FERRAMENTAS OPENSOURCE PARA AVALIAÇÃO DE SEGURANÇA EM APLICATIVOS DESENVOLVIDOS E EXECUTADOS NO SISTEMA OPERACIONAL ANDROID[1]

Diego de Souza Maia

Resumo: Dispositivos móveis e seus aplicativos estão cada vez mais proporcionando conveniência para a população. No entanto, o seu uso tem despertado interesse de pessoas mal-intencionadas que visam obter informações dos usuários de dispositivos móveis. O objetivo deste trabalho é encontrar ferramentas *opensource* que possam ser utilizadas para identificar de forma simples e com o menor custo possível se os aplicativos disponibilizados e executados no sistema operacional Android possuem as preocupações de segurança necessárias para proteção das informações pessoais de seus usuários. Foram realizadas análises bibliográficas em livros, artigos, periódicos e sites que permitiram conhecer o que já foi estudado até o momento sobre o tema. Como resultado, obteve-se um relatório técnico contendo as ferramentas *opensource* encontradas, características principais e situações de uso. Dentre os resultados, destacam-se as distribuições para testes de segurança em aplicativos móveis, que reúnem uma grande quantidade de ferramentas para análise de segurança em um único local.

Palavras-chave: Análise de Segurança. Android. Aplicativos. Opensource.

1 INTRODUÇÃO

A evolução dos dispositivos móveis como *smartphones* e *tablets*, juntamente com a expansão das tecnologias de internet móvel, proporcionam cada vez mais conveniência, facilidade e praticidade para à população. No entanto, como identificar de forma simples e com o menor custo possível se os aplicativos disponibilizados para o sistema operacional Android possuem as preocupações de segurança necessárias para proteção das informações pessoais de seus usuários?

O objetivo da pesquisa é encontrar ferramentas *opensource* que possam ser utilizadas para identificar de forma simples e com o menor custo possível se os aplicativos disponibilizados e executados no sistema operacional Android possuem as preocupações de segurança necessárias para proteção das informações pessoais de seus usuários. É

[1] Artigo apresentado como Trabalho de Conclusão do Curso de Especialização em Gestão de Segurança da Informação, da Universidade do Sul de Santa Catarina, como requisito parcial para a obtenção do título de Especialista em Gestão da Segurança da Informação.

importante salientar que o trabalho não tem como objetivo testar a efetividade das ferramentas econtradas e sua compatibilidade com as diferentes versões existentes do Android.

Para que os objetivos fossem atingidos e a resposta para o problema da pesquisa fosse encontrada, foram realizadas análises bibliográfica em livros, artigos, periódicos e sites que permitiram conhecer o que já foi estudado até o momento sobre o tema. Também foram analisados estudos de caso e outros trabalhos que realizaram pesquisas semelhantes com o tema em questão. Como resultado, obteve-se um relatório técnico contendo as ferramentas *opensource* encontradas, suas características principais e situações de uso.

O resultado desse trabalho irá contribuir para futuras análises em aplicativos, de modo que elas possam ser realizadas de forma simples, com o menor custo possível e de acordo com os requisitos mínimos de segurança necessários para proteção das informações pessoais de seus usuários.

O artigo está dividido em seis seções. A segunda seção apresenta os conceitos necessários para compreensão do tema e alcance dos objetivos do projeto, nela são descritas informações sobre o sistema operacional Android, sua arquitetura e controles de segurança. A terceira seção apresenta as principais áreas na segurança de aplicativos móveis e tipos de análise de segurança. Na quarta seção são apresentadas ferramentas *opensource* para a realização de testes de segurança em aplicativos Android. Já a quinta seção é composta pela análise dos dados coletados contendo as ferramentas *opensource* encontradas. Por fim, a sexta e última seção relata as considerações e conclusões do autor.

2 SISTEMA OPERACIONAL ANDROID

A primeira versão do sistema operacional Android foi lançada em setembro de 2008, três anos após ter sido comprado pela Google. Já em meados de 2011 dispositivos móveis utilizando o sistema operacional Android já totalizavam 50% do mercado, tornando-se em 2012 uma das plataformas mais utilizados no mercado de dispositivos móveis (GUNASEKERA, 2012).

Conforme pesquisas realizadas pelo International Data Corporation, IDC (2017), divulgadas em maio do ano passado, o sistema operacional Android ainda lidera o

mercado global de dispositivos móveis com aproximadamente 85% de participação em relação aos demais sistemas.

2.1. Arquitetura

Arquitetura do sistema operacional Android era inicialmente dividida em: Kernel, bibliotecas, máquina virtual Dalvik, estrutura do sistema e aplicativos. O Kernel era responsável pela parte de interação entre os aplicativos e o hardware do dispositivo, as bibliotecas responsáveis pela tradução dos comandos trocados entre o Kernel e a estrutura do sistema promovendo assim a comunicação entre elas, já a máquina virtual Dalvik era responsável por executar aplicativos desenvolvidos na linguagem Java. Por fim, a estrutura do sistema fornecia componentes, serviços e interfaces para serem utilizados pelos aplicativos (agenda, leitor de e-mail, navegador de internet) que por sua vez são a camada mais visível do sistema ao usuário (GUNASEKERA, 2012; CHELL et al., 2015).

A partir da versão 4.4 do sistema Android, a Google disponibilizou de forma experimental, uma nova ferramenta para execução de aplicativos dentro do sistema chamada de Android Run-Time ou simplesmente ART, projetado para melhorar o desempenho dos aplicativos e reduzir o consumo de bateria dos dispositivos, substituindo definitivamente a máquina virtual Dalvik a partir da versão 5.0 do Android (CHELL et al., 2015).

Quando um usuário instala algum aplicativo, os dados são armazenados dentro do diretório */data/app/*, já os atributos do *app*, como por exemplo as permissões de acesso a recursos do sistema, estarão catalogados nos arquivos */data/system/packages.xml* e */data/system/packages.list*. Os aplicativos que fazem parte do sistema ficam armazenados dentro do diretório */system/app/* (CHELL et al., 2015).

2.2. Controles de Segurança

Segundo Gunasekera (2012) a separação de privilégios, o gerenciamento de permissões e a assinatura de código dos aplicativos são alguns dos controles de segurança implementados pelo sistema operacional Android para proteger os aplicativos de ataques originados por outros aplicativos instalados no mesmo dispositivo. O arquivo *AndroidManifest.xml*, presente em todos os aplicativos desenvolvidos para o sistema Android, implementa uma desses controles de segurança. Uma das funções do arquivo é

definir quais sãos as permissões de segurança necessárias para o aplicativo interagir com outros aplicativos e com o sistema operacional (CHO et al., 2013).

Conhecido como Sandbox, a separação de privilégios exige, por exemplo, que cada aplicativo seja executado somente com sua identificação de usuário e grupo, logo nenhum aplicativo poderá executar ou obter informações de outros aplicativos a não ser que essa permissão seja solicitada de forma explícita e visível durante sua instalação ou inicialização, caracterizando assim o gerenciamento de permissões. Já a assinatura digital é requisito obrigatório para qualquer aplicativo desenvolvido para o sistema Android, pois é através dela que o sistema operacional irá estabelecer uma relação de confiança para conceder acessos a recursos do sistema (GUNASEKERA, 2012).

De acordo com Chell et al. (2015), o sistema operacional Android, desde a versão 3.0, dispõe de uma ferramenta chamada Full disk encryption (FDE) desenvolvida para criptografar unidades, pastas ou arquivos armazenados dentro do dispositivo. Quando ativado, o FDE irá solicitar a senha do usuário na inicialização do dispositivo e utilizará essa senha para criptografar e descriptografar os arquivos armazenados na partição */data*, adicionando uma camada de proteção às informações do usuário e das aplicações armazenadas nessa partição. Rai (2013) complementa, afirmando que a plataforma Android fornece ferramentas, chamadas de Application Programming Interface ou API, para que os aplicativos possam criptografar e descriptografar informações.

A ofuscação do código-fonte é uma técnica realizada com o intuito de dificultar a interpretação de classes, campos, métodos, comandos ou configurações do aplicativo durante um ataque de engenharia reversa. Essa técnica não evita que o atacante decifre as informações presentes no binário de um aplicativo, mas ajuda a dificultar e retardar este processo (POPA, 2011;GUNASEKERA, 2012; SCAMBRAY; ROUSE; STANFIELD, 2013).

Como principais técnicas para ofuscação do código-fonte podem ser citadas a técnica de ofuscação de nomes, que altera os identificadores de classes, métodos e variáveis dentro do código-fonte, substituindo os caracteres originais por outros sem sentido para dificultar a compreensão. Essa técnica pode diminuir o tamanho do aplicativo, sendo vantajosa para utilização em aplicações móveis. Já a técnica de ofuscação dos dados, tem como objetivo trocar as variáveis de local, alterando a ordem

das informações dentro de uma estrutura de dados através da substituição de um valor por uma expressão. Assim, a ofuscação de dados acaba alterando a forma com que as informações ficam armazenadas na memória do dispositivo (POPA, 2011).

Ainda em relação a técnicas de ofuscação, o ofuscamento da estrutura do código é utilizado para alterar o controle de fluxo da aplicação, mudando a ordem em que as instruções dentro do programa são executadas. O ofuscamento da estrutura também pode adicionar ao programa um código falso, para dificultar a compreensão do fluxo realizado pelas informações dentro do código original. A técnica de ofuscamento das informações de depuração, que visa ocultar nomes de arquivos de log e consequentemente os dados armazenados dentro destes arquivos. Por fim, a técnica de inserção de marcas d'água, adiciona informações ao código-fonte, como uma espécie de carimbo, sem alterar os dados que ali estão (POPA, 2011).

3 PRINCIPAIS ÁREAS NA SEGURANÇA DE APLICATIVOS MÓVEIS

De acordo com OWASP (2018) apesar dos sistemas operacionais tornarem-se cada vez mais seguros, a falta de preocupação com a segurança durante o desenvolvimento de um aplicativo para dispositivos móveis pode colocar em risco o armazenamento e a transmissão de informações de seus usuários.

3.1. Armazenamento local

O armazenamento de informações confidenciais de maneira inadequada pode expor estes dados de forma indevida a outros aplicativos em execução no mesmo dispositivo, em cópias de segurança, armazenamento em nuvem. Também deve-se levar em conta o roubo ou perda do dispositivo móvel, o que facilitaria o acesso físico ao dispositivo e a recuperação das informações. Por isso, qualquer informação confidencial do usuário deve ser armazenada de forma segura, utilizado métodos de criptografia ou outro recurso de segurança adequando, OWASP (2018).

3.2. Comunicação

Assim como no armazenamento local, qualquer informação transmitida entre um aplicativo instalado no dispositivo móvel e um serviço remoto, deve ser enviada

utilizando um canal criptografado. Tal medida visa garantir a confidencialidade e a integridade das informações trocadas entre as partes, OWASP (2018).

3.3. Autenticação e autorização

De acordo com OWASP (2018), o processo de envio das credenciais do usuário para posterior autorização de acesso em um sistema é uma tarefa realizada frequentemente por aplicações móveis. Apesar da autenticação ser realizada no lado do servidor, o dispositivo pode armazenar informações utilizadas durante a sessão ou até mesmo as próprias credenciais de acesso do usuário.

3.4. Código-fonte e Engenharia reversa

As técnicas de engenharia reversa permitem ter acesso ao código-fonte do aplicativo utilizando métodos para desmontar, descompilar e reempacotar o aplicativo. Obtendo acesso ao código torna-se possível, através de uma análise estática e manual, revisá-lo para identificar vulnerabilidades nos binários do aplicativo, como por exemplo, descobrir se ele está criptografado e qual o tipo de criptografia utilizado durante a troca ou armazenamento das informações. Também é possível alterar variáveis chaves para fazer o sistema ignorar autenticações e validações, portanto aplicativos não devem armazenar, validar dados ou depender de autenticações somente no lado do cliente (SCAMBRAY; ROUSE; STANFIELD, 2013).

3.5. Tipos de análise de segurança

3.5.1. Análise estática

De acordo com OWASP (2018), para ser considerada estática, a análise de segurança de um aplicativo deverá examinar seus componentes sem que este seja executado. Esse tipo de análise pode ser executado através de uma avaliação manual ou automática do código-fonte do aplicativo.

3.5.1.1. Análise manual do código-fonte

Este método compreende a revisão do código-fonte de um aplicativo de forma manual através da busca por palavras chave ou uma avaliação linha a linha, visando identificar vulnerabilidades que possam colocar em risco a segurança da aplicação, (OWASP, 2018).

3.5.1.2. Análise automática do código-fonte

Já este tipo de análise, é caracterizado pela utilização de ferramentas que automatizam a avaliação do código-fonte da aplicação buscando sua conformidade em relação a um conjunto de regras ou práticas de programação segura pré-determinados, (OWASP, 2018).

3.5.2. Análise dinâmica

Para ser considerada dinâmica, a análise de segurança deverá examinar o aplicativo durante sua execução, objetivando encontrar vulnerabilidades nos mecanismos de controle de segurança da aplicação que fornecem proteção contra problemas de autenticação, autorização, configuração e comunicação, por exemplo. Esse tipo de análise também pode ser executado de forma manual ou automática, assim como a anterior. No entanto, seu resultado pode não ser tão efetivo se comparado a uma análise estática, (OWASP, 2018).

4 FERRAMENTAS OPENSOURCE

Essa seção apresenta ferramentas *opensource* para a realização de testes em aplicativos Android e distribuições para testes de segurança em aplicativos.

4.1. Ambiente de teste

Emuladores são programas instalados em computadores capazes de executar instruções como se fossem um dispositivo móvel, porém, nem todos os recursos de hardware de um dispositivo podem ser emulados. Enquanto isso os simuladores, são programas que podem repetir fielmente o comportamento de um dispositivo móvel. Nesse sentido, Chell et al. (2015) destacam o Virtualbox com Android x86, o Youwave e o Android Emulator do Android Studio, como sendo os emuladores Android que consomem menor recurso computacional e consequentemente apresentam um melhor desempenho, quando comparados a outros emuladores.

4.2. Ferramentas para análise estática e engenharia reversa

A seguir serão apresentas ferramentas para realização de análises estáticas e engenharia reversa.

4.2.1. AndroBugs

Conforme Lin (2015), o AndroBugs é uma ferramenta utilizada para identificar vulnerabilidades e problemas de segurança em aplicativos Android.

4.2.2. Androguard

O Androguard é baseado em Python e foi desenvolvido para abrir e descompilar aplicativos para o Android, (DESNOS, 2018).

4.2.3. Dex2jar

Ferramenta capaz de desempacotar o arquivo com extensão *.apk*, convertendo os arquivos *.dex* extraídos em arquivos *.jar* para análises posteriores, (MANJUNATH; COLLEY, 2011).

4.2.4. Apktool

O Apktool pode ser utilizado para recuperar um pacote ou arquivo pertencente a um aplicativo, próximo do seu estado de origem. Essa ferramenta pode ser considerada útil para acessar o estado inicial ou original de uma informação a qual não se tem disponível no momento, (CHELL et al., 2015).

4.2.5. JAADAS

Criado em 2014, o JAADAS (Joint Advanced Application Defect Assessment) é uma ferramenta escrita em Java e Scala desenvolvida para análise de uso indevido de API (Application Programming Interface), negação de serviço local e controle de fluxo entre processos, (JAADAS, 2017).

4.2.6. JAD

Ferramenta que permite decompilar arquivos Java Class ao seu estado de origem. De acordo com CHELL et al. (2015), além de não oferecer uma interface gráfica para o usuário, o JAD pode não ser considerada uma ferramenta confiável quando comparada a outras similares.

4.2.7. Jadx

Considerado uma ferramenta promissora por CHELL et al. (2015), o Jadx é um decompilador DEX que funciona tanto via linha de comando quanto por interface gráfica (GUI), possibilitando ao usuário navegar pelo código decompilado.

4.2.8. JD-GUI - Java Decompiler

Ferramenta utilizada para exibir o código-fonte de aplicativos desenvolvidos utilizando a linguagem de programação Java, (JAVA DECOMPILER).

4.2.9. QARK (Quick Android Review Kit)

De acordo com OWASP (2018) e Infosec Institute (2015), o QARK (Quick Android Review Kit) é uma ferramenta utilizada para realizar análises estáticas de segurança no código-fonte ou em APKs.

4.3. Ferramentas para análise dinâmica

A seguir serão apresentas ferramentas para realização de análises dinâmicas em aplicativos e no sistema operacional Android.

4.3.1. Drozer

O Drozer, chamado antigamente de Mercury, foi desenvolvido para realizar análises de vulnerabilidades em aplicativos ou dispositivos Android. A ferramenta pode interagir com a VM Dalvik, outros aplicativos e até mesmo com o sistema operacional, como se fosse um aplicativo inofensivo instalado no dispositivo móvel, (MWR LABS, 2015).

4.3.2. OWASP ZAP

Ferramenta desenvolvida para realizar testes de penetração e scanner de vulnerabilidades em aplicações Web. Com ela é possível realizar, por exemplo, testes de interceptação de dados e controle de acesso, (OWASP, 2017).

4.4. Ferramentas de apoio e Pentest

Abaixo são apresentas ferramentas de apoio e também ferramentas utilizadas em testes de penetração (Pentest).

4.4.1. ADB

O ADB ou Android Debug Bridge, proporciona uma interface via linha de comando para comunicação com o dispositivo real ou emulado, (CHELL et al., 2015).

4.4.2. DD

Presente nativamente em sistemas Linux e derivados como o Android, pode ser utilizado para gerar cópia bit-a-bit de arquivos, (CHELL et al., 2015).

4.4.3. DU

Presente nativamente em sistemas Linux e derivados, pode ser utilizado para listar informações de diretórios ou arquivos (CHELL et al., 2015).

4.4.4. DIFF

Ferramenta desenvolvida para realizar um comparativo entre arquivos, listando as diferenças existentes entre eles, (FSF, 2016).

4.4.5. MD5Deep

Ferramenta utilizada para calcular *hashes* de arquivos e diretórios. Pode operar de forma recursiva, pode comparar uma lista de hashes conhecidos com outros elementos de entrada, (KORNBLUM, 2014).

4.4.6. Netcat

O Netcat é utilizado para transferir dados para uma unidade de armazenamento localizada fora do dispositivo, visando assim, não comprometer a integridade dos dados e principalmente das unidades de armazenamento internas que poderão servir como evidências para análises posteriores. Este utilitário compõe o pacote de ferramentas chamado BusyBox, (WOLFE, 2013).

4.5. Distribuições para testes de segurança em aplicativos móveis

A seguir são apresentas distribuições ou *frameworks* incluindo várias ferramentas desenvolvidas para a realização de testes de segurança em aplicativos e sistema operacional Android.

4.5.1. Appie

Ambiente Integrado Portátil Pentesting para Android. De acordo com Agrawal (2018), o Appie foi desenvolvido para ser executado no sistema operacional Windows e contém ferramentas úteis para a realização de avaliações de segurança, análise forense e de *malware* em sistemas Android. Contendo aproximadamente 1,5 GB de tamanho o Appia pode ser armazenado e transportado em um pendrive. Entre as ferramentas incluídas no *framework* estão: Android Debug Bridge, Apktool, AndroBugs Framework, AndroGuard, Androwarn, Atom, ByteCodeViewer, Burp Suite, Drozer, dex2jar, Eclipse IDE com Android Developer Tools, Introspy-Analyzer, Java Debugger, jadx, Java Runtime Environment (JRE), Jd-Gui, Mozilla Firefox com alguns *addons* de segurança

Pidcat, SQLite Database Browser, SQLmap, Volatility Framework e alguns comandos UNIX (ls, cat, chmod, cp, find, git, unzip, mkdir, ssh, openssl, keytool, jarsigner).

4.5.2. Android Tamer

O Android Tamer é um *framework* baseado no sistema operacional Linux Debian 8, que contém ferramentas para a realização de avaliações de segurança, análise de *malware*, testes de invasão e engenharia reversa no sistema operacional Android e em aplicativos desenvolvidos para ele. Conforme AndroidTamer Team (2017), a versão 4 do Android Tamer é compatível com os virtualizadores VirtualBox e VMware Player. Para executar o *appliance* virtual com 5.1 GB de tamanho é necessário no mínimo 1 GB de memória RAM. Atualmente as ferramentas inclusas no *framework* são: adb wrapper, apk2java, drozer-checks, Android SDK, Android NDK, Android Studio, proguard, visualvm, gradle, MobSF, drozer, findbug, flawfinder, aapt, pidcat, dex2jar, enjarify, apktool, jd-gui, jad, jadx, smali, OWASP ZAP, Burpsuite-free, w3af, nikto, nmap, sslscan, wireshark/tshark , tcpdump, skipfish, wapiti, ratproxy, volatility, autopsy, dc3dd, dcfldd, dff, ext4magic, scalpel, sleuthkit, exif, metacam, exiftags, exifprobe, testdisk, steghide, guymager, rdd, fatcat, foremost, fastboot, heimdall e flashrom.

4.5.3. AndroL4b

Conforme OWASP (2018), o AndroL4b é uma máquina virtual baseada no sistema operacional Ubuntu Mate, composta por diversas ferramentas para avaliar aplicativos desenvolvidos para o sistema Android e realizar análises de segurança, engenharia reversa e análise de *malware*. Dentre as ferramentas disponíveis no AndroL4b versão 3, Shahin (2017) destaca as seguintes: APKStudio, Lobotomy, DroidBox, ClassyShark, Smartphone Pentest Framework (SPF), Radare2, Frida, ByteCodeViewer, Mobile Security Framework (MobSF), Drozer, APKtool, AndroidStudio, BurpSuite, Wireshark, MARA, FindBugs-IDEA, AndroBugs Framework e Qark.

4.5.4. Santoku Community Edition

De acordo com NowSecure (2014), o Santoku CE é baseado no sistema operacional Linux Lubuntu e contém ferramentas voltadas para a realização de avaliações de segurança, análise de *malware*, análise forense em sistemas Android. Atualmente, dentre as principais ferramentas inclusas na versão 0.5 do Santoku CE são: Android SDK Manager, AXMLPrinter2, Fastboot, Heimdall, SBF Flash, Burp Suite, Ettercap, Mercury,

nmap, OWASP ZAP, SSL Strip, w3af, Zenmap, Chaosreader, dnschef, DSniff, TCPDUMP, Wireshark, AFLogical Open Source Edition, Android Brute Force Encryption, ExifTool, libimobiledevice, scalpel, Sleuth Kit, Androguard, Antilvl, APK Tool, Baksmali, Dex2Jar, Jasmin, JD-GUI, Mercury, Radare2 e Smali.

5 ANÁLISE DOS DADOS COLETADOS

Para a realização das análises estáticas foram encontradas as ferramentas AndroBugs, JAADAS e QARK, capazes de identificar vulnerabilidades em aplicativos desenvolvidos para o sistema Android. Elas podem ser utilizadas tanto para validar a autenticação e autorização de acesso quanto a proteção de informações armazenadas localmente no dispositivo.

A análise do código-fonte pode ser realizada através de técnicas de engenharia reversa. Para abrir, converter e/ou descompilar os aplicativos foram encontradas as ferramentas: Androguard, Dex2jar, JAD e Jadx. Após obter acesso ao código do aplicativo, uma análise estática e manual pode ser realizada utilizando as ferramentas: Jadx e Java Decompiler. Esse tipo de análise deve buscar identificar se o aplicativo utiliza técnicas de ofuscação de código, o que dificultaria a compreensão das informações dentro do código original.

Para as análises dinâmicas, foram encontradas as ferramentas *opensource* Drozer e OWASP ZAP. Enquanto a primeira ferramenta realiza análises de vulnerabilidades em aplicativos e dispositivos Android, a segunda intercepta e analisa a troca de dados realizada entre o dispositivo e aplicações Web para identificar se as informações são enviadas utilizando um canal criptografado.

Além de ferramentas utilizadas para a realização das análises estáticas e dinâmicas, a pesquisa também encontrou ferramentas que podem ser utilizadas como apoio durante os testes. Enquanto as ferramentas ADB e Netcat podem ser utilizadas, respectivamente, para acessar o dispositivo via linha de comando e transferir arquivos para fora dele. As ferramentas DD, DU, DIFF e MD5Deep, desenvolvidas inicialmente para o sistema operacional Linux, são utilizadas para manipular arquivos e diretórios dentro do dispositivo.

Por fim, a pesquisa encontrou distribuições baseadas no sistema operacional Linux como o Android Tamer, AndroL4b e Santoku CE, que reunem várias das ferramentas citadas anteriormente em um só lugar e podem ser utilizadas para a realização de análises estáticas, dinâmicas, engenharia reversa e análise de código-fonte de aplicações desenvolvidas para o sistema Android. Além das distribuições citadas anteriormente, o *framework* Appie é uma alternativa que pode ser utilizada no sistema operacional Windows para realização de análises de segurança em aplicativos Android.

6 CONCLUSÃO

Este trabalho realizou uma revisão bibliográfica afim de encontrar ferramentas *opensource* que pudessem ser utilizadas em avaliações de segurança em aplicativos disponibilizados e executados no sistema operacional Android, visando identificar se os mesmos possuem as preocupações de segurança necessárias para proteção das informações pessoais de seus usuários.

Após a realização da pesquisa, foi possível identificar que existem ferramentas *opensource* que podem auxiliar a identificar de forma gratuita se os aplicativos disponibilizados para o sistema operacional Android possuem as preocupações de segurança necessárias para proteção das informações pessoais de seus usuários.

Vale salientar que existem outras ferramentas capazes de realizar análises dinâmicas além do Drozer e OWASP ZAP, no entanto, elas não foram relacionadas no trabalho pois se tratavam de ferramentas proprietárias, fugindo do objetivo da pesquisa, a qual buscava encontrar ferramentas *opensource* para realização das análises de segurança.

Dentre os resultados da pesquisa, destacam-se também, as distribuições para testes de segurança em aplicativos móveis como Appie, Android Tamer, AndroL4b e Santoku CE, as quais reunem em um único local várias das ferramentas encontradas na pesquisa. Essas distribuições e *frameworks* proporcionam praticidade e agilidade às pessoas que pretendem realizar analises de segurança em aplicativos.

Como pesquisa futura, sugere-se a realização de um estudo de caso utilizando ferramentas relacionadas no trabalho, afim de identificar a efetividade das mesmas em uma análise de segurança em aplicativos Android.

REFERÊNCIAS

AGRAWAL, Aditya. **Appie - Android Pentesting Portable Integrated Environment**, 2015. Disponível em: <https://manifestsecurity.com/appie>. Acesso em: junho de 2018

ANDROIDTAMER TEAM. **AndroidTamer Tools**, 2017. Disponível em: <https://tools.androidtamer.com>. Acesso em: junho de 2018.

CHELL, Dominic et al. **The mobile application Hacker's handbook**. John Wiley & Sons, 2015.

CHO, T. et al. **Potential vulnerability analysis of mobile banking applications**. In: ICT CONVERGENCE (ICTC), 2013 INTERNATIONAL CONFERENCE ON, 2013. Anais. p. 1114–1115.

DESNOS, Anthony. **Androguard Documentation - Release 3.2.0,** 2018. Disponível em: <https://media.readthedocs.org/pdf/androguard/latest/androguard.pdf>. Acesso em: julho de 2018.

FSF, Free Software Foundation. **GNU Diffutils**, 2016. Disponível em: <https://www.gnu.org/software/diffutils>. Acesso em: junho de 2018.

GUNASEKERA, S. **Android apps security**. Apress, 2012.

IDC, I. D. C. **Smartphone OS**, 2017. Disponível em: <https://www.idc.com/promo/smartphone-market-share/os>. Acesso em: maio de 2018.

INFOSEC INSTITUTE. **QARK - Uma ferramenta para avaliações automatizadas de aplicativos Android,** 2015. Disponível em: <https://resources.infosecinstitute.com/qark-a-tool-for-automated-android-app-assessments/#gref>. Acesso em: julho de 2018.

JAADAS. **Joint Advanced Application Defect Assessment for Android Application**, 2017. Disponível em: <https://github.com/flankerhqd/JAADAS>. Acesso em: julho de 2018.

JAVA DECOMPILER. **JD-GUI - Java Decompiler**. Disponível em: <http://jd.benow.ca>. Acesso em: julho de 2018.

LIN, Yu-Cheng. **Androbugs framework: An android application security vulnerability Scanner**, 2015. Blackhat Europa 2015. Disponível em: <https://www.blackhat.com/docs/eu-15/materials/eu-15-Lin-Androbugs-Framework-

An-Android-Application-Security-Vulnerability-Scanner.pdf>. Acesso em: julho de 2018.

MANJUNATH, V.; COLLEY, M. **Reverse engineering of malware on android. SANS Institute InfoSec Reading Room,** 2011.

MWR LABS. **Drozer User Guide**, 2015. Disponível em: <https://labs.mwrinfosecurity.com/assets/BlogFiles/mwri-drozer-user-guide-2015-03-23.pdf>. Acesso em: julho de 2018.

NOWSECURE. **Santoku Linux**, 2014. Disponível em: <https://santoku-linux.com>. Acesso em: julho de 2018.

OWASP. **Mobile Security Testing Guide - 1.0 Release,** 2018. Disponível em: <https://leanpub.com/mobile-security-testing-guide>. Acesso em: junho de 2018.

OWASP. **OWASP ZAP 2.6 - Getting Started Guide**, 2017. Disponível em: <https://github.com/zaproxy/zaproxy/releases/download/2.6.0/ZAPGettingStartedGuide-2.6.pdf>. Acesso em: julho de 2018.

POPA, M. **Techniques of program code obfuscation for secure software**. Journal of Mobile, Embedded and Distributed Systems, v. 3, n. 4, p. 205–219, 2011.

RAI, P. O. **Android application security essentials**. Packt Publishing Ltd, 2013.

SCAMBRAY, J.; ROUSE, J.; STANFIELD, M. **Hacking exposed mobile: security secrets & solutions**. McGraw Hill Professional, 2013.

SHAHIN, Amir H.. **AndroL4b**, 2017. Disponível em: <https://github.com/sh4hin/Androl4b>. Acesso em: junho de 2018.

WOLFE, Brandon L.. **High Performance Network Communication between High Frequency Application Servers and Android Tablets**, 2013.