreparedStatement* objektů. Jedná se o předem kompilované SQL dotazy, do kterých jsou vložené parametry. Ty jsou do dotazu vkládány až po kompilaci, tedy vždy jako hodnoty. Nemohou být tedy interpretovány jako SQL dotaz a nemohou modifikovat přechozí, již zkompilovanou, podobu SQL dotazu.

```
val statement: PreparedStatement = dbConnetion.prepareStatement(
    "SELECT * FROM users WHERE email = ? AND
    pin = ? LIMIT 1"
).apply {
    setString(1, "example@example.com")
    setString(2, "'' OR 1=1")
}
```

Ukázka kódu 15 Využití PreparedStatement pro dosazení uživatelských vstupů do SQL dotazu

Vložený text „’OR 1=1” je v tomto případě považován pouze za vstupní hodnotu a je tedy porovnán vůči uživatelově pinu v textové podobě. Pin se zadaným textem se neshodují, a tak databáze nevrátí žádný výsledek.

Aktuálně je doporučovaným přístupem pro práci s SQLite databází v systému Android knihovna Room z oficiální kolekce knihoven Jetpack Compose. Jedná se o abstraktní vrstvu nad SQLite databází umožňující efektivnější a flexibilnější práci s uloženými daty. Jednou z výhod knihovny je při jejím správném užití i obrana proti *SQL injection*. Využívají se opět předem kompilované dotazy. [13]

6.2.3 SQLite – zneužití a obrana proti čtení dat ze souboru

Soubory SQLite databáze jsou uložené v privátním adresáři aplikace – konkrétně v */data/data/<application-package-name>/databases*. Pojmenování souboru s daty databáze se shoduje s názvem databáze zvoleným v aplikaci. Kvůli uložení souborů v privátním adresáři aplikace lze využít stejných zranitelností jako pro úložiště typu *Shared Preferences* a *Data Store*.

Soubory databáze v zařízení nejsou v základu šifrované, a proto je možné je po jejich získání jednoduše otevřít v některé SQLite aplikaci – například DB browser for SQLite. [14]

K souborům databáze za normálních okolností nelze přistupovat, protože jsou v privátním úložišti aplikace v adresáři */data/data/<application-package-name>/databases*/{název databáze}. Toto omezení lze ale obejít pomocí root přístupu do telefonu.

Další možností získání souborů s daty databáze je zneužití automatických záloh telefonu, pokud má aplikace nastavený atribut v manifestu *allowBackup* na *true*. Jedná se o zneužití stejného principu, jako je detailněji popsáno v kapitole 6.1.4 (Získání dat z *Shared Preferences* nebo z *Data Store* v zařízení bez root práv). Obranou je buď nastavení atributu *allowBackup* na hodnotu *false*, čímž ale dojde k znemožnění zálohování aplikace na Google Drive cloudové úložiště. Druhou možností je vyjmutí databázových souborů ze zálohy aplikace. To lze provést pomocí nastavení v konfiguraci XML souboru pro atribut *fullBackupContent*. [8] Viz **Ukázka kódu 10** Ukázka nastavení atribut *allowBackup* a odkazu na konfigurační soubor pro upravení pravidel zálohy v Manifestu aplikace.

Pomocí souboru *backup_rules.xml* a využití atributů *include* a *exclude* lze upravit nastavení, zda soubory databáze při záloze ukládat, nebo ne. [8]

```
<full-backup-content>
	<include domain="database" path="."/>
	<exclude domain="database" path="app-database"/>
</full-backup-content>
```

Ukázka kódu 16 Ukázka XML konfiguračního souboru *backup_rules* pro upravení chování zálohy souboru s daty SQLite databáze

Aplikace po prvním spuštění po obnovení ze zálohy musí být schopna data nezahrnutá v záloze získat znovu, což může být problematické.

Obranou proti získání obsahu databáze ze souboru je tedy pouze šifrování dat databáze. Databáze v základu neposkytuje možnost šifrování dat, proto je třeba data buď manuálně před zápisem šifrovat anebo využít dalších nástrojů třetích stran, jako například open source projekt Android Database SQLite Cipher (*https://github.com/sqlcipher/android-database-sqlcipher*), který šifrování databáze zajistí automaticky.

6.2.4 Alternativní databáze

Pro ukládání dat do databáze lze místo SQLite využít alternativy, jakými jsou například Real, Cupboard, Firestore a další. Každá z těchto databází může obsahovat svá bezpečnostní rizika. Vzhledem k tomu, že alternativních databází je velké množství, tak se tato práce jejich riziky nezabývá a poukazuje pouze na rizika spojená s použitím oficiálně doporučované databáze SQLite.

6.3 Soubory

Pro uložení větších souborů, jako jsou například obrázky nebo dokumenty, není vhodné využití databáze ani úložiště typu klíč – hodnota. Programátor aplikace si může zvolit z několika adresářů pro uložení souborů. Každý z adresářů má jinou úroveň zabezpečení, viditelnost pro ostatní aplikace a rozdílný přistup pro práci s uloženými soubory. Je klíčové, aby programátor správně zvolil adresář podle povahy dat, která do něj plánuje ukládat, jinak by mohlo dojít k odhalení citlivých soborů ostatním aplikacím anebo naopak nemožnosti ostatních aplikací nebo uživatele k souborům přistupovat, i když tak bylo původně zamýšleno.

6.3.1 Privátní soubory aplikace

Pro ukládání citlivých souborů je nejlepší volbou privátní úložiště aplikace. K tomuto úložišti má přístup pouze vlastnická aplikace a této aplikaci je vždy k dispozici bez nutnosti od uživatele získávat oprávnění na přístup k úložišti zařízení. Přístup k adresáři je zajištěn skrze metody na instanci třídy *Context*.

```
// Privátní soubory aplikace
val appFile = context.filesDir

// Privátní cache soubory aplikace
val appCacheFile = context.cacheDir
```

Ukázka kódu 17 Ukázka přístupu do privátního úložiště aplikace

K souborům uloženým v privátním úložišti sice za normálních okolností nelze přistupovat, nicméně toto omezení lze obejít pomocí root přístupu do telefonu. Provedení rootu zařízení je popsáno v kapitole 5 Root zařízení. Po získání root přístupu je již možné tyto privátní soubory získat stejně, jako by byly veřejně přístupné.

Dalším potenciálním rizikem jsou už několikrát zmiňované automatické zálohy aplikace nebo telefonu, pokud má aplikace nastavený atribut v manifestu *allowBackup* na *true*. Jedná se o zneužití stejného principu, jako úložiště *Shared Preferences* nebo *Data Store*. Obrana je opět totožná – buď nastavení atributu *allowBackup* na hodnotu *false*, čímž ale dojde k znemožnění zálohování aplikace na Google Drive cloudové úložiště nebo vyjmutí uložených souborů ze zálohy aplikace. To lze provést pomocí nastavení v konfiguraci XML souboru pro atribut *fullBackupContent*. [8] Viz Ukázka kódu 10 Ukázka nastavení atribut *allowBackup* a odkazu na konfigurační soubor pro upravení pravidel zálohy v Manifestu aplikace.

Pomocí souboru *backup_rules.xml* a využití atributů *include* a *exclude* lze upravit nastavení, zda privátní soubory aplikace při záloze ukládat, nebo ne. [8]

```xml
<full-backup-content>
	<include domain="file" path="."/>
	<exclude domain="file" path="backup.csv "/>
</full-backup-content>
```

Ukázka kódu 18 Ukázka XML konfiguračního souboru *backup_rules* pro upravení chování zálohování privátních souborů aplikace

Výkonově i implementačně náročnějším řešením je zašifrování souborů ukládaných do úložiště. Lze využít například *AES (Advanced Encryption standard)* šifrování.

6.3.2 Veřejné soubory aplikace

Pro ukládání souborů aplikace, které nejsou přímo určeny pro čtení ostatními aplikacemi nebo uživatelem, ale nevadí, nebo je naopak žádoucí, aby k nim měl uživatel nebo jiné aplikace přístup, je vhodné využít externí úložiště aplikace. Vhodné může být například pro uložení konfiguračních souborů, které jsou do zařízení nahrávány manuálně. Jedná se tedy o úložiště aplikace, k jehož souborům je ale veřejný přístup.

Přístup k adresáři je zajištěn skrze metody instance třídy *Context*.

```kotlin
// Veřejné soubory aplikace
val appFile = context.getExternalFilesDir(null)

// Veřejné cache soubory aplikace
val appCacheFile = context.externalCacheDir
```

Ukázka kódu 19 Ukázka přístupu do veřejného úložiště aplikace

Je nutné dát pozor na to, že i přes to, že se jedná o úložiště aplikace, mohou k němu ostatní aplikace přistupovat v případě, že mají povolení číst úložiště zařízení. Proto není vhodné pro ukládání citlivých souborů.

6.3.3 Média

Uložení médií, jako jsou obrázky, videa nebo audiozáznamy, které jsou určeny pro sdílení i s ostatními aplikacemi, se realizuje pomocí rozhraní *Media Store*. Soubory jsou ukládány do sdílených adresářů určených pro daný *MIME* typ – tedy např. *image/**, *audio/** a podobně. I přes to, že cílem je usnadnit a standardizovat sdílení mediálních souborů, nemělo by se rozhraní *Media Store* využívat pro mediální soubory obsahující citlivá data, která nemají být sdílena mimo aplikaci, protože by došlo k jejich odhalení ostatním aplikacím. V tomto případě je vhodné využít privátní úložiště aplikace, jako je popsáno v kapitole 6.3.1 Privátní soubory aplikace.

6.3.4 Sdílené úložiště

Pro přístup do sdíleného úložiště se dříve využívalo získání povolení od uživatele za běhu aplikace na čtení a zápis do úložiště, konkrétně *READ_EXTERNAL_STORAGE* a *WRITE_EXTERNAL_STORAGE*. Tato povolení nicméně představovala určitou bezpečnostní hrozbu, protože pokud je aplikace získala, mohla číst veškerá sdílená data v úložišti telefonu. To by z podstaty věci nemuselo být špatně, nicméně uživatel neměl žádnou kontrolu nad tím, co aplikace v úložišti dělá a k jakým souborům přistupuje. To mohlo vést v případě škodlivé aplikace ke skenování úložiště a hledání potenciálně zneužitelných souborů. Proto bylo udělování povolení k přístupu ke všem souborům omezeno a od verze Android 10 standardní aplikace získá povolení pouze k práci s tzv. *scoped storage*. To umožňuje práci pouze s médii. Pro práci s dokumenty je nutné využít *Storage Access Framework*. Ten ale po uživateli vyžaduje pokaždé přesně vybrat, ke kterým souborům aplikace může přistupovat nebo případně kam data může uložit.

Od verze Android 11 (vydaného roku 2020) bylo povolení *WRITE_EXTERNAL_STORAGE* označeno jako zastaralé (deprecated) a i když o něj aplikace zažádá, nebude mít žádný efekt a aplikace přístup k souborům ve sdíleném úložišti nezíská.

7 Autentizace a autorizace

Aplikace v dnešní době neslouží pouze k jednoduchým úkonům, ale může se jednat o aplikace mobilního bankovnictví, správu pojištění, e-shopu s uloženou platební kartou a další, které vyžadují vyšší míru zabezpečení. Je zřejmé, že při prvním spuštění je nutné uživatele autentizovat a dovolit tak aplikaci přístup k jeho účtu. Nicméně je nutné vyřešit chování, které následuje po zavření aplikace a jejím opětovném spuštění. Nejbezpečnějším způsobem by bylo uživatele vždy po uzavření aplikace odhlásit. To zamezí situaci, že telefon bude používat někdo jiný a získá tak plný přístup k aplikaci a provedení zásadních operací, jako jsou například platby kartou. Toto chování ale nekoresponduje s tím, co uživatel očekává od mobilní aplikace.

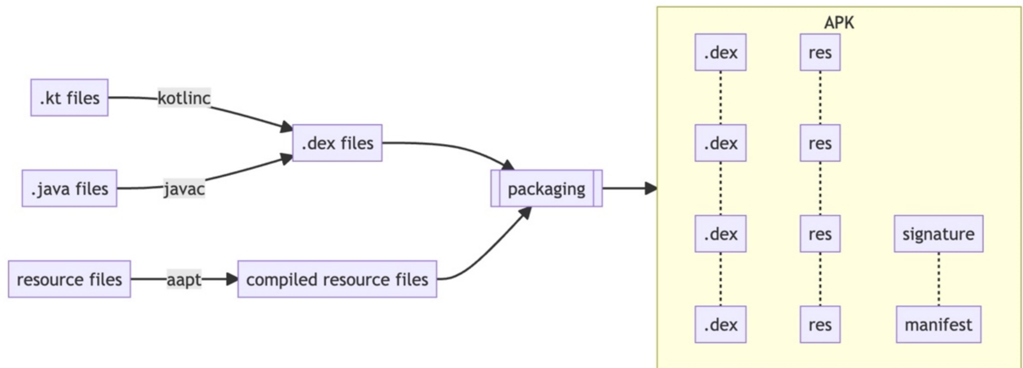
Standardem je, že aplikace si přihlášení uživatele uloží a při opětovném spuštění aplikace ho již po uživateli nevyžaduje. Toto chování ale představuje riziko. V praxi se osvědčilo používání jednodušší formy zabezpečení jako alternativy ke klasickému přihlášení. Tu může představovat například použití pinu, otisku prstu nebo rozpoznání obličeje. Jedná se pro uživatele o znatelně příjemnější a jednodušší formu ověření, která ale stále minimalizuje hrozbu zneužití pouhým získáním přístupu do telefonu. Dobrým zvykem je také toto jednodušší ověření vyžadovat pokaždé, když chce uživatel v aplikaci provést zásadnější operaci. Příkladem může být u bankovní aplikace převod prostředků na cizí účet.

Uživatel by měl mít také možnost mobilní zařízení na dálku odhlásit od svého účtu, například pomocí webového rozhraní nebo přihlášením do aplikace na jiném zařízení. Důvodem je zamezení možnému zneužití zařízení a v něm nainstalované aplikace v případě, že dojde k jeho ztrátě nebo odcizení.

8 Dekompilace aplikace

Dekompilace aplikace je jednou ze základních technik, která se používá pro zjištění funkčnosti aplikace, nalezení chyb a umožnění tak jejího napadení.

Dekompilací se rozumí opačný proces ke kompilaci, kdy je kompilovaný nízkoúrovňový zdrojový kód převeden zpět na kód ve vyšším programovacím jazyce, ve kterém pravděpodobně byla aplikace naprogramována.

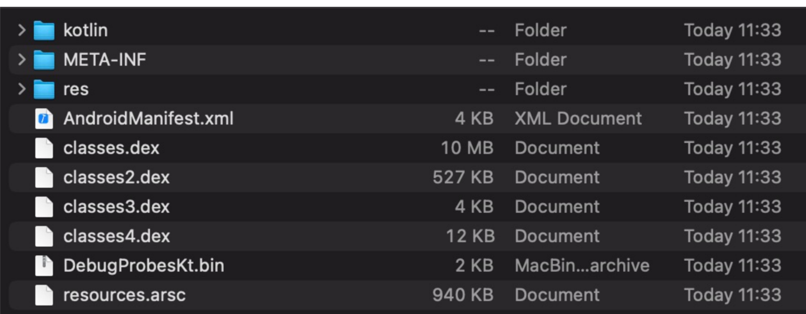


Obrázek 5 Proces kompilace aplikace [15]

Pro dekompilaci Android aplikace je nutné mít přístup k APK instalačního souboru aplikace. Pokud je aplikace publikována na Google Play, tak lze APK soubor získat s využitím různých nástrojů. Jedním z nich je například online služba *https://apkcombo.com/downloader/*.

APK soubor obsahuje veškerá potřebná data pro instalaci aplikace do telefonu – kompilované zdrojové kódy aplikace a použitých knihoven, verze knihoven, konfigurační soubor Android manifest, použité obrázky a další. APK soubor si lze představit jako ZIP soubor, který lze rozbalit, např. pomocí programu unzip. Po rozbalení souboru je možné vidět strukturu uložených dat, kdy ale veškeré zdrojové soubory jsou v kompilované podobě. Nelze číst ani konfigurační *AndroidManifest.xml* soubor. Samotné zdrojové kódy jsou uložené v souboru *classess.dex*, v případě překročení DEX limitu metod, který čítá 65 636, jsou kompilované zdrojové kódy rozděleny do více souborů s číselným sufixem,

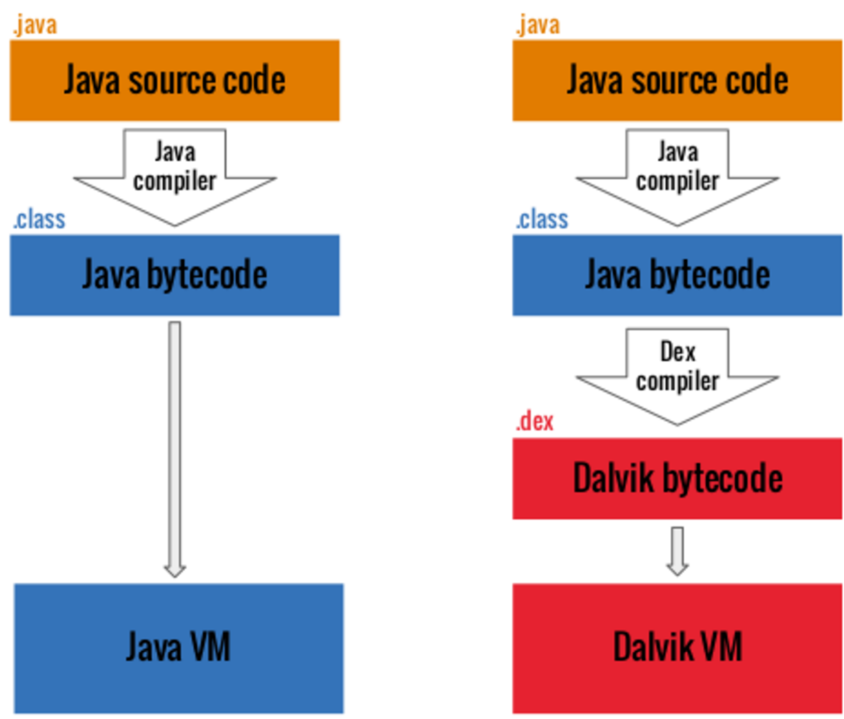
např. *classess.dex* a *classess2.dex*. V DEX souborech se nachází kompilovaný zdrojový kód určený pro spuštění na Dalvik VM (Android alternativa Java VM) a je pro člověka v nečitelné podobě.



Obrázek 6 Znázornění obsahu APK souboru po rozbalení programem unzip [autor]

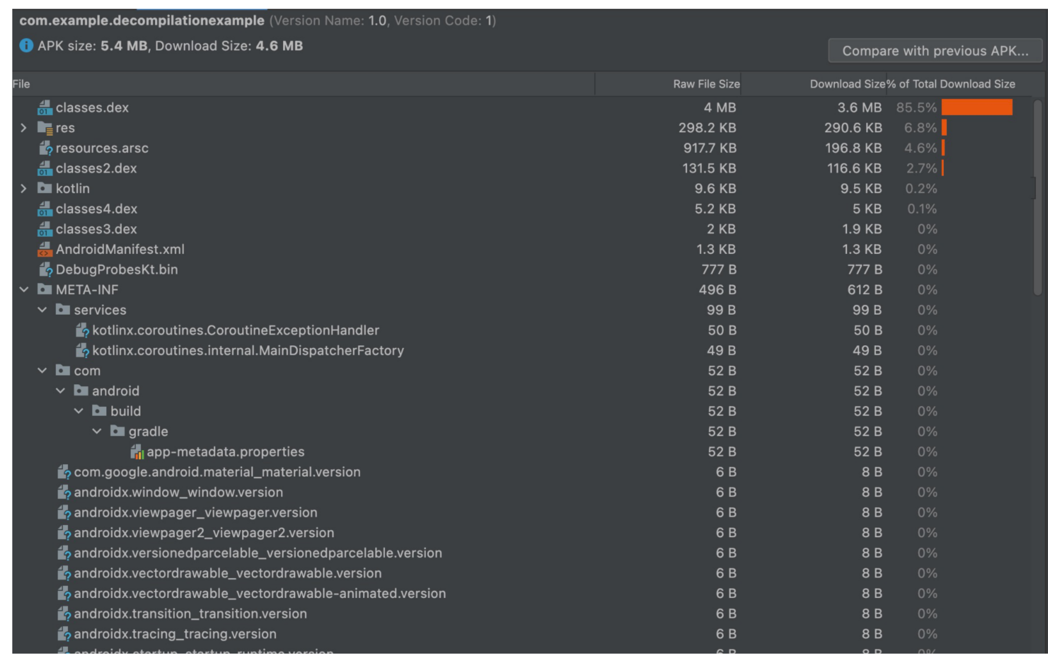

```
AndroidManifest.xml 〉 No Selection
X1   6Ù(4VnÇú™»,¸ :Nhé¥⁄:dnvÜö¨%(x”2V®‡,@Ñxfi6Th~ê®‹$@bthemelabeliconnameprotectionLevel
debuggableexported
    authoritiesvalue
minSdkVersion
    versionCode
    versionNametargetSdkVersion
    allowBackuprequired
    supportsRtlextractNativeLibsfullBackupContentcompileSdkVersioncompileSdkVersionCodenameappComponentFactorydataExtractionRules1
    .013actionactivityandroidandroid.intent.action.MAIN
    android.intent.category.LAUNCHER&androidx.core.app.CoreComponentFactory+androidx.emoji2.text.EmojiCompatInitializer.androidx.lifecycle
    .ProcessLifecycleInitializerandroidx.startup'androidx.startup.InitializationProviderandroidx.window.extensionsandroidx.window
    .sidecar
    applicationcategory
    com.example.decompilationexampleIcom.example.decompilationexample.DYNAMIC_RECEIVER_NOT_EXPORTED_PERMISSION—com.example.decompilationexample
    .MainActivity1com.example.decompilationexample.androidx-startup*http://schemas.android.com/apk/res/android
intent-filtermanifest   meta-datapackage
permissionplatformBuildVersionCodeplatformBuildVersionNameprovider
    uses-libraryuses-permissionuses-sdkÄ`
    $
    pÄéØÍÎrsz>˘˘˘˘*∞˘˘˘˘˘˘˘˘,*
˘˘˘˘*
    *˘˘˘˘!*˘˘˘˘.&&˘˘˘˘0˘˘˘˘!˘˘˘˘1˘˘˘˘
L˘˘˘˘˘˘˘˘5*
    ˘˘˘˘*
    ˘˘˘˘!˘˘˘˘˘˘˘˘5L
    ˘˘˘˘˘˘˘˘/*''*˘˘˘˘
    ˘˘˘˘˘˘˘˘/8˘˘˘˘˘˘˘˘4*''˘˘˘˘˘˘˘˘4Ï˘˘˘˘˘˘˘˘$
*˘˘˘˘*˘˘˘˘*˘˘˘˘*˘˘˘˘˘˘˘˘*
˘˘˘˘˘˘˘˘*˘˘˘˘˘˘˘˘*˘˘˘˘*˘˘˘˘**˘˘˘˘t˘˘˘˘˘˘˘˘*˘˘˘˘*˘˘˘˘*((*˘˘˘˘˘˘˘˘$!˘˘˘˘˘˘˘˘+8"˘˘˘˘˘˘˘˘*"˘˘˘˘˘˘˘˘8$˘˘˘˘˘˘˘˘%*$˘˘˘˘˘˘˘˘%!˘˘˘˘˘˘˘˘+˘˘˘˘˘˘˘˘L
    (˘˘˘˘˘˘˘˘3*""*˘˘˘˘(˘˘˘˘˘˘˘˘3L+˘˘˘˘˘˘˘˘3*##*˘˘˘˘+˘˘˘˘˘˘˘˘3`/˘˘˘˘˘˘˘˘2*!!*˘˘˘˘*))L3˘˘˘˘˘˘˘˘-**  3˘˘˘˘˘˘˘˘-L6˘˘˘˘˘˘˘˘-**
    6˘˘˘˘˘˘˘˘-/˘˘˘˘˘˘˘˘2˘˘˘˘˘˘˘˘$˘˘˘˘˘˘˘˘,˘˘˘˘*
```


Obrázek 7 Nečitelný obsah AndroidManifest.xml souboru po rozbalení APK souboru programem unzip [autor]



Obrázek 8 Rozdíl kompilace klasické Java aplikace a Android aplikace [16]

Pokročilejším způsobem rozbalení APK souboru je využití funkce Android Studia – Analyze APK. Ta zajistí dekompilaci části souborů, jako např. XML, a tak je možné získat větší, ale stále velmi omezené množství informací o aplikaci. Primárně ze souboru AndroidManifest.xml, který obsahuje metadata jako seznam aktivit a služeb aplikace, registrované tzv. „broadcast receivers", povolení aplikace a další. Nicméně samotné zdrojové kódy aplikace jsou stále v nečitelné podobě.



Obrázek 9 Znázornění obsahu APK souboru pomocí Android Studia [autor]

Pro možnost čtení zdrojových souborů je nutné provést jejich dekompilaci. Jednou z možností je provést ji pomocí nástroje apktool (*https://ibotpeaches.github.io/Apktool/*). Jeho použití je jednoduché, na vstupu stačí poskytnout pouze cestu k APK souboru aplikace a program spustit z příkazového řádku. [17]

```
apktool d application.apk -o ./output/new-directory
```

Ukázka kódu 20 Ukázka spuštění příkazu apktool pro dekompilaci aplikace ze souboru *application.apk* do souboru *./output/new-directory*

Po provedení tohoto příkazu je výstupem adresář s daty a dekompilovaným zdrojovým kódem ve formátu „smali“. Jedná se o již čitelnější soubory, nicméně stále nejde o originální zdrojové kódy. Avšak tato úroveň dekompilace již např. umožňuje číst a také upravit konfigurační soubor *AndroidManifest.xml* a nastavit aplikaci jako *debuggable*, což otevírá spoustu možností zkoumání aplikace. Opětovná kompilace aplikace v tomto stavu je velice jednoduchá pomocí následujícího příkazu ve složce s dekompilovanými a případně upravenými zdrojovými soubory.

```
apktool b application.jar -o ./output-destination
```

Ukázka kódu 21 Ukázka spuštění příkazu apktool pro kompilaci aplikace ze souboru *application.jar* do souboru APK

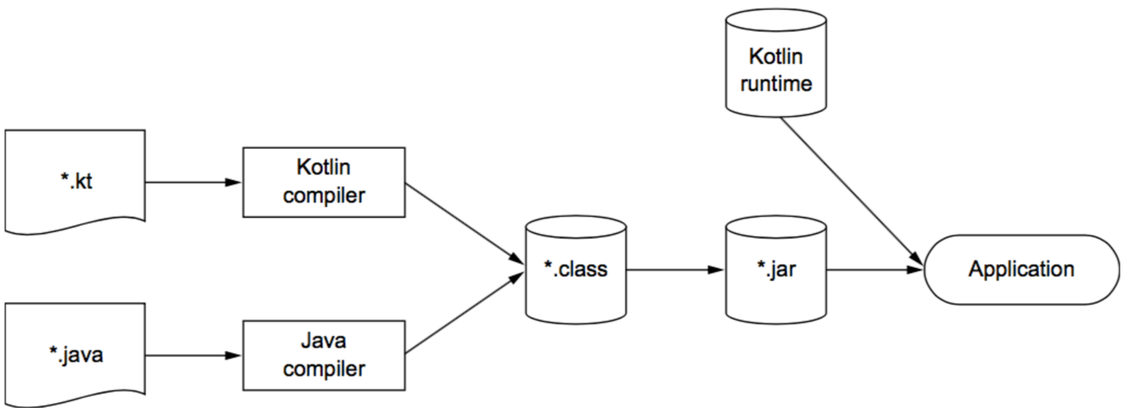
Nicméně pro účely *reverse engineeringu* aplikace je nutné převést zdrojové kódy do původního jazyku Java nebo Kotlin. Na to lze využít program dexToJar, který lze spustit opět jednoduchým příkazem. [18]

```
./dex-tools-2.1/d2j-dex2jar.sh  -f -o app.jar ./app.apk
```

Ukázka kódu 22 Příkaz na spuštění programu dexToJar pro dekompilaci zdrojových kódů aplikace v souboru app.apk do jazyka Java nebo Kotlin

Výstupem je jar soubor se zdrojovými kódy v java bytecode s příponou *.class*. Pro převod zdrojových souborů na čitelný zdrojový kód s příponou *.java* lze využít např. program JD-GUI. Vstupem je pouze převedený jar soubor. V GUI je poté zobrazen obsah .class souboru převedený na .java soubor.

Pokud aplikace byla naprogramována v jazyce Kotlin, tak i přesto lze využít předchozí metodu. Nicméně výstupem bude Java kód. Důvodem je, že jazyky Kotlin a Java používají pro svůj běh *Java Virtual Machine* a jsou tedy kompilované do stejného Java bytecode, pouze využívají rozdílné kompilátory. Jazyky jsou také mezi sebou interoperabilní, což umožňuje je v jednom zdrojovém kódu kombinovat a navzájem využívat. Tyto vlastnosti vysvětlují, proč lze dekompilovat Android aplikaci napsanou v jazyce Kotlin do jazyka Java.



Obrázek 10 Znázornění obsahu APK souboru pomocí Android Studia [19]

```
// Z kódu je pro přehlednost použita pouze jedna ukázková metoda

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        setSupportActionBar(binding.toolbar)

        val navController =
findNavController(R.id.nav_host_fragment_content_main)
        appBarConfiguration =
AppBarConfiguration(navController.graph)
        setupActionBarWithNavController(navController,
appBarConfiguration)



        binding.fab.setOnClickListener { view ->
            Snackbar.make(view, "Replace with your own action",
Snackbar.LENGTH_LONG)
                .setAction("Action", null).show()
        }
    }
```

Ukázka kódu 23 Původní zdrojový kód v jazyce Kotlin využitý pro ukázku dekompilace do jazyka Java v následující ukázce kódu

```
  protected void onCreate(Bundle paramBundle) {
    super.onCreate(paramBundle);
    ActivityMainBinding activityMainBinding3 =
ActivityMainBinding.inflate(getLayoutInflater());
    Intrinsics.checkNotNullExpressionValue(activityMainBinding3,
expression: "inflate(LayoutInflater)");
    this.binding activityMainBinding3;
    ActivityMainBinding activityMainBinding4 = null;
    ActivityMainBinding activityMainBinding2 =
activityMainBinding3;
    if (activityMainBinding3 = null) {

Intrinsics.throwUninitializedPropertyAccessException("binding");
        activityMainBinding2 = null;
    }
    setContentView((View) activityMainBinding2.getRoot());
    activityMainBinding3 = this.binding;
    activityMainBinding2 = activityMainBinding3;
    if (activityMainBinding3 = null) {

Intrinsics.throwUninitializedPropertyAccessException("binding");
        activityMainBinding2 = null;
    }

    setSupportActionBar(activityMainBinding2.toolbar);
    NavController navController =
ActivityKt.findNavController((Activity) this, viewId: 2131231031);
    NavGraph navGraph = navController.getGraph();
    MainActivity$onCreate$$inlined$AppBarConfiguration$default$1
    mainActivity$onCreate$$inlined$AppBarConfiguration$default$1
        =
MainActivity$onCreate$$inlined$AppBarConfiguration$default$1.INSTANCE
;
    AppBarConfiguration appBarConfiguration2
        = (new
AppBarConfiguration.Builder(navGraph)).setOpenableLayout(null).setF
allbackOnNavigateUpListener(new
MainActivity$inlined$sam$i$androidx_navigation_ui_AppBarConfigurati
on_OnNavigateUpListener$0(

mainActivity$onCreate$$inlined$AppBarConfiguration$default$1)).build(
);
    this.appBarConfiguration = appBarConfiguration2;
    MainActivity mainActivity = this;
    AppBarConfiguration appBarConfiguration1 =
appBarConfiguration2;

   // Kód metody je pro přehlednost zkrácen
  }
```

Ukázka kódu 24 Část výstupu dekompilace aplikace psané v jazyce Kotlin dekompilované do jazyku Java

Dále existují alternativní dekompilátory, které převádí java bytecode do zdrojového kódu jazyka Kotlin. Příkladem je open source projekt kotlin-decompiler dostupný v online repozitáři na serveru GitHub [20]. Nicméně tyto dekompilátory nejsou zdaleka tak rozšířené, jako pro jazyk Java.

9 Obfuskace zdrojového kódu

Instalační balíček APK lze dekompilovat a je tak možné procházet a číst zdrojový kód aplikace, jak je popsáno v kapitole 8 Dekompilace aplikace. To představuje riziko, protože kód aplikace by měl být v ideálním případě psán přehledně a po jeho přečtení by mělo být zřejmé, jak funguje. V rámci vývoje je toto žádoucí, nicméně pro útočníky to odhaluje možnosti zneužití aplikace. Tím může být například nalezení zneužitelné programátorské chyby, přístup k citlivým údajům jako jsou API klíče, nebo umožnění jednoduché úpravy aplikace, aby se chovala jinak, než bylo zamýšleno a například zobrazila citlivá data nebo odesílala pozměněné požadavky na server. Dekompilaci zdrojového kódu aplikace z balíčku APK nelze zabránit. Je tedy nutné hledat způsoby, jak co nejvíce ztížit rozklíčování zdrojového kódu a odhalení, jak aplikace funguje. K tomu slouží tzv. obfuskace kódu. Ta se provádí před kompilací kódu a umožňuje jeho znepřehlednění pomocí přejmenování všech tříd, atributů, metod a proměnných na názvy, které z pohledu programátora nedávají smysl.

Těmito kroky dojde k obrovskému znepřehlednění zdrojového kódu až do míry, kdy je velice složité až nemožné ho číst. Potencionální útočník by musel aplikaci tzv. krokovat („debugovat“) a zjišťovat, co která třída, metoda nebo atribut znamenají a dělají, což by bylo časově velice náročné.

Na platformě Android byl dříve pro obfuskaci kódu používán nástroj Proguard, nicméně od verze Android Gradle pluginu 3.4.0 (duben 2019) je používán nástroj R8 compiler [21].

Oba nástroje nad rámec obfuskace poskytují další funkce:

- Zmenšení velikosti kódu smazáním komentářů a nepoužitého nebo nedosažitelného kódu.
- Odstranění nepoužitých obrázků, knihoven a dalších zdrojů.
- Optimalizace a zmenšení kompilovaných DEX souborů.

Obrázek 11 Porovnání struktury balíčků po dekompilaci mezi obfuskovaným a neobfuskovaným kódem [autor]

```java
public final Method c(String paramString) {
  b<String, Method> b1 = this.a;
  Method method1 = (Method)b1.getOrDefault(paramString, null);
  Method method2 = method1;
  if (method1 == null) {
    System.currentTimeMillis();
    method2 = Class.forName(paramString, true, a.class.getClassLoader()).getDeclaredMethod("read", new Class[] { a.class });
    b1.put(paramString, method2);
  }
  return method2;
}

public final Method d(Class<? extends c> paramClass) {
  String str = paramClass.getName();
  b<String, Method> b1 = this.b;
  Method method2 = (Method)b1.getOrDefault(str, null);
  Method method1 = method2;
  if (method2 == null) {
    Class clazz = b(paramClass);
    System.currentTimeMillis();
    method1 = clazz.getDeclaredMethod("write", new Class[] { paramClass, a.class });
    b1.put(paramClass.getName(), method1);
  }
  return method1;
}
```

Obrázek 12 Ukázka dekompilovaných obfuskovaných metod v jazyce Java zobrazených v programu JD-GUI [autor]

9.1 Rizika při nepoužití obfuskace kódu v Android aplikaci

Situace, kdy je aplikace publikována bez použití nástrojů Proguard nebo R8 při vytváření APK souboru aplikace, představuje velké riziko.

9.1.1 Zobrazení debug informací

Jednou z funkcí výše zmiňovaných nástrojů je odstranění nedosažitelného kódu. Na platformě Android se typicky jedná o kód sloužící k logování stavu aplikace, který je obalen v podmínce, aby došlo k jeho vykonání pouze v případě, že je aplikace v tzv. debug módu.

```
if (BuildConfig.DEBUG) {
     Log.d("Payment". "Received tokenized card - $cardToken");
}
```

Ukázka kódu 25 Podmínka pro zápis logu do konzole pouze v případě, že je aplikace spuštěna v debug módu

Při nepoužití výše zmíněných nástrojů při kompilaci nedojde k odstranění těchto částí kódu. Kód sice není při spuštění verze aplikace, která není označena jako debug varianta, spuštěn, nicméně toto chování je možné v nastavení manifestu aplikace upravit, a to konkrétně nastavením aplikace, aby se jednalo o debug variantu.

```
<manifest
  xmlns:android="http://schemas.android.com/apk/res/android"
  ... >
     <application>
          android:debuggable="true"
          ...
     </application>
  ...
</manifest>
```

Ukázka kódu 26 Nastavení aplikace pro spuštění v debug módu v Manifestu

Při opětovné kompilaci aplikace s tímto nastavením a jejím spuštění začnou být tato data vypisována do konzole. To může způsobit odhalení citlivých informací, ale primárně také umožnit jednodušší zjištění, jak aplikace funguje, komunikuje se serverem a nakládá s daty. Rozsah možných škod závisí na detailnosti a strukturovanosti logování, kterou programátor dané aplikace zvolil.

9.1.2 Zjištění funkčnosti aplikace a její modifikace

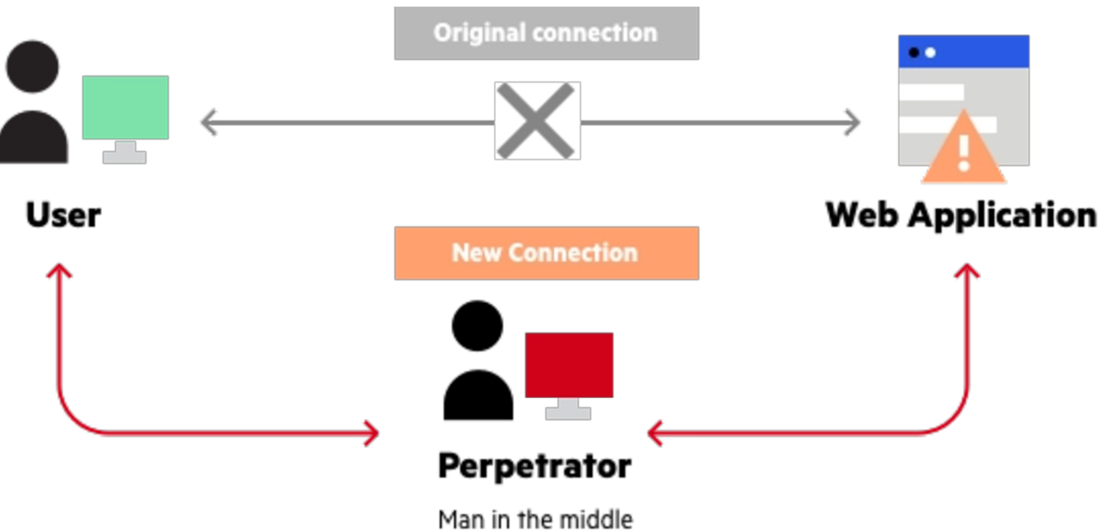
Při dekompilaci neobfuskované aplikace je kód čitelný úplně stejně, jako kdyby měl útočník přístup přímo k jeho zdrojovým kódům. Orientace v kódu, jeho pochopení a možné odhalení zranitelností je pak poměrně jednoduché. V aplikaci je pak možné najít bezpečnostní chyby, které lze využít pro získání citlivých dat uživatelů. Kód aplikace lze také upravit a vložit do něj škodlivý zdrojový kód a aplikaci pak distribuovat uživatelům mimo Google Play. Aplikace se pak tváří jako originální verze, nicméně na pozadí může např. odesílat citlivá data uživatelů na servery útočníka.

10 Bezpečnost síťové komunikace

Systém Android od nových verzí znatelně omezuje nezabezpečenou komunikaci. Například pro možnost použití nešifrovaného přenosu dat přes protokol HTTP je nutné tuto skutečnost explicitně povolit v manifestu aplikace. I přes tyto snahy ale systém nemůže zabránit odposlechnutí nebo dokonce úpravě komunikace. O obranu před tímto útokem se musí starat samotná aplikace.

10.1 Odposlech nebo úprava síťové komunikace

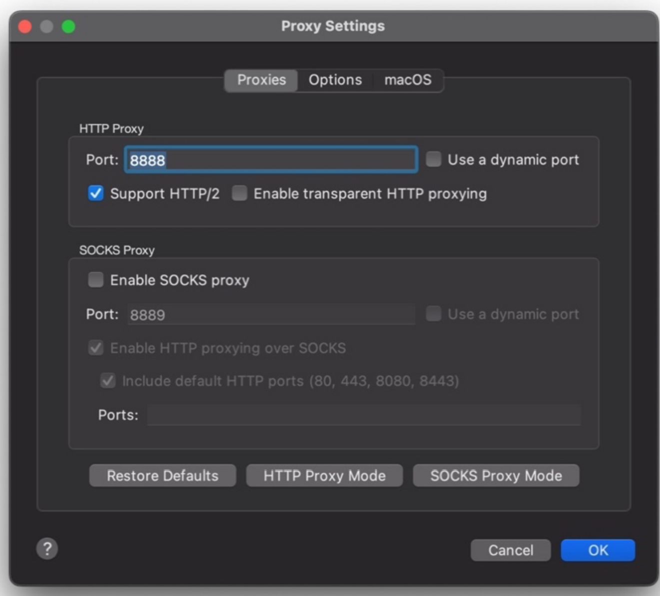
Android aplikace v drtivé většině případů komunikují se serverem pomocí standardu HTTP nebo HTTPS. Komunikaci s využitím HTTP je z povahy protokolu velice jednoduché odchytit a případně upravit. HTTPS tuto skutečnost poměrně komplikuje, nicméně i tak lze data v určitých případech přečíst. HTTPS je ve zkratce šifrovaná komunikace, která je postavena na důvěře v certifikační autority. Nicméně pokud aplikace nevyužívá dalších technik zabezpečení, tak je poměrně jednoduché komunikaci rozšifrovat a odposlechnout. Lze využít nástroje jako Charles Proxy nebo Wireshark, díky kterým lze provést útok typu Man in the middle (MITM). Často je tento útok zmiňován v kontextu webových nebo desktopových aplikací, nicméně problém se stejně tak týká i aplikací pro systém Android.



Obrázek 13 Znázornění útoku typu Man in the middle [22]

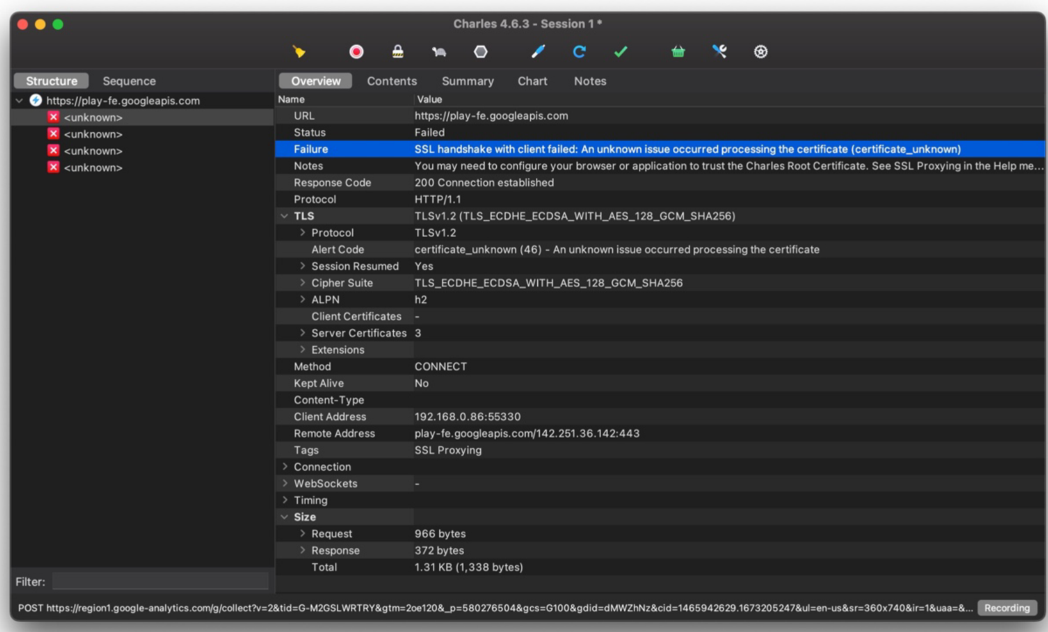
10.1.1 Android 6 a starší

Pro účely demonstrace odposlechu komunikace je využit nástroj Charles Proxy. Pro přesměrování toku dat stačí telefon připojit kabelem k počítači, kde běží program Charles Proxy. Poté je v telefonu nutné pro aktuální zdroj internetového připojení nastavit manuální proxy server. Jako IP adresa je použita IP adresa počítače, na kterém je Charles Proxy spuštěna. Číslo portu je pak standardně 8888, případně ho lze upravit v nastavení proxy v programu Charles Proxy.



Obrázek 14 Nastavení portu proxy serveru programu Charles Proxy [autor]

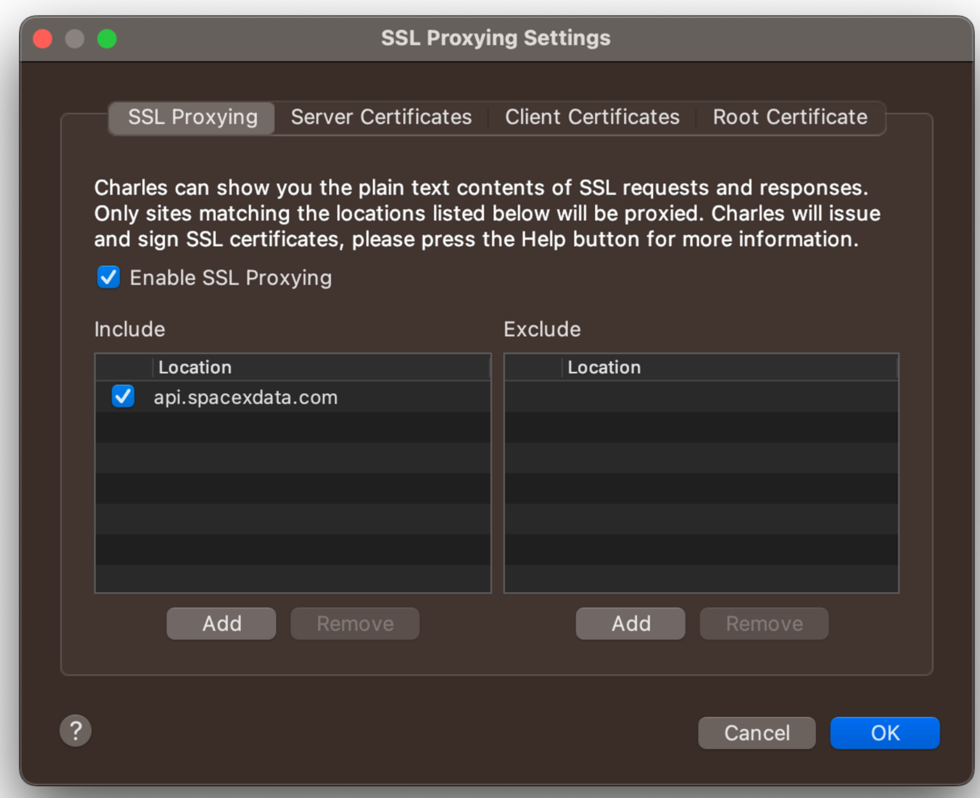
Po spuštění aplikace, webového prohlížeče nebo jakékoliv síťové komunikace telefonu, dojde k zobrazení záznamu tohoto požadavku ve výpisu programu Charles Proxy. Nicméně pro HTTPS požadavky dojde ihned k jejich zastavení s chybou, že se nepodařilo navázat tzv. SSL handshake.



Obrázek 15 Zobrazení chyby při pokusu čtení HTTPS komunikace bez validního certifikátu v programu Charles Proxy [autor]

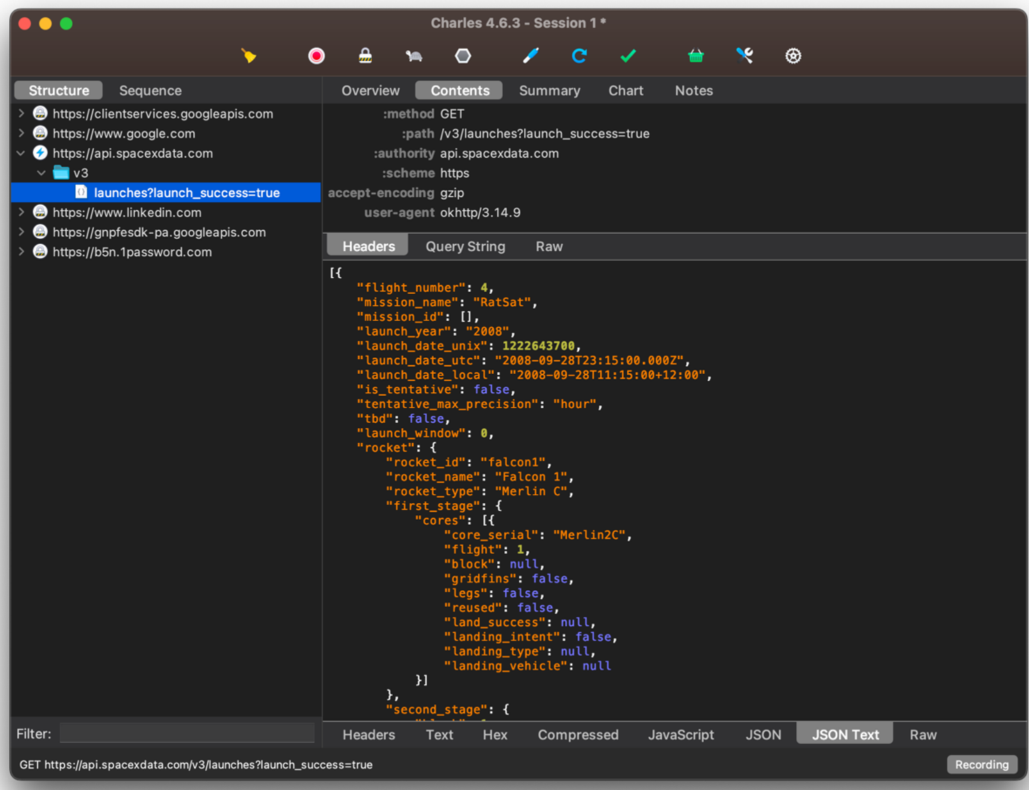
K překonání této překážky je do zařízení nutné nainstalovat HTTPS certifikát poskytnutý k programu Charles Proxy na adrese *http://chls.pro/ssl*. Pro funkčnost odkazu je nutné, aby telefon již komunikoval přes proxy server vytvořený programem Charles Proxy, protože certifikát není umístěn veřejně na internetu, ale na základě zaregistrování požadavku proxy serverem je navrácen z instalace programu.

Jako poslední krok je třeba povolit tzv. *SSL proxying* pro doménu, pro kterou chceme komunikaci zobrazit. Nastavení se provádí v menu *Proxy – SSL proxying settings*.



Obrázek 16 Nastavení *SSL proxying* pro požadovanou doménu [autor]

Jakmile je certifikát nainstalován, tak nejen že nové HTTPS požadavky jsou úspěšně zpracovány, ale je také možné v záložce „Contents" zobrazit obsah jednotlivých požadavků v čitelné podobě.



Obrázek 17 Zobrazení odpovědi z REST API z odposlechnuté HTTPS komunikace [autor]

Pokud předchozí chyba stále přetrvává pouze u některých požadavků, implikuje to, že aplikace využívá obranu, která je detailně popsána v kapitole 10.2 (Obrana proti útoku Man in the middle (MITM)).

10.1.2 Android 7 a novější

Od verze Android 7 (vydaného roku 2016) není možné použít nástroje jako je Charles Proxy na odposlechnutí komunikace instalací vlastního certifikátu [23]. Komunikaci tedy nelze odposlechnout u aplikace stažené přímo z Google Play, nicméně stále je možné aplikaci dekompilovat a upravit nastavení manifestu aplikace pro akceptaci uživatelsky instalovaných certifikátů.

Nejdříve je nutné v aplikaci vytvořit XML konfigurační soubor v adresáři *res/xml/network_security_config.xml* s následujícím obsahem. [9]

```xml
<network-security-config>
    <base-config>
        <trust-anchors>
            <certificates src="user" />
        </trust-anchors>
    </base-config>
</network-security-config>
```

Ukázka kódu 27 Konfigurační XML soubor pro nastavení důvěry v uživatelsky instalované certifikáty

Poté je nutné v manifestu aplikace tento soubor nastavit jako zdroj nastavení pro zabezpečení sítě. [24]

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:networkSecurityConfig="@xml/network_security_config" ... >
        ...
    </application>
</manifest>
```

Ukázka kódu 28 Nastavení odkazu na XML konfigurační soubor pro úpravu nastavení zabezpečení síťové komunikace

Aplikaci je po těchto úpravách nutné znovu zkompilovat a manuálně nainstalovat do zařízení (viz kapitola 8. Dekompilace aplikace). Poté je již možné postupovat stejně, jako pro telefony s Android 6 a starší.

10.1.3 Zařízení s provedeným root

V případě, že není možné umožnit vynucení validace uživatelsky instalovaných certifikátů, případně je nutné odposlechnout komunikaci z již nainstalované aplikace, tak poslední možností je provést root zařízení (viz kapitola 5 Root zařízení). Tento proces umožní root přístup k operačnímu systému a tím pádem i možnost upravovat systémové certifikáty. Postup pro získání root přístupu do systému se liší podle verzí systému Android i výrobce telefonu. Pro vložení vlastního certifikátu, aby mu systém důvěřoval, je nutné vložit nový certifikát do složky */system/etc/security/cacerts/* pomocí připojení dočasného úložiště. [25]

10.2 Obrana proti útoku Man in the middle (MITM)

Obranou proti odposlechu HTTPS komunikace je validace certifikátu, který je pro komunikaci používán. Ověření certifikátu je možné několika způsoby, každý je jinak náročný a přináší s sebou i určité nevýhody.

10.2.1 Certificate pinning

Jedná se o nejjednodušší variantu, kdy dochází k validaci celého certifikátu. Nevýhodou tohoto řešení je, že při každém obnovení certifikátu je nutné vydat novou verzi aplikace s připnutým novým certifikátem, jinak aplikace nebude schopna se serverem komunikovat. A když vezmeme v potaz, že ne všichni uživatelé pravidelně aplikace aktualizují, tak se ani zdaleka nejedná o vhodné řešení. [26]

Pro připnutí certifikátu lze použít systémovou komponentu *TrustManager*. Soubor s certifikátem je vložen do zdrojových souborů aplikace do cesty */res/raw*. V kódu aplikace je pak načten jako Input Stream. [24]

```
val certStream = resources.openRawResource(R.raw.demo_cert)
val keyStoreType = KeyStore.getDefaultType()
val keyStore = KeyStore.getInstance(keyStoreType)

keyStore.load(certStream, null)
```

Ukázka kódu 29 Načtení vlastního certifikátu ze zdrojových souborů aplikace typu *raw*

Poté je vytvořena instance trust manageru za použití klíče obsahující daný certifikát.

```
val tmAlgorithm = TrustManagerFactory.getDefaultAlgorithm()
val tmFactory = TrustManagerFactory.getInstance(tmAlgorithm)

trustManagerFactory.init(keyStore)
```

Ukázka kódu 30 Nastavení využití vlastního certifikátu pomocí systémové komponenty *TrustManager*

Dále stačí už jen získat SSL Context napojený na nově vytvořený *TrustManager* s certifikátem a realizovat pomocí něj spojení se serverem.

```
val sslContext = SSLContext.getInstance("TLS")
sslContext.init(null, tmFactory.trustManagers, null)
val url = URL("http://www.example.com/")
val urlConnection = url.openConnection() as HttpsURLConnection
urlConnection.sslSocketFactory = sslContext.socketFactory
```

Ukázka kódu 31 Způsob realizace HTTPS spojení se serverem, aby došlo k využití instalovaného vlastního certifikátu

10.2.2 Public key pinning

Nevýhodu připnutí celého certifikátu a nutnosti pravidelné aktualizace aplikace lze odstranit připnutím pouze veřejného klíče certifikátu za podmínky, že při rotaci certifikátu na serveru zůstane veřejný klíč certifikátu stejný. Nativní řešení připnutí pouze veřejného klíče nebo hashe nepodporuje, a proto je nutné použít knihovnu, například *OkHttp*, která představuje defacto standard pro zajištění HTTPS komunikace aplikace se serverem. [27]

Získání veřejného klíče z certifikátu je možné např. pomocí nástroje openssl.

```
openssl x509 -in /path/to/certificate.cer  -pubkey -noout
```

Ukázka kódu 32 Získání veřejného klíče z lokálně uloženého certifikátu pomocí nástroje *openssl*

Připnutí klíče se provede pomocí třídy *CertificatePinner*, jehož instance je pak využita při vytváření instance *OkHttp* klienta.

```
val certificatePinner = CertificatePinner.Builder()
    .add(
        "www.example.com",
        "sha256/ZC3lTYTDBJQVf1P2V7+fibTqbIsWNR/X7CWNVW+CEEA="
    ).build()

val okHttpClient = OkHttpClient.Builder()
    .certificatePinner(certificatePinner)
    .build()
```

Ukázka kódu 33 Připnutí veřejného klíče certifikátu pomocí třídy *CertificatePinner*

Tímto způsobem je možné připnout i více veřejných klíčů najednou. Pro navázání poté stačí, aby certifikát odpovídal pouze jednomu veřejnému klíči.

10.2.3 Network security configuration

Od verze Android 7 (vydaného roku 2016) je možné provést připnutí certifikátu pomocí veřejného klíče. Hash veřejného klíče certifikátu je nutné uvést v souboru *res/xml/network_security_config.xml* jako atribut *pin-set.*

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config cleartextTrafficPermitted="false">
        <domain includeSubdomains="true">yourdomain.com</domain>
        <pin-set>
            <pin digest="SHA-256">
9hdyeJFIEmx2Y01oXXXXXXXXXXmmSFZhBXXXXXXXXXX=
            </pin>
            <pin digest="SHA-256">
9Pacxtmctlq2Y73orFOOXXXXXXXXXXZhBXXXXXXXXXX=
            </pin>
        </pin-set>
    </domain-config>
</network-security-config>
```

Ukázka kódu 34 Konfigurační XML soubor pro přidání veřejného klíče certifikátu. Klíč je pro účely ukázky upraven na smyšlenou hodnotu.

Pokud již v aplikaci není soubor nastaven jako výchozí konfigurace zabezpečení sítě, je třeba přidat odkaz na něj do manifestu aplikace (soubor *AndroidManifest.xml*).

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
 xmlns:android="http://schemas.android.com/apk/res/android"
 package="com.packagename">

      <application
 android:networkSecurityConfig="@xml/network_security_config">
          ...
      </application>
</manifest>
```

Ukázka kódu 35 Nastavení odkazu na konfigurační XML soubor pro úpravu zabezpečení síťové komunikace v Manifestu aplikace

11 Interakce s ostatními aplikacemi

V systému Android existuje několik způsobů, jak mezi sebou mohou aplikace komunikovat a předávat si data. Vývojář musí být při implementaci jakéhokoliv z těchto způsobů obezřetný, protože chybnou definicí může jednoduše umožnit přístup ostatním aplikacím k soukromým datům uživatele anebo jim umožnit vyvolat v aplikaci akce, které by veřejně dostupné být neměly. [28]

11.1 Intents

Třída Intent (v překladu „účel“) se využívá pro spouštění aktivit (*Activity*), služeb (*Service*) anebo doručení a obdržení zprávy (*Broadcast*). A to jak pro vlastní aplikaci, tak i pro aplikace ostatní. [29]

Existují 2 typy Intentů:

- ***Explicit intents*** – specifikují akci, kterou má provést konkrétní aplikace. Tu specifikují pomocí názvu jejího balíčku (*package name*). Nejčastěji se využívají pro spuštění komponent vlastní aplikace, jako jsou aktivity a služby.
- ***Implicit intents*** – specifikují akci, která má být provedena, ale už nespecifikují aplikaci, která danou akci má provést. Výběr aplikace k provedení akce je realizován systémem tak, že uživateli nabídne k spuštění možné aplikace, které danou akci podporují.

Tento mechanismus umožňuje aplikacím přijímat vstupní data od ostatních aplikací, což je v řadě případů žádoucí. Explicitní intenty se mohou využívat například pro sdílení textu na specifickou sociální síť, například Twitter. Implicitní intenty zase mohou být použity pro spuštění navigace na určité místo v jakékoliv navigační aplikaci podle uživatelova výběru.

Ve většině případů aplikace, nebo alespoň většina aktivit aplikace, nemá důvod umožnit jejich spuštění za účelem zpracování akce ostatním aplikacím. Toto chování lze omezit pomocí nastavení atributu *exported* na hodnotu *false* pro každou Aktivitu nebo Službu definovanou v Manifestu aplikace, u které toto chování není žádoucí.

```xml
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.packagename">

      <application
        ... >
        <activity
          android:name=".ui.MainActivity "
          android:exported="false"
          ... />
         . . .

     </application>

  </manifest>
```

Ukázka kódu 36 Nastavení atributu aktivity *exported*, aby nebylo možné aktivitu spustit ostatními aplikacemi

11.2 Intent filtry

Aby mohla aplikace zpracovávat požadavky ostatních aplikací na provedení určité akce, musí pro své Aktivity specifikovat tzv. Intent filtry. Ty lze nastavit v Manifestu aplikace.

Intent filtr vyžaduje následující vstupní atributy:

- **Akce** – Specifikuje, kterou akci aplikace nabízí k vykonání. Může jít buď o předdefinované akce, jako například *android.intent.action.SEND*, *android.intent.action.VIEW*, nebo o vlastní definované akce.
- **Kategorie** – Určuje, do které kategorie aplikace spadá. Například *android.intent.category.APP_MAPS*, *android.intent.category.APP_MESSAGING* a další. Slouží k bližšímu specifikování výběru aplikací při požádání systému o zpracování akce.
- **Typ zasílaných dat** – Nepovinný atribut, který určuje typ zasílaných dat. Například text/plain, image/* a další.

```
<activity android:name="ShareActivity" android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
```

Ukázka kódu 37 Nastavení Aktivity aplikace na možnost zpracování odeslání textových dat pomocí Intent filtru

Intent filtry je žádoucí co nejpřísněji specifikovat, aby nedošlo k spuštění nežádoucí akce. Zároveň je třeba vstupní data z těchto akcí validovat, aby nedošlo k spuštění nežádoucí akce.

11.3 Content providers

V překladu poskytovatelé obsahu. Umožňují sdílení dat mezi aplikacemi. Opět stejně jako u Intent filtrů platí, že je žádoucí co nejpřísněji specifikovat jejich omezení, aby nedošlo k poskytnutí nežádoucích dat jiným aplikacím. Nelze jednoznačně říci, že některá kombinace nastavení *Content Provideru* je špatná a představuje bezpečnostní riziko, vždy záleží na konkrétním případu. Je tedy nutné důsledně kontrolovat rozsah poskytnutého přístupu pro daný *Content Provider*.

12 Závěry a doporučení

Výsledkem práce je poukázání na nejčastější případy, kdy vinou vývojáře aplikace může dojít k způsobení její zranitelnosti. Zkušenosti z praxe autora práce se shodují s obsahem práce a s poznatky ze zdrojů, které byly k tvorbě práce využity.

Je patrné, že v poslední době se bezpečnost operačního systému Android neustále zlepšuje. Stále ještě nelze říci, že je na úrovni např. konkurenčního operačního systému iOS od společnosti Apple, který běží na jejich telefonech iPhone.

Společnost Google, vyvíjející operační systém Android, každým rokem zvětšuje své snahy o zlepšení bezpečnosti systému. A to jak z uživatelského pohledu, tak i z toho vývojářského. Snaha eliminovat co největší množství potenciálních bezpečnostních chyb, kterých se mohou vývojáři Android aplikací dopustit, je patrná při každé nové verzi systému.

Jak ale popisuje tato práce, stále existuje určitá množina potenciálních chyb a hrozeb, kterých se může vývojář aplikace dopustit a umožnit tak zranitelnost aplikace. Nejlepší prevencí před těmito chybami je znalost těchto potenciálních chyb a s nimi spojených rizik ze strany vývojáře, která mu umožní se jich vyvarovat. V této práci jsou vyjmenovány ty nejčastější chyby, je vysvětleno jejich riziko a zároveň je navrhnuto jejich řešení.

Bezpečnost aplikace není ovšem pouze na jejím vývojáři. Ovlivňuje ji také sám uživatel svým chováním, kterým může riziko zvyšovat nebo snižovat. Například instalací aplikací pouze z oficiálního obchodu s aplikacemi Google Play riziko napadení systému výrazně snižuje.

Podle aktuálního trendu viditelného na posledních verzích operačního systému Android a s ním spojených SDK lze předpokládat, že bezpečnost platformy se bude i nadále zvyšovat. Stejně tak i prostor pro chybu ze strany vývojáře bude čím dál tím menší a bude převažovat snaha společnosti Google eliminovat riziko tím, že vývojářům neumožní, aby se chyby vůbec mohli dopustit.

13 Seznam použité literatury

[1] STATCOUNTER. Mobile Operating System Market Share Worldwide. *StatCounter Global Stats* [online]. 12 2022 [vid. 2022-12-03]. Dostupné z: https://gs.statcounter.com/os-market-share/mobile/worldwide

[2] CRĂCIUNESCU, Denis. The Layers of the Android Security Model. *Medium* [online]. 15. listopad 2020 [vid. 2022-12-03]. Dostupné z: https://proandroiddev.com/the-layers-of-the-android-security-model-90f471015ae6

[3] MUELLER, Bernhard, Sven SCHLEIER, Jeroen WILLEMSEN a Carlos HOLGUERA. *OWASP Mobile Application Security Testing Guide (MASTG)*. B.m.: The OWASP Foundation, 2022. v1.5.0. ISBN 978-1-257-96636-3.

[4] HAZARIKA, Skanda. How to root your Android smartphone: Google, OnePlus, Samsung, Xiaomi, and more. *XDA Developers* [online]. 15. listopad 2021 [vid. 2023-02-25]. Dostupné z: https://www.xda-developers.com/root/

[5] Save key-value data. *Android Developers* [online]. [vid. 2023-02-20]. Dostupné z: https://developer.android.com/training/data-storage/shared-preferences

[6] App Architecture: Data Layer - DataStore. *Android Developers* [online]. [vid. 2023-02-20]. Dostupné z: https://developer.android.com/topic/libraries/architecture/datastore

[7] Installing protoc. *proto-lens* [online]. [vid. 2023-02-25]. Dostupné z: http://google.github.io/proto-lens/installing-protoc.html

[8] Back up user data with Auto Backup. *Android Developers* [online]. [vid. 2023-03-11]. Dostupné z: https://developer.android.com/guide/topics/data/autobackup

[9] DROIDCON. Unpacking Android Security: Part 2 — Insecure Data Storage. *droidcon* [online]. 15. červen 2022 [vid. 2023-03-11]. Dostupné z: https://www.droidcon.com/2022/06/15/unpacking-android-security-part-2-insecure-data-storage/

[10] EncryptedSharedPreferences. *Android Developers* [online]. [vid. 2023-02-25]. Dostupné z: https://developer.android.com/reference/androidx/security/crypto/EncryptedSharedPreferences

[11] Save data using SQLite. *Android Developers* [online]. [vid. 2023-03-11]. Dostupné z: https://developer.android.com/training/data-storage/sqlite

[12] SQL injection. *Android Developers* [online]. [vid. 2023-03-11]. Dostupné z: https://developer.android.com/topic/security/risks/sql-injection

[13] DOLAN, Matthew. Android Security: SQL injection with the Room Persistence Library. *Medium* [online]. 29. říjen 2021 [vid. 2023-03-11]. Dostupné

z: https://appmattus.medium.com/android-security-sql-injection-with-the-room-persistence-library-69f4e286960f

[14] *About - DB Browser for SQLite* [online]. [vid. 2023-03-12]. Dostupné z: https://sqlitebrowser.org/about/

[15] CANDELLIER, Baptiste. Debugging and reviewing your Android dependencies with apktool. *Bedrock Tech Blog* [online]. 20. červen 2022 [vid. 2023-01-22]. Dostupné z: https://tech.bedrockstreaming.com/2022/06/20/android-apktool-decompiling.html

[16] *Android Platform Overview - OWASP Mobile Application Security* [online]. [vid. 2023-01-28]. Dostupné z: https://mas.owasp.org/MASTG/Android/0x05a-Platform-Overview/#android-architecture

[17] *Apktool - Documentation* [online]. [vid. 2023-01-22]. Dostupné z: https://ibotpeaches.github.io/Apktool/documentation/

[18] PAN, Bob. *dex2jar* [online]. Java. 28. leden 2023 [vid. 2023-01-28]. Dostupné z: https://github.com/pxb1988/dex2jar

[19] Getting Started in Kotlin. *Ted Hagos* [online]. 15. srpen 2018 [vid. 2023-01-28]. Dostupné z: https://tedhagos.com/posts/kotlin-getting-started/

[20] BURTON, Joseph. *Earthcomputer/kotlin-decompiler* [online]. Java. 10. leden 2023 [vid. 2023-01-29]. Dostupné z: https://github.com/Earthcomputer/kotlin-decompiler

[21] GOOGLE LLC. Shrink, obfuscate, and optimize your app. *Android Developers* [online]. [vid. 2022-12-20]. Dostupné z: https://developer.android.com/studio/build/shrink-code

[22] IMPERVA INC. What is MITM (Man in the Middle) Attack | Imperva. *Learning Center* [online]. [vid. 2022-12-05]. Dostupné z: https://www.imperva.com/learn/application-security/man-in-the-middle-attack-mitm/

[23] *SSL Certificates • Charles Web Debugging Proxy* [online]. [vid. 2022-12-19]. Dostupné z: https://www.charlesproxy.com/documentation/using-charles/ssl-certificates/

[24] Network security configuration. *Android Developers* [online]. [vid. 2023-01-10]. Dostupné z: https://developer.android.com/training/articles/security-config

[25] *Intercepting HTTPS on Android* [online]. [vid. 2023-01-08]. Dostupné z: https://httptoolkit.com/blog/intercepting-android-https/

[26] *Certificate and Public Key Pinning | OWASP Foundation* [online]. [vid. 2023-01-10]. Dostupné z: https://owasp.org/www-community/controls/Certificate_and_Public_Key_Pinning

[27] *Overview - OkHttp* [online]. [vid. 2023-02-20]. Dostupné z: https://square.github.io/okhttp/

[28] Interacting with Other Apps. *Android Developers* [online]. [vid. 2023-03-25]. Dostupné z: https://developer.android.com/training/basics/intents

[29] Intents and Intent Filters. *Android Developers* [online]. [vid. 2023-03-25]. Dostupné z: https://developer.android.com/guide/components/intents-filters

Zadání bakalářské práce

Autor: **Filip Oborník**

Studium: I1900235

Studijní program: B1802 Aplikovaná informatika

Studijní obor: Aplikovaná informatika

Název bakalářské práce: **Bezpečnost Android aplikací z pohledu vývojáře**
Název bakalářské práce AJ: Security of Android applications from the developer's perspective

Cíl, metody, literatura, předpoklady:

Cíl práce: Poukázat na možná bezpečnostní rizika a chyby při tvorbě mobilních aplikací pro platformu Android, kterých se vývojář může dopustit a navrhnout možná řešení k jejich minimalizaci nebo eliminaci.

1. Seznámit s problematikou bezpečnosti při tvorbě Android aplikací.
2. Poukázat na potenciální bezpečnostní chyby, kterých se vývojář může dopustit.
3. Popsat možné varianty zneužití bezpečnostních chyb.
4. Navrhnout řešení, která může vývojář aplikace využít, aby bylo riziko zneužití bezpečnostní chyby minimalizováno nebo eliminováno.
5. Závěr a zhodnocení vývoje pro platformu Android s ohledem na bezpečnost.

- MUELLER, Bernhard, Sven SCHLEIER, Jeroen WILLEMSEN a Carlos HOLGUERA. OWASP Mobile Application Security Testing Guide (MASTG). B.m.: The OWASP Foundation, 2022. v1.5.0. ISBN 978-1-257-96636-3.

- CRĂCIUNESCU, Denis. The Layers of the Android Security Model. Medium [online]. 15. listopad 2020 [vid. 2022-12-03]. Dostupné z: https://proandroiddev.com/the-layers-of-the-android-security-model-90f471015ae6

Zadávající pracoviště: Katedra informatiky a kvantitativních metod,
Fakulta informatiky a managementu

Vedoucí práce: Ing. Milan Košťák

Datum zadání závěrečné práce: 26.1.2021