UNIVERSIDADE DA BEIRA INTERIOR
Engenharia

Assessing and Addressing the Security of Persistent Data in the Android Operating System

Francisco Dias Pereira Nunes Vigário

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática
(2º ciclo de estudos)

Orientador: Professor Doutor Pedro Ricardo Morais Inácio

Covilhã, Outubro de 2015

Dedication

"... to my parents, for all they have done to be able to finish this cycle of studies. To my advisor, since he was key to my success. To my girlfriend, Catarina Oliveira, for all the support and patience during this phase of my life."

Acknowledgments

First of all, I would like to thank my father, Orlando, and my mother, Maria de Lurdes, for all the effort they have made during my academic path and for all the support and encouragement they gave me. Without them, it would be impossible for me to achieve this stage of my life.

Next, I would like to acknowledge my supervisor, Professor Pedro Ricardo Morais Inácio, for the opportunity to integrate a dynamic research group and for all the support he gave me throughout the university years, especially during the course of my master thesis.

I am also grateful to my girlfriend, Catarina Oliveira, who was key to the success of my academic path. I appreciate all the support, the patience in the the most difficult moments, and all the motivation she provided me with to continue in the bad moments.

I appreciate all the help I have been given over the years from all my family, friends and class-mates.

I am thankful to EyeSee, Lda., specially to Eng. João Redol, for partially financing this work.

Last, but not least, I would like to express my gratitude to all the colleagues of the Multimedia Signal Processing - Covilhã (MSP-Cv) laboratory, based at Universidade da Beira Interior (UBI) and part of the Instituto de Telecomunicações (IT). I am particularly grateful to Miguel Neto, for all the support and knowledge he passed to me since the first day I joined the group.

Resumo

A adoção generalizada de dispositivos móveis, e a maneira como os utilizadores interagem com eles, levou ao desenvolvimento de novas formas de execução, distribuição e instalação de aplicações, abrindo também caminho para novos modelos de negócio. As aplicações móveis podem ser compradas como uma única peça de software, ou algumas das suas características, extensões ou conteúdos podem ser também objeto de compra. Em muitas aplicações para o Sistema Operativo Android, os itens pagos são, na maioria das vezes, obtidos através das chamadas *compras na aplicação* (da expressão *in-app purchases*), um recurso que permite ao utilizador fazer micro pagamentos no contexto da aplicação via Google *Play Store*. A receita de muitos programadores é obtida através da venda de pequenos recursos após lançar uma versão gratuita ou paga da sua aplicação. Por outro lado, as capacidades crescentes dos dispositivos móveis, juntamente com a sua natureza pessoal, transforma-os em agregadores de informação privada e alvos interessantes para os atacantes. Além disso, os dados de controlo armazenados nos dispositivos podem ser manipulados para alterar o fluxo de programas e aceder a funcionalidades bloqueadas ou a conteúdo sem pagar.

Esta dissertação está focada em problemas de armazenamento seguro de dados em dispositivos móveis, especialmente em sistemas Android. A principal contribuição é a quantificação da suscetibilidade à manipulação e à exposição de dados em aplicações Android através de um exaustivo estudo de muitas aplicações retiradas da loja oficial Android. Este estudo incluiu a construção de dois conjuntos de dados com um total de 1542 aplicações (849 jogos e 693 aplicações comuns) e a análise humana de cada um deles. O método consistia em: utilizar as aplicações num *smartphone*; transferi-las para um computador com um sistema operativo Linux de seguida; analisar e modificar os seus dados (quando possível); e por fim transferi-las de volta para o ambiente Android. Todo o procedimento aproveita a funcionalidade de *backup* disponibilizada pelo sistema operativo e usa apenas ferramentas disponíveis livremente, não sendo necessário permissões de administração no dispositivo móvel, comprovando a viabilidade da abordagem.

No caso do conjunto de jogos, verificou-se que pelo menos 1 em cada 6 era suscetível à manipulação de dados, o que significa que foi possível obter itens pagos sem qualquer pagamento. No caso do conjunto de aplicações comuns, 1 em cada 5 é suscetível ao mesmo problema de manipulação de dados ou foi possível obter informação armazenada sensível, como palavras-passe e números de identificação pessoal em texto limpo. As aplicações vulneráveis não incluem mecanismos para evitar que os dados sejam vistos ou modificados, o que constituiria a melhor forma de atenuar o problema. Com base no que foi aprendido, várias propostas generalistas para resolver o problema são discutidos no final da dissertação.

Palavras-chave

Android, Manipulação de Dados, Integridade, Segurança em Dispositivos Móveis, Sistema Operativo Móvel, Segurança, Armazenamento

Resumo alargado

Introdução

Este capitulo, escrito em língua Portuguesa, vem apresentar um resumo do corpo desta dissertação. Inicialmente começa-se por apontar o enquadramento deste trabalho, tal como a descrição do problema endereçado e os objectivo propostos. As contribuições principais resultantes deste programa de mestrado são posteriormente enumeradas. É descrito um pouco do trabalho relacionado nesta área, tal como é feita a apresentação dos conjuntos de dados levados a análise e o método utilizado durante esta análise. Os resultados obtidos da análise dos dois conjuntos de dados são apresentados na secção seguinte. Na penúltima secção são apresentado alguns esquemas para solucionar o problema encontrado neste trabalho. O capitulo termina com as principais conclusões retiradas deste trabalho, bem como algumas direções para trabalho futuro.

Enquadramento, Descrição do Problema e Objetivos

Os dispositivos móveis, principalmente os *smartphones* e *tablets* têm sofrido um acréscimo na sua utilização, parte do dia a dia de muitas pessoas, tendo vindo a substituir os computadores pessoais [MBK+12]. Com isto as maiores empresas do sector, tais como Google, Apple e Microsoft, dedicaram-se ao desenvolvimento de sistemas operativos para dispositivos móveis, tais como o Android [Goo15a] e o iOS [App15d] e à criação de plataformas e *kits* de programação e à criação de hardware dedicado, como o iPhone [App15e] ou o Nexus [Goo15c]. No que respeita ao software para dispositivos móveis, este segue um modelo diferente do tradicional, visto que muitos programadores preferem disponibilizar o software de forma gratuita inicialmente, sendo que assim conseguem ter um maior número de utilizadores, recorrendo a micro-pagamentos como fonte de rendimento. Estes micro-pagamentos são usados para desbloquear ou instalar funcionalidades adicionais, que inicialmente se encontrem indisponíveis ou bloqueadas. De forma a facilitar os utilizadores, este processo de instalação e pagamentos é efetuado através as *app stores*, sendo que os maiores distribuidores de sistemas operativos para dispositivos móveis têm as suas próprias lojas, como a Google Play [Goo15b], Apple Store [App15b] e Windows Phone Store [Mic15]. No caso da loja do sistema operativo Android, as peças de software que possibilitam a realização de micro-pagamentos dentro do software são catalogadas como possuindo *in-app purchases*.

O sistema operativo Android, desenvolvido pela Google, é o sistema que, em 2015, tinha uma maior cota de mercado, sendo que cerca de 75% dos dispositivos trazem este sistema operativo instalado [FBL+15]. Devido a este sucesso, muitos programadores focaram as suas atenções no desenvolvimento de aplicações para o mesmo. Reflexo disso é o aumento do número de aplicações disponíveis para download na Google Play. Por exemplo, entre junho e novembro de 2013, o total de aplicações disponíveis para download cresceu de 887,202 para 1,107,476, o que corresponde a um aumento de cerca de 25% [VGN14]. O aumento das aplicações disponiveis na Google Play tem continuado a aumentar, e à data de escrita da dissertação, já se encontram disponíveis mais de 1,633,000 aplicações. Este aumento também é explicado pelo facto de muitos programadores com poucos conhecimentos conseguirem construir as suas aplicações através de tutoriais disponíveis na Internet.

Este programa de mestrado foi focado no problema de armazenamento inseguro de dados em

dispositivos móveis com o sistema operativo Android (acima da versão 4.0.0). Em condições normais, o sistema operativo assegura que os dados internos de uma aplicação apenas são acedidos por esta, a não ser que outras aplicações tenham permissões explicitas para tal. No entanto, existem várias formas de contornar esta funcionalidade do sistema operativo. Uma destas formas é utilizando a ferramenta de *backup* disponível no Android. Esta ferramenta permite que os utilizadores realizem *backup* das aplicações instaladas no seu dispositivo para um computador pessoal e neste *backup* são incluidos os ficheiros internos da aplicação. A possibilidade de manipular os dados de uma aplicação sobre determinadas condições (explorada neste trabalho) é também conhecida na comunidade Android [XDA15]. Estas alterações podem ter influencia no fluxo normal do software, visto que os programadores utilizam estes ficheiros para armazenar dados auxiliares ao software e dados referentes ao utilizador.

O principal objetivo deste trabalho consiste em avaliar a suscetibilidade à manipulação e exposição dos dados em aplicações Android, utilizando para isso a ferramenta de *backup*. A ideia é avaliar o quanto é fácil tirar proveito da manipulação dos dados para obter algum ganho, analisar a quantidade de informações pessoais ou confidenciais que podem ser obtidas utilizando os meios disponíveis, quantificar em que medida estas vulnerabilidades afetam o universo de aplicações Android e, por fim, propor soluções para o problema identificado. De modo a obter todos os objetivos propostos, o trabalho foi dividido em cinco fases:

- A primeira fase consiste em realizar uma revisão da literatura especializada nesta área de conhecimento;

- A segunda fase consiste em realizar várias experiências não documentadas, de modo a construir o método a ser utilizado durante a análise das aplicações Android;

- Durante a terceira fase serão construidos dois conjunto de dados com aplicações e jogos Android. Será construido um *script* para filtrar apenas as aplicações que contenham, como característica, o facto de permitirem compras dentro da aplicação, sendo que esta seleção é feita de forma aleatória;

- A quarta fase consiste na análise das aplicações e jogos que se encontram nos conjuntos de dados construidos na fase anterior;

- Na quinta fase analisam-se as várias possibilidades para resolver os problemas de segurança encontrados durante este trabalho.

Principais Contribuições

A principal contribuição deste programa de mestrado consiste na realização de um estudo detalhado da suscetibilidade a manipulação e exposição de dados por parte de aplicações Android. Neste estudo foram utilizados os dois conjuntos de dados de aplicações construidos, sendo analisadas um total de 1542 aplicações, para conseguir obter um resultado conciso. Esta contribuição principal pode ser sub dividida em duas, como se segue:

1. A análise de 849 jogos para Android e a avaliação da sua susceptibilidade à manipulação de dados foi assunto num artigo cientifico com o título *Assessment of the Susceptibility to Data Manipulation of Android Games with In-app Purchases*, apresentado e aceite para

publicação nas atas da 30th *Internacional Conference on ICT Systems Security and Privacy Protection* (IFIP SEC 2015), realizada em Hamburgo, Alemanha, entre 26 e 28 de Maio de 2015 [VNF+15]. Esta conferência neste ano teve um rácio de aceitação de 20%.

2. A segunda consistiu na análise de 693 aplicações comuns disponíveis para Android e a avaliação da suscetibilidade para manipulação e exposição de dados, este foi assunto de um artigo cientifico com o título *On the Susceptibility to Data Manipulation and Information Exposure of Free Android Apps with In-app Purchases,* apresentado e aceite para publicação nas atas do 7º Simpósio de Informática (INFORUM 2015), realizado na Covilhã, Portugal, nos dias 7 e 8 de Setembro de 2015 [Fra15].

Aparte de estas principais contribuições, esta dissertação contem várias propostas de solução para o problema de manipulação de dados através da ferramenta de *backup*.

Estado da Arte

O aumento dos utilizadores de dispositivos móveis tem levado ao aumento da preocupação com a segurança destes dispositivos. Nos últimos anos, alguns investigadores da especialidade têm focado grande parte do seu trabalho nesta área, sendo que a vertente que mais tem sido investigada é a de malware para dispositivos móveis. Em 2011, Zurutuza [BZNT11] desenvolveu uma ferramenta para detetar malware baseado no comportamento da aplicação, à qual deu o nome de Crowdroid. Outra forma utilizada para a deteção de malware nestes dispositivos é através da analise do manifesto da aplicação, tal como descreveu Sato no seu trabalho [RS13]. Ainda relacionado com a deteção de malware, existem várias ferramentas online [Weg15, Vie15, Joe15, Fel15] disponíveis para qualquer programador ou utilizador comum submeter as aplicações desenvolvidas por si, ou aplicações que tenham intenção de instalar no seus dispositivo, para que esta seja alvo de análise e se perceba se tem malware ou não. A OWASP é uma fundação sem fins lucrativos criada em 1 de dezembro de 2011 que tem como principal objetivo a promoção da programação segura. Em 2010 sentiram necessidade de criar um projeto dedicado as aplicações móveis, sendo responsáveis pela criação de um top 10 de vulnerabilidades em aplicações móveis. A vulnerabilidade explorada neste trabalho encontra-se em segundo nesta escala, sendo que Christian Håland [Chr13] analisou, na sua dissertação de mestrado, várias aplicações, tais como o jogo 4Pics1Word e detetou que este jogo era vulnerável a este tipo de vulnerabilidade, visto que permitia aumentar o dinheiro virtual alterando os ficheiros auxiliares da aplicação.

Conjunto de Dados e Análise de Aplicações Android

O primeiro conjunto de dados construido para realizar este estudo é constituído apenas por jogos disponíveis para download na Google Play, tendo todos como característica comum o facto de permitirem compras dentro do jogo. Este conjunto de dados foi constituído durante os meses de Setembro e Novembro de 2014. Para conjunto de dados foram considerados 849 jogos, divididos por 15 diferentes categorias, sendo que a categoria com um maior número de aplicações foi a de *Ação*, com 141 jogos. Outra divisão que se pode aplicar ao conjunto diz respeito à popularidade de cada jogo, sendo que o intervalo de 1000000 a 5000000 é que mais aplicações contém (296 jogos neste intervalo).

O segundo conjunto de dados é constituído por aplicações comuns (excluindo jogos) disponíveis para download na Google Play de forma gratuita. Todas partilham da mesma característica

(i.e., de possibilitarem a compra dentro da aplicação). Este conjunto é composto por 693 aplicações dividido por 24 categorias. Antes da sua constituição, e contrariamente ao anterior, foi necessário estabelecer um limite de 30 aplicações por categoria para este conjunto, para não correr o risco de ter um número muito desfasado de aplicações por categoria. Apenas as categorias Desporto, Compras e Bibliotecas e Demonstrações não atingiram o limite de 30 aplicações. Tal como no conjunto anterior, neste também foi analisada a divisão no que diz respeito ao número de downloads, sendo o intervalo de 100 000 a 500 000 downloads aquele mais mais aplicações continua (com 197 aplicações). No que diz respeito ao método utilizado para realizar a análise das aplicações mencionadas nos conjuntos referidos anteriormente, esta incorpora 3 fases principais:

1. Começa-se por instalar o jogo ou a aplicação no dispositivo com sistema operativo Android. Analisam-se quais as funcionalidades que se encontram bloqueadas e identificam-se outros valores de interesse durante esta experimentação. Por exemplo, no caso do primeiro conjunto, era nesta fase que se jogava um pouco o jogo até atingir um determinado valor de moedas ou items virtuais, para obtenção de um valor de referência para procura aquando da análise no computador.

2. A aplicação é transferida para um computador com o sistema operativo Linux, usando o método descrito abaixo, onde os dados são inspecionados e por vezes alterados. Por exemplo, nos jogos, tenta-se encontrar o valor de moedas obtido no jogo e altera-se esse valor. A aplicação é de novo colocada num pacote com o fim de ser novamente enviada para o dispositivo.

3. Por último, a aplicação ou o jogo é testado no dispositivo de forma a perceber se as alterações efetuadas resultaram com sucesso.

O método que comporta a transferência e restauro da aplicação Android para um computador é constituído por um conjunto de 10 passos. Para os efetuar é necessário que o dispositivo esteja ligado a um computador através de um cabo Universal Serial Bus (USB). No computador que contenha ambiente Linux é apenas necessário instalar a ferramenta Android Debug Bridge (ADB), visto que todas as outras ferramentas necessárias para a realização deste método vêm por defeito aquando da instalação do sistema operativo.

Avaliação da Suscetibilidade à Manipulação de Dados em Aplicações Android com Compras Dentro da Aplicação

No capitulo 4 apresentam-se os resultados obtidos após a analise dos conjuntos referidos anteriormente. Foram analisados 849 jogos, sendo que 148 estavam suscetíveis à manipulação dos dados, o que corresponde a 17,43%, utilizando o método que foi referido anteriormente. A categoria mais afetada por este método é a categoria de *Cultura Geral*, visto que 50% dos jogos desta categoria são suscetíveis à manipulação de dados. No entanto, esta categoria é das que contem menos jogos a ser analisados: apenas 6. Este resultado pode ser explicado pelo facto destes jogos funcionarem com ligação à Internet. As compras na aplicação servem para ter acesso a informação ou ajudas para conseguir responder as questões do jogo. Nos jogos vulneráveis, esta informação é guardada nos ficheiros internos sem mecanismos de integridade. A categoria de *Corridas* e *Arcada* continham 28% e 26% de jogos suscetíveis à manipulação de dados, respetivamente, ocupando o segundo e terceiro lugar, respetivamente. Neste tipo de jogos, as compras

dentro da aplicação estão relacionadas com o dinheiro virtual utilizado no jogo. Por exemplo, no caso dos jogos de corridas, este dinheiro é utilizado para comprar novos carros. Nos jogos vulneráveis, o valor correspondente ao dinheiro virtual que o utilizador tem é guardado em texto limpo em ficheiros do formato eXtensible Markup Language (XML), ficheiros de texto ou em base de dados SQLite. Esta informação é guardada sem recurso a cifras, mecanismos de integridade ou códigos de autenticação. De notar que as categorias *Role-playing Game (RPG)*, *Família* e *Música* não têm qualquer jogo com vulnerabilidade, diretamente relacionado com o facto de, para estes jogos, os pagamentos servirem sobretudo para retirar publicidade ou descarregar conteúdo da Internet, cujo pagamento pode ser, portanto, sempre validado.

No que respeita à análise dos jogos em relação à sua popularidade, temos os intervalos de *downloads* de 5000−10000 e de 50000−100000, com 50% e 30% de jogos vulneráveis. A principal conclusão retirada desta análise é que o número de jogos vulneráveis não está dependente da sua popularidade, visto que mesmo no intervalo correspondente à maior popularidade (100000000−500000000) havia dois suscetíveis à manipulação dos dados, o que corresponde a aproximadamente a 15% de jogos neste intervalo.

Outra análise efetuada aquando deste estudo foi perceber que tipo de ficheiros era utilizado para guardar os dados dos jogos vulneráveis. Concluiu-se que 76% dos jogos vulneráveis utiliza ficheiros XML para armazenar os dados suscetíveis à manipulação, apenas 9% dos jogos utilizam base de dados SQLite para armazenar estes dados, os restantes 14% utilizam outro tipos de ficheiros tais como JavaScript Object Notation (JSON) ou ficheiros de texto. Interessante perceber que apenas 21 jogos não permitem realizar o seu *backup*, tendo a propriedade `android:allowBackup` a `false` no ficheiro `AndroidManifest.xml`.

Na segunda fase do estudo foram analisadas 693 aplicações comuns e disponíveis de forma gratuita na Google Play. No entanto apenas 377 dessas aplicações foram consideradas nos resultados finais, correspondendo a uma redução de cerca de 54% ao conjunto de dados inicial. Neste caso, foi inicialmente aplicado um filtro para remover as aplicações que apenas permitiam remover a publicidade, as que não permitiam fazer o *backup* da aplicação, as que não eram compatíveis com o dispositivo utilizado nesta análise (BQ Aquaris E5 FHD) ou as que não tinham qualquer funcionalidade para comprar, estando assim mal catalogadas. As categorias menos afetadas pela aplicação do filtro foram as de *Cuidados Médicos* e *Educação*, conservando 80% das aplicações existentes no conjunto de dados inicial. No lado oposto, as categorias mais afetadas por este filtro foram as de *Banda Desenhada* e *Desporto*, onde apenas foram consideradas 37% das aplicações nos resultados finais. No que se refere aos intervalos de *download*, o mais afetado foi o correspondente a 100-500 *downloads*, com apenas 9% das aplicações a serem consideradas nos resultados finais. O intervalo que menos sofreu com o corte foi o de 50000000 − 100000000, visto que 67% das aplicações foram consideradas para a análise final. Para a análise das aplicações resultantes, foram consideradas vulneráveis todas aquelas que estavam suscetíveis à manipulação dos dados ou que mostravam informações sensíveis do utilizador, tais como nome ou senha de utilizador.

A categoria de aplicações mais suscetível ao método utilizado foi a categoria de *Multimédia e Vídeo*, com 46% vulneráveis. De seguida, aparece a categoria de *Finanças*, com 42% de aplicações vulneráveis. No caso da categoria *Multimédia e Vídeo*, este resultado pode ser explicado pelo facto muito comum dos programadores colocarem funcionalidades premium de edição de

vídeo bloqueadas, sendo necessário pagar para as desbloquear. O controlo de acesso a estas funcionalidades é feito via variáveis de estado cujo valor é guardado nos ficheiros auxiliares à aplicação. No caso da categoria das *Finanças*, a taxa de vulnerabilidade é elevada porque também há mais a permitir autenticação remota, e, infelizmente, a guardar dados sensíveis em texto limpo no armazenamento local. Por exemplo, foi possível encontrar Personal Identification Numbers (PINs), e até fazer o seu reajuste por manipulação de dados em algumas aplicações. Nas categorias de *Empresas*, *Bibliotecas e Demonstrações*, *Fotografia e Produtividade* não foi encontrada qualquer aplicação suscetível ao método aplicado.

No que diz respeito à análise considerando o número de *downloads*, foi o intervalo de 50000000 a 100000000 que reuniu o maior numero de aplicações vulneráveis, com cerca de 33%. De salientar que, das aplicações vulneráveis, 70% utilizam ficheiros XML para armazenar os dados, enquanto que os restantes 30% utilizam base de dados SQLite. Das aplicações analisadas, 87 permitiam ao utilizador fazer autenticação (*login*) na aplicação e, destas, 23 guardavam as credenciais de acesso em texto limpo, o que corresponde a cerca de 26%.

Mecanismos para Prevenção de Manipulação de Dados em Sistemas Android

A funcionalidade de *debug* que foi integrada no sistema operativo Android veio de certa forma facilitar a vida dos programadores, visto que com esta ferramenta conseguem fazer *debug* das aplicações que estão a desenvolver diretamente no dispositivo. Esta permite a instalação de aplicações, entre outras funcionalidades, utilizando a linha de comandos. Esta permite também realizar o *backup* de aplicações que se encontrem instaladas no dispositivo, e para isso basta que este esteja conectado via USB com um computador e utilizar o ADB.

Na fase final deste mestrado tentou-se encontrar uma resolução para o problema estudado. Para isso desenharam-se 4 soluções, descritas com mais detalhe no capitulo 5. A primeira solução é baseada na alteração da ferramenta de *backup* de modo a incorporar mecanismos de integridade. Sendo assim, ao fazer o *backup* de uma aplicação para um computador, era calculado um Message Authentication Code (MAC) ou uma assinatura digital e, aquando do restauro da mesma aplicação, era de novo calculado o valor e apenas permitido o restauro da aplicação em caso válido. A segunda abordagem é semelhante à primeira, sendo apenas adicionado o armazenamento das chaves utilizadas para o calculo do MAC ou da assinatura digital num servidor remoto, sendo estas transferidas para o servidor através de um canal criptograficamente seguro. Esta solução iria permitir o *backup* de uma aplicação num determinado dispositivo e o restauro noutro dispositivo diferente, pressupondo uma ligação à Internet. A terceira solução utiliza a arquitectura dos processadores Acorn Risc Machine (ARM) proposta em 2009 chamada ARM TrustZone. Esta arquitectura simula dois *mundos*, em que um é designado por *mundo seguro*. Seria, de resto, neste mundo seguro que seria calculado o MAC ou a assinatura digital, garantindo assim que os respetivos valores eram calculados de forma segura. A quarta opção é semelhante à anterior, diferindo apenas no facto do resultado das operações efetuadas no mundo seguro serem enviadas para um servidor remoto através de um canal seguro.

Outra possível solução para o problema endereçado neste programa de mestrado é a integração de primitivas criptográficas nas próprias aplicações, visto que só assim se consegue resolver o problema em dispositivos em modo *root*. No entanto, este cuidado já tem de partir do lado do programador que, ao realizar a aplicação, tem de se preocupar com a segurança da mesma e dos seus utilizadores. Caso o programador utilize bases de dados para guardar os dados da

aplicação, este já tem um conjunto de extensões que lhe facilitam na proteção dos dados, tais como SQLite Encryption, wxSQLite [Zet15], entre outros.

Conclusão e Trabalho Futuro

Os objetivos deste trabalho foram concluídos com sucesso, particularmente através da constituição e análise dedicada de um vasto conjunto de jogos e aplicações móveis. Para realizar a análise, foram criados dois conjuntos de dados, sendo estes constituídos por aplicações ou jogos retirados da Google Play de forma gratuita. Todos tinham uma característica comum: o facto de permitirem compras dentro da aplicação ou jogo. No conjunto dos dois conjuntos de dados foram consideradas 1542 aplicações ou jogos. No conjunto constituído por jogos, a categoria mais abrangente foi a de *Acção* constituída por 141 jogos. No que diz respeito ao número de *downloads*, o intervalo com o maior número de jogos foi 1000000 — 5000000, com 296 jogos. No conjunto constituído por aplicações comuns, foi estabelecido um teto máximo de aplicações por categoria de forma a não ter uma discrepância no número de aplicações entre categorias. Este limite foi de 30 aplicações por categoria. Foram consideradas 24 categorias presentes na Google Play para esta parte do estudo, sendo que o intervalo de *downloads* com maior número de aplicações foi o de 100000 a 500000.

No que diz respeito à analise dos jogos, o resultado mostrou que aproximadamente 17,5% dos jogos se encontravam suscetíveis ao método utilizado, o que corresponde a 148 jogos vulneráveis, num universo de 849 considerados. De notar também que os jogos vulneráveis utilizam maioritariamente ficheiros XML para armazenar os dados utilizados pelo jogo. Na análise ao segundo conjunto de dados foi concluído que cerca de 20% das 377 aplicações consideradas no estudo final eram suscetíveis ao método utilizado. Neste conjunto, 87 das aplicações permitiam que o utilizador fizesse *login* na aplicação, tendo-se constatado que um subconjunto de 23 aplicações guardavam as credenciais do utilizador em texto limpo, nos ficheiros da aplicação. Por fim, foram apresentadas algumas alterações à ferramenta nativa de *backup* como forma de resolução parcial dos problemas encontrados.

No que diz respeito às linhas de trabalho futuro, é dito que seria interessante desenhar um método semelhante ao apresentado neste trabalho para diferentes sistemas operativos móveis, tais como iOS ou Windows Mobile, e perceber de que forma se comportam perante a mesma abordagem. Outra linha de trabalho seria aumentar o numero de aplicações analisadas, especialmente para aplicações em compras na aplicação, já que seria interessante perceber o seu comportamento também e se a qualidade em termos de segurança é motivada pelo retorno monetário no programador. Uma maior automatização do método seria também interessante, visto que assim se podia realizar a análise a um maior número de aplicações, e até mesmo criar uma ferramenta para os programadores testarem as suas aplicações antes de as submeterem para a Google Play. Por último, seria interessante implementar uma das soluções apresentadas no capitulo 5 de modo a adicionar uma proteção extra para as aplicações móveis em Android.

Abstract

The widespread adoption of mobile devices and the way users interact with them led to the development of new ways of implementing, distributing and installing applications, paving the way for new business models also. Mobile applications can be bought as a single piece of software, or some of its features, extensions or contents may be the subject of purchases. In many mobile applications for the Android Operating System (OS), paid items are most of the times delivered via *in-app purchases*, a feature that enables a user to make micro-payments within the application context via Google Play store. The revenue of many developers is obtained by selling small features after releasing a free or paid version of their application. On the other hand, the increasing capabilities of mobile devices, along with their personal nature, turns them into aggregators of private information and interesting targets for attackers. Additionally, control data stored in the devices may be manipulated to change the flow of programs and potentially access blocked features or content without paying.

This dissertation is focused on secure data storage problems in mobile devices, particularly on Android systems. Its main contribution is the quantification of the susceptibility to data manipulation and exposure of Android applications through an exhaustive study of many applications downloaded from the official Android store. This study included the construction of two data sets with a total number of 1542 applications (849 games and 693 common applications) and the human analyses of each one of them: the applications were first used in a smartphone, then transfered to a computer with a Linux OS, their data was analyzed and modified (when possible), the transfered back to the Android environment. The entire procedure takes advantage of the backup utility provided by the OS and using only freely and readily available tools, and does not require administrative permissions on the mobile device. proving the feasibility of the approach.

In the case of the games data set, it was found that at least 1 in each 6 was susceptible to data manipulation, meaning that it was possible to obtain paid items without any payment. In the case of the common applications data set, 1 in each 5 was either susceptible to the same data manipulation problem or were storing sensitive data like passwords and Personal Identification Numbers (PINs) in plaintext. Vulnerable applications do not include mechanisms to prevent the data from being eavesdropped or modified, which would be the preferred way of attenuating the problem. Based on lessons learned, several proposals to solve the problem from a general perspective are discussed towards the end of the dissertation.

Keywords

Android, Data Manipulation, Integrity, Mobile Security, Mobile Operating System, Security, Storage

Contents

List of Figures

List of Tables

Acronyms

ADB Android Debug Bridge

AES Advanced Encryption Standard

APK Android Application Package

API Application Programming Interface

ARM Acorn Risc Machine

ASCII American Standard Code for Information Interchange

CCM Counter with CBC-MAC

CLI Command-line Interface

DoS Denial-of-Service

GUID Globally Unique Identifier

HTTP Hypertext Transfer Protocol

ID Identifiers

IMEI International Mobile Station Equipment Identity

IT Instituto de Telecomunicações

JCA Java Cryptography Architecture

JSON JavaScript Object Notation

MAC Message Authentication Code

MITM Man-In-The-Middle

MMS Multimedia Messaging Service

Ms.C. Master of Science

MSP-Cv Multimedia Signal Processing - Covilhã

NS Non-Secure

OFB Output Feedback

OS Operating System

OWASP Open Web Application Security Project

PC Personal Computer

PIN Personal Identification Number

RC4 Rivest Cipher 4

REE Rich Execution Environment

RPG Role-playing Game

RSA Rivest Shami Adleman

SDK Software Development Kit

SEE SQLite Encryption Extension

SHA256 Secure Hash Algorithm 256

SMS Short Message Service

SOAP Simple Object Access Protocol

SoC System-on-Chip

SSL Secure Sockets Layer

TCB Trusted Computing Base

TEE Trusted Execution Environment

TLS Transport Layer Security

UBI Universidade da Beira Interior

UID Unique Identification Number

USB Universal Serial Bus

XML eXtensible Markup Language

XSS Cross-site Scripting

Chapter 1

Introduction

This dissertation describes the work performed to obtain the master's degree in Computer Science and Engineering from the University of Beira Interior. This chapter describes the motivation and scope of the dissertation, contains the problem statement and defines the objectives, enumerates the main contributions and describes adopted approach for solving the problem, and presents the overall organization of the document.

1.1 Motivation and Scope

In the last decade, mobile devices, specially smartphones and tablets, became part of the everyday life of many people, many times replacing Personal Computers (PCs) [MBK+12]. Sensing the potential of such market, and effectively contributing to its success afterwards, Google, Apple and Microsoft developed mobile Operating Systems (OSs) (such as Android [Goo15a] and iOS [App15d]) and programming platforms and kits, assembled dedicated hardware (e.g., iPhone [App15e] or Nexus [Goo15c]) and worked on different business models and strategies to obtain revenue and deliver contents into such platforms. In terms of software or functionality delivery, these models are fundamentally different from the classical ones. The potential to get to a larger number of users led to the adoption of the micro-transactions in this area, meaning that applications became cheaper (they are simpler than desktop or server applications also) and that it is possible to buy only a limited number of functionalities or content at a time, more easily. The process of installing and paying was simplified via the so-called *app stores*, which congregate both functionalities (and others) in user-friendly manner. The major vendors and OS developers maintain some of those stores, namely Google Play [Goo15b], Apple Store [App15b] and Windows Phone Store [Mic15]. It is also possible to buy items within the mobile applications, a functionality called *in-app purchase* in the case of Android applications, with Software Development Kits (SDKs), such as the one of Android, easing the process of integrating the functionality in the development phase.

In 2015, Android was the OS installed in the majority of smartphones and tablets worldwide, and Google Play was holding 75% of the total market [FBL+15]. There are several reasons for this success. One of the most important is the fact that Android is open source and based on Linux, which enables mobile devices manufacturers to more easily reach the market after assembling a given device, since they do not need to develop the overlying software, provided that the hardware is compatible with Linux and specific drivers.

The success of the *app stores*, along with the flexible business models, have been motivating developers. To obtain revenue, they can follow two main businesses strategies: either selling the application at an initial price; or deliver it free of charge, and ask for money in exchange of additional content of functionalities. Paid applications may also offer the option for users to buy contents or new features. Notice that a third option for obtaining revenue, based on placing

advertisements in free or paid applications, is also available. Nonetheless, in many cases, it is possible to remove them via an *in-app purchase*, falling into the previous categories.

Contrary to what happens in the Apple Store, where applications undergo a review process before publication [App15c], Android applications are available for users do download soon after being submitted to Google Play, without much reanalysis [And15d]. The developer license costs $25 [And15e]. This also contributes to the success of the store, though it is on the origin of security issues, dictating a fast growing pace. Between June and November of 2013, the total number of available applications increased from 887,202 to 1,107,476, representing an increase of more than 25% in the number of applications. In the same period, the number of downloads increased by 37% , which also reflects the increase of users of this system [VGN14]. Currently, there are more than 1,633,000 applications available for the Android OS [App15a]. Unfortunately, the ease of developing for mobile platforms, namely for Android, is also having consequences in terms of the quality of the applications. Many developers are able to build and deploy an applications with little knowledge after following tutorials on the Internet, which may lead to lack of security for the data the applications handle.

Given their personal nature, mobile devices are hubs for private and confidential data. Also because of that, mobile OSs integrate security features that isolate their several components, namely applications, storage and sensors. Nonetheless, their widespread adoption combined with the motivation of potentially obtaining private data or financial gains, makes of these devices interesting targets for attackers or even typical users. It is known that many users try to circumvent payment mechanisms in computer systems, searching the web for software cracks or pirated copies of the applications. It is thus increasingly important to study the security of mobile devices.

This dissertation reflects a work performed in the scope of a master's program in the area of Computer Science and Engineering. It falls in the intersection of the (mobile) OSs and computer security fields, focusing on the security of the data stored in mobile devices within the context provided above. Under the 2012 version of the Association for Computing Machinery (ACM) Computing Classification System, a de facto standard for computer science, the scope of the master's program, reflected in this dissertation, falls within the categories named:

- **Security and privacy∼Software security engineering**
- *Security and privacy∼Mobile and wireless security*
- Software and application security
- Database and storage security

1.2 Problems Statement and Objectives

This master's program addresses the problem of insecure data storage in mobile devices with the Android OS (up to version 4.0). Generally speaking, it deals with the problem of having developers overlooking storage related security aspects, only relying on the OS mechanisms to assure them. While, under normal conditions, the Android OS assures that no application can access or modify data from other applications, unless explicitly permitted, there are several

ways of circumventing such controls. Some of them imply gaining privileged permissions over the OS, and potentially voiding the the warranty of the device, others derive from legitimate functionalities. One example of such functionalities is the one provided by the backup utility, which comes native with Android. It provide users with means for backing up their applications (along with their internal data) to a personal computer. The possibility to manipulate the data under such conditions is known in the Android community [XDA15], and has probably been used to construct cracks for games and applications [Whi15]. Since internal data is used to store private information from users and often to control the flow of the software (e.g., save files), this constitutes an interesting problem.

The main objective of this work is therefore to assess the susceptibility to data manipulation and exposure of Android applications, namely by exploiting the backup utility. This objective can be further decomposed into the following:

- Assessing how easy it is to take advantage of data manipulation to obtain some gain, e.g., getting access to paid features or content without purchasing them;
- Analyze how much private or confidential information can be obtained using the aforementioned means;
- Quantifying to which extent the vulnerabilities affect the universe of Android applications;
- Propose solutions for the identified problem.

1.3 Adopted Approach for Addressing the Problem

In order to achieve the aforementioned objectives, the research work of this master's program was divided into the following five phases:

1. The first phase consisted reviewing part of the specialized literature in this area of knowledge, in order to get acquainted with the problem under analysis and with the concepts and tools supporting the subsequent phases of the project;
2. The second phase included several non-documented experiments to get used to the method used to perform the analysis of the Android applications. This method is based on the native Android backup utility. This part of the work comprised studying the several details specific to the utility, as well as getting to know the other tools used in the process. Moreover, it included setting up an initial version of the environment used to conduct the experiments of phase four. The method used to analyze each application is explained with more detail in Chapter 3.
3. The third phase was devoted to the construction of two large data sets with Android applications. This part of the work was composed by the tasks of identifying the target applications, analyzing the download procedure from Google Play, building scripts for downloading (to local storage) and gathering meta-data for the applications, and reviewing the data sets and meta-data for mistakes or consistency problems. The usage of large data sets was useful to conservatively quantify the severity of the insecure storage problems analyzed in the scope of this work;

4. The fourth phase consisted in the application of the method, adapted during the second phase, to the two data sets built in the third phase. The results obtained during the procedure were also analyzed and documented at this stage;

5. The fifth phase comprised the analysis of several possible general solutions to the security problems addressed in the work, elaborating on lessons learned during previous phases and emphasizing some of the inherent limitations.

1.4 Main Contributions

The main contribution of this master's program was the detailed study of the susceptibility of Android applications to data manipulation and exposure. The usage of two large data sets of applications (a total of 1542 applications) permitted obtaining a concise, yet conservative, idea of the extent of the addressed problems in the Android universe. To the best of the knowledge of the author, there is no other work following the same approach and studying such a large number of applications at the time of writing this dissertation. The main contribution can be divided into two smaller ones, as follows:

1. The thorough analysis of 849 Android games and the assessment of their susceptibility to data manipulation, which was the subject of the paper entitled *Assessment of the Susceptibility to Data Manipulation of Android Games with In-app Purchases*, presented and accepted for publication in the proceedings of the *30th International Conference on ICT Systems Security and Privacy Protection* (IFIP SEC 2015), held in Hamburg, Germany, between the 26th and the 28th of May, 2015 [VNF+15]. The acceptance ration of the conference this year was of 20%;

2. The thorough analysis of 693 Android applications and the assessment of their susceptibility to data exposure and manipulation, which was the subject of the paper entitled *On the Susceptibility to Data Manipulation and Information Exposure of Free Android Apps with In-app Purchases*, presented and accepted for publication in the proceedings of the *7th Simpósio de Informática* (INForum 2015), held in Covilhã, Portugal, between the 7th and the 8th of September, 2015 [Fra15].

Apart from the aforementioned contributions, this dissertation contains several proposals for solving the problem of data manipulation via backup utility. These proposals are discussed from the feasibility and security points of view in Chapter 5.

1.5 Dissertation Overview

The overall organization of the dissertation reflects the several phases of this work. The main body of this document is divided in 6 chapters, summarily described as follows:

- Chapter 1 — **Introduction** — on which this section is included, approaches the main topics of this Master of Science (Ms.C.) dissertation by presenting the motivation and scope of this work. The adopted approach for solving the problem is outlined also in this chapter, prior to the enumeration of its main contributions and some outcomes in terms of publications.

- Chapter 2 — **Related Work and Background** — discusses works that are close to the one de-

scribed herein or that were important during the course of this work. It also provides some background on the Android OS and on concepts important to the understanding of other parts of the document. It discusses malware for Android and the Open Web Application Security Project (OWASP) top 10 mobile threats.

- Chapter 3 — **Data Sets and Analysis of Android Applications** — describes the datasets used for studying the susceptibility of Android apps to data manipulation or to data exposure. Apart from the two main datasets (one comprised of Android games and the other consisting of normal applications), which are characterized regarding popularity and category, the method used to analyze data exposure or manipulate the app behavior is also described in there.

- Chapter 4 — **Assessment of the Susceptibility to Data Manipulation of Android Applications with In-app Purchases** — discusses the most interesting findings obtained after applying the method to the two datasets considered in the scope of this work. Specifically, it quantifies how many apps were susceptible to data manipulation and elaborates on the data exposure problem for the apps under analysis.

- Chapter 5 — **Mechanisms for Preventing Data Manipulation in Android Systems** — proposes solutions to the problems identified in the previous sections, enumerating their advantages and disadvantages, sometimes considering the possibility of using hardware based mechanisms to cope with the security requirements.

- Chapter 6 — **Conclusions and Future Work** — finalizes this dissertation by presenting the major conclusions of this work and identifying some lines for future work.

Chapter 2

Related Work and Background

2.1 Introduction

A the time of writing of this dissertation, the interest in the security of mobile applications was trending, mostly due to the fact that mobile devices and applications are being increasingly used to to store and process private data. One of the main focuses on this area has been *malware*, since attackers have also begun putting more effort in the development of malicious software for such devices. Nonetheless, research in other topics has also been increasing, notably after the appearance of the OWASP mobile security project. This chapter provides an overview on related works, contextualizing the main problem addressed in this dissertation. Section 2.2.1 starts with a brief discussion on the Android OS security. The malware topic is the subject of section 2.2.2, while Section 2.2.3 focuses on the security risks identified by the OWASP project for Android. The discussion then converges to a more detailed discussion of one of the risks in particular, namely the one of *Insecure Data Storage*. Finally, section 2.3 describes one of the tools that was central to this work.

2.2 Related Work

This section starts with a discussion on research work for vulnerabilities found on the Android OS kernel, to then focus on the Malware threat. Later on, the discussion will be dedicated to the OWASP risks, namely on the one that contextualizes better this work.

2.2.1 Android OS Security

The Android OS is based on Linux and is especially prepared to run on devices with screentouch. As with the Linux kernel, Android also contains vulnerabilities that have not been detected. Hei et al., [HDL13] found two vulnerabilities in OS Kernel in 2013. These researchers examined the source code of the `GT-P7500_OpenSource.zip` and `GT-P7510_OpenSource.zip` packages. They found two vulnerabilities in the `nvhost_ioctl_ctrl_module_regrdwr` functions implemented in the `dev.c` file, namely in `nvhost_write_module_regs` and `nvhost_read_module_regs`. Privilege escalation type of vulnerability is the first on mentioned in the document. Authors found that the functions resort to offsets to find the Unique Identification Number (UID) in the system, but that these offsets were not dully verified to be within the expected bounds. By manipulating system variables, it would be possible to set the UID to 0, which is root, after which full control of the device was possible. They reported a successful exploitation of the vulnerability in a Samsung Galaxy 10.1. A Denial-of-Service (DoS) was the second type of vulnerability pointed out by the authors which, according to them, is very easy to exploit using, e.g., `nvfuzz.c`. This vulnerability would enable an attacker to crash the device due to buffer overflow problems in another function handling system variables. They reported the vulnerabilities and the solution was incorporated in later versions (this was discovered in Honeycomb) of the OS.

The aforementioned work shows that there are vulnerabilities in all layers of the mobile OSs, with some of them introduced after adapting existing kernels (e.g., the previous vulnerabilities affect the Android kernel only). Many of these vulnerabilities, namely buffer overflow related, are exploited in the wild to root or jailbreak these OSs.

2.2.2 Android OS Malware

Within the mobile security area, malware is one of the topics receiving more attention since 2011. Unfortunately, the popularity of mobile devices, their widespread adoption and the relatively ease way of implementing and deploying applications has been driving malware developers into this environment also. Research has been focused on developing solutions for detecting malicious software.

In 2011, Zurutuza et al. [BZNT11] developed a new system for detecting malware for Android. Their approach, which they called Crowdroid, is based on behavior. The system consisted of an application that needs to be installed on the smartphone of users, available to download from Google Play at the time. This application would look for and pick up system calls made by the other applications that were being used by the user, and obtain their behavior. The *sensor* application would then send the collected data to a remote server to be further processed. The entire process is divided into three main phases: (i) data acquisition; (ii) data manipulation; and (iii) malware analysis and detection. The first phase was responsible for obtaining and sending the data to the remote server, and takes place on the device. The second phase consists on the pre-processing and uniformization of the data for processing, and it is performed on the remote server already. The final phase is performed on the server also, and concerns the malware detection after processing the system calls.

Sato et al. [RS13] proposed another approach for the detection of malware in Android resorting to the analysis of the Manifest file [And15c] present in all applications. This file contains information regarding the application, such as permissions, the libraries, target platform, the definition of the main activity and other custom authorizations, along with key information for the correct functioning of the application. The approach consisted in several methods and it begins with an analysis of the `AndroidManifest.xml`. After extraction of the information of interest, the produced data is compared with a previously prepared list of keywords with the objective of calculating a malignancy score. The following formula is used for this purpose:

$$P = \frac{M-B}{E},$$

where P is the value that will be used to classify the application (malignancy score), M is the number of malicious strings, B represents the number of benign strings, and finally E denotes the total number of items taken into consideration. For example, authors of this work state that permissions such as `READ_PHONE_STATE` and `INTERNET` are amongst the most popular in malicious software. As such, they may have a contribute to detecting such software. Other authors [WMW+12] have been adapting machine learning algorithms such as K-Nearest Neighbours, Bayesian networks among others, to the classification of mobile applications also.

Also regarding malware analysis for Android, it is possible to mention several freely available online platforms for reviewing applications before publication [Weg15, Vie15, Joe15, Fel15]. Most of these platforms accept the `.apk` file and output a detailed report shortly after. They

resort to several techniques, namely static analysis of the code. These platforms can be used by developers to realize which kind of vulnerabilities the applications possess or may be affected by. Because the input file is an `.apk`, end users can also use these platforms before installing applications.

2.2.3 OWASP Top 10 Mobile

OWASP is a non-profit foundation that was brought online on December 1, 2001. It is an open community that aims to create methodologies, documentation, articles, tools, among other things, always with the objective of the practice of safe programming, so as to contribute to an ecosystem of reliable applications. It is mostly known for the efforts in producing lists of risks for some types of applications, notably Web applications. Nonetheless, with the increased usage of mobile devices led to the creation of a sub-project specially focused on this type of applications in 2010. Figure 2.1 shows the list of the top 10 risks encountered in mobile applications according to OWASP.

Figure 2.1: The OWASP Top 10 Mobile Risks.(adapted from [OWA15])

The list of the top 10 risks for mobile applications according to OWASP [OWA15] can be briefly described as follows (M2 is discussed in the subsequent section):

- **M1 - Weak Server Side Controls** – the major risk in the list concerns the possibility of having weak access controls and Application Programming Interfaces (APIs) in the server side of a given mobile application. Many applications are nowadays designed with two components, one of which is working in a remote server. Alternatively, the remote server is only used to store data. Problems in these servers may give access to private or confidential data, or provide means to disseminate malware. Server side problems may also be exploited to rapidly disseminating code, e.g., via Cross-site Scripting (XSS) in applications running a `webkit`.

- **M3 - Insufficient Transport Layer Protection** – If a mobile application uses a server/client architecture, it is important that communications are properly secured. The best way to do so is by applying encryption and integrity mechanisms to the channel using high-quality cryptographic mechanisms and established protocols. It prevents eavesdropping the channel for sensitive information such as passwords and usernames. Nonetheless, according to OWASP, mobile applications do not frequently protect network traffic. They sometimes use the Secure Sockets Layer (SSL)/Transport Layer Security (TLS), but only during authen-

tication, and not in other phases of communication. Sometimes also, developers integrate TLS in an incorrect manner, due to limited knowledge on the area.

In his master's thesis [Jam14], James King analyzed various security topics mentioned by OWASP. He studied the FourGoats application within the context of Insufficient Transport Layer Protection and was able to successfully intercept packets flowing between the application and the server using the Burp Proxy v1.5. In this application, logging into the service was sending the authentication data via a POST message, unencrypted and non-authenticated, clearly showing the severity of the problem. The message could be eavesdropped and modified without detection.

In the dissertation entitled *An Application Security Assessment of Popular Free Android Applications*, Christian Haland [Chr13] stated he found two applications with this type of vulnerability. The QuizBattle game, which required one to register in order to get into the game, was using only Hypertext Transfer Protocol (HTTP) to convey such data to the server. The other application was used to remove or share names of photos on Instagram, which actually used TLS 1.0 to communicate with the API available in `http://instagram.com/api/v1/` for authentication purposes, but the remaining part of the communication was performed using HTTP only. The application was thus susceptible to session highjaking.

In [FHM+12], Fahl et al. described a study to understand how the SSL/TLS was used to protect data over the network for communications of Android applications. The poor integration of this protocol or lack thereof leads to attack situations, namely to Man-In-The-Middle (MITM). For this study, these authors developed the `MalloDroid`, a small application that can used to find broken SSL certificate validation procedures in Android applications. In their work, along with the developed tool, a total of 13,500 applications available for free download in Google Play were also analyzed. They concluded that about 8% of those applications were vulnerable to MITM attacks.

- **M4 - Unintended Data Leakage** — This type of vulnerability occurs when the programmer keeps sensitive application data improperly on the mobile device, in locations that are easy to access for other applications. This type of vulnerability may lead to sensitive information extraction. This type of problem may motivate the development of specialized malicious applications for ex-filtration of data from mobile devices.

- **M5 - Poor Authorization and Authentication** — Some applications use device related information as a means of authentication sending it to remote servers. An example of such informations is the International Mobile Station Equipment Identity (IMEI). For this type of vulnerability, James King [Jam14] analyzed the *Herd Financial* application that uses the device hardware Identifiers (ID) to authenticate at the user login. Curiously, the device ID was transmitted to the server in clear text. In the case an attacker can get his hands on a valid device ID, and after realizing that it may be used to authenticate the application on the server, he can use it to authenticate maliciously on the server, thus gaining access to legitimate user data.

- **M6 - Broken Cryptography** — The risk at position six of the top 10 is related with the fact that, when used, cryptography mechanisms are either weak or badly integrated. The

design of the security part of the application is bad in many situations. For example, encryption keys are stored hardcoded in the code, even when high grade ciphers are being used, leaving the data exposed to attackers that obtain the code and reverse engineer it.

- **M7 - Client Side Injection** — This risk concerns the possibility of having malicious code being injected and running on the mobile device or remote server via an application. Typically, such code is injected through forms of data that are not properly sanitized. The injected malicious code aims commonly at stealing sensitive information such as passwords, cookies, personal information and so on.

- **M8 - Security Decisions Via Untrusted Inputs** — Vulnerabilities falling into this category concern techniques used by developers to distinguish users or providing different functionalities. For example, sometimes, hidden values are used to represent users with different levels or permission, and these values may be somehow manipulation with devious intentions. For example, an attacker can intercept calls to a web service and change the sensitive parameters. This process can lead to improper behavior on the part of the application or even to change the permissions to a higher level, thus giving access to areas or data that he or she should not have access to.

- **M9 - Improper Session Handling** — In order to facilitate communication between the application and the server, mobile applications use session tokens to keep state through protocols such as HTTP or Simple Object Access Protocol (SOAP). If authentication is successful, the server issues a session cookie for the application. Incorrect session handling occurs when the access token is shared unintentionally with an attacker during a communication with the server.

- **M10 - Lack of Binary Protections** — this risk is associated with the fact that no proper protections are added to the compiled form of the applications, which may result in exposing the application and the user to a wide variety of attacks. For example, most Android applications can be reversed engineered; the code and the structure of the project can be studied, eventually modified, recompiled and redistributed.

2.2.4 Insecure Data Storage

The risk identified by *Insecure Data Storage* in the OWASP top 10 mobile project has a more lengthy discussion in this chapter because this work was focused precisely on this aspect. This category includes those methods that may be used to somehow compromise the data stored in the internal file system of the device. The tentative to compromise data may come from (malicious) users or from malware with internal or external access to the device. It is widely known that easy access to the entire file system is provided after rooting or jailbreaking and Android OS or iOS, respectively. Yet, it is commonly assumed that, under normal circumstances, no application can access the internal storage spaced of the others, and developers assume that they can store sensitive data such as usernames, passwords or application data in the internal memory, leaving aside concerns to encrypt of assure the integrity of the data [And15g].

Christian Håland [Chr13], in his dissertation, examined various applications amongst the most popular of the store at the time. One of the analysis he made was in relation to the data storage insecurity. In his study he considered the problem of the rooted devices giving access to

sensitive data, which he defined as data being stored on the mobile device without encryption. The author analyzed the 4Pics1Word game, a games that went viral in the beginning of 2013. This game was about finding a word to define a set of images. To build the word, a set of 12 characters was provided, and virtual money could be used in-game to remove characters out of these 12, thus making the solution easier to find. These coins could be bought with real money. After an analysis of the application files, he realized that the application was keeping sensitive data in clear text, such as current photo ID, current level and number of coins owned by the user. With a simple change in the right file, he was thus able to increase the number of coins, becoming effective after restarting the game. 4Bilder1Feil was another game that the author analyzed. It was similar to the previous one and it was found susceptible to the same problem, since the application stores the total number of lives in clear text. After changing this number with a common text editor and restarting, the game would immediately assume the new value. Lastly, the same author analyzed the Wordfeud Free, which is a game similar to the traditional Scrabble. The game offers the possibility to connect two players via Internet to play in pairs. For that, it requires login with a Facebook account and, after analysis, the author found that the password was stored in clear text in the game files.

The FourGoats application was analyzed by James King [Jam14] within the scope of his master's, with focus on the insecure data storage topic. The author also stated that the attack could only be carried out if the device was rooted, since it required access and manipulation of internal application files. The application considered by the author allowed the user to log into the application, and it was thus possible that the application was storing the login details for later login, given that a *Remember Me* option was available. According to the author, this application saves the user credentials (username, password) in clear text in a SQLite file.

A researcher from Palo Alto Networks, C. Xiao, conducted a study on a set of 12.351 applications to find out how many of them were not allowing backup via the native utility. The results of the research were delivered in the form of a presentation with the title *Insecure Internal Storage in Android*, which was presented in the Taiwan Conference (HITCON) [Cla15]. He concluded that only 556 were not allowing backup, corresponding to only about 6% of the dataset. As the backup of an application for a computer is one way to get access to internal files, this could be a protection for this type of vulnerability. This study was mostly supported by automated means to download and reverse engineer the Android applications, and subsequently finding the `android:allowBackup` property in the `AndroidManifest.xml`. This study did not included a manual or in-depth analysis of those applications, contrarily to what was done here .

2.3 Android Debug Bridge

The Android Debug Bridge (ADB) [And15b] is a command-line tool, typically included with the Android SDK, providing means to communicate with phisically connected or virtual devices running the aforementioned OS. It is comprised by three main components:

- The *Client*, which is the tool facing the developer and can be invoked in a Command-line Interface (CLI) environment via the `adb` command. This program is capable of interpreting several sub-commands, such as `shell` and `logcat`;

- The *Server*, which runs also on the developer machine and listens to commands sent via

the client. It runs as a backgroud process. If it is not running when invoked by `adb`, it is automatically started. It is responsible for communicating with adb daemons running on connected devices (see below);

- The *Daemon*, which is a process running on Android devices, waiting for connections from the ADB *Server* when debugging is permitted.

In order to use ADB, one needs to either connect a physical device through network or Universal Serial Bus (USB), or set up and spawn an Android virtual machine. It is also required to enable USB debugging on the target OS, e.g., via `Developer Options`. Normally, the `Developer Options` are hidden in the Android OS, but it can be easily *unlocked* by finding the *Build Number* in `Settings` $\longrightarrow$ `About phone` and pressing it seven times. The `Developer Options` will then be reachable via `Settings`. Starting from the Android 4.2.2 version, the successful connection to the OS is only possible after explicitly (via dialogue box) accepting an Rivest Shami Adleman (RSA) key of the Server from the Android OS side. This particular step was added by the Android developers to prevent the feature from being used in an abusive manner without permission of the device owner.

ADB comprises a valuable asset for developers, providing straightforward means to easily install applications and debug them during execution. It also enables (to a certain extent) navigating the the file system of a device, amongst other features. Nonetheless, as a means to easily reach the mobile system, it also opened another attack vector.

Hwang et al. [HLKR15] presented a series of attacks that can be exploited through the ADB tool. The authors divided these attacks into several categories. For example, they mention the *Private Data Leakage* category, in which *Message Tracking* is possible. Typically, an Android application needs permissions to handle or see Short Message Services (SMSs) or Multimedia Messaging Service s (MMSs) but, since the system broadcasts the message after arrival, it is possible to obtain them via ADB (even without permissions) via the `dumpsys` utility, which aggregates all the information regarding notifications. One can even access messages from different users. Containment between applications can also be bypassed in some cases. For example, using the ADB `run-as` utility, an attacker (or the developer) can change its UID and Globally Unique Identifier (GUID) and acceses the files for particular applications at a time, thereby gaining access to private data. This category is referred to as *Private Database Access*. Another category is the one of *Behavior Interference*, in which they include the possibility to change the screen size to render it useless to the user of the device, forcing him or her to reboot the OS. This DoS effect can be achieved via the ADB `wm` utility. This command-line tool was of critical importance for the work described herein.

2.4 Conclusions

The interest in mobile security topics has been increasing, motivated by the growing market and also by the specific details surrounding the involved technology. This chapter provided some context on that subject, focusing on some works that have some resemblances with the one described herein. Nonetheless, none of the previous works tested throughly a big number of applications from Google Play in the same manner. In here, the applications are downloaded and tested in a computer, one by one, though a script was build to help systematizing part of

the procedure. The closest work concerns a study to find how many applications would allow backing up to a PC, but it was limited to downloading and finding the `allowBackup` tag in the `AndroidManifest.xml` file. The work described in this dissertation gives a better idea of the severity of this problem. The fact that OWASP currently considers that the Insecure Data Storage is the second major risk for mobile applications also justifies this study.

At the end of the chapter, a brief explanation of the ADB tool was included. The discussion can now flow into the details concerning the method used to assess if applications are storing values in an insecure manner, in which ADB plays a major role.

Chapter 3

Data Sets and Analysis of Android Applications

3.1 Introduction

Conducting a study on the security of the storage of Android applications with the characteristics enunciated in the introduction required building up data sets of such applications. In the scope of this work, two different and mutually excluded data sets were constructed: the first one was comprised of Android Games only, while the other contained many common Android applications (excluding games). Both data sets were build by humanly selecting some of the games and applications from the Google Play store and by letting a program, written in Java, collect and fill in the metadata of the applications in a MySQL database. Though mutually excluded, all applications had the *in-app purchases* characteristic as a common feature.

This chapter describes both data sets and the method used to analyze the susceptibility of the applications to data exposure or manipulation. The Android Games data set is described in section 3.2, while the Common Applications data set is explained in section 3.3. The method used to transfer, study and manipulate each game or applications is discussed with more detail in section 3.4.

3.2 Android Games Data Set

The first data set assembled in the scope of this work was just composed by games available in the *Google Play Store* [Goo15b]. All applications that were considered in this work were coming from the official store, mostly because this is the source of most installations worldwide. This data set was built and analyzed during the months of September and November of 2014. It is comprised by 849 games, which can be divided into different categories, as they are organized in the games section of *Google Play*. Table 3.1 shows the number of games falling into each one of the categories that were considered in this analysis.

Table 3.2 provides a different, equally important, perspective over the data set, namely in terms of the popularity of the games. It segregates the several games into 11 different ranges concerning their respective number of downloads. Table 3.2 shows that the data set gathers popular and less popular games. The procedure to collect games made no distinction regarding the number of downloads on purpose, as the study was aiming to assess if the vulnerabilities were present in any kind of mobile applications, and not restricted to a particular subset (e.g., to the most popular ones). Regardless of that, it is easy to verify that there are less games with downloads up to 100000, mostly because unpopular games are short-lived in such stores. The largest slide of the data set is comprised by games with 1 to 5 billion downloads, corresponding to approximately 35% of the entire data set. Nearly 71% had more than 1 billion downloads.

By not focusing on specific groups or characteristics of the applications (apart from the *in-app*

Table 3.1: Number of analyzed games by category.

Category	Number of Games
Action	141
Arcade	138
Puzzle	99
Casual	91
Strategy	91
Sports	58
Racing	50
Simulation	39
Adventure	37
Role Playing	35
Card	34
Word	16
Family	13
Trivia	6
Music	1
Total	849

Table 3.2: Number of games in the data set per number of downloads.

Number of Downloads	Number of Games-per-Interval
100 000 000 - 500 000 000	13
50 000 000 - 100 000 000	16
10 000 000 - 50 000 000	111
5 000 000 - 10 000 000	110
1 000 000 - 5 000 000	296
500 000 - 1 000 000	106
100 000 - 500 000	157
50 000 - 100 000	23
10 000 - 50 000	11
5 000 - 10 000	4
1 000 - 5 000	2

purchase feature), and unlike prior studies on Android OS security, e.g., [FCH+11, FGW11, EOM09, BKvOS10], which considered applications with greater popularity in *Google Play* (or Android Market, as it was previously designated), it is possible to later try to potentially relate the popularity of applications with *Insecure Data Storage* problems.

3.3 Android Applications Data Set

The analysis of Android games was stipulated as the starting point for this project, which could later be expanded to other types of applications available on *Google Play*. The second data set created in the scope of this project was initially comprised of 693 applications with the *in-app purchases* characteristic, excluding games and Live Wallpapers and Widgets (see below). Similarly to what was done for the previously described data set, its construction, namely the process of selecting and downloading the Android Application Packages (APKs) and storing metadata concerning each application in a MySQL database, was done by hand and resorting to a program written in Java. The 693 applications were then revised one by one during the assessment of the susceptibility to data manipulation, and some of them had to be left out of the final analysis due to reasons discussed in chapter 4. As shown in the Table 3.3, 24 of the 26 existing categories in the application section of Google Play were considered for this study. The

categories Live Wallpaper and Widgets were not considered because they contain applications that are already included in other categories, while the other categories have no intersections.

Table 3.3: Number of downloaded applications for the second data set by category.

Category	Number of Applications
Books & Reference	30
Business	30
Comics	30
Communication	30
Education	30
Entertainment	30
Finance	30
Health & Fitness	30
Libraries & Demo	11
Lifestyle	30
Media & Video	30
Medical	30
Music & Audio	30
News & Magazines	30
Personalization	30
Photography	30
Productivity	30
Shopping	23
Social	30
Sports	30
Tools	29
Transportation	30
Travel & Local	30
Weather	30
Total	693

Since the number of applications in Google Play is growing and is larger than the number of games, it was deliberately decided to limit the number of downloaded applications for each category to 30. The main reason behind this decision was to keep the assessment manageable (part of the study was performed manually). This is also noticeable in table 3.3. It was not possible to get 30 applications for all categories, because in some of them there were not enough applications with the *in-app purchases* feature. As with the previous data set, an effort was made so as to not favor popular applications during the selection procedure. Table 3.4, depicting the number of applications per download interval, shows that most of the data set is comprised by applications in the range starting at 10000 and ending at 50000000 downloads, but also that all intervals have at least 1 hit. This is mostly due to the morphology of the Google Play applications, since it is more frequent to find them in the intermediary intervals than in the bottom or top ones.

3.4 Method for Analyzing Android Applications

The overall method behind this work has 3 major phases: (i) collecting applications and games; (ii) analyzing each one of the software pieces in the data sets; and (iii), summarizing the results. This section is focused on phase 2. It contains the detailed description of the method used to analyze each one of the games and applications in the data sets, which was adapted from

Table 3.4: Number of applications in the second data set per number of downloads.

Number of Downloads	Number of Applications-per-Interval
100 000 000 - 500 000 000	4
50 000 000 - 100 000 000	9
10 000 000 - 50 000 000	42
5 000 000 - 10 000 000	40
1 000 000 - 5 000 000	120
500 000 - 1 000 000	71
100 000 - 500 000	197
50 000 - 100 000	61
10 000 - 50 000	89
5 000 - 10 000	17
1 000 - 5 000	23
500 - 1 000	6
100 - 500	11
50 - 100	2
10 - 50	1

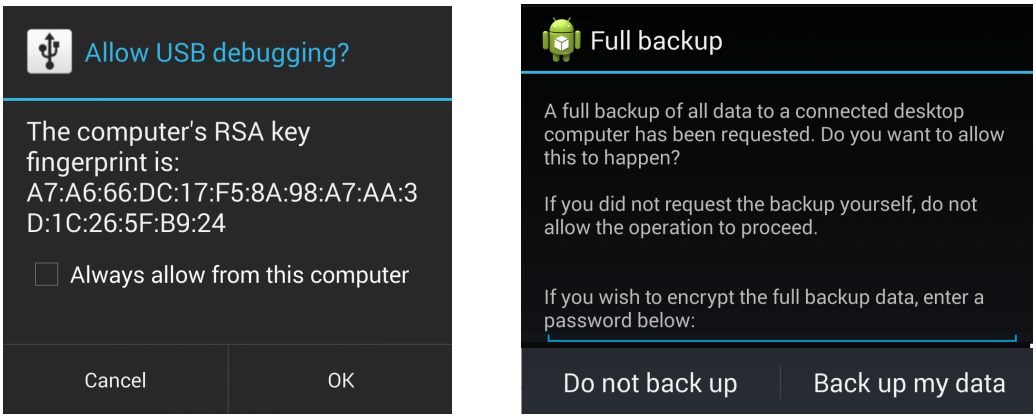
the procedure in [XDA15]. From the description, it will become clearer that this process was almost entirely manual, though some tools helped in some of the steps. From a broad and naive perspective, the aforementioned method can be decomposed in 3 major parts also:

1. Initially, the application or game was installed in a smartphone with the Android OS. Their features were then tested. For example, blocked or available features were enumerated. Values of interest, which could somehow determine the flow of the application, were also identified. In the case of games, the values of interest could be the number of coins of the avatar or the current number of the level. Sometimes, the game (or application) was used up to a given point of interest, as the point at which it was necessary to buy some item via *in-app purchases*.

2. The application was then transferred to a desktop computer running a Linux OS, using the procedure described below, where the data files were inspected and sometimes modified. For example, if a file with the (previously mentioned) number of coins was found, it would be appropriately manipulated to contain a different value. The application was then packaged and transferred back to the Android environment.

3. Finally, the application or game was tested to see if the manipulation was successful in some noticeable way.

One of the most concerning details of this study is that no special privileges are needed in any of the OSs used. The method required the usage of an Android smartphone (though a virtualized device could also be used) and a computer with a specific software installed. In this case, a non-rooted `BQ Aquaris E5 FHD` with the `4.4.2` Android version was used, but any smartphone or tablet with an Android OS version greater than *4.0.0* would suffice, since it was this version that introduced the backup utility [And15f]. The backup utility is crucial for the method, as it provides the means to transfer the application to the computer and back. The Android OS needed to have the debug mode enabled.



(a) RSA computer key. (b) Backup Utility.

Figure 3.1: Some screen options that appear during the method.

For the transference of the applications to the computer, a USB cable and connection was used. The communications were managed using the ADB tool [And15b], which needs to be previously installed in the computer. It is part of the Android SDK. The ADB tool interacts via command line, and lets the user communicate with a emulator instance or Android connected device in a simple manner. On the desktop computer, apart from the ADB tool, tools such as `pax`, `tar`, `OpenSSL`, `dd` and `grep` need to be available also. Typically, `OpenSSL` is used to perform cryptographic tasks but, in this case, it was used to compress and decompress packages resorting to the *zlib* library. The `pax` tool is used to read and write files, as well as copying directory hierarchies. `dd` and `tar` are responsible for converting files to the tar format and extracting their contents, respectively. Finally, `grep` is used for searching patterns in files using regular expressions. Most of these tools are included in common Linux distributions. Ubuntu 14.04 was the OS in the desktop computer.

The transference, manipulation and analysis, and restoring of the Android applications can be comprehensively decomposed in the 10 steps detailed below. Most of those steps can be carried out in a traditional Linux shell, resorting to the aforementioned tools:

1. The first step consists in connecting the device to the computer via USB. As the debug mode is active, when the connection is performed, the security parameters of Android OS ask the user to allow the computer to use the smarthphone in debug mode, showing also the RSA public key of the computer, as shown in figure 3.1a. The `OK` button should be pressed, otherwise the method will not work.

2. The app is then backed up to the computer with a command similar to
   ```
   > adb backup -f data.ab -apk PATH
   ```
   The `PATH` parameter represents the game path on the smartphone internal storage and should be replaced by the real game path. This command will trigger a new activity on the smarthphone (illustrated on the right side of figure 3.1b), that asks the user about the authorization to perform the backup operation for that application. The user should select the option with label `Back up my data` in order to conclude the backup process successfully. After successful execution of this command, a compressed and non-encrypted file (in this case referred to as `data.ab`) is generated. This file contains a header with 24 bytes, along with the files that composed the application.

3. The third step is where the header of the `data.ab` file is removed and the remaining part is converted into a tar file. This will enable the consequent extraction of the compressed files. The commands for performing both tasks are similar to
```
> dd if=data.ab bs=1 skip=24 |openssl zlib -d > data.tar
```

4. The fourth step consists into getting a perfect (i.e., ordered) list of the files that are inside of the file generated in the previous step. This a important step, since in the restore process, the files of the application need to be assembled in the same order in which they were backed up, or an error will be issued by the utility stating that the package has been changed. This task can be performed using the following command in the shell
```
> tar -tf data.tar > data.list
```

5. In the fifth step the `data.tar` file is decompressed. This operation allows access to all files in a readable format. The extracted files are under a newly created folder with the name of the respective application. This is achieved via
```
> dd if=data.ab bs=1 skip=24 |openssl zlib -d |tar -xvf -
```

6. With access to all files of the application, it is possible to analyze and modify data files. In the scope of this work, the most interesting files were the ones potentially holding sensitive data like credentials or the ones whose contents impact the flow of the applications. Most of the times, the `grep` tool was used to find values of interest in files, with commands similar to
```
> grep -R "xxx" app/PATH/
```
This command searches for the string `xxx` in all files contained in the `app/PATH/` folder or sub-folders (`-R` option). Having identified the files, they were modified using the most appropriate means (e.g., a text editor was used to edit `XML` files, while `SQLite` was used to manipulate SQLite3 databases).

7. The seventh step delineates the beginning of the restoring process. It follows the process of the backup and modification steps in a reverse order. In the first place, is necessary to compress all files in the same order they were uncompressed. This information was saved in the *data.list* file (see fourth step). The command to perform this task is
```
> cat data.list |pax -wd > newdata.tar
```
The successful execution of the command generates a new *tar* file (`newdata.tar`), containing all files in the order defined in data.list.

8. It is then needed to re-insert the header removed in step three. This header is fixed and known for decrypted Android backups and can be replicated via
```
> echo -e "ANDROID BACKUP\n1\n1\nnone" > backup.ab
```
The previous command creates the file `backup.ab` with the string `ANDROID BACKUP`. The file generated in this step is the file that will be restored to the smarthphone in the last step.

9. The ninth step is where the header is concatenated with compressed application file. This may be achieved via
```
> openssl zlib -in newdata.tar >> backup.ab
```
Notice that this command compresses `newdata.tar` and concatenates the result at the end of the `backup.ab` file.

10. Finally, the application is restored via

    ```
    > adb restore backup.ab
    ```

    The Android device needs to be connected to the computer (and recognized by the system) for the command to succeed. A message will be spawned in the Android device asking for permission to restore the application.

When looking for information exposure problems (e.g., credentials), following the method up to the sixth step was enough. When assessing the susceptibility to data manipulation, it was necessary to carry out the entire method and then test the application in the Android environment. For example, in the case of games, it was necessary to see if the inflicted modifications did not render it useless (corrupted) and if they enabled cheating the game or access paid functionalities.

3.5 Conclusions

This chapter is focused on two phases of the method that was used to evaluate if the data files of Android applications could be manipulated to change their behavior. It presented details that characterize the two data sets that were build. One of the data sets is comprised of games only, while the second one contains typical applications. The first is comprised of 849 software pieces and the second contained, initially, 693 applications. The applications were mostly chosen manually and then fed to program for gathering their information. An effort was made not to favor any particular Google Play category or download range, though the higher number of applications in some ranges translated into more of those in the data sets.

The method used to transfer and analyze the applications in a desktop computer does not require administrator privileges on the Android device, exploiting the native Backup utility provided by the mobile OS. It uses only open source and readily available tools, emphasizing that any user with some background or following tutorials can perform it to access potentially private data or access paid features. Having described the data sets and this method, it is possible to proceed to the discussion of the results in the following chapter.

Chapter 4

Assessment of the Susceptibility to Data Manipulation of Android Applications with In-app Purchases

4.1 Introduction

This chapter presents the results obtained after applying the method described in the previous chapter to the applications and games of the data sets. It is thus divided into two main sections, each devoted to the discussion of the results for each one of the data sets: section 4.2 is focused on Android games, while section 4.3 is dedicated to typical Android applications. Near the end of the chapter, section 4.4 elaborates on the pertinence of the results in terms of the threat they represent.

4.2 Analyzing Android Games with In-App Purchases

The study reported in this section concerns the data set of 849 free (to download) games with the *in-app purchase* characteristic. From those 849, a total of 148 were susceptible to data manipulation performed using the method described in section 3.4, which corresponds to 17,43% of the tested games. It can be said that this number is significant, specially if one takes into account that the procedure that was employed was not that sophisticated. Malicious users with more free time and motivation, focused in one application only, would easily bring this number up. At least 1 in each 6 games of *Google Play* are vulnerable and it is easy to build scripts that automate the method used in the scope of this work for any given application (discussed with more detail below).

Figure 4.1 provides a perspective over the relationship between the number of games that were found to be susceptible to data manipulation with the total number of games in the data set for each category considered in *Google Play*. The *Trivia* category is the one with a larger number of vulnerable games, with 50% found to be susceptible to data manipulation. Nonetheless, it should be also mentioned that this is the categories with fewer games (only 6 games). Usually, this type of game is prepared to work both online and offline. As such, many come with all the features (paid or free) implemented, though the paid ones may be blocked via some programming logic. *In-app purchases* are typically used to provide access to information or tricks that helps answering certain questions of the game. In the vulnerable games, the data controlling the purchased add-ons was stored in plaintext files without any integrity mechanism. The next two most vulnerable categories are the ones of *Racing* and *Arcade*, with 28% and 26% of the games are susceptible to data manipulation, respectively. In these games, the paid add-on functionality is either associated with the virtual money (used within the game) or with faster ways to progress in the game, namely buying certain virtual items. These results are directly related with the fact that the values that represent the virtual items are easily found in the data files, namely in plaintext eXtensible Markup Language (XML) or text files, or in SQLite databases.

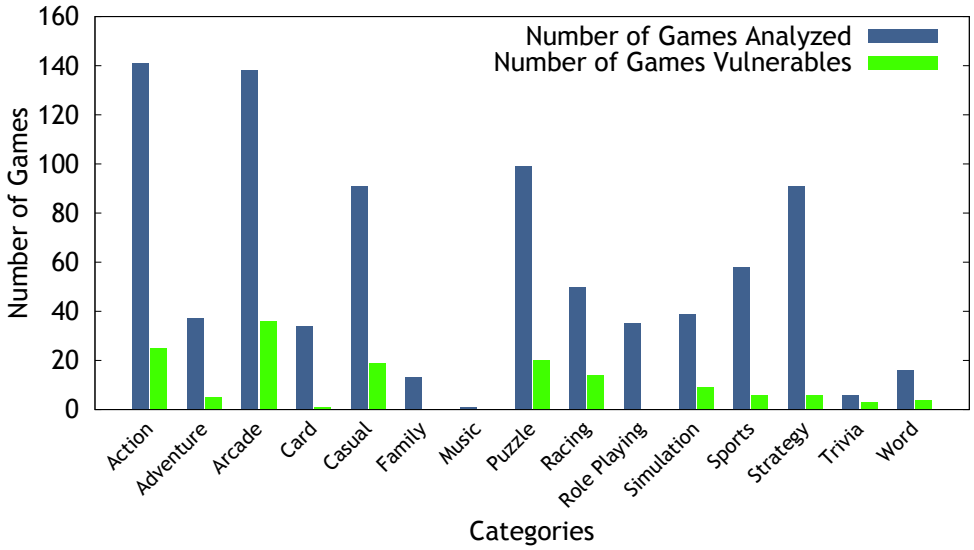


Figure 4.1: Total number of games versus the number of games susceptible to data manipulation per category.

They are stored without encryption, integrity or authentication codes.

The categories of *Role Playing*, *Family* and *Music* had no vulnerable games. These results are explained by the fact that, for these types of game, the *in-app purchases* are used to remove advertising, or for example buy the whole game or expansions (new levels, characters or weapons). Sometimes, applications allow the user to make purchases within the application but require additional files (or a new version of the game) to be downloaded. As such, the features are neither installed (but blocked) with the free version of the application, nor potentially interesting data files are available for manipulation before payment. Usually, purchases requiring the device to communicate with the remote server are less susceptible to this type of attack because, in such cases, it is not about changing the flow or status information of the application anymore. The games in the Role Playing category are typically the ones that change more during the lifetime of the game (e.g., new characters are added at different levels, downloaded on-demand). These changes are also leveraged to improve security or solving reported flaws (i.e., a patch is delivered more regularly to such games). This behavior contrasts with the one of games in the *Arcade* category, in which changes are rare after their initial release.

The categories with less susceptible applications to data manipulation are the ones of *Cards* and *Strategy*, with 2.9% and 6.5% of vulnerable games in the data sets, respectively. These values can be explained by the fact that games belonging to these categories are designed to work online and the user data is typically stored remotely. The data is checked (local and remote copies of files are compared) frequently and manipulated values are detected and overwritten. For example, it was noticed that some games allowed changing some values in the PC, but they were then restored when the connection was reestablished.

A different view over the results is provided in Figure 4.2. In this case, the number of vulnerable games is plotted against the respective download interval. The total number of games of the data set in each interval is also depicted, for comparison purposes.

The intervals containing more games susceptible to data manipulation are the ones of 5000−10000

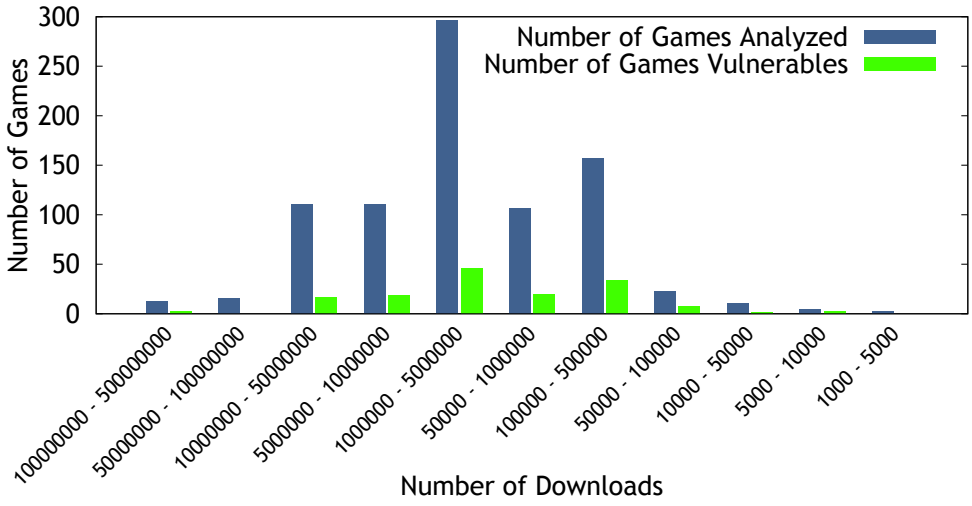


Figure 4.2: Total number of games versus the number of games susceptible to data manipulation, segregated by popularity.

and 50000−100000, with 50% and 30% of vulnerable games. The interval delimited by 5000 and 10000 had only 4 games in the data set and the statistic is thus not significant (2 out of the 4 were vulnerable). There were 27 games in the second interval, 7 of them were found vulnerable. The most important conclusion taken from this part of the work is that the Susceptibility to data manipulation does not seems to be related with the popularity of the game, since the number of vulnerable games is larger in intervals with more samples and vice-versa, with larger fluctuations in scarcer intervals. In the interval corresponding to the highest level of popularity (100000000−500000000), there were 2 vulnerable games in a total of 13, adding up to approximately 15%. Curiously, the data set had 16 games in the preceding range (50000000−100000000) and none was found vulnerable. The interval with more samples in the data set was the one delimited by 1000000 and 5000000, containing a total of 296 games. 46 of these games were found vulnerable, which corresponds to approximately 15%.

Figure 4.3 shows the types of file that are used to store application data in the internal storage of the Android OS for the analyzed games that were found vulnerable. From the pie chart, it is possible to conclude that, in 76% of them, the data is saved in XML files, while only 9% use SQLite. The remaining 14% use other file types, such as JavaScript Object Notation (JSON) or text files. This distribution was expected since XML comprises a simple and standard format for storing data and, in Android, one of the suggested storage options for saving application data is known as *Shared Preferences*, which is used to save values from primitive JAVA types in XML format via the `SharedPreferences` class of the SDK. From these results, it is possible to also conclude that data manipulation can be done in different file formats. Data stored in SQLite databases is also stored in American Standard Code for Information Interchange (ASCII) (except if encoded using application logic) which also enables one to easily find patterns in the files using tools like `grep`.

Interestingly, only 21 games of the data set (which represents approximately 2.5%) had the `android:allowBackup` property set to `false` in the `AndroidManifest.xml`. Setting such property to `false` constitutes one way of protecting the application against data manipulation in non-rooted OSs, since legitimate users can no longer back it up to a computer. Unless specifically set in the manifest, this property is set to `true` by default. The results show that only a few

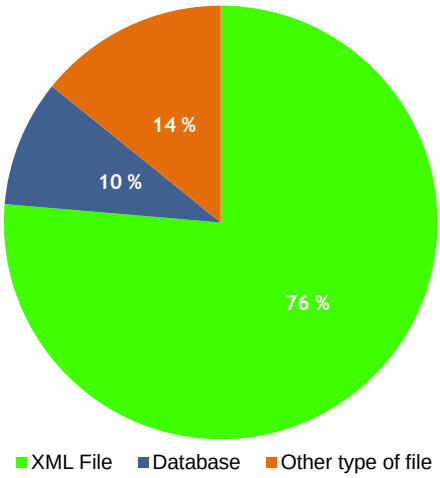


Figure 4.3: Type of file used to save application data in the Android internal storage in vulnerable games.

developers set it up intentionally. On the other hand, if the OS is rooted, manipulation can still occur without requiring transferring the backup.

Finally, it should be mentioned that there were 27 games cataloged with the *in-app purchases* characteristic that did not really possess any functionality for purchases within the application. This is probably due to erroneous selections from the developers when uploading to the store. In another 3.9% of the data set (i.e., 27 games), the in-app purchase feature was used as a means to remove advertisements or to buy their respective full version, which are of less importance to this study. Normally, purchasing the full version requires downloading additional files and, as such, the game is not vulnerable in the sense considered in this work.

4.3 Analyzing Android Applications with In-App Purchases

The study described above was performed prior to the one discussed in this section. Because of that, the procedure for building and analyzing Android applications (excluding games) was different. On the one hand, it was more compartmentalized in the sense that the applications were first identified and only later downloaded and superficially analyzed; on the other hand, filters were applied to the data set before the more detailed analysis via transference to the computer. Initially, 693 applications from 24 categories of the *Google Play* store were identified. Their links were then fed to the Java program responsible for downloading them and collecting their metadata. All applications in which the *in-app purchases* were used to remove advertisements or having the `android:allowBackup` set to `false` in the `AndroidManifest.xml` were then excluded from further analysis. Applications that were not compatible with the Android device in which their features were tested (the *BQ Aquaris E5 FHD*) or those that really had no functionality to buy anything (despite the *in-app purchase* characteristic) were also left out. Actually, the applications referred in last were of less interest for this study, mostly because they are typically extremely simple applications with little value to users. De decision to apply these filters were mostly based on lessons learned from the initial work on Android games. After applying these filters, the data set was reduced to 377 applications, corresponding to 54% of the original set. Figure 4.4 provides an idea on how these filters affected the data set for each category.

The chart in Figure 4.4 clearly shows that the *Medical* and *Education* categories were the cate-

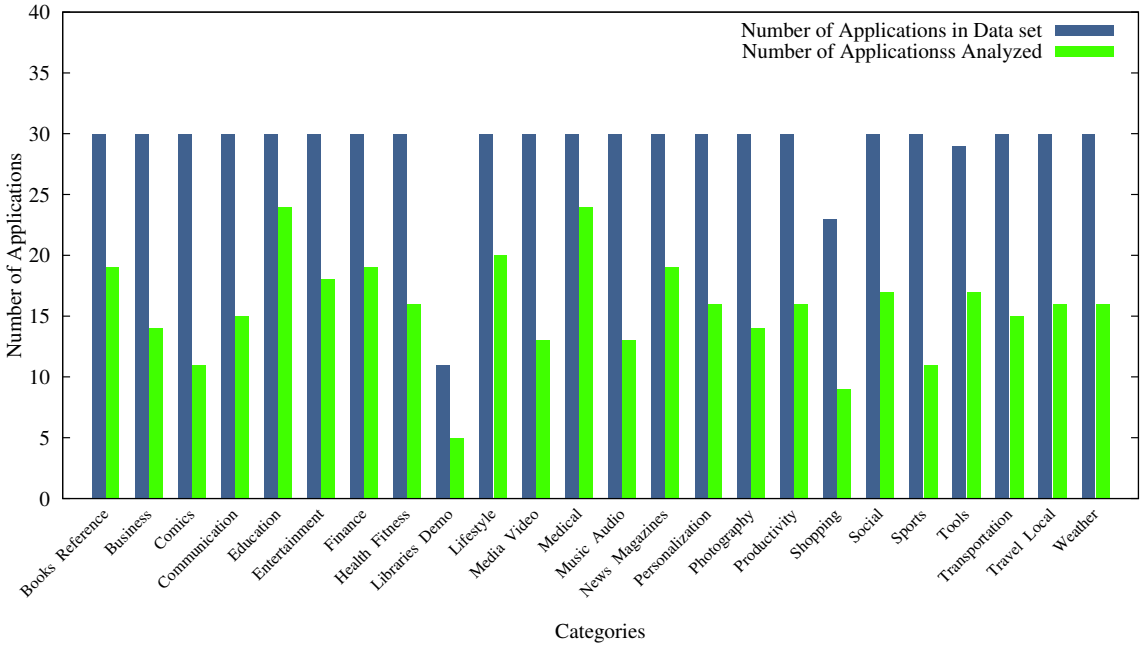


Figure 4.4: Number of downloaded applications vs. number of applications fulfilling the conditions to analysis, divided per category.

gories that had more applications to be analyzed, both with 80% of applications to be considered in the manual analysis phase. This categories have typically more complex applications and the *in-app purchases* are more commonly used to add functionality, rather than for advertising removal. On the opposite side one can find the categories of *Comics* and *Sports*, with only about 37% of applications considered for further analysis. The cut in the *Comics* category is explained by the fact that the respective applications are commonly used to download and read comics for free, with developers resorting to advertising as the source of income or, alternatively, to their removal via purchases. The reasons behind the cut for the *Sports* category are similar. The respective applications are mostly used to relay sport related news to users, namely game results free of charge, using advertising as the source of income. The remaining 20 categories suffered cuts close to the average (i.e., 54%).

The impact of the filters in the data set in terms of popularity is depicted in Figure 4.5. Considering only percentages, the interval corresponding to 10−50 downloads had the highest rate of not excluded applications (100% of the applications in the initial data set made it through the next phase). Nonetheless, this number is not very significant since there were only one application in the group. The intervals corresponding to 50000000−100000000 and 5000000−10000000 downloads come immediately after, with 67% and 62% non-filtered out applications, respectively.These results led to the conclusion that popular applications are often more complex where the functionality of *in-app purchases* is more frequently used to obtain items and extra features rather than for removing advertisements. In the opposite side, the intervals of 500−1000 and 100−500 had the highest exclusion rates, where only 33% and 9% of the applications were kept, respectively. These findings also corroborate the statement that popularity and more elaborated ways of using purchases in applications are related.

Having described the conditions that narrowed down the data set, the results obtained after aplying the method discussed in Section 3.4 are now included. These results are provided in two

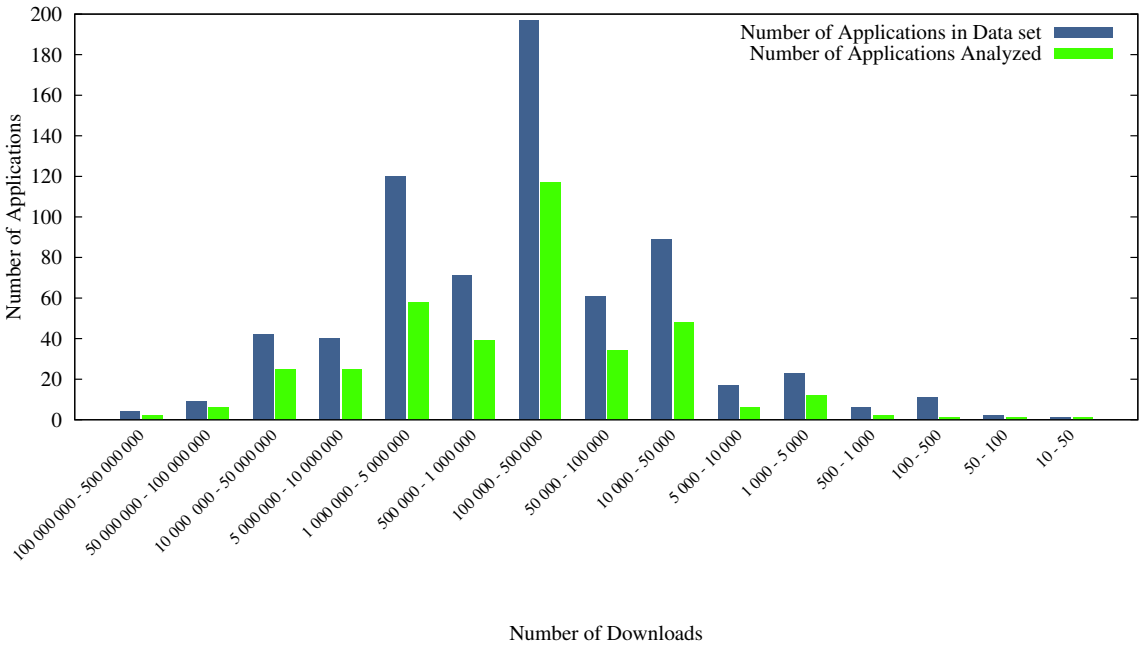


Figure 4.5: Number of downloaded applications vs. number of applications fulfilling the conditions to analysis, divided per popularity.

main perspectives, similarly to what was previously done for the Android games. The study of these applications was more profound than the aforementioned one, because it included searching for data exposure problems too. In this case, all applications that were either susceptible to data manipulation or whose sensitive information was exposed were considere vulnerable. *Data exposure* refers to having sensitive information such as credentials (e.g., username and passwords) stored in plaintext in the data files.

Figure 4.6 shows the number of applications susceptible to data manipulation or exposure against the respective number of applications in the subset. The values are segregated by category. The category with a greater number of vulnerable applications is the one of *Media & Video*, with approximately 46% of vulnerable applications (six out of 13). The *Finance* category follows closely with approximately 42% of vulnerable applications (eight out of 19). Applications in the *Media & Video* are comprised by editors and readers. In the case of editors, it is common for developers to pack the entire features of a fully fledged editor into the free version of the application, which are initially blocked using application logic and values. These features can be unblocked using *in-app purchases*, which explains these results. The prominent percentage obtained for the *Finance* category is more due to data exposure, as the respective applications frequently include a login functionality, and store the credentials in auxiliary files without applying proven security mechanisms. Many times, developers assume that the application is protected by the OS and that it is not possible to access its files outside the application environment, which is not enough. Moreover, it was possible to verify that many of these applications were using Personal Identification Numbers (PINs) to somehow control the access to their features, and that the PIN was not only stored in plaintext in the data files, as it could also be manipulated to a desired value without detection.

It is worth mentioning that there were four categories in which no vulnerable applications were found, namely *Business*, *Libraries & Demo*, *Photography* and *Productivity*. In most of their

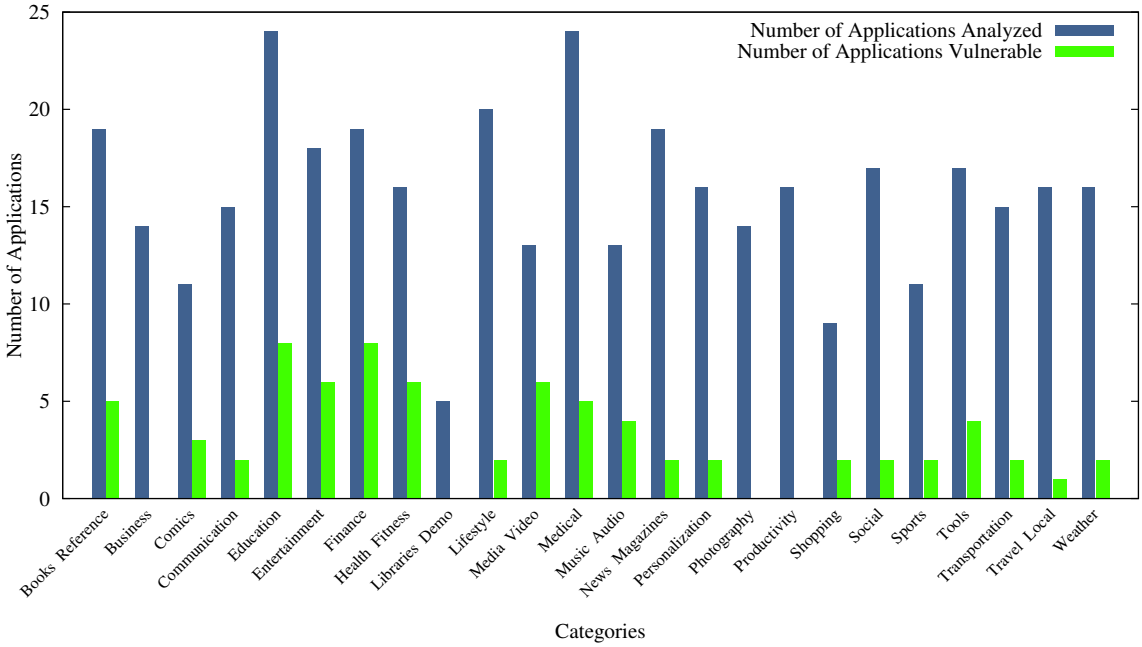


Figure 4.6: Number of applications susceptible to data exposure or manipulation vs. number of analyzed applications, divided by category.

respective applications, the *in-app purchases* were either used to remove advertisements (hence not making it to the final analysis), or the paid features or expansions require downloading entirely new files, leaving no room for data manipulation *a priori*.

The relation between popularity and susceptibility to data manipulation or exposure is illustrated in Figure 4.7. From its analysis, it can be concluded that the interval with more vulnerable applications is the one delimited by 50000000 and 100000000, with approximately 33% failing in the utilized method. This range corresponds to very popular applications. There are five intervals with no vulnerable applications, namely the four first ranges and the one containing the applications with more than 5000000000 downloads. These intervals were only represented by one or two applications in the data subset, due to reasons already discussed before.

Focusing solely on the vulnerable applications, Figure 4.8 puts the preferred formats to store the data into perspective. The pie chart shows that approximately 70% of the vulnerable applications (52 out of 74) were using XML files to store data and credentials, a number similar to the one obtained for games and depicted in Figure 4.3. The remaining 30% were using SQLite databases.

In the subset of applications considered for analysis, there were 87 with a built-in login functionality. From these, 23 of them were storing the credentials (e.g., usernames, passwords or PINs) in plaintext files, corresponding to 26%. Moreover, in this second data set, there were more applications not allowing backup via the `android:allowBackup` property in the `AndroidManifest.xml` than in the games data set.

4.4 Discussion on the Severity of the Findings

The percentage of Android applications and games susceptible to data manipulation is significant if several aggravating factors are taken into account:

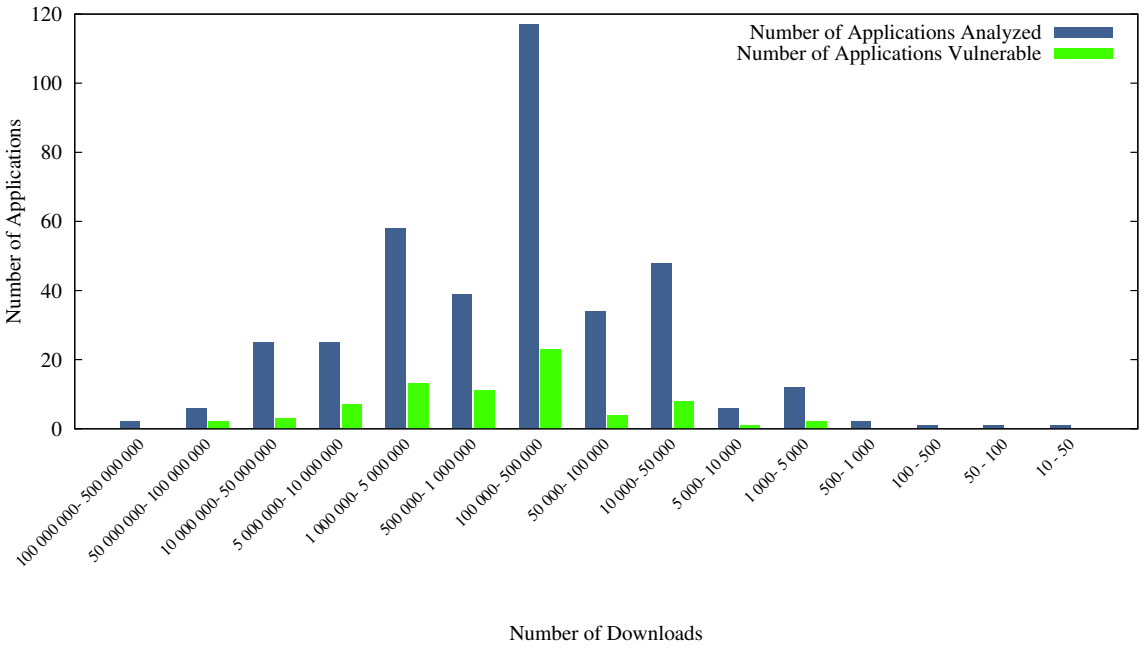


Figure 4.7: Number of applications susceptible to data manipulation vs. number of analyzed applications, divided by popularity.

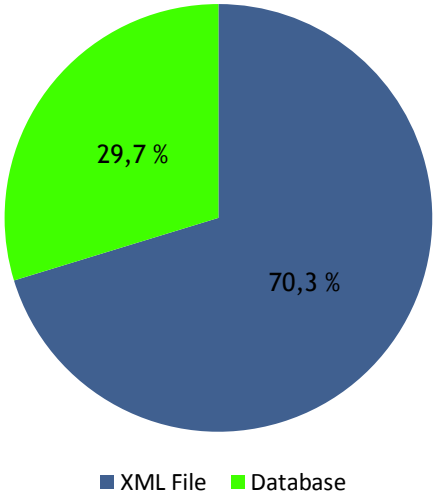


Figure 4.8: Type of file used to save application data in the Android internal storage in vulnerable applications.

1. First, it should be emphasized that the method employed was very general and not application specific. Potentially more prominent numbers would have been achieved if the analysis was more tailor-suited and if there was more time for each individual analysis.

2. The vulnerability (or the major entry point for exploring it) derives from a native OS mechanism and utility, which is actually useful to end users. The main purpose of the backup utility is beneficial and should not be removed. Furthermore, no administrator privileges are required on the Android OS to transfer, analyze or manipulate the applications.

3. It is already possible to find programs and scripts in the wild that effectively exploit the method for specific games, such as *My Talking Tom*.

4. In the worst case scenario, it was found that 1 in each 6 applications or roughly 1 in each 5 games were prone to data manipulation using only well known and freely available tools. Apart from the activation of the debug mode in the Android OS, the connection of the USB cable and the acceptance of the connection, the entire procedure can be performed from a shell, and can be easily automated via scripts.

5. Many users leave the smartphone unlocked over desks, being easy to quickly connect a USB cable and download the applications to steal private information such as PINs or passwords.

6. Finally, several design and implementation artifacts, discovered during the course of this work, provide a clear indication that the problem can only be worst. There was one application storing information from other users (such as e-mail addresses and usernames) locally, while another one enabled impersonating an arbitrary user just by manipulating the username after a successful authentication. These problems are not only due to an over confidence on the security provided by the OS, but also to an insufficient knowledge of the platform by the developers.

4.5 Conclusions

This chapter summarizes the main results and findings obtained during the intermediate and lengthier part of this Ms.C. program. It also brings the focus to the severity of the problem at hands. The analysis of the susceptibility to data manipulation and exposure of Android applications and games was performed for those with the *in-app purchases* characteristic, for which the costs of being cracked are more significant and that may effectively motivate attackers. Unfortunately, within the Android universe, the method to alter application data can be automated, and performed by people with only little expertize and using open source tools.

Results show that the data of 1 in every 6 games and 1 in every 5 applications can be modified to obtain functionality or virtual items that would otherwise need to be paid for. Many applications (e.g., from the *Finance* category) are storing PINs or passwords in plaintext in XML or SQLite, which are easily researchable using regular expressions, since the values are represented in ASCII.

While some developers are setting the backup option off for their applications, the solution should instead include the integration of adequate and well established security mechanisms, such as encryption and integrity checks. These mechanisms should be integrated depending on the requirements of the applications. The problem is not simple to solve and the next chapter elaborates on possible general solutions.

Chapter 5

Mechanisms for Preventing Data Manipulation in Android Systems

5.1 Introduction

This chapter discusses several potential solutions to the previously identified problem. The solutions are dividided into two main groups: the ones may be integrated on the OS itself, which are more generic; and the ones that may be natively integrated in the mobile application. While the former does not require the developers to be concerned (as much) with the security problem, the second enables fine-grained control over security requirements, namely data confidentiality and integrity. The later requires developers to know more about security mechanisms and their correct integration in applications.

This chapter starts from the presentation of the current way of functioning of the backup procedure in section 5.2, to then discuss local and remote software based solutions for the backup utility in sections 5.2.2 and 5.2.3, respectively. Proposals for enhancing each of these solutions using Acorn Risc Machine (ARM) TrustZone technology are then discussed in sections 5.2.4 and 5.2.5. Lastly, several resources that developers may use to secure their applications are descried in section 5.3.

5.2 Mandatory Message Authentication Code

The USB debugging functionality was integrated in the Android OS with the intention of making the life of developers a lot easier. It enable them to perform debugging directly on the devices instead of simulators. In many cases, it is in fact faster to compile, prepare the package and install the mobile application on a physical device connected to the development computer, than to use an emulated one. Additionally, there are functionalities that are not that easily simulated in a virtual environment, namely interaction with user and sensor related. Combined with ADB, USB debugging comprises a powerful tool. They allow installing applications, send commands to the device, navigate in the Android system with the traditional Linux commands, perform backups, amongst other features and via command line. Unfortunately, USB debugging also paved the way for some vulnerabilities and constitutes a way to easily install applications from unknown sources. It is also via USB debugging that the backup of Android applications (using the native OS utility) is possible. The next subsection will thus briefly describe its functioning.

5.2.1 Current Functioning of the Backup utility

Figure 5.1 contains a high level scheme that represents the functioning of the Android backup utility. It shows that, in order to perform a backup, it is necessary to have the phone connected via USB to the computer (actually, it would be possible to avoid the usage of the USB cable if a solution line WiFi ADB [Goo15d] was used, though debugging has to be active either way). The scheme shows that the backup utility is part of the OS. The output of this utility is an archive

that contains the data of 1 or more applications. The archive contains files whose access is not permitted in the non-rooted OS environment.

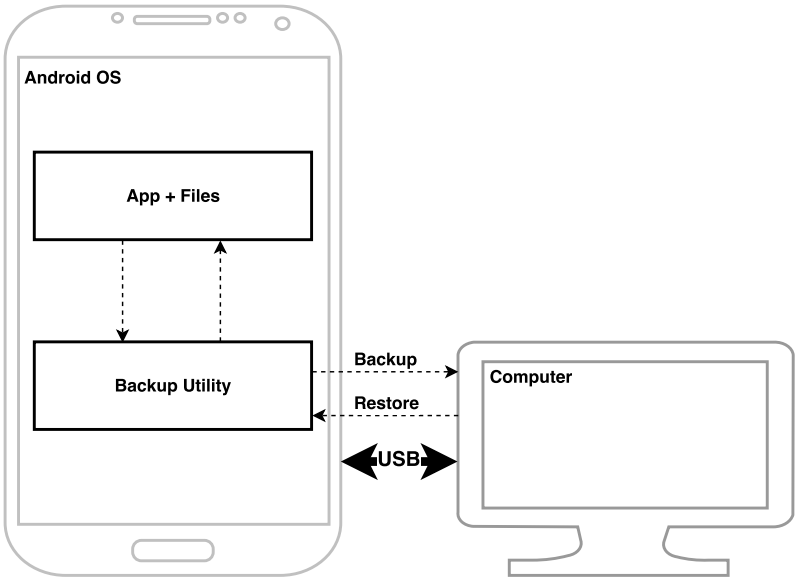


Figure 5.1: Scheme of the functioning of the Android OS backup utility.

The objective of the backup utility is to provide legitimate users with means to backup their applications, which may be useful in many situations, including for when he or she needs to switch paid applications to a new phone. Due to its purpose, the entire contents and data of an application needs to be included in the backup archive, giving rise to the problem exploited in this work. The utility provides the option to set a password for protecting the resulting file, by encrypting it with a derived key. If the field is left empty, no encryption is applied.

5.2.2 Approach #1 — Software Modification to the Backup Utility

The simpler (and perhaps most logical) approach to solve the problem is to integrate a suitable (mandatory) integrity mechanism in the backup utility. Figure 5.2 shows where this solution would be positioned using a scheme similar to the previous one.

This solution is the simpler because it consists solely in the integration of programming logic that produces an integrity code when a backup is to be created, and checks that integrity during restoration. The most suitable integrity mechanisms for this situation would be either a Message Authentication Code (MAC) or a digital signature. This first approach assumes that the integrity code (MAC or digital signature) is added to the backup archive and sent to the storage device with it (storing the integrity codes on the device would not favor portability). These solution (as well as the others), do does not pose any burden in terms of computational or storage overhead.

There are two problems with this approach: (i) the keys used to generate or verify the integrity code need to exist on the device, or there should be a publicly known mechanism that enables deriving them from user input; (ii) if the keys are only on the device, it is difficult to backup from one device to another. A mechanism to transfer the keys would be needed. Nonetheless, this mechanism would increase the difficulty of malicious attempts.

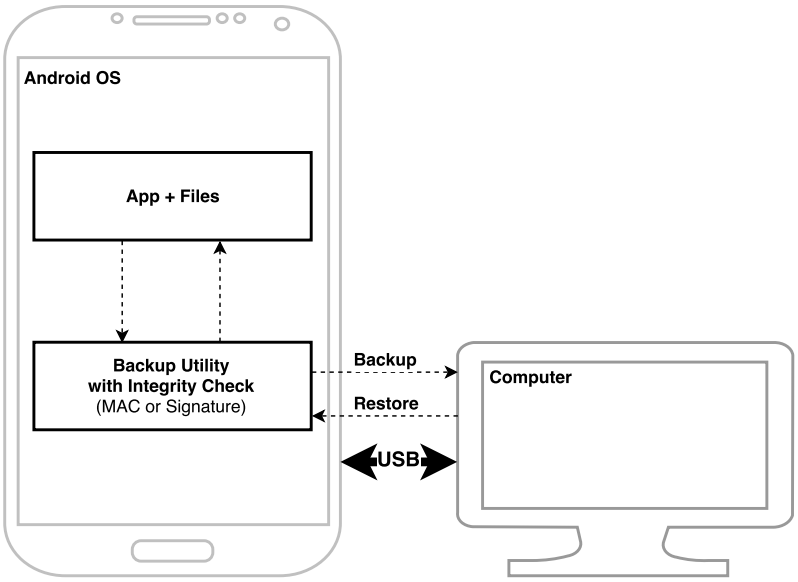


Figure 5.2: Solution #1 — Inclusion of cryptographically secure integrity mechanisms in the backup utility.

5.2.3 Approach #2 — Software Modification to Backup Utility with Networking and Server for Storing MACs or Digital Dignatures

It is possible to address the two problems mentioned in the final part of the previous section by adding a third party to store keys and integrity codes. A scheme for this approach is depicted in Figure 5.3.

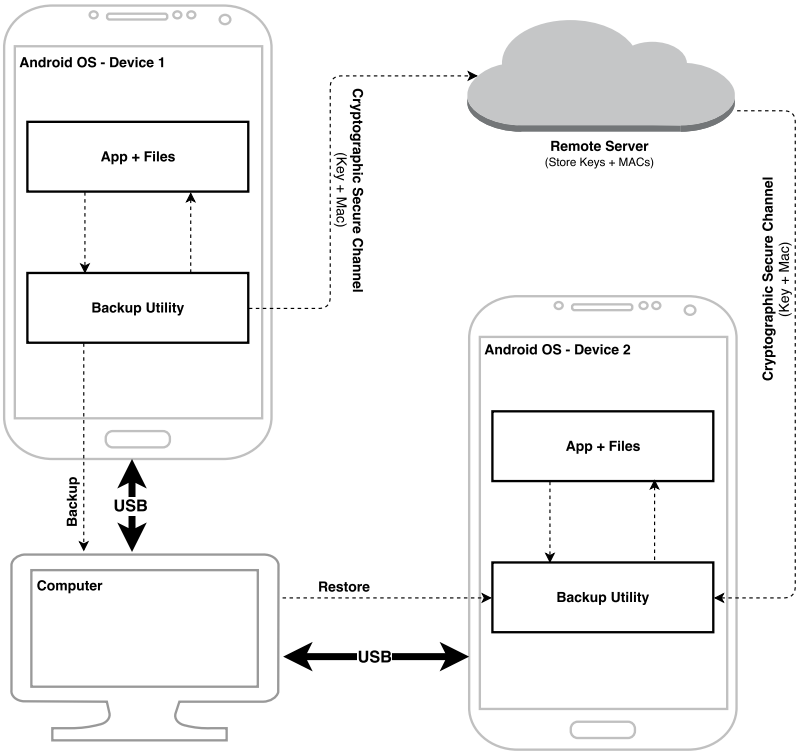


Figure 5.3: Solution #2 — Adding Internet connectivity when backing up applications and storing integrity codes and keys remotely.

In this approach, the ability to communicate with the Internet is added as a module to the Backup

utility also, providing that it already contains the necessary programming logic to generate and verify the integrity codes of the backups. The utility asks the server for the integrity key (e.g., private key from a public-key cryptosystem) when the backup procedure starts. It then generates the archive and the integrity code with indexing information. The code is then sent to the remote server and the key is safely erased from the internal memory of the device. The archive is stored at the user chosen location. When restoring, the utility receives the archive and asks the server for the respective key (in the case of a public-key cryptosystem, the public key may already be in the device) and integrity code. The process should only be successful if the integrity is verified.

The scheme highlights that the integrity keys and codes need to be transmitted using a cryptographically secure channel, which adds overhead to the whole process. Ideally, when using this scheme, the secret keys (e.g., symmetric key for MAC or the private key of a digital signature scheme) would only be transfered to the device when required, instead of being stored permanently there. This would diminish their exposure and the risk of being illegitimately obtained. To successfully modify an application, an attacker would have to either (i) compromise the remote server, (ii) produce a collision, or compromise the keys. The best moment to compromise the keys would be when they are ate the device for backing up or restoring activities.

This solution would solve the problem of transferring applications to different devices. Nonetheless, in order for it to function properly, it requires always Internet connectivity.

5.2.4 Approach #3 — Software Modification to Backup Utility and a TCB

The ARM processor architecture is the most used technology in smartphones and tablets. In 2009, those responsible for this architecture implemented the ARM TrustZone, which is a hardware security technology incorporated into recent ARM processors since ARMv6, including ARM Cortex A8, A9 and A15 processors. It provides a security extension named ARM System-on-Chip (SoC) [Fur00], that enables having several systems allocated in only one chip.

ARM SoC is based on the idea of two execution worlds, as shown in Figure 5.4. One world is called the *Normal World*, or Rich Execution Environment (REE), while the other is called the *Secure World*, or Trusted Execution Environment (TEE). These two worlds are separated by hardware mechanisms, and have different levels of privileges. This division ensures that instructions occurring in the normal world do not have access to the secure world, yet the *secure world* has access to the *normal world* providing some conditions are met. The system uses processor bit, kown as Non-Secure (NS)-bit to decided the world where code will be executed.

The approach described in this section leverages the previously presented technology to address the problem of having the keys or the backup utility compromised. Its high level scheme is included in Figure 5.5 and its functioning is similar to the one presented in Section 5.2.2. In this case, the backup utility continues to run in the normal world, but the routine to calculate or verify the integrity code runs in the secure world. Since the approach assumes that the integrity code generalization and check is mandatory, it is also assumed that the integrity of the backup utility is verified, so that the mechanism is not circumvented. It is also assumed that the integrity keys can only be accessed from the secure world.

This solution is more secure in the sense that the keys are more difficult to compromise. This

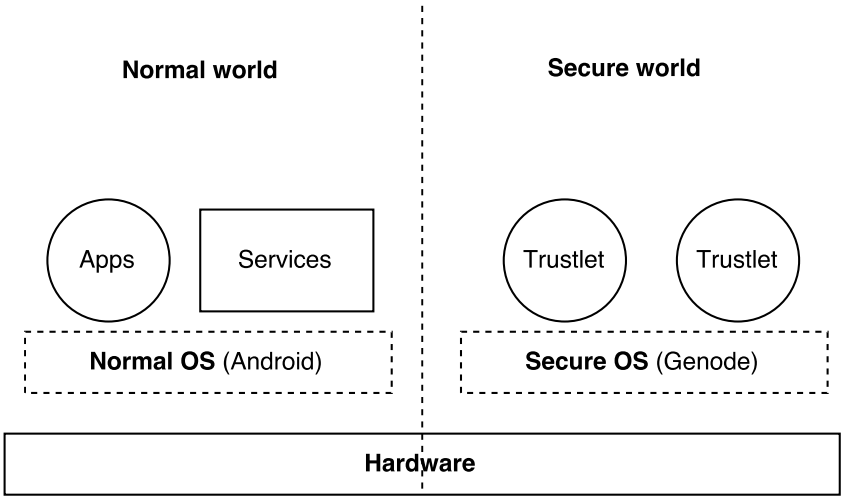


Figure 5.4: High level diagram showing the difference between the traditional and the ARM SoC architectures.

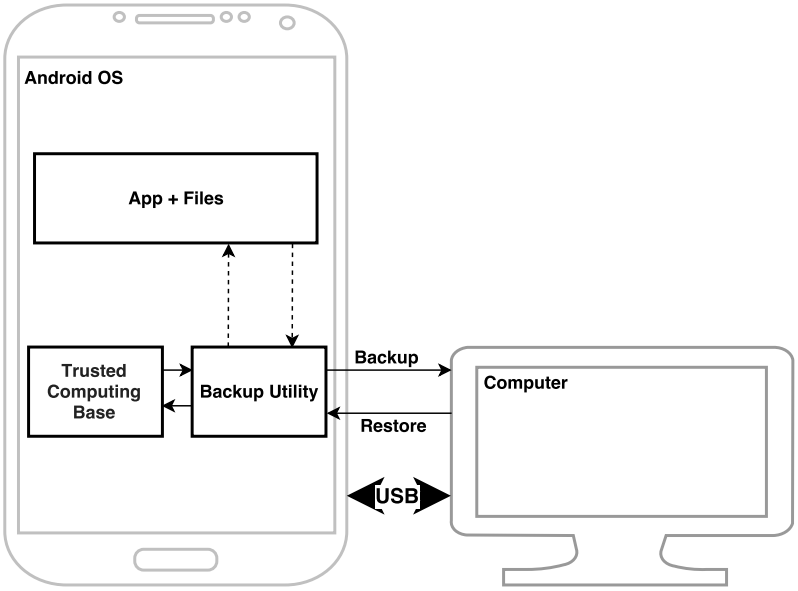


Figure 5.5: Solution #3 — Backup utility interacts with a *trustlet* for the purpose of generating and verifying integrity codes.

approach does not require access to the Internet, which brings the portability problem into focus again. In this case, this problem can be translated into a key distribution problem. If it were to happen, transmission of keys for different devices would have to be done live, using trustlets running in secure worlds between the two devices. Otherwise (offline), the keys would have to be stored out of the device at some point, even if protected, which could lead to their compromise.

5.2.5 Approach #4 — Software Modification to the Backup Utility and a TCB with Networking and Server for Storing MACs or Digital Signatures

The last approach presented herein combines the two previous ones and the high level scheme of its functioning is depicted in Figure 5.6. In this case, the backup utility interacts with a trustlet, which runs in a secure world and is responsible for communicating also with a remote server for storing integrity keys and codes. The trustlet assures that the cryptographically secure channel is established, functioning as a wormhole.

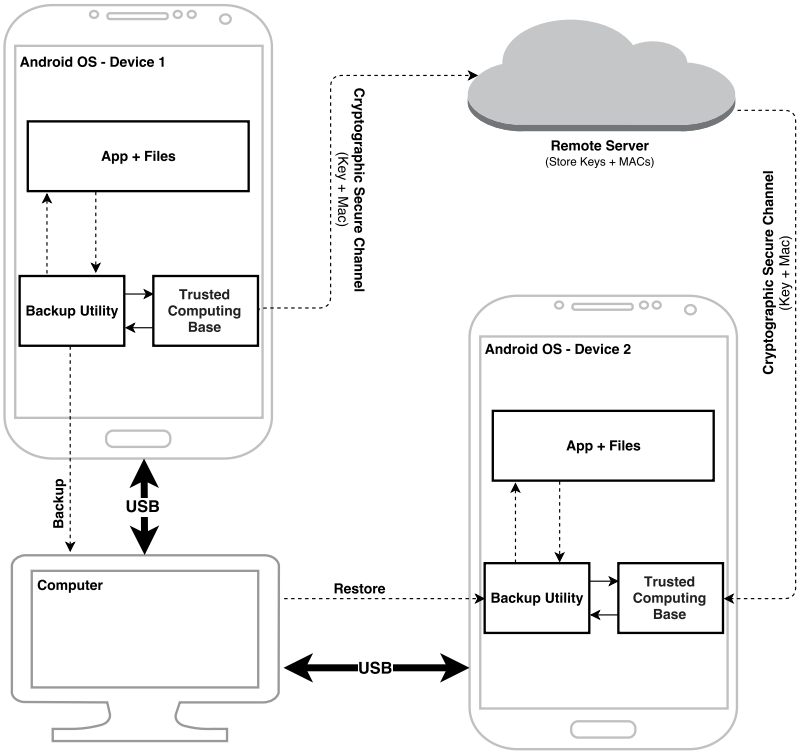


Figure 5.6: Solution #4 — Backup utility interacts with a *trustlet* for the purpose of generating and verifying integrity codes and uses a remote server to store integrity codes and keys.

The major drawback of this approach is that it assumes Internet connectivity. This assumption may be relaxed for the moment in which a backup is being performed, since the integrity code and keys may be momentarily stored locally, and sent to the remote server afterwards. Restoration will not be possible before that data is updated to the remote server. It is assumed that the Internet connectivity is available during the restoring procedure, as the application should only be installed back on the device if the validity is checked. The assumption of having Internet connectivity may hinder the usage of the utility, since some users may not have it available when needed.

5.3 Usage of Encryption Primitives

The previous solutions do not address the information exposure nor solve the problem of data manipulation in rooted devices. The later problem is the most difficult to handle, because with such administrative privileges, the application does not need to leave the primary environment to be analyzed or modified.

While the description of the previous approaches was focused on the integrity problem, it could be easily extended to the confidentiality issue to. The author opted to leave it like that for the sake of the explanation. Integrating appropriate encryption and decryption mechanisms in the backup utility or in a trustlet, after the backup and before restoring, respectively, would effectively solve the data exposure problem, providing that the decryption keys could not be compromised. The four approaches would have to assume that encryption keys were generated (or pre-distributed) for the backup process to proceed. In the case of having a remote server and Internet connection involved, the discussion would mention that those keys would have to be sent and retrieved securely too.

Addressing the possibility of the device to be rooted could be done, at least to some extent, via integration of cryptographic primitives in the mobile applications. This would require developers to know the mechanisms and how to use them. Unfortunately, depending on how the mechanisms are integrated and the available resources (e.g., the existence of a secure world in the device), this process may either result in a secure application or in an added difficulty for attacker.

In terms of software resources, and since applications for the Android OS are developed in `JAVA` programming language, developers may resort to the classes of the Java Cryptography Architecture (JCA), which is very complete and documented. It includes the implementation of primitives such as Advanced Encryption Standard (AES) and RSA (via the `Cipher` class), and Secure Hash Algorithm 256 (SHA256) (via the `Digest` class), amongst many others. In terms of storage in databases, and since SQLite is the native engine provided with Android, developers have many options to choose from too, namely:

- **SQLite Encryption Extension (SEE)**, an official extension to SQLite, created by its developers, which offers support for four different ciphers, namely, Rivest Cipher 4 (RC4), AES-128 in Output Feedback (OFB) mode, AES-128 in Counter with CBC-MAC (CCM) mode and AES-256 in OFB mode;

- **wxSQLite3** [Ulr15], which is a wrapper for the SQLite3 developed in C++ programming language. It is designed to be used in programs based on the `wxWidgets` library. Since version 1.9.8 that supports the AES-256 encryption algorithm;

- **SQLCipher** [Zet15], an open source library that was developed by Zetetic LLC. It had its first release in November 2008, being an update published in 2011 with support for Android OS. This library makes available the secure and transparent 256-bit AES encryption of SQLite database files, meaning that the library acts as a middleware, and all instructions and code are implemented as before;

- **SQLiteCrypt** [Han15], a small library implemented in C programming language, which also handles the interactions with the database for the application in a transparent manner. The data is secured using the AES-256 encryption algorithm;

- **BotanSqlite3** [ran15], a module to be used with SQLite3, encrypts the entire database using algorithms supported by Botan; and

- **SQLiteCrypto** [and15a], which was developed by andbrain.com with the purpose of providing means to encrypt SQLite databases transparently to the end user.

The SEE, SQLiteCrypt and SQLiteCrypto require the purchase of a license.

5.4 Conclusions

The discussion in this chapter leads to the conclusion that the problem addressed in this master's program is not easy to solve. On the one hand, the backup utility seems like a legitimate feature to provide to users; on the other hand, since all the data is local and in (physical) possession of the owner of the device, it may be difficult to secure. Solutions based on a safe processing environment comprise the best ones and adding a trusted remote server for storing keys and integrity codes (and potentially the backups) solve the problem of portability and key management. Actually, it was previously shown that mobile applications whose state is store remotely are not easily affected by data manipulation attempts. More recently, Google has also started offering online backups of applications.

Dealing with the possibility of having the device rooted requires further measures. Integrating security mechanisms in their applications, namely encryption and integrity related primitives for the data stored in the device, will increase the amount of work an attacker would have to employ to modify or obtain data, if not rendering it impossible. Additionally, the attacker would have to potentially study each application with more detail, instead of using a more generic method such as the one used in the scope of this work. There are several resources readily available resources for accomplishing that, some of them with little development overhead.

Chapter 6

Conclusions and Future Work

This chapter is divided into two sections. The first section presents the main conclusions of the work done during this master's program, while the second section contains suggestions for future work.

6.1 Main Conclusions

The number of mobile users has been increasing significantly, along with the number of mobile devices, platforms and applications. The number of security issues in the mobile world has also been following this trend, justifying the research on this topic. Chapter 2 discusses a small part of that research, an highlights that one of the subtopics receiving more attention is the one concerning malware. The pertinence is such that OWASP, previously known for the work on Web security, has created a community to discuss issues related to security on mobile devices. Similarly to what has done for Web applications, OWASP has created a list of the top 10 security flaws for mobile applications. *Weak server side controls*, which refers to flawed APIs at the servers with which the application communicates with, is at the top of the list, immediately followed by the *Insecure Data Storage* flaw, which is the general category of security problems in which this work is focused in. It was discussed that there are several papers published in this topic, but most of them are based in the assumption that the device is rooted. The closest work to the one presented here was focused on assessing how many applications of Google Play were accepting backups, with results from an automated script showing that up to 94% of the software in that store was allowing it.

This work required building two datasets, presented in chapter 3. One of datasets consisted only of games and the other one of common mobile applications, all publicized as having the in-app purchases functionality. A total of 1542 games or applications were analyzed. In the games dataset, the category with more samples was the one of Action games, with 141 samples. In terms of download range, the most represented interval was the one delimited by 1000000 and 5000000, with 296 games. As for the common applications dataset, it was necessary to first define a maximum number of downloads for each category available in Google Play. On the one hand, applications from all categories were necessary, one the other, this universe is larger than the one of games. The maximum was set to 30 and, from the different 24 categories in Google Play, only 3 did not have the amount of required applications. In this dataset, the download interval congregating the largest amount of applications was the one going from 100000 to 500000. The initial phase of this work was human made, but later it was semi-automated via the implementation of programs in JAVA. The author believes that the amount of applications used in this study is sufficient to have confidence in the final results, which fulfills one of the objectives of this master's program. The method used to analyze each one of the applications or games, also described in the same chapter, had three major phases, none of which comprising the requirement to root devices or installing closed-source or paid software. The method can

be performed with minimal computer skills, which also favors the feasibility of the approach and of this study.

In the case of the Android games with in-app purchases, the results of the assessment show that approximately 17.5% of the entire dataset was susceptible to data manipulation, corresponding to 148 matches in 849 games or, in other words, 1 in each 6 games was found vulnerable. Amongst other details, the approach enabled changing levels or cheat in terms of number of virtual coins, items or premium features. These numbers are conservative, because the method was not that specialized for each one of the applications. The author believes that a motivated attacker will achieve even better results after delving deeper into the functioning of a target application and getting to know better its files. The same analysis showed that XML files were the preferred type for storing data, followed by text files and SQLite databases, in this order. In the case of games, this problem is not so harmful to users as it is for developers, since the micro-payments may be part of their income. The security problem seems to be transversal to all categories and download intervals.

The analysis of the second dataset was focused on assessing susceptibility to data exposure too (apart from data manipulation). The initial screening of the applications led to shrinking the size of the dataset to the most interesting ones (e.g., the ones in which the in-app purchases were only used to remove advertisements were not considered). The final dataset had 377 applications, and it was found that at least 1 in every 5 were susceptible to at least one of the aforementioned problems. These results were conservative by the same reasons mentioned before. One of the most interesting findings was that, from the 87 applications containing login screens (or similar login functionality), 23 were storing credentials (e.g., passwords and PINs) in plaintext. In some cases, it was possible to reset credentials by deleting files. The security problems in these applications are damaging to both parties. The developer looses money and credibility (some of the applications may be used for financial transactions); and the user may have its credentials compromised. For example, there may be desktop OS malware whose purpose is to scan for connected Android devices, and try to backup the application to read its contents. The malware may pack all the necessary tools, including ADB.

Solving the problems studied in the scope of this work is not easy. In the opinion of the author of this dissertation, removing the backup utility (or the possibility to perform the backup of applications) is not a solution because the main functionality provided is legitimate and useful in some situations. It would not solve the information exfiltration problem in rooted devices either. Chapter 5 contains a discussion on some possible solutions to the data manipulation and exposure problem when the backup utility is used. All solutions presume that the backup utility is modified so as to include mandatory (and potentially transparent) integrity and confidentiality mechanisms. The usage of a remote server combined with secure processing devices is the best solution. Either way, securing data and applications should be performed via the integration of security into the applications, which requires developers to learn more about information and system security.

Lastly, it can be concluded that all of the objectives defined for this master's program were achieved.

6.2 Directions for Future Work

One of the most direct future lines of work includes studying the susceptibility to data manipulation and exposure for other mobile OSs, namely for iOS. This work will certainly require assessing if a method similar to the one used in the scope of this work can be used to analyze iOS applications and games. It would be interesting to see if the iOS or Windows Mobile OSs were more careful, in terms of integration of security mechanisms, or if the OSs have additional protections to minimize the problems discussed herein.

Expanding the datasets comprises another line of work. It would be interesting to also test free applications without in-app purchases, to see if the financial gain motivates the developers to construct more secure applications or not. Additionally, and this is more of a wish because of its nature, it would be good to also analyze paid applications and games.

The automation of the method described in this work will also be a subject of research in the future. The backup utility was included in most recent version of the Android OS at the time of writing this dissertation and it is foreseen that this tool will not be removed in the future, due to its primary purpose. As such, it is pertinent to search for ways to automate the method. Nonetheless, this task is hard because different developers may use different methods to store data and even small changes may compromise the procedure. For example, it was noticed that similar virtual values were stored using different names in databases and files (e.g., coins can be stored in variables or columns named `coins`, `points` or `money`). Achieving a fully automated procedure would perhaps enable the implementation of an analysis tools that developers could use to test their application before submitting it to Google Play.

The comparison of these results with the ones of tools designed for the Insecure Data Storage of OWASP has been also considered along the way. The idea is to see if the available vulnerability assessment tools identify the same applications that were vulnerable to data manipulation according to the procedure used in this work.

Last but not least, the implementation of prototypes for some of the solutions outlined in Chapter 5 would be an important line of future work. With this idea in mind, the source code of the backup utility was already studied superficially in the final phase of the master's program, though the modification (solution #1) was not produced yet.

Bibliography

[and15a] andbarin. SQLiteCrypto 1.0.1 full Code Source API | andbrain.com [online]. 2015. Available from: www.andbrain.com/product/sqlitecrypto/ [cited 8 September 2015]. 40

[And15b] Android Developers. Android Debug Bridge | Android Developers [online]. 2015. Available from: http://developer.android.com/tools/help/adb.html [cited 23 September 2015]. 12, 19

[And15c] Android Developers. App Manifest | Android Developers [online]. 2015. Available from: http://developer.android.com/guide/topics/manifest/manifest-intro.html [cited 23 September 2015]. 8

[And15d] Android Developers. Dashboards | Android Developers [online]. 2015. Available from: http://developer.android.com/tools/publishing/publishing_overview.html [cited 04 August 2015]. 2

[And15e] Android Developers. Dashboards | Android Developers [online]. 2015. Available from: http://developer.android.com/distribute/googleplay/start.html [cited 04 August 2015]. 2

[And15f] Android Developers. Dashboards | Android Developers [online]. 2015. Available from: https://developer.android.com/about/dashboards/index.html [cited 13 April 2015]. 18

[And15g] Android Developers. Security Tips | Android Developers [online]. 2015. Available from: http://developer.android.com/training/articles/security-tips.html [cited 23 September 2015]. 11

[App15a] AppBrain. Number of available Android applications - AppBrain [online]. 2015. Available from: www.appbrain.com/stats/number-of-android-apps [cited 13 April 2015]. 2

[App15b] Apple. Official apple store [online]. 2015. Available from: http://store.apple.com/us [cited 13 April 2015]. ix, 1

[App15c] Apple Inc. App Review - App Store - Apple Developer [online]. 2015. Available from: https://developer.apple.com/app-store/review/ [cited 04 August 2015]. 2

[App15d] Apple Inc. iOS 9 - Apple [online]. 2015. Available from: http://www.apple.com/ios/ [cited 23 September 2015]. ix, 1

[App15e] Apple Inc. iPhone - Apple [online]. 2015. Available from: www.apple.com/iphone/ [cited 23 September 2015]. ix, 1

[BKvOS10] David Barrera, H. Güneş Kayacik, Paul C. van Oorschot, and Anil Somayaji. A methodology for empirical analysis of permission-based security models and its application to android. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 73--84, New York, NY, USA, 2010. ACM. Available from: http://doi.acm.org/10.1145/1866307.1866317. 16

[BZNT11] Iker Burguera, Urko Zurutuza, and Simin Nadjm-Tehrani. Crowdroid: Behavior-based malware detection system for android. In *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '11, pages 15--26, New York, NY, USA, 2011. ACM. Available from: http://doi.acm.org/10.1145/2046614.2046619. xi, 8

[Chr13] Christian Håland. An Application Security Assessment of Popular Free Android Applications. Master's thesis, Norwegian University of Science and Technology, 2013. xi, 10, 11

[Cla15] Claud Xiao and Ryan Olson. Insecure Internal Storage in Android - Palo Alto Networks BlogPalo Alto Networks Blog [online]. 2015. Available from: http://researchcenter.paloaltonetworks.com/2014/08/insecure-internal-storage-android/ [cited 23 September 2015]. 12

[EOM09] William Enck, Machigar Ongtang, and Patrick McDaniel. On lightweight mobile phone application certification. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 235--245, New York, NY, USA, 2009. ACM. Available from: http://doi.acm.org/10.1145/1653662.1653691. 16

[FBL+15] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M.S. Gaur, M. Conti, and M. Rajarajan. Android security: A survey of issues, malware penetration, and defenses. *Communications Surveys Tutorials, IEEE*, 17(2):998--1022, Secondquarter 2015. ix, 1

[FCH+11] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. Android permissions demystified. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 627--638, New York, NY, USA, 2011. ACM. Available from: http://doi.acm.org/10.1145/2046707.2046779. 16

[Fel15] Felix Matenaar, Patrick Schulz, Mark Schloesser, Andreas Galauner. dexter [online]. 2015. Available from: http://dexter.dexlabs.org/ [cited 23 September 2015]. xi, 8

[FGW11] Adrienne Porter Felt, Kate Greenwood, and David Wagner. The effectiveness of application permissions. In *Proceedings of the 2Nd USENIX Conference on Web Application Development*, WebApps'11, pages 7--7, Berkeley, CA, USA, 2011. USENIX Association. Available from: http://dl.acm.org/citation.cfm?id=2002168.2002175. 16

[FHM+12] Sascha Fahl, Marian Harbach, Thomas Muders, Lars Baumgärtner, Bernd Freisleben, and Matthew Smith. Why eve and mallory love android: An analysis of android ssl

(in)security. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 50--61, New York, NY, USA, 2012. ACM. Available from: http://doi.acm.org/10.1145/2382196.2382205. 10

[Fra15] Francisco Vigário, Miguel Neto, Musa G. Samaila, Mário M. Freire and Pedro R. M. Inácio. On the susceptibility to data manipulation and information exposure of free android apps with in-app purchases. In Inforum 2015, editor, *Atas do Inforum 2015*, Inforum 2015. INFORUM, 09 2015. Available from: http://inforum.org.pt/INForum2015/programa. xi, 4

[Fur00] Steve Furber. *ARM System-on-Chip Architecture*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2000. 36

[Goo15a] Google. Android [online]. 2015. Available from: https://www.android.com/ [cited 23 September 2015]. ix, 1

[Goo15b] Google. Google Play [online]. 2015. Available from: https://play.google.com/store [cited 13 April 2015]. ix, 1, 15

[Goo15c] Google. Nexus – Google [online]. 2015. Available from: www.google.pt/nexus/ [cited 23 September 2015]. ix, 1

[Goo15d] Google Play. WiFi ADB - Debug Over Air - Android Apps on Google Play [online]. 2015. Available from: https://play.google.com/store/apps/details?id=com.ttxapps.wifiadb&hl=en [cited 14 September 2015]. 33

[Han15] Hanoi. Transparent SQLite database encryption [online]. 2015. Available from: http://sqlite-crypt.com/ [cited 8 September 2015]. 39

[HDL13] Xiali Hei, Xiaojiang Du, and Shan Lin. Two vulnerabilities in android OS kernel. In *Proceedings of IEEE International Conference on Communications, ICC 2013, Budapest, Hungary, June 9-13, 2013*, pages 6123--6127, 2013. Available from: http://dx.doi.org/10.1109/ICC.2013.6655583. 7

[HLKR15] Sungjae Hwang, Sungho Lee, Yongdae Kim, and Sukyoung Ryu. Bittersweet adb: Attacks and defenses. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '15, pages 579--584, New York, NY, USA, 2015. ACM. Available from: http://doi.acm.org/10.1145/2714576.2714638. 13

[Jam14] James King. Android Application Security with OWASP Mobile Top 10 2014. Master's thesis, Luleå University of Technology, 2014. 10, 12

[Joe15] Joe Security LLC. Automated Malware Analysis - APK Analyzer [online]. 2015. Available from: www.apk-analyzer.net [cited 23 September 2015]. xi, 8

[MBK+12] Ildar Muslukhov, Yazan Boshmaf, Cynthia Kuo, Jonathan Lester, and Konstantin Beznosov. Understanding users' requirements for data protection in smartphones.

In *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering Workshops*, ICDEW '12, pages 228--235, Washington, DC, USA, 2012. IEEE Computer Society. Available from: http://dx.doi.org/10.1109/ICDEW.2012.83. ix, 1

[Mic15] Microsoft. Windows Phone Apps+Games Store [online]. 2015. Available from: www.windowsphone.com [cited 13 April 2015]. ix, 1

[OWA15] OWASP. Projects/OWASP Mobile Security Project - Top Ten Mobile Risks - OWASP [online]. 2015. Available from: https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Top_Ten_Mobile_Risks [cited 23 September 2015]. xxi, 9

[ran15] randombit. Botan: Crypto and TLS for C++11 - Botan [online]. 2015. Available from: http://botan.randombit.net/ [cited 8 September 2015]. 39

[RS13] Shigeki Goto Ryo Sato, Daiki Chiba. Detecting android malware by analyzing manifest files. In *Proceedings of the Asia-Pacific Advanced Network 2013 v. 36, p. 23-31*, 2013. Available from: http://dx.doi.org/10.7125/APAN.36.4. xi, 8

[Ulr15] Ulrich Telle. wxSQLite3: wxSQLite3 [online]. 2015. Available from: http://wxcode.sourceforge.net/docs/wxsqlite3/index.html [cited 8 September 2015]. 39

[VGN14] Nicolas Viennot, Edward Garcia, and Jason Nieh. A measurement study of google play. In *The 2014 ACM International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '14, pages 221--233, New York, NY, USA, 2014. ACM. Available from: http://doi.acm.org/10.1145/2591971.2592003. ix, 2

[Vie15] Vienna University of Technology, Secure Systems Lab / Computer Security Group at UCSB. Anubis - Malware Analysis for Unknown Binaries [online]. 2015. Available from: https://anubis.iseclab.org [cited 23 September 2015]. xi, 8

[VNF+15] Francisco Vigário, Miguel Neto, Diogo Fonseca, Mário M. Freire, and Pedro R. M. Inácio. Assessment of the susceptibility to data manipulation of android games with in-app purchases. In *ICT Systems Security and Privacy Protection - 30th IFIP TC 11 International Conference, SEC 2015, Hamburg, Germany, May 26-28, 2015, Proceedings*, pages 528--541, 2015. Available from: http://dx.doi.org/10.1007/978-3-319-18467-8_35. xi, 4

[Weg15] Wegilant. Appvigil - Cloud based Android App Security Scanner [online]. 2015. Available from: https://appvigil.co [cited 23 September 2015]. xi, 8

[Whi15] White Cheats. My Talking Tom Cheats - Coins Hack Android and IOS [online]. 2015. Available from: whitecheats.com/my-talking-tom-cheats/ [cited 25 September 2015]. 3

[WMW+12] Dong-Jie Wu, Ching-Hao Mao, Te-En Wei, Hahn-Ming Lee, and Kuo-Ping Wu. Droidmat: Android malware detection through manifest and api calls tracing. In *Proceedings of the 2012 Seventh Asia Joint Conference on Information Security*, ASIAJCIS

'12, pages 62--69, Washington, DC, USA, 2012. IEEE Computer Society. Available from: http://dx.doi.org/10.1109/AsiaJCIS.2012.18. 8

[XDA15] XDA Forums. [GUIDE] How to extract, create or edit android adb backups | Android Development and Hacking | XDA Forums [online]. 2015. Available from: http://forum.xda-developers.com/showthread.php?t=2011811 [cited 27 August 2015]. x, 3, 18

[Zet15] Zetetic. SQLCipher - Zetetic [online]. 2015. Available from: https://www.zetetic.net/sqlcipher/ [cited 8 September 2015]. xv, 39