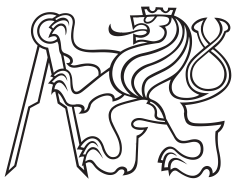
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra ekonomiky, manažerství a humanitních věd

Možnosti sledování událostí na systému Android a jejich využití pro odhalování podezřelého chování aplikací

Jan Piskáček
piskaja2@fel.cvut.cz

Únor 2015
Vedoucí práce: Ing. Ondřej David

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra ekonomiky, manažerství a humanitních věd

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Piskáček** Jan

Studijní program: Softwarové technologie a management
Obor: Manažerská informatika

Název tématu:

Možnosti sledování událostí na systému Android a jejich využití pro odhalování podezřelého chování aplikací

Pokyny pro vypracování:
1. Průzkum a analýza různých typů událostí sledovatelných na platformě Android.
2. Výběr vhodných událostí k popisu chování malware.
3. Vytvoření prototypu aplikace pro sběr vybraných událostí.
4. Ověření aplikace na testovacích datech a vyhodnocení sebraných dat.
5. Vyhodnocení přínosu a efektivity této metody.

Seznam odborné literatury:
1. Android Programming: The big Nerd Ranch Guide, Bill Philips 2. Android Malware and Analysis. Ken Dunham.
3. Dokumentace na: https://developer.android.com/

Vedoucí bakalářské práce: Ing. Ondřej David – AVAST Software. s.r.o.

Platnost zadání: do konce letního semestru 2015/2016

L.S.

Doc.Ing. Jaroslav Knápek, CSc. — vedoucí katedry

Prof.Ing. Pavel Ripka, CSc. — děkan

V Praze dne 10.2.2015

Poděkování / Prohlášení

Rád bych poděkoval vedoucímu této bakalářské práce Ing. Ondřeji Davidovi za mnoho cenných rad, názorů a doporučení, které mi předal při tvorbě mé bakalářské práce. Dále bych rád poděkoval Ing. Jiřímu Sejtkovi za návrh tématu této práce.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 22. 5. 2015

.....

Abstrakt / Abstract

Práce se zabývá možnostmi odchytávání událostí na operačním systému Android. Z událostí, které je možné zachytit, jsou poté vybrány ty, které mohou pomoci k odhalení potenciálně škodlivého chování aplikací. Na základě vybraných událostí je implementován prototyp aplikace pro Android pro jejich sběr. Nakonec je aplikace otestována na testovacích datech a vyhodnocen potenciální přínos tohoto přístupu při detekci malwaru.

Klíčová slova: malware, Android, aplikace, odchytávání událostí

This work deals with the possibility of monitoring events on the Android operating system. The events that can be helpful in detecting potentially malicious behaviour of applications are then selected. On the basis of selected events a prototype of Android application is implemented. Finally, the application is tested on test data and the potential benefits of this approach to malware detection are evaluated.

Keywords: malware, Android, application, monitoring of events

Title translation: The possibility of monitoring events on Android and its use for detecting suspicious behavior of applications

Obsah /

Tabulky / Obrázky

Kapitola 1
Úvod

Android je operační systém pro mobilní zařízení založený na jádru operačního systému Linux. V roce 2003 byla založena společnost Android, Inc., s cílem vytvořit konkurenci k operačnímu systému Symbian. Velkým zlomem byl rok 2005, kdy byla tato společnost koupena společností Google se záměrem vstoupit na trh s mobilními telefony.

V roce 2007 byla platforma Android představena skupinou technologických firem Open Handset Alliance v čele s firmou Google a dalšími významnými členy jako Samsung, HTC, Intel a Qualcomm, kteří se všichni podílejí na vývoji otevřeného standardu pro mobilní zařízení [1]. Velice důležitou charakteristikou Androidu je fakt, že je to software s otevřeným zdrojovým kódem (*open source software*).

Od roku 2010 byl zaznamenán obrovský nárůst počtu prodaných mobilních zařízení s Androidem, který tak v tomto ohledu suverénně předčil všechny ostatní mobilní operační systémy. S tímto velkým nárůstem počtu zařízení se zvýšil samozřejmě i počet aplikací, jejich uživatelů a vývojářů. Tento růst s sebou samozřejmě přináší i čím dál větší počet škodlivých aplikací [2] – malwaru (zkratka pro *malicious software*).

Příkladem škodlivého chování aplikací je automatické posílání SMS zpráv na prémiová čísla bez uživatelova vědomí, sbírání citlivých informací o uživateli (*spyware*) a dále forma vyděračství, které se docílí zašifrováním dat na telefonu a následným požadavkem na poplatek pro dešifrování dat (*ransomware*) [3]. Nejčastější motivací autorů malwaru pro Android je tedy vydělat peníze přímo nebo zjistit důležité informace o uživateli, a tyto informace posléze zpeněžit. Mobilní telefon je zařízení, které mají lidé často celý den u sebe, a tudíž je velice dobře využitelné k zjištění citlivých osobních informací o jeho uživateli [4].

Standartním zdrojem aplikací je služba Google Play. Všechny aplikace publikované prostřednictvím této služby by měly dodržovat určitá pravidla [5], jednak aby se udržela kvalita aplikací, a také aby bylo pevně definováno chování, které není povoleno. Navíc jsou všechny publikované aplikace podbrobeny malware skenu. Díky tomu je počet škodlivých aplikací na službě Google Play velice nízký. Existují však i neoficiální zdroje aplikací a na nich už není výskyt malwaru neobvyklý. Nejvíce je malware cílený na čínsky, rusky a anglicky mluvící uživatele, přičemž je to především Čína, kde se nachází mnoho alternativních internetových obchodů s aplikacemi.

V současné době patří mezi nejobvyklejší techniky detekce malwaru detekce založené na signatuře. Tato metoda funguje v principu tím způsobem, že pokud se v souboru najdou předem definované řetězce bajtů, je soubor označen jako malware. Signatury také mohou obsahovat zástupné znaky (*wildcards*), které zastupují jakýkoliv bajt. Lze tak vytvořit detekci, která je v principu obecnější než detekce bez zástupných znaků, což je žádoucí, protože cílem je pokrýt co největší počet malwaru co nejmenším počtem detekcí. Další technikou je detekce na základě vnější podobnosti. V tomto případě jsou definována určitá pravidla, která musí být splněna, aby byla aplikace označena jako malware. Mezi taková pravidla patří například charakteristiky jednotlivých sekcí spustitelného kódu aplikace, povolení, která si aplikace vyžádá, nebo charakteristiky řetězců, které se v aplikaci nacházejí.

Tvůrci malwaru často vytvářejí více verzí svých škodlivých aplikací a snaží se jejich detekci a odhalení ztížit pomocí šifrování a obfuskace zdrojového kódu [6]. Na druhé straně je obfuskace užitečná i pro vývojáře, kteří tím mohou do jisté míry ztížit reverzní inženýrství jejich softwarových produktů. Z těchto důvodů je v některých případech těžké vytvořit efektivní signatury [4].

Cílem této práce je zjistit možnosti sledování událostí na operačním systému Android za účelem odhalení potenciálně škodlivého chování aplikací. Motivací pro sledování událostí je možnost jeho využití jakožto alternativního zdroje informací o aplikacích, na jehož bázi by se daly aplikace kategorizovat jako potenciálně škodlivé nebo dokonce přímo jako malware. Dalším cílem je pak ověřit, jestli je tento přístup k odhalování malwaru efektivní či nikoliv. Předpokládaným přínosem této práce je zjištění, zda vůbec sledování událostí může přispět k odhalování potenciálně škodlivých aplikací či nikoliv.

Kapitola 2
Základy Androidu

Platformu Android lze rozdělit do čtyř vrstev, jak je patrné z obrázku 2.1. V nejnižší vrstvě je operační systém založený na jádře Linuxu, který poskytuje hardwarovou abstrakci pro vyšší vrstvy [7]. Zajišťuje management paměti, procesů, baterie, síťového připojení a další. Protože je Android určen především na mobilní zařízení, velký důraz je kladen na šetření baterie. V mobilních zařízeních se používá paměť flash, která má oproti pevnému disku menší počet zápisů. Důkazem důrazu na spotřebu baterie a flash paměť je například to, že Android nepodporuje stránkování na disk a řeší nedostatek volné operační paměti větším sdílením paměti a případně ukončením běžících procesů.

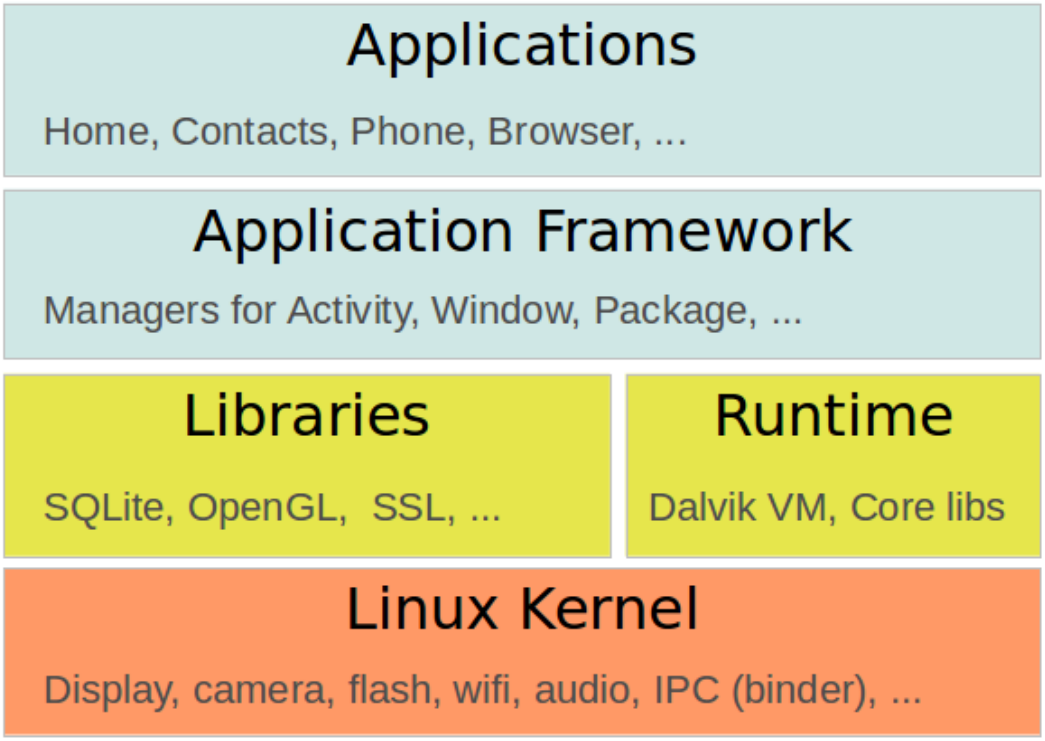
Druhou vrstvou je vrstva knihoven a běhového prostředí (*runtime*). Zde se nachází knihovny SGL pro 2D grafiku, OpenGL-ES pro 3D grafiku, Media framework pro zobrazení obrázků, videa a přehrávání zvuku. SQLite pro databáze, OpenSSL pro kryptografické funkce, dále WebKit – engine pro vykreslování webových stránek a Bionic, což je společností Google upravená verze standartní knihovny jazyka C pro Android.

Do této vrstvy se řadí i virtuální stroj (*virtual machine*), ve kterém běží všechny aplikace a řadí se tam i knihovny pro Android napsané v Javě. Aplikace mohou být napsány v C/C++ a zkompilovány do nativního kódu nebo v jazyce Java a pak být interpretovány. Interpretuje je virtuální stroj, kterým je Dalvik Virtual Machine (DVM) respektive Android Runtime (ART), který DVM od verze 5.0 (Lollipop) zcela nahradil. Důvodem použití jiného virtuálního stroje než Java Virtual Machine (JVM) je potřeba efektivnějšího využití paměti a procesoru na mobilních zařízeních a také fakt, že technologie a koncepty použité v JVM jsou patentovány společností Oracle. JVM má zásobníkovou architekturu zatímco DVM je založeno na registrech, to znamená, že používá méně instrukcí, které jsou složitější oproti instrukcím JVM.

Každá aplikace běží ve své vlastní instanci virutálního stroje a ve svém vlastním procesu. Každé aplikaci je také přiděleno user ID. Aplikace nemohou přistupovat k souborům a procesům jiných aplikací pokud jim to není explicitně povoleno. Tyto mechanismy přispívají k izolaci aplikace od zbytku systému a tím i k větší bezpečnosti.

Třetí a čtvrtou vrstvou je vrstvy aplikací a aplikačního frameworku. Aplikační framework poskytuje aplikacím API, díky kterému mohou jednoduše přistupovat k mnoha funkcím, kontrolovat životní cyklus aplikačních komponent, sdílet data ostatním aplikacím, přistupovat k souborům pro uživatelské rozhraní a zjišťovat informace o ostatních nainstalovaných aplikacích. Poskytuje také funkce pro zjištění polohy telefonu, identifikace zařízení (IMEI) a SIM karty (IMSI).

Nejvyšší vrstvou už jsou samotné aplikace instalované uživatelem nebo aplikace předinstalované, například webový prohlížeč, SMS klient, adresář kontaktů a prohlížeč obrázků.



Obrázek 2.1. Přehled vrstev Androidu, převzato z [8].

2.1 Komponenty aplikace

Jsou celkem 4 typy komponent, které jsou reprezentovány třídami *Activity* , *Service*, *BroadcastReceiver* a *ContentProvider*. Jedna aplikace se může skládat z více komponent. Každá z těchto komponent představuje samostatnou jednotku a často i vstupní bod aplikace. Komponenty mohou v principu fungovat i bez přítomnosti ostatních komponent stejné aplikace. Komponenty jsou aktivovány asynchronně pomocí zpráv, které se nazývají *Intent*. Intenty jsou explicitní nebo implicitní. Explicitní intenty v sobě nesou konkrétním název komponenty, která má být volána, zatímco implicitní intenty specifikují jen typ komponenty, o tom, která se zavolá může rozhodnout uživatel. Zachytávání Intentů je způsob kterým lze zjistit co se na systému děje, například instalace nové aplikace, přijmutí SMS a zapnutí zařízení. Právě odchytávání intentů je jedna z možností jak odhalit přítomnost potenciálně nežádoucí aplikace.

2.1.1 Activity

Aktivita představuje jednu „obrazkovku“ s uživatelským rozhraním. Aktivity mohou být nezávislé na ostatních komponentách aplikace, v praxi to znamená, že pokud máme aplikaci, která obsahuje aktivitu pro zobrazení obrázku a potřebujeme z jiné aplikace obrázek zobrazit, je možné tuto aktivitu použít i přes fakt, že k jiné aplikaci vůbec nepatří. Používání aktivit ostatních aplikací funguje tak, že aktivita definuje, jaký typ Intentů je schopna úspěšně zpracovat. Po vyslání Intentu je pak systémem nebo samotným uživatelem vybrána nejvhodnější aktivita pro jeho zpracování. Aktivita je vždy spuštěna v hlavním vlákně aplikace, které je také označováno jako *UI thread* [9].

2.1.2 Service

Service je komponenta pro běh na pozadí, to znamená, že narozdíl od Aktivity nemá uživatelské rozhraní. Je určena pro déle trvající operace, ale její nevýhodou je, že neběží ve svém vlastním vlákně, ale ve vlákně hlavním. Pokud chceme docílit, aby běžela ve svém vlastním vlákně, musí dědit od třídy IntentService, která sama dědí od třídy Service. Service se používá například pro stahování dat z Internetu, v době kdy není na popředí žádná Aktivita dané aplikace.

2.1.3 BroadcastReceiver

BroadcastReceiver je komponenta, která je schopna odchytávat události (konkrétně objekty typu *Intent*) vysílané systémem nebo ostatními aplikacemi. Jsou dva typy vysílaných broadcastů. Prvním typem je *normal broadcast*, který zachytí všichni posluchači s tím, že není definováno v jakém pořadí budou jejich metody *onReceive()* zavolány. Druhým typem je *ordered broadcast*, ve kterém jsou všichni registrovaní posluchači voláni postupně podle priority. Danou zprávu může pak posluchač poslat dál a nebo broadcast zrušit. Existuje i jednodušší verze BroadcastReceiveru, a to LocalBroadcastReceiver, kde zpráva neopouští aplikaci, ze které je odeslána [9].

BroadcastReceiver je důležitou komponentou, díky které můžeme sledovat události jako zapnutí mobilního telefonu, přijetí SMS zprávy, změnu připojení k síti, nainstalování nové aplikace a dokončení stahování. K odhalování podezřelého chování je dobrým kandidátem.

2.1.4 ContentProvider

Třída ContentProvider umožňuje aplikacím sdílet data. Ostatní aplikace je pak mohou data číst a upravovat, pokud je to povoleno. Data poskytovaná ContentProviderem jsou prezentována v tabulkách podobným tabulkám v relačních databázích.

2.2 Formát a uspořádání APK

Android application package (APK) je formát, ve kterém jsou distribuovány aplikace pro Android. Je založen na formátu Java Archive (JAR), který vychází z formátu zip, používanému ke kompresi. APK v sobě typicky obsahuje tyto soubory a složky:

```
* ---- assets/
   |-- lib/
         |-- armeabi/
         |-- x86/
   |-- META-INF/
               |-- MANIFEST.MF
               |-- CERT.RSA
               |-- CERT.SF
   |-- res/
         |-- drawable/
         |-- layout/
   |-- AndroidManifest.xml
   |-- classes.dex
   |-- resources.arsc
```

Výpis 2.1. Formát APK

Ve složce *assets* se mohou nacházet jakékoliv soubory, které aplikace potřebuje, obrázky, databáze, hudba a v případě malwaru například i zašifrované soubory, které při běhu aplikace dešifruje. Složka *lib* obsahuje zkompilovaný nativní kód, jednotlivé podsložky odpovídají různým architekturám procesorů. *META-INF* obsahuje digitálně podepsaný certifikát, který je použit pro podepsání aplikace, respektive jejích souborů, vyjma souborů nacházejících se ve složce *META-INF*. Certifikát nemusí být podepsaný certifikační autoritou, většina certifikátů je díky tomu *self-signed*. Podpis tedy neslouží k autentizaci APK, ale pro ověření toho, že je aplikace updatována stejným autorem (tzn. držitelem privátního klíče daného certifikátu), kterým byla vytvořena. Složka *META-INF* obsahuje také kontrolní součty ostatních souborů aplikace. Složka *res* (odvozeno od *resources*) obsahuje obrázky a soubory ve formátu xml udávající rozložení uživatelského rozhraní.

Soubor *classes.dex* obsahuje kód aplikace spustitelný v Dalvik Virtual Machine. Formát dex je použit místo standartního Java formátu class. Soubory dex je možné nahrát i za běhu aplikace pomocí třídy DexClassLoader a volat metody pomocí balíčku java.lang.reflect. Nahrání za běhu je jedna z technik, kterou malware aplikace používají. Například si zašifrovaný soubor dex se škodlivým kódem ukryjí ve složce assets a pak za běhu dešifrují a nahrají do virtuálního stroje k vykonání [10].

Soubor AndroidManifest.xml v sobě obsahuje důležité informace o aplikaci. Obsahuje jméno balíčku, které slouží jako unikátní identifikátor aplikace . Je v něm uvedena verze aplikace, nejnižší podporovaný level API a potřebné hardwarové vybavení pro fungování aplikace (např. Bluetooth nebo kamera). Manifest také obsahuje seznam komponent aplikace a jména jejich tříd v kódu. U každé komponenty je také možné uvést seznam Intentů, které je schopna zpracovat. Z bezpečnostního hlediska je v manifestu nejdůležitější seznam povolení (*permissions*), která aplikace požaduje. Při instalaci se informace o požadovaných povoleních zobrazí uživateli, přičemž má uživatel možnost přijmout instalaci nebo ji odmítnout. Některé malware aplikace spolehají na to, že uživatelé tyto informace nečtou a bezmyšlenkovitě instalaci povolí. Mezi potenciálně nebezpečná povolení patří zasílání SMS, monitorování odchozích hovorů, čtení systémových logů, upravení různých nastavení, zjištění polohy (pomocí GPS, Wi-Fi nebo triangulace z GSM sítě), čtení kontaktů a zapisování historie a záložek webového prohlížeče.

Kapitola 3
Průzkum událostí sledovatelných na platformě Android

Na operačním systému Android existují mechanismy, díky kterým lze sledovat události, které se na zařízení odehrávají. Pod pojmem událost se myslí akce nebo změna co v systému nastane. Může se například jednat o přijetí SMS zprávy, přístup na webovou stránku v prohlížeči, ale i věci v nižších vrstvách, jako zpřístupnění práv *root* a vysunutí SD karty. Události mohou mít také odlišný původ, některé jsou následkem akcí uživatele, jiné aktivitou nainstalovaných aplikací a další mohou mít původ v samotném operačním systému. Ne v mnoha případech lze přímo zjistit, kdo je původcem určité události. Na druhou stranu podrobnější pohled na časovou souslednost a kontext událostí má potenciál odhalit příčinu a původ událostí ostatních.

K lepší orientaci ve sledovatelných událostech se nabízí rozdělění událostí do kategorií podle několika různých kritérií. Jedním takovým kritériem je původ události, diskutovaný v minulém odstavci. Z výše uvedených důvodů je však toto kritérium velmi nepraktické. Dalším kritériem je periodicita sledování události. V některých případech, jako je například nainstalování nové aplikace není potřeba periodicky kontrolovat, jestli byla nainstalována, lze si zaregistrovat *receiver*, který bude zavolán vždy, kdy k nainstalování nové aplikace nastane. Opačný případ je zjištění, jestli nedošlo k zpřístupnění práv *root*, kde je potřeba aktivně v intervalech kontrolovat, jestli nedošlo ke změnám. Dalším kritériem, podle kterého lze události rozdělit je podle implementace, nejlépe podle tříd, které Android k tomuto účelu poskytuje, jako například `ContentObserver`, `BroadcastReceiver` a `FileObserver`. Dalším kritériem je také potřeba práv root k sledování události, některé události totiž bez těchto práv standartně sledovat nelze.

Každé z uvedených kritérií má své výhody a nevýhody, navíc se do jisté míry vzájemně překrývají. Z toho důvodu jsem události rozdělil do různých kategorií bez pevného kritéria, někde totiž dává smysl řídit se více podle způsobu implementace, jinde zase podle věcného významu události. Následuje hrubé rozdělení do kategorií:

1. Události zachycené pomocí komponenty *BroadcastReceiver* příjmající objekty typu *Intent*, které v sobě obsahují další data a informace, které záleží na typu vysílaného *Intentu*.
2. Změny dat poskytovaných ostatními aplikacemi, tyto změny lze sledovat pomocí třídy *ContentObserver*.
3. Změny souborů sledovatelných pomocí třídy *FileObserver*.
4. Informace o ostatních procesech a komponentách právě běžících na systému poskytované třídou *ActivityManager*.
5. Zjištění informací o nainstalovaných aplikacích poskytovaných třídou *PackageManager*.
6. Zpřístupnění práv *root*, které lze obecně zjistit více způsoby.
7. Čtení systémových logů pomocí nástroje *logcat*.
8. Události vytvářené přímo uživatelem, například dotyky obrazovky.

9. Sledování senzorů, které zařízení poskytuje, jako jsou senzory pohybu, polohy a teploty.
10. Sledování síťové komunikace, kterého lze docílit více způsoby s různě podrobnými výsledky.

Následující podkapitoly budou věnovány podrobnějšímu popisu sledování událostí rozdělených do uvedených kategorií.

3.1 BroadcastReceiver

BroadcastReceiver (dále jen receiver) je komponenta aplikace, která umí přijímat broadcasty. Broadcast je zpráva o události, prakticky jsou broadcasty využívány jako mechanismus pro posílání zpráv a informací mezi jednotlivými komponentami aplikací [9]. Broadcasty mohou být odeslány ostatními aplikacemi nebo systémem samotným. Aby mohl receiver příjmat broadcasty, musí být zaregistrován, a toho lze docílit dvěma způsoby: staticky a dynamicky.

Staticky zaregistrovaný receiver musí být zaznamenán v souboru AndroidManifest.xml pomocí tagu `<receiver>`, v něm lze dále nastavit, jaké broadcasty chce daný Receiver příjmat, a to pomocí tagu `<intent-filer>`, a také jestli muže přijímat broadcasty z jiných zdrojů než je aplikace, ke které patří pomocí atributu `android:exported`. Dále lze pomocí atributu `android:permission` nastavit, aby Receiver příjmal broadcasty jen od takových aplikací, které vlastní určitou *permission*. Za zmínku stojí i další atribut `android:process` vynucující běh receiveru ve svém vlastním procesu.

Dynamicky zaregistrovaný receiver se registruje v kódu aplikace pomocí volání metody `registerReceiver()`[1]) třídy `Context`. V případě, že daný receiver už nepotřebujeme, má tento způsob registrace výhodu v tom, že je jednodušší receiver zrušit, než v případě statické registrace.

Pro implementaci receiveru stačí zdědit od třídy `BroadcastReceiver` a překrýt metodu `onReceive()`. V této metodě se nedoporučuje vykonávat jakékoliv operace, které potřebují více času, systém totiž po 10 sekundách považuje receiver za kandidáta k zablokování nebo zrušení. Jakmile se vykoná všechen kód v `onReceive()`, objekt receiveru je považován za neaktivní a systém má povoleno ukončit proces, ve kterém běží. Z tohoto důvodu se nedoporučuje volat jakékoliv asynchronní operace, protože by se mohlo stát, že v době kdy, bude potřeba zpracovat takovou operaci, bude už receiver neaktivní.

Z bezpečnostních důvodů nemohou od verze Androidu 3.1 receivery příjmat jakékoliv Intenty, pokud uživatel ani jednou manuálně nespustil aplikaci, ke které daný receiver patří [9]. Pokud ji však uživatel zapne alespoň jednou, receivery této aplikace budou moci přijmat Intenty i přes následné vypnutí a zapnutí zařízení. Uživatel může také manuálně v nastavení receiver deaktivovat.

`BroadcastReceiver` ve své metodě `onReceive()` příjmá jako parametr objekt typu `Intent`. Intenty jsou obecně v Androidu využívány ke startování jednotlivých komponent, a to konkrétně všech kromě ContentProvideru. Je potřeba zdůraznit, že Intenty odeslané pomocí metody `startActivity()` jsou určeny pouze Aktivitám a proto nemohou být přijaty BroadcastReceivery. Totéž platí i naopak. V našem případě nás zajímají jen Intenty, se kterými pracuje receiver. Aplikace mohou vysílat broadcasty pomocí metody `sendBroadcast()` společně s Intentem jako jejím parametrem.

[1]) Musí se však dbát na to, aby byl Receiver i odregistrován pomocí metody `unregisterReceiver()`, jinak hrozí chyba *leaked broadcast receiver*.

Po přijetí Intentu v metodě `onReceive()` lze zjistit, o jaký typ Intentu se jedná pomocí metody `getAction()`, která vrátí řetezec definující danou akci, například `android.intent.action.BOOT_COMPLETED`. Pro přijmání určitých intentů je potřeba, aby měla aplikace patřičné *permissions* ve svém Manifestu. Třída Intent má mnoho dalších metod k uložení a získání dat a informací, které v něm jsou uloženy. Příkladem takové metody je `getExtras()`.

Zde je seznam význačných typů Intentů, většina z nich je odesílána systémem a z bezpečnostích důvodů nemohou být odeslány nikým jiným. K přijetí některých Intentů je také potřeba mít dostatečná povolení deklarovaná v AndroidManifest.xml.

- `android.intent.action.ACTION_SHUTDOWN`
- `android.intent.action.BOOT_COMPLETED`
- `android.intent.action.REBOOT`

Tato trojice Intentů je vyslána systémem při vypnutí, zapnutí nebo restartování zařízení. Největší využití ze všech má `BOOT_COMPLETED`, a to v případech, kdy potřebuje aplikace běžet na pozadí už od zapnutí zařízení. Konkrétně tedy zachytí tento Intent a v metodě `onReceive()` sama zapne Service pomocí dalšího Intentu.

- `android.intent.action.PACKAGE_ADDED`
- `android.intent.action.PACKAGE_CHANGED`
- `android.intent.action.PACKAGE_DATA_CLEARED`
- `android.intent.action.PACKAGE_FULLY_REMOVED`
- `android.intent.action.PACKAGE_REMOVED`
- `android.intent.action.PACKAGE_REPLACED`
- `android.intent.action.MY_PACKAGE_REPLACED`
- `android.intent.action.PACKAGE_RESTARTED`

Skupina Intentů, které se vysílají při akcích okolo instalování a odebírání aplikací (resp. balíčku - *packages*). Většina z nich v sobě kromě jména balíčku dané aplikace nese také informaci o User ID (UID) aplikace, což je číslo přidělené aplikaci při prvním nainstalování, toto číslo by mělo pro jednu aplikaci zůstat po celou dobu stejné, kromě případů, kdy je přeinstalována. Zajímavým Intentem z této skupiny je `PACKAGE_CHANGED`, který je vyslán v případě aktivování či zablokování komponent nainstalovaných aplikací, toho lze docílit např. voláním metody `setComponentEnabledSetting()` třídy `PackageManager`.

- `android.intent.action.UID_REMOVED`

  Vyslán při odebrání User ID ze systému.

- `android.provider.Telephony.SMS_RECEIVED`
- `android.intent.action.DATA_SMS_RECEIVED`
- `android.provider.Telephony.SMS_REJECTED`
- `android.provider.Telephony.SMS_SERVICE_CATEGORY_PROGRAM_DATA_RECEIVED`
- `android.provider.Telephony.SMS_DELIVER`

Intenty signalizující akce spojené s příjmáním SMS zpráv. Díky těmto Intentům lze zjistit obsah právě přijaté SMS a také číslo odesílatele. Receiver zaregistrovaný k těmto Intentům je velice často využívaný ve *spyware* aplikacích ke špehování příjmaných SMS zpráv nebo v *malware* aplikacích, které se snaží zaregistrovat uživatele k službám, které stojí peníze, bez jeho vědomí.

- `android.provider.Telephony.WAP_PUSH_DELIVER`

- `android.provider.Telephony.WAP_PUSH_RECEIVED`

Slouží k příjmání WAP Push zpráv. Ty jsou určeny k posílání odkazů ke stažení obsahu jako jsou vyzvánění, obrázky nebo tapety.

- `android.intent.action.NEW_OUTGOING_CALL`
- `android.intent.action.PHONE_STATE`

Používané k zachytávání příchozích a odchodzích hovorů. V případě odchozího hovoru je Intent odeslán před samotným vytáčením a obsahuje volané číslo, které lze i změnit. V případě příchozích hovorů jsou rozlišovány 3 různé stavy: `IDLE`, `OFFHOOK` a `RINGING`.

- `android.net.conn.CONNECTIVITY_CHANGE`
- `android.net.wifi.WIFI_STATE_CHANGED`

Slouží k sledování stavu připojení k síti. Intent v sobě drží referenci na objekt typu `NetworkInfo`, ve kterém jsou obsaženy podrobnější informace o právě vysílané změně.

- `android.net.wifi.p2p.CONNECTION_STATE_CHANGE`
- `android.net.wifi.p2p.THIS_DEVICE_CHANGED`

Intenty k signalizaci změny peer2peer Wi-Fi připojení.

- `android.intent.action.DOWNLOAD_COMPLETE`
- `android.intent.action.DOWNLOAD_NOTIFICATION_CLICKED`

Intenty vysílané třídou `DownloadManager` po úspěšném stažení souboru. `DownloadManager` je systémová služba určená k stahovaní přes HTTP protokol. Poskytuje API aplikacím k jednoduššímu stahování.

- `android.intent.action.PROXY_CHANGE`

Slouží k ohlášení změny používané proxy.

- `android.bluetooth.adapter.action.CONNECTION_STATE_CHANGED`
- `android.bluetooth.device.action.ACL_CONNECTED`
- `android.bluetooth.device.action.ACL_DISCONNECTED`

Intenty sloužící k vysílání změn v Bluetooth připojení.

- `android.intent.action.MEDIA_EJECT`
- `android.intent.action.MEDIA_MOUNTED`
- `android.intent.action.MEDIA_UNMOUNTED`

Používané k zjištění, jestli je připojena externí paměť (SD karta) a jestli je k dispozici.

- `android.intent.action.EXTERNAL_APPLICATIONS_AVAILABLE`
- `android.intent.action.EXTERNAL_APPLICATIONS_UNAVAILABLE`

Tyto Intenty posílají seznam aplikací, které byly nainstalovány na externí paměť a právě se staly přísupnými nebo naopak už přístupné nejsou. Kromě názvů balíčků aplikací v sobě obsahuje také seznam UID těchto aplikací.

- `android.app.action.DEVICE_ADMIN_ENABLED`

Aplikace, které chtějí získat oprávnění administrátora zařízení musí implementovat receiver, který čeká na tento Intent. Tento Intent bude poslán jen vybrané aplikaci, kterou uživatel zvolil jako administrátora zařízení.

- `android.intent.action.BATTERY_LOW`

Je odeslán v případě docházející baterie.

- `android.intent.action.SCREEN_OFF`
- `android.intent.action.SCREEN_ON`
- `android.intent.action.USER_PRESENT`
- `android.intent.action.WALLPAPER_CHANGED`
- `android.intent.action.CONFIGURATION_CHANGED`

`SCREEN_OFF` je odeslán v případě, kdy zařízení nereaguje (např. v režimu spánku) a nemusí při tom nutně být vypnutá obrazovka, jak by název mohl napovídat. `SCREEN_ON` je odeslán v přesně opačném případě než `SCREEN_OFF`. `USER_PRESENT` je vyslán poté, co uživatel zadá heslo nebo gesto pro odemčení zařízení (*keyguard*). `WALLPAPER_CHANGED` odeslán při změně tapety. `CONFIGURATION_CHANGED` hlásí změnu nastavení reprezentovaných třídou `Configuration`, mezi ně patří orientace obrazovky a změna klávesnice a jazyka.

- `android.intent.action.DOCK_EVENT`

Signalizuje připojení zařízení do dokovací stanice.

3.2 ContentObserver

Třída `ContentObserver` slouží ke sledování změn obsahu poskytovaného ostatními aplikacemi, a to konkrétně obsahu poskytovaného komponentou *ContentProvider*. Pro funkci ContentObserveru je ho potřeba zaregistrovat pomocí metody `registerContentObserver()` třídy `ContentResolver`. Důležitý parametr této metody je *Uniform Resource Identifier* (URI), který určuje jaký obsah bude sledován. Po úspěšném zaregistrování bude v ContentObserveru zavolána metoda `onChange()` vždy, když se obsah změní. Existuje také možnost vytvořit `ContentObserver` s objektem typu `Handler` v konstruktoru, metoda `onChange` bude pak zavolána ve vlákně, ve kterém byl příslušný `Handler` vytvořen. [9]

Důležitým faktem je, že při volání `onChange()` sice víme, že se obsah změnil, ale informaci o tom, co se konkrétně změnilo už nemáme, z toho důvodu je potřeba si nějakým způsobem ukládat jednotlivé stavy a pak porovnávat rozdíly mezi nimi. Nabízející se řešení je uložit si seřazený seznam položek a při každém novém stavu vyhledávat položky v novém seznamu pomocí binárního půlení ve starém seznamu. Tímto způsobem lze najít položky, které přibyly.

Všechny URI, které jsou standartně poskytovány systémem Android jsou uloženy jako konstanty v třídách v balíčku `android.provider`. Obecně je bezpečnější používat konstanty uložené v těchto třídách oproti napevno daným konstantám v zdrojovém kódu, URI by se mohla teoreticky v budoucnu změnit a pak by druhý z uvedených způsobů přestal fungovat.

Zde je výčet zajímavých URI:

- `content://com.android.launcher.settings/favorites?Notify=true`
- `content://com.android.launcher2.settings/favorites?Notify=true`

Tyto URI slouží k přístupu k ikonám na domovské ploše.

- `content://sms/inbox`
- `content://sms/sent`
- `content://sms/outbox`

URI ke čtení odeslaných a přijatých zpráv základní aplikace pro práci s SMS zprávami. K přístupu je třeba mít povolení `READ_SMS`.

- `content://com.android.contacts`

  Používáno ke čtení kontaktů, nutné mít povolení `READ_CONTACTS`.

- `content://browser/bookmarks`
- `content://com.android.chrome.browser/bookmarks`
- `content://com.sec.android.app.sbrowser.browser/bookmarks`
- `content://org.mozilla.firefox.db.browser/bookmarks`

URI historie a záložek obvyklých prohlížečů, názvy mohou být matoucí, vyskytuje se tam sice jen slovo *bookmarks*, ale ve skutečnosti obsahují i historii prohlížení. Ke čtení je potřeba povolení `READ_HISTORY_BOOKMARKS`.

- `content://call_log/calls`

  Slouží ke čtení záznamů hovorů, je potřeba mít povolení `READ_CALL_LOG`.

3.3 FileObserver

Dalším způsobem jak sledovat změny a události na zařízení s Androidem je pomocí třídy `FileObserver`. Třída `FileObserver` umožňuje sledovat události týkající se souborů, konkrétně jejich vytvoření, smazání, dále změny v souboru, přesunutí a také otevření a zavření souboru pro čtení či zápis. Všechny vyjmenované akce budou zaznamenány bez ohledu na to, který proces danou akci vykonal [9].

Sledování adrešářů pomocí FileObserveru nefunguje rekurzivně, to znamená, že změny budou hlášeny jen pro soubory a adresáře o úroveň pod sledovaný adresář a ne hlouběji. Pro sledování všech souborů a podadresářů se tedy musí FileObserver doimplementovat tak, aby fungoval rekurzivně [11]. Další důležitou věcí, na kterou je třeba si dávat pozor, je aby nebyl FileObserver uvolněn *garbage collectorem*, z toho důvodu je nutné držet si referenci na vytvořený FileObserver po celou dobu, co bude jeho funkce potřeba.

K implementaci stačí vytvořit vlastní třídu dědící od třídy `FileObserver` a překrýt metodu `onEvent()`, ta bude zavolána vždy, když dojde ke změně. Jejím parametrem je cesta k souboru, u kterého došlo ke změně a informace o zachycené akci. Cestu k souborům, které chceme sledovat musíme zadat hned v konstruktoru. Samotné sledování začíná a končí voláním metod `startWatching()` a `stopWatching()`.

Sledování souborů tímto způsobem je limitováno tak, že aplikace, ve které byl FileObserver vytvořen musí mít práva čtení sledovaného souboru. Na Androidu z bezpečnostních důvodů nemají standartně aplikace právo čtení souborů ostatních aplikací. Vyjímkou je případ, kdy aplikace ukládají svá data na SD kartu, protože všechny soubory z SD karty mohou číst i zapisovat všechny ostatní aplikace. Čtení je povoleno i v adresáři `/system`, ve kterém dělají změny zejména aplikace určené k získání root privilegií, pomocí FileObserveru tedy lze jejich aktivitu zaznamenat. Aby mohl `FileObserver` číst soubory, které nejsou na SD kartě a patří ostatním aplikacím, je potřeba získat root práva a na požadovaných souborech povolit práva pro čtení.

3.4 ActivityManager

`ActivityManager` je třída pomocí které lze zjistit informace o právě běžících aplikacích. Umožňuje výpis stavu vybrané aplikace a výpis využití operační paměti na základě

process id (pid). Dále poskytuje informace o běžících procesech, lze zjistit, jaké aplikace v těchto procesech běží a k tomu UID těchto procesů. Kromě informací o procesech lze také zjistit informace o běžících *Service* komponentách, a to např.: jméno Service, dobu od zapnutí Service, čas poslední aktivity a pid procesu, ve kterém běží.

ActivityManager má také možnost ukončit běh procesů běžících na pozadí (tzn. těch, které nejsou přímo viditelné uživatelem). Příklad výpisu pid procesů a všech aplikací, které v nich běží:

```
ActivityManager activityManager = (ActivityManager)
getSystemService(ACTIVITY_SERVICE);

List<ActivityManager.RunningAppProcessInfo> runningAppProcessInfoList =
activityManager.getRunningAppProcesses();

String[] pkgList;
ActivityManager.RunningAppProcessInfo runningAppProcInfo;

for (int i = 0; i < runningAppProcessInfoList.size(); i++) {
    runningAppProcInfo = runningAppProcessInfoList.get(i);
    Log.i(TAG, "pid: " + runningAppProcInfo.pid);
    pkgList = runningAppProcInfo.pkgList;
for(int j = 0; j < pkgList.length; j++) {
Log.i(TAG, "Package name: " + pkgList[j]);
    }
}
```

Výpis 3.2. Kód pro získání pid právě běžících procesů.

3.5 PackageManager

Pomocí třídy `PackageManager` lze zjistit informace o ostatních nainstalovaných aplikačních balíčcích, převážně se jedná o informace uložené v souboru *AndroidManifest.xml* daného balíčku. Seznam nainstalovaných aplikací se získá pomocí metody `getInstalledPackages()` a informace o jedné konrétní aplikaci pomocí `getPackageInfo()`. Informace o jednotlivých komponentách aplikačních balíčku lze získat pomocí metod `getActivityInfo()`, `getReceiverInfo()` a `getServiceInfo()`. [9]

Dále lze pomocí metod `getPackagesHoldingPermissions()` a `checkPermission()` zjistit, jaká povolení byla aplikaci přiděleňa.

Další zajímavé metody jsou `queryIntentActivities()`, `queryIntentServices()` a `queryBroadcastReceivers()`. Pomocí těchto metod lze zjistit seznam všech komponent, které jsou schopny zpracovat požadovaný *Intent* (tak jak je uvedeno v Manifestu). Tímto způsobem lze například získat seznam receiverů, které čekají na přijetí SMS, což je obvyklá praktika *spywaru*. Je možnost také zjistit, jestli má aplikace receiver registrovaný na Intent `android.app.action.DEVICE_ADMIN_ENABLED`, který je potřeba implementovat, pokud se aplikace chce stát Device administrátorem[1]). V poslední řadě lze také zjistit jméno aplikačního balíčku na základě *uid*.

Z informací z Manifestu si lze relativně snadno vytvořit obrázek, co daná aplikace vyžaduje a čeho je schopna, proto je `PackageManager` k odhalení potenciálně nežádoucích aplikací velice užitečný. Implementace se nejlépe nabízí v kombinaci s BroadcastReceiverem čekajícím na nainstalování nového aplikačního balíčku.

[1]) Detailnější informace na `http://developer.android.com/guide/topics/admin/device-admin.html`

3.6 Detekce práv root

Detekci root práv lze provést mnoha způsoby. Skutečnost, že zařízení skutečně běží s právy root, bohužel nelze ověřit s jistotou, nicméně existuje množství metod, kterými je možné tuto skutečnost ověřit s vysokou pravděpodobností. Ke zjištění, zda má zařízení nastavena práva root je možné použít některé z následujících metod [12].

1. Podle digitálního podpisu jádra systému.
2. Podle přítomnosti speciálních aplikací.
3. Podle přítomnosti spustitelných souborů *su*.
4. Podle práv zápisu do konkrétních adresářů.
5. Podle reakce na pokus o vykonání příkazu *su*.

Prvním způsobem je zkontrolování konfigurace a souborů, které se často liší právě na základě toho, jestli má zařízení práva root. Sem patří zjištění, jestli je jádro operačního systému podepsáno oficiálním klíčem od vývojáře či nikoliv. Pokud není, znamená to, že je na zařízení pravděpodobně nainstalována alternativní ROM, a ty mají ve většině případů zpřístupněná root práva. Zde je příklad zdrojového kódu provádějící tento test [13]:

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

Výpis 3.3. Metoda `firstRootCheck()`.

Pokud by proměnná `buildTags` obsahovala řetězec `release-keys`, znamenalo by to, že je na zařízení nainstalovaná oficiální ROM, což samo o sobě o rootnutí zařízení nic nevypovídá. Pokud ale obsahuje řetězec `test-keys`, je rootnutí zařízení pravděpodobné, ale nedá se to tvrdit s naprostou jistotou.

Další způsob je hledání nainstalovaných aplikací, které jsou k získání root práv určeny. Tato metoda stejně jako předchozí není stoprocentně spolehlivá. Zde jsou příklady jmen aplikačních balíčků, které jsou určené k získání root práv:

- `eu.chainfire.supersu`
- `com.noshufou.android.su`
- `com.ramdroid.appquarantine`
- `com.thirdparty.superuser`

Kromě kontroly nainstalovaných aplikací lze i zkontrolovat přítomnost binárních spustitelných souborů s názvem *su* (*super user*), díky kterým lze spustil terminál s právy root. Nejčastější cesty, kde se *su* nachází jsou:

- `/system/bin/su`
- `/system/xbin/su`
- `/sbin/su`
- `/system/su`

Dalším znakem indikujícím práva root jsou povolená práva pro zápis adresářů, které standartně zapisovatelné nejsou, mezi tyto adresáře patří:

- `/data`
- `/system`

- /system/bin
- /system/sbin
- /system/xbin
- /vendor/bin
- /sys
- /sbin
- /etc
- /proc

V neposlední řadě lze příkaz *su* vykonat a ověřit, jestli příkaz funguje či nikoliv, poté případně zavolat příkaz *id*, díky kterému lze ověřit, že daný proces běží s *uid* rovnému nule, protože v Linuxových systémech má účet s root právy vždy *uid* rovno nule.

Důležitým faktem zůstává, že ani jedna z uvedených metod není bezpodmínečně spolehlivá. Také existují aplikace, které jsou přímo určené k maskování root práv v zařízení. Na druhou stranu, i když nejsou uvedené metody stoprocentní, pořád poskytují cenné informace.

3.7 Čtení logů

Operační systém Android v sobě obsahuje systém pro logování událostí a čtení logů vytvořených aplikacemi a systémovými komponentami. Systém pro logování se skládá ze čtyř bufferů poskytovaných jádrem operačního systému, první z nich *main* je určen pro logování aplikací, druhý *events* je pro informace o událostech na systému, dále *radio* pro zaznemanávání akcí souvisejících s připojením do mobilní sítě, v poslední řadě buffer *system* určený pro debugging. [14]

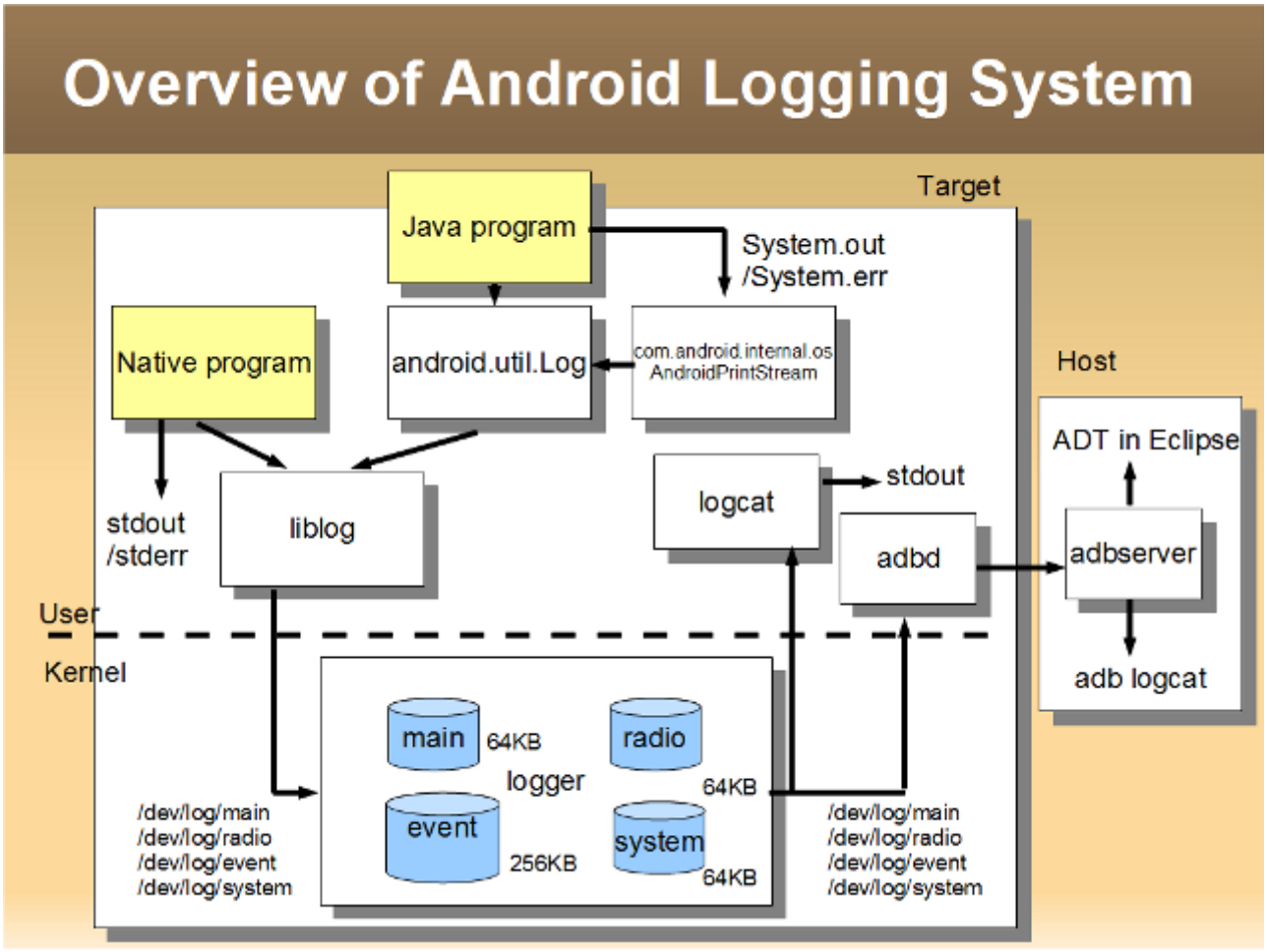
Dále existuje třída pomocí které lze do logů zapisovat, tou je `android.util.Log`. Další částí je program pro čtení logů jménem *logcat*. Program *logcat* lze spustit přes *adb* (*Android Debug Bridge*) pro čtení logů v reálném čase na počítači, ke kterému je zařízení připojeno. Standartně čte *logcat* jen buffer *main*, ale lze nastavit i čtení ostatních bufferů.

Logcat je možné číst i pomocí programového kódu přímo v aplikaci, je k tomu ale potřeba povolení `android.permission.READ_LOGS`. Od verze Androidu 4.1 Jelly Bean není pro běžné aplikace možné toto povolení dostat, jediným způsobem jak toto omezení obejít je získat práva root. S právy root pak lze aplikacím přidat povolení, které by standartně nemohly dostat, v tomto případě by vykonání příkazu přímo v aplikaci vypadalo takto:

```
Runtime.getRuntime()
.exec("su -c pm grant <jmeno_package> android.permission.READ_LOGS");
```

Výpis 3.4. Vykonání příkazu *su* přímo ve zdrojovém kódu.

Výstup programu *logcat* je možné filtrovat podle tagu a priority. Tag představuje komponentu ze které zpráva pochází, to znamená, že tag může být například `ActivityManager`. Priorita indikuje důležitost dané zprávy, celkem má 7 úrovní: *verbose*, *debug*, *info*, *warning*, *error*, *fatal* a *silent*. Události, které se dají pomocí *logcat* zachytit, jsou zapntí jakékoliv komponenty Activity, čtení přijaté SMS a přístup na URL pomocí webového prohlížeče.



Obrázek 3.1. Přehled logovacího systému na Androidu, převzato z [15].

3.8 Monitorování uživatelského rozhraní

Monitorování dotyků obrazovky i mimo vlastní aplikaci lze docílit vytvořením *Service*, ve kterém je poté vytvořen velice malý fragment uživatelského rozhraní. V tomto fragmentu pak zaregistrovat `OnTouchListener`, který bude sledovat všechny dotyky mimo tento fragment a jelikož je tento fragment velmi malý a okem prakticky neviditelný, všechny dotyky budou spadat mimo fragment a zároveň si uživatel fragmentu nevšimne [16]. Tímto lze monitorovat všechny dotyky, kromě dotyků v notifikační a navigační části uživatelského rozhraní a také dotyků vykonaných před odemčením obrazovky.

Samo o sobě sledování dotyků obrazkovky asi nemá schopnost zjistit, co se na zařízení děje, ale v kombinaci s ostatními sledovanými událostmi může pomoct v detekci potenciálně škodlivého chování ostatních aplikací. Příkladem může být událost odeslání SMS a zároveň absence jakéhokoliv uživatelského dotyku ve vymezeném čase před odesláním dané SMS. V tomto příkladě sice s naprostou jistotou nemůžeme tvrdit, že chování je škodlivé, ale je minimálně podezřelé.

3.9 Senzory

Zařízení s Androidem mohou mít k dispozici senzory. Senzory jsou schopné měřít zrychlení, teplotu, rotaci zařízení, magnetické pole a tlak. Toto měření se také dá svým

způsobem považovat za události, co se na systému dějí, ale pro odhalení potenciálně škodlivého chování aplikací pravděpodobně není užitečné. Jako zajímavost lze uvést *proof of concept*[1]) malware jménem Soundcomber [17], který snímá řeč. Analýzou řeči je pak schopný identifikovat například číslo kreditní karty.

3.10 Monitorování síťové komunikace

Monitorovat síťovou komunikaci ostatních aplikací na Androidu lze více způsoby.

Nejjednodušší, ale zároveň nejvíce limitovaný způsob je pomocí třídy `TrafficStats`, která poskytuje metody pro zjištěšní počtu odchozích a příchozích packetů či bajtů. Metody jsou schopné vrátit počet packetů či bajtů pro dané *uid*, konkrétně se jedná o metody `getUidRxBytes()` a `getUidRxPackets()` pro přijaté, a `getUidTxBytes()` a `getUidTxPackets()` pro odeslané. Tímto způsobem však nelze zjistit IP adresy, se kterými aplikace komunikují, zjitit lze jen velikost odeslaných a přijatých dat.

Dalším způsobem je použití packetového analyzátoru *tcpdump*. *Tcpdump* pracuje v příkazové řádce a je schopen zachytit veškerou síťovou komunikaci rodiny protokolů TCP/IP. Výstup programu může být textový nebo ve formátu *pcap*, který lze dále analyzovat např. pomocí programu *Wireshark*. Tento způsob zachytávání packetů poskytuje mnoho informací a podrobností, ale jeho velkou nevýhodou je fakt, že *tcpdump* pro svůj běh vyžaduje práva root.

Dále je možné sledovat otevřené *tcp* a *udp* sockety pomocí čtení souborů ve složce `/proc/net/`. Informace je možné číst ze souborů *tcp*, *tcp6*, *udp* a *udp6*. Tyto soubory mohou být čteny všemi aplikacemi, i bez speciálních práv nebo povolení a bez práv root. Na tyto soubory ale nelze použít třídu `FileObserver`, protože data získaná z těchto souborů nejsou uložena na disku, ale jsou generována automaticky pokaždé, kdy je ze souborů čteno. Je tedy potřeba v pravidelných intervalech dané soubory číst.

Další možností je použití třídy `VpnService`, pomocí které lze zachytávat odchozí a příchozí pakety. Tato třída ale není k tomuto účelu určena. Způsob, který zachytávání paketů umožňuje spočívá ve vytvoření VpnService, přes kterou je přesměrována veškerá komunikace přes *tcp* a *udp*. Zachycené pakety musí být po zpracování dále odeslány původnímu adresátovi, aby byla síťová komunikace funkční.

[1]) tzn. funkční model doposud jen teoreticky předpokládaného návrhu

Kapitola 4
Výběr vhodných událostí k popisu chování malware

K výběru vhodných událostí k popisu potenciálně škodlivého chování aplikací je potřeba pochopit, o co se tyto aplikace snaží a jak svého cíle dosahují. V následujících kapitolách budou uvedeny vybrané události a konkrétní příklady malware aplikací, které by mohly být potenciálně sledováním těmchto událostí odhaleny. Je zcela možné, že na sledovaném systému vyvstane podezřelý sled událostí, i když se na něm ve skutečnosti žádný malware nenachází. Je proto nutné brát události opatřené tímto sledováním nikoli jako detekci samotnou, jako spíše ukazatel pravděpodobnosti, že se na daném zařízení malware nachází a takto označená zařízení či aplikace podrobit hlubšímu zkoumání.

4.1 BroadcastReceiver

Jedna z potenciálně nebezpečných věcí je instalování nebo odinstalování aplikací bez uživatelova vědomí, proto je vhodné tyto události sledovat, a to pomocí receiveru přijmajícím tyto Intenty:

- `android.intent.action.PACKAGE_ADDED`
- `android.intent.action.PACKAGE_CHANGED`
- `android.intent.action.PACKAGE_REMOVED`
- `android.intent.action.PACKAGE_REPLACED`
- `android.intent.action.UID_REMOVED`

Pokud se zachytí událost `PACKAGE_ADDED` můžeme zjistit informace o právě nainstalované aplikaci díky *PackageManageru*. Intent `PACKAGE_CHANGED` je odeslán, pokud je některá z komponent ostatních aplikací zapnuta nebo vypnuta. Malware často vypíná svou hlavní *Activity* komponentu, protože tím dosáhne vymazání své ikonky z menu. Uživatel si pak hůře všimne toho, že je škodlivá aplikace na jeho zařízení nainstalována. Následující úryvek kódu pochází z malware pojmenovaného *HideIcon* [18], který se dostal i na Google Play. Zdrojový kód byl získán dekompilací pomocí programu *jadx*. Na úrykvu je vidět volání metody `setComponentEnabledSetting()` s takovými parametry, které způsobí zmizení ikony aplikace z menu. Zdroj malwaru: [19]

```
File file = new File(new File(b.a()), "Pconfig.txt");
if (file.exists()) {
   CompassActivity compassActivity2 = this.a;
   String a = k.a(file);
   if (a.f(a) && Integer.valueOf(a).intValue() == 1) {
      this.a.getPackageManager()
      .setComponentEnabledSetting(this.a.getComponentName(), 2, 1);
   }
}
```

Výpis 4.5. Úryvek kódu získaného z malwaru pojmenovaného *HideIcon*.

Jiné rodiny malware jsou zaměřeny na zprávy SMS, a to na odesílání SMS na premiová čísla nebo registraci služeb přes SMS bez uživatelova vědomí. Zachytávání Intentů pro přijetí SMS zprávy je tak vhodnou událostí k potenciálnímu odhalení malwaru tohoto typu. Dále je malware schopen i vytáčet telefonní čísla, z toho důvodu je dobré poslouchat i Intenty, které jsou vyslány při zahájení hovoru nebo vytočení čísla.

- `android.provider.Telephony.SMS_RECEIVED`
- `android.intent.action.DATA_SMS_RECEIVED`
- `android.intent.action.NEW_OUTGOING_CALL`
- `android.intent.action.PHONE_STATE`

Malware často stahuje nové aplikace a soubory do zařízení a v případě, že k tomu použije třídu `DownloadManager`, je možné příjmat Intenty po zdárném stažení souboru.

- `android.intent.action.DOWNLOAD_COMPLETE`

4.2 ContentObserver

Další motivací malwaru je nějakým způsobem přimět uživatele k tomu, aby si prohlédl obsah nějaké webové stránky, může to být například reklama a nebo také podvodná stránka, která se pokouší uživatele oklamat. Vhodným nástrojem, jak uživatele přimět k zobrazení daného obsahu je nainstalování ikonky do aplikačního launcheru, která po kliknutí otevře webový prohlížeč. Dále také umělé přidání záložek nebo historie do webového prohlížeče. Z těchto důvodů je vhodné využít *ContentObserver*. Prohlížečů je však mnoho a neexistuje jedno sjednocené URI, ze kterého by se dala historie získat, musí se tedy používat URI pro každý konkrétní prohlížeč. Příklady URI populárních webových prohlížečů:

- `content://com.android.launcher.settings/favorites?Notify=true`
- `content://browser/bookmarks`
- `content://com.android.chrome.browser/bookmarks`
- `content://com.sec.android.app.sbrowser.browser/bookmarks`
- `content://org.mozilla.firefox.db.browser/bookmarks`

Sledování odchozích SMS zpráv je velice užitečné k odhalení aplikací, které odesílají SMS zprávy na prémiová čísla. URI pro odchozí zprávy je:

- `content://sms/outbox`

Příkladem malwaru posílajícím SMS zprávy je trojan *Scavir*. *Scavir* je zaměřen na rusky mluvící uživatele a do zařízení se snaží dostat pod záminkou falešného antivirového skenu. Úryvek kódu posílajícího SMS:

```
if (cc > 0) {
   smsText = new StringBuilder(String.valueOf(smsText))
   .append("+2").toString();
}
if (!this.currentMNC.equals(Activator.MTS_MNC)) {
   sms.sendTextMessage((String)pair.first, null, smsText, sentPI, null);
} else if (cc < 2) {
   sms.sendTextMessage((String)pair.first, null, smsText, sentPI, null);
}
cc++;
```

Výpis 4.6. Úryvek kódu získaného z malwaru jménem *Scavir*.

4.3 FileObserver

Pomocí *FileObserveru* lze sledovat změny souborů na SD kartě a ve složce `/system`. Některý malware se snaží získat práva root. Metody, které k tomu používá často zahrnují i modifikaci souborů v adresáři `/system`, a proto má sledování tohoto adresáře potenciál tuto činnost odhalit. Malware může také být charakterizován konkrétními soubory, které ukládá na SD kartu, takže i sledování souborů na SD kartě může k odhalení pomoci.

4.4 ActivityManager

Seznam spuštěných aplikací získaný pomocí třídy `ActivityManager` může pomoci odhalit zdroj ostatních zachytávaných událostí nebo naopak dokázat, že určitá aplikace nemůže být původcem událostí, protože v čase kdy k nim došlo nebyla aplikace zapnuta.

4.5 PackageManager

Pomocí třídy `PackageManager`, lze zjistit povolení, která ostatní aplikace požadují. Díky tomu lze identifikovat potenciálně nebezpečné aplikace, které mají mnoho povolení. Tabulka potenciálně nebezpečných povolení:

ACCESS_COARSE_LOCATION	ACCESS_FINE_LOCATION
BIND_DEVICE_ADMIN	BIND_VPN_SERVICE
BLUETOOTH_ADMIN	CALL_PHONE
GET_ACCOUNTS	GET_TASKS
INSTALL_PACKAGES	INSTALL_SHORTCUT
INTERNET	KILL_BACKGROUND_PROCESSES
MOUNT_UNMOUNT_FILESYSTEMS	PROCESS_OUTGOING_CALLS
READ_CALL_LOG	READ_CONTACTS
READ_HISTORY_BOOKMARKS	READ_LOGS
READ_PHONE_STATE	READ_SMS
RECEIVE_BOOT_COMPLETED	RECEIVE_SMS
RECORD_AUDIO	SEND_SMS
UNINSTALL_SHORTCUT	WRITE_HISTORY_BOOKMARKS
WRITE_SECURE_SETTINGS	WRITE_SMS

Tabulka 4.1. Tabulka potenciálně nebezpečných povolení.

Dále lze také zjistit, na jaké Intenty má aplikace registrované receivery. Potenciálně nebezpečné receivery poslouchají na tyto Intenty:

- `android.provider.Telephony.SMS_RECEIVED`
- `android.intent.action.NEW_OUTGOING_CALL`
- `android.app.action.DEVICE_ADMIN_ENABLED`

4.6 Detekce práv root

Veškeré aplikace určené k získání root práv na zařízení jsou potenciálně nebezepečná, protože s právy root má aplikace nad zařízením plnou kontrolu, může například číst všechny soubory a tím potanciálně získat citlivá data o uživateli. Zjištění, jestli jsou k dispozici práva root je tak k odhalení potenciálně nežádoucích aplikací vhodné.

4.7 Čtení logů

Čtení logů dokáže nahradit následující povolení:

- READ_CONTACTS
- GET_TASKS
- READ_HISTORY_BOOKMARKS
- READ_SMS

Z bezpečnostích důvodů není možné od verze Androidu 4.1 Jelly Bean číst systémové logy bez práv root. Čtení logů je tedy z velké části nahraditelné ostatními událostmi, to však neznamená, že nepřináši žádnou přidanou hodnotu, lze sledovat spouštění systémových komponent, interakci uživatele s obrazovkou. Nevýhodou je obrovské množství informací, které k odhalení potenciálně škodlivého chování nepomáhají a tudíž vytvářejí informační šum. Sledování logů z uvedených důvodů není do budoucna perspektivní metodou pro odhalování potenciálně škodlivých aplikací.

4.8 Monitorování uživatelského rozhraní

Monitorování interakce uživatele s obrazovkou nemá samotné možnost odhalit jakékoliv potenciálně škodlivé chování aplikací. V kombinaci s ostatními sledovatelnými událostmi však může mít určitou hodnotu. Může totiž pomoci odhalit s jakou pravděpodobností se určité události odehrály jako následek přímé interakce s uživatelem, a tato informace už přínosná je.

4.9 Monitorování síťové komunikace

Monitorování síťové komunikace je pro odhalení malwaru na zařízení velice užitečné. Malware často dostává příkazy ze vzdáleného serveru k vykonání, tímto způsobem může autor malwaru prakticky ovládat zařízení na dálku a spouštět funkce, které chce. Další využití je v případě, kdy malware aplikace stahuje nový binární soubor ke spuštění, což je poměrně efektivní technika, jak škodlivý software skrýt. Pokud máme k dispozici databázi serverů, ze kterých pochází škodlivé aplikace nebo s nimi škodlivé aplikace komunikují, má potom monitorování síťové komunikace potenciál k jejich odhalení.

Kapitola 5
Implementace

Na základě výběru událostí vhodných k odhalení potenciálně škodlivých aplikací na systému byla implementována aplikace ve vývojovém prostředí přímo určeném k vývoji pro Android s názvem *Android Studio*. *Android Studio* je založeno na vývojovém prostředí *IntelliJ IDEA*. *Android Studio* výrazně ulehčuje konfiguraci, vývoj a následné nasazení aplikací pro Android a jeho jediným konkurentem je prostředí *Eclipse* s pluginem ADT (Android Development Tools). Informace k implementaci čerpány z [20].

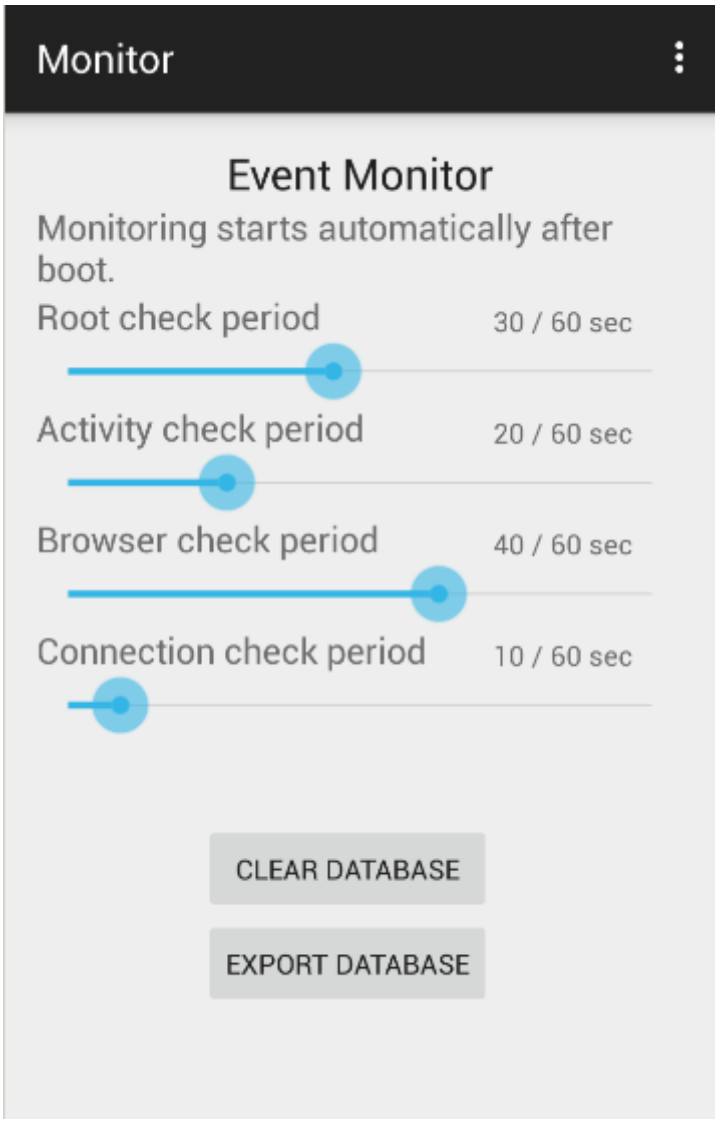
Struktura zdrojového kódu vytvořeného projektu je následující (všechny soubory mají příponu `.java`):

```
Package name: cz.cvut.pisky.monitor
* ---- BootReceiver
   |-- MainActivity
   |-- constants/
   |           |-- EventConst
   |           |-- PeriodConst
   |-- database/
   |           |-- DbHelper
   |-- event/
   |           |-- Event
   |           |-- EventDao
   |-- observers/
   |           |-- LauncherIconObserver
   |           |-- RecursiveFileObserver
   |           |-- SmsSentObserver
   |-- receivers/
   |           |-- CallReceiver
   |           |-- DownloadReceiver
   |           |-- PackageReceiver
   |           |-- SmsReceiver
   |-- runnable/
   |           |-- InitRunnable
   |           |-- ActivityCheck
   |           |-- BrowserCheck
   |           |-- NetCheck
   |           |-- RootCheck
   |-- service/
   |           |-- InitService
   |           |-- MyIntentService
   |           |-- ScreenTouchService
   |-- util/
               |-- Util
```

Výpis 5.7. Struktura vytvořeného projektu.

5.1 MainActivity

Představuje jedinou Activity komponentu celé aplikace, a tím pádem i jediné uživatelské rozhraní aplikace. Umožňuje uživateli nastavit periodu, se kterou se budou provádět jednotlivé úkony. Dále obsahuje dvě tlačítka, první pro vymazání všech záznamů událostí z databáze a druhé ke zkopírování databáze na interní paměť, kterou lze číst.



Obrázek 5.1. Uživatelské rozhraní MainActivity.

5.2 BootReceiver

Třída `BootReceiver` je komponentou *BroadcastReceiver*, která je spuštěna po zapnutí zařízení. Slouží k zapnutí komponent běžících na pozadí, a to konkrétně `InitService` a `ScreenTouchService`.

5.3 Adresář constants

Obsahuje třídy `EventConst` a `PeriodConst`. Třída `EventConst` v sobě drží konstanty pro ukládání událostí do databáze, typicky to jsou názvy událostí, například řetězec `Package added`.

Třída `PeriodConst` obsahuje periody, ve kterých se zjišťují informace o událostech, o které se konkrétně jedná bude uvedeno dále.

5.4 Adresář database

Obsahuje třídu `DbHelper` dědící od třídy `SQLiteOpenHelper`. Obsahuje metody pro přístup a správu databáze, konkrétně o vytvoření tabulky s událostmi, dále pro vymazání veškerých záznamů z této tabulky a také metodu pro zkopírování celé databáze na interní čitelnou paměť.

Třída `DbHelper` je implementována jako návrhový vzor *Singleton* (*thread safe*[1]) verze), pokud bychom totiž do databáze přistupovali z více vláken skrz více instancí `DbHelper`, mohlo by dojít k jejich vzájemnému zablokování.

5.5 Adresář event

Instance třídy `Event` představují zaznamenanou událost, má atributy `id`, `dateTime`, `milisFromBoot`, `type`, `info`, `extras`.

Atribut `type` představuje typ dané událost, `info` představuje pokud možno stručnou informaci o dané událost a atribut `extras` je použit k uložení podbrobnějších záznamů o události.

Atributy `dateTime` a `milisFromBoot` představují čas zaznamenání události. Důvodem k zaznamenání času ve dvou atributech je ten, že `dateTime` je získán pomocí třídy `Date`, a je zaznamenána v člověkem jednoduše čitelném formátu, její nevýhodou je fakt, že odpovídá uživatelem nastavitelnému času, a tak je potenciálně nepřesným ukazatelem. Z tohoto důvodu existuje atribut `milisFromBoot`, který vždy obsahuje počet uplynulých milisekund od zapnutí zařízení až po zaznamenání události. Třída `EventDao` slouží k samotnému ukládání záznamů do vytvořené databáze

5.6 Adresář observers

V tomto adresáři jsou třídy implementující *ContentObservery* čekající na změny v uložených SMS, při změně `SmsSentObserver` zajistí získání všech odeslaných SMS a uložení odchozích čísel do databáze. Dále `RecursiveFileObserver`, určený ke sledování změn v souborech ve složkách `/system` a `/sdcard` a k uložení těchto změn do databáze. Všechny observery v této složce jsou registrovány v metodě `run()` třídy `InitRunnable`.

5.7 Adresář receivers

Adresář se třídami dědící od třídy `BroadcastReceiver`.

`CallReceiver` čeká na odchozí hovory a ukládá vytáčená čísla do databáze. `DownloadReceiver` sleduje událost dokončení stahování pomocí třídy `DownloadManager`, ale

[1]) Pojem *thread safe* v případě návrhového vzoru Singleton znamená, že i v případě práce s více vlákny bude s jistotou existovat jen jedna instance dané třídy.

žádné další informace neposkytuje. `PackageReceiver` čeká na události nainstalování a odinstalování aplikací, dále změna komponent aplikací a jejich aktualizace. Navíc při události nainstalování nové aplikace prověří pomocí odeslání intentu třídě `MyIntentService` potenciálně nebezpečná povolení, která byla nové aplikaci přidělena a uloží jejich záznam do databáze. Po nainstalování nové aplikace podobným způsobem také ověří, jestli právě nainstalované aplikace neobsahují komponenty schopné číst příchozí SMS zprávy.

5.8 Adresář runnable

Obsahuje třídy `InitRunnable`, `ActivityCheck`, `BrowserCheck`, `NetCheck` a `RootCheck`.

Třída `InitRunnable` je spuštěna ve svém vlastním vlákně a je zodpovědná za první spuštění všech metod, které jsou určené k periodickému zaznamenávání událostí. V této třídě je vytvořen objekt třídy `Handler`, který se používá k dodávání `Runnable` objektů k vykonání ve vlákně, ve kterém je *handler* vytvořen. Všechny metody `run()` *runnable* objektů dodané skrz tento *handler* jsou tímto způsobem spuštěny popořadě v daném vlákně. Zbytek tříd v adresáři `runnable/` je určen k periodickému sledování událostí a instance těchto tříd jsou dodávány skrz handler vytvořený ve třídě `InitRunnable`.

Třída `ActivityCheck` zjišťuje informace o tom, jaké aplikace právě na systému běží. Třída `BrowserCheck` sleduje historii a záložky základního předinstalovaného prohlížeče. Třída `NetCheck` čte soubory `/proc/net/tcp` a `/proc/net/tcp6` a dále je převádí do stručnějšího formátu k uložení do databáze. Třída `RootCheck` spouští testy popsané v kapitole 3.6. Všechny uvedené třídy ukládají zjištěné informace do databáze.

Mechanismus, kterým se docílí periodického spuštění požadovaných metod spočívá ve volání metody třídy `Handler` s názvem `postDelayed()`. Kostra třídy, která periodicky vykonává požadovaný kód je vidět na úryvku kódu. Pro vysvětlení jsou vloženy komentáře.

```
public class PeriodicCheck implements Runnable {
    private static final String TAG = "PeriodicCheck";
    private Context context = null;
    private Handler handler = null;
    private EventDao eventDao = null;

/* handler, díky kterému je možné periodicky spouštět
        metodu run(), je předán v konstruktoru */
    public PeriodicCheck(Context context, Handler handler) {
        this.context = context;
        this.handler = handler;
        this.eventDao = new EventDao(this.context);
    }

/* metoda run() zavolá všechny požadované metody pomocí
  this.callAllChecks() a dále předá instanci sebe sama
  handleru ke spuštění za dalších (PeriodConst.checkPeriod * 1000)
  milisekund */
    @Override
    public void run() {
        Log.i(TAG, "run() called from thread: "
                + Thread.currentThread().getId());
        this.callAllChecks();
```

```
        this.handler.postDelayed(this, PeriodConst.checkPeriod * 1000);
    }

/* sjednocující metoda pro všechny kontroly,
   aby byla metoda run() přehlednější  */
    public void callAllChecks() {
        this.check();
    }

// metoda k získání dané události a její uložení do databáze
    private void check() {
        Log.i(TAG, "check() called ");

        // zde je standartně kód získávající danou událost

        // následující řádek ukládá informace do databáze
        this.eventDao.createEvent(EventConst.CHECK, "info", "extras");
    }
}
```

Výpis 5.8. Kostra třídy pro periodické vykonávání metod.

5.9 Adresář service

Obsahuje třídy `InitService`, `MyIntentService` a `ScreenTouchService`.

`InitService` je spuštěna po zapnutí zařízení a slouží k vytvoření nového vlákna, do kterého jako `Runnable` objekt vkládá instanci třídy `InitRunnable`. Důvodem k vytvoření nového vlákna je snaha o plynulé uživatelské rozhraní. Kdyby nové vlákno vytvořeno nebylo, v důsledku by to znamenalo, že většina periodických akcí k získávání událostí by byla spuštěna v hlavním vlákně, ve kterém běží i uživatelské rozhraní, a to může vést k jeho zaseknutí a dále to zhoršuje jeho odezvu.

`MyIntentService` je určena k vykonávání jednorázových akcí, a to ve svém vlastním vlákně. Mezi tyto akce patří zkopírování databáze na interní paměť, vymazání databázových záznamů a zjištění nebezpečných povolení přidělených dané aplikaci.

`ScreenTouchService` je komponenta service spuštěna při zapnutí zařízení. Je spuštěna v hlavním vlákně, a to z toho důvodu, že je určena k zachytávání dotyků obrazovky a obsahuje prvek uživatelského rozhraní.

5.10 Adresář util

Obsahuje pomocné metody využívané napříč projektem.

Kapitola 6
Testování

Pro účely testování byl použit emulátor Genymotion, zařízení HTC Vision s nainstalovanou neoficiální ROM CyanogenMod s verzí Androidu 4.2.2 Jelly Bean (API level 17) a dále tablet LG tablet V500 s oficiální verzí Androidu 4.4.2 KitKat (API level 19).

K pohodlnějšímu a rychlejšímu prohlížení ukládaných záznamů o událostech jsem použil standartní třídu pro logování `android.util.Log`. Po spuštění vytvořené aplikace lze pak snadno ukládané logy číst pomocí příkazové řádky a programu *adb* příkazem `adb logcat`. Zobrazované logy lze pak jednoduše filtrovat, např. vývojového prostředí *Android Studio* umožňuje nastavení podrobného filtrování logů.

6.1 PackageReceiver a ScreenTouchService

Odchytávání událostí jsem nejprve testoval na jednodušších aplikacích. Jako demonstraci jsem vybral nainstalování aplikace z Google Play jménem Electronic Function Generator [21]. Ukázka demonstuje funkčnost tříd `PackageReceiver`, `ScreenTouchService` a `MyIntentService`. Log zachycených událostí při nainstalování aplikace byl upraven pro lepší čitelnost a vypadá následovně:

```
===== NEW ENTRY =====
Time: 611323
Type: Screen touch
Info: MotionEvent
Extras: none
===== NEW ENTRY =====
Time: 622821
Type: Screen touch
Info: MotionEvent
Extras: none
===== NEW ENTRY =====
Time: 634107
Type: Package added
Info: Package name: com.erdemaslan.functiongenerator
Extras: UID: 10146, Replacing: false
===== NEW ENTRY =====
Time: 634148
Type: Permission check
Info: Package name: com.erdemaslan.functiongenerator
Extras: Permissions granted to com.erdemaslan.functiongenerator:
        android.permission.INTERNET
===== NEW ENTRY =====
Time: 634250
Type: Query intent check
Info: none
```

```
Extras:
ACTIVITY INTENTS
Intent: android.app.action.ADD_DEVICE_ADMIN
Package: com.android.settings/.DeviceAdminAdd

BROADCAST INTENTS
Intent: android.intent.action.NEW_OUTGOING_CALL
Packages:
cz.cvut.pisky.monitor/.receivers.CallReceiver
com.lge.qremote/.settings.PhoneStateBroadcastReceiver
com.avast.android.mobilesecurity/.app.filter.core.OutgoingCallReceiver

Intent: android.app.action.DEVICE_ADMIN_ENABLED
Packages:
com.lge.email/.service.EmailDevicePolicyReceiver
com.lge.email/.service.EmailDevicePolicyReceiverNoSdCard
com.lge.email/.service.EmailDevicePolicyReceiverDPM
com.lge.email/.service.EmailDevicePolicyReceiverUserAccount
com.lge.email/.service.EmailDevicePolicyReceiverNoSdCardWifiOnly
com.lge.email/.service.EmailDevicePolicyReceiverWifiOnly
com.google.android.gms/.mdm.receivers.MdmDeviceAdminReceiver

Intent: android.provider.Telephony.SMS_RECEIVED
Packages:
com.avast.android.mobilesecurity/.app.filter.core.MessageReceiver
com.lge.email/.receiver.SmsReceiver
com.lge.p2p/.msg.transaction.MsgReceiver_phone
com.google.android.gms/.icing.proxy.SmsMonitor
cz.cvut.pisky.monitor/.receivers.SmsReceiver
```

Výpis 6.9. Výpis událostí zachycených pomocí tříd *PackageReceiver* a *ScreenTouchService*

První dvě události jsou typu *Screen touch* a odpovídají kliknutí v aplikaci Google Play na tlačítko *install* a dále na tlačítko *accept*, pro akceptování požadovaných povolení. Následuje typ události *Package added*, která dále obsahuje UID nainstalované aplikace a jméno jejího aplikačního balíčku. Po nainstalování aplikace automaticky následuje kontrola požadovaných povolení, to odpovídá událost typu *Permission check*. V tomto případě je vidět pouze jen jedno potenciálně nebezpečné povoleni `INTERNET`. Poslední akcí automaticky spuštěnou po nainstalování nové aplikace, je *Query intent check*, která má za úkol zjistit veškeré aplikace, které mají zaregistrované receivery na požadované *intenty*. Při důkladném zkontrolování je vidět, že nově nainstalovaná aplikace tyto receivery zaregistrovány nemá.

6.2 RootCheck

Dalším testem je ověření zda funce pro zjištění root opravdu fungují, za tímto účelem bylo použito již zmíněné zařízení HTC Vision s ROM CyanogenMod, které má root práva odemčená. Periodický výpis na tomto zařízení vypadá takto:

Kontrola root práv ukazuje hodnotu `true`, pokud výsledek jejího testu naznačuje, že práva root jsou k dispozici. Na výpisu je vidět, že dva z šesti vykonaných testů ukazují, že root je k dispozici.

```
===== NEW ENTRY =====
Time: 132677
Type: Root check
Info: Build.TAGS true
Extras: none
===== NEW ENTRY =====
Time: 132971
Type: Root check
Info: otacerts.zip false
Extras: none
===== NEW ENTRY =====
Time: 133009
Type: Root check
Info: Superuser.apk false
Extras: none
===== NEW ENTRY =====
Time: 133040
Type: Root check
Info: At least one su or busybox binary present: true
Extras: /system/bin/su present: true,
/system/xbin/su present: true,
/system/xbin/busybox present: true
===== NEW ENTRY =====
Time: 133101
Type: Root check
Info: At least one rooting package present: false
Extras: none
===== NEW ENTRY =====
Time: 133138
Type: Root check
Info: At least one directory writable: false
Extras: none
```

Výpis 6.10. Ukázka periodického výpisu získaného třídou *RootCheck*.

6.3 ActivityCheck

Dalším testem je ověření událostí zachycených třídou `ActivityCheck` pomocí třídy `ActivityManager`. Na výpisu je vidět seznam procesů a komponent service spuštěných na zařízení, tyto informace jsou užitečné k odhalení potenciálního zdroje ostatních zachytávaných událostí. Informace jsou pomocí třídy `ActivityCheck` zaznamenávány periodicky. Výpisy byly záměrně zkráceny, protože obsahují mnoho záznamů a výčet všech není účelem této ukázky.

```
===== NEW ENTRY =====
Time: 1521981
Type: Process check
Info: Number of running processes: 29
Extras:
PID: 2952 UID: 10004 Package names: com.android.browser
PID: 2187 UID: 10035 Package names: com.google.android.location
```

```
PID: 1953 UID: 10039 Package names: com.android.systemui
PID: 2013 UID: 10024 Package names: com.android.inputmethod.latin
PID: 2092 UID: 10035 Package names: com.google.android.gms
PID: 2212 UID: 10019 Package names: com.android.location.fused
PID: 1803 UID: 1000 Package names: com.google.android.backup,
com.android.providers.settings
PID: 2160 UID: 10035 Package names: com.google.android.gsf,
 com.google.android.syncadapters.contacts, com.google.android.gms
PID: 2850 UID: 10066 Package names: cz.cvut.pisky.monitor
PID: 2147 UID: 10047 Package names: com.android.smspush
// zde zkráceno

===== NEW ENTRY =====
Time: 1522037
Type: Service check
Info: Number of running services: 30
Extras:
Name: com.android.phone.TelephonyDebugService
PID: 2025 UID: 1001 Active since: 25959
Name: com.google.android.gms.gcm.GcmService
PID: 2160 UID: 10035 Active since: 37079
Name: com.google.android.location.internal.GoogleLocationManagerService
PID: 2092 UID: 10035 Active since: 32773
Name: com.android.bluetooth.pbap.BluetoothPbapService
PID: 0 UID: 1002 Active since: 48500
Name: com.android.location.fused.FusedLocationService
PID: 2212 UID: 10019 Active since: 27278
Name: com.google.android.gms.common.stats.GmsCoreStatsService
PID: 2288 UID: 10035 Active since: 38296
Name: com.android.smspush.WapPushManager
PID: 2147 UID: 10047 Active since: 25580
Name: com.android.bluetooth.pan.PanService
PID: 0 UID: 1002 Active since: 19935
Name: com.google.android.location.NetworkLocationService
PID: 2187 UID: 10035 Active since: 26612
// zde zkráceno
```

Výpis 6.11. Výpis informací o běžících procesech získaných pomocí třídy *ActivityCheck*.

6.4 RecursiveFileObserver

Funkce třídy `RecursiveFileObserver` způsobí při vytvoření, změně a následném smazání souboru *test_file* v cestě `/sdcard` následující výpis. První záznam odpovídá vytvoření souboru, druhý záznam odpovídá změně obsahu souboru a poslední záznam odpovídá jeho smazání.

```
===== NEW ENTRY =====
Time: 333846
Type: File observer
Info: Event: CREATE, Path: /sdcard/test_file
Extras: none
===== NEW ENTRY =====
Time: 336379
Type: File observer
Info: Event: MODIFY, Path: /sdcard/test_file
Extras: none
===== NEW ENTRY =====
Time: 338431
Type: File observer
Info: Event: DELETE, Path: /sdcard/test_file
Extras: none
```

Výpis 6.12. Výpis událostí zachycených při změnách sledovaných souborů pomocí třídy *RecursiveFileObserver*.

6.5 SmsSentObserver a SmsReceiver

Při odeslání a přijetí SMS jsou zaznamenány tyto informace:

```
===== NEW ENTRY =====
Time: 281392
Type: SMS received
Info: Originating address: +420 xxx xxx xxx
Extras: none
===== NEW ENTRY =====
Time: 281568
Type: SMS sent
Info: Sent sms messages:
      ID: 1
      Address: xxx xxx xxx Date: 1432134066262
      Type: 2
Extras: none
```

Výpis 6.13. Výpis informací získaných při odeslání a přijetí SMS.

Telefonní čísla jsou v ukázce záměrně skryta a obsah SMS není zaznamenáván vůbec, nicméně i obsah SMS by mohl být užitečný k určení, zda je SMS výsledkem nějaké nežádoucí činnosti.

Problém při monitorování odeslaných SMS je v tom, že na některých zařízeních není možné přímo přistupovat jen k odeslaným zprávám, ale jen ke všem zprávám najednou. Při jakékoliv změně v databázi SMS se pak spustí daný observer a vypíše odeslané SMS, to však znamená, že nikde není informace o tom, jaká konkrétní zpráva byla odeslána, ale jednoduše máme seznam odeslaných SMS. Požadovanou informaci musíme zpětně zjistit.

6.6 NetCheck

Pro ověření sledování tcp připojení byl otevřen webový prohlížeč a zadána adresa https://www.google.com.

```
===== NEW ENTRY =====
Time: 2970799
Type: TCP check
Info: /proc/net/tcp entries:
Remote address: 216:58:213:4:443
Status: TCP_ESTABLISHED
Packages: com.android.browser
Remote address: 216:58:208:33:443
Status: TCP_ESTABLISHED
Packages: com.android.browser
Remote address: 216:58:213:4:443
Status: TCP_ESTABLISHED
Packages: com.android.browser
Extras: none
```

Výpis 6.14. Informace o otevřených spojení aplikace *com.android.browser*

6.7 Ransomware Simplocker ukázka

Pro testování událostí na zařízení s nainstalovaným malwarem jsem použil emulátor Genymotion, konkrétně obraz zařízení Galaxy Nexus s verzí Androidu 4.4.2. Na emulátor byl nainstalován malware *Simplocker* [22] se jménem balíčku *com.fbsmanager.umgr*. Zde je výběr zajímavých událostí zaznamenaných po instalaci a spuštění této klamné aplikace. Výstup byl zformátován k lepšímu čtení.

```
===== NEW ENTRY =====
Time: 394643
Type: Package added
Info: Package name: com.fbsmanager.umgr
Extras: UID: 10052, Replacing: false

===== NEW ENTRY =====
Time: 394683
Type: Permission check
Info: Package name: com.fbsmanager.umgr
Extras:
Permissions granted to com.fbsmanager.umgr:
android.permission.CALL_PHONE
```

```
android.permission.CAMERA
android.permission.WRITE_EXTERNAL_STORAGE
android.permission.INTERNET
android.permission.READ_PHONE_STATE
android.permission.RECEIVE_BOOT_COMPLETED
android.permission.SEND_SMS

===== NEW ENTRY =====
Time: 394708
Type: Query intent check
Info: none
Extras:
ACTIVITY INTENTS
Intent: android.app.action.ADD_DEVICE_ADMIN
Package: com.android.settings.DeviceAdminAdd
BROADCAST INTENTS
Intent: android.intent.action.NEW_OUTGOING_CALL
Package: cz.cvut.pisky.monitor.receivers.CallReceiver
Intent: android.app.action.DEVICE_ADMIN_ENABLED
Package: com.fbsmanager.umgr.GhhjIr
Intent: android.provider.Telephony.SMS_RECEIVED
Package: com.android.mms.transaction.PrivilegedSmsReceiver
Package: cz.cvut.pisky.monitor.receivers.SmsReceiver

===== NEW ENTRY =====
Time: 855764
Type: Package changed
Info: Package name: com.fbsmanager.umgr
Extras: UID: 10052 Component list: com.fbsmanager.umgr.Main

===== NEW ENTRY =====
Time: 895847
Type: TCP6 check
Info: /proc/net/tcp6 entries:

Remote address: D83A:D02E:0000:FFFF:0000:0000:0000:0000:0050
Status: TCP_CLOSE_WAIT
Packages:
com.android.keychain com.android.providers.settings
com.android.inputdevices com.android.settings android

Remote address: 8CD3:0F60:0000:FFFF:0000:0000:0000:0000:1466
Status: TCP_ESTABLISHED
Packages: com.fbsmanager.umgr
Extras: none

===== NEW ENTRY =====
Time: 955965
Type: TCP6 check
Info: /proc/net/tcp6 entries:

Remote address: 616B:865A:0000:FFFF:0000:0000:0000:0000:1466
Status: TCP_SYN_SENT
```

```
Packages: com.fbsmanager.umgr

Remote address: D83A:D02E:0000:FFFF:0000:0000:0000:0050
Status: TCP_CLOSE_WAIT
Packages:
com.android.keychain com.android.providers.settings
com.android.inputdevices com.android.settings android
Extras: none

===== NEW ENTRY =====
Time: 1015141
Type: Service check
Info: Number of running services: 16
Extras:
Name: com.android.phone.TelephonyDebugService
PID: 566 UID: 1001 Active since: 22184,
Name: com.fbsmanager.umgr.RJLHgVP
PID: 1286 UID: 10052 Active since: 845714,
Name: com.android.bluetooth.hdp.HealthService
PID: 907 UID: 1002 Active since: 26911,
Name: com.android.systemui.ImageWallpaper
PID: 481 UID: 10043 Active since: 21159,
Name: cz.cvut.pisky.monitor.service.InitService
PID: 1091 UID: 10051 Active since: 28924,
Name: com.android.systemui.SystemUIService
PID: 481 UID: 10043 Active since: 20826,
Name: cz.cvut.pisky.monitor.service.ScreenTouchService
PID: 1091 UID: 10051 Active since: 28925
// zde zkráceno
```

Výpis 6.15. Vybrané informace zachycené po nainstalování *Simplockeru*.

První tři události jsou spuštěny ihned po nainstalování. Je vidět, že aplikace požaduje poměrně mnoho potenciálně nebezpečných povolení a dále je v události typu *Query intent check* vidět, že má zaregistrovaný receiver na intent k získání práv administrátora zařízení. To je častá praktika malware aplikací, které se pokouší o uzamčení obrazovky zařízení. Další podezřelou událostí je *Package changed* a to konkrétně změna komponenty `com.fbsmanager.umgr.Main`. Aplikace se tím snaží skrýt ikonu z aplikačního menu. Následují dva výpisy, na kterých je vidět, že aplikace má otevřené spojení. Jako poslední je výpis na pozadí běžících service komponent na systému a v nich i `com.fbsmanager.umgr.RJLHgVP`, kde je vidět nesmyslné jméno komponenty. Autoři malware totiž často vytvářejí více verzí malware, kde automaticky generují nové názvy, co nemají žádný smysl.

Při důkladnějším pohledu na sebraná data je v tomto případě vidět, že je relativně snadné potenciálně nebezpečnou aktivitu zazanamenat, ale v praxi označovat konrétní aplikace jako malware pomocí této metody je podle mého názoru moc riskantní a mohlo by vést k velkému množství *false positives*[1]).

[1]) V tomto kontextu to znamená chybné označení programu jako malware.

Kapitola 7
Vyhodnocení přínosu a efektivity řešení

Při sledování událostí si můžeme všimnout, že prakticky nikdy nemůžeme s úplnou jistotou říci, jaká aplikace tuto událost způsobila. Samozřejmě existují i případy, kdy můžeme s velkou pravděpodobností říci, že je na zařízení malware a označit aplikaci, kterou za malware považujeme. To se však jeví jako možné pouze v případech méně sofistikovaného malwaru, který se nesnaží svou aktivitu na zařízení skrývat, jeho účelem je dostat se na zařízení pod falešnou záminkou a tam jednorázově vykonat škodu 6.15. Mezi takové jednorázové aktivity patří uzamčení telefonu a následné vydírání uživatele nebo odeslání SMS zprávy na prémiová čísla, která si autor malwaru sám zaregistroval a tím pádem obdrží za tyto odeslané SMS peníze. Čím méně je malware sofistikovaný a snaží se jednorázově vykonat dané nežádoucí akce, tím více je k odhalení pomocí navrhované metody náchylný, protože je jednodušší zpozorovat podezřelou aktivitu ve sledovaných událostech.

Problém však nastává u sofistikovanějšího malwaru. Autoři malwaru ke svému účelu používají například legitimní aplikace, a to tím způsobem, že kód legitimní aplikace poupraví a pak jí nechají volně ke stažení na internetu. V sesbíraných datech pomocí navržené aplikace je pak daleko obtížnější odhalit, protože je těžší určit, která aplikace za dané události zodpovídá. Pokud je navíc přidaný kus kódu umístěn někde, kde není často či pravidelně volán, odhalení je ještě více ztíženo.

Dalším problémem je fakt, že pokud bychom chtěli na základě sledovaných událostí přímo určit, která aplikace je malware, ponese to s sebou pravděpodobně vysoký počet falešných poplachů (*false positives*), a to je v praxi velmi nežádoucí. Kromě toho může dojít i k chybnému identifikování škodlivého chování, jen díky náhodnému sledu událostí, které vypadají jako nežádoucí, ale ve skutečnosti se o škodlivou aktivitu nejedná.

Z výše uvedených důvodů nepovažuji sledování událostí jako vhodného kandidáta k přímé detekci malwaru, ale spíše jako potenciálního pomocníka a ukazatel rizika, či pravděpodobnosti, že se malware na zařízení nachází. Zajímavým tématem by bylo určení této pravděpodobnosti na základě strojového učení. Použití strojového učení k odhalení malwaru není v antivirovém softwaru neobvyklé.

Sledování událostí by také mohlo sloužit jako pomocník při získávání potenciálně nežádoucích vzorků aplikací k podrobnější analýze.

Kapitola 8
Závěr

Cílem této bakalářské práce byl průzkum sledovatelných událostí na platformě Android a jejich využití k odhalení potenciálně škodlivého chování aplikací. Praktickou částí této práce bylo vytvoření prototypu aplikace pro Android za účelem sběru vybraných událostí, které by mohly chování malware charakterizovat. Dále bylo cílem zjistit, do jaké míry je tato metoda užitečná k detekci podezřelého chování aplikací na zařízení.

Při průzkumu sledovatelných událostí jsem vycházel především z oficiální dokumentace, dále jsem na základě informací o chování malwaru pro Android vybral z událostí ty, které mají potenciál určitá nebezpečená chování definovat. Na základě tohoto výběru byl navržen a implementován prototyp aplikace pro sběr vybraných událostí. Aplikace byla otestována na příkladovém malwaru *Simplocker* a bylo ukázáno, že při analýze sebraných událostí lze potenciální hrozbu identifikovat. Problémem přímé detekce malwaru pomocí navrhované metody se však jevil potenciál ke vzniku *false positives*. I když není tato metoda k přímé detekci vhodná, její využití spočívá spíše v nalezení potenciálních hrozeb a zjištění míry podezřelého chování na zařízení.

Domnívám se, že aby bylo možné přínos a využití této metody ještě přesněji a s větší jistotou ověřit, bylo by nutné mnohem rozsáhlejší testování. Dalším navazujícím krokem k této práci a k využití navržené metody v praxi je tedy testování použité metody na větším počtu zařízení a s daleko větší a rozmanitější skupinou vzorků. Důkladnější analýzou výstupů takového testování by pak bylo možné stanovit podrobnější kritéria pro podezřelé chování aplikací. Nakonec by zbývalo řešení nasadit v experimentálním režimu přímo do produkce, aby bylo možné získat a analyzovat reálná data.

I přes všechny problémy díky kterým se tato metoda ve výsledku jeví jako poměrně nespolehlivá, věřím, že je možné s její pomocí rozpoznat určité příznaky malwarového chování a podrobit aplikace na potenciálně infikovaném zařízení hlubšímu zkoumání. Minimalně lze tímto způsobem získat zdroj potenciálně škodlivých vzorků.

Seznam použitých zdrojů

[1] WIKIMEDIA FOUNDATION, Inc. *Android (operating system)* [online]. [cit. 2015-05-02]. Dostupné z:
http://en.wikipedia.org/wiki/Android_%28operating_system%29

[2] VLČEK, Ondřej. As Mobile Malware Hits the Million Samples Mark It Becomes More Devious than Ever Before. In: *Avast blog* [online]. 2014. vyd. [cit. 2015-04-07]. Dostupné z:
https://blog.avast.com/2014/09/09/mobile-malware-becomes-more-devious/

[3] CHIEN, Eric. SYMANTEC CORPORATION. *Motivations of Recent Android Malware* [online]. [cit. 2015-05-04]. Dostupné z:
https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/motivations_of_recent_android_malware.pdf

[4] DUNHAM, Ken, Shane HARTMAN, Jose Andre MORALES, Manu QUINTANS a Tim STRAZZERE. *Android malware and analysis*. xx, 222 pages. ISBN 1482252198.

[5] GOOGLE INC. [online]. [cit. 2015-04-07]. Dostupné z:
https://play.google.com/intl/ALL_cz/about/developer-content-policy.html

[6] APVRILLE, Axelle, Ruchna NIGAM a Martijn GROOTEN. *Obfuscation in Android malware, and how to fight back* [online]. [cit. 2015-05-10]. Dostupné z:
https://www.virusbtn.com/virusbulletin/archive/2014/07/vb201407-Android-obfuscation

[7] TECHOTOPIA. *An Overview of the Android Architecture* [online]. [cit. 2015-05-13]. Dostupné z:
http://www.techotopia.com/index.php/An_Overview_of_the_Android_Architecture

[8] VOGEL. *Introduction to Android development* [online]. 2015-04-12 [cit. 2015-04-07]. Dostupné z:
http://www.vogella.com/tutorials/Android/images/xandroidsoftwarelayer10.png.pagespeed.ic.cqHvO0-Uh9.png

[9] GOOGLE INC. *Android Developers* [online]. [cit. 2015-04-25]. Dostupné z: https://developer.android.com/index.html

[10] ZHOU, Yajin a Xuxian JIANG. An Analysis of the AnserverBot Trojan. [online]. 2011 [cit. 2015-04-07]. Dostupné z:
http://www.csc.ncsu.edu/faculty/jiang/pubs/AnserverBot_Analysis.pdf

[11] *Android: FileObserver monitors only top directory* [online]. [cit. 2015-05-11]. Dostupné z:
http://stackoverflow.com/questions/16448002/android-fileobserver-monitors-only-top-directory

[12] GRUBER, Eric. *Android Root Detection Techniques* [online]. 2013-12-02 [cit. 2015-05-08]. Dostupné z: https://blog.netspi.com/android-root-detection-techniques/

[13] KEVIN. *Determine if running on a rooted device* [online]. [cit. 2015-05-08]. Dostupné z:
http://stackoverflow.com/questions/1101380/determine-if-running-on-a-rooted-device

[14] *Android Logging System* [online]. [cit. 2015-05-10]. Dostupné z:
http://elinux.org/Android_Logging_System

[15] Android Logging System. KOBABAYSHI, Tetsuyuki. KYOTO MICROCOMPUTER CO. [online]. [cit. 2015-05-10]. Dostupné z:
http://elinux.org/images/c/c9/Android-logging-kmc-kobayashi.png

[16] SHI, Jinghao. *Monitor Screen Touch Event in Android* [online]. [cit. 2015-05-10]. Dostupné z:
http://jhshi.me/2014/11/09/monitor-screen-touch-event-in-android/

[17] SCHLEGEL, Roman, Kehuan ZHANG, Xiaoyong ZHOU, Mehool INTWALA, Apu KAPADIA a XiaoFeng WANG. *Soundcomber: A Stealthy and Context-Aware Sound Trojan for Smartphones* [online]. [cit. 2015-05-04]. Dostupné z:
http://homes.soic.indiana.edu/zhou/files/soundcomber-ndss11.pdf

[18] CHEETAH MOBILE. *Hideicon malware hits Google Play* [online]. [cit. 2015-05-13]. Dostupné z:
http://www.cmcm.com/blog/en/security/2015-01-21/531.html

[19] *Contagio Mobile: Mobile malware mini dump* [online]. [cit. 2015-05-15]. Dostupné z:
http://contagiominidump.blogspot.cz/

[20] PHILLIPS, Bill a Brian HARDY. *Android programming: the Big Nerd Ranch guide.* 1st ed. xxii, 602 pages. ISBN 03-218-0433-3.

[21] ASLAN, Erdem. GOOGLE, Inc. *Google Play - Electronic Function Generator* [online]. [cit. 2015-05-10]. Dostupné z:
https://play.google.com/store/apps/details?id=com.erdemaslan.functiongenerator

[22] CHRYSAIDOS, Nikolaos. AVAST SOFTWARE. *Mobile Crypto-Ransomware Simplocker* [online]. [cit. 2015-05-11]. Dostupné z:
https://blog.avast.com/2015/02/10/mobile-crypto-ransomware-simplocker-now-on-steroids/

Příloha A
Seznam zkratek

ADB	Android Debug Bridge
ADT	Android Developer Tools
API	Application Programming Interface
APK	Android application package
ART	Android Runtime
DVM	Dalvik virtual machine
HTTP	Hypertext Transfer Protocol
IMEI	International Mobile Station Equipment Identity
IMSI	International Mobile Subscriber Identity
IPC	Inter-process communication
JAR	Java Archive
JVM	Java virtual machine
PID	Process ID
ROM	Read-only memory
SD	Secure Digital
SGL	Skia Graphics Library
SIM	Subscriber Identity Module
SMS	Short Message Service
SQL	Structured Query Language
SSL	Secure Sockets Layer
SU	super user
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UI	User interface
UID	User ID
URI	Uniform resource identifier
URL	Uniform resource locator
WAP	Wireless Application Protocol
XML	Extensible Markup Language

Příloha B
Obsah přiloženého CD

```
* ---- cz.cvut.pisky.monitor.apk
   |-- Monitor.zip
   |-- piskaja2_bp.pdf
```

Vytvořená aplikace *cz.cvut.pisky.monitor.apk*, projekt vytvořený v prostředí *Android Studio* Monitor.zip a soubor piskaja2_bp.pdf s touto prací.