Платформи и модели за мобилна сигурност

Росен Пасарелски*, Антони Славински*

* Нов Български Университет София, Департамент „Телекомуникации“,
София 1000, България, ул. „Монтевидео“ 21,
e-mail: rpasarelski@mail.bg

Резюме. *Целта на тази статия е да представи основните мобилни платформи на водещи компании в ИТ сферата като – Apple, Microsoft, Google и да разгледа и анализира нивото на мобилна сигурност. Главен акцент се поставя на най-популярната и разпространена операционна система за мобилни смарт устройства – Андроид. В доклада се представят и анализират уязвимости в платформата и се предлагат и изследват техники и способи за тестване и предотвратяване на възможните пробиви и последствия за крайните потребители.*

Platforms and models for mobile security (Rosen Pasarelski, Antoni Slavinski - *New Bulgarian University Sofia, Department of Telecommunications, Sofia 1000, Bulgaria, 21, Montevideo Str., e-mail: rpasarelski@mail.bg). Resume. The purpose of this article is to present the major mobile platforms of leading IT companies such as Apple, Microsoft, Google and review and analyze the level of mobile security. The main focus is placed on the most popular and popular operating system for mobile smart devices - Android. The report presents and analyzes vulnerabilities in the platform, proposing and investigating techniques and techniques for testing and preventing possible breakthroughs and the ultimate end-users.*

1. Въведение

Развитието на мобилните мрежи и услуги води несъмнено до генезис и при мобилните устройства предназначени за крайните потребители. Мобилните системи в своята същност са дефинирани за телефония, даваща възможност на абонатите да разговарят помежду си на далечно разстояние. С появата на мобилният интернет и пакетния пренос на данни се променят коренно нещата в мрежата. Мобилните телефони стават интелигентни – смартфони и все по-често се използват от потребителите за разглеждане на страници в Интернет, за Facebook или Viber приложения и други. Мобилните телефони се трансформират в джобни компютри с мощен хардуер и софтуер. За тяхното цялостно функциониране започват да отговарят операционни системи и платформи на водещи компании в ИТ сферата като – Apple, Microsoft, Google. Като пионер в областта на мобилните смарт устройства компанията Apple въвежда своя операционна платформа наречена iOS. Тя е предназначена само за нейните мобилни устройства съответно смартфоните – iPhone, таблетите – iPad и мобилните музикални плеъри – iPod. Операционната система iOS е затворена платформа и не всеки програмист има достъп до нея да я развива и разработва.

Компанията Microsoft е водеща в софтуерната компютърна индустрия и заема най-голям пазарен дял при настолните компютри и лаптопите. В областта на интелигентните телефоните не успява да се наложи като лидер, но развива свои мобилни смартфони и отделна операционна система наречена Windows Mobile. Платформата на Microsoft също е затворена операционна среда като iOS на Apple.

Мобилната операционна система Андроид (Android) е реализирана от Google през 2007г. Тя е представена от компанията като платформа, поддържаща обширна гама от устройства, включваща мобилни телефони, таблети, телевизори, ноутбуци и други. До настоящият момент Андроид е най-популярната и разпространена мобилна операционна система, като заема дял повече от 65% от световния пазар на интелигентни смартфони. Платформата Android е с отворен изходен код, което я прави достъпна за всеки програмист, който желае да я развива и

разработва. Заинтересованите могат да изтеглят и да реализират своя собствена система (source.android.com/source/downloading.html).
Приложенията на Google, предвидени за голяма част от смарт телефоните с Android, са със затворен код, което прави платформата не изцяло отворена. Много производители на мобилни устройства модифицират изходния код на Android, за да прилегне по-добре на техния хардуер или мобилна мрежа. Това означава, че много устройства включват собствени драйвери със затворен код, а също така и приложения, което е проблем с актуализирането им с новите версии на Android. Всичко това води до проблем с платформата, а именно че много различни версии на Android работят с различни конфигурации на същия софтуер в разнообразни устройства. С други думи казано два мобилни смартфона с един и същ хардуер, но на различни мобилни оператори и мрежи могат да работят с много различаващ се софтуер. На това може да се акцентира като важен проблем на сигурността в платформата Андроид.

2. Платформи и модели за мобилна сигурност

2.1 Модел за сигурност на iOS

Платформата iOS претърпява значително развитие през годините от своето създаване. Основно генезиса й е по отношение на структурата на операционната система и модела за сигурност. Когато смартфонът - iPhone е пуснат за първи път на пазара, компанията Apple заявява публично, че е реализиран по начин, по който не възнамерява да разрешава на приложенията на трети страни да действат на устройството. Разработчици и потребители са инструктирани да изграждат или използват уеб приложения и да имат достъп до тези приложения чрез вградения уеб браузър на iPhone. За известно време това означава, че само с направен от Apple софтуер може да работи устройството и съответно изискванията за сигурност са много по-малко. Липсата на приложения от трети страни пречи на потребителите да се възползват максимално от iPhone. Това предизвиква находчиви програмисти да започнат да намират начини за рутване или разбиване (jailbreak) на устройствата и възможности за инсталиране на софтуера на трети страни. В отговор на това и на потребителското търсене през 2008г. компанията Apple представя актуализирана версия на iOS, която включва поддръжка за нова услуга известна като App Store. В App Store се предлага на потребителите възможността да намерят и инсталират приложения на трети страни. Услугата App Store изживява страхотен бум с милиони пуснати приложения и милиарди тегления. С това Apple започва да включва допълнителни мерки за мобилна сигурност в операционната система. Ранните версии на iOS осигуряват малко по отношение на защитата. Всички процеси действат с root-привилегии за суперпотребител. Процесите не са тествани и ограничени по отношение на това какви системни ресурси могат да достъпват. Подписването на кода не е използвано за потвърждаване произхода на приложенията и за контролиране изпълнението на посочените приложения. Не са предоставени функции като Address Space Layout Randomization (ASLR) или Position Independent Executable (PIE) с поддръжка за ядрото, системни компоненти, библиотеки или приложения. Реализирани са и изключително малко хардуерни контроли за предотвратяване на хакерски атаки върху устройствата. С течение на времето Apple започва да въвежда подобрена функционалност за мобилна сигурност. Приложенията на трети страни се изпълняват под по-малко привилегирован потребителски акаунт, наречен мобилен. Добавена е поддръжка за тестване, която ограничава приложенията до редуциран набор от системни ресурси. Вградена е поддръжка за проверка на кода. С това допълнение приложенията е необходимо да бъдат проверени и утвърдени от Apple, за да е позволено тяхното изпълнение. В крайна сметка се въвеждат и функциите ASLR за ядрото, за други компоненти на операционната система и библиотеките, както и възможност за времево компилиране на Xcode известно като PIE. Функцията PIE изисква приложението да се зарежда с различен базов адрес при всяко изпълнение, което прави експлоатацията на уязвимости на конкретното приложение доста по-трудни.

Всички тези промени и подобрения в платформата iOS я прави водеща по отношение на мобилната сигурност и уязвимостите в операционната система.

В статията основен обект на изследване и анализиране е моделът за сигурност на най-популярната и използвана мобилна платформа – Android.

2.2 Андроид модел за сигурност

Архитектурата на платформата Android се състои от четири основни слоя:

- Linux ядро.
- Собствени библиотеки /Виртуална Dalvik машина (Dalvik Virtual Machine).
- Приложна работна рамка (Application Framework)
- Приложен слой.

Ядрото базирано на Linux предоставя същата функционалност за Android, която прави и в Linux среда - дава възможност на приложенията да взаимодействат с хардуерни компоненти, както и да управляват процесите и паметта. Ядрото играе важна роля в модела за сигурност на платформата.

Следващият слой се състои от собствени библиотеки, които осигуряват достъп до функционалност използвана от приложенията за Android, като например OpenGL за 2D / 3D графики, SQLite бази данни и други. В този слой са включени и виртуалната Dalvik машина и опорни Java библиотеки, като съставят Андроид Runtime компонен, осигуряващ основна функционалност използвана от Java приложения.

Следващото ниво е работната рамка за приложения. Тя осигурява способ за достъп на приложенията до различна функционалност като:

- телефонна функционалност - осъществяване/получаване на повиквания, кратки съобщения, достъп до Интернет.
- Функционалност свързана със създаване на елементи в потребителския интерфейс.
- Функционалност за достъп до ресурсите на файловата система.
- Функционалност за достъп до GPS и други.

Приложната рамка играе важна роля в модела за сигурност на платформата Андроид.

Последният слой в архитектурата е слоя за приложения. Тези приложения обикновено се пишат на програмния език Java и се компилират в Dalvik байт-код, като се използва комплекта за разработка на софтуер (SDK). Създадените приложения комуникират с по-горните слоеве в архитектурата на Андроид операционната система.

Моделът за защита на платформата Android е базиран на разрешения. Това означава, че дадено приложение, за да извърши каквото и да е действие е необходимо изрично да му бъде предоставено разрешение за изпълнение. Тези разрешения се изпълняват на две нива в архитектурата на операционната система Android:

- на ниво – ядро
- на ниво - приложение.

В началото на доклада се представя как ядрото обработва разрешенията и как това добавя сигурност към платформата.

Ядрото на операционната система Android като основа е Linux базирано. То осигурява сигурност, използвайки идеята за контрол на достъпа, основаващ се на потребители и групи. Различните ресурси и операции, до които ядрото предоставя достъп се ограничава въз основа на разрешенията на потребителя. Тези разрешения могат да бъдат фино настроени, за да се даде на потребителя достъп само до определени ресурси. В Android всички приложения получават уникален потребителски идентификатор. Това ограничава приложенията до достъп само до ресурси и функционалност, на които им е било изрично разрешено. По този начин се гарантира, че приложенията не могат да имат достъп до ресурсите на други приложения въз основа на собствеността на файла, определени от идентификационния номер на потребителя. Също така се обезопасява достъп до хардуерни компоненти, за които не са получили разрешение за използване.

Приложната рамка осигурява друго ниво на контрол на достъпа. За достъп до ограничената функционалност, предоставена от Application Framework, приложението за Android трябва да декларира разрешение за този компонент в неговия файл-манифест (AndroidManifest.xml). Тези заявени разрешения след това се показват на потребителя по време на инсталиране, давайки му възможност да инсталира приложението с исканите разрешения или изобщо да не инсталира приложението. След като приложението бъде инсталирано, то се ограничава до компонентите, за които има разрешение за използване. Например приложение, което изисква разрешение от android.permission.INTERNET, може да отвори връзка с Интернет.

Към настоящият момент са дефинирани голям брой разрешения за Android, като списък може да се види на интернет адреса: developer.android.com/reference/android/Manifest.permission.html. Тези разрешения са за използване на базовата функционалност на Android. Освен това, приложенията могат да определят свои собствени разрешения, което означава, че реалният брой разрешения, налични на устройство с Android, може да бъде в стотици. Тези разрешения могат да бъдат разделени на четири основни категории:

- **нормални** разрешения - с нисък риск, които

дават достъп до нечувствителни данни или функции. Тези разрешения не изискват изрично одобрение от потребителя по време на инсталиране.

- **Опасни** - тези разрешения предоставят достъп до важни функции и данни и изискват изрично одобрение от потребителя по време на инсталиране.
- Разрешение тип - **подпис** - този клас разрешение може да бъде дефиниран от приложението в неговия файл-манифест. Функционалността, издадена на приложения, които декларират това разрешение, може да бъде достъпна само от други приложения, които са подписани от същия сертификат.
- Разрешение тип - **Подпис или Система** - това е разрешениие идентично с разрешението тип - подпис, но приложенията инсталирани на /системния дял на устройството, имащи повишени привилегии, също могат да достъпват тази функционалност.

От гореказаното можем да отчетем, че всички приложения за Android трябва да бъдат подписани, за да се инсталират. Платформата Android позволява само подписани сертификати, така че разработчиците могат да генерират собствен сертификат, за да подписват своите приложения.

Освен сигурността на ниво приложение, Андроид осигурява някои допълнителни мерки за сигурност. Администраторски интерфейс (ASLR) е добавен в операционната система, за да затрудни атаки на нарушители свързани с проблеми с компрометиране на паметта. Интерфейсът ASLR включва случайно определяне на местоположението на ключови секции от паметта. Друга защита в паметта е битът No eXecute (NX) или неизпълним бит. Това позволява да направите настройки на секциите да не се изпълняват, което помага да се предотвратят атаки за компрометиране на паметта.

2.2.1 Компоненти на приложенията за Андроид

Приложението за платформата Андроид се състои от четири различни типа компоненти. Всеки компонент на приложението представлява различна входна точка, в която системата или друго приложение на едно и също мобилно устройство може да влезе. Колкото повече компоненти са експортируеми (Android: изнесени), толкова по-сериозна е възможността за атака, тъй като тези компоненти могат да бъдат извиквани от други потенциално злонамерени приложения. Приложенията използват предимно намерения, които са асинхронни съобщения, за осъществяване на междупроцесорна или вътрешно компонентна комуникация. Компонентите на приложенията могат да бъдат представени, както следва:

- Дейности - определят самостоятелен екран на потребителския интерфейс на приложението.
- Доставчици на съдържание - предоставят възможността за заявяване, вмъкване, актуализиране или изтриване на специфични за приложението данни към други приложения и вътрешни компоненти. Приложението може да съхранява действителните данни в SQLite база данни или във файл. Некоректно написаните доставчици на съдържание са потенциална заплаха за външна атака като са изложени на враждебни приложения или са уязвими към SQL инжекции или други видове хакерски атаки.
- Броудкаст приемници - отговарят на намеренията за броудкаст-излъчване. Приложенията не трябва да се доверяват напълно на получените данни от намеренията за броудкаст-излъчване, защото може да са изпратени от недоброжелателно приложение или данните да са възникнали от отдалечена система.
- Услуги - работят във фонов режим и извършват продължителни операции. Услугите се стартират от друг компонент, когато той изпраща намерение. Това ни дава основание да твърдим, че услугите също не е редно да се доверяват безкрайно на данните, съдържащи се в намерението.

2.2.2 Съхранение на данни

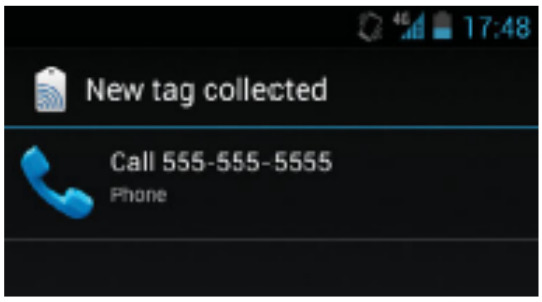
За съхранение на данни Андроид приложенията могат да използват вътрешно хранилище, като съхраняват данни в енергонезависима флаш памет (NAND flash) или използват външна памет за съхранение - SD карта. По принцип файловете съхранявани на външна памет са обществено достъпни за всички приложения. Файловете във вътрешното хранилище по подразбиране са частни за конкретно приложение. Изключение може да бъде единствено ако самото приложение промени

разрешенията на файловете по подразбиране за Linux. Също така съхраняването на всякакви важни и чувствителни данни без правилно използване на криптографски механизми върху мобилното устройство, независимо от това дали приложението използва вътрешно или външно хранилище, може да доведе до проблеми с изтичането на информация.

Приложенията за Андроид могат свободно да създават какъвто и да е тип файл, но Android API се предлага с поддръжка на SQLite бази данни и файлове със споделени преференции, съхранявани в XML-базиран формат. Поради това често се забелязват тези типове файлове, докато се преглеждат частните данни или обществените данни, свързани с целевото приложение. От гледна точка на сигурността, използването на релационни бази данни от страна на клиента очевидно въвежда възможността за SQL (инжекции) атаки срещу Андроид приложения чрез намерения или други входове, като например мрежов трафик и т.н.

2.2.3 Близки по поле комуникации - NEAR FIELD COMMUNICATION (NFC)

Така наречените близки по поле комуникации (NFC) описват набор от стандарти за радиокомуникации между устройствата. Тези устройства включват NFC тагове, подобни на RFID етикетите и някои RFID тагове, безконтактни смарт карти и мобилни устройства. Устройствата за NFC комуникират в много малък диапазон от няколко сантиметра. На фигура 1 е представен NFC маркер, съдържащ телефонен номер, който се чете от устройство с Android операционна система.



Фиг.1 Четене на NFC таг с Android телефон

Стандартът NFC влезе в Андроид още през 2010г. с пускането на версията Gingerbread. Първият NFC-съвместим Андроид телефон е Samsung Nexus S. Началното изпълнение на NFC е доста ограничено. Разширение настъпва с пускането на Android версия 2.3.3. Дотогава Android поддържа четене и писане на различни формати на NFC тагове. При Android версия 2.3.4 се стига до режим на емулация на карти, който позволява на мобилното устройство да подражава на NFC смарткарта, така че друг NFC четец да може да чете данни от защитния елемент (SE), който се намира вътре в устройството. Първото приложение за използване на режима на емулация на карти е Google Wallet, пуснато с Android версия 2.3.4. При версия 4.0 се добавя режим "peer to peer" (p2p), който позволява на две устройства с възможност за NFC да комуникират директно. Имплементацията в платформата Android на тази способност се нарича Android Beam и позволява на потребителите да споделят данни между приложения, като докосват заедно устройствата си.

Понастоящем NFC се използва за различни цели, включително мобилни плащания с Google Wallet. NFC таговете се използват в рекламите и с пускането на Android 4.1, повече приложения поддържат Android Beam за трансфер на данни.

2.2.4 Развитие на платформата Android

Компанията Google предоставя комплект за разработка на софтуер (SDK), който позволява на разработчиците да изграждат и дебъгват приложения за Android (developer.android.com/sdk/index.html).

Комплектът SDK на Android е налице на различни платформи като Windows, Mac OS X и Linux. Всеки, който се интересува от откриването и използването на уязвимости в операционната система Android и в приложенията за Android, може да се запознае със SDK и свързаните с него инструменти. Тези инструменти са полезни, както за разработчиците на Android, така и за изследователите по сигурността.

- **Емулатор за Андроид**

Комплектът SDK на Андроид предоставя виртуален емулатор за мобилни устройства (developer.android.com/tools/help/emulator.html), който позволява на разработчиците да тестват своите приложения за Android без реално мобилно устройство, представено на фигура 2.

Емулаторът симулира хардуерни функции, които са общи за повечето мобилни устройства с операционна система Android, като например процесор ARMv5, симулирана SIM карта и дялове на флаш паметта. Емулаторът дава на разработчиците и изследователите в областта на

сигурността възможността бързо да тестват приложенията за Android в различни версии на платформата, без да е необходимо да притежават голям брой мобилни устройства.

Фиг.2 Емулатор за Android

Въпреки, че емулаторът със сигурност е ценен инструмент, съществуват редица значителни недостатъци при извършването на тестове за сигурност с емулатор. Например, виртуално устройство с Android (AVD) не може да получава или да извършва действителни телефонни разговори или да изпраща или получава SMS съобщения. Ето защо не е препоръчително използването на емулатора за тестване на приложения, които изискват комуникация през мобилната мрежа (например приложения, които могат да получават символи за сигурност или еднократни пароли чрез SMS). Чрез осъществяване на телефония и SMS емулация, е възможно да се изпращат SMS съобщения до целево приложение, със задачата да се види как приложението обработва входа или дали няколко виртуални устройства комуникират помежду си. Други полезни функции на емулатора включват възможността за дефиниране на HTTP / HTTPS прокси сървър и възможност за пренасочване на мрежовия порт, за да се пресече и манипулира трафика между целевото приложение, изпълняващо се в емулатора, и различните крайни точки за уеб услуги.

➢ **Мост за дебъг в Андроид**

Мостът за дебъг в платформата Андроид - Android Debug Bridge (ADB) е инструмент на командния ред, който позволява да се комуникира с мобилно устройство чрез USB кабел или AVD, работещи в емулатор, както е представено на Фигура 3. Клиентът на ADB се свързва с демона на устройството, работещ на TCP порт 5037.

Фиг. 3 Мост за дебъг в Android

Мостът ADB предлага голям брой команди, но най-полезните при тестване на сигурността на конкретно приложение могат да бъдат представени, както следва:

- Install - копира файловата папка на приложението (APK), което е форматът на файла използван от Google за разпространение на приложения към мобилното устройство и инсталира приложението.
- logcat - показва информация за регистрацията в конзолата. Полезна за използване команда ако дадено приложение или основната операционна система записва важна информация.
- Pull - копира файл от мобилното устройство във файловата система.
- Push - копира файл от файловата система към мобилното устройство.
- Shell - стартира отдалечена обвивка на мобилното устройство, което позволява да се изпълняват произволни команди.

2.2.5 Уязвимости в Андроид

По напред в статията бе споменато, че ресурсите, до които има достъп приложението са ограничени от модела за сигурност на Андроид. Има достъп само до файлове, които притежава или файлове на външната SD памет и достъп само до функционалност на устройството, която е поискана и потвърдена по време на инсталацията чрез манифест файла на Android. Този модел предотвратява възможности злонамерени приложения да извършват нежелани действия или да имат достъп до важни данни. Ако дадено приложение може да работи под основен root-потребител, този модел за сигурност се разпада.

➢ **Рутване (Rooting)**

Дадено приложение, изпълнявано под основния root-потребител, може да има директен достъп до ресурсите на устройството, като заобикаля

необходимите проверки за разрешения и потенциално да има пълен контрол над устройството и останалите инсталирани приложения. Въпреки, че Андроид общността има тенденция да вижда рутването като начин потребителите да придобият повече контрол над устройството си, да могат да инсталират допълнителен софтуер или дори персонализиран ROM на платформата. Заплахата от това идва обаче, че и злонамерено приложение може да използва същите техники, за да придобие контрол над дадено устройство. Като примери могат да се представят няколко популярни революционни пробива (експлойта):

- **Пробивът GingerBreak**

Експлойтът GingerBreak е открит от групата - The Exploid Crew през 2011г. Той предоставя метод за придобиване на основни - root права на много Android устройства с версия Gingerbread Android 2.3.x и някои устройства с Froyo 2.2.x и Honeycomb 3.xx. Този конкретен експлойт продължава да бъде популярен поради броя на устройствата, които все още използват версията Gingerbread.

Пробивът в GingerBreak работи, като използва уязвимост в мениджър демона / system / bin / vold. Има метод - DirectVolume :: handlePartitionAdded, който задава индекс на масив, като използва цялото число, което му се предава. Методът прави проверка на максималната дължина на цялото число, но не проверява дали числото е с отрицателна стойност. Чрез изпращане на съобщения, съдържащи отрицателни цели числа към vold чрез Netlink socket, кодът на експлойта може да има достъп до произволни местоположения на паметта. След това кодът се записва в глобалната офсетова таблица (GOT), за да се презапишат няколко функции (като strcmp () и atoi ()) с повиквания към системата (). След това с друг разговор към vold, може да се изпълни друго приложение чрез системата (), с повишените привилегии на vold, тъй като vold е в / системния дял. В този случай кодът на експлойта извиква sh и продължава до remount / system с права за четене и писане - read/writable, което позволява суперпотребител-su и всяко друго приложение да бъде инсталирано.

Уязвимостта в GingerBreak е била опакована в няколко популярни инструмента за извличане на информация като SuperOneClick и са създадени и някои приложни пакети за свързване с едно кликване.

- **Уязвимост на init chmod / chown при Ice Cream Sandwich**

Този метод на рутване на мобилни устройства за първи път подлежи на обсъждане на форума xda-developers от потребителя wolf849 (forum.xda-developers.com/showthread.php?t=1622628).

Уязвимостта на init е представена в началото на пускането на версията на Android 4.0.x Ice Cream Sandwich. В случай, че скриптът init.rc има запис като следния

mkdir / data / local / tmp 0771 shell shell

init ще зададе собствеността и разрешенията на директорията на shell-потребител (ADB потребител), дори ако командата mkdir не е успешна.

Ако устройството е конфигурирано така, че / data / local са с права за писане от shell-потребителя е възможно да се създаде символна връзка в / data / local към друга директория като / system. Когато устройството се рестартира init се опитва да създаде директория и се проваля, но все пак задава разрешения определени в init.rc. За пример можем да представим следното:

- ако предходният ред бе в init.rc, създавайки

връзката от / local / data / tmp към / system това би позволило на shell-потребителя достъп за четене/писане до системния дял след рестартиране на устройството:

ln -s / system / data / local / tmp .

След като shell-потребителят има достъп за четене/писане на системния дял / system, атакуващият може да използва debugfs инструмент за добавяне на файлове към системния дял като суперпотребител - su. Този метод се използва за получаване на основни root-права до различни мобилни устройства, включително и един от най-разпространените телефони Samsung Galaxy S3.

- **Прихващане на мрежовия трафик**

За да се идентифицират уязвимости като SQL инжекции или заобикаляне на автентификацията в бек-енда на уеб услугите, с които взаимодействат приложенията за Андроид, е необходимо първо да се наблюдават и изследват и след това да се манипулира мрежовия трафик. Тук се акцентира върху прихващането на HTTP или HTTPS трафик, тъй като повечето приложения използват тези протоколи. При начален анализ на сигурността се започва с използването на инструменти за подслушване на мрежата като tcpdump или Wireshark.

- **Добавяне на надеждни CA сертификати**

Повечето приложения за Андроид, за които се твърди, че са сигурни, използват TLS за намаляване на риска от атаки от типа човек по средата (man-in-the-middle) и правилно проверяват и валидират сертификат. Ето защо е необходимо да добавим свои собствени доверени CA сертификати на Android устройството, преди действията по прихващане и манипулиране на HTTPS трафика. Това няма да доведе до причиняване на грешка по време на фаза на преговорите за ръкостискане на TLS. Платформата Android поддържа DECT-кодирани X.509 сертификати, използващи разширението .crt, както и сертификати X.509, запазени в PKCS#12 файлове с разширение .p12.

- **Атаки базирани на намерения**

Намеренията са основният метод за комуникация между процесите (IPC) използван от приложенията в платформата Android. Приложенията могат да изпращат намерения за стартиране или предаване на данни към вътрешни компоненти или намерения до други приложения. Когато дадено приложение изпраща външно намерение, операционната система Android го обработва, като търси инсталирано приложение с дефиниран филтър за намерение, който съответства на намерението за броудкаст. Ако намери съответстващ филтър за намерение, то намерението се доставя към приложението. Ако приложението в момента не работи, това го стартира. Филтърът за намерение може да бъде много специфичен с потребителски разрешения и действия или може да бъде общ (например android.provider.Telephony.SMS_RECEIVED). Ако Android намери повече от едно приложение със съвпадащ филтър за намерения, той подканва потребителя да избере кое приложение да използва. Когато дадено приложение приеме намерение, то може да извлича данни, свързани с намерението от оригиналното по произход приложение.

Въпреки това, злонамерените приложения могат да използват намерения за активиране на други приложения. В някои случаи за получаване на достъп до функционалност, за която нямат разрешение или за инжектиране на данни в други приложения. В различни случаи злонамереното приложение може да причини срив на програмата или извършване на неочаквани действия.

- **Инжектиране на команди**

Чрез методът за инжектиране на команди е възможно да се изследва и анализира приложение, което има способността да създава файлове в SD-карта с потребителско дефинирано име. Тук има фрагмент от файла AndroidManifest.xml, където се определя услугата:

```
<service android:name="FileCreatorService">
<intent-filter>
<action android:name="com.test.CreateFile" />
</intent-filter>
        </service>
```

Това пък е методът, по който се създава файлът:

```
protected void onHandleIntent(Intent intent) {
String input = intent.getStringExtra("fileName");
String s = Environment.getExternalStorageDirectory() + "/"
+ input;
try {
Runtime.getRuntime().exec(new String[]{"sh", "-c", "touch
" + s});
} catch (IOException e) {
}
    }
```

Приложението получава потребителско име от потребителския интерфейс и го изпраща на услугата чрез заявлението. Анализираната услуга е декларирала филтъра за намерение, така че приема само намерения, които имат действие зададено в com.test.CreateFile. Когато услугата получи валидно намерение, тя извлича името на файла от намерението и продължава да генерира директно файла, като извиква touch-команда чрез Runtime-обекта. След това злонамереното приложение може да генерира намерение по този начин:

```
Intent intent = new Intent();
intent.setAction("com.test.CreateFile");
intent.putExtra("fileName", "myFile.txt;cat
/data/data/com.test/secrets.xml >
/sdcard/secrets.xml");
          startService(intent);
```

Този код създава намерение и задава действието на com.test.CreateFile, което съответства на филтъра за намерение на анализираното приложение. Тогава добавя експлойт-низ. Уязвимото приложение ще свърже този низ с командата-touch, за да генерира конкретен файл, обаче експлойт-низът включва втора команда:

```
cat /data/data/com.test/secrets.xml > /sdcard/secrets.xml
```

Тази команда копира файла secrets.xml от директорията с личните данни на уязвимото приложение на SD-картата, където ще може да се чете изцяло. Злонамереното приложение може да включва всякакви shell-команди в полезния товар. Ако уязвимото приложение е инсталирано на системния дял или работи под root-потребител, можем да изпратим команди, които ще се изпълняват с повишени привилегии.

Най-доброто решение за защита срещу атаки базирани на намерение е да се комбинира следното противодействие, когато е възможно:

• Ако компонент на приложението трябва да декларира филтър за намерение и този компонент не е необходимо да се излага на външни приложения може да се зададе опцията - android: exported = false в файла AndroidManifest.xml:

```
<service android:name="FileCreatorService
android:exported=false">
<intent-filter>
<action android:name="com.test.CreateFile" />
</intent-filter>
</service>
```

Това ограничава компонента да отговаря само на намеренията, изпратени от оригиналното по произход приложение.

Заключение

Промените на пазара на мобилни телефони и услуги с въвеждането на мобилния Интернет и пакетният пренос на данни води до съществени промени в изискванията за мобилна сигурност. Уязвимости в съществуващите платформи са причина за множество и разнообразни злонамерени хакерски атаки, на които е необходимо ежедневно и съществено противодействие. Големите компании разработчици на мобилни платформи трябва да осигуряват своите операционни системи с различни способи, които да помагат и да не излагат крайните потребители на възможности за пробив в устройствата, системата и личната им информация. В доклада се акцентира на най-популярната и разпространена операционна система за мобилни смарт устройства – Андроид. Представят се и се анализират основни и революционни пробиви в платформата и механизми за тяхното противодействие.

Литература

[1] Ho Man, R. Choo., Mobile Security and Privacy, 2016
[2] Thiel D., iOS Application Security, 2016
[3] Petrov G., B. Balabanov, Dimensioning and evaluation of the radio frequency spectrum, /CEEC/, 2016
[4] Elenkov N., Android Security Internals, 2014
[5] National Institute of Standards and Technology, Managing the Security of Mobile Devices in the Enterprise, 2014
[6] Misra A., A. Dubey, Android Security: Attacks and Defenses, 2013
[7] Пасарелски Р., Универсални мобилни телекомуникационни системи, 2013
[8] Gunasekera S., Android Apps Security, 2012
[9] Furnell S., Mobile Security: A Pocket Guide, 2009
[10] Станчева А., В. Къдрев, Смущения между LTE мобилни потребителски устройства, 2017.