GUSTAVO ROTONDO

PROSEC – UMA SOLUÇÃO PARA O CONTROLE DE PROCESSOS MALICIOSOS NA PLATAFORMA ANDROID

Bagé
2016

GUSTAVO ROTONDO

PROSEC – UMA SOLUÇÃO PARA O CONTROLE DE PROCESSOS MALICIOSOS NA PLATAFORMA ANDROID

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Computação da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Computação.

Orientador: Érico Holff do Amaral

Bagé
2016

Ficha catalográfica elaborada automaticamente com os dados fornecidos
pelo (a) autor(a) através do Módulo de Biblioteca do
Sistema GURI (Gestão Unificada de Recursos Institucionais).

R848p Rotondo, Gustavo

PROSEC – UMA SOLUÇÃO PARA O CONTROLE DE PROCESSOS MALICIOSOS NA PLATAFORMA ANDROID / Gustavo Rotondo.
93 p.

Trabalho de Conclusão de Curso (Graduação) -- Universidade Federal do Pampa, ENGENHARIA DE COMPUTAÇÃO, 2016.
"Orientação: Érico Amaral".

1. Android. 2. Dispositivos Móveis. 3. Segurança. 4. Sistemas operacionais. I. Título.

GUSTAVO ROTONDO

PROSEC – UMA SOLUÇÃO PARA O CONTROLE DE PROCESSOS MALICIOSOS NA PLATAFORMA ANDROID

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Computação da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Computação.

Trabalho de Conclusão de Curso defendido e aprovado em: 6 de Julho de 2016

Banca examinadora:

_____

Prof. Dr. Érico Marcelo Holff do Amaral
Orientador
UNIPAMPA

_____

Prof. Dr. Leonardo Bidese de Pinho
UNIPAMPA

_____

Profª. Drª. Sandra Dutra Piovesan
UNIPAMPA

Dedico esse trabalho a todas as pessoas que, de alguma forma, contribuíram para que a implementação deste trabalho de conclusão de curso fosse possível.

AGRADECIMENTO

A Deus primeiramente, aos meus pais por todo o apoio durante a graduação e aos colegas e professores do curso de Engenharia de Computação por toda troca de conhecimentos e experiências.

"Sua falta de fé é perturbadora. ”

Darth Vader - Star Wars - Episódio I: A Ameaça Fantasma

RESUMO

Com o crescente uso do sistema operacional Android, os dispositivos móveis têm sido alvo de diversos ataques. Estes ataques têm como objetivo principal a captura de informações pessoas do usuário. Embora existam várias aplicações que visam prover segurança do sistema *Android*, estas aplicações não realizam uma análise em baixo nível no sistema, deixando assim brechas para que aplicações que são executadas em uma camada inferior daquelas que o antivírus tem acesso possam ter controle do dispositivo e capturar as informações pessoais do usuário. O objetivo deste trabalho é propor e implementar uma ferramenta de processos ativos em dispositivos que tenham embarcado o sistema operacional Android. Foi verificado que através da análise e controle destes processos é possível incrementar a segurança do sistema, uma vez que todas as aplicações que estão em execução, sejam elas em uma camada mais alta ou mais baixa no sistema podem ser capturadas e analisadas pela ferramenta proposta.

Palavras Chave: Android. Segurança de sistema. Processos.

ABSTRACT

With the increasing use of the Android operating system, mobile devices have been the target of several attacks. These attacks are mainly intended to capture user information people. While there are several applications that aim to provide security of the existence of anti-virus software available today, these applications do not perform an analysis at a low level in the system, thus leaving loopholes for applications that run at a lower those that antivirus layer has access can take control the device and capture personal information. The objective of this research is to propose and implement a tool that performs the analysis of processes on the device that has embedded the Android operating system. It was found that through the analysis and control of these processes is possible to increase system security for all applications that are running, whether on a higher layer or lower in the system can be captured and analyzed by the proposed tool.

Keywords: Android. System security. Process.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ADB - *Android Debug Bridge*
AES - *Advanced Encryption Standard*
API - *Application Programming Interface*
ARM - *Advanced RISC Machine*
DES - *Data Encryption Standard*
FTP - *File Transfer Protocol*
GPS – *Global Position System*
IDE - *Integrated Development Environment*
IEEE - Instituto de Engenheiros Eletricistas e Eletrônicos
IOS - *iPhone Operation System*
IP – *Internet Protocol*
MMS - *Multimedia Message Service*
PDA - *Personal digital assistants*
PROSEC – *Process Security*
RAM – *Random Access Memory*
ROM – *Read Only Memory*
SDK - *Software Development Kit*
SO – Sistema Operacional
SMS - *Short Message Service*
SOC – *System On Chip*
UML - *Unified Modeling Language*
URL - *Uniform Resource Locator*
XML - *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Devido ao avanço tecnológico das últimas décadas, o computador tornou–se cada vez mais presente no nosso cotidiano. Nos primórdios da computação tínhamos imensas salas, as quais eram ocupadas por enormes computadores que, se comparados aos atuais, apresentavam um desempenho baixíssimo (SONG, 2010). Atualmente contamos com computadores cada vez menores e com grande poder computacional, dentre eles os chamados *smartphones* que gradativamente tornaram-se mais comuns nos dias de hoje. Com isso, os usuários passaram a armazenar suas informações pessoais em seus dispositivos, o que pode se tornar um problema caso o aparelho venha sofrer algum tipo de ataque malicioso.

No universo de dispositivos móveis disponíveis atualmente no mercado, existem diversas plataformas, dentre as quais o sistema Android vem se destacando pelo expressivo número de usuários. (NORBEN *et al*. 2013). Dentre os usuários de *smartphones*, 84,7% fazem uso do sistema Android, o qual possui aproximadamente 1,3 milhão de aplicações disponíveis para *download* dentre as quais apenas 1% das aplicações foram consideradas maliciosas (HAMMAN, 2014). Por conta do significativo número de usuários e por ser um sistema de código aberto, esta plataforma é uma das mais propícias a ataques de invasores e por isso necessita de constantes atualizações e uso de aplicações para incremento de segurança (RAMALHO *et. al,* 2013).

Um estudo realizado entre 2010 e 2011 analisou 1260 amostras de possíveis aplicações maliciosas e foi constatado que 86% das amostras eram *trojans* e 51% dessas aplicações tinham como objetivo a coleta de informações dos usuários (ZHOU, 2012). Analisando esses dados, pode-se concluir que a plataforma Android é amplamente utilizada em sua maioria por usuários que não possuem conhecimento técnico e devido a isso sofre constantes ameaças e ataques, o que demonstra a necessidade de alguma ferramenta para o acréscimo de segurança que não faça a varredura somente de aplicações e arquivos que são visíveis ao usuário, mas sim uma varredura mais profunda e completa do sistema. Os sistemas que serão verificados possuem aplicações, a qual geram processos. Esses processos representam as tarefas que estão em execução (AMOROSO, 2009).

1.1 Problema de Pesquisa

De acordo com Braga *et.al* (2012), a partir de 2011 o crescimento no número de aplicações e usuários de dispositivos móveis foi o grande fator de influência no surgimento de aplicações maliciosas para estas plataformas. O sistema operacional mais afetado era o sistema Android por conta do grande número de usuários e por ser uma plataforma de código-fonte aberto.

Estando ciente do crescente número de usuários e do grande número de aplicações maliciosas que visam, de alguma forma, explorar vulnerabilidades do sistema operacional com o intuito de capturar algum dado privado do usuário ou danificar o sistema, foi proposto o seguinte problema de pesquisa: É possível propor e implementar uma solução para o monitoramento de processos no sistema operacional Android que identifique de forma efetiva aplicações maliciosas neste sistema?

1.2 Objetivos Gerais

Este trabalho objetiva propor uma solução de segurança para dispositivos móveis por meio do projeto e implementação de uma ferramenta para a identificação e monitoramento de processos maliciosos ou desconhecidos no sistema. O intuito é manter o usuário informado sobre todas as aplicações ativas e incrementar a segurança no dispositivo.

1.3 Objetivos Específicos

Este trabalho vislumbra implementar e validar um modelo para avaliação de potenciais processos perigosos que estejam em execução em um determinado dispositivo móvel, a fim de garantir a manutenção dos princípios básicos da segurança da informação.

- Fazer um estudo do estado da arte sobre segurança no sistema Android*;*
- Analisar processos em segundo plano em dispositivos móveis;
- Criar o filtro de segurança para a análise dos processos em execução;

- Levantar os requisitos da aplicação proposta;
- Desenvolver a aplicação proposta para a plataforma Android;
- Implementar e avaliar os testes de funcionalidade e usabilidade;
- Coletar e analisar os resultados obtidos.

1.4 Estrutura do Texto

O trabalho contém a seguinte estrutura: o primeiro capitulo é constituída da introdução sobre o assunto que objetiva fazer uma apresentação geral sobre o tema e apresentar os objetivos e problema de pesquisa. O segundo capitulo traz os conceitos mais relevantes e a revisão da literatura. Esta etapa é responsável pelo embasamento teórico do estudo, na qual são abordados os seguintes assuntos: levantamento de referências sobre dispositivos móveis, arquiteturas disponíveis para dispositivos móveis, sistemas operacionais para *smartphones* com ênfase no sistema Android e sua estrutura, segurança em dispositivos móveis e trabalhos correlatos. O terceiro capítulo apresenta a metodologia utilizada no estudo. São apresentadas todas as etapas executadas e o que cada uma resultou. O quarto capitulo mostra a implementação da ferramenta proposta. Os resultados e discussões preliminares são apresentados no quinto capitulo. O sexto e último capitulo apresenta as conclusões e uma discussão sobre trabalhos futuros.

2 CONCEITOS GERAIS E REVISÃO DE LITERATURA

Desde a década de 1990, houve um grande aumento no desenvolvimento de soluções tecnologia móvel, comunicação sem fio e telefonia por satélite. Tal evolução se dava pela demanda de comunicação em lugares remotos que não dispunham de meios suficientes para que os usuários pudessem se comunicar com o mundo externo. (MANDEL *et.al, 1997).*

Notou-se também uma grande evolução na área de dispositivos computacionais móveis, entre eles os *laptops*, PDAs e telefones celulares. Ao decorrer do final do século XX, as tecnologias móveis se tornaram evidentes no cotidiano e o número de usuários que passou a utilizar essas tecnologias.

Tendo em vista o grande investimento nestas tecnologias e o crescente número de usuários adeptos, surgiu um novo conceito: computação móvel. A computação móvel pode ser representada com um novo paradigma computacional em que possibilita o usuário tenha acesso a serviços independentemente do local em que o mesmo se encontre (NAKAMURA, 2003). Neste contexto, o termo mobilidade está alinhado à descrição de portabilidade, a qual define que computadores devem possuir dimensões reduzidas, ao ponto de serem facilmente transportados e manuseados (GATTO, 2012).

Com o passar do tempo, o telefone celular se tornou mais presente no cotidiano do ser humano. O dispositivo foi passando por diferentes transformações tecnológicas a fim de suprir as necessidades de manter o usuário conectado com o mundo exterior. Criado em 1974 o aparelho *Motorola DynaTac* foi desenvolvido para competir com os telefones instalados em carros. O aparelho pesava 794 gramas e tinha um custo de aproximadamente 4 mil dólares. A figura 1 apresenta o *Motorola DynaTac.* (TERRA, 2014)

Figura 1 – Evolução dos Celulares

a. Motorola DynaTac **b. J-Sho4** **C. iPhone** **d. Nexus ONE** **e. Galaxy s6 Edge**

Fonte: Terra (2015)

Em 1999, o conceito de *smartphone* começou a ficar mais difundido com o lançamento do modelo *J-Sho4* da *Sharp.* O aparelho foi o primeiro na história a ter uma câmera digital embutida. A figura 1.b apresenta o J-Sho4.

No ano de 2007, a Apple lançou o aparelho que seria o divisor de águas no universo de telefones celulares. O *iPhone* foi o primeiro *smartphone* lançado. Com ele era possível tirar fotos, checar e-mails, navegar pela Internet, reproduzir vídeos e músicas (TECMUNDO, 2014). O primeiro iPhone está representado na figura 1.c.

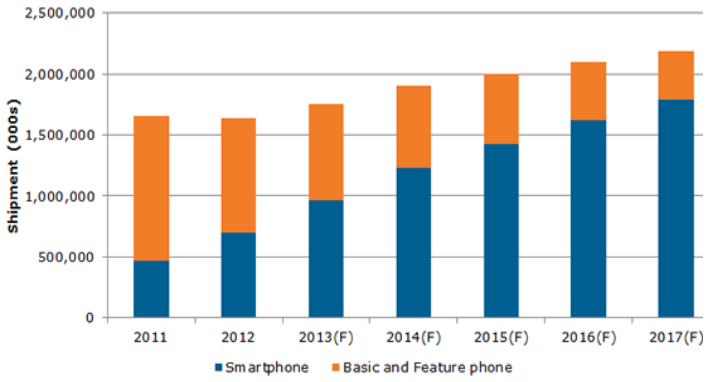
Em 2008, a gigante de buscas *Google* lançou um *smartphone* para concorrer com o lançamento da *Apple*, o *Iphone 3GS.* O *Google Phone* ou *HTC Nexus One* como era conhecido, foi o primeiro *smartphone* a vir com o sistema Android embarcado. O dispositivo era equipado com uma configuração superior da encontrada no dispositivo da *Apple*. O *Google Phone* era equipado com um processador de 1GHz, 512 MB de memória RAM e 512 MB de memória ROM. (TECMUNDO, 2010). O primeiro *smartphone* com o sistema Android é mostrado na figura 1.d.

Atualmente os *smartphones* possuem grande poder computacional. No ano de 2015 a *Samsung* lançou o seu modelo top de linha *Galaxy s6 Edge*, o qual possui um processador de oito núcleos e 4 GB de memória RAM. (SAMSUNG, 2015). A figura 1.e ilustra o aparelho da *Samsung.*

Na transição da primeira para a segunda década do século XXI, o telefone convencional foi se tornando menos comum em nosso cotidiano, dando lugar a *smartphones* cada vez mais avançados. Estima-se que em 2006, existiam aproximadamente 900 milhões de telefones celulares convencionais e 100 milhões de *smartphones*. De acordo com LI *et.al*, (2010) em 2015, a maioria dos aparelhos

celulares comercializados serão *smartphones.* A figura 2 demonstra a relação de vendas de telefoneis convencionais e *smartphones* e uma estimativa de vendas de *smartphones* e telefones convencionais para os anos de 2016 e 2017.

Figura 2 – Vendas de telefones convencionais e *smartphones*



Fonte: Li *et al.* (2010)

Com grande aceitação do usuário em começar a substituir seus computadores convencionais por *smartphones*, o dispositivo móvel começou a ser alvo de ataques maliciosos. Estas ações têm como objetivo acessar informações pessoais do usuário, redirecionamento de sites para acessos em propagandas, envio de mensagens e realização de ligações telefônicas sem consentimento do usuário (AFONSO, 2013).

Normalmente, quando um sistema está infectado por algum tipo de *malware,* o utilizador do dispositivo não tem conhecimento de que o mesmo está infectado. Assim como nos computadores convencionais, foram desenvolvidos *softwares* antivírus para incrementar a segurança do dispositivo móvel. Todavia, grande parte destas aplicações realizam a busca de programas maliciosos e arquivos maliciosos deixando os processos em execução sem verificação. A solução para este tipo de vulnerabilidade poderia ser constituída de uma ferramenta que fizesse uma varredura em um nível mais inferior do sistema, monitorando os processos nativos dos dispositivos e processos gerados por aplicações instaladas com ou sem o consentimento do usuário.

2.1 Arquiteturas de Dispositivos Móveis

Com a evolução dos dispositivos móveis foi necessário criar uma arquitetura especifica para este tipo de dispositivo, a qual deveria apresentar um desempenho aceitável, atrelado a um baixo consumo energético, a fim de prover o conceito de portabilidade ao usuário. Atualmente, dois modelos de arquiteturas são encontrados nos *smartphones*, sendo elas a arquitetura *ARM* e a arquitetura X86-64 desenvolvida pela Intel (MORIMOTO, 2012).

O primeiro chip ARM foi fabricado pela *Acron Computers Limited* na década de 80. A empresa buscava um processador para *desktop*s da próxima geração. O desenvolvimento do chip foi motivado pela inexistência de processadores com desempenho aceitável e instruções simplistas (ZANONI, 2013).

Jaggar (1997) explica que o modelo ARM7 é um microprocessador com tamanho de palavra de 32 bits com baixo consumo de energia e por essa característica, foi amplamente embarcado em sistemas portáteis. O chip possuía 3 estágios de pipeline e um *clock* de 80 MHz e adotava a arquitetura de Von Neumann para a implementação de sua memória. Os dados e instruções ocupavam um simples endereço e então eram acessados individualmente.

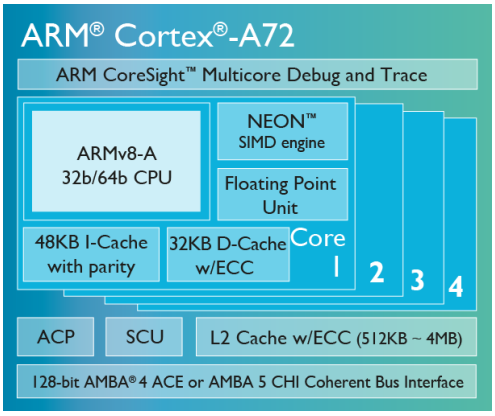
Na década de 90, a arquitetura ARM começou a ser embarcada nos dispositivos móveis pela *Apple* em seu *PDA Newton*. A partir desse momento, várias outras empresas começaram a embarcar seus dispositivos com processadores ARM, com isso a arquitetura *ARM* vem sendo a mais utilizada atualmente nos dispositivos móveis. (ZANONI, 2013).

Dentre os dispositivos modernos, equipados com processadores ARM, destaca-se o *chip* Cortex-A72, um microprocessador projetado para suportar instruções de 64 bits e funcionar com frequência de 2.5 GHz. Uma das principais tecnologias embarcadas no chip é a configuração conhecida como *big.LITTLE*. Essa tecnologia permite em um único soc[1] duas arquiteturas embarcadas, ou seja, podemos ter em um único *smartphone* dois processadores, geralmente um de alto desempenho e alto consumo de energia e outro de baixo desempenho e baixo consumo de energia (ARM, 2015). A figura 3 ilustra o diagrama de blocos do *System on Chip (*soc) mais atual desenvolvido pela ARM. Na imagem é possível perceber a existência dos

[1] *System on chip. Todos os componentes de um computador em um único circuito integrado.*

núcleos do processador, da cache utilizada pelo mesmo e dos outros componentes que compõe o soc.

Figura 3 - Diagrama de blocos do processador Cortex-A72



Fonte: ARM (2015)

2.2 Sistemas Operacionais para Dispositivos Móveis

Junto com o avanço dos dispositivos móveis e uma maior necessidade dos usuários necessitarem ter acesso a seus dados em qualquer lugar a qualquer hora, os sistemas operacionais para *smartphones* tiveram que se adaptar para atender os requisitos dos usuários. Com isso os *smartphones* passaram a substituir cada vez mais o computador convencional, pois com os sistemas operacionais bem evoluídos, foram surgindo aplicações que exerciam as mesmas funções das aplicações encontradas nos computadores. (CANDIDO, 2013).

O primeiro sistema operacional desenvolvido para dispositivos móveis foi o *Palm OS,* desenvolvido pela Palm Inc. para equipar os PDA[2]. O *Palm OS 1.0* tinha como objetivo os para aparelhos com tecnologia *touchscreen* e dispunha de uma interface gráfica simplista e está representado na figura 4.

[2] PDA - *Personal digital assistants* ou assistente pessoal digital.

Figura 4 - PDA equipado com sistema operacional Palm OS 1.0

Fonte: Wikipédia (2015)

Entretanto, no mercado atual, três sistemas operacionais se destacam pela quantidade de usuários e recursos disponíveis, sendo eles: Android, Windows Phone e iOS. A figura 5 apresenta a disparidade das plataformas.

Figura 5 – Adoção dos sistemas operacionais para dispositivos móveis

Period	Android	iOS	Windows Phone	BlackBerry OS	Others
Q1 2015	78.0%	18.3%	2.7%	0.3%	0.7%
Q1 2014	81.2%	15.2%	2.5%	0.5%	0.7%
Q1 2013	75.5%	16.9%	3.2%	2.9%	1.5%
Q1 2012	59.2%	22.9%	2.0%	6.3%	9.5%

Fonte: IDC (2015)

Estas plataformas atualmente ocupam a posição de principais sistemas operacionais para dispositivos móveis sendo grandes concorrentes entre si. Conforme pesquisa feita pela *International Data Corporation* (IDC, 2015), o sistema operacional Android era apontado como sendo o sistema mais utilizado nos *smartphones* abrangendo 78% dos usuários. Logo em seguida temos o iOS que é gerenciado pelo *Apple* com 18,3%. Em terceiro lugar o *Windows Phone* com 2,7%.

O sistema Android foi inicialmente concebido para ser um *software* para gerência de câmeras digitais. Entretanto, os fundadores da Android Inc. perceberam o potencial do mercado de dispositivos móveis e decidiram, em 2004, criar um sistema operacional para *smartphones* para concorrer com os sistemas existentes na época. A empresa Android Inc*.* foi comprada pela *Google* que em 17 de agosto de 2005, passou a ser proprietária do sistema Android. O objetivo era o desenvolvimento de uma plataforma móvel baseada em Linux para dispositivos móveis. Em 22 de outubro, a *Google* lançou oficialmente o primeiro dispositivo com o sistema operacional embarcado (GUIMARÃES, 2013).

Ao longo de sua trajetória, o sistema operacional Android passou por diversas atualizações que incluíram novas funcionalidades ao sistema, correções de *bugs* e relacionados à segurança. O *kernel* do sistema também foi atualizado. Com isso, novas funcionalidades foram inclusas no sistema que, por sua vez, passou a gerar mais processos para gerir essas novas funcionalidades. Com o aumento dos sensores embarcados nos dispositivos móveis, novamente mais processos passaram a gerados para o controle desses novos sensores. A tabela 1 mostra todas as versões do sistema Android bem como a data de lançamento e as características fundamentais de cada nova versão.

Tabela 1- Versões do sistema Android

Versão	Características	Data de Lançamento
Android 1.0	O sistema possuía o *Android Market* que permitia o download de aplicações. Foi incluso também acesso a câmera digital, navegador *web* e suporte a notificações na barra de *status.*	Setembro de 2008
Android 1.1	Suporte para salvamento de anexos em mensagens, atualização da API[3], mais recursos na utilização do *Google Maps*.	Fevereiro de 2009
Android 1.5	Suporte para teclado na tela, gravação e reprodução de vídeos, emparelhamento com dispositivos *Bluetooth,* rotação da tela	Abril de 2009

[3] *Application Programming Interface* ou interface de programação de aplicativos. Funciona para verificar a compatibilidade de uma aplicação com alguma versão do sistema operacional.

	utilizando o giroscópio embarcado no dispositivo, suporte para *Widgets*[4] e animação de *boot.*	
Android 2.0	Múltiplas contas de *e-mail*, suporte para flash e zoom no aplicativo de câmera, limpeza de SMS e MMS e calendário.	Outubro de 2009
Android 2.2	Suporte a armazenamento de aplicações em sistemas externos tais como cartão micro SD. Foi aprimorada a velocidade do sistema e do navegador. Utilizando como base o Android *2.2* foram lançadas mais 3 atualizações sendo elas 2.2.1, 2.2.2 e 2.2.3, porém sem nenhum recurso relevante, apenas correções de *bugs.*	Maio de 2010
Android 2.3	Teclado digital foi redesenhado e teve ganhos de desempenho. Foi aprimorado também o uso do giroscópio, sendo possível que aplicações de terceiro tenham acesso ao sensor do aparelho.	Dezembro de 2010
Android 3.0	Versão do sistema Android voltada para tablets.	Fevereiro de 2011
Android 4.0	Implementação de sistema multitarefa.	Outubro de 2011
Android 4.1	Suporte a uma taxa de quadros a 60 frames por segundo, suporte a notificações expandidas, *Widgets* redimensionáveis e maior velocidade na transição de aplicativos e telas.	Junho de 2012
Android 4.4	Como características, teve mudanças na interface, suporte a dispositivos vestíveis e diversas novas melhorias de usabilidade.	Outubro de 2013

[4] Versões miniaturizadas de aplicativos fixadas na tela principal

Android 5.0	Suporte para processadores 64 *bits* e a nova máquina virtual Java ART[5] que se difere da máquina *Dalvik* na maneira com que lidam com uma determinada aplicação. Enquanto a *Dalvik* faz o compilamento em tempo de execução, a *ART* executa um pré compilamento dos aplicativos, diminuindo assim o tempo de abertura das aplicações.	Novembro de 2014

Fonte: Próprio autor (2015)

A próxima versão do sistema operacional Android está em fase de testes e será conhecido como Android *M* ou *Marshmallow*. Dentre as novidades, foi incluso um novo sistema de gerenciamento de energia conhecido como projeto *Doze*. Foi percebido que aplicativos taxados como simples, faziam a requisição de diversas permissões do sistema, tais como acesso ao GPS, contatos, câmera, etc. Com isso implementou-se o Android *M* (6.0), um novo sistema de permissão de aplicações, no qual o usuário poderá liberar especificamente as permissões para uma nova aplicação. A figura 6 demonstra as diferentes versões do sistema Android e a percentagem de aparelhos que utilizam essas versões.

[5] *Android Runtime*, a nova máquina virtual implantada na versão 5.0 do sistema *Android*

Figura 6 - Fragmentação do sistema operacional Android

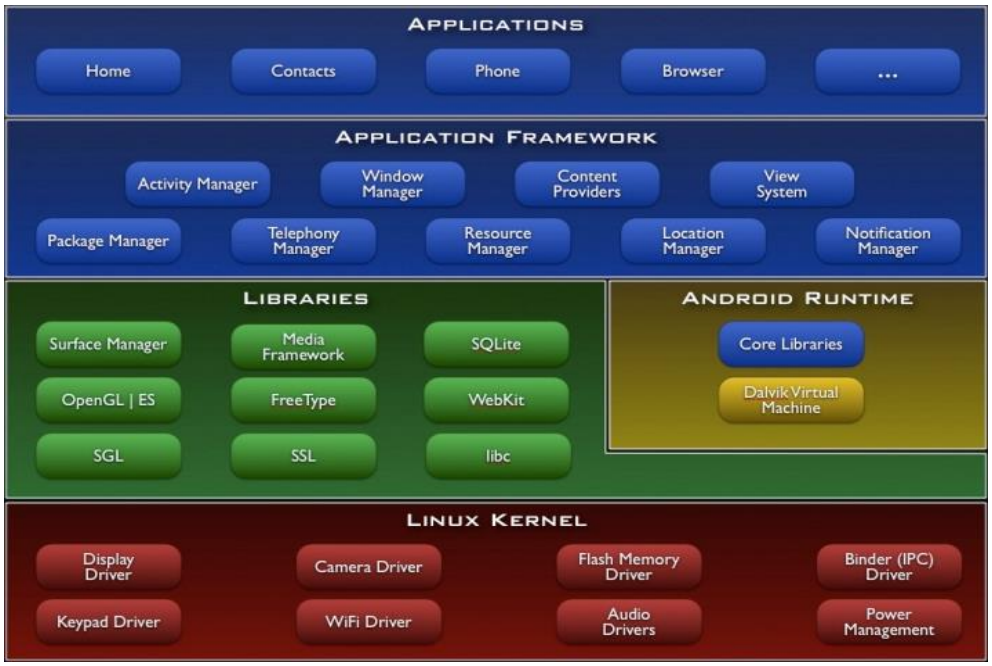
Version	Codename	API	Distribution
2.2	Froyo	8	0.3%
2.3.3 - 2.3.7	Gingerbread	10	5.6%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	5.1%
4.1.x	Jelly Bean	16	14.7%
4.2.x		17	17.5%
4.3		18	5.2%
4.4	KitKat	19	39.2%
5.0	Lollipop	21	11.6%
5.1		22	0.8%

Fonte: *Developer Android* (2015)

Atualmente o sistema Android se encontra na versão 5.1.1 conhecida como *Lollipop*. A atual versão está instalada em aproximadamente 0,8% dos dispositivos móveis e a versão mais utilizada é a versão 4.4 conhecida como *KitKat*, presente em 39,2% dos dispositivos móveis (ANDROID, 2015).

O sistema operacional Android é divido em cinco partes, conforme apresentado na Figura 7.

Figura 7 - Diagramas de componentes do sistema Android



Fonte: Bordim (2012)

A base do sistema é composta pelo *kernel* baseado em Linux. O *kernel* provê vários serviços essenciais ao sistema Android, entre eles o gerenciamento de memória do sistema, *drivers* e questões de segurança e energia. A segunda parte do sistema é responsável pelas bibliotecas utilizadas pelo sistema para um melhor funcionamento do Android. Tem-se a biblioteca responsável pela parte gráfica do sistema, bibliotecas que fazem a gerência de banco de dados, (no caso a biblioteca *SQLite)*. Ao lado tem-se a máquina virtual Java *Dalvik*, que é responsável pela execução dos aplicativos no sistema operacional Android. Na camada acima, encontram-se os *frameworks* das aplicações, que por sua vez fornecem todas as funcionalidades necessárias para o desenvolvimento das aplicações. A última camada concentra as aplicações do sistema, tais como câmera, discador, navegador, dentre outras (BORDIM, 2012).

2.3 Segurança de Sistemas

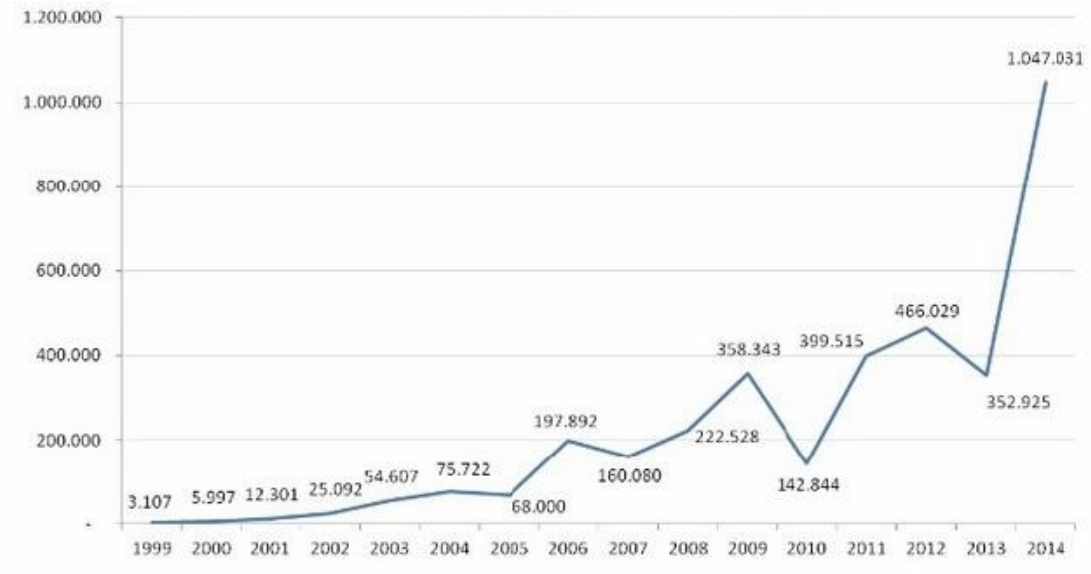
A área de segurança da informação trata dos recursos necessários para proteger informações e sistemas. Segundo Phonenix (2008), a segurança da informação de uma empresa garante em muitos casos, a continuidade de um negócio, incrementa a estabilidade de informações.

Maia (2013) citou que a segurança da informação é sustentada por 4 pilares, sendo eles:

- Integridade: Garante a integridade da informação enviada, ou seja, garante que a informação não foi modificada por algum invasor;
- Autenticidade: Garante que a origem da informação é autêntica;
- Disponibilidade: Garante que o sistema esteja sempre disponível;
- Confidencialidade: Garante que terceiros não terão acesso à informação.

Juntamente com o crescente uso do computador, os ataques a sistemas computacionais vêm crescendo igualmente. Uma pesquisa realizada em 2014 pelo Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil (CERT.br), mostrou que os ataques cibernéticos cresceram 197% em relação ao ano de 2013. Entre os ataques mais comuns estavam as fraudes e ataque de negação de serviço. A figura 8 ilustra o crescente ataque a sistemas computacionais. (CERT.br, 2015).

Figura 8 - Relatos de ataques computacionais



Fonte: CERT.br (2014)

Seguindo o crescente número de usuários de computadores, o aumento de *smartphones* estimulou que *crackers* focassem seus ataques em dispositivos móveis com o intuito de roubo de dados pessoais e senhas de serviços bancários. Um relatório elaborado pela *Kaspersky Labs* no segundo trimestre de 2015 informou que surgiram 291.800 novos programas que tinham como objetivo danificar o dispositivo móvel ou adquirir as informações contidas no mesmo. Tais ataques cresceram 2,8 vezes em relação ao primeiro trimestre de 2015. O relatório mostrou também que foram detectados mais de 1 milhão de pacotes de instalação de *malware* para dispositivos moveis, um aumento de 7 vezes quando comparado ao primeiro trimestre de 2015.

Os principais alvos desses aplicativos maliciosos são as aplicações bancárias. O relatório apresentou o *Trojan-SMS.AndroidOS.OpFake.cc* como o maior responsável por esse tipo ataque, infectando 29 aplicativos bancários (CRIMES PELA INTERNET, 2015).

Por ser o sistema operacional para dispositivos móveis mais utilizados atualmente, o Android é o sistema que mais sofre com os ataques. Uma pesquisa realizada pela *Symantec demonstrou* que, em 2014, 1 milhão de aplicações disponíveis para Android eram *malwares* disfarçados. Porém, graças a política adotada pela *Google*, uma minoria dessas aplicações estava na *Google Play Store*.

O risco real era maior de aplicações oriundas de lojas de aplicativos paralelas. O estudo apontou que na loja de aplicações da *Apple*, o número de aplicações maliciosas era de apena 3 (GLOBO, 2015).

2.4 Segurança no Sistema Android

Com a ampla utilização do sistema operacional *Android*, o SO virou alvo de ataques de *cyber* criminosos, que tentam por meios de vírus e outras pragas virtuais obter informações sobre o usuário ou roubar dados pessoais. Um dos recentes problemas foi detectado em agosto de 2013, conhecido como Master Key (BLACK HAT CONFERENCE, 2013). Trata-se de uma falha em que o atacante modifica o arquivo apk[6] e consegue inserir na aplicação, códigos maliciosos que possibilitam a abertura de brechas no sistema, sendo possível o roubo de informações sigilosas. Tal falha afetava em média 900 milhões de aparelhos que rodavam a versão 1.6 ou superior do Android. Esta falha foi consertada na versão 4.3 do sistema operacional.

Segundo o *Chief Executive Officer* da Kapersky, Eugene Kapersky, 99% dos vírus disponíveis para dispositivos móveis, tem como alvo o sistema operacional Android*.* Ainda segundo Eugene, o grande número de ameaças se deve ao grande número de usuários do sistema (TECHTUDO, 2015).

Ainda sobre vulnerabilidades no sistema Android, outra pesquisa realizada pela *Kapersky Labs,* apontou que em 2014, os números de ataques financeiros contra usuários de Android triplicaram. O número de ataques, de acordo com as informações divulgadas pelo estudo, cresceu 3,25 vezes em comparação com o mesmo período do ano anterior, aumentando de 711.933 para 2.317.194 (*KAPERSKY LABS,* 2015).

Para contornar o problema, a *Google* adota várias políticas de verificação de aplicações em sua loja oficial. Em relação aos *malwares,* um dispositivo infectado poderá enviar à *Google* informações sobre o *app* e sua origem. A empresa usará um banco de dados de programa de *malware* para comparar com o *app* e verificar se o mesmo é prejudicial ao aparelho (GOOGLE, 2015)

Mesmo com todas as políticas de envio de aplicações à *Google Play Store*, pesquisadores da Universidade de Indiana desenvolveram uma ferramenta que

[6] *Android Package*. Instalador de aplicações para dispositivos *Android.*

detecta aplicativos maliciosos para Android. Após a varredura, a ferramenta identificou 127.429 aplicações maliciosas em diversas lojas de aplicativos. A pesquisa verificou 1,2 milhões de aplicações em 33 lojas distintas. Um ponto grave elencado pelos pesquisadores foi em relação à loja oficial de aplicações, a qual teve 33 mil aplicativos reconhecidos como maliciosos (CHEN *et al*. 2015).

Outra pesquisa realizada pela *Check Point* demonstrou que um aplicativo contido na *Google Play Store* estava explorando uma vulnerabilidade recém descoberta para obter um elevado nível de acesso para o sistema operacional Android de muitos usuários, ignorando as permissões. Rotulada pelos pesquisadores como "*Certifi-Gate*", essa falha de segurança é causada por versões inseguras de ferramentas de administração remota instaladas pelos fabricantes e operadoras para fornecer serviço remoto, incluindo versões do cliente *TeamViewer, CommuniTake Remote Care, e MobileSupport da Rsupport.* A pesquisa indicou que a aplicação maliciosa foi instalada em mais de 100.000 dispositivos e que a *Google* tomou providências ao suspender o *download* da aplicação (ARS TECHINCA, 2015).

Um estudo mais recente sobre vulnerabilidades apontou uma grave falha que afeta os dispositivos com as versões 2.2 até a versão 5.1 do sistema Android. Conhecida como *Stagefrigh,* a vulnerabilidade foi taxada como a pior já descoberta pois usando apenas o número de telefone da vítima, crackers mal-intencionados podem enviar arquivos multimídia através do sistema MMS para garantir acesso ao dispositivo sem que o usuário perceba. A invasão ainda pode acontecer enquanto o aparelho estiver em modo de espera e possibilita que o criminoso apague todos os rastros de sua investida. A falha atinge cerca de 95% dos dispositivos móveis que executam o sistema Android (AVAST, 2015).

Para Fedler (2013), as deficientes soluções de antivírus para a plataforma Android e a falta de métodos de detecção eficazes para o comportamento dinâmico, ou seja, heurística ou abordagens de assinatura baseadas em objetos do sistema de arquivos arbitrários, resultam de limitações da plataforma. Segundo o Projeto *Android Open Source* (2014), qualquer *software* instalado em dispositivos Android não pode acessar os diretórios de outros aplicativos, então não é possível analisar arquivos de outros aplicativos e identificar algum comportamento malicioso. Qualquer *malware* não incluído nas bases de dados dos aplicativos antivírus não será detectado. Portanto, considera-se o nível de proteção oferecido pelos antivírus

na plataforma Android insuficiente. Em tese, uma aplicação que consiga verificar o sistema em um nível mais baixo poderia incrementar a segurança de um dispositivo Android.

2.5 Trabalhos Correlatos

O desenvolvimento deste trabalho é baseado no estudo e experimentos sobre o sistema operacional Android e suas fragilidades, buscando desta maneira, identificar aplicações que poderiam ser utilizadas aplicações podem ser utilizadas por usuários com o intuito de elevar o nível de segurança de sistemas móveis. Concomitante a este estudo, alguns trabalhos correlatos foram investigados, a fim de se verificar o estado da arte sobre soluções de segurança para o sistema operacional Android.

Zhou e Jian (2012) relatam em sua pesquisa intitulada "*Dissecting Android Malware: Characterization and Evolution*" que o rápido crescimento do sistema Android aponta para a necessidade de uma evolução nos padrões de segurança no sistema. O estudo descreve que diariamente novos *malwares* são criados com o intuito de se propagarem e roubarem informações pessoais dos dispositivos móveis. Como resultado do estudo foi constatado que 86% dos *malwares* analisados pelos pesquisadores eram obtidos por meio de aplicações disponíveis na própria loja de aplicativos da Google, sendo que destes, 36% se instalavam na raiz do sistema, onde conseguiam causar danos irreparáveis e 45% faziam uso de ligações e envio de mensagens, sem o consentimento do usuário, acarretando em cobranças indevidas na linha telefônica. Suas conclusões apontaram para a suscetibilidade a falhas do Android, as quais podem acarretar sérios prejuízos ao usuário final.

Afonso *el.al* (2013) mostrou que a partir de análise em aplicações disponíveis em lojas de *apps*[7], que grande parte tem potencial para ser um aplicativo malicioso. Foram coletadas 5.855 aplicações as quais foram submetidas ao Andrubis·, uma ferramenta capaz de analisar o sistema de forma dinâmica em busca de *malwares*. Com o uso da ferramenta, os seguintes comportamentos foram observados:

- *Hosts* acessados: Das aplicações analisadas, 29,18% (1.520) acessaram um *host* relacionado a propaganda. Além disso, 13,32% (694) se conectaram um host com finalidade de monitorar o usuário, e 1,79% (93)

[7] Aplicações para sistemas computacionais.

acessaram uma URL conhecida por ser usada por exemplares de *malware;*

- Vazamento de informações: Das aplicações analisadas, 13,88% (723) vazaram informações pela rede. Entre as informações vazadas, estão o número de IMEI e do telefone do dispositivo;
- Permissões subvertidas: 0,67% (35) das aplicações analisadas alteraram alguma permissão do sistema, 34 modificaram a permissão que disponibiliza o acesso à Internet e uma que permite o acesso a informações do aparelho;
- Mensagens e ligações: 0,13% (7) aplicações fizeram o envio de SMS ou MMS.

Schmidt (2009) diz que o sistema Android vem sofrendo constantes ataques de *malwares* e, dentre os sistemas operacionais para dispositivos moveis, este é o que mais sofre com a falta de segurança. A pesquisa também destacou uma grande vulnerabilidade quanto a dispositivos com perfil de administração (*root*), a qual permitiu reiniciar o dispositivo e também possibilitou a troca da senha do aparelho. Ao realizar uma análise em *malwares* coletados, foi verificado que a grande maioria infectava o dispositivo através de um arquivo executado, o que reforça a tese de se criar uma ferramenta para verificar processos gerados a partir de instalações das aplicações. O maior efeito colateral observado por Schmidt, foi a manipulação de arquivos do sistema, seguindo da desativação de aplicações. Essa é uma informação preocupante, pois dentro da manipulação de arquivos, está o envio de informações pessoais para o atacante, tais como as senhas de serviço.

Segundo Pilz (2012), a eficácia de aplicações antivírus deve ser questionada quanto a detecções de falso positivo. Realizou-se uma verificação em aplicações comumente instaladas por usuários e verificou-se que quando esta aplicação demanda mais recursos do dispositivo, tais como uso de processador e memória, existem grandes chances do *software* antivírus detectar a aplicação como maliciosa.

3 METODOLOGIA

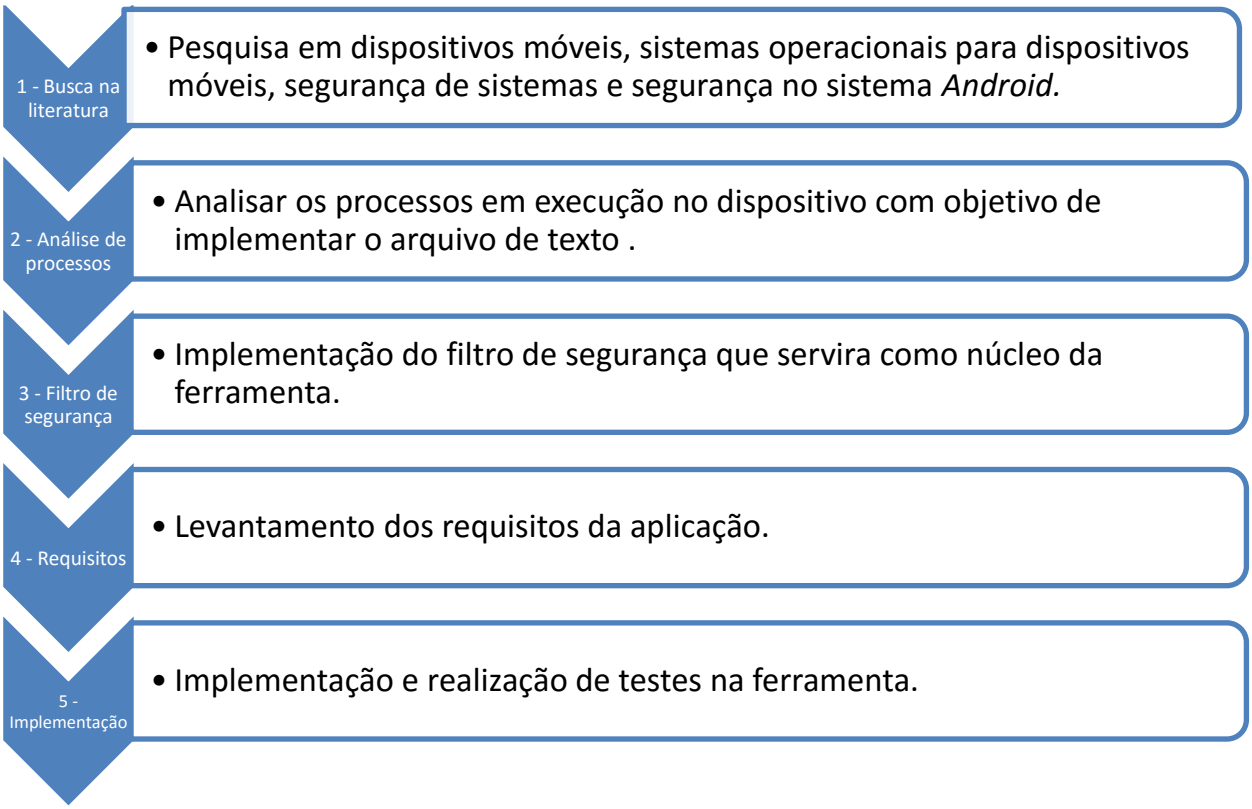
Gil (2008) explica que uma pesquisa cientifica pode ser classificada em 3 tipos:

- Pesquisa Exploratória: Proporciona uma maior familiaridade com o problema. Pode envolver pesquisas bibliográficas e entrevistas com pessoas experientes no assunto;
- Pesquisa Descritiva: Descreve as características de uma determinada população ou fenômeno. Uma das peculiaridades utilizadas está no uso de técnicas padronizadas para realizar uma coleta de dados;
- Pesquisa Explicativa: Identifica os fatores que determinam ou que contribuem para a ocorrência dos fenômenos. É o tipo que mais aprofunda o conhecimento da realidade, porque explica a razão, o porquê das coisas.

A realização deste trabalho teve cunho exploratório e aplicado, pois tem como objetivo proporcionar maior entendimento sobre o tema em estudo e, propor soluções que resolvam os problemas encontrados.

Para a implementação da ferramenta propôs-se uma metodologia para atingir os objetivos específicos que está ilustrada na figura 9:

Figura 9 - Metodologia utilizada



Fonte: Próprio autor (2015)

- A primeira etapa constitui-se de um levantamento literário sobre materiais que possam servir como referência no momento de pesquisa. Considerou-se a afinidade de assuntos, sendo eles: segurança, dispositivos móveis e gerência de processos em sistemas operacionais. A pesquisa foi realizada em acervos já conhecidos na área da computação tais como: IEEE, *ACM Digital Library*, periódicos da CAPES e livros referentes ao assunto;
- A segunda etapa da pesquisa concentrou-se na análise dos processos que são executados em segundo plano no dispositivo móvel. Foram analisados os processos nativos de acordo com o versionamento do sistema operacional. Além dos processos nativos, foram analisados os processos gerados pelas aplicações mais comumente utilizadas. Tal análise levou em consideração o número de downloads realizados na *Google Play Store*;
- O desenvolvimento do filtro de segurança ocorreu na terceira etapa. Para a sua implementação, foi necessário ter conhecimento dos processos nativos do sistema para que estes não sejam afetados pelo filtro de segurança. Além dos

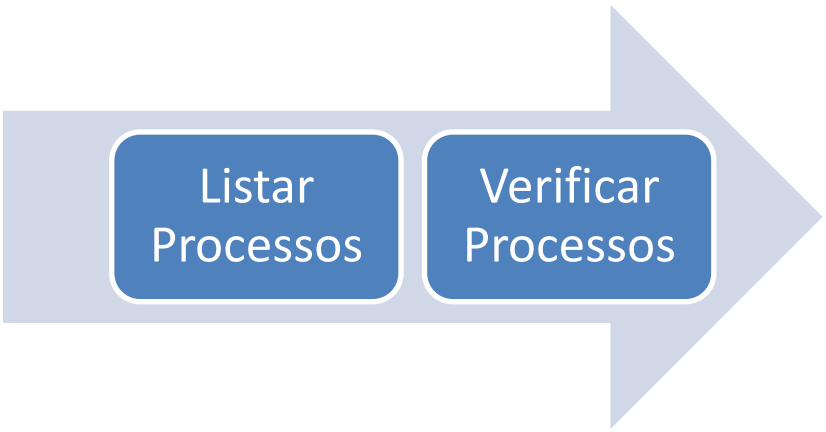
processos nativos, devera ter-se o discernimento das aplicações mais utilizadas pelos usuários e com isso observar os processos gerados;

- A quarta etapa constituiu no levantamento de requisitos da aplicação. Precisou serem analisados quais os objetivos deveriam ser atingidos e quais funções o software deveria executar. Para a aplicação de segurança, o software deveria alertar o usuário caso detecte algum processo potencialmente malicioso e enviar um alerta ao usuário que poderá ou não encerrar tal processo. Caso o usuário deseje ignorar o alerta, o processo será reconhecido como processo conhecido do sistema, caso não, a ferramenta mostrara algumas opções ao usuário. Se por ventura o usuário não tiver pleno conhecimento sobre o processo capturado, será também fazer uma análise de risco do aplicativo que terá um grau de periculosidade como retorno para o usuário;
- A construção da aplicação e a realização de testes ocorreu na quinta etapa do projeto. Para o desenvolvimento será utilizado a *IDE Android Studio* e linguagem de programação Java. A aplicação será desenvolvida utilizando a *Application Programming Interface* (API) 19 (*KitKat* 4.4.2) do sistema operacional por ser a versão mais utilizada atualmente. A sexta e sétima etapas constituíram-se na elaboração dos testes de funcionalidade e na coleta e análise dos resultados obtidos. Para os testes, foi estudada uma métrica para mensurar a complexidade da execução de uma determinada função do software. Tendo como exemplo a ação de encerrar um processo, quantas etapas deverão ser executadas. A fase de coleta de resultados serviu para analisar os resultados obtidos com a execução da aplicação em diversos dispositivos diferentes a fim de detectar algum mau funcionamento da aplicação e/ou problemas de compatibilidades com versões diferentes do sistema operacional Android.

4 SOLUÇÃO PROPOSTA - PROSEC

Esta etapa objetiva a demonstração dos modelos propostos e métodos utilizados na confecção deste estudo. A princípio, elaborou-se um modelo simples no qual era feita a verificação de processos e uma breve filtragem. A figura 10 ilustra o modelo adotado.

Figura 10 – Modelo inicial proposto



Fonte: Próprio autor (2015)

Neste modelo, a ferramenta realiza a listagem dos processos que estão em execução no dispositivo e faz uma breve comparação com um processo que teve o seu nome salvo no código fonte. A princípio, a identificação dos processos é realizada com base no registro de processos maliciosos reconhecidos. Se os nomes dos processos forem iguais, o sistema enviará um alerta ao usuário avisando que o sistema pode estar em perigo. Este modelo tem como limitação o método de comparação de processos.

Em uma nova proposta denominada modelo 2, todos os processos do sistema são comparados com uma lista, onde estão contidos os processos nativos do sistema Android e também com os processos das aplicações com maior número de downloads realizados na *Google Play Store,* além de outros processos capturados em dispositivos reais que serviram como base para armazenar processos gerador por dispositivos de diferentes fabricantes. O *software* desenvolvido com base nestes requisitos, inicialmente realiza a listagem dos processos em execução no dispositivo.

Após a listagem é feita a comparação dos processos com um arquivo de texto que contém apenas processos confiáveis. Esse arquivo foi nomeado como *WhiteList* e, através dele que é feita a comparação com os processos verificados com a ferramenta. Caso todos os processos verificados estejam contidos na *WhiteList*, a ferramenta enviara um aviso ao usuário informando que o dispositivo está seguro. Caso contrário, o usuário será informado que um potencial processo malicioso está em execução no sistema e o mesmo terá 3 opções referentes ao processo capturado:

- O usuário poderá ignorar o processo reconhecido, caso o mesmo reconheça o processo como seguro. Após esta etapa, o processo anteriormente capturado será adicionado automaticamente ao repositório *WhiteList* para que o sistema não o reconheça posteriormente como processo malicioso;
- Outra opção possível ao usuário é quando mesmo não reconhece o processo detectado e deseje desinstalar a aplicação que gere o processo;
- A terceira opção constituiu da exibição dos detalhes referentes aquele processo, tais como nome do pacote de instalação, permissões requeridas pelo processo e análise de risco do aplicativo. Caso o usuário não tenha conhecimento do processo, poderá então fazer uma análise de risco que o processo capturado representa.

A figura 11 ilustra o segundo modelo proposto.

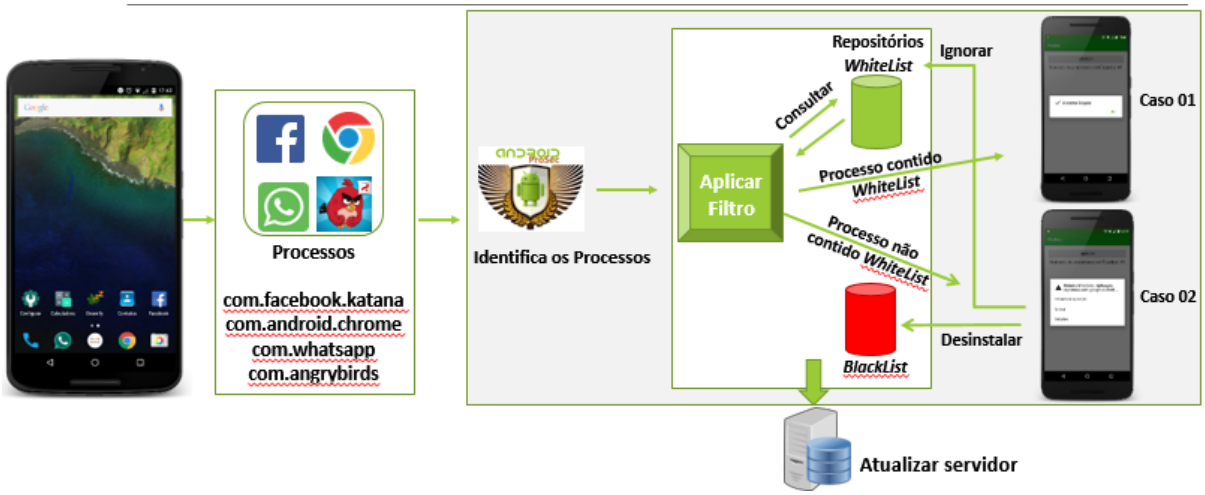
Figura 11 – Segundo modelo proposto

Fonte: Próprio autor (2015)

Ao analisar o segundo modelo proposto, viu-se que o mesmo supria todas as necessidades encontradas no modelo anterior, dentre elas uma verificação mais

completa dos processos em execução. Após a adoção do segundo modelo, a ferramenta proposta foi nomeada ProSec (*Process Security*). A figura 12 demonstra o funcionamento da ferramenta.

Figura 12– Infográfico da ferramenta



Fonte: Próprio autor (2016)

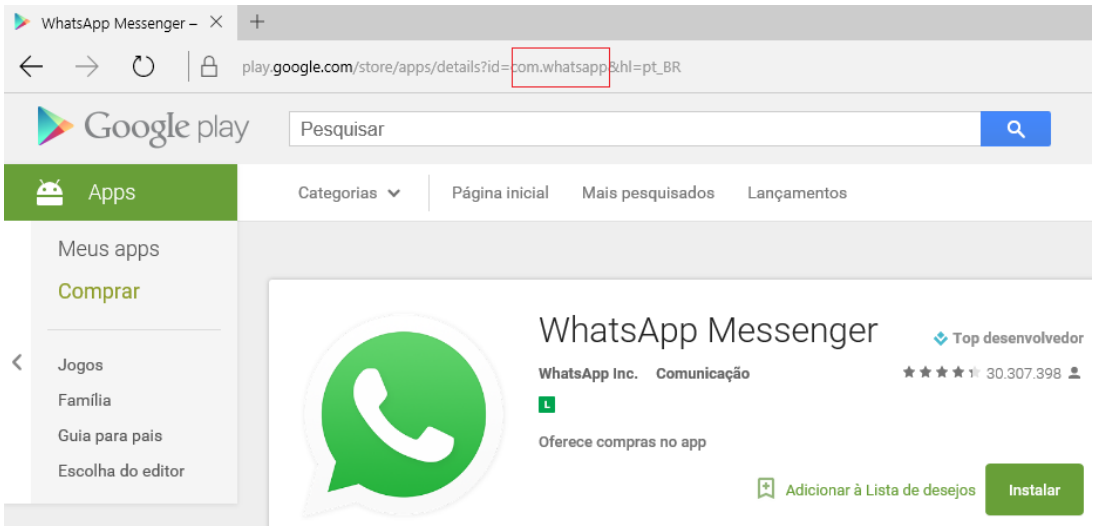
A ferramenta proposta irá analisar os processos que estão em execução e através de um filtro, a mesma ira consultar repositórios e farão a correta classificação do processo. Um processo poderá ser classificado em seguro (gerado por aplicações seguras), ou processos maliciosos (processos gerados por aplicativos que objetivam o roubo de informações do usuário ou danificar o dispositivo).

4.1 Filtro de Segurança

O filtro de segurança é o modulo responsável por armazenar e analisar os processos que estão em execução no sistema. Para a criação do repositório contendo os processos conhecidos, pesquisou-se na loja de aplicativos da *Google* quais os APP que continham maior número de *downloads* realizados. Um fator que facilitou a criação do arquivo *WhiteList* foi a identificação de processo, realizada a partir do reconhecimento da URL do aplicativo, quando pesquisado na *Google Play Store*. A partir da URL é possível extrair o processo gerado pela aplicação, conforme

exemplo demonstrado na figura 13, o qual apresenta a pesquisa do aplicativo *WhatsApp* e o reconhecimento da mesma com a identificação da sua URL.

Figura 13 - Link do aplicativo na *Google Play Store*



Fonte: *Google Play Store* (2015)

Quando comparado ao processo gerado pela aplicação, percebe-se que o nome do processo é exatamente o mesmo quando comparado ao extraído da URL, ou seja, ***com.whatsapp***. A figura 14 ilustra o processo gerado pelo sistema.

Figura 14 - Análise de processos no dispositivo

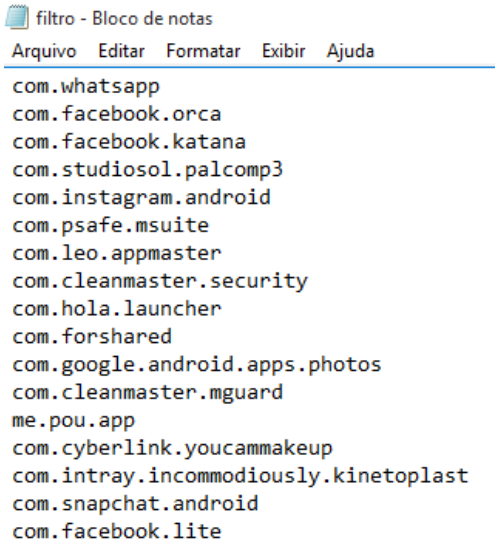
```
15313  0    0% S    77 1687696K  97916K  bg u0_a119  com.whatsapp
```

Fonte: Próprio autor (2015)

Adotando esse modelo de extração de processos, realizou-se uma pesquisa de quais eram os 120 aplicativos (80% dos quais eram listados no *ranking*) mais baixados da *Play Store* (Play Store, 2015) e posteriormente inseriu-se em um arquivo os processos extraídos da URL. Essa foi uma primeira versão do arquivo. A figura 19 demonstra a disposição dos processos no arquivo criado. Na segunda etapa do processo de criação do filtro de segurança implementou-se uma ferramenta auxiliar (denominada SalvaProcessos) para coletar os processos em execução de dispositivos recentemente formatados e prontos para uso. Após a formatação dos

dispositivos e a instalação das aplicações por parte do usuário, executou-se o aplicativo SalvaProcessos a qual gerou como saída um arquivo contendo todos os processos em execução. Este procedimento foi realizado em 7 aparelhos das fabricantes Asus, LG, Motorola, Samsung e Gradiente. Após as coletas concatenou-se os arquivos resultantes, criando único log com todos os processos. A *WhiteList* final ficou constituída das 120 aplicações mais baixadas da *Google Play Store* juntamente com os processos em execução de 7 dispositivos. A *WhiteList* final é constituída por 1909 processos. A figura 15 ilustra uma parte da *WhiteList* final contendo alguns processos seguros.

Figura 15 – Parte da *WhiteList* contendo os processos das aplicações



Fonte: Próprio autor (2015)

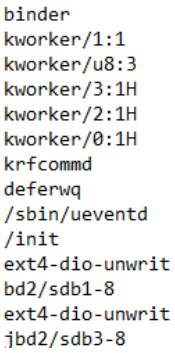
Após a criação do log contendo os processos gerados por aplicações, foram analisados os processos nativos do sistema. A princípio utilizou-se um emulador para capturar os processos em execução. O emulador *Genymotion*[8] permite a completa emulação de um dispositivo Android no computador, além de total integração com IDE de programação *Android* Studio. Foi escolhido por ser o emulador que requisita menos recurso computacional se comparado ao emulador nativo da IDE. Como o emulador utiliza uma versão AOSP[9] do sistema Android foi possível extrair uma grande quantidade de processos gerados pelo sistema, no qual

[8] Disponível em: https://www.genymotion.com/

[9] *Android Open Source Project* – Versão sem customização de fabricantes.

está presente em todos os dispositivos que executem o sistema Android. Na análise, foram consideradas as versões Je*lly Bean, KitKat, Lollipop* e *Marshmallow*. A figura 16 demonstra uma parte dos processos extraídos do emulador.

Figura 16 - *WhiteList* contendo os processos nativos do sistema



Fonte: Próprio autor (2015)

Devido ao número de versões ativas do sistema Android, foi necessário realizar a coleta de processos em execução em diferentes dispositivos de diferentes fabricantes para garantir uma maior confiabilidade nos resultados obtidos.

4.2 Funcionamento

Ao iniciar a execução da ferramenta, o filtro de segurança analisará os processos e os classificarão de acordo com o arquivo de texto implementado no projeto. Inicialmente serão dois arquivos: um contendo os processos classificados como seguros que serão armazenados sobre o título de *WhiteList* e outros processos extraídos de fóruns de segurança que ficarão classificados como não seguros que serão armazenados sobre o título de *BlackList*. Ao iniciar a execução da ferramenta, o sistema verificará a existência do arquivo de texto com os processos seguros, caso seja a primeira vez que a aplicação é executada em um aparelho, um arquivo de texto contendo todos os processos classificados como seguros é baixado de um servidor, caso o arquivo já exista, este processo é ignorado. O arquivo contendo as aplicações maliciosas é criado no momento em que a ferramenta capturar um processo classificado como malicioso, ou seja, que não esteja no arquivo de texto de dados contendo os processos classificados como seguro. Caso o usuário opte por remover aquela aplicação capturada pela

ferramenta, o nome do processo capturado é salvo neste repositório para que futuramente o sistema impeça a instalação daquela aplicação. O processo de criação do arquivo de texto com processos maliciosos é feito somente uma vez. Nas próximas execuções, é feita a verificação da se tal arquivo de texto já está criado. Caso positivo, a ferramenta só fará a inserção dos processos no banco existente. A figura 17 ilustra a disposição do filtro de segurança.

Figura 17 - Disposição do filtro de segurança

Fonte: Próprio autor (2015)

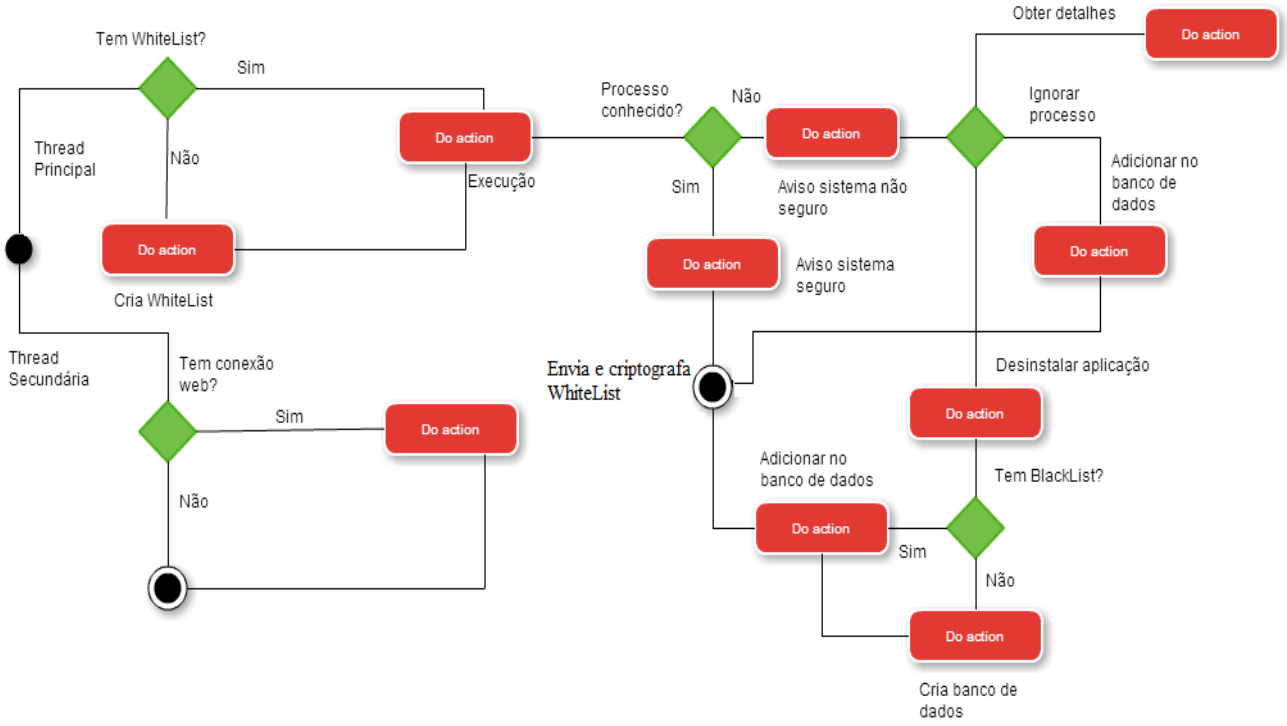
Caso o filtro capture um processo que não esteja na *WhiteList* que contém os processos classificados como seguros ou que capture algum processo que esteja incluso na *WhiteList* com processos maliciosos, o sistema irá enviar um alerta ao usuário que terá como opções:

- Desinstalar a aplicação que gerou aquele processo;
- Ignorar o processo caso o mesmo seja proveniente de uma aplicação que o usuário instalou manualmente, porém não esteja inserido na *WhiteList*;
- Exibir detalhes sobre o processo capturado.

Ao selecionar a opção de remoção da aplicação, será enviado um comando para encerrar o processo capturado e posteriormente a desinstalação da aplicação. O primeiro comando é necessário, pois, caso o processo capturado tenha privilégios de administrador, o sistema fica incapaz de desinstalar a aplicação. Quando selecionada a opção de ignorar o processo, a ferramenta incluirá o mesmo na *WhiteList* no qual estão contidos os processos classificados como seguros. Tal procedimento será feito para que o filtro de segurança não detecte novamente este processo. Ao selecionar a opção de exibição de detalhes do processo, o usuário pode verificar todas as permissões que a aplicação demanda para correto funcionamento. Ao selecionar uma determinada permissão, será exibido um breve resumo sobre a mesma. A segunda opção permite que o usuário acesse a aplicação capturada diretamente na loja de aplicativos *Google Play Store*, obtendo assim mais informações sobre a aplicação capturada e opiniões dos usuários. A terceira opção é constituída da análise de riscos do processo capturado. Criou-se uma *thread* secundária que fica encarregada de verificar se o dispositivo possui acesso à *web.* Caso o aparelho esteja conectado, será feito o *download* de uma *WhiteList* e *BlackList* mais atuais armazenadas em um servidor e posteriormente será feito o *upload* da *WhiteList* e *BlackList* para o servidor juntamente com a realização da criptografia da *WhiteList*. O diagrama de atividades da ferramenta está representado na figura 18.

Para desenvolver a apresentação do modelo proposto, desenvolveu-se a UML da ferramenta. Segundo Furlan (1998), os diagramas UML possuem uma notação padrão e bastante compreensível que permite abstrair certos aspectos do sistema proposto, facilitando o entendimento do mesmo.

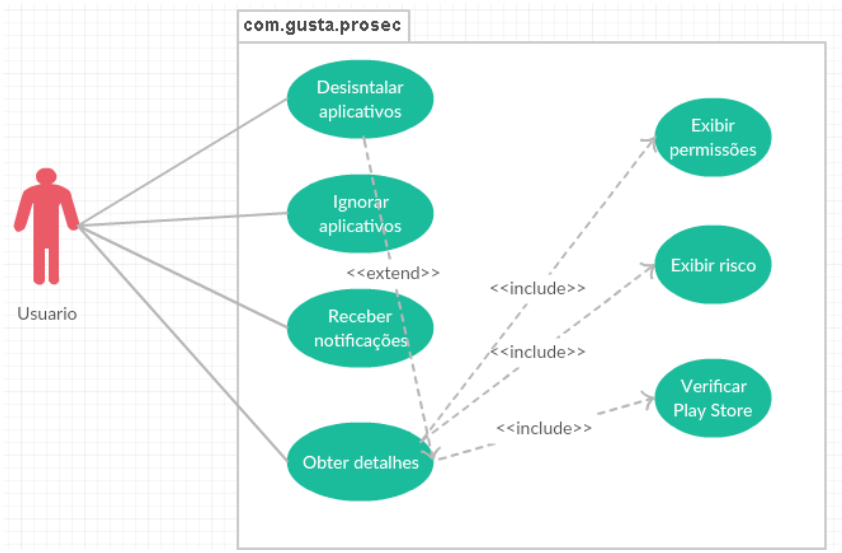
Figura 18 – Diagrama de atividades



Fonte: Próprio autor (2015)

As funcionalidades do ProSec estão descritas na figura 19 que ilustra os casos de uso. Em princípio a aplicação fornecerá ao usuário a opção de listar os processos que estão em execução no dispositivo, ignorar o processo capturado caso julgue necessário e exibir os detalhes do processo. A última opção disponibiliza ao usuário a alternativa de listar as permissões requeridas pelos processos, a opção de desinstalar a aplicação e opção de procurar a aplicação na *Google Play Store*

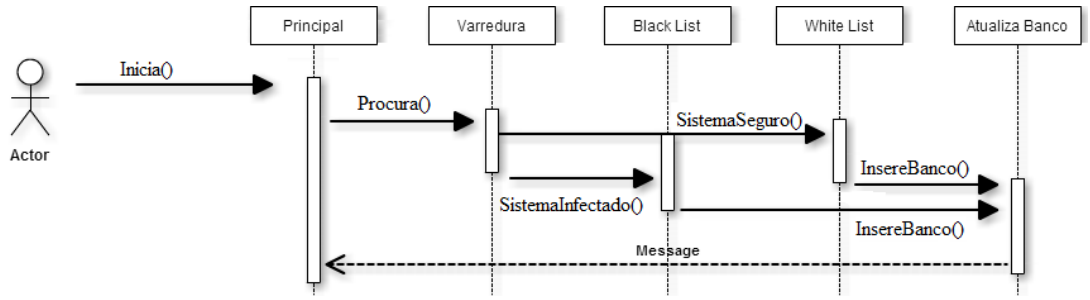
Figura 19 – Casos de uso



Fonte: Próprio autor (2015)

A figura 20 mostra o diagrama de sequência da aplicação.

Figura 20 – Diagrama de Sequência



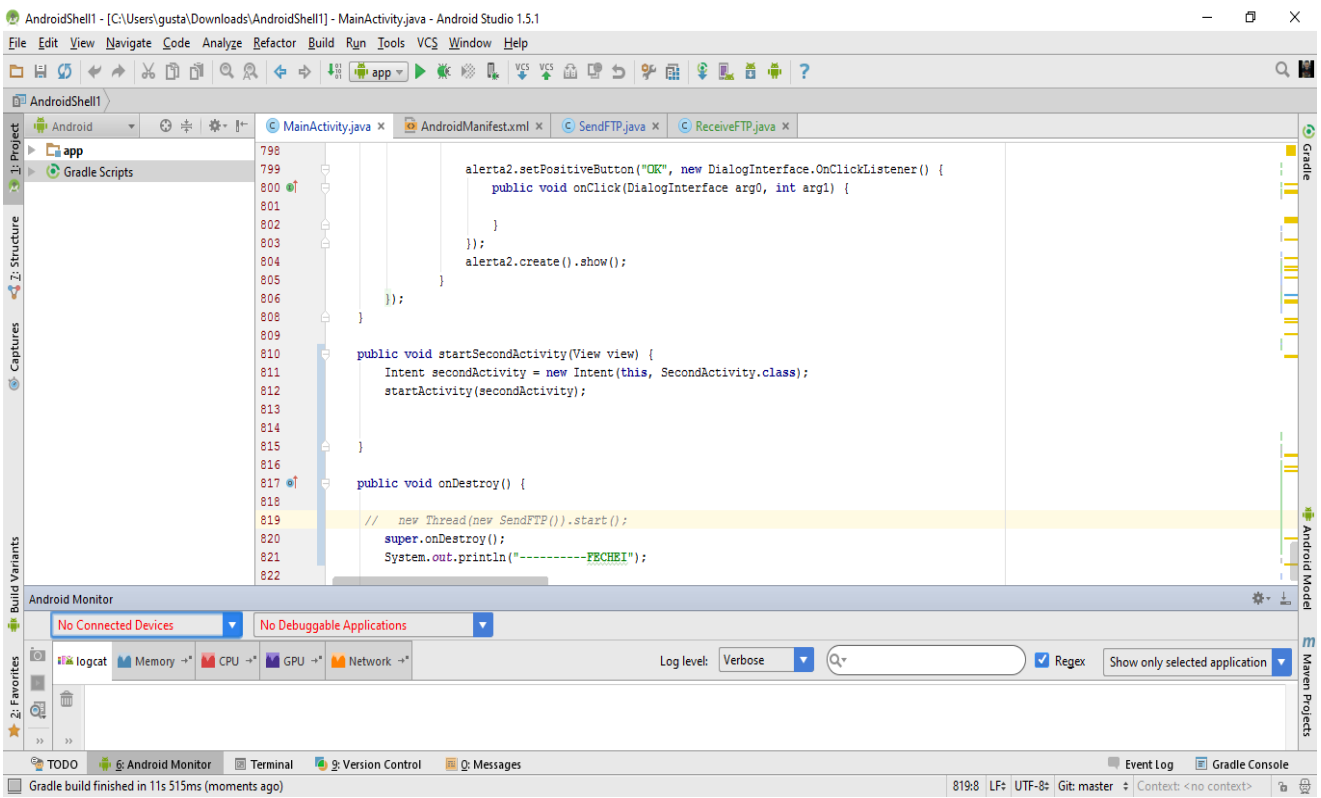
Fonte: Próprio autor (2015)

Ao iniciar a aplicação, o usuário inicia a varredura do sistema e caso o processo seja reconhecido como seguro ou maligno, o mesmo é inserido em seu respectivo arquivo de log. Após a finalização da varredura, é feito o processo de atualização da *WhiteList* por meio de um servidor *File Transfer Protocol (*FTP).

4.3 Implementação

A implementação da ferramenta foi feita utilizando a linguagem de programação Java, a qual tem suporte nativo por parte do sistema Android e a interface gráfica foi implementada utilizando a linguagem *eXtensible Markup Language* (XML) para maior compatibilidade com versões mais antigas do sistema e por demandar menos recursos de processamento. Escolheu-se a IDE *Android Studio*[10] que está ilustrada na figura 21. Utilizou-se a versão 2.1 e esta IDE foi a escolhida por ter sido feita pela própria *Google*, com isso a ferramenta já vem pronta para que qualquer desenvolvedor comece a programar seus aplicativos. Dentre as IDE's *NetBeans* e *Eclipse*, se fazia necessário a instalação do *Software Development Kit* (SDK), e isso por sua vez poderia ocasionar mau funcionamento proveniente de passos errados durante a sua instalação.

Figura 21 - IDE *Android Studio*



Fonte: Próprio autor (2015)

10 - Disponível em: https://developer.android.com/sdk/index.html

Juntamente com a IDE de desenvolvimento, utilizou-se o emulador *Genymotion* que permitiu que a transferência da aplicação fosse feita de maneira mais eficiente, sem precisar de um dispositivo real para realizar os testes da aplicação. A conexão se dava por meio do protocolo *Android Debug Bridge* (ADB).

Com o ambiente de programação pronto para iniciar o desenvolvimento, foram analisadas maneiras de extrair os processos que estavam em execução no dispositivo. Para realizar a listagem dos processos, foram utilizadas estruturas que permitem a execução de comandos diretamente no *shell* do sistema Android. Como o sistema é oriundo do sistema *Linux*, utilizou-se o comando *top* para listar os processos em execução. O processo é iniciado quando o usuário clica no botão nomeado *btnVerifica*. A função responsável pelo método recebeu o nome de Executa e retorna uma *string* a qual é passada para o *textArea* para ser visualizada pelo usuário.

Para que a verificação de processos fosse feita, utilizou-se o método de comparação de processos exibidos com o arquivo que contém os processos seguros. Ao iniciar a função Executa, todo conteúdo da *WhiteList* e do *textArea* são copiados para um *ArrayList* para que as comparações sejam feitas. Quando ocorre uma comparação e a ferramenta identifica que um determinado processo apareceu no *textArea* e o mesmo não estava contido na *WhiteList*, uma variável *booleana* tem seu valor alterado para *true* (verdadeiro) e o processo capturado é salvo em um terceiro *ArrayList*. Este *ArrayList* será utilizado para armazenar os processos classificados como potencialmente maliciosos. O processo se comparação está demonstrado na figura 22.

Figura 22 – Comparação de processos

```
ArrayList<String> virusList = new ArrayList<~>();
for (int i = 0; i < textView.size(); i++) {
    boolean NaoExiste = true;
    for (int ii = 0; ii < texto.size(); ii++) {
        if (textView.get(i).equals(texto.get(ii))) { // comparação de saida do textView com whiteList
            NaoExiste = false;
            teste = false;
            // break; //termina o loop interior
        }
    }
    if (NaoExiste) {
        teste = true;
        virusList.add(textView.get(i)); //Adiciona à lista de virus
        break;
    }
}
```

Fonte: Próprio autor (2015)

Para sinalizar ao usuário a identificação de um processo suspeito, implementou-se uma classe que permite o envio de notificações. O sistema informa se foi encontrado um processo suspeito caso a variável booleana esteja em um estado *true* (verdadeiro) ou se o sistema estiver seguro caso a variável booleana estiver em estado *false* (falso). É emitido o alerta na forma sonora e visual e através de dispositivos executando o sistema *Android Wear*.

Ao identificar um processo suspeito, uma rotina específica disponibiliza ao usuário um conjunto de ações a serem realizadas para que o usuário determine qual ação tomar quanto ao processo capturado. Como explicado, o usuário poderá ignorar o processo, obter detalhes do processo ou desinstalar a aplicação geradora do processo capturado. Caso o usuário decida ignorar o processo capturado, o sistema carrega o arquivo *WhiteList* e insere o processo no arquivo para que a ferramenta não venha alarmar o usuário. Para a opção de remoção do processo, são utilizados pacotes que utilizam o nome do pacote da aplicação capturada para desinstalar a aplicação. A figura 23 ilustra este procedimento. Ao desinstalar a aplicação capturada, o sistema verifica se a *BlackList* existe no dispositivo. Caso o arquivo não exista, é feito a criação do mesmo e posteriormente a inclusão do nome do processo desinstalado. Caso o arquivo exista, somente a parte de inserção é realizada.

Figura 23 – Método: Desinstalação da aplicação

```
try {
    Uri packageURI = Uri.parse("package:" + ameaca);
    Intent uninstallIntent = new Intent(Intent.ACTION_DELETE, packageURI);
    startActivity(uninstallIntent);
} catch (Exception e) {
    Toast.makeText(MainActivity.this, "Não foi remover o processo pois o mesmo é do sistema", Toast.LENGTH_LONG).show();
}
// salva aplicativo desinstalado na lista
final File fileBlack = new File(getFilesDir().getPath() + "/BlackList.txt");
if (!fileBlack.exists()) {
    try {
        fileWriter = new FileWriter(fileBlack, true);
        fileWriter.append(ameaca);
        fileWriter.append("\n");
        fileWriter.flush();
    } catch (Exception e) {

    } finally {
        if (fileWriter != null) {
            try {
                fileWriter.close();
```

Fonte: Próprio autor (2015)

A opção de “Obtenção de detalhes” disponibiliza ao usuário o recurso “Exibição permissões” que o mesmo requisita ao sistema, de abrir na *Google Play Store* a página referente ao processo capturado ou realizar uma análise de risco da aplicação a qual ira atribuir um peso a cada permissão requisitada pelo aplicativo. A opção de “Exibição Permissões” fornece ao usuário todas as permissões requisitadas pelo processo. O sistema exibe estas permissões em uma *ListView* e ao clicar sobre determinada posição, é exibido brevemente na tela uma rápida explicação sobre determinada permissão. Este processo é ilustrado na figura 24 (*MANIIFEST PERMISSION ANDROID*, 2015).

Figura 24 – Método: Exibição de permissões

```
PackageInfo pack = getPackageManager().getPackageInfo(ameaca, PackageManager.GET_PERMISSIONS);
PackageManager pm = getPackageManager();
ApplicationInfo galleryInfo = pm.getApplicationInfo(ameaca, PackageManager.GET_META_DATA);

final String[] requestedPermissions = pack.requestedPermissions;
AlertDialog.Builder alertDialog = new AlertDialog.Builder(MainActivity.this);
LayoutInflater inflater = getLayoutInflater();
View convertView = (View) inflater.inflate(R.layout.custom, null);
alertDialog.setView(convertView);

final ListView lv = (ListView) convertView.findViewById(R.id.listView1);
ArrayAdapter<String> adapter = new ArrayAdapter<>(MainActivity.this, android.R.layout.simple_list_item_1, requestedPermissions);
lv.setAdapter(adapter);
```

Fonte: Próprio autor (2015)

O módulo de abertura do processo "*Google Play Store"* também faz uso de pacotes específicos o qual utiliza do nome do processo capturado e executa rotinas para abrir a loja de aplicativos. A etapa de abertura da loja de aplicações está ilustrada na figura 25 (ANDROID, 2015). Esta implementação foi pensada a partir da situação em que um processo malicioso é capturado e o usuário deseja obter mais informações sobre a aplicação. Abrindo diretamente na *Play Store*, o usuário pode conferir os comentários de outros usuários referentes a aquela aplicação especifica.

Figura 25 – Método: Integração com *Google Play Store*

```
case 1: // ABRE APP NA PLAY STORE
    String url = "https://play.google.com/store/apps/details?id=" + ameaca + "&hl=pt_BR";
    Intent i = new Intent(Intent.ACTION_VIEW);
    i.setData(Uri.parse(url));
    startActivity(i);
```

Fonte: Próprio autor (2015)

Reconhecendo que a ferramenta pode capturar processos, sobre os quais o usuário não tenha conhecimento, implementou-se um módulo que realiza a análise de risco das permissões concedidas aos processos, baseado na proposta de Jacobsen (2014). Este recurso exibe uma mensagem informando ao usuário o nível de risco (baixo, médio ou alto) que o processo representa para o seu dispositivo. Para o funcionamento deste módulo (avaliação de riscos), a aplicação avalia as permissões mais comuns entre *malwares* e no final atribui um peso para cada permissão requisitada. Dentre as permissões mais perigosas encontravam-se as

permissões de acesso à Internet, obter o *status* da conexão Wi-Fi e conexão do telefone. Ao final da análise das permissões, os riscos de cada permissão eram somados e avaliados. O estudo de Jacobsen (2014) mostrou que ao analisar um conjunto de *malwares*, as permissões requisitadas por esta aplicação geravam um índice de 34 pontos. Portanto, reconhecendo este perfil é possível afirmar que um processo capturado com risco maior ou igual a 34 pontos deve ser classificado com um alto risco. As permissões e seus pesos analisados pelo modulo de avaliação de riscos são apresentadas na tabela 2.

Tabela 2 - Permissões e pesos atribuídos

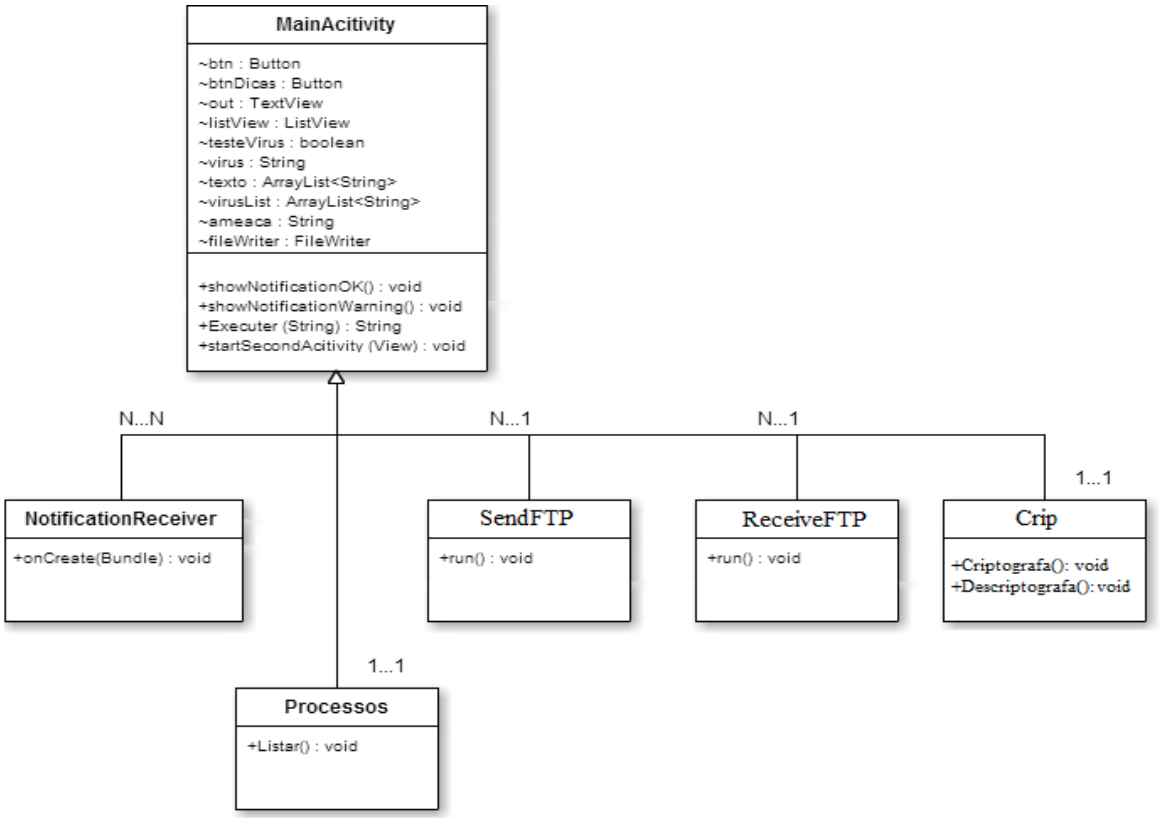
Permissão	**Peso**
ACCESS_LOCATION_EXTRA_COMMANDS	1
CAPTURE_AUDIO_OUTPUT	1
GET_TASKS	1
CAMERA	1
READ_CALL_LOG	1
DISABLE_KEYGUARD	2
WRITE_APN_SETTINGS	2
RESTART_PACKAGES	2
READ_CONTACTS	2
ACCESS_FINE_LOCATION	2
ACCESS_COARSE_LOCATION	2
WRITE_SMS	3
WRITE_SETTINGS	3
WRITE_CONTACTS	3
VIBRATE	3
RECEIVE_SMS	4
SEND_SMS	4
CALL_PHONE	4
READ_SMS	5
WRITE_EXTERNAL_STORAGE	5
ACCESS_WIFI_STATE	5
INTERNET	6
ACCESS_NETWORK_STATE	6
READ_PHONE_STATE	6

Fonte: Jacobsen (2014)

Ao final da análise do processo capturado, a ferramenta alertará o usuário sobre o risco da aplicação. Caso esta tenha um risco alto, o ProSec disponibiliza a opção de desinstalar o *software*, recurso fundamental para manter a integridade do sistema.

A seguir, a figura 26 nos mostra o diagrama de classes da ferramenta. A classe *MainActivity* é a classe principal do programa. A classe Processos é responsável pela criação da *WhiteList* caso a mesma não exista. A classe *NotificationReceiver* é responsável pela implementação do sistema de notificações em caso de sistema vulnerável ou sistema seguro. As classes *SendFTP* e ReceiveFTP são responsáveis por toda parte de conexão com o servidor. A classe Crip é a responsável por realizar a criptografia e descriptografia do arquivo *WhiteList* e *BlackList.*

Figura 26 – Diagrama de Classes

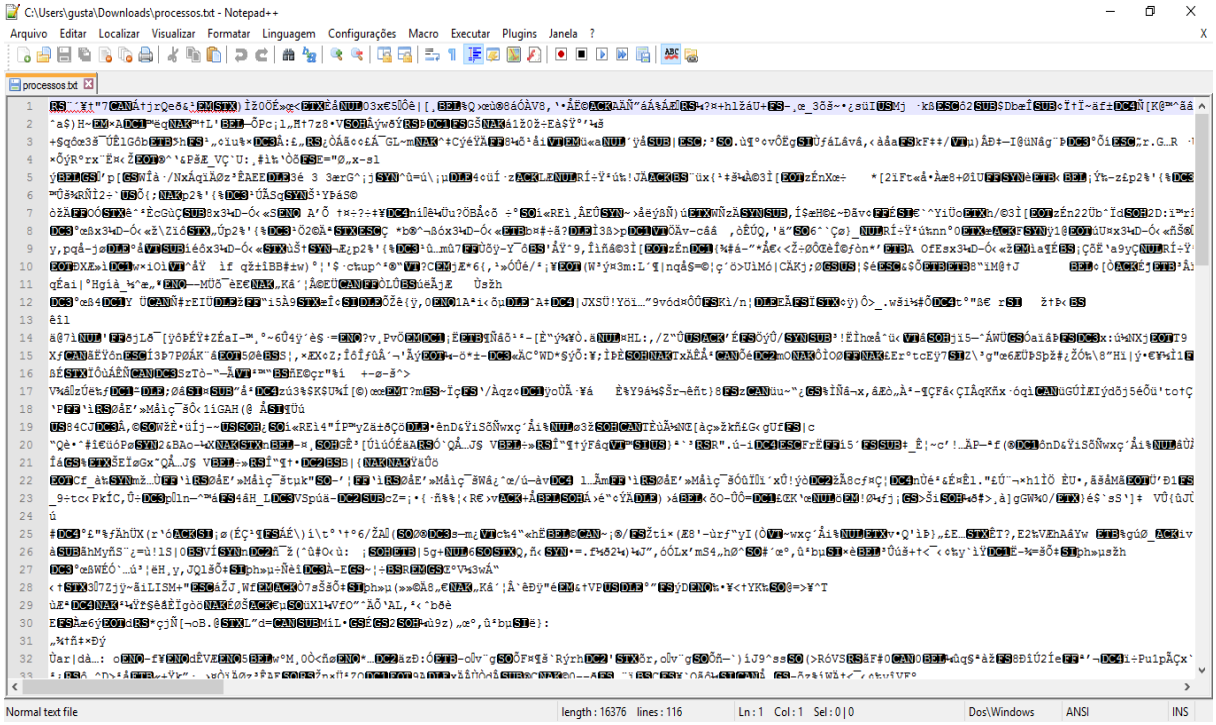


Fonte: Próprio autor (2015)

Para o envio do repositório ao servidor, será feito a encriptação desse arquivo. Utilizou-se o método DES (*Data Encryption Standard*) o qual se mostrou a

melhor opção quando comparado com o método AES (*Advanced Encryption Standard*) quando analisado o tempo para realizar a criptografia do arquivo. Ao receber o arquivo de log, o servidor *FTP* deverá ser capaz de descriptografar o arquivo recebido, analisar o seu conteúdo, criar um novo arquivo com os processos, e posteriormente enviar para os aparelhos já criptografados. A figura 27 ilustra o arquivo contendo os processos depois do procedimento de criptografia.

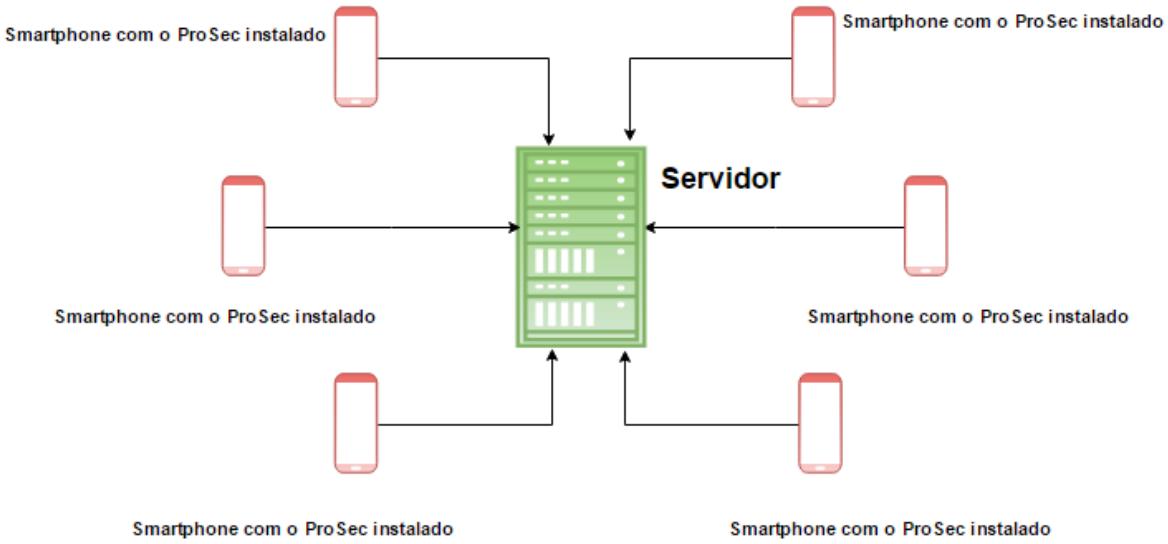
Figura 27 – *WhiteList* criptografado



Fonte: Próprio autor (2015)

Criou-se uma classe *SendFTP e ReceiveFTP* que implementa a funcionalidade *Web* a ferramenta. O intuito é ter um sistema cliente servidor no qual todos os dispositivos que executem a ferramenta ProSec façam o envio de seus arquivos contendo os processos capturados. Ao receber essas informações, um servidor irá reunir todos os processos recebidos em dois arquivos denominados *WhiteList* e *BlackList*. Posteriormente o servidor irá reenviar esses arquivos para todos os dispositivos com o intuído de aumentar a base de dados e assim manter a ferramenta atualizada. O procedimento descrito está ilustrado na figura 28.

Figura 28 – Modelo cliente servidor



Fonte: Próprio autor (2015)

Atualmente a ferramenta está apta para enviar os processos para o servidor e receber um novo arquivo atualizado, por meio do protocolo FTP. A figura 29 ilustra o procedimento de *upload* da WhiteList e *BlackList* contido no servidor.

Figura 29 – Conexão com servidor FTP para *upload* da WhiteList e *BlackList*

```
FTPClient ftp = new FTPClient();
try {

    String diretorio = dir + "/WhiteList.txt";
    String diretorioCrip = dir+"/White.txt";
    String diretorioBlack = dir+"/BlackList.txt";
    String nomeArquivo = "WhiteList.txt";
    String nomeArquivoCrip = "White.txt";
    String nomeArquivoBlack = "BlackList.txt";
    ftp.connect("host", 21);
    ftp.login("user", "pass");
    InputStream arqEnviar = new FileInputStream(diretorio);
    InputStream arqEnviarCrip = new FileInputStream(diretorioCrip);
    InputStream arqEnviarBlack = new FileInputStream(diretorioBlack);
    ftp.setFileTransferMode(ftp.BINARY_FILE_TYPE);
    ftp.setFileType(FTPClient.BINARY_FILE_TYPE);
    ftp.enterLocalPassiveMode();
    ftp.storeFile(nomeArquivo, arqEnviar);
    ftp.storeFile(nomeArquivoCrip, arqEnviarCrip);
    ftp.storeFile(nomeArquivoBlack, arqEnviarBlack);
    ftp.logout();
    ftp.disconnect();
```

Fonte: Próprio autor (2015)

A figura 30 demonstra o procedimento de *download* do WhiteList e *BlackList* para o servidor.

Figura 30 – Conexão com servidor FTP para *download d*a WhiteList *e* BlackList

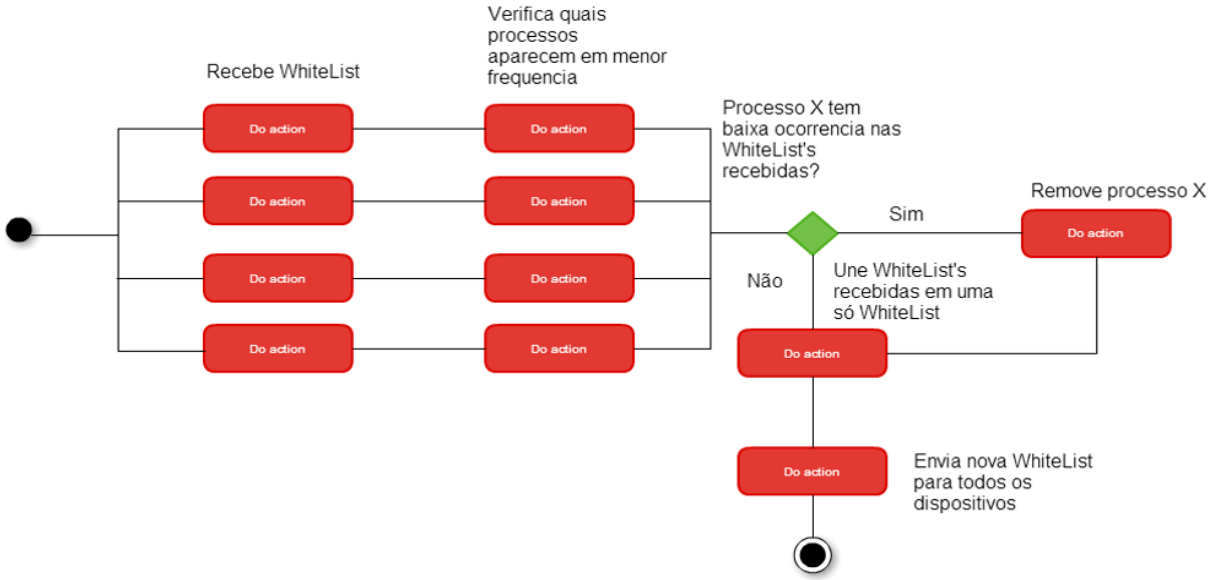
```
FTPClient ftpClient = new FTPClient();
try {

    ftpClient.connect(server, port);
    ftpClient.login(user, pass);
    ftpClient.enterLocalPassiveMode();
    ftpClient.setFileType(FTP.BINARY_FILE_TYPE);
    String WhiteList = "/WhiteList.txt";
    File downloadWhiteList = new File(dir+"/WhiteList.txt");
    OutputStream outputStream1 = new BufferedOutputStream(new FileOutputStream(downloadWhiteList));
    boolean success = ftpClient.retrieveFile(WhiteList, outputStream1);
    String WhiteCrip = "/WhiteList.txt";
    File downloadWhiteCrip = new File(dir+"/WhiteList.txt");
    OutputStream outputStream2 = new BufferedOutputStream(new FileOutputStream(downloadWhiteCrip));
    boolean success2 = ftpClient.retrieveFile(WhiteCrip, outputStream2);
    String BlackList = "/WhiteList.txt";
    File downloadBlackList = new File(dir+"/WhiteList.txt");
    OutputStream outputStream3 = new BufferedOutputStream(new FileOutputStream(downloadBlackList));
    boolean success3 = ftpClient.retrieveFile(BlackList, outputStream1);
```

Fonte: Próprio autor (2015)

Para realizar a conexão, são passados como parâmetros o IP e porta do servidor, nome de usuário e senha. Por ser uma classe para validar a conexão, futuramente será feito uso de uma API para o envio da senha criptografada para o servidor. A partir disso, é necessário ter em mãos o diretório em que o arquivo contendo os processos seguros está salvo e o nome do mesmo. O procedimento é executado em uma *thread* secundária que é iniciada juntamente com a aplicação. A figura 31 ilustra o diagrama de atividades do servidor *Web.* Para evitar uma falha na ferramenta quando o dispositivo não possuir acesso à *web*, criou-se uma condição na qual é verificada se o aparelho possui ou não uma conexão válida. Caso positivo, é feita a conexão com o servidor para o recebimento da *WhiteList*, caso contrário inclui-se uma *WhiteList* dentro do pacote da aplicação que servirá no caso em que o aparelho não tiver acesso à Internet. Esse arquivo poderá ser atualizado quando a própria aplicação receber *updates* caso a mesma seja enviada para a loja de aplicativos da *Google*.

Figura 31 – Diagrama de atividades do servidor *Web*



Fonte: Próprio autor (2015)

Para evitar que o atacante crie uma aplicação maliciosa e manualmente a insira como processo confiável, será implementado futuramente no servidor FTP uma verificação nos arquivos recebidos que consiste em analisar as ocorrências dos processos nas *WhiteList's* recebidas. Se um processo aparece em uma única

WhiteList ou em um número reduzido de arquivos, o processo é ignorado e não será incorporado na *WhiteList* final.

5 VALIDAÇÃO DO APLICATIVO E ANÁLISE DE RESULTADOS

Esta etapa tem como objetivo a realização de testes da ferramenta proposta visando validar as suas funcionalidades. Foram analisados os requisitos funcionais e usabilidade da ferramenta. A tabela 3 apresenta os dispositivos utilizados e quais aplicações estavam instaladas nos aparelhos.

Tabela 3 - Dispositivos e aplicações utilizadas

Dispositivo	**Aplicativos**
Motorola Moto X Style (Android 6.0)	*Facebook, WhatsApp, Chrome, Wolfram Alpha, Google Tradutor, AirDroid, Gmail, Google Play Store*, ProSec
Gradiente Tegra Note (Android 5.1.1)	*Google Play Music, Google Tradutor, VLC, Spotify, UC Browser, Bit Torrent* e ProSec
Sistema Android emulado (Android 4.4.2)	Aplicativos nativos do sistema

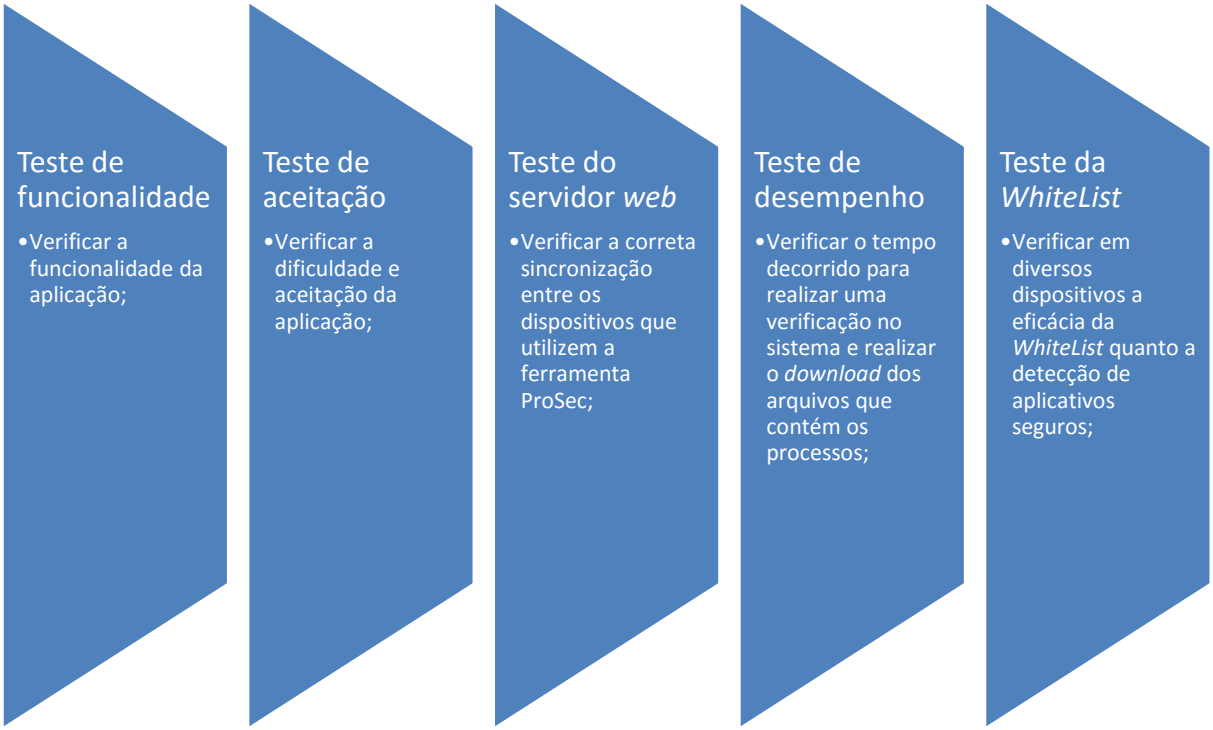
Fonte: Próprio autor (2016)

A realização de testes neste projeto seguiu conceitos de testes de softwares propostos por ROCHA (2001). Rocha definiu os quatro principais níveis de testes de *softwares*. São eles:

- Teste de Unidade: Tem por objetivo explorar a menor unidade do projeto, procurando provocar falhas ocasionadas por defeitos de lógica e de implementação em cada módulo, separadamente;
- Teste de Integração: avalia o software em busca de falhas por meio da utilização do mesmo, como se fosse um usuário final;
- Teste de Aceitação: simulam operações de rotina do sistema de modo a verificar se seu comportamento está de acordo com o solicitado;
- Teste de Regressão: Consiste em se aplicar, a cada nova versão do software ou a cada ciclo, todos os testes que já foram aplicados nas versões ou ciclos de teste anteriores do sistema.

Dentre os testes apresentados, foram selecionados os modos que mais se alinham ao projeto, sendo escolhidos os Testes de Aceitação e Integração. Nós os testes de integração foram inclusos os testes descritos na figura abaixo.

Figura 32 – Testes realizados



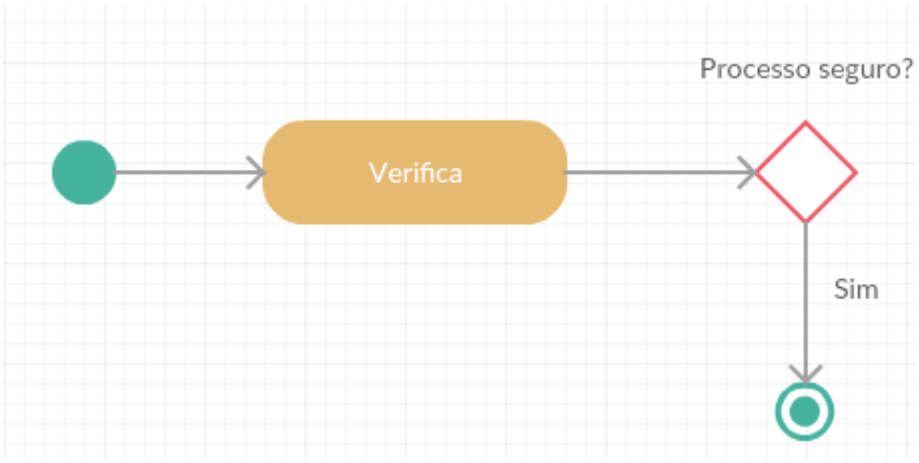
Fonte: Próprio autor (2016)

5.1 Teste funcional

Testes funcionais também conhecidos como “teste de caixa preta” são técnicas nas quais se ignora o comportamento interno do *software* e somente são analisadas as saídas geradas pelo mesmo (MALDONADO, 2014). O primeiro teste funcional proposto teve como objetivo testar a verificação ocorrida em um sistema que não tivesse instalado nenhum aplicativo considerado malicioso. Com isso, espera-se um sistema que tenha como entrada nenhum aplicativo considerado

malicioso e apresente uma saída, informando ao usuário um aviso de sistema seguro. A figura 33 apresenta a rotina do teste proposto.

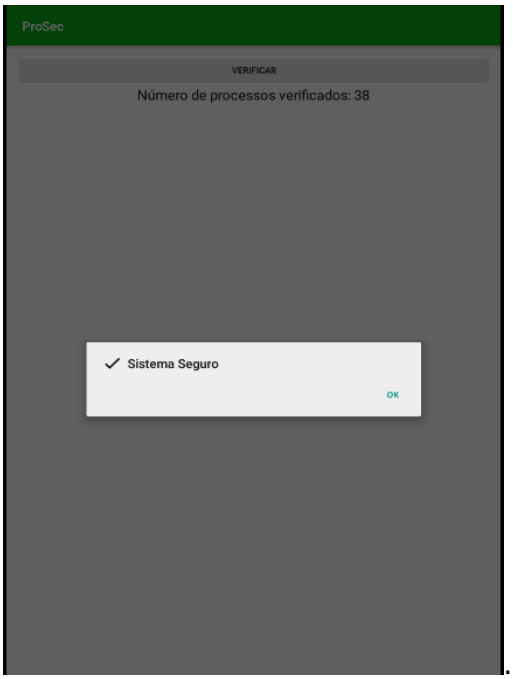
Figura 33 – Rotina do teste proposto



Fonte: Próprio autor (2016)

Como esperado, a ferramenta não capturou nenhum processo potencialmente perigoso e emitiu corretamente o aviso de sistema seguro. O aviso está ilustrado na figura 34.

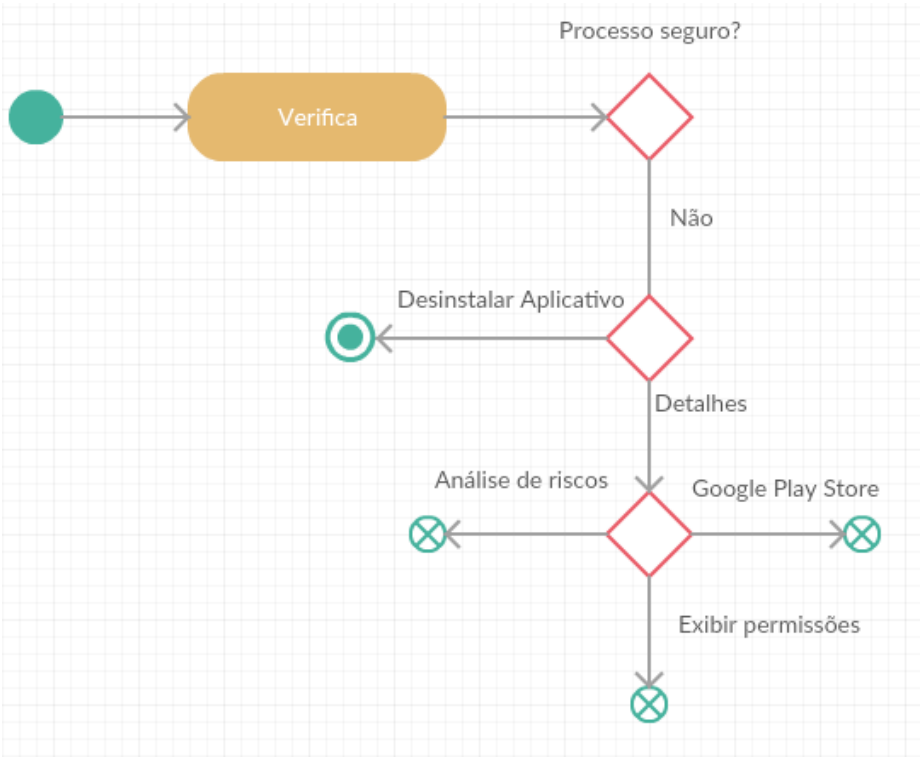
Figura 34 – Aviso de sistema seguro



Fonte: Próprio autor (2016)

A segunda rodada de testes consistiu em um ambiente emulado no qual inseriu-se um aplicativo malicioso. Conforme Wyatt (2011) relatou, novos *malwares* foram encontrados na loja de aplicativos *Google Play Store* naquele ano, entre elas uma aplicação na qual é possível desenhar na tela do aparelho. A aplicação em si foi nomeada *Paint Master* e estima-se que a mesma infectou cerca de 120 mil dispositivos. Inseriu-se essa aplicação manualmente no sistema emulado e posteriormente esta foi instalada e executada. A figura 35 ilustra o diagrama da segunda rodada de testes propostos.

Figura 35 – Segunda rodada de testes realizados



Fonte: Próprio autor (2016)

Para testar a eficácia da ferramenta ProSec quanto a detecção de aplicativos potencialmente maliciosos, temos como a entrada um aplicativo que não esteja na lista de processos confiáveis e a saída esperada é um aviso para o usuário de que seu dispositivo pode estar vulnerável. Para demonstrar todas as possíveis opções com a ferramenta ProSec, a cada nova execução de uma determinada funcionalidade da ferramenta, a aplicação foi encerrada e executada novamente e por isso é apresentando uma quantidade diferente de processos analisados. A figura 36 mostra o aviso enviado ao usuário.

Figura 36 – Aviso de sistema em perigo

Fonte: Próprio autor (2016)

Conforme apresentado, o sistema conseguiu de forma correta identificar um processo que estava em execução no dispositivo, e que não estava incluso na lista de processos confiáveis do sistema. O próximo passo constituiu na obtenção de detalhes do processo capturado, o qual consistiu em analisar as permissões requeridas pelo aplicativo capturado e obter uma análise de risco daquele aplicativo. Os detalhes estão mostrados na figura 37.

Figura 37 – Obtenção de detalhes

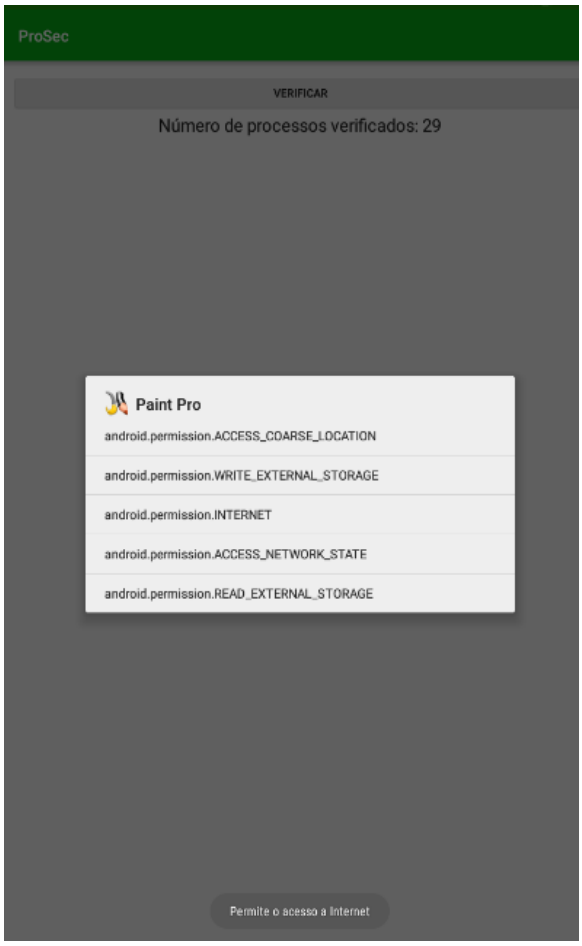
Fonte: Próprio autor (2016)

Ao selecionar a opção de detalhes, o usuário pode escolher dentre as seguintes opções:

- Permissões**:** Lista as permissões requisitas pelo aplicativo;
- *Play Store:* Abrir o aplicativo capturado diretamente na *Google Play Store;*
- Risco: Analisa o risco da aplicação capturada.

Ao escolher a opção de permissões, o usuário é informado de todas as permissões que o aplicativo requer para seu funcionamento e ao selecionar cada uma das permissões é exibido no rodapé do aparelho uma breve informação sobre a permissão selecionada. Na situação a seguir, selecionou-se a permissão que permite o acesso do aplicativo a Internet. A figura 38 mostra o processo de exibição das permissões.

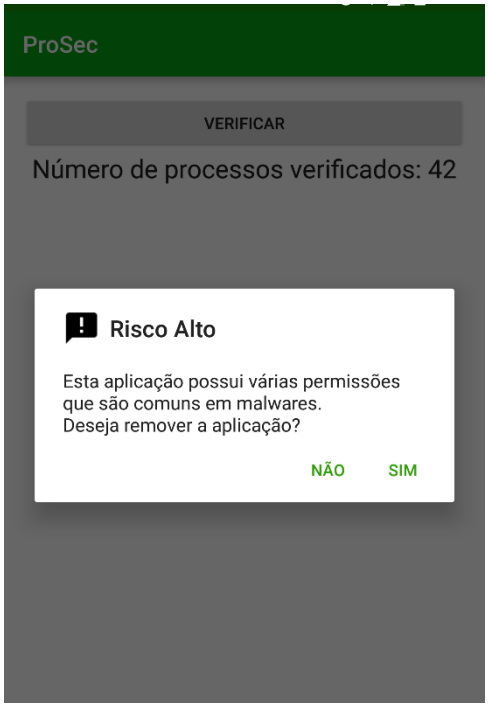
Figura 38 – Obtenção das permissões



Fonte: Próprio autor (2016)

A próxima etapa constituiu na exibição do risco da aplicativo capturado. Esse método faz uma análise das permissões requeridas e informa para o usuário se a aplicação representa um baixo, médio ou alto risco. Conforme feita a análise de risco na aplicação capturada, viu-se que a métrica utilizada reproduz um resultado eficaz apresentando o *malware* capturado como sendo de alto risco para o sistema. A figura 39 mostra a ferramenta com o resultado da análise de risco da aplicação.

Figura 39 – Análise de riscos

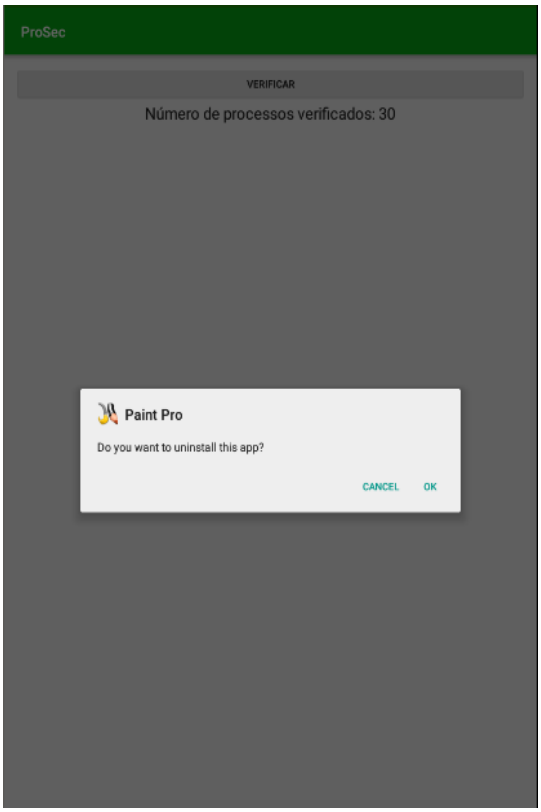


.

Fonte: Próprio autor (2016)

Ao identificar uma aplicação com risco alto o ProSec sugere a desinstalação da mesma. O método de remoção de um aplicativo com alto risco está ilustrado na figura 40. O método de exibição do aplicativo na *Google Play Store* não foi possível executar pois o aplicativo capturado não estava mais presente na mesma.

Figura 40 – Remoção do aplicativo capturado



Fonte: Próprio autor (2016)

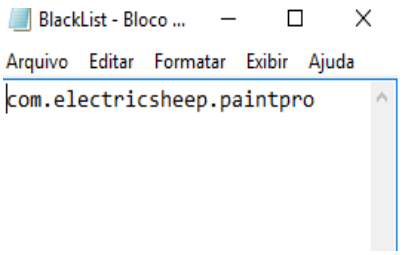
Após a desinstalação do aplicativo capturado efetuou-se novamente uma verificação no sistema e, conforme a figura 41, constatou-se que o sistema estava seguro.

Figura 41 – Aviso de sistema seguro

Fonte: Próprio autor (2016)

Verificou-se também se ocorreu a correta inserção do aplicativo capturado (alto risco) na lista de processos maliciosos (*BlackList*), o qual armazena os processos suspeitos que foram capturados e removidos. Analisando a figura 42 observa-se que o aplicativo capturado foi corretamente inserido na *BlackList*.

Figura 42 – Inserção do processo capturado na *BlackList*



Fonte: Próprio autor (2016)

5.2 Teste de aceitação e *interface*

Os testes de aceitação e *interface* tem como finalidade a verificação das facilidades de utilização, com foco no nível de dificuldade apresentado pelo usuário ao manusear o *software* desenvolvido (MALDONADO, 2014). Os testes foram compostos por 4 tarefas distintas simuladas com a ferramenta ProSec e, contando com a participação de 4 usuários diferentes. Para a realização destes experimentos removeu-se a indicação de processo do aplicativo WhatsApp da *WhiteList,* com o intuito de registrar este como um *malware*, não sendo necessário inserir um processo realmente malicioso no sistema durante os testes. As tarefas propostas foram as seguintes:

- Iniciar uma varredura no sistema. (Tarefa 1);
- Exibir as permissões do aplicativo e obter informações sobre elas. (Tarefa 2);
- Fazer uma análise de risco do aplicativo. (Tarefa 3);
- Desinstalar a aplicação capturada. (Tarefa 4).

Nielsen (2000) propôs testes de interface baseado na contagem de cliques no qual percebeu-se uma maior aceitação por parte dos usuários quando produtos de um *e-commerce* ou função de algum *software* encontravam-se a no máximo 4 cliques da tela inicial. Com isso, foram contabilizados os números de cliques para que cada tarefa fosse completada e ao final do teste atribuiu-se uma nota para a realização das tarefas e a confiança nos resultados. As notas foram divididas em 5 intervalos sendo 1 como difícil ou nada satisfeito e 5 como fácil e muito satisfeito (LLAURADÓ, 2015). A tabela 4 demonstra os resultados obtidos na realização das tarefas com os usuários finais.

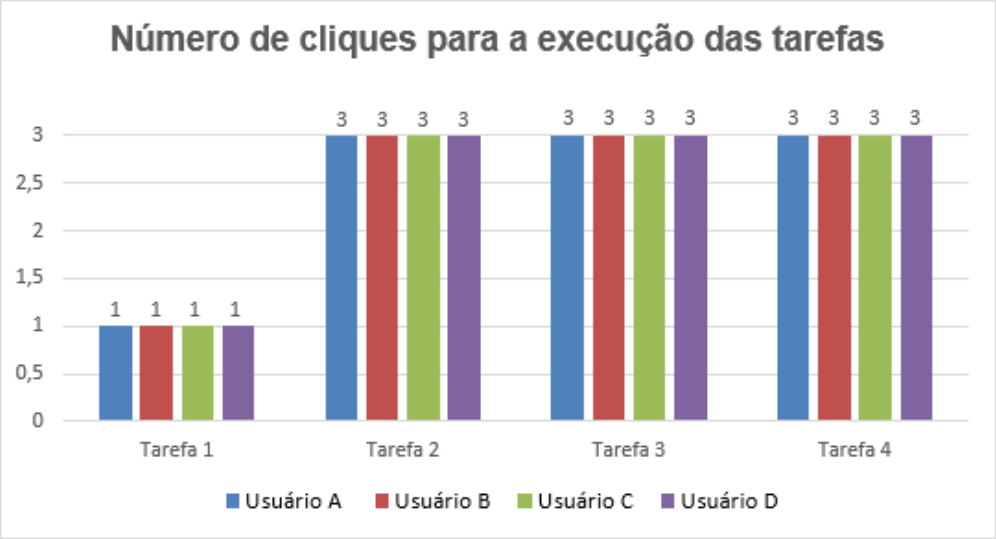
Tabela 4 - Tabelas com as contagens de cliques na execução das tarefas

Usuário A				
Tarefas Realizadas				
	Tarefa 1	Tarefa 2	Tarefa 3	Tarefa 4
Nº cliques	1	3	3	3
Usuário B				
Tarefas Realizadas				
	Tarefa 1	Tarefa 2	Tarefa 3	Tarefa 4
Nº cliques	1	3	3	3
Usuário C				
Tarefas Realizadas				
	Tarefa 1	Tarefa 2	Tarefa 3	Tarefa 4
Nº cliques	1	3	3	3
Usuário D				
Tarefas Realizadas				
	Tarefa 1	Tarefa 2	Tarefa 3	Tarefa 4
Nº cliques	1	3	3	3

Fonte: Próprio autor (2016)

A figura 43 ilustra os números de cliques realizados pelos 4 usuários nas 4 tarefas propostas com base na tabela 4.

Figura 43 – Número de cliques para a execução das tarefas



Fonte: Próprio autor (2016)

Como resultado da avaliação por parte dos usuários, a média dos cliques para a tarefa 1 foi de clique, e para as tarefas 2, 3 e 4 foram de 3 cliques.

A tabela 5 demonstra as notas atribuídas pelos usuários para a confiança dos resultados mostrados pela aplicação e pela facilidade de manuseio da aplicação.

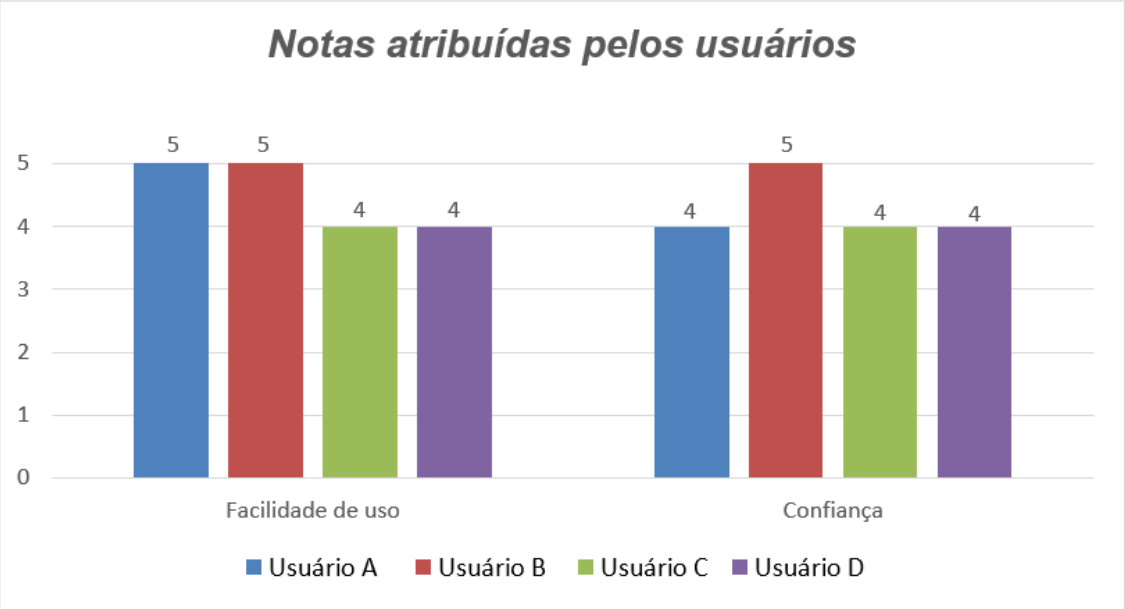
Tabela 5 - Notas atribuídas a cada requisito

Usuário A		
Tarefas Realizadas		
	Facilidade de uso	Confiança
Nota	5	4
Usuário B		
Tarefas Realizadas		
	Facilidade de uso	Confiança
Nota	5	5
Usuário C		
Tarefas Realizadas		
	Facilidade de uso	Confiança
Nota	4	4
Usuário D		
Tarefas Realizadas		
	Facilidade de uso	Confiança
Nota	4	4

Fonte: Próprio autor (2016)

A figura 44 ilustra as notas atribuídas pelos usuários baseadas na tabela 5.

Figura 44 – Notas atribuídas pelos usuários

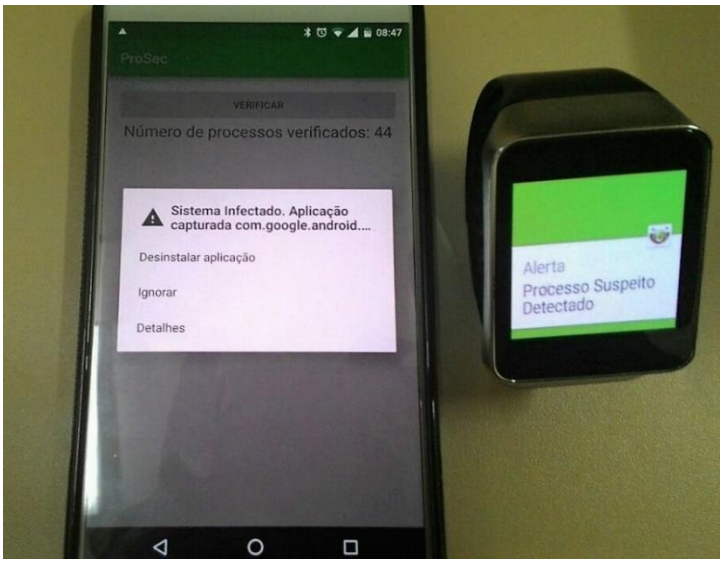


Fonte: Próprio autor (2016)

Como resultado da avaliação por parte dos usuários, a média das notas atribuídas para a facilidade de uso foi de 4,5 e para confiança foi de 4,25.

Com o intuito de melhorar a interação do usuário com o ProSec, esta ferramenta foi desenvolvida para emitir avisos ao usuário de forma visual, sonora e através de dispositivos vestíveis que estejam conectados ao *smartphone* do usuário. A figura 45 demonstra a notificação emitida pelo ProSec, rodando em um *smartphone Motorola Moto X Style*, enviada a um dispositivo vestível *Samsung Galaxy Gear Live* (Android 6.0.1). Segundo Mann (1998), um computador vestível é um computador que está alocado no espaço pessoal do usuário, controlado pelo usuário, e possui constância de operação e interação, ou seja, está sempre ligado e sempre acessível.

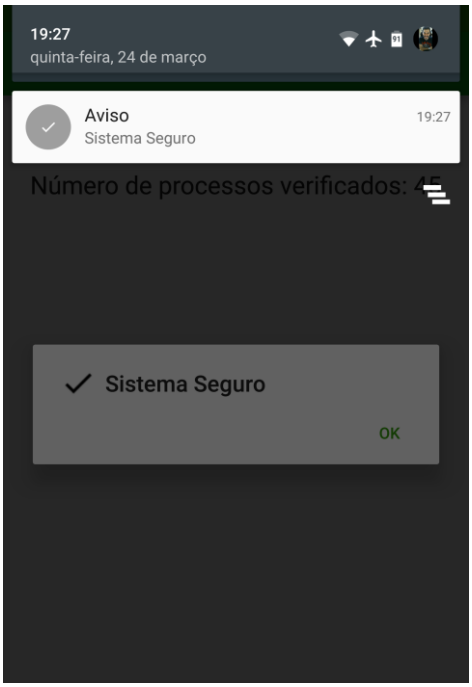
Figura 45 – Notificação sendo emitida para o dispositivo *Android Wear*



Fonte: Próprio autor (2016)

Na figura 46 é apresentada uma notificação visual desta ocorrência no dispositivo.

Figura 46 – Notificação do sistema



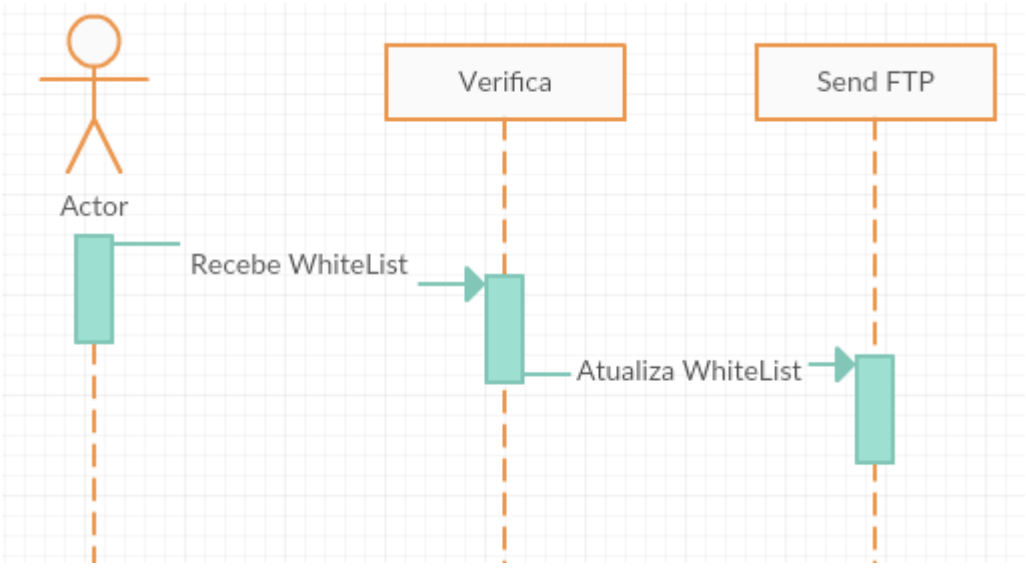
Fonte: Próprio autor (2016)

Conforme os testes realizados, concluiu-se que as notificações emitidas pela app ProSec funcionaram corretamente, apresentando as notificações pertinentes quanto a sistema seguro ou vulnerável.

5.3 Teste do servidor *Web*

Os testes com o servidor *web* serviram para averiguar a eficiência da atualização da *WhiteList* e *BlackList*. Para o experimento foram utilizados dois dispositivos: um *Asus Zenfone 5* executando a versão 5.0.2 do sistema Android e um *Motorola Moto X Style* executando a versão 6.0 do sistema Android nos quais foram removidos o processo gerado pelo aplicativo de mensagens *WhatsApp* e tinha como objetivo verificar se após a inserção do aplicativo *WhatsApp* como processo confiável, a ferramenta seria capaz de ignorar o processo gerado. Escolheu-se essa aplicação pois a mesma é comumente utilizada nos dispositivos móveis e eliminou-se a necessidade de instalar um *malware* nos aparelhos utilizados. A figura 47 apresenta o diagrama do teste realizado.

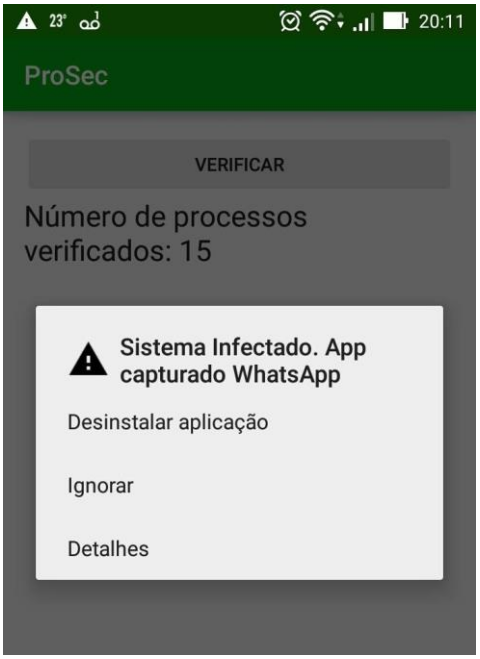
Figura 47 – Diagrama de testes do servidor *Web*



Fonte: Próprio autor (2016)

A primeira execução ocorreu no dispositivo *Zenfone* e consistiu em verificar se a ferramenta capturava a aplicação *WhatsApp*. O momento da captura está ilustrado na figura 48.

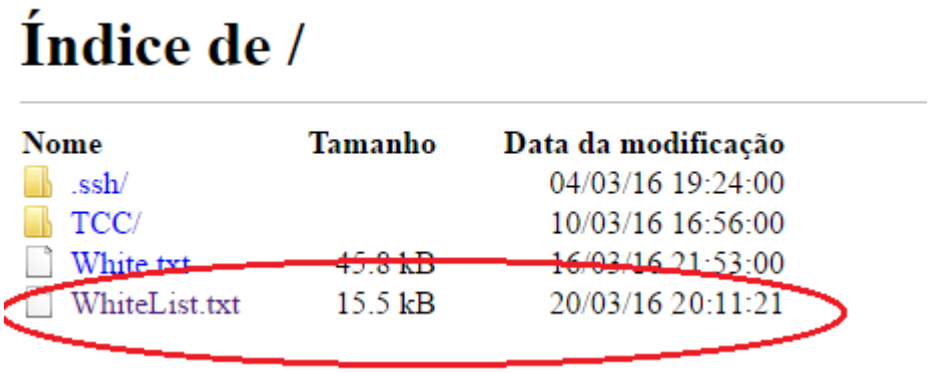
Figura 48 – Captura do aplicativo *WhatsApp* no *Asus Zenfone*



Fonte: Próprio autor (2016)

Ao finalizar a etapa anterior, executou-se o processo de ignorar a aplicação capturada com o objetivo de adicionar a mesma na lista de processos confiáveis. Quando o usuário encerrou a aplicação, o dispositivo efetuou a conexão com o servidor *Web* e enviou sua *WhiteList* atualizada para o servidor. A figura 49 demonstra no servidor FTP o arquivo corretamente atualizado em concordância com o horário do *smartphone Zenfone 5* (20:11).

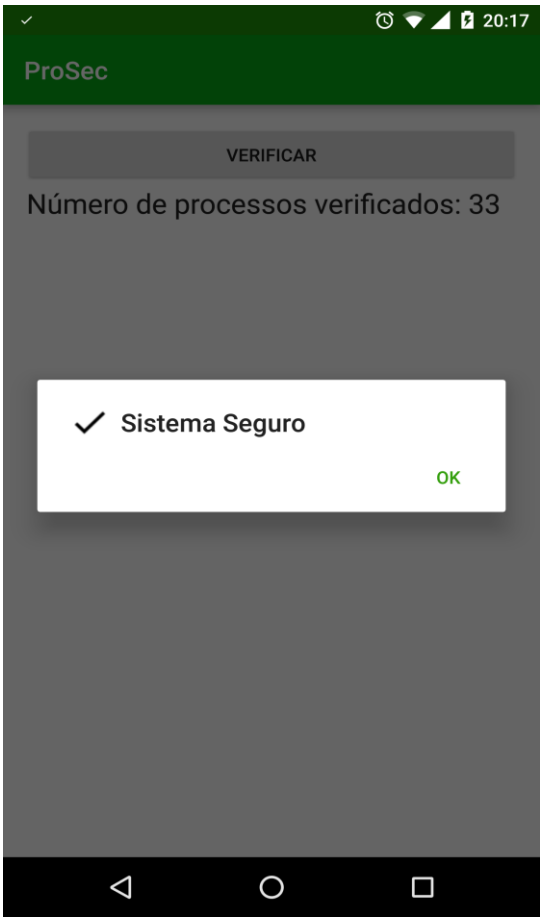
Figura 49 – *WhiteList* atualizada no servidor FTP



Fonte: Próprio autor (2016)

A segunda etapa do teste constituiu-se em verificar em outro dispositivo que continha a aplicação *WhatsApp,* se a ferramenta ProSec iria capturar corretamente o aplicativo ou não. Verificou-se que no momento da abertura da ferramenta ProSec, o aparelho conectou-se ao servidor *Web* e fez o *download* de uma *WhiteList* mais atualizada disponível, com isso ao executar a verificação, constatou-se que a ferramenta ProSec não capturou o aplicativo *WhatsApp* indicado a correta atualização *d*a. O aviso de sistema seguro está representado na figura 50.

Figura 50 – Aviso de sistema seguro no dispositivo *Motorola Moto X Style*



Fonte: Próprio autor (2016)

Como última etapa dos testes com o servidor *Web,* verificou-se que a conexão com o servidor *Web* para envio e recebimento da *WhiteList* armazenada no aparelho ocorreu de forma satisfatória e atingindo o objetivo de manter sempre uma *WhiteList* atualizada nos aparelhos que estejam executando a ferramenta ProSec.

5.4 Teste de desempenho

Para aferir o desempenho da ferramenta realizou-se um teste de verificação de processos em três dispositivos com configurações de hardware e versões do sistema Android distintos e um teste para verificar o tempo total para concluir o *download* da *WhiteList.*

Os dispositivos utilizados na realização do primeiro teste de desempenho estão descritos na tabela 6.

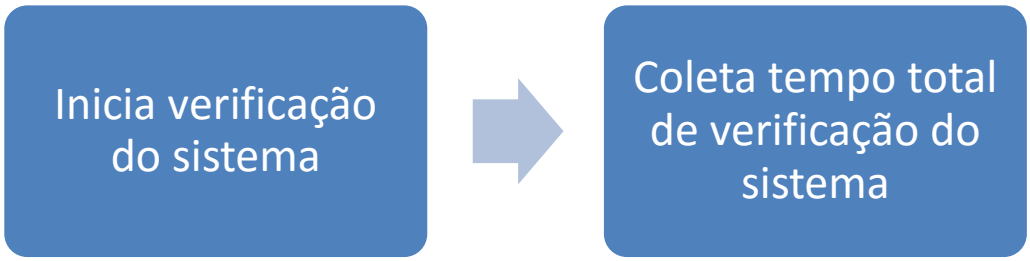
Tabela 6 - Dispositivos utilizados

Dispositivo	Processador	Memória RAM
Samsung Galaxy S II	Dual core 1.2 GHz	1 GB
Asus Zenfone 5	Dual core 1.6 GHz	2 GB
Motorola Moto X Style	Hexa core 2.2 GHz	3 GB

Fonte: Próprio autor (2016)

O teste ocorreu utilizando medições de tempo disponibilizadas pela própria IDE, e foi feita a verificação em cada dispositivo 3 vezes seguidas. A cada nova rodada de verificações excluiu-se o *App* ProSec. A figura 51 ilustra a metodologia de testes utilizada.

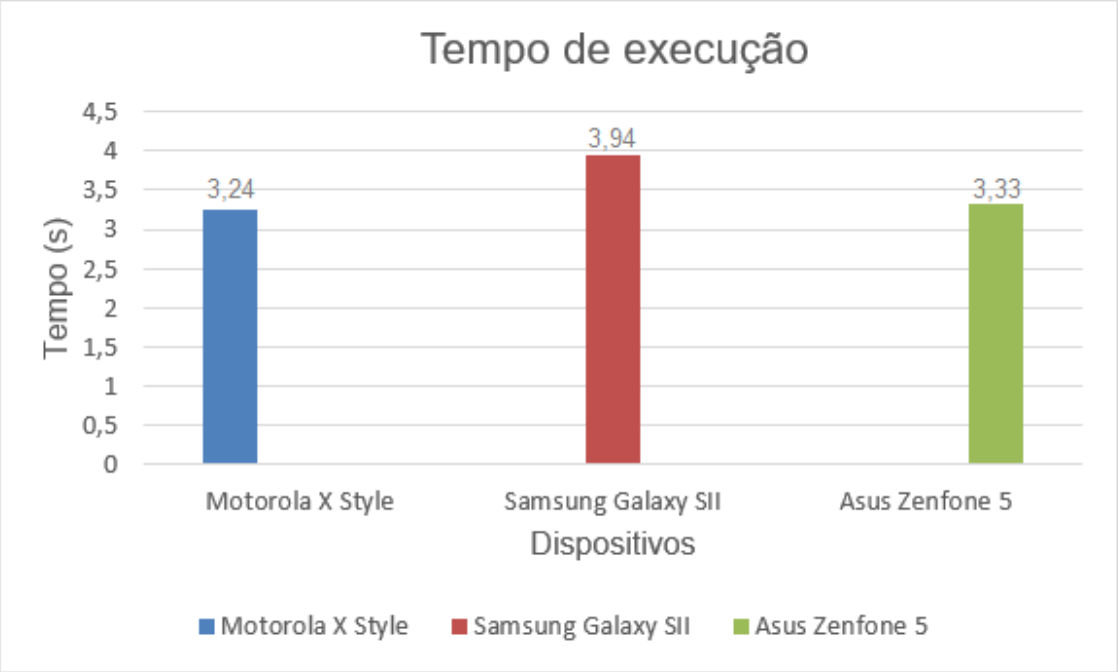
Figura 51 – Metodologia utilizada



Fonte: Próprio autor (2016)

Verificou-se que nas mesmas condições de uso, o dispositivo da *Samsung* realizou o procedimento em 3,94 segundos. O dispositivo *Motorola* apresentou um *delay* de 3,24 segundos para realizar a mesma tarefa e o *smartphone* Asus apresentou um *delay de* 3,33 segundos. Levando em consideração que o dispositivo *Samsung Galaxy S II*, lançado no primeiro trimestre de 2011, não houve grande disparidade de tempo com o dispositivo da *Motorola* que possui configurações de *hardware* bem superiores e que foi lançado no último trimestre de 2015. A figura 52 ilustra os diferentes os tempos de execução do aplicativo.

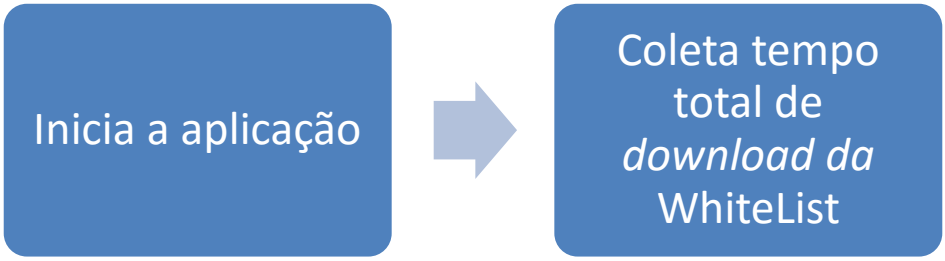
Figura 52 – Tempo para execução do aplicativo



Fonte: Próprio autor (2016)

O próximo teste de desempenho consistiu em analisar a ferramenta fazendo uso de uma conexão 3G e 2G de dados móveis e uma conexão WiFi para aferir o tempo de *download* da *WhiteList*. O tamanho do arquivo contento os processos que será recebido do servidor é de 15,5kB. O procedimento de testes foi o mesmo dos testes realizados anteriormente. Fez-se uso de funções da IDE para medições de tempo, e posteriormente fez-se o teste 3 vezes e utilizou-se a média aritmética dos tempos obtidos. A figura 53 ilustra a metodologia utilizada para aferir o tempo de *download*.

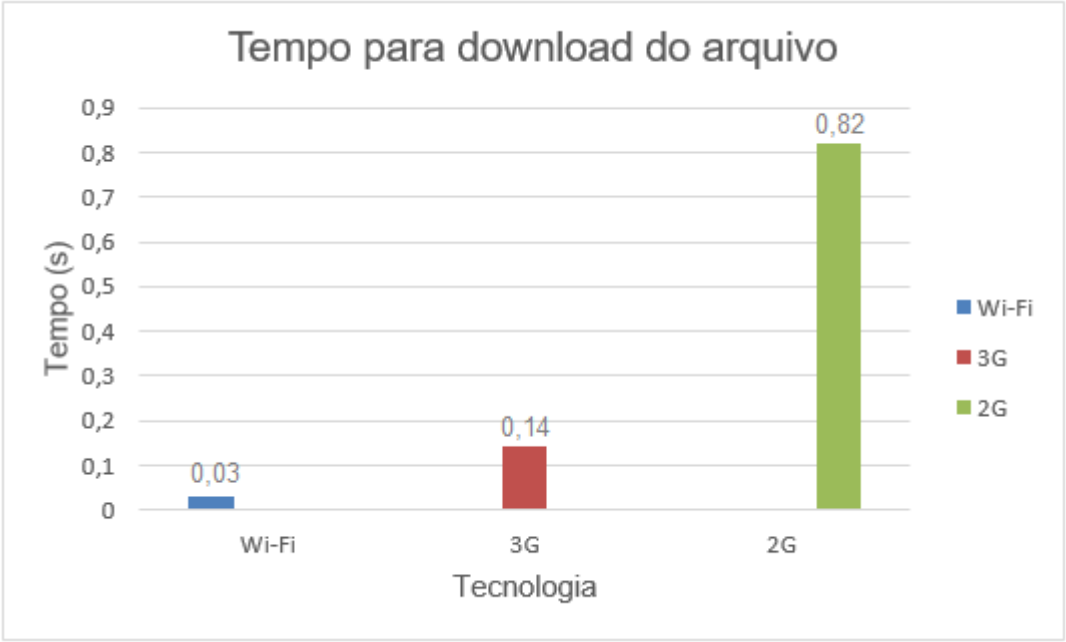
Figura 53 – Rotina para cálculo de tempo



Fonte: Próprio autor (2016)

Com o dispositivo conectado a uma rede Wi-Fi, o tempo decorrido para realizar o *download* da *WhiteList* foi de 0,03 segundos. Com o dispositivo conectado em uma rede 3G o tempo decorrido foi de 0,14 segundos e em uma rede 2G o tempo decorrido foi de 0,82 segundos. O dispositivo utilizado para a realização dos testes foi um *Motorola Moto X Style.* O gráfico ilustrado na figura 54 demonstra os 3 tempos obtidos.

Figura 54 – Tempo para conclusão do *download* da *WhiteList*

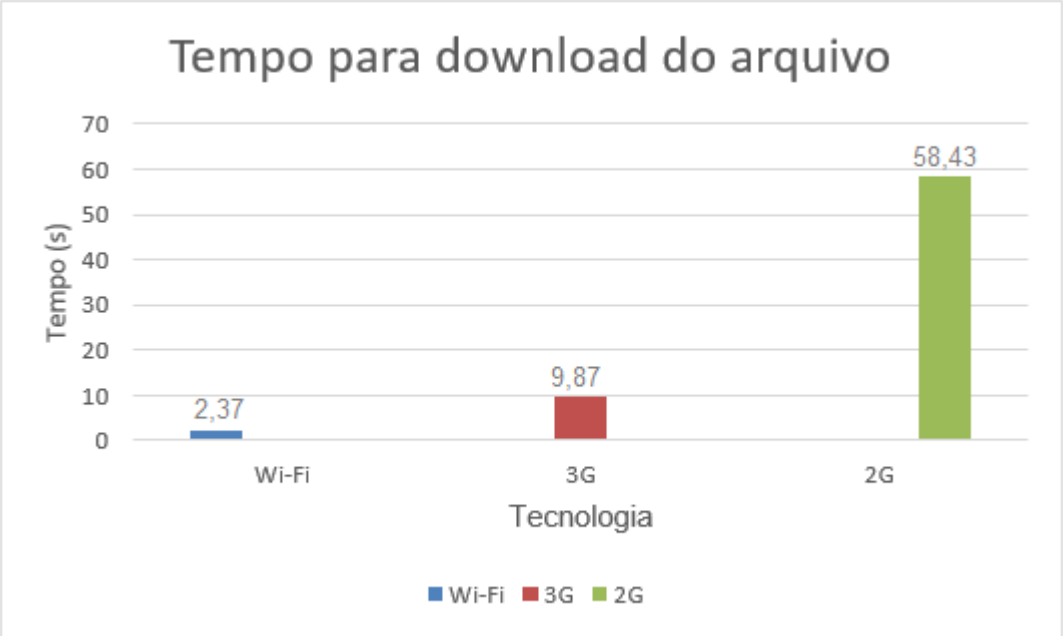


Fonte: Próprio autor (2016)

Percebeu-se que mesmo em conexões precárias como por exemplo uma conexão 2G, a ferramenta conseguiu realizar o *download* da *WhiteList* em um curto espaço de tempo.

Como a *WhiteList* testada é uma *WhiteList* inicial e por isso tem seu tamanho reduzido, criou-se um teste de carga para aferir o tempo de *download* de uma *WhiteList* contendo aproximadamente 55 mil processos armazenados, o qual resultou em um arquivo com aproximadamente 1,01 megabytes. O procedimento realizado foi o mesmo realizado anteriormente, mudando apenas o tamanho do arquivo. O gráfico ilustrado na figura 55 demonstra os 3 tempos obtidos.

Figura 55 – Tempo para conclusão do *download* da *WhiteList* modificada



Fonte: Próprio autor (2016)

Com o dispositivo conectado a uma rede Wi-Fi, o tempo decorrido para realizar o *download* da *WhiteList* foi de 2,37 segundos. Com o dispositivo conectado em uma rede 3G o tempo decorrido foi de 9,87 segundos e em uma rede 2G o tempo decorrido foi de 58,43 segundos.

5.5 Teste da *WhiteList*

Essa seção de testes objetivou verificar a eficácia da *WhiteList* a fim de constatar a efetividade na liberação de processos seguros para o sistema. O método consistiu em instalar a ferramenta ProSec em diversos dispositivos e realizar uma análise sobre os processos capturados e posteriormente adiciona-los ao arquivo contendo os processos seguros (*WhiteList*). A cada aplicação capturada, realizou-se uma pesquisa na *Google Play Store* com intuito de averiguar a sua procedência. A tabela 7 apresenta os dispositivos utilizados e as aplicações capturadas pela ferramenta.

Tabela 7- Dispositivos utilizados

Dispositivo utilizado	**Processos capturados**
Motorola Moto X Style	com.google.wereable.app;
Asus Zenfone 5	com.google.android.gms.feedback; com.asus.internal.fdctoolstate;
Samsung Galaxy S II	com.enlightment.screenshot; org.cyanogenmod.bugreport;
Nvidia Tegra Note 7	Não identificou processos

Fonte: Próprio autor (2016)

Com base nos resultados obtidos constatou-se a efetividade da *WhiteList,* pois a mesma reconheceu todas as aplicações seguras nos dispositivos analisados. Descrição dos resultados observados:

- O *smartphone Motorola* capturou um processo oriundo de um dispositivo vestível, no qual estava conectado;
- O dispositivo *Zenfone* capturou um processo nativo da fabricante Asus e o processo nativo do sistema responsável pelo envio de relatórios de erros para a *Google*;
- O dispositivo *Samsung* capturou um aplicativo para realizar *screenshots* da tela. O aplicativo em questão encontra-se na *Play Store* sob o nome de

Captura de Tela Fácil[11]. O outro processo capturado também foi um processo responsável por coleta de relatório de erros no dispositivo;

- O *tablet* Gradiente não capturou nenhum processo, indicando que todos os aplicativos instalados no dispositivo estavam inclusos na *WhiteList.*

5.6 Discussões

Com base nos testes realizados, verificou-se que a ferramenta conseguiu cumprir com os objetivos as quais foram propostos. Os testes de funcionalidade retornam um resultado satisfatório quanto a análise de processos e as opções apresentas ao usuário. No quesito de usabilidade, verificou-se que a ferramenta teve boa aceitação por parte dos usuários quanto a facilidade de uso e confiança nos resultados obtidos.

Os testes de desempenho mostraram que a ferramenta não consome grande poder computacional sendo ela executada em aparelhos com configurações bem distintas trazendo para o usuário o resultado da verificação em pouco tempo.

Por último, os testes da *WhiteList* indicaram que a lista está bem concisa conseguindo armazenar a grande maioria das aplicações mais comuns para dispositivos Android.

[11] - Disponível em: https://play.google.com/store/apps/details?id=com.enlightment.screenshot

6 CONSIDERAÇÕES FINAIS

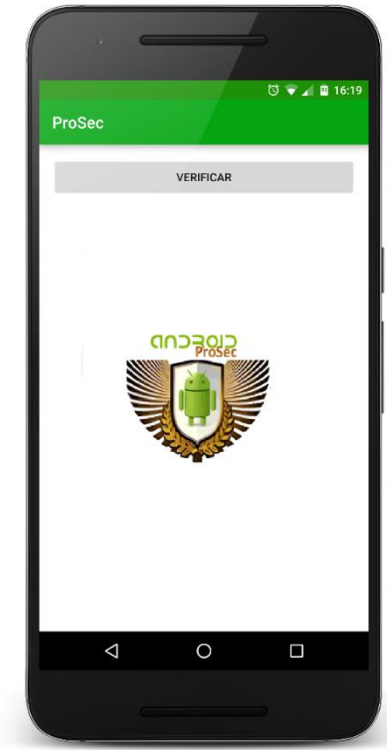
Ao identificar a utilização em larga escala dos dispositivos moveis e questões fundamentais de segurança relacionados a este contexto, este projeto avaliou e propôs uma solução para o reconhecimento de possíveis processos maliciosos no sistema, com base em uma estrutura na qual temos armazenado quais processos não representam riscos para o usuário. A retirada de processos ocorreu a partir de uma análise na loja de aplicativos *Google Play Store* e em dispositivos previamente formatados para que fosse possível a coleta de processos que fossem nativos do sistema.

Após os testes realizados constatou-se que a ferramenta ProSec conseguiu de forma eficiente reconhecer processos que não eram conhecidos do sistema e com isso alertar o usuário sobre processos desconhecidos. Quando foi inserido manualmente um *malware* no sistema, além de detectar o processo como desconhecido, a ferramenta foi capaz também de verificar o risco da aplicação a qual foi corretamente classificada como uma aplicação de alto risco.

Os testes de aceitação e interface mostraram que a ferramenta possui bons aspectos de usabilidade e eficiência e pode ser manuseada por usuários que não possuam conhecimento técnicos e que o resultado apresentado pela ferramenta pode ajudar no incremento da segurança de dispositivos que executem o sistema Android.

O aplicativo que implementa o modelo de análise de processos criado foi concluído com êxito, mantendo bons aspectos de usabilidade e eficiência, comprovados por testes. A figura 56 ilustra a tela inicial da aplicação finalizada.

Figura 56 – Ferramenta finalizada



Fonte: Próprio autor (2016)

Partindo do problema de pesquisa inicial do trabalho (É possível propor e implementar uma solução para o monitoramento de processos no S.O Android que identifique de forma efetiva aplicações maliciosas neste sistema?), conclui-se que esta é uma solução viável para o incremento dos níveis de segurança de dispositivos móveis que executem o sistema operacional Android, portanto os objetivos traçados nesse trabalho foram alcançados.

6.1 Trabalhos futuros

Para os trabalhos futuros, será adotado um modelo de banco de dados para armazenar os processos e, juntamente com o nome do processo será utilizado a assinatura do desenvolvedor. O acréscimo deste parâmetro na verificação de processos irá incrementar a eficiência do aplicativo. Será também implementada a verificação do sistema em *background* a qual fará constantes verificações no sistema quando a ferramenta detectar algum novo processo que esteja em execução. Outra função planejada é a melhoria no método de comparação de processos fazendo o uso de busca binária e paralelismo.

REFERÊNCIAS

AFONSO, Vitor M., et al. "**Um sistema para análise e detecção de aplicações maliciosas de Android**." SBSEG 2013.

AMOROSO Danilo, **"O que são processos de um sistema operacional e por que é importante saber".** Disponível em <http://www.tecmundo.com.br/memoria/3197-o-que-sao-processos-de-um-sistema-operacional-e-por-que-e-importante-saber.htm>

ANDROID "**Manifest Permission".** Disponível em < http://developer.android.com/intl/pt-br/reference/android/Manifest.permission.html>. Acesso em 23/10/2015.

APPLE WWDC "**WWDC 2014 Session Videos".** Disponível em < https://developer.apple.com/videos/wwdc/2014/>. Acesso em 01/09/2015

ARM, "**Cortex-A72 Processor**" Disponível em < http://www.arm.com/products/processors/cortex-a/cortex-a72-processor.php>. Acesso em 27/08/2015.

ARRUDA "**A história dos processadores**". Disponível em < http://origin.tecmundo.com.br/historia/2157-a-historia-dos-processadores.htm?utm_source=tecmundo&utm_medium=error-pdf>. Acesso em 20/08/2015

ARSTECHINA "**Major Android remote-access vulnerability is now being exploited".** Disponível em <http://arstechnica.com/security/2015/08/major-android-remote-access-vulnerability-is-now-being-exploited/>. Acesso em 03/09/2015

AVAST "**Mobile Security: Stagefright - nova vulnerabilidade Android coloca o seu dispositivo em risco".** Disponível em < https://www.avast.com/pt-br/faq.php?article=AVKB230>. Acesso em 05/09/2015

BATISTA, A. L. P., Dellaquila, B. L., & da Rocha, G. "**Análise da Segurança de Aplicativos na Plataforma Android Através da Adoção de Patterns"**. 2013

BLACK HAT CONFERENCE “**Android Master Key Vulnerability Makes Us Safer**”. Disponível em <http://www.esecurityplanet.com/mobile-security/black-hat-android-master-key-vulnerability-makes-us-safer.html>. Acesso em 01/09/2015

BLUE PHOENIX. “**Boas práticas de segurança**”. Disponível em: <www.bluephoenix.pt>. Acesso em 06/09/2015

BORDIN, M. V. “**Introdução a Arquitetura Android**”.2012

BRAGA, Alexandre Melo; NASCIMENTO, Erick Nogueira do; “PALMA, Lucas. Introdução à Segurança de Dispositivos Móveis Modernos – “**Um Estudo de Caso em Android”**. Minicursos do XII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais, 2012.

CAMPOS, AN. "**Sistema de segurança da informação**." 2006.

CANDIDO “**Sistemas Operacionais dos dispositivos móveis”.** Disponível em < https://plantaovirtual.wordpress.com/2013/09/23/t-i-sistemas-diversos-sistemas-operacionais-dos-dispositivos-moveis/>. Acesso em 28/08/2015.

CERT.br” **Incidentes Reportados ao CERT.br -- janeiro a dezembro de 2014”.** Disponível em < http://www.cert.br/stats/incidentes/2014-jan-dec/analise.html>. Acesso em 20/08/2015.

CHEN, Kai, et al. **"Finding Unknown Malice in 10 Seconds: Mass Vetting for New Threats at the Google-Play Scale**." 24th USENIX Security Symposium (USENIX Security 15). USENIX Association.

CIBRÃO, D., & Gonçalves, R. “**Segurança no Android**”.

COSTA et.*al.* “**COMPARATIVE EVALUATION OF OPERATING SYSTEMS FOR DEVICES MOBILE: FOCUS ON FUNCTIONALITY**”. UNIFESP, São Paulo 2011.

CRIMES PELA INTERNET “**Ataques malware triplicam nos dispositivos móveis**”. Disponível em < http://www.crimespelainternet.com.br/ataques-malware-triplicam-nos-dispositivos-moveis/>. Acesso em 01/09/2015.

DEVMEDIA “**Artigo Engenharia de Software - Introdução a Teste de Software”.** Disponível em < http://www.devmedia.com.br/artigo-engenharia-de-software-introducao-a-teste-de-software/8035>. Acesso em 13/03/2016.

DO ESPÍRITO SANTO, Adrielle Fernanda Silva. "**SEGURANÇA DA INFORMAÇÃO**."

FEDLER, Rafael; KULICKE, Marcel; SCHUTTE, Julian. “**An antivirus API for Android malware recognition. In: Malicious and Unwanted Software**:" The Americas"(MALWARE), 2013 8th International Conference on. IEEE, 2013. p. 77-84.

FIGUEIREDO, Carlos MS; NAKAMURA, Eduardo. “**Computação móvel: Novas oportunidades e novos desafios**”. T&C Amazônia, p. 16-28, 2003.

FURLAN, José Davi. “**Modelagem de objetos através da UML-the unified modeling language”**. Makron books, 1998.

GATTO, Sandro “**O Futuro da Mobilidade**”. Disponível em: <http://www.administradores.com.br/artigos/tecnologia/o-futuro-da-mobilidade/67448/>. Acesso em: 17/04/2014.

GIL, Antônio Carlos. **Como elaborar projetos de pesquisa.** 5. ed. São Paulo: Atlas, 2008.

GLOBO, “**Em 2014, 1 milhão de apps para Android eram malwares disfarçados”**. Disponível em < http://g1.globo.com/tecnologia/noticia/2015/04/em-2014-1-milhao-de-apps-para-android-eram-malwares-disfarcados.html>. Acesso em 02/09/2015.

GOOGLE PLAY “**Termos de Serviço do Google Play**”. Disponível em <https://play.google.com/intl/pt-BR_br/about/play-terms.html>. Acesso em 06/09/2015

GUIMARÃES, Gleyser “**A história do sistema operacional Android**”. Disponível em:

<http://www.dsc.ufcg.edu.br/~pet/jornal/agosto2013/materias/historia_da_computacao.html. Acessado em 06/09/2015>. Acesso em 06/09/2015

HAMMAN, “i**OS Android e Windows Phone: números dos gigantes comparados**”. Disponível em < http://www.tecmundo.com.br/sistema-operacional/60596-ios-android-windows-phone-numeros-gigantes-comparados-infografico.htm >. Disponível em 07/04/2016.

IDC “**Smartphone OS Market Share, 2015 Q2”.** Disponível em < http://www.idc.com/prodserv/smartphone-os-market-share.jsp>. Acesso em 24/08/2015

JAGGAR, Dave. "**ARM architecture and systems**." IEEE micro 4 1997.

JACOBSEN Wilson, “**ANÁLISE E PROPOSTA DE UM MODELO PARA AVALIAÇÃO DE RISCOS EM APLICATIVOS PARA MÓVEIS”.** Bagé – RS. 2014

JOHNSON, Ryan, and Angelos Stavrou. **"Forced-path execution for android applications on x86 platforms**." Software Security and Reliability-Companion (SERE-C), 2013 IEEE 7th International Conference on. IEEE, 2013.

JON STOKES **"ARM fills out CPU lineup with Cortex A5**". Ars Technica. Retrieved 2012-10-18. 2009

JUNG, Ho-Won; KIM, Seung-Gweon; CHUNG, Chang-Shin. **“Measuring software product quality: A survey of ISO/IEC 9126**”. IEEE software, n. 5, p. 88-92, 2004.

KASPERSKY LABS, “**Kaspersky Internet Security for Android”.** Disponível em < http://brazil.kaspersky.com/downloads/versoes-de-teste/android-security>. Acesso em 04/08/2015

LLAURADÓ, **“Escala de Likert: o que é e como utilizá-la”.** Disponível em < http://www.netquest.com/blog/br/escala-likert/>. Acesso em 08/07/2015

MAIA “**O que é Segurança da informação**”. Disponível em < http://segurancadainformacao.modulo.com.br/seguranca-da-informacao>. Acesso em 22/08/2015.

MALDONADO, José Carlos et al. “**Introdução ao teste de software”**. São Carlos, 2004.

MANDEL, ARNALDO, IMRE SIMON, and Jorge L. DeLYRA. "**Informação: computação e comunicação**." Revista USP 35 (1997): 10-45.

MANN Steve, “**Definition of Wearable Computer"** Toronto, 1998.

MORIMOTO **“Medfield: começa a guerra ARM x x86 nos smartphones”.** Disponível em < http://www.hardware.com.br/artigos/medfield/>. Acesso em 21/11/2015

NIELSEN, Jakob; LORANGER, Hoa. “**Prioritizing web usability”**. Pearson Education, 2006.

NORBEM “**COMPARATIVE EVALUATION OF OPERATING SYSTEMS FOR DEVICES MOBILE: FOCUS ON FUNCTIONALITY”**. Disponível em < http://revistas.unifacs.br/index.php/rsc/article/download/2581/1950.>. Acesso em 07/04/2016.

LEE, Meng “**Introdução ao Desenvolvimento de Aplicativos para o Android.”**

LI, Xun, et al. "**Smartphone evolution and reuse: Establishing a more sustainable model**." Parallel Processing Workshops (ICPPW), 2010 39th International Conference on. IEEE, 2010.

PHONE ARENA “**10 iconic features of each major Android update, from Cupcake to Lollipop**”. Disponível em <http://www.phonearena.com/news/10-iconic-features-of-each-major-Android-update-from-Cupcake-to-Lollipop_id62162#cMS9QbdCpK8m4BTC.99>. Acesso em 02/09/2015

PILZ, Hendrik. “**Building a test environment for Android anti-malware tests. In: VB Conference**”. 2012. p. 1-7.

PLAY STORE” **Google Play Store”.** Disponível em <https://play.google.com/store/apps>

PRESSMAN, R. S., “**Software Engineering: A Practitioner’s Approach**”, McGraw-Hill, 6th ed, Nova York, NY, 2005.

RAMALHO, Abraão Lincon da S.; DOTTO, Roan Siviero; DE OLIVEIRA CARNEIRO, Suliane**. ANDROID–O NOVO ALVO DE VÍRUS**. In: Workshop de Tecnologia da Região Fronteira Oeste–Anais. 2013.

ROCHA, A. R. C., MALDONADO, J. C., WEBER, K. C. et al., “**Qualidade de software – Teoria e prática**”, Prentice Hall, São Paulo, 2001.

SAMSUNG “**Galaxy s6 Edge +”.** Disponível em < http://www.samsung.com/br/consumer/mobile-devices/smartphones/galaxy-s/SM-G925IZKQZTO>. Acesso em 27/08/2015

SCHMIDT, Aubrey-Derrick, et al. "**Smartphone malware evolution revisited: Android next target?**" Malicious and Unwanted Software (MALWARE), 2009 4th International Conference on. IEEE, 2009.

SONG Wun, “**A Evolução dos Computadores: do ENIAC ao Jaguar**”, IME/USP (2010). Disponível em < http://www.ime.usp.br/~song/mac412/historia.pdf>. Acesso em 07/04/2016

SWANSON, Steven, and Michael Bedford Taylor. "**Greendroid: Exploring the next evolution in smartphone application processors**." Communications Magazine, IEEE 49.4 112-119. 2011

TECHTUDO, “**Exclusivo: Android é alvo de 99% dos vírus para smartphones, diz Kaspersky**”. Disponível em < http://www.techtudo.com.br/noticias/noticia/2015/08/android-e-alvo-de-99-dos-virus-para-smartphones-diz-eugene-kaspersky.html>. Acesso em 13/10/2015

TECMUNDO, “**Há 7 anos, o primeiro iPhone era anunciado**”. Disponível em <http://www.tecmundo.com.br/iphone/48924-ha-7-anos-o-primeiro-iphone-era-anunciado.htm>. Acesso em 25/08/2015

TECMUNDO, “**Nexus One: o primeiro smartphone da Google**”. Disponível em <http://www.tecmundo.com.br/ces-2010/3301-nexus-one-o-primeiro-smartphone-da-google.htm>. Acesso em 25/08/2015

TERRA, “**História do celular”.** Disponível em <http://tecnologia.terra.com.br/celular/celulares/>. Acesso em 25/08/2015.

TOLEDO, M. “**Gerenciamento de Processos no Linux**”. 2009

WYATT Tim, “**Update: Security Alert: DroidDreamLight, New Malware from the Developers of DroidDream”.** Disponível em < https://blog.lookout.com/blog/2011/05/30/security-alert-droiddreamlight-new-malware-from-the-developers-of-droiddream/>. Acesso em 13/03/2016.

ZANONI, Felipe Sanches. “**Avaliação de arquitetura paralela para smartphones e tablets utilizando GNU/Linux e MPI para processamento de dados**”. Tese de Doutorado. Universidade de São Paulo.

ZHOU, Yajin; JIANG, Xuxian. “**Dissecting android malware: Characterization and evolution. In: Security and Privacy (SP)”**, 2012 IEEE Symposium on. IEEE, 2012. p. 95-109.