UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE ELETRICIDADE

Vladimir Bezerra de Oliveira

HoneypotLabsac: Um Framework de Honeypot Virtual para o Android

São Luís
2012

Vladimir Bezerra de Oliveira

HoneypotLabsac: Um Framework de Honeypot Virtual para o Android

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Eletricidade da Universidade Federal do Maranhão como requisito parcial para a obtenção do grau de MESTRE em Engenharia de Eletricidade.

Orientador: Ph.D. Zair Abdelouahab

Ph.D em Ciência da Computação – UFMA

São Luís

2012

0482h Oliveira, Vladimir Bezerra de Oliveira

HoneypotLabsac: Um *Framework* de *Honeypot* Virtual para o *Android* / Vladimir Bezerra de Oliveira 0482h Oliveira. – São Luís, 2012.

86 f.

Orientador: Ph.D. Zair Abdelouahab.

Impresso por computador (fotocópia).

Dissertação (Mestrado) – Universidade Federal do Maranhão, Programa de Pós-Graduação em Engenharia de Eletricidade. São Luís, 2012.

1. Segurança. 2. *Honeypot*. 3. *Honeypot* para Dispositivos Móveis.

Ficha elaborada pelo Serviço de Catalogação da biblioteca Central da UESPI ureste de Sousa Lima (Bibliotecário) CRB-3//1215

I. Abdelouahab, Zair, orient. II. Título.

CDU 005.8

HONEYPOTLABSAC: UM FRAMEWORK DE HONEYPOT VIRTUAL PARA O ANDROID

Vladimir Bezerra de Oliveira

Dissertação aprovada em 26 de Junho de 2012.

Prof. Zair Abdelouahab, PhD.
(Orientador)

Prof. Omar Andres Carmona Cortes, Dr.
(Membro da Banca Examinadora)

Prof. Aristófanes Corrêa Silva, Dr.
(Membro da Banca Examinadora)

Aos meus pais Heraldo (in memoriam) e Maria Jane, minha querida esposa Anita Myrtes pelo apoio incondicional.

Resumo

Os dispositivos móveis, como os *Smartphones*, tornaram-se indispensáveis nos dias atuais, devido ao aumento do poder de processamento, maior espaço de armazenamento de dados, baterias com maior autonomia de tempo, conexão a rede *wireless* e à rede 3g. O Sistema Operacional *Android* é uma plataforma completa para dispositivos móveis principalmente para aparelhos celulares inteligentes, desenvolvida pela Google em 2008. Ganhando nesse contexto cada vez mais espaço no mercado mundial, devido ser *open-source*, ou seja, código fonte aberto. Ataques a telefones celulares não é uma prática atual, o primeiro vírus virtual denominado de *Cabir*, foi desenvolvido em 2004, e visava exclusivamente o sistema operacional *Symbiam*. Estudos apresentam grande evolução de ataques digitais ao sistema operacional *Android*. Os *Honeypots* (ferramentas que dispõem de diversas funcionalidades e que tem como objetivo principal enganar o invasor) podem ser bastante úteis no âmbito de segurança de rede. Eles fazem com que o atacante pense que está interagindo de fato com um sistema operacional, mas na verdade o atacante está sendo monitorado. Neste sentido, o presente trabalho foi realizado com o objetivo de desenvolver um *Framework* (arcabouço) para gerar *Honeypot* virtual a nível de aplicação para o sistema operacional *Android*. Os procedimentos metodológicos para elaboração deste trabalho foram: pesquisa bibliográfica (artigos, dissertações e literaturas especificas). Conclui-se que é possível monitorar o atacante de dispositivos móveis através do *Honeypot* gerado pelo *Framework* desenvolvido, de forma a ser usado como uma ferramenta em segurança de redes baseados em iludir. Diante da experiência vivenciada neste estudo, expomos algumas recomendações, pontos imprescindíveis para melhorias do tema abordado, como: mais visibilidade ao *Honeypot* e ampliação para outros sistemas operacionais móveis.

Palavras-chave: *Smartphone*, *Honeypot*, *Framework*, Sistemas Operacionais *Android*.

Abstract

Mobile devices such as Smartphones, have become indispensable nowadays, due their increased processing power, more room for data storage, batteries with greater time autonomy, connection to wireless networks and 3G networks. . The Android Operating System is a complete platform for mobile devices principally for Smartphones developed by Google in 2008. It is gaining an increasingly global market space, due to its open-source code. Attacks on mobile phones are not a current practice. The first virtual virus called Cabir, was developed in 2004, and it is concerned only the Symbian operating system. Studies show great evolution of digital attacks to the Android operating system. Honeypots (tools that have many features such as deceive the attacker) can be quite useful in the context of network security. They make the attacker think that he is actually interacting with an operating system, but in fact the attacker is being monitored. Therefore, the present thesis is aimed to develop a Framework (framework) to generate a virtual Honeypot at the level of application for the Android operating system. The methodological procedures for the preparation of this work are the: bibliography research articles, essays and literature specific.. In this work, we show that the attacker can be monitored in mobile devices through a Honeypot generated by the framework developed here in order to be used as a tool in network security based on deception. From our experience in this study, we report some essential recommendations points for improving and expanding this work.

Keywords: Smartphone, Honeypot, Framework, Android Operating Systems .

Agradecimentos

À Deus, Pai celestial, que tudo sabe e pode, que nunca me abandonou nas horas mais difíceis.

A minha querida esposa, Anita Myrtes, mola propulsora para esse desfecho, agradeço pelo carinho, amor, paciência e companheirismo.

Aos meus preciosos pais, Heraldo e Jane, somos a prova que o amor é maior do que a distância eterna e terrena, mesmo frente a saudade e a enfermidade.

Ao extraordinário educador e orientador, Prof. Dr. Zair Abdelouahab, por seus ensinamentos inspiradores, paciência e grandeza d'alma, a reverencio.

Ao Programa de Pós-Graduação PPGEE sem ele não existiria o Aluno.

Aos professores, Dra. Maria da Guia e Denivaldo Lopes, pelas dicas inestimáveis bem como as cobranças, em prol da construção de um bom trabalho. A todos os companheiros de laboratório que se dispuseram a colaborar para realização deste trabalho, em especial as prezadas colegas, Lianna Mara, Jessica Bassani e Amélia Acácia, pelos momentos bons e ruins vivenciados e superados, bem como aos colegas Mário Henrique e Valéria Priscila na implementação do Projeto.

Aos meus filhos, irmãs e irmãos, jóias preciosas, agradeço o carinho e apoio.

Agradeço profundamente àqueles que me acolheram e me trouxeram até aqui, em especial Raimundo Neto.

A tantos outros, por acreditar e colaboradores excepcionais.

"O futuro pertence àqueles que acreditam na beleza dos seus sonhos."

Eleanor Roosevelt.

Lista de Figuras

1.1 Crescimento em volume de *Malwares* móvel para *Android* em 2011 [1] . . 17

2.1 Arquitetura sistema operacional *Android* [2] 22

3.1 Classificação de sistemas existentes [3] 36

3.2 Arquitetura do *Honeydroid* [4] 42

3.3 Modelo do *Framework* HoneyM [5] 43

3.4 Diagrama da Arquitetura *Honeyd* [6] 45

3.5 Modelo em Camadas do NIDIA [7] [8] 46

4.1 Arquitetura do *FrameworkLabsac* 52

4.2 Diagrama de Pacote do *FrameworkLabsac* 53

4.3 Diagrama de Classes do *FrameworkLabsac* 55

4.4 Log gerado pelo *HoneypotLabsac* 60

4.5 Diagrama em Camada *HoneypotLabsac* 61

4.6 Diagrama da Arquitetura *HoneypotLabsac* 62

4.7 Diagrama de Classe do HoneypotLabsac 63

4.8 Caso de Uso Capturar Dados 65

4.9 Diagrama de Sequência de Capturar Dados 66

4.10 Diagrama de Pacote do HoneypotLabsac 67

4.11 Tela principal do *HoneypotLabsac* 68

4.12 Tela de inicialização dos serviços do *HoneypotLabsac* 69

4.13 Tela de configuração das portas de comunicação dos serviços de *HTTP* e *Telnet* e do IP para o servidor de log do *HoneypotLabsac* 70

5.1 Cenário Proposto 71

5.2 Cenário de Teste com Smartphone 72

5.3 Log do Serviço Telnet 73

5.4 Log do Serviço *HTTP* 74

5.5 Log do SMS recebido pela Operadora de Celular 74

5.6 Cenário de Teste *HoneypotLabsac* Instalado em uma máquina virtual usando o sistema operacional *Android* x86 75

5.7 Integração entre o *HoneypotLabsac* e o NIDIA 78

Lista de Tabelas

1.1 Vendas de *Smartphones* nos *U.S.* [9] 17

2.1 Níveis de Interatividade [10] 28

3.1 Resumo dos requisitos (Satisfeito(•) Parcialmente satisfeito(˜) Não satisfeito(◦) [11] 40

3.2 Comparação Qualitativa entre os trabalhos das Seções: 3.2 49

5.1 Comparação Qualitativa 77

Lista de Siglas

3G Terceira Geração de Padrões e Tecnologias de Telefonia Móvel.

AP Access Point.

APIs Application Programming Interfaces.

ARM Advanced RISC Machine.

ASF Apache Software Foundation.

GPS Global Positioning System.

ICMP Internet Control Message Protocol.

iOS Sistema Operacional iPhone.

IP Internet Protocol.

LABSAC Laboratório de Sistemas e Arquiteturas Computacionais.

NIDIA Network Intrusion Detection System Based on Intelligent Agents.

OHA Open Handset Alliance.

PCs Computadores (Desktop).

PDAs Personal Digital Assistent.

QEMU Emulador do SDK Android.

RIM Research In Motion.

SMS Short Message Service.

TCP Transmission Control Protocol.

UDP User Datagram Protocol.

WiFi Wireless Fidelity.

Sumário

Lista de Figuras **ix**

Lista de Tabelas **xi**

Lista de Siglas **xii**

1 Introdução **15**

1.1 Cenário e Definição do Problema 16

1.2 Objetivos Gerais e Específicos 18

1.3 Organização do Trabalho 18

2 Fundamentação Teórica **20**

2.1 Sistema Operacional *Android* 20

2.2 Conceitos Básicos de Segurança de Dados 25

2.3 *Honeypots* - Metodologia para Iludir 25

2.4 Ataques Virtuais a Dispositivos Móveis 30

2.5 Framework 32

2.6 Considerações Finais 33

3 Trabalhos Relacionados **35**

3.1 Classificações de Sistemas Operacionais Móveis 35

3.2 Honeypots para Dispositivos Móveis 41

3.3 Honeypots 44

3.4 Projeto NIDIA 46

3.5 Comparação Qualitativa 48

3.6 Considerações Finais 50

4 *HoneypotLabsac*: Um *Framework* de *Honeypot* Virtual para o *Android* 51

4.1 Aplicabilidade do *Framework* 51

4.2 Arquitetura do *Framework* 52

4.3 Utilização do *Framework* 54

4.4 Criando um Serviço a partir do *FrameworkLabsac* 57

4.5 Aplicabilidade do *HoneypotLabsac* 59

4.6 Arquitetura do *HoneypotLabsac* 61

4.7 Implementação do *HoneypotLabsac* 62

4.8 Caso de Uso do *HoneypotLabsac* 65

4.9 Usando o *HoneypotLabsac* 68

4.10 Considerações Finais 70

5 Testes e Resultados 71

5.1 Integração com o NIDIA 77

6 Conclusões 79

Referências Bibliográficas 82

1 Introdução

O Mercado mundial de dispositivos móveis (*Smartphones, Tablets, Personal Digital Assistent (PDAs), Netbooks, Notebooks*) está cada vez mais atrativo. Nesta categoria destacam-se os *Smartphones*, principalmente devido a preços cada vez mais acessíveis e ao avanço da tecnologia empregada nestes dispositivos.

Os dispositivos móveis tornam-se imprescindíveis no dia a dia das pessoas. Chamam-se a atenção pelas grandes melhorias nas características técnicas: aumento no poder de processamento, com microprocessadores mais rápidos; maior espaço para armazenar dados; baterias com tempo de autonomia maior que suas antecessoras; conexão a rede *wireless*; conexão a rede Terceira Geração de Padrões e Tecnologias de Telefonia Móvel (3G); etc.

A operacionalidade de tais equipamentos tem ampliado o acesso às redes sociais, navegação na Internet, acesso a transações bancárias, realização de pedidos e acesso a e-mails, que são executadas diretamente destes dispositivos móveis. Os *Smartphones* necessitam de um sistema operacional para gerenciamento de *hardware* e aplicativos. Dentre estes, destacam-se: *Windows Mobile* da *Microsoft* [12] [13], *Symbiam OS* da *Symbian* [14] [13], *Mobile Linux* [15], *BlackBerry OS* da Research In Motion (RIM) [16] [13], *Android* da *Google* [2] [13] e o Sistema Operacional iPhone (iOS) da *Apple* [17] [13].

Como uma das mais recentes plataformas do mercado computacional, o sistema operacional *Android* ganha cada vez mais espaço no mercado mundial, principalmente devido a ser *open-source* [2]. Os fabricantes de sistemas operacionais móveis, com o objetivo de conquistar mercado com seus equipamentos, oferecem vários tipos de aplicativos que podem ser adquiridos através da Internet, em seus sites oficiais. Isso faz com que usuários sintam-se mais confiantes em fazer *download* de aplicativos pela Internet e instalá-los em seus dispositivos móveis. No entanto, essa confiança pode ser bastante prejudicial, pois os aplicativos podem estar infectados com *software* malicioso (*vírus, worm, trojan, malwares*) [18].

Se tal infecção acontecer, poder-se-á dar acesso a recursos não autorizados dos *Smartphones* e ocorrer furto de dados importantes, compreendido (furto) como ato de subtrair, para si ou para outrem, coisa alheia móvel. Diante deste cenário de risco, há uma necessidade de ofertar uma segurança eficaz para plataforma dos sistemas operacionais móveis, como imperativo de natureza legal.

Nesse contexto, é importante desenvolver novas técnicas de aprendizado e ferramentas de defesas contra ataques digitais. Como o uso de *Honeypot* com o objetivo de detectar ataques internos e/ou ataques externos, coletar código malicioso (*malwares*) [19].

1.1 Cenário e Definição do Problema

A telefonia móvel nunca esteve tão desenvolvida. Com a terceira geração 3G de comunicação sem fio, houve um crescimento enorme no uso de dispositivos móveis com microprocessadores avançados. Esses recursos fornecem a base para um novo ambiente computacional, repleto de oportunidades, como por exemplo, Internet por banda larga, pagamento por celular, tv por assinatura e serviços *Push to talk*. Os empresários têm investido fortemente no mercado de aplicação para a telefonia móvel. A mistura atual de informação e acessibilidade proporcionada pelos aplicativos dos telefones móveis, sementes oriundas de estilos de vida empresarial e social, abrem oportunidades de lucrar com a fragilidade dos usuários [18].

Ataques a dispositivos móveis, destacando os *Smartphones*, estão se tornando cada vez mais comum nos dias atuais, especialmente por esses dispositivos estarem conectados a Internet a todo tempo, através de redes 3G ou conectados a redes *wireless*, sendo que os ataques atuais variam entre *worms, botnets* e *trojans* [4].

O sistema operacional *Android* está em constante crescimento de aceitação pelos usuários de dispositivos móveis, conforme pesquisa realizada pela empresa *comScore Mobiles* [9]. A *Google* com o seu sistema operacional *Android* ocupava em Dezembro de 2011 o primeiro lugar em vendas de *Smartphones* nos Estados Unidos, correspondendo a 47,3% com assinantes a partir de 13 anos de idade, conforme Tabela 1.1.

Total de Usuários de Smartphones com mais de 13 anos nos U.S., Dezembro 2011	
Plataforma	Usuários de *Smartphones* (%)
Google	47,3%
Apple	29,6%
RIM	16,0%
Microsoft	4,7%
Symbian	1,4%

Tabela 1.1: Vendas de *Smartphones* nos *U.S.* [9]

A empresa *TrendLabs* [1] em uma pesquisa recente relata que *malwares* móveis aparentemente tomaram o mundo como uma tempestade, pegando usuários desprevenidos [1]. Na Figura 1.1 observa-se a evolução no crescimento do volume de *malwares* móveis, em especial para o sistema operacional *Android* [1].

Figura 1.1: Crescimento em volume de *Malwares* móvel para *Android* em 2011 [1]

[1]O TrendLabs é um centro multinacional de pesquisa, desenvolvimento e suporte com o compromisso de vigilância de ameaças e prevenção de ataques digitais. [1]

Diante do cenário descrito, consideremos o seguinte questionamento, como conhecer e aprender com os ataques em dispositivos móveis e diminuir os danos causados por esses ataques?

Com o objetivo de conhecer e aprender com os novos tipos de ataques cibernéticos, que ocorrem diariamente aos dispositivos móveis e diminuir os danos causados por eles, propõe-se a construção de um mecanismo para desenvolver um *Honeypot* para o sistema operacional *Android*.

1.2 Objetivos Gerais e Específicos

Este trabalho propõe, de forma geral, o desenvolvimento de um modelo de técnicas que possa aprender com os ataques cibernéticos ao sistema operacional *Android*.

No sentido de alcançar o objetivo geral pretendido, buscar-se-á atingir os seguintes objetivos específicos:

- Desenvolver um *Framework* para *Honeypot* voltado para o sistema operacional *Android*;
- Integrar a arquitetura proposta ao projeto NIDIA,[2] direta e indiretamente;
- Desenvolver um *Honeypot* de baixa interatividade a partir do *Framework* gerado.

1.3 Organização do Trabalho

A presente dissertação está dividida em cinco capítulos. No primeiro têm-se uma visão geral do trabalho proposto e os objetivos a serem atingidos.

O Capítulo 2 apresenta a fundamentação teórica, mostram-se os conceitos da base da proposta, mostrando a arquitetura e funcionalidades do sistema operacional *Android*, apresentando alguns conceitos básicos da segurança de dados, descreve a definição e características de um *Honeypot*, descreve uma taxonomia

[2]O Projeto NIDIA (*Network Intrusion Detection System Based on Intelligent Agents*) foi criado em 2001, na UFMA

sobre ataques virtuais a dispositivos móveis e mostra o fundamentos básicos de um *Framework*.

O Capítulo 3 aborda algumas pesquisas já desenvolvidas sobre os sistemas operacionais móveis, descreve sobre *Honeypots* e sobre *Honeypots* para dispositivos móveis, suas contribuições, vantagens e desvantagens.

O Capítulo 4 apresenta o mecanismo gerado a partir do objetivo, o desenvolvimento de um *Framework* para criação de *Honeypot* virtual e demonstra *Honeypot* virtual gerado a partir do *Framework* para o sistema operacional *Android*.

O Capítulo 5 apresenta os testes feitos com a instalação do *HoneypotLabsac* no sistema operacional Android em dispositivo móvel real e na arquitetura x86 através da virtualização e quais os resultados obtidos.

O Capítulo 6 traz a conclusão, ressaltando as contribuições, limitações, desafios e trabalhos futuros.

2 Fundamentação Teórica

Neste capítulo englobam-se definições e características do sistema operacional *Android* e suas variações; definições e aplicabilidade de *Honeypots*, contextualização de ataques virtuais a dispositivos móveis e definições de *Framework*.

2.1 Sistema Operacional *Android*

O sistema operacional *Android* é uma plataforma completa para dispositivos móveis [20], especificamente para aparelhos celulares inteligentes (*Smartphones*). Essa plataforma é constituída de sistema operacional, *middleware*, aplicativos e interface com o usuário. O *Android* é o primeiro projeto de uma plataforma *open-source*, com licença de uso Apache Software Foundation (ASF) [21]. A licença do tipo ASF permite que alterações sejam feitas no código fonte e não sejam obrigatoriamente compartilhadas [21]. O sistema foi inicialmente desenvolvido pela *Google*, e lançado no mercado de aparelhos celulares em 2008 [20] [19]. Logo em seguida, a *Google* criou alianças com outras empresas de vários segmentos no mercado de telefonia celular para dar continuidade ao projeto. Essa aliança foi denominada de Open Handset Alliance (OHA), que atualmente conta com 84 empresas que englobam toda estrutura que faz parte do processo de telefonia móvel [2] [22] [23].

O sistema operacional *Android* tem em sua base o *kernel* do *Linux*, versão 2.6 [24] [23]. Este sofreu alterações para que ganhasse todas as funcionalidades de um dispositivo móvel (*Smartphone*). Dentre essas alterações destacam-se:

- *binder* – é usado para a comunicação entre processos. Toda comunicação entre processos no *Android* passa pelo *binder*. Para o desenvolvedor de aplicações *Android*, o processo é transparente, já que é abstraído pelas bibliotecas do sistema. O *binder* garante que nenhum processo tem acesso à região de memória de outro processo;

- *ashmem* – um novo mecanismo de compartilhamento de memória, onde dois processos podem se comunicar através desta região compartilhada. É mais leve e simples de usar, e tem melhor suporte a dispositivos com pouca memória, já que tem a capacidade de descartar regiões de memória compartilhada de maneira segura em caso de pouco espaço disponível;

- *wakelocks* – se um dispositivo *Android* ficar um tempo sem ser usado, entrará em modo de baixo consumo para garantir economia de bateria;

- *oom handling* – controla o uso de memória do sistema e encerra processos se verificar que a memória disponível está inferior a um valor mínimo aceitável.

O sistema de arquivos ficou bem diferente de um sistema *Linux* convencional. Tem dois diretórios principais; o *data*, que armazena os dados das aplicações, e o *system*, com as bibliotecas *system/lib*, serviços (*system/bin* e*system/xbin*) e aplicações *java* (*system/app*). O *Kernel* do *Linux* funciona, como uma camada de abstração entre os componentes de hardware e software. Após essas alterações passou a não ser considerada uma distribuição *Linux* convencional [2]. O sistema operacional *Android* utiliza uma máquina virtual chamada *Dalvik* totalmente otimizada para trabalhar com dispositivos móveis que executa os aplicativos desenvolvidos na linguagem de programação Java [20].

A Figura 2.1 apresenta a arquitetura do sistema operacional *Android*, composta por quatro níveis: *Applications, Applicattion Framework, Libraries,* e *Linux Kernel*, detalhadas a seguir.

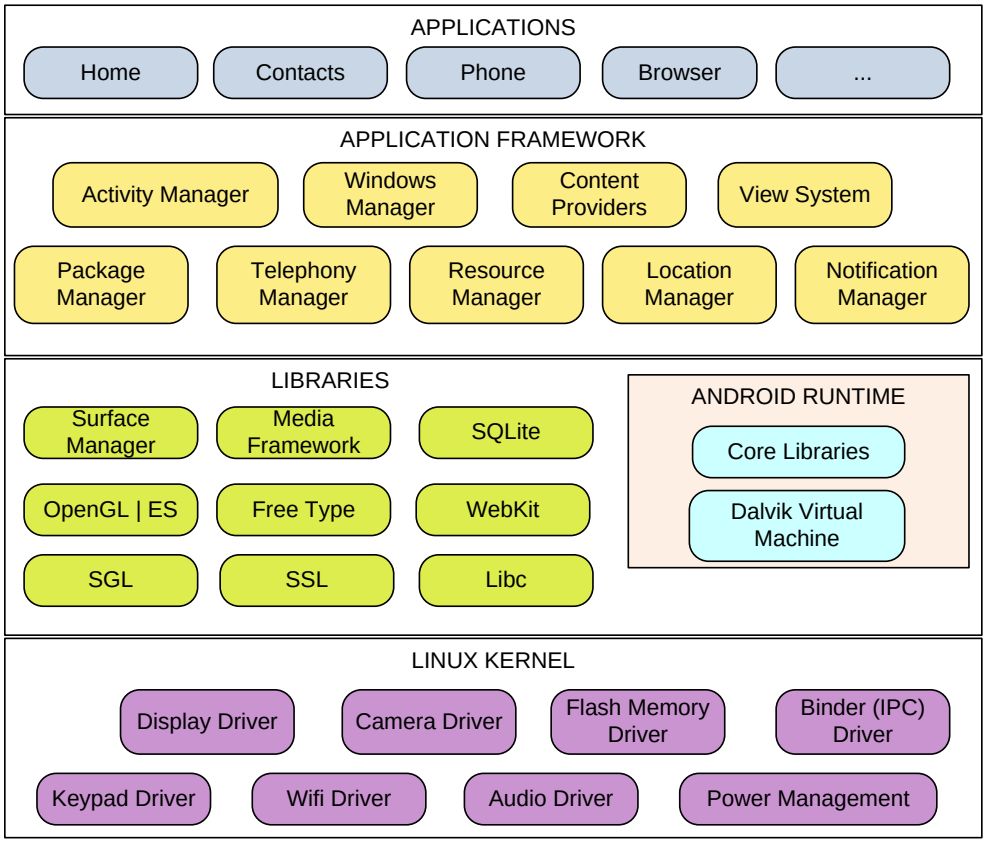

Figura 2.1: Arquitetura sistema operacional *Android* [2]

Applications (Aplicativos) – a camada superior é a de aplicativos, onde se encontram todos os aplicativos imprescindíveis do *Android* (escrito na linguagem de programação Java), incluindo um cliente de e-mail, programa Short Message Service (SMS), calendário, mapas, navegadores, gerenciador de contatos, agendas e outros que venham a ser implementados pela comunidade de desenvolvedores de software.

Application Framework (Arcabouço de aplicações) – é a base para as aplicações, as quais os desenvolvedores têm acesso completo. O *Android* lhes oferece a capacidade de criar aplicações extremamente ricas e inovadoras. Os programadores estão livres para aproveitarem o *hardware* do dispositivo, as informações de localização de acesso, a execução de serviços de fundo, definirem alarmes, notificações para adicionar a barra de status, entre outros. O *Framework* foi criado para facilitar a vida dos desenvolvedores através da reutilização de componentes. Dentre os principais elementos desta camada tem-se:

- *Activity Manager* – gerencia o ciclo de vida de todas as *activities* (por Exemplo, quando iniciar e quando terminá-las), possibilitando o deslocamento de uma *activity* para outra e assim por diante;

- *Package Manager* – é utilizado pela *activity manager* para ler as informações dos pacotes de arquivos do *Android (APK's)*. O *package manager* se comunica com o restante do sistema e diz quais os pacotes estão sendo utilizados no dispositivo e quais são as capacidades destes pacotes;

- *Windows Manager* – gerencia as apresentações de todas as janelas, qual a janela estará ativa ou não, dar suporte a *activity manager*;

- *Content Providers* – proporciona o compartilhamento de dados, possibilitando a troca de informações entre os aplicativos. O Android em si inclui o *Content Providers* para gerenciar os dados, tais como áudio, vídeo, imagens e informações de contato pessoal;

- *View System* – disponibiliza todo o tratamento gráfico para a aplicação, como botões, *layout* e *frames*.

Na camada *Application Framework* (Arcabouço de aplicações) encontram-se ainda outros elementos, como *Location Service, Bluetooth Service, Wi-Fi Service, USB Service e Sensor Service*. Todos os conjuntos de componentes disponíveis na plataforma *Android* possuem acesso pleno às Application Programming Interfaces (APIs), utilizadas pelo núcleo da plataforma.

Libraries (Bibliotecas) – O *Android* inclui um conjunto de bibliotecas C / C++ usadas por diversos componentes do seu sistema operacional. Os desenvolvedores têm acesso às bibliotecas através do *Framework* de aplicações. Entre as mais importantes pode-se citar *SQLite*, um banco de dados *open-source* de alta qualidade e de domínio público que está disponível em vários sistemas operacionais móveis [24], e as bibliotecas de Mídias as quais permitem reproduzir arquivos de áudio, vídeo e fotografias com os formatos mais populares como mpeg4, JPEG, entre outros.

Android Runtime – O *Android* inclui um conjunto de bibliotecas que fornecem a maioria das funcionalidades disponíveis nas principais bibliotecas da linguagem de programação Java. A Máquina *Virtual Dalvik* – é uma máquina virtual baseada

em registradores, projetada e escrita por Dan Bornstein em conjunto com outros engenheiros da *Google*, especialmente para o sistema operacional *Android*. A Dalvik com esta estrutura baseada em registradores consegue:

- Evitar perdas de instruções e acesso desnecessário à memória;
- Alta densidade de semântica por instrução;
- 30% menos instruções;
- 35% menos códigos por unidade;
- 35% mais bytes de *stream* de instrução úteis, sendo executado 2 código por vez.

A máquina *Dalvik* foi projetada para rodar em sistemas com baixa frequência de CPU, pouca memória RAM disponível e sistema operacional sem espaço de memória *swap*. A *Dalvik* é a máquina virtual que executa os aplicativos do sistema operacional *Android*. Durante o *boot* do sistema operacional, é criada uma instância especial da *Dalvik* denominado *Zygote*, a partir desse processo, outras máquinas virtuais são instanciadas com *fork*() quando necessárias para executar outras aplicações. Esta máquina executa arquivos com extensão .dex, um formato de *bytecode* desenhado para ocupar menos espaço e carregar rapidamente. O sistema operacional *Android* permite o uso de *multi-threads* que é suportado pela máquina virtual, junto com outras utilidades, como o *Garbage Collector*, otimizado para ser executado em aplicativos com poder de processamento e memória reduzidos.

Linux Kernel – O sistema operacional *Android* é baseado no *Kernel* 2.6 do *Linux*. Essa camada é responsável pela segurança, gerenciamento de memória e gerenciamento de processos. Nele o desenvolvedor de aplicativos não terá acesso. Os aplicativos fazem uso indireto do *Kernel* através de APIs de níveis superiores.

Sistema Operacional *Android* Arquitetura x86

O projeto do sistema operacional *Android* x86 foi fundado por Chih-Wei Huang (Taiwan) e Sun Yi (EUA) [25]. A arquitetura x86 é uma modificação da versão original da arquitetura de microprocessadores Advanced RISC Machine (ARM), modificada e adaptada para o *Android* x86. Esse sistema tem como objetivo principal oferecer suporte completo para netbooks e notebooks.

2.2 Conceitos Básicos de Segurança de Dados

Em se tratando de sistemas digitais, todos os *softwares* podem apresentar vulnerabilidades na segurança. Quando presentes no código do sistema, elas permitem que vírus de computador ou operantes mal-intencionados, invadam o sistema e comprometam sua integridade. Estas vulnerabilidades são descobertas pelo próprio desenvolvedor do sistema, por usuários, ou, em um pior cenário, por criminosos. Em qualquer destes casos, o desenvolvedor precisa acelerar o processo para corrigí-las e disponibilizar uma atualização com a vulnerabilidade eliminada [26].

Um sistema seguro é aquele que fornece informações íntegras somente a usuários autenticados e autorizados, no momento em que elas são pedidas, através de requisições válidas e identificadas, não permitindo que estas informações sejam recebidas, observadas ou alteradas por terceiros não autorizados [26].

Alguns conceitos básicos sobre segurança de dados devem ser levados em consideração para identificar e tratar possíveis falhas de segurança, como: confidencialidade, integridade e disponibilidade [26].

- Confidencialidade – Garante que as informações armazenadas em um sistema de computação ou transmitidas através de uma rede de computadores, sejam acessadas ou manipuladas somente pelos usuários devidamente autorizados.
- Integridade – Garante que a informação processada ou transmitida, chegue ao seu destino exatamente da mesma forma como partiu da origem.
- Disponibilidade – Garante que o sistema de computação continue operando sem degradação de acesso e provê recursos aos usuários autorizados quando necessário.

2.3 *Honeypots* - Metodologia para Iludir

A primeira referência à implementação de mecanismos de acompanhamento das atividades de invasores data de 1988, quando Clifford Stoll tornou público a história da invasão ocorrida nos sistemas do *Lawrence Berkeley Laboratory* (LBL), ele localizou e encurralou em seu sistema o hacker Hunter. As

técnicas que ele utilizou (acompanhou e registrou todos os passos do invasor pelo sistema) para muitos autores são as percursoras dos *Honeypots* atuais.

Em 1997, Fred Cohen desenvolveu o *Deception Toolkit* (DTK), a primeira ferramenta de código aberto cujo objetivo é explicitamente iludir atacantes. Esta emula diversas vulnerabilidades e coleta as informações sobre os ataques sofridos. Nesta época surgiu o termo *Honeypot* como definição para um recurso de segurança preparado especificamente para ser sondado, atacado ou comprometido, e para registrar essas atividades.

Após o surgimento do DTK diversas outras tecnologias de *Honeypots* foram desenvolvidas, incluindo diversos produtos comerciais como o Cybercop Sting, o NetFacade e o NFR BackOfficer Friendly. Uma outra data importante é o ano de 1999, com a criação do projeto *Honeynet* por Lance Spitzner e mais 30 especialistas de segurança. [27]

Segundo a literatura existem várias definições para *Honeypots*, como mostrado a seguir:

Honeypots são ferramentas de estudos de segurança, onde sua função principal é colher informações do atacante [27]. Ainda pode-se acrescentar que os *Honeypots* são ferramentas que dispõem diversas funcionalidades, que podem ser implementadas a partir do que se quer alcançar [10].

Os autores Hoepers et al. [19], definem *Honeypot* como um recurso computacional de segurança dedicado a ser sondado, atacado ou comprometido. Já Spitzner [10] em sua definição diz, que *Honeypot* é uma ferramenta de estudos de segurança onde sua função principal é colher informações do atacante, através de monitoramento.

Os *Honeypots* podem ser bastante úteis no âmbito de segurança da rede, pois têm como objetivo principal enganar o invasor, ou seja, fazer com que o atacante pense que está interagindo realmente com um sistema operacional ou um serviço qualquer oferecido. Mas na verdade o atacante está sendo monitorado e todos os seus passos sendo gravados em um arquivo de log, contendo no mínimo as seguintes informações: data e hora do ataque, IP de origem (atacante) e IP de destino (atacado) e tipo de ataque. Depois de gerado o arquivo de log e gravado em outra máquina esse arquivo pode ser investigado. Após a análise desse arquivo de log o administrador da rede

tem como tomar algumas medidas com o objetivo de diminuir as vulnerabilidades da rede.

A comunidade de segurança de redes enfrenta diariamente vários desafios. Um deles é obter informações valiosas através dos dados brutos coletados. Diariamente empresas coletam um grande volume de dados, incluindo logs de *Firewall* e de sistemas, e alertas de detecção de intrusão. No entanto, toda essa quantidade de dados pode se transformar em apenas mais um amontoado de ocorrências, se não for usada corretamente [10]. Por outro lado, *Honeypots* coletam uma pequena quantidade de dados, mas estes são de grande valor pelo fato de não fazerem parte do sistema de produção, todo tráfego destinado a eles é considerado malicioso. Esse tráfego pode ser considerado valiosíssimo, podendo ser usado em modelos estatísticos, análises periódicas, detecção de ataques, ou até mesmo pesquisa de metodologias ofensivas [10].

Na realidade, os *Honeypots* não fazem nenhum tipo de prevenção, os mesmos fornecem informações adicionais de valor inestimável [27]. Os mesmos podem ser divididos em nível de interatividade, ou seja, em que nível de interatividade os serviços oferecidos estão sendo executados. Alguns autores [19] [6] [28] declaram apenas dois níveis. São eles:

- Baixa Interatividade – Provê serviços falsos, que estão sendo executados aguardando conexões em uma determinada porta e interage com o atacante enviando respostas falsas. Os serviços podem ser emulados individualmente por ferramentas, sendo a mais conhecida o *Honeyd*, como também podem ser colocados em funcionamento através de máquinas virtuais [6].

- Alta interatividade – os atacantes interagem diretamente com os sistemas operacionais, aplicações ou serviços reais. Este tipo oferece uma maior interação com o atacante, visto que o acesso acontece a um dispositivo real com todas suas características. Porém, nesse caso é necessário mais atenção na sua implantação, pois demanda uma maior estrutura de segurança para evitar que sejam comprometidos, podendo ser utilizado como ponte para a realização de novos ataques [19] [28].

O Autor Marcelo [27] relata que os autores Spitzner, Baumann e Northent fazem referência também a *Honeypots* de média interatividade, situando-se entre a baixa e alta interatividade. Na Tabela 2.1, tem-se uma comparação entre os níveis de interatividade possíveis em um *Honeypot*.

Nível de interatividade	Trabalho para instalar e configurar	Trabalho para ativar e manter	Coleta de informações	Nível de risco
Baixo	Fácil	Fácil	Limitada	Baixo
Médio	Mediano	Mediano	Variável	Médio
Alto	Difícil	Difícil	Extensa	Alto

Tabela 2.1: Níveis de Interatividade [10]

O autor Provos [6] faz diferença entre *Honeypots* físicos e virtuais. *Honeypot* físico é uma máquina real na rede com seu próprio endereço IP. Um *Honeypot* virtual é a simulação de serviços ou de um sistema operacional por outra máquina que responde ao tráfego de rede enviado para o *Honeypot* virtual.

De acordo com os autores [10] [29] pode-se citar algumas vantagens em se usar *Honeypots*, são elas:

- Pequenos conjuntos de dados, de grande valor – os *Honeypots* capturam pequenas quantidades de dados, mas estes são de grande valor, já que todo tráfego capturado de um *Honeypot* representam atividades maliciosas.
- Novas ferramentas e táticas – os *Honeypots* são projetados para capturar toda e qualquer informação relacionada às atividades que envolvam interação com o mesmo, podendo revelar novas ferramentas e novas táticas empregadas pelos atacantes em suas investidas.
- Informação – *Honeypots* coletam todas as informações que chegam até eles e poucas ferramentas podem registrar com tanta eficácia este tipo de captura, assim como os *Honeypots* fazem.
- Simplicidade – os *Honeypots* de baixa interatividade são conceitualmente simples. Não existem algorítimos complexos tanto no processo de desenvolvimento

quanto no processo de implantação, não há a necessidade de se manter tabelas de estados ou assinaturas que devam ser atualizadas periodicamente. Quanto mais simples forem as tecnologias envolvidas, menos suscetíveis a falhas ou má-configurações, estas serão.

- Encriptação e IPv6 – diferente de outras tecnologias, tais como Sistemas de Detecção de Intrusão, pois trabalham bem em ambientes criptografados ou com IPv6. Não importa o que seja lançado sobre os *Honeypots*, eles irão detectar e capturar tais informações.

Assim como qualquer tecnologia, os *Honeypots* possuem seus pontos fracos. Por isso, eles devem ser utilizados como um complemento para a segurança da rede de uma instituição e não substituir as boas práticas de segurança, as políticas de segurança, o sistema de gerenciamento de correções de segurança (*patches*) e outras ferramentas de segurança, como *Firewall* e *IDS*. Os autores Spitzner e Herrera [10] [29] ressaltam como principais desvantagens:

- Visão limitada de tráfego: Os *Honeypots* podem rastrear e capturar atividades que interagem diretamente com eles, e por isso, não irão capturar ataques direcionados a outros sistemas, como por exemplo *Firewall* e *IDS*, a menos que, o atacante ou a ameaça, interaja também de alguma maneira com os *Honeypots*.

- Risco: Todas as tecnologias de segurança envolvem riscos. Assim como *Firewalls* podem ser penetrados, mecanismos de encriptação correm o risco de serem quebrados e Sistemas de Detecção de Intrusão podem falhar na detecção de ataques, os *Honeypots* não são infalíveis. Especificamente, os *Honeypots* estão propensos a serem tomados por atacantes que obtenham sucesso na invasão nos sistemas hospedeiros, por eventuais falhas de configuração ou por conseguirem escalar de maneira adequada seus privilégios. Dependendo do tipo a ser implantado, este pode oferecer tanto quanto, ou até menos vulnerabilidade, que um Sistema de Detecção de Intrusão falho, enquanto que existem determinados tipos que oferecem riscos reais às redes, uma vez que sejam comprometidos.

- Ausência de tráfego: ocorre quando a tecnologia não é bem aplicada, implicando em gastos sem retorno, já que nada será monitorado.

Ao pesquisar sobre *Honeypot* verificou-se alguns desafios a serem superados, são eles:

- Monitoramento – um *Honeypot* tem como uma de suas características o monitoramento de suas atividades;
- Log para auditoria – é vital manter um arquivo de log intacto para reconstrução de ataques;
- Contenção – é sabido que *Honeypots* são feitos para serem comprometidos, porém o *Honeypot* comprometido não pode servir de trampolim para novos ataques;
- Visibilidade – o *Honeypot* quanto mais atraente e visível melhor, pois vai chamar mais atenção do atacante [4] [6].

Muitos trabalhos são realizados no sentido de explorar os benefícios oferecidos pelos recursos dos *Honeypots* em segurança de redes. Alguns realizam aplicações em outras tecnologias como: redes sem fio [30], redes de sensores [31], redes veiculares [32], e em dispositivos móveis [4] [5]. Também os utilizam como recursos educacionais, no ensino e pesquisa, na geração de dados estatísticos sobre tráfego malicioso em redes e na Internet [33] e no combate a *spams* [34].

2.4 Ataques Virtuais a Dispositivos Móveis

Os ataques virtuais a dispositivos móveis não são recentes. O primeiro vírus virtual desenvolvido para telefone celular foi em 2004. Esse vírus denominado "*Cabir*" teve sua propagação via *Bluetooth* e visava exclusivamente o sistema operacional *Symbiam* [14]. A série 60 do *Symbiam* foi seguido por uma sequência de vírus e *trojans*, que eram propagados através do *Bluetooth* e do *Multimedia Messaging Service (MMS)* Serviço de Mensagem Multimídia [18]. Como não existe uma padronização universal de nomenclaturas de ameaças a dispositivos móveis, Dunham, K., et al. [35] usa uma terminologia própria para segurança móvel. São elas:

- *Ad/Spyware* – são programas potencialmente indesejáveis, que podem incluir *End User License Agreement* (*EULA*). Este tipo de programa executa várias

ações indesejáveis sem o consentimento do usuário. Comumente envolve o aparecimento de anúncios (*pop-up's*) e ainda estuda o perfil do usuário.

- *Bluebug* – explora uma vulnerabilidade do *Bluetooth* para efetuar chamadas telefônicas com valor diferenciado (mais caro).

- *BlueChop* – é um ataque de negação de serviço que interrompe uma rede Piconet[1].

- *BruteForce* ou (procura exaustiva) – é um algoritmo comum, mas de uso geral que consiste em enumerar todas as soluções possíveis e verificar se cada uma satisfaz o problema, por exemplo, um algoritmo para encontrar os divisores de um número natural **n** é enumerar todos os inteiros de 1 a **n**, e verificar para cada um, se ele dividido por **n** resulta em resto 0.

- *Denial-of-Service (DoS)* – é um ataque destinado a prejudicar e/ou negar o uso de um dispositivo móvel, serviço ou rede.

- *Exploit* – pode ser um *software* ou ações que tentam aproveitar uma vulnerabilidade para executar ações indesejáveis.

- *Hacking defaults* – são técnicas usadas para invadir aparelhos ou *software* que utilizam senhas, definições e/ou configuração padrão.

- *MalwareMóvel* – é um *software* que executa ações maliciosas em dispositivos móveis.

- *Snarf* – é o furto não autorizado de dados. *Snarf* é a gíria usada para furtar informações de outro dispositivo. Por exemplo, o *Pbstealer* é um *Information scavengers* (caçadores de informação) que envia o livro de endereço do usuário através do *Bluetooth* para outros dispositivos, que estão com os seus *Bluetooths* ligados dentro de uma área de alcance de 10 metros em média. Os telefones celulares ou *Smartphones* são os alvos prediletos desse tipo de *malware*, pela sua organização estrutural e também por uma *API* onde a maioria dos sistemas operacionais móveis tem, permitindo a todos os aplicativos acesso direto ao livro de endereço [18].

[1]Uma rede Piconet é uma rede *Bluetooth* formada por até 8 dispositivos, sendo 1 mestre e os demais escravos

- *Trojan* – é um *software* malicioso e se disfarça de algo que não é. Não se replica, apenas furta informação e na maioria das vezes envia para quem o enviou ou para seu criador. Exemplos: *Trojan* Destrutivos *Skrulls* e o *Cardblock*. *Skrulls* foram os primeiros *trojans* destrutivos. Quando instalados escrevem *non-functioning* (não-funcionamento) nas aplicações, as mesmas ficam inúteis (inacessíveis) e seus ícones são substituídos por um ícone de uma caveira e ossos cruzados. *Cardblock* define uma senha aleatória no cartão de memória removível, fazendo os dados do usuário inacessíveis. Tem ainda o *Trojan Drever*, que tem como objetivo desabilitar o *software* antivírus no dispositivo móvel [18].

- *Vírus* – é um *software* malicioso, intencionalmente se multiplica através de programas ou arquivos, em um dispositivo móvel, com o objetivo de alterar a forma como esse dispositivo funciona. a disseminação de um vírus ocorre seguindo dois critérios, é auto executável e se replica.

- *Worm* – é *software* malicioso, cria uma cópia de si mesmos, isto é, cria um clone e tem como objetivo a disseminação pela rede.

A maioria das ameaças virtuais descritas para dispositivos móveis foram oriundas dos computadores PC's (Desktop) [18]. Essas ameaças sofreram algumas modificações para serem utilizados em dispositivos móveis. Alguns podem até serem versões dos originais, mas seus objetivos são obter informações não autorizadas pelo usuário [18] [35].

2.5 Framework

Framework tem várias definições dependendo de sua aplicabilidade. Aqui se define *Framework* para desenvolvimento de *software* (aplicativos): *Frameworks* são técnicas de reuso orientadas a objetos. Uma das definições mais usada pelo autor *Johnson* [36] é, "Um *Framework* é um padrão reutilizável para todo ou parte de um sistema, que representa através de conjuntos de classes abstratas como as instâncias destas classes interagem". Outra definição bem comum é "Um *Framework* é o esqueleto de uma aplicação que pode ser customizado por um desenvolvedor de aplicação". Ainda pode-se dizer, que um *Framework* geralmente é representado por um conjunto

mínimo de classes abstratas com suas interações entre as instâncias. Os *Frameworks* podem ser classificados em caixa-branca e caixa-preta [36] [37]:

- Os *Frameworks* caixa-branca, baseiam-se nos mecanismos de herança e ligação dinâmica presentes em orientação a objetos. Os recursos existentes em um *Framework* caixa-branca são reutilizados e estendidos a partir de herança de classes do *Framework* e sobrecarga de métodos pré-definidos. Métodos são definidos em interfaces ou classes abstratas e devem necessariamente ser implementados por uma aplicação.

- Os *Frameworks* caixa-preta, são baseados em componentes de *software*. A extensão da arquitetura é feita a partir de interfaces definidas para componentes. Os recursos existentes são reutilizados e estendidos por meio de definição de um componente adequado a uma interface específica e integração de componentes em um *Framework*, que utiliza padrões de projeto.

Quanto ao contexto de utilização os *Frameworks* são classificados em: *Framework* de Infraestrutura de Sistema, *Framework* de Integração de Middleware e *Framework* de Aplicação [37].

- *Frameworks* de infraestrutura de sistema tratam de questões de projeto como sistemas operacionais, comunicação, interfaces gráficas e linguagens de programação.

- *Frameworks* de integração de Middleware são responsáveis por integrar aplicações distribuídas e componentes em uma mesma arquitetura.

- *Frameworks* de aplicação tratam de questões de projeto de domínios de aplicação, como telecomunicações, finanças, produção e educação.

2.6 Considerações Finais

Neste capítulo foram apresentados os principais conceitos necessários para o entendimento deste trabalho. De forma ampla, foram vistos: Uma visão detalhada da plataforma *Android* desenvolvida pela *Google*. Foram descritos, também,

o sistema operacional *Android* para plataforma x86, usada nos dispositivos com microprocessadores Intel. Apresentaram-se os conceitos básicos para segurança de dados e definições sobre *Honeypot* e suas características.

Foram destacados os ataques virtuais para dispositivos móveis, apresentando-se uma taxonomia própria e exemplificando alguns tipos de *malware* móvel. Por fim, foi definido *Framework* usado para desenvolvimento de *software*, e classificou-se *Framework* quanto à utilização. No terceiro capítulo será descrito de forma direta e indiretamente os trabalhos relacionados à proposta de dissertação.

3 Trabalhos Relacionados

Neste capítulo será abordada a revisão bibliográfica realizada com o objetivo de identificar pesquisas desenvolvidas sobre sistemas operacionais móveis, suas contribuições, vantagens e desvantagens. O estado da arte sobre o assunto serviu de fundamentação teórica e contribuiu para o desenvolvimento do método proposto nesta dissertação. Para melhor entendimento, os trabalhos relacionados são divididos em seções, de acordo com a classificação dos sistemas operacionais para dispositivos móveis, *Honeypots* em geral e sobre o projeto Network Intrusion Detection System Based on Intelligent Agents (NIDIA).

3.1 Classificações de Sistemas Operacionais Móveis

Nesta Seção consideram-se os trabalhos que, de alguma forma, classificam os sistemas operacionais móveis em diversas modalidades. Com esta classificação foi possível identificar qual o sistema operacional a ser utilizado como base para desenvolvimento da proposta desta dissertação.

Secure Software Installation on Smartphones

O Autor Barrera [3] analisou as plataformas de dispositivos móveis mais populares a partir de 2010, *Android, BlackBerry, Apple iOS* e *Symbian*. Apresentou uma classificação de instalação de *softwares* de terceiros em *Smartphones* e mostrou como cada fornecedor de sistema operacional trata as aplicações quando o assunto é segurança. Ele distribuiu em três modelos genéricos a classificação de instalação de *softwares* desenvolvidos por terceiros, *Walled Garden Model, Guardian Model* e *End-user Control Model*. A Figura 3.1 ilustra como foi definido cada um destes modelos e em seguida serão detalhados.

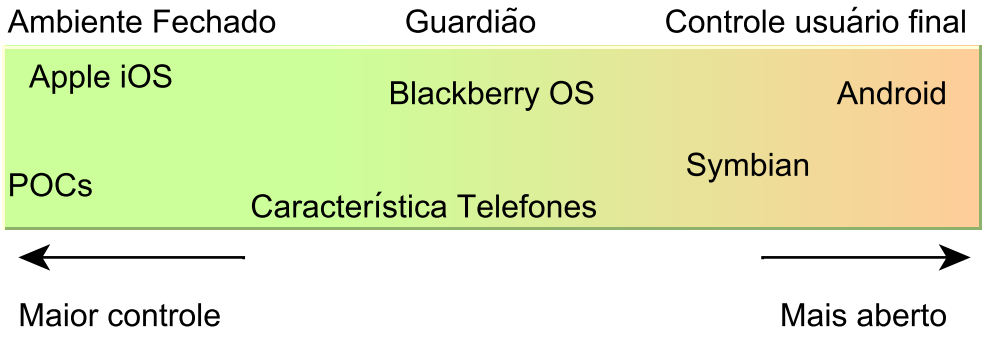

Figura 3.1: Classificação de sistemas existentes [3]

- *Walled Garden Model (Modelo Ambiente Fechado) (WGM)* – No *WGM* o fornecedor do *Smartphone* ou do sistema operacional tem o controle total sobre instalação de *softwares* de terceiros nos dispositivos do usuário final. Os aplicativos precisam ser avaliados e liberados pelo fornecedor do sistema operacional para poderem ser baixados através de seus mercados.

- *Guardian Model (Modelo Guardião) (GM)* – No *GM* as decisões de segurança são delegadas para qualquer pessoa que tenha informações suficientes de manter uma boa política de segurança, desde o fornecedor do sistema operacional, da operadora de celular ou de um administrador de sistema de uma empresa, que já controla a política em outros dispositivos.

- *End-user Control (Modelo Controle Usuário-final) (EUC)* – O modelo *EUC* considera o usuário como o responsável por toda a instalação do *software* e, pelas decisões de segurança que precisam ser tomadas para a devida instalação do aplicativo. Os usuários são livres para instalarem *software* a partir de qualquer fonte (site, cartão de memória, *app Market* do fornecedor dos dispositivos ou dos sistemas operacionais), compreendendo o risco de que algum ou todos os aplicativos podem estar infectados com arquivos maliciosos, pois não se tem uma avaliação efetiva dos aplicativos ou ainda dos desenvolvedores.

Conforme a Figura 3.1, o trabalho *Secure Software Installation on Smartphones* classifica as plataformas de *Smartphones* entre modelos ambientes fechado, modelo guardião e o modelo controlado pelo usuário, conforme descrito a seguir:

- *iPhone* – As aplicações desenvolvidas para o iOS são escritas em *Objective-C*. Para melhorar a segurança no iOS, por padrão, nenhuma aplicação desenvolvida por terceiros pode ler ou gravar dados fora do seu próprio diretório. Restringir aplicativos dessa maneira requer que os desenvolvedores usem APIs registradas para acessar recursos protegidos. As aplicações de terceiros para serem publicadas na *iTunes App Store* precisam primeiro serem aprovadas pela *Apple*, por ser esta a forma de controlar os aplicativos. Barrera [3] classifica a plataforma *iPhone* como ambiente fechado.

- *Android* – As aplicações são desenvolvidas na linguagem de programação Java. Por padrão, os aplicativos só tem acesso de leitura e escrita para arquivos em seu próprio diretório. A *Google* tem participação mínima quando os aplicativos de terceiros são disponibilizados em seu *Android Market* e nenhuma participação quando o aplicativo é baixado em qualquer site. Ela só remove a aplicação do *Android Market* quando o conteúdo é considerado incompatível com os termos de uso ou após a confirmação do relato de atividades maliciosas. A plataforma *Android* é classificada por Barrera, [3] como uma plataforma onde o usuário final tem o controle sobre as instalações de novos aplicativos no dispositivo móvel.

- *Blackberry OS* – Foi desenvolvido pela *Research In Motion (RIM)*, suporta aplicativos de terceiros desenvolvidos na linguagem de programação Java. As aplicações de terceiros são distribuídas em seu *appworld* com prévia autorização e através de sites de terceiros, sem autorização da *RIM*. O *Blackberry OS* foi projetado para o mercado empresarial; a empresa através do responsável pela área de segurança de dados, faz um controle dos aplicativos a serem instalados nos aparelhos por ela distribuída a seus funcionários. Por exemplo, os administradores podem criar uma política para decidir que os aplicativos baixados de sites de terceiros não são permitidos, mas esses aplicativos são instalados por meio de *appworld* da *Research In Motion (RIM)*. Barrera [3] classifica a plataforma *Blackberry OS* como uma plataforma gerenciada através de um profissional, assemelhando-se ao modelo Guardião.

- *Symbian* – Sistema operacional da Nokia, o mais antigo, existe desde 1990. Os aplicativos de terceiros podem ser desenvolvidos em várias linguagens de programação como *Symbian C/C++, JavaME, Python, Ruby*, entre outras.

Os aplicativos para serem instalados nos dispositivos precisam ser assinados digitalmente pela Fundação *Symbian* através do programa *Symbian signed*, mas a plataforma permite ao usuário configurar, se os aplicativos não autorizados devem ser: sempre permitidos, nunca ou solicitar permissão a um usuário de cada vez. Para Barrera [3] a plataforma do *Symbian*, fica entre os modelos de guardião e o modelo onde o controle é do usuário final.

Assim concluímos que não há uma padronização para instalação de *software* de terceiros quando o assunto tratado é segurança. A padronização fica dependendo de fatores como: política dos fornecedores de *Smartphones*; política dos fornecedores dos sistemas operacionais e um conhecimento mais aprofundado dos usuários.

Operating Systems for Mobile Computing

Os autores Sharon e Anderson, [38] fazem uma comparação entre os sistemas operacionais para dispositivos móveis mostrando as dificuldades encontradas pelos usuários. São elas:

- Interface de difícil manuseio;
- Digitação de texto em dispositivo pequeno (Teclado e Tela);
- Velocidade da rede;
- Navegação na web móvel;
- Falta de aplicações.

Em suas explanações eles relatam o quê levou o sistema operacional *Android* a se destacar em relação aos seus concorrentes, considerando as dificuldades relatadas anteriormente:

- Interface de difícil manuseio – O *Framework* do *Android* permite que os fabricantes de *hardware* possam construir interfaces amigáveis, como telas *touch-screen* (sensíveis ao toque).
- Digitação de texto em dispositivo pequeno (Teclado e Tela) – O *Android* permite uso de teclados deslizantes *QWERTY*, como o *Blackberry*, e por padrão o *Android* usa o teclado base do telefone ou um teclado *touch-screen* (sensíveis ao toque).

- Velocidade da rede – Todos os dispositivos existentes na proposta do *Android* serão baseados em redes 3G.

- Navegação na web móvel – O *Android* fornece um *built-in* navegador de web completo, capaz de carregar pagina web reais, não apenas as versões pequenas para dispositivos móveis.

- Falta de aplicações – O *Android* é um sistema operacional *open-source*. Os usuários podem desenvolver aplicativos baseado na linguagem de programação Java e implantá-los em seus dispositivos sem maiores dificuldades. Seu concorrente mais próximo é o *iOS* da *Apple*. Pela politica de segurança da *Apple*, um aplicativo deve ser aprovado antes de ser ofertado aos seus usuários.

Desse modo, concluíram que dos sistemas operacionais que existem no mercado para dispositivos móveis, o *Android* está estrategicamente bem posicionado para ser a tecnologia que permite aos fabricantes de *hardware*, engenheiros, empresas de desenvolvimento e indivíduos inovadores possam trabalhar em conjunto para produzir algo realmente surpreendente.

A survey of platforms for mobile networks research

O autor Oliver [11] analisa as cinco plataformas de *Smartphones* mais populares *Android, BlackBerry, iOS da Apple, Symbian* e *Windows Mobile*. Esta análise tem como foco as APIs ou Interface de Programação de Aplicativos. Os *Smartphones* são fabricados com funcionalidade física diferentes, por isso, as APIs da plataforma poderão não estar disponível em todos os *Smartphones*. Na Tabela 3.1 vemos uma comparação entre estas APIs, mostrando as vantagens e desvantagens no seu uso.

	Android (Linux)	BlackBerry	iPhone	Symbian	Windos Mobile
Network scanning	●	○	●	˜	●
Interface selection	○	●	●	●	●
Bluetooth I/O	○	●	○	●	●
Interface control	●	●	○	○	●
Background processing	●	●	●	●	●
Energy monitoring	●	●	●	●	●
Power saving control	●	●	˜	●	●
Low memory management	●	●	●	●	●
Persistent storage	●	●	●	●	●
Location sensing	●	˜	●	●	●

Tabela 3.1: Resumo dos requisitos (Satisfeito(●) Parcialmente satisfeito(˜) Não satisfeito(○) [11]

Cada plataforma tem o seu próprio ambiente de desenvolvimento que suporta diferentes conjuntos de APIs e suas linguagens de programação. Earl Oliver, [11] chegou à conclusão que as plataformas móveis não satisfazem plenamente as necessidades dos pesquisadores, e sua escolha resume-se nas necessidades individuais de cada pesquisador. Não há qualquer plataforma ou dispositivo que possa fazer tudo de forma confiável [11].

Esta Seção serviu de base para nortear a determinação do sistema operacional *Android*, objeto de estudo deste trabalho, é uma plataforma *open-source*, onde o usuário final pode baixar aplicativo de qualquer site ou do *app Market*, seus aplicativos são desenvolvidos na linguagem de programação Java, e os desenvolvedores e/ou usuários avançados, podem desenvolver aplicações e instalar sem maiores dificuldades.

Pode-se observar que o sistema operacional *Android* está ganhando cada vez mais o mercado mundial de sistema operacional para dispositivos móveis conforme foi descrito na introdução, consequentemente, cresce também a quantidade de ataques cibernéticos ao mesmo.

3.2 Honeypots para Dispositivos Móveis

Nesta Seção serão apresentadas pesquisas fortemente relacionadas com o tema desta dissertação.

Poster: honeydroid – creating a smartphone honeypot

Os autores Collin et al. [4] criam um *Honeypot* para *Smartphone* que usa o sistema operacional *Android* denominado de *HoneyDroid*. Eles optaram por trabalhar diretamente com *hardware* do telefone móvel real para que tivessem acesso à rede da operadora móvel com a intenção de aumentar a visibilidade de seu *Honeypot*.

O *HoneyDroid* virtualizou os principais dispositivos (Memória flash, Modem e Wireless Fidelity (WiFi)) (tópico 1 na Figura 3.2) para ter o controle de toda interação entre o sistema operacional *Android* e o *hardware*. Desta forma há um monitoramento eficiente, gerando arquivos de log e armazenamento destes em um local não acessível ao sistema operacional *Android*. Essa atividade é feita pelo Virtual *Harddisk* ao executar o componente *Snapshot* (tópico 2 na Figura 3.2). garantindo a integridade dos logs gerados. O Modem Virtual implementa filtros que atenuam qualquer tentativa de *malware* cometer fraudes, e assim, implementa a funcionalidade de contenção (tópico 3 na Figura 3.2).

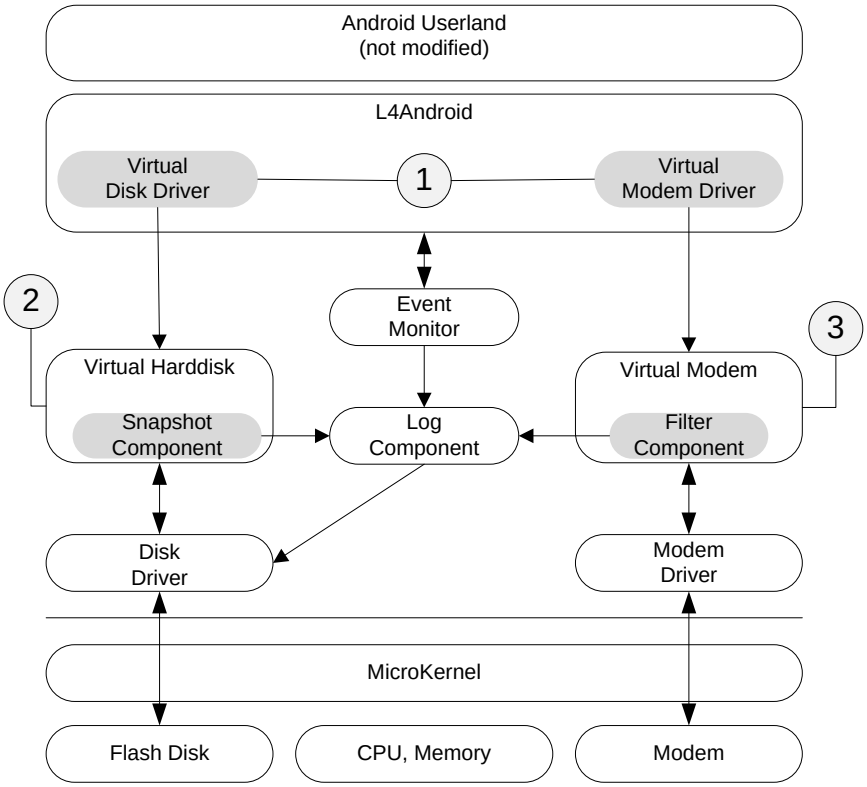

Figura 3.2: Arquitetura do *Honeydroid* [4]

A desvantagem dessa abordagem é que o *HoneyDroid* não se comporta exatamente da mesma forma que o sistema operacional *Android* original. Como o *HoneyDroid* virtualizou os principais dispositivos, gerando uma sobrecarga na CPU do dispositivo móvel, que deve ser levada em conta, pois essa sobrecarga poderia ser detectada pelo invasor e então o seu ataque consequentemente, escaparia do *Honeypot*. A Figura 3.2 mostra a arquitetura do *HoneyDroid*.

Os autores chegaram à conclusão que é viável construir um *Honeypot* para dispositivos móveis e empregaram a virtualização para criar um sistema completo, que possa repetir um ataque e com o nível de arquitetura apresentada; eles afirmam serem os primeiros a criarem *Honeypot* para dispositivo móvel.

HoneyM: a Framework for implementing virtual honeyclients for mobile devices

Os autores Oconnor e Sangster [5] desenvolveram ”HoneyM"’ um *Framework* para *HoneyClient* virtual para dispositivos móveis desenvolvido em linguagem de programação python.

HoneyClient é um tipo de *honeypot* cujo objetivo é descobrir vulnerabilidades ou códigos maliciosos que afetem uma máquina ou um aplicativo cliente, por exemplo um navegador web (Browser). Esse tipo de *Honeypot* normalmente direciona um cliente a realizar um acesso a um servidor e monitora os resultados [39].

O *HoneyM* foi usado para simular um *iPhone* da *Apple*. O *Framework* do *HoneyM* permite a simulação de vários protocolos: *Bluetooth*, infravermelho, Global Positioning System (GPS), 3G, e WiFi, de um dispositivo móvel conforme é apresentado na Figura 3.3.

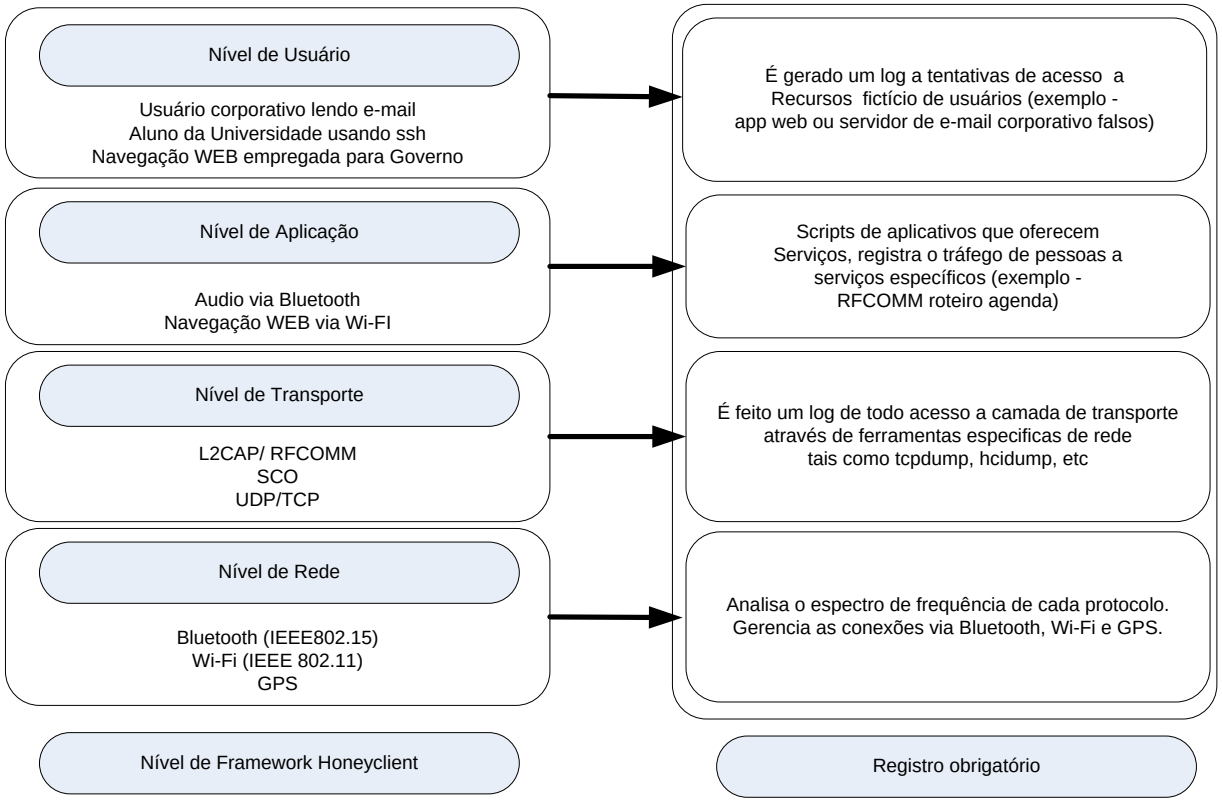

Figura 3.3: Modelo do *Framework* HoneyM [5]

Os autores testaram seus *HoneyClients* em dois ambientes favoráveis a ataques, *DEFCON Hacker Conference* e *the SANSWireless Security Course*. Através destes ambientes se depararam com algumas restrições legais ao usar os *HoneyClients*, uma vez que através deles podem ser gerados log de todo espectro da rede sem fio para

determinar a existência de ataques. Essa técnica pode ser considerada ilegal por ser interpretada como uma escuta ilegal. Concluíram que o *HoneyM* pode ser usado para descobrir novos ataques contra dispositivos móveis, localizados na periferia de uma rede de computadores. A versão inicial do *HoneyM* simula com sucesso dispositivos móveis com os seguintes protocolos de comunicação: *Bluetooth*, infravermelho, GPS, 3G, e WiFi.

3.3 Honeypots

Nesta Seção é apresentada um *Framework* para criar *Honeypots* que simula várias assinaturas de sistemas operacionais em nível de rede. Esse trabalho está fortemente relacionado a proposta desta dissertação. Na Seção 1.2 tem-se descrito como um dos objetivos o desenvolvimento de um *Framework* para *Honeypot* voltado para o sistema operacional *Android*.

Virtual honeypots from botnet tracking to intrusion detection

O autor Provos et al. [6] desenvolve um *Framework* denominado "*Honeyd*"' que simula vários *Honeypots* virtuais. Esses *Honeypots* virtuais simulam assinaturas de sistemas computacionais em nível de rede, conhecidas como *Fingerprint*.

Cada *Honeypot* pode simular a assinatura de um sistema operacional em uma máquina real, de serviços e de protocolos como Transmission Control Protocol (TCP), Internet Control Message Protocol (ICMP) e User Datagram Protocol (UDP). O *Honeyd* suporta Internet Protocol (IP) e responde as requisições feitas na rede para os *Honeypots* virtuais de acordo com os serviços que foram configurados em cada um deles.

A arquitetura do *Honeyd* consiste em vários componentes conforme é mostrado na Figura 3.4, e destacam-se: *Personality Engine, Packet Dispatcher, Routing* e *Services*.

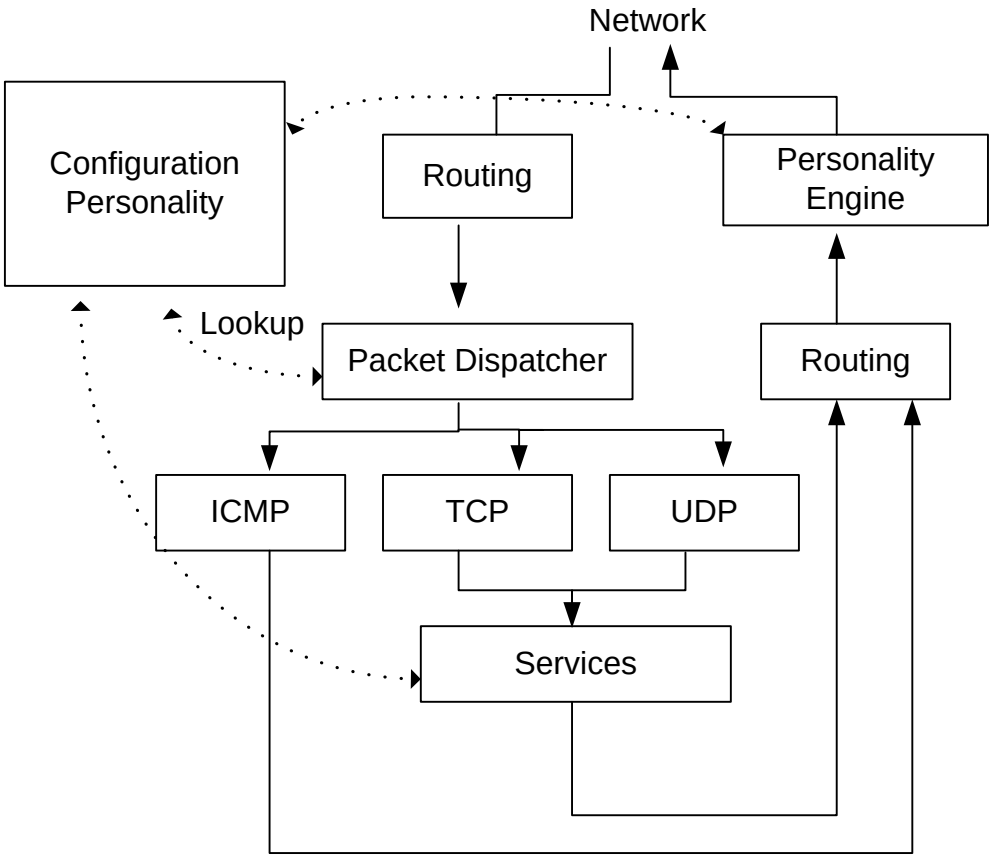

Figura 3.4: Diagrama da Arquitetura *Honeyd* [6]

- *Personality Engine* – é a simulação do comportamento da assinatura de um determinado sistema operacional. Chamada também de personalidade de um *Honeypot* virtual.

- *Packet Dispatcher* – é o distribuidor de pacote central (ICMP, TCP, UDP), os pacotes de entrada são processados pelo distribuidor de pacotes, que compara se é um dos três protocolos usados (ICMP, TCP, UDP), consulta o banco de dados de configuração para procurar um *Honeypot* virtual que corresponda ao endereço IP de destino. Se não existir usa um modelo padrão, e sendo protocolo diferente registra e descarta.

- *Routing* – São as topologias de roteamento usadas para enganar os adversários e ferramentas de mapeamento de rede.

- *Services* – São os serviços que são simulados com o objetivo de enganar os adversários.

Esta Seção teve uma grande influência para que desenvolvêssemos o nosso *Framework* que é a nossa ferramenta para estanciar um *Honeypot* para o sistema operacional *Android*.

3.4 Projeto NIDIA

Arquiteturalmente, o NIDIA é composto por camadas [7] [8]. Cada camada possui atividades a desempenhar, sendo que estas atividades são executadas através do comportamento dos agentes que a compõe [35]. A comunicação entre as camadas para troca de informações das atividades ocorrem através destes.

Na Figura 3.5 é apresentada a arquitetura do NIDIA. Cujas funcionalidades das camadas são descritas a seguir:

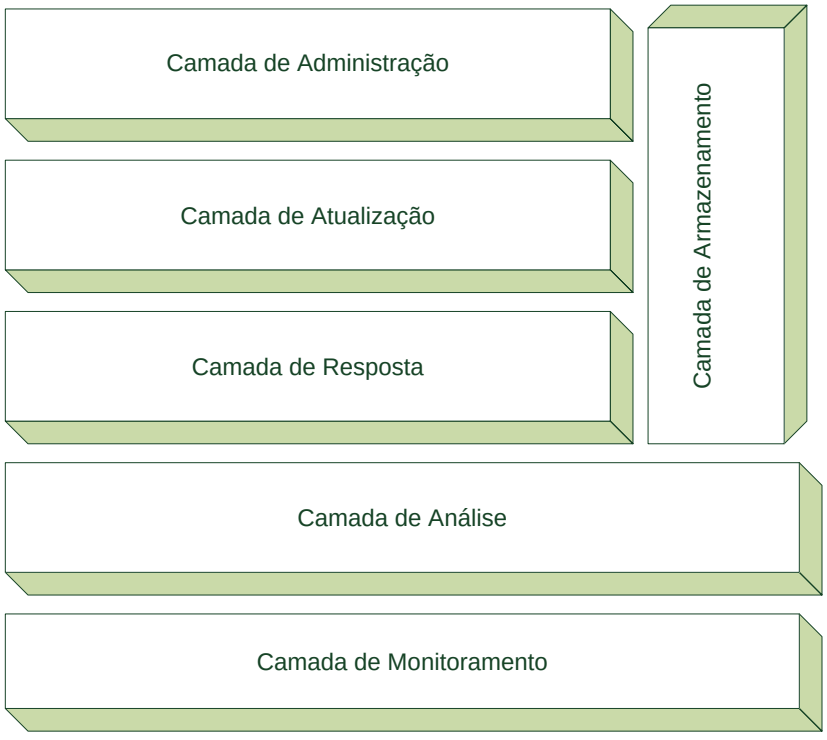

Figura 3.5: Modelo em Camadas do NIDIA [7] [8]

- Camada de Monitoramento – responsável por capturar a ocorrência de eventos ocorridos e repassar informações sobre o mesmo para a camada de análise. Nesta camada estão localizados os agentes SMA (System Monitoring Agent). Agentes sensores que realizam o monitoramento, sendo de dois tipos:

  - Agentes sensores de rede – responsável por capturar os pacotes que estão trafegando na rede;
  - Agentes sensores de host – trabalham coletando informações em tempo real de um host em particular e disponibilizando-as para análise [40].

- Camada de Análise – responsável pela análise de eventos recebidos da camada de monitoramento. Nesta camada, os eventos coletados são formatados de maneira que padrões de ataques possam ser identificados e posteriormente a confirmação de um ataque. Para isso, utilizam-se bases de conhecimento, como a base de dados de padrões de intrusões (*IIDB, Incidents of Intrusion and Forensic Information DataBase*), a base de dados de incidentes de intrusão (*DFDB, Standard of Intruders and Intrusion DataBase*) e a base de estratégias (*STDB, Strategy DataBase*). Nesta Camada se encontram os agentes SEA, Agente de Avaliação de Segurança do Sistema (System security Evaluation Agent) responsável por verificar se o evento corresponde, de fato, a uma tentativa de invasão ou trata-se apenas de um falso positivo. Para auxiliar nessa tarefa, ele faz uso de informações das bases de dados de incidentes de intrusão (DFDB) e estratégias (STBD). Como saída, esse gera um nível de alerta associado à severidade do evento [8].

- Camada de Reação (Respostas) – responsável por tomar contramedidas caso um incidente de segurança seja detectado. Com base no parecer do SEA, esta camada deve tomar uma contramedida de acordo com as bases de dados de estratégia (STDB) e de ações (RADB, Reaction DataBase). Nesta camada localizam-se os agentes SCA (System Controller Agent), responsáveis pelo controle das ações que o sistema deve tomar em caso de uma tentativa de invasão [41];

- Camada de Atualização – responsável pela atualização das bases de informações. As consultas poderão ser feitas diretamente de qualquer camada, porém inserções devem ser feitas somente através desta camada. Ela terá também a responsabilidade de manter a integridade e consistência das informações armazenadas. Nesta camada, localizam-se os agentes SUA (System Updating Agent). Estes são responsáveis pela atualização das bases DFDB, IIDB, RADB e STDB [42].

- Camada de Administração – responsável pela administração e integridade de todos os agentes do sistema. Nesta camada, localizam-se os agentes MCA (Main Controller Agent) [7].

- Camada de Armazenamento – a responsabilidade de manter de forma persistente as informações provenientes das demais camadas é a atribuição desta camada. Nesta não existem agentes, ela foi idealizada para comportar as bases de dados utilizadas pelo NIDIA [40]. Segue descrição das mesmas [42]:

  - *STDB (Strategy DataBase)* é a base de dado responsável por registrar as estratégias adotadas por uma organização qualquer em relação à sua politica de segurança.
  - *RADB (Reaction DataBase)* estão contidas as informações referentes as ações que devem ser tomadas de acordo com a severidade do ataque detectado.
  - *IIDB (Incidents of Intrusion and Forensic Information DataBase)* registra os danos causados por ataques bem sucedidos e tentativas de ataques. Este contém informações que podem ser úteis na identificação de tentativas de ataques provenientes de uma mesma origem ou domínio, ou simplesmente serem usadas em investigações futuras.

O Projeto NIDIA desde a sua primeira implementação no ano de 2001, vem passando por várias extensões, ganhando novas funcionalidades através de novos IDs's que foram inseridos no projeto. Esse processo de evolução se faz necessário devido aos avanços tecnológicos que traz consigo uma gama de novos ataques.

3.5 Comparação Qualitativa

Esta Seção apresenta de forma sintética através da Tabela 3.2, a comparação feita entre os trabalhos apresentados nas Seções 3.2 *Honeypots* para dispositivos móveis e 3.3 *Honeypots*. Para comparar os trabalhos foram levadas em conta as seguintes características:

- *Framework* – Objetivando reuso de software.

- Nível de Aplicação – São aplicativos que são instalados nos sistemas operacionais móveis.
- Nível de Sistema Operacional – Simulam os sistemas operacionais conhecidos como *Fingerprints*.
- Nível de Kernel – Trabalha na camada abaixo do sistema operacional.
- Virtualização de *Hardware* – Trabalha com hardware virtualizado.
- Baixa Interatividade – Provê serviços e assinaturas de sistemas operacionais (*Fingerprints*) falsos.
- Alta Interatividade – Provê sistemas operacionais, aplicações ou serviços reais.
- Sistema Operacional *Android* – Trabalha diretamente com o sistema operacional *Android*.
- Outros Sistemas Operacionais – Não trabalha com sistema operacional *Android*.
- Sistema Operacional Móvel – Trabalha com sistema operacional móvel.

Características	*Honeydroid*	*HoneyM*	*Honeyd*
Framework	Não	Sim	Sim
Nível de Aplicação	Não	Não	Não
Nível de Sistema Operacional	Não	Não	Sim
Nível de Kernel	Sim	Não	Não
Virtualização de *Hardware*	Sim	Não	Não
Baixa Interatividade	Não	Sim	Sim
Alta Interatividade	Sim	Não	Não
Sistema Operacional *Android*	Sim	Não	Não
Outros Sistemas Operacionais	Não	Sim	Sim
Sistema Operacional Móvel	Sim	Sim	Não

Tabela 3.2: Comparação Qualitativa entre os trabalhos das Seções: 3.2

3.6 Considerações Finais

Neste capítulo foram relatados diversos trabalhos que possuem relação direta ou indireta com a problemática abordada neste estudo, objetivando verificar, conhecer e analisar as pesquisas que estão sendo feitas nas áreas de sistema operacional para dispositivos móveis e de *Honeypots* para dispositivos móveis.

Após conhecer os trabalhos aqui apresentados e fazer uma comparação entre os mesmos, optou-se por fazer um *Framework* para *Honeypot* virtual, em nível de aplicativo para o sistema operacional *Android*.

4 *HoneypotLabsac*: Um *Framework* de *Honeypot* Virtual para o *Android*

Neste capítulo será abordado o mecanismo gerado pela proposta inicial, o *HoneypotLabsac*, um *Framework* para desenvolvimento de *Honeypot* virtual para dispositivos móveis que usam o sistema operacional *Android*. explicaremos sua aplicabilidade, a arquitetura e utilização do *Framework*; será detalhado como criar um serviço a partir do *Framework*, e também será explicado a aplicabilidade, a arquitetura e utilização do *HoneypotLabsac*.

4.1 Aplicabilidade do *Framework*

Na Seção 2.5 foram elencadas definições e classificações de *Frameworks*. Vimos que eles podem ser classificados quanto ao contexto de utilização, que podem ser "*Framework* de Infraestrutura de Sistema", "*Framework* de Integração de *Middleware*" e "*Framework* de Aplicação" [37].

A nossa proposta para desenvolvimento do *FrameworkLabsac*, que é um *Framework* de Aplicação, foi inspirada em alguns trabalhos descritos nas Seções 3.2 e 3.3. De forma, diferente dos demais, a nossa proposta esta embasada na simplicidade de coletar dados de um dispositivo móvel conectado a uma rede *wireless*, executando um *Honeypot* virtual em nível de aplicativo.

Inicialmente, o *FrameworkLabsac* gera *Honeypot* virtual para o sistema operacional *Android*, podendo mais tarde ser expandido para outros sistemas operacionais de dispositivos móveis. O objetivo da criação do *Framework* está diretamente relacionado a reuso de *software*, pois o projeto pode ser estendido a qualquer momento, inserindo novos serviços e protocolos a serem emulados. Na próxima Seção será mostrada a arquitetura do *FrameworkLabsac* através dos diagramas de camada e de pacote.

4.2 Arquitetura do *Framework*

Conforme mostrado na Figura 4.1 tem-se a arquitetura do *FrameworkLabsac*. Uma arquitetura simples que trabalha sobre o sistema operacional *Android*. O *FrameworkLabsac* possui a finalidade de fornecer serviços a serem executados no sistema operacional *Android*. Ao se trabalhar com serviços nesta plataforma obtêm-se algumas vantagens que se destacam, são elas – o serviço é executado em segundo plano (*Background*), geralmente vinculado a algum processo que deve ser executado por tempo indeterminado; o serviço não precisa de interface gráfica [21].

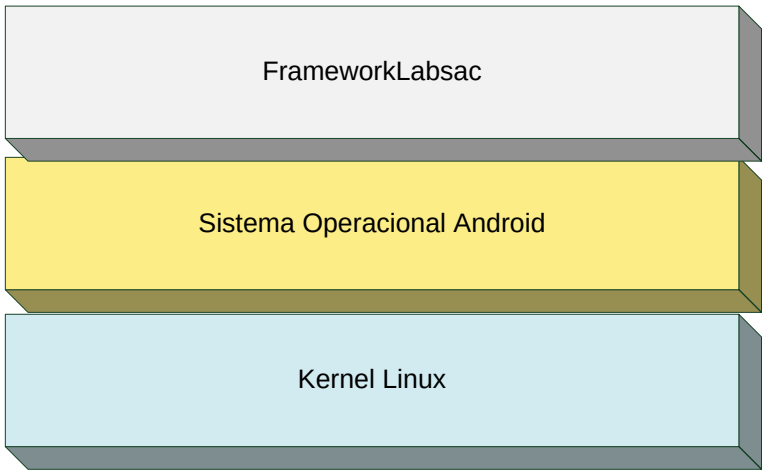

Figura 4.1: Arquitetura do *FrameworkLabsac*

Na Figura 4.2 apresentamos o diagrama de pacotes do *FrameworkLabsac*. Cada pacote tem suas funcionalidades que serão descritas a seguir.

O pacote *core* – responsável pela criação da interface com o usuário e na comunicação entre a interface gráfica e o funcionamento dos serviços. O pacote *core* é composto por 3 classes: *HoneypotActivity, HoneypotApplication* e *HoneypotStarter*. Esse conjunto de classes tem a finalidade de inicializar os componentes de tela; a imagem de cada serviço utilizado; e as portas de comunicação dos serviços; verificar se o serviço esta sendo executado; como também por inicializar ou encerrar um serviço; e faz a verificação de conectividade com a rede.

Já o pacote *serviceBasis* – responsável pela conexão, realização das notificações geradas, e tem implementado 5 classes: *GeneralService, HoneypotProtocol, ProcessAdminRequests, ProtocolProcessing* e *ProtocolResponse*. Esse conjunto de classes tem o objetivo de criar a interface de comunicação, direcionar e verificar trocas de

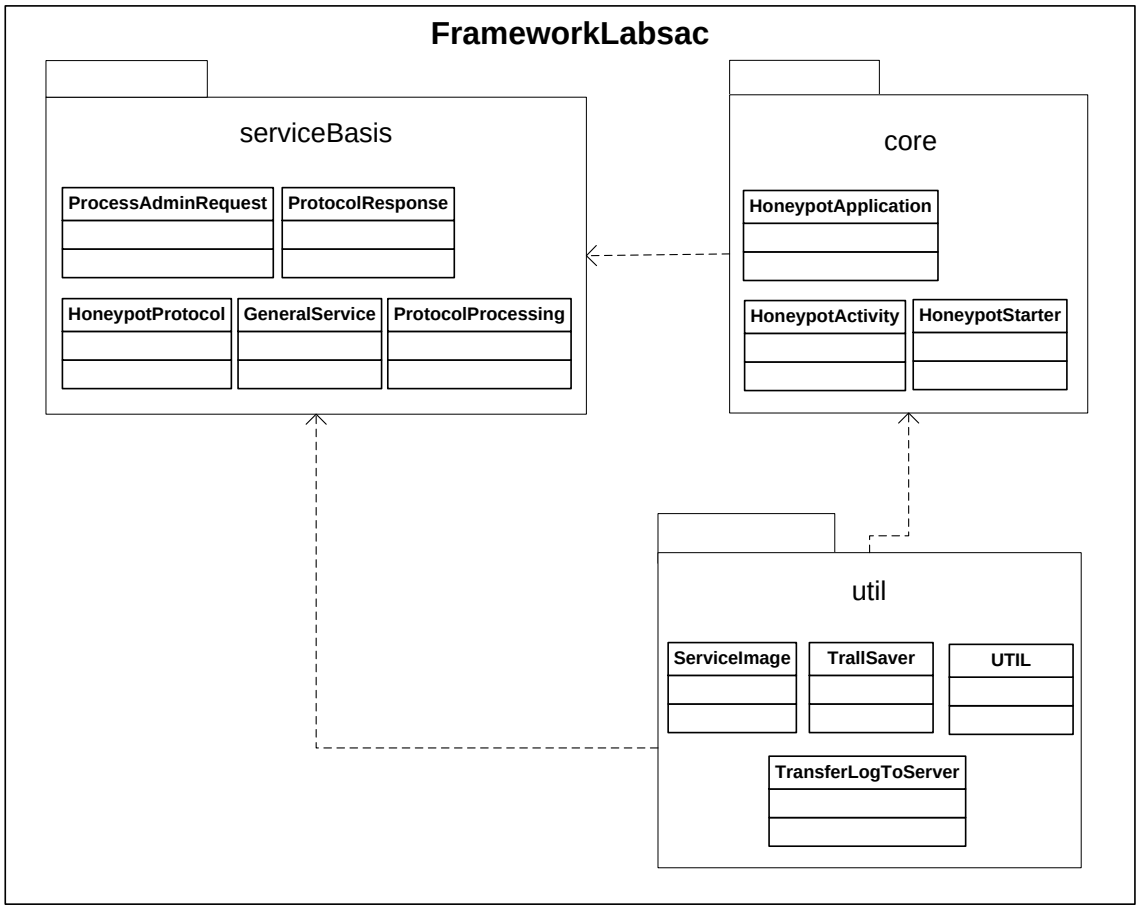

Figura 4.2: Diagrama de Pacote do *FrameworkLabsac*

mensagens e o controle de tráfego, bem como responder as requisições feitas pelos clientes (atacantes) conectados.

O pacote *util* – com suas 4 classes implementadas: *ServiceImage, TrailSaver, TransferLogToServer* e *UTIL*, é responsável por criar uma estrutura para passar informações de configuração dos serviços a serem utilizados. Dá suporte ao salvamento das atividades do aplicativo em um arquivo localizado dentro da memória do dispositivo móvel, transfere os arquivos de log gerados para um servidor externo de log, e por último, mas não menos importante, também gera os arquivos de depuração.

4.3 Utilização do *Framework*

A fim de utilizar a ideia do *FrameworkLabsac* para a criação de *Honeypots* virtuais aplicáveis aos dispositivos móveis, que usam como sistema operacional a plataforma *Android*, o *Framework* foi desenvolvido usando a linguagem de programação Java, em um ambiente de desenvolvimento integrado a *IDE (Integrated Development Environment) Eclipse* [43] e o *SDK Android*, que pode ser executado nas IDE's *Eclipse, Intellije e Netbeans* [2].

O *SDK Android* é um *software* utilizado no desenvolvimento de aplicações no *Android*, é composto por um Emulador do SDK Android (QEMU) na simulação de um aparelho celular, ferramentas utilitárias e uma *API* completa para a linguagem de programação Java, com todas as classes necessárias à desenvolver aplicações para o sistema operacional *Android* [21]. O *SDK Android* sendo executado na *IDE Eclipse* é a forma mais utilizada pelos desenvolvedores [21]. Já existe um *plug-in* customizado, chamado *ADT (Android Development Tool)* inicializando diretamente da *IDE Eclipse* o emulador do *Android* [20].

Um desenvolvedor que queira utilizar o *FrameworkLabsac*, na criação de um novo serviço deve conhecer as classes que o compõem. Como pode-se observar na Figura 4.3 ilustrando o diagrama de classes do *FrameworkLabsac*, é uma tarefa fácil estender a partir do mesmo. A seguir será descrito cada classe.

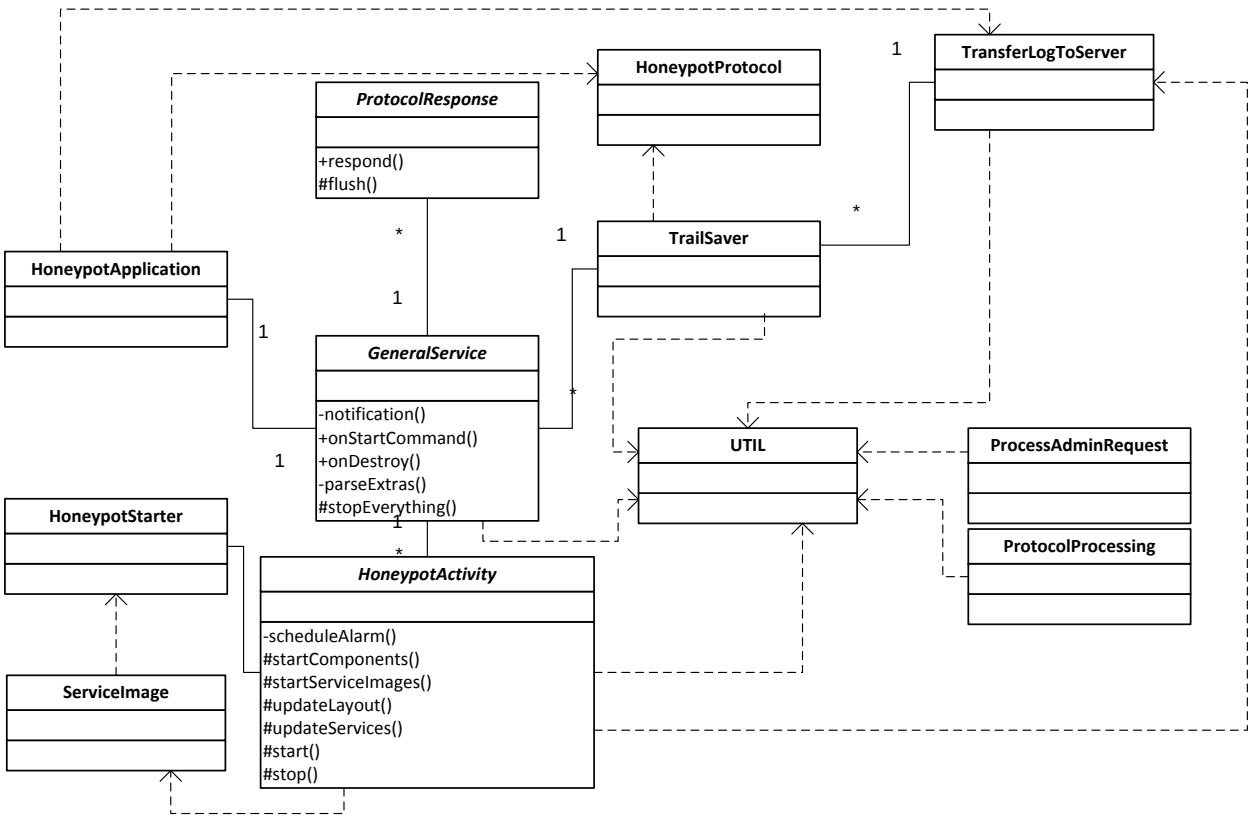

Figura 4.3: Diagrama de Classes do *FrameworkLabsac*

- *HoneypotActivity* – classe principal do pacote *core*, usada para criar a interface do aplicativo, a interface de configuração e tem o controle sobre o o mesmo. Essa classe tem vários métodos abstratos, essenciais para o funcionamento dos serviços no sistema operacional *Android*. Os métodos são:

  - *scheduleAlarm* () – responsável por configurar o tempo de transmissão do arquivo de log do dispositivo móvel para o servidor de log;
  - *startComponents* () – tem a responsabilidade de inicializar os componentes de tela da aplicação;
  - *startServiceImages* () – cada serviço utilizado tem sua imagem inicializada por esse método;
  - *updateLayout* () – atualiza o layout redesenhando as opções de tela;
  - *updateServices* () – atualiza e verifica se o serviço iniciado está sendo executado;
  - *start* () – inicia o serviço;

  - *stop* () – finaliza o serviço.

- *HoneypotApplication* – essa classe faz parte do pacote *core*, responsável em inicializar componentes ao abrir a aplicação. Também programa a periodicidade do tempo para os envios dos logs para um servidor;

- *HoneypotStarter* – também compõe o pacote core, dá suporte à classe *HoneypotAcitivity*, verificando a conectividade na rede, inicializando um serviço com sua porta de comunicação associada e usando o protocolo correspondente. Para a classe *HoneypotStarter* realizar essas tarefas ela pede suporte as classes *MSGTYPE*, *ServiceImage* e *UTIL* do pacote *util*;

- *GeneralService* – pertence ao pacote *serviceBasis*, é uma classe abstrata, sendo também um serviço do sistema operacional *Android* que usa *socket* para conexão. Dar a base para a construção dos serviços dos protocolos. Seus métodos são:

  - *notification* () – responsável por realizar a notificação, por exemplo, se determinado serviço está sendo executado;
  - *onStartCommand* () – chamado pelo sistema toda vez que o serviço é iniciado;
  - *onDestroy* () – tem a responsabilidade de encerrar o serviço uma vez esse método chamado;
  - *parseExtras* () – analisa o ”Extras” (uma coleção contendo chaves e respectivos valores), recebido através de uma *Intent*; retornando verdadeiro após analise de todo ”Extras” com dados da porta de comunicação e serviço a ser executado;
  - *stopEverything* () – tem a finalidade de parar todos os serviços executados, fechando o servidor *socket*, cancelando inclusive as notificações;

- *HoneypotProtocol* – associa o tipo de protocolo à resposta que deve ser dada ao atacante, é responsável por criar o arquivo de log, recuperar o IP do dispositivo e o IP do atacante para ser inserido no log;

- *ProcessAdminRequests* – tem a responsabilidade de verificar se determinado serviço está sendo executado;

- *ProtocolProcessing* – responsável pelo processamento de cada cliente (atacante) que se conecta. Inicia e Termina uma conexão, e a cada conexão feita com o cliente (atacante) é criada uma nova instância;
- *ProtocolResponse* – dá suporte a criação de um protocolo, através de seus métodos abstratos. Ela responde as solicitações dos clientes (atacantes) chamando o método correspondente ao protocolo que está sendo usado;
  - *respond* () – esse método é chamado para responder às solicitações do cliente. Chama o método correspondente, em conformidade com a sua implementação na classe filha;
  - *flush* () – esse tem a responsabilidade de enviar a resposta para o cliente.
- *ServiceImage* – tem a responsabilidade de gerar a estrutura do serviço criado, implementa a serialização dos dados, e é usada por outras classes como por exemplo; a classe *HoneypotActivity*;
- *TrailSaver* – gera os arquivos de log, salvando na memória do celular;
- *TransferLogToServer* – transfere os arquivos de log gerado para um servidor;
- *UTIL* – escreve a mensagem dos logs de depuração.

4.4 Criando um Serviço a partir do *FrameworkLabsac*

Nesta Seção será explicado com detalhes como criar um serviço a partir do *FrameworkLabsac*:

- inicialmente instala-se o (.jar) do *FrameworkLabsac* no *eclipse*, e cria-se um projeto;
- em seguida dentro do projeto cria-se um pacote chamado protocolo;
- dentro deste pacote, cria-se uma classe com o nome sugestivo para o serviço ou protocolo a ser usado, (exemplo: protocolo.Telnet);
- estende-se a classe *ProtocolResponse* do *FrameworkLabsac* para obter as respostas referente a troca de informação no serviço. Ao estender a classe *ProtocolResponse* deve-se implementar o método *"respond"*, que é responsável por enviar a resposta

ao usuário (atacante). Esse método recebe o que foi solicitado, portanto dentro desse método deve-se fazer o filtro de possíveis comandos como, por exemplo, no caso do *Telnet* do *Android*, esse serviço oferece telas e subtelas que devem ser criadas para dar uma maior interatividade com o atacante.

- para fazer o retorno da resposta, utiliza-se o método *"flush"*, que está definido como "final", esse método recebe uma *string*, que contém a resposta em questão.

- para o protocolo ficar escutando nas portas de comunicação, ou seja, sendo executado em *background*, é necessário criar serviços para cada um. Para isso existe a classe *GeneralService*. Deve-se implementar somente o método *onStartCommand*. Dentro dele será configurado somente a notificação.

- para criar a interface do aplicativo e as interfaces de configuração e controle, a classe *HoneypotActivity* deve ser estendida. Para utilizá-la será necessário usar variáveis do tipo booleanas para cada serviço proposto, que irão monitorar se o serviço está sendo executado ou não. Variáveis do tipo inteiro para guardar os valores das portas de comunicação usadas e também variáveis do tipo *ServiceImage* (abstraem toda a informação relevante para o serviço se executado). Seja qual for o método utilizado para inicializar ou finalizar um serviço (um botão, um *checkbox*) deve ser inicializado através do método *onCreate*. O método abstrato *startServiceImages* deve ser implementado fazendo a inicialização das variáveis do tipo *ServiceImage*. Esse método é chamado toda vez que o aplicativo é aberto, através do método *onResume*.

- O método abstrato *startComponents* serve para inicializar qualquer componente visual da aplicação, além de inicializar as variáveis do tipo inteiro que contêm os valores das portas de cada serviço de protocolo.

- O método *updateLayout* serve para atualização dos componentes, sendo chamado toda vez que o aplicativo é aberto, mas pode ser utilizado quando for necessário também, dependendo de como está estruturado o aplicativo. Pode-se usar as variáveis do tipo booleano que servem para controlar se os serviços estão executando. Antes de qualquer outra linha de código, deve-se ser chamado o método *updateLayout* da classe pai, cuja finalidade é de atualizar o estado de cada serviço.

- O método abstrato *updateServices* serve para atualizar as variáveis do tipo booleano, que servem para controlar o estado do serviço. Utiliza-se o método estático *checkPort* da classe *HoneypotStarter* par checar a porta de comunicação. Esse método é sempre chamado pelo método *updateLayout* da classe "pai".

- Para inicialização ou finalização dos serviços são utilizados os métodos "start e stop", que são métodos do tipo "final" que vêm na classe "pai".

4.5 Aplicabilidade do *HoneypotLabsac*

Na Seção 2.3, foram descritas algumas funcionalidades sobre *Honeypots* como, ferramentas de estudos de segurança, *Honeypots* de baixa, média e alta interatividade, e *Honeypots* reais e virtuais. Aqui destacam-se mais funcionalidades, os *Honeypots* que funcionam como monitores de portas de comunicação. Estes *Honeypots* ficam escutando as portas de comunicação que são alvos de atacantes e geram um arquivo de log [29].

Ainda pode-se classificar *Honeypot* por sua função. Um *Honeypot* como sistema falso, além de monitorar uma porta, engana os atacantes ao interagir com eles, dando respostas as requisições feitas ao serviço que está sendo emulado, como acontece de fato ao se usar um serviço que está sendo executado em um sistema real [10].

Conforme visto na Seção 2.3 o valor primário de um *Honeypot* de baixa interação é coletar especificamente tentativas de conexão e sondagens não autorizadas, além de alertar sobre ataques automáticos (*worms*), dentre outras pragas. Esses tipos de *Honeypots* basicamente fornecem as seguintes informações através de seu arquivo de log, conforme descrito pelos autores Provos et al. [6]:

- Data e horário dos ataques;

- Endereço IP e porta de origem do ataque;

- Endereço IP e porta de destino do ataque;

- Atividades do atacante (no caso de uma interação com o serviço emulado).

O *HoneypotLabsac* é um *Honeypot* de baixa interatividade e do tipo sistema falso. O *HoneypotLabsac* emula os seguintes serviços do sistema operacional *Android*: *Telnet* e *http*, gerando um arquivo de *log* dessas conexões e dos envios e recebimentos de SMS transmitidos pela conexão emulada do *Telnet*, e transmitidos por outro dispositivo móvel usando a operadora de telefonia móvel.

O arquivo de log gerado pelo *HoneypotLabsac* fornece as seguintes informações:

- Data e horário da conexão feita ao dispositivo móvel pela rede *wireless*;
- endereço IP e porta de origem (atacante);
- endereço IP e porta de destino (atacado); e
- os comandos digitados.

Na Figura 4.4 é mostrado um exemplo do arquivo de log gerado pelo *HoneypotLabsac*.

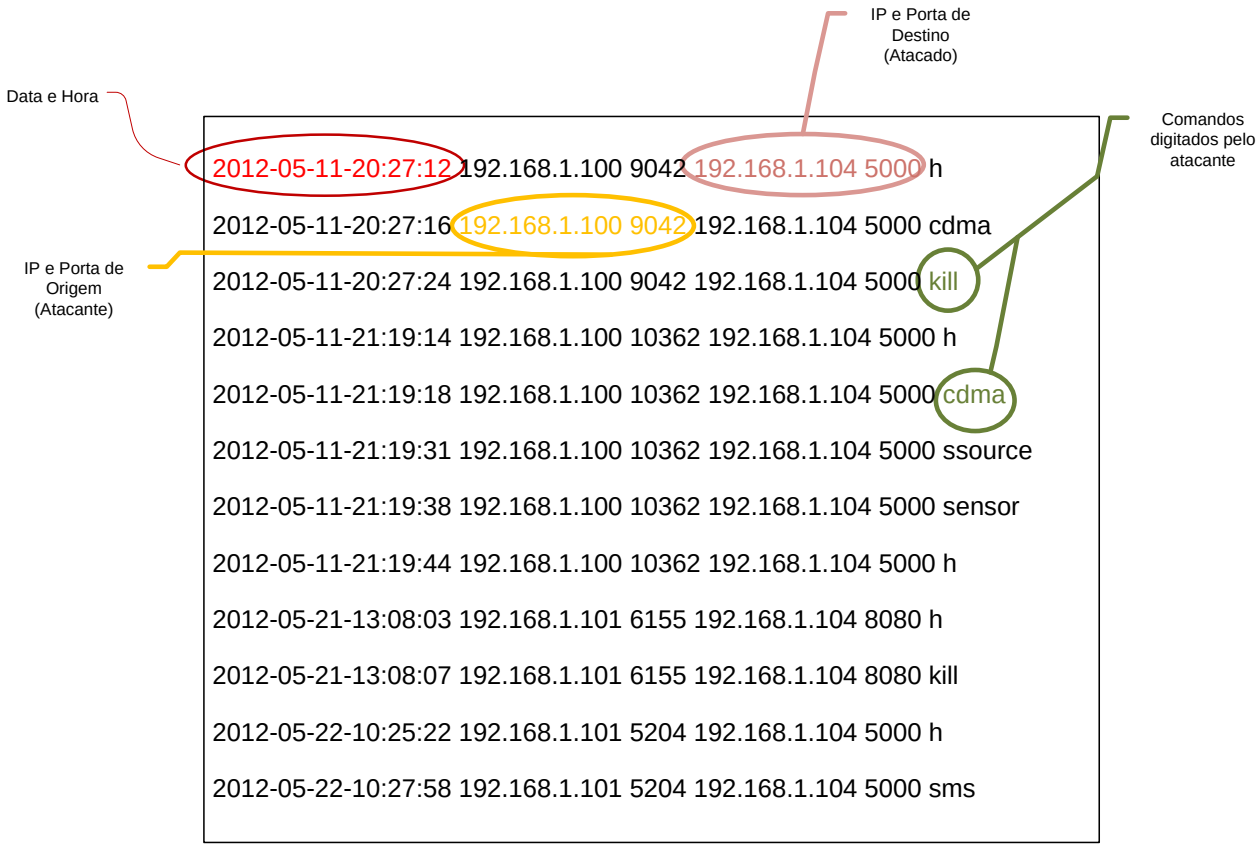

Figura 4.4: Log gerado pelo *HoneypotLabsac*

O arquivo de log gerado pelo *HoneypotLabsac* segue o padrão descrito em Provos et al. [6] usado para o nível de serviços, já que o nosso *Honeypot* emula serviços. O objetivo desta padronização é a reutilização do arquivo por outros trabalhos que fazem parte do Laboratório de Sistemas e Arquiteturas Computacionais (LABSAC) ou não.

4.6 Arquitetura do *HoneypotLabsac*

Nessa Seção será apresentada a arquitetura do *HoneypotLabsac*. Como pode-se observar é uma arquitetura simples, segue a mesma estrutura da arquitetura do *FrameworkLabsac*, que foi apresentada com detalhes na Seção 4.2. A Figura 4.5 apresenta através de um diagrama de camada a arquitetura do *HoneypotLabsac*.

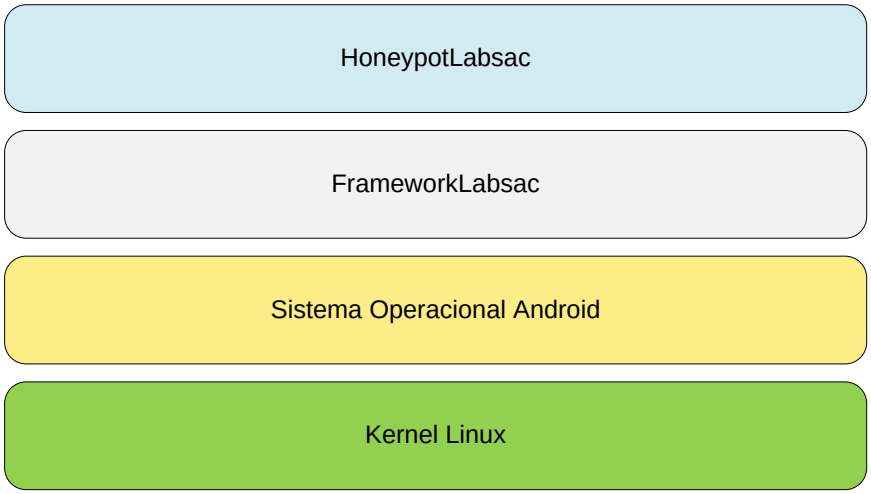

Figura 4.5: Diagrama em Camada *HoneypotLabsac*

Ao detalhar a arquitetura do *HoneypotLabsac* pode-se observar que ela consiste em alguns componentes como é mostrado na Figura 4.6, e destacam-se os *LOGComponent*, Serviços e SMS.

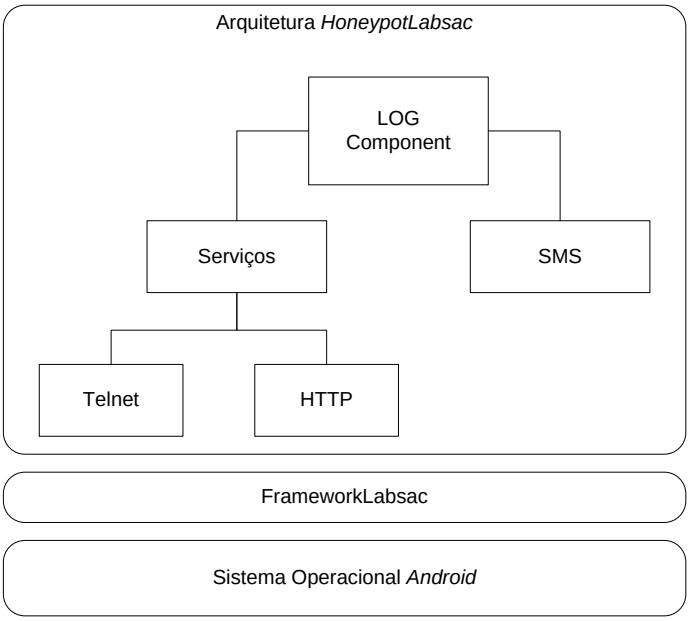

Figura 4.6: Diagrama da Arquitetura *HoneypotLabsac*

- *LOGComponent* – esse componente é responsável por gerar e armazenar toda informação gerada através das conexões dos serviços emulados *Telnet* e *HTTP*, além dos SMS recebidos através do serviço emulado *Telnet* e através da conexão com a operadora de celular. Assim é gerado os arquivos de log de cada serviço emulado pelo *HoneypotLabsac*.
- Serviço *Telnet* – tem o objetivo de emular o serviço *Telnet* do sistema operacional *Android*. Ao emular o serviço dar-se a sensação ao atacante que ele está acessando o serviço de *Telnet* real do sistema operacional *Android*.
- Serviço *HTTP* – tem a finalidade de emular o serviço *HTTP*, retornando uma informação falsa para o atacante.

4.7 Implementação do *HoneypotLabsac*

O *HoneypotLabsac* foi implementado usando a tecnologia Java [44], o ambiente de desenvolvimento integrado a *IDE (Integrated Development Environment) Eclipse* [43] e o *SDK Android* [2] [21].

Na Figura 4.7 é apresentado o diagrama de classe do *HoneypotLabsac* e mostra seus respectivos relacionamentos. As funcionalidades das classes do diagrama apresentado serão detalhadas nesta sub-sessão.

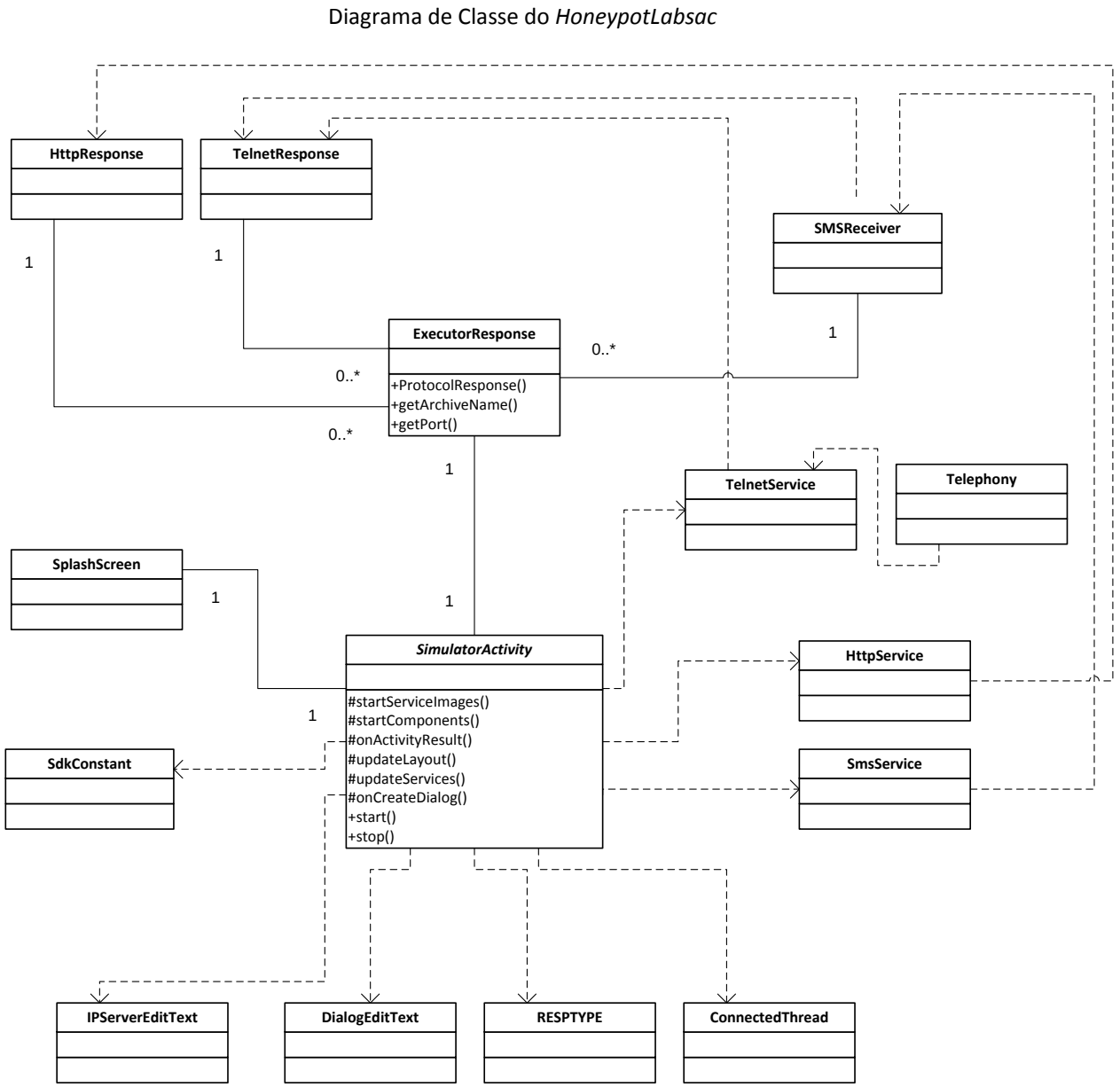

Figura 4.7: Diagrama de Classe do HoneypotLabsac

Na sequência serão explicadas as funcionalidades de cada classe contida na arquitetura do *HoneypotLabsac*. Para criarmos o serviço *Telnet*, inicialmente foi criado o pacote *br.labsac.honeypotLabsac.protocol.telnet*, que contém 3 classes: *Sdkconstant, Telephony* e a *TelnetResponse*.

- A classe SdkConstant – é uma notação Java (utilizada para definir metadados). Indica um valor constante de campo que deve ser exportado para ser utilizado na ferramenta *SDK Android*.

- A classe *Telephony* – é a classe responsável por criar mensagens SMS através do uso do *Telnet* do sistema operacional *Android*.

- A classe *TelnetResponse* é responsável em emular o *Telnet* do sistema operacional *Android* e dentro de suas funcionalidades estão contidos o envio e recebimento de SMS, e também pode-se consultar o nível de bateria do *Smartphone*.

Para o serviço *HTTP* tem o pacote *br.labsac.honeypotLabsac.protocol.http* que contém a classe *HttpResponse*. Ela é responsável por dar as respostas às requisições do cliente (atacante).

O pacote *br.labsac.honeypotLabsac.protocol* contém a classe *ExecutorResponse* que tem por finalidade definir os valores das portas de comunicações onde vão ser executados os falsos serviços.

O pacote *br.labsac.honeypotLabsac* contém 3 classes: *SimulatorActivity, SMSReceiver* e *SplashScreen*.

- A classe *SimulatorActivity* – tem a função de iniciar ou parar o aplicativo *HoneypotLabsac*.

- A classe *SMSReceiver* – é responsável de receber o SMS enviado pelo serviço *Telnet* e gerar o log do SMS recebido.

- A classe *SplashScreen* – é a tela de abertura do *HoneypotLabsac*.

O *pacote br.labsac.honeypotLabsac.services* tem a finalidade de gerenciar os serviços de uma forma geral e possui 3 classes:*HttpService, SmsService* e *TelnetService*. Essas classes correspondem aos serviços que estão sendo emulados pelo *HoneypotLabsac*.

- A classe *HttpService* – tem a função de iniciar o serviço *HTTP*.

- A classe *SmsService* – tem a função de iniciar o serviço para receber SMS das operadoras de celular e da conexão efetuada pelo serviço de *Telnet*.

- A classe *TelnetService* – tem a função de iniciar o serviço *Telnet*.

O pacote *br.labsac.honeypotLabsac.util* contém a classe *RESPTYPE* que tem o controle pelo tipo de resposta.

O pacote br.labsac.honeypotLabsac.view contém classes que auxiliam na interface de usuário, são elas *DialogEditText* e *IPServerEditText*. São *views* personalizadas.

4.8 Caso de Uso do *HoneypotLabsac*

Na Figura 4.8 é mostrado um caso de uso de capturar dados. O *HoneypotLabsac* estando ativo (sendo executado em *Background*) qualquer interação que aconteça com ele através dos serviços emulados *Telnet* e *HTTP*, e o envio ou recebimento de SMS através da conexão *Telnet* ou pela operadora de celular são gerados arquivos de logs. Esses arquivos de logs gerados são, primeiramente, armazenados na memória interna do dispositivo móvel e após um tempo determinado, esse arquivo é enviado para a base de dados do NIDIA.

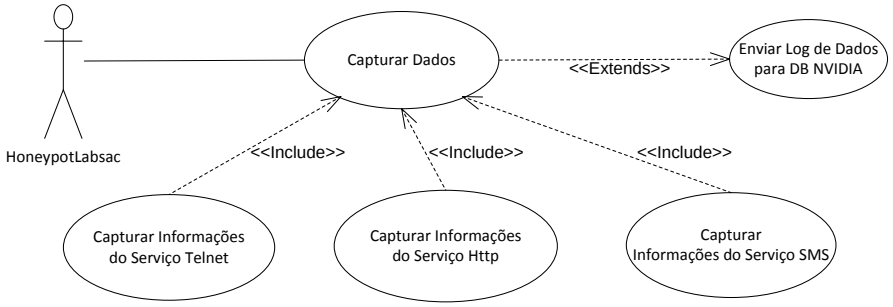

Figura 4.8: Caso de Uso Capturar Dados

Na Figura 4.9 é mostrado com detalhes através do diagrama de sequência como acontece a interação para a captura de dados do *HoneypotLabsac*.

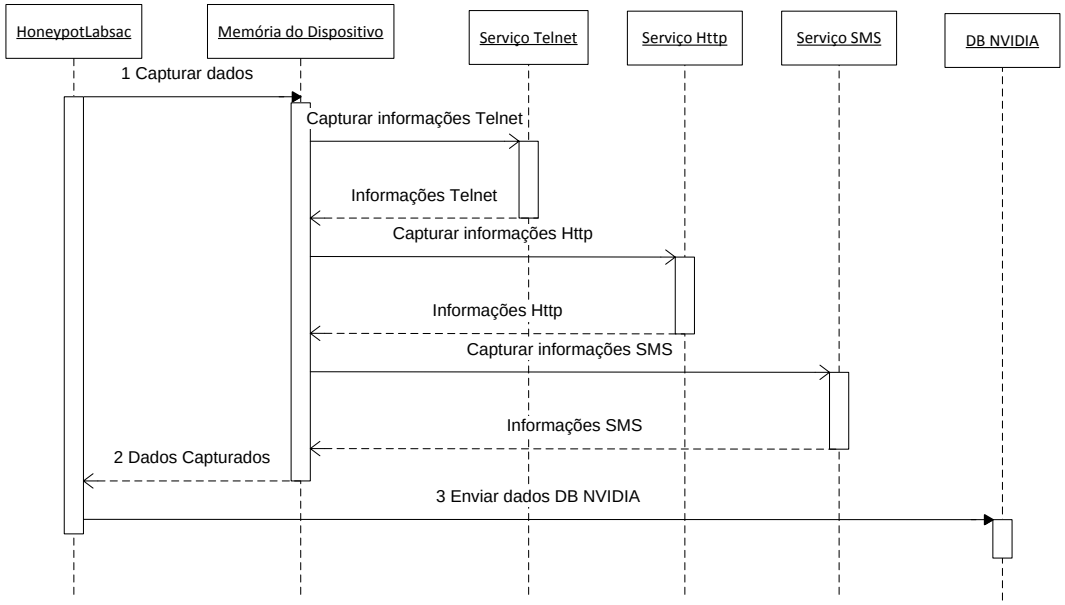

Figura 4.9: Diagrama de Sequência de Capturar Dados

Na Figura 4.10 é apresentado o diagrama de pacote do *HoneypotLabsac* e mostra a integração entre o *FrameworkLabsac* e o *HoneypotLabsac* através dos pacotes e suas classes.

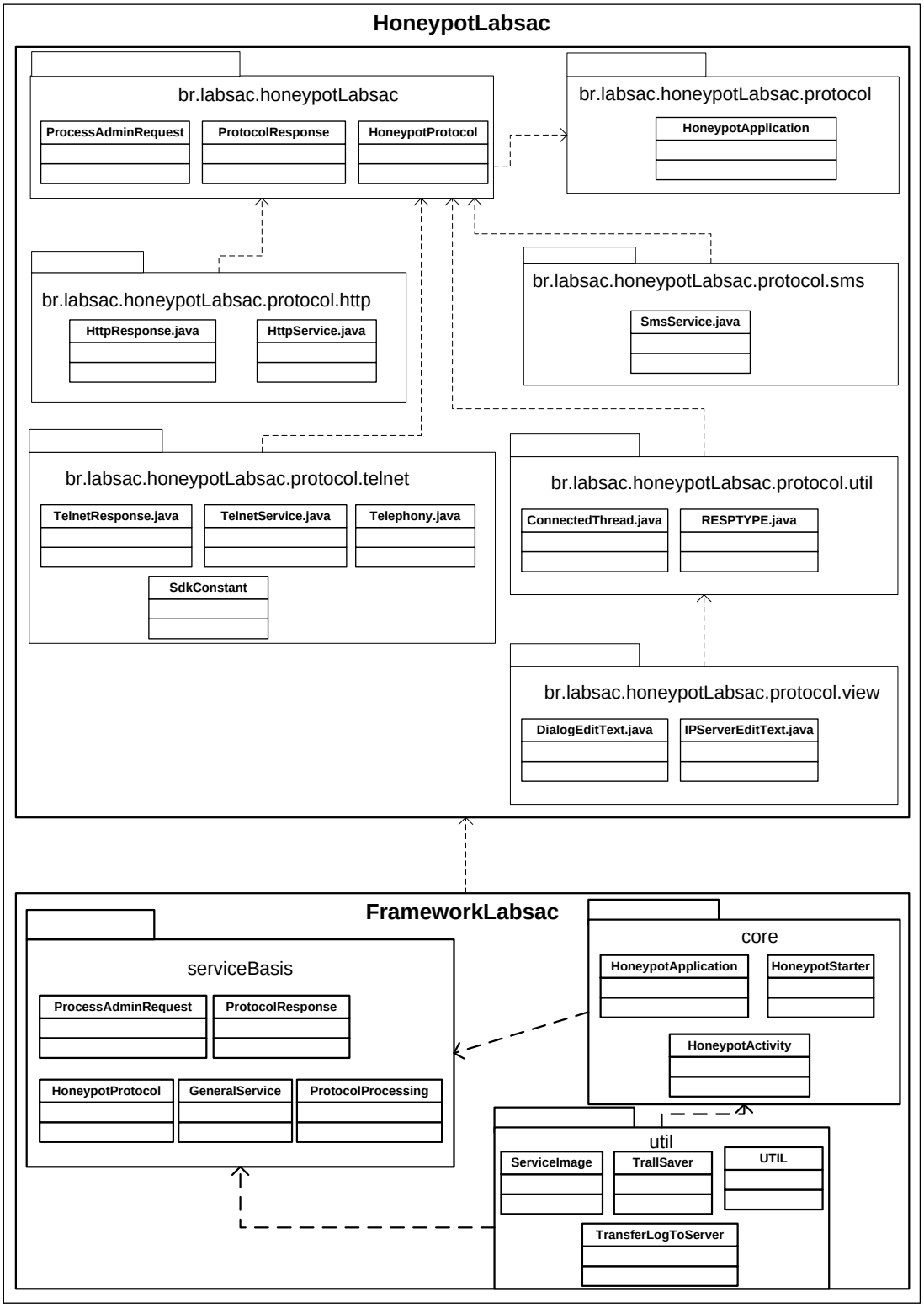

Figura 4.10: Diagrama de Pacote do HoneypotLabsac

4.9 Usando o *HoneypotLabsac*

O *HoneypotLabsac* é uma ferramenta especifica para o sistema operacional *Android*, muito fácil de instalar e configurar. As configurações são feitas através das telas do aplicativo, onde se pode escolher qual serviço emular e em que porta de comunicação esses serviços vão ser emulados; pode-se ainda configurar o número IP do servidor de log e a porta de comunicação entre o servidor de log e o dispositivo móvel. O *HoneypotLabsac* é uma ferramenta para administradores de redes que querem descobrir algum invasor através de um dispositivo móvel conectado em sua rede.

Na Figura 4.11 é mostrada a tela principal do *HoneypotLabsac*, apresentando os serviços disponíveis que podem ser emulados e os *checkbox* correspondente para marcá-los.

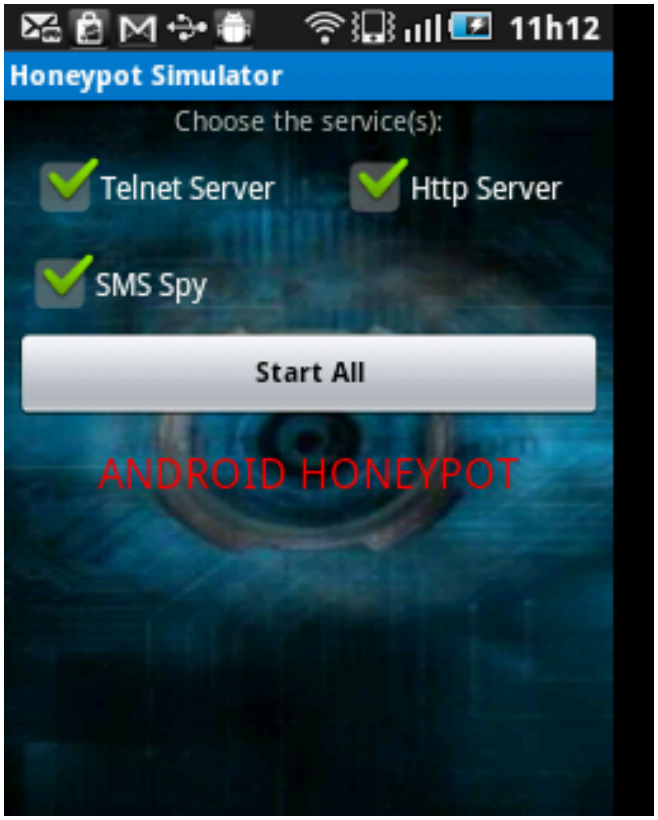

Figura 4.11: Tela principal do *HoneypotLabsac*

Na Figura 4.12 é mostrada a tela na qual ativa-se os serviços, pressionando o botão "*Start Marked*"', após esses serviços terem sido marcados na tela principal.

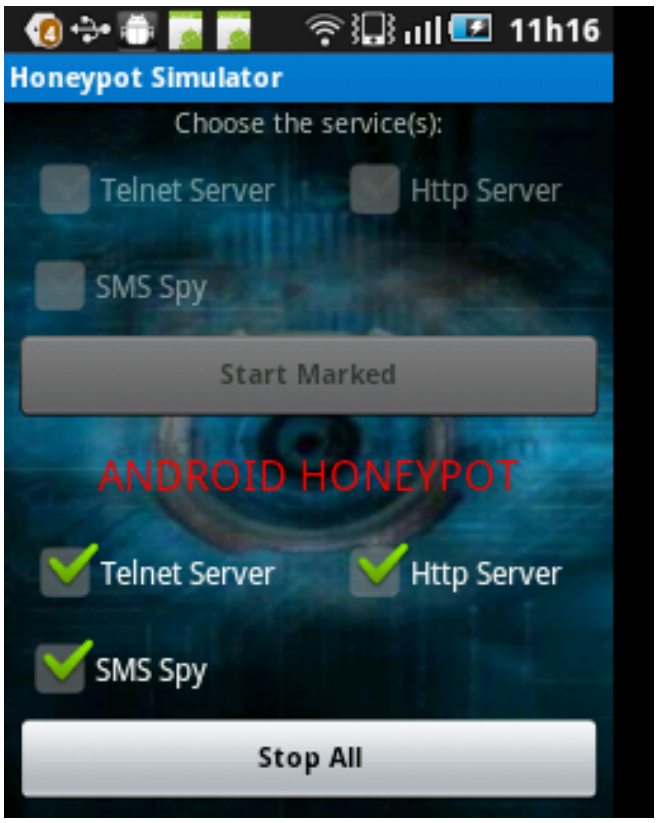

Figura 4.12: Tela de inicialização dos serviços do *HoneypotLabsac*

Na Figura 4.13, nesta tela são feitas as configurações das portas de comunicação para os serviços de *HTTP* e *Telnet*, também é configurado o número IP e número da porta para o servidor de log. A transferência do arquivo de log do dispositivo móvel para o servidor de log só ocorrerá após essa configuração.

Figura 4.13: Tela de configuração das portas de comunicação dos serviços de *HTTP* e *Telnet* e do IP para o servidor de log do *HoneypotLabsac*

4.10 Considerações Finais

Neste capítulo foi apresentado o mecanismo definido na proposta inicial, que contém o *FrameworkLabsac* e o *HoneypotLabsac* um *Honeypot* virtual a nível de aplicativo para funcionar no sistema operacional *Android*, com o objetivo de compreender os ataques que são efetuados aos dispositivos móveis.

Na Seção 3.3 é apresentado o *Honeyd*, um *Framework* para criar *Honeypots* virtuais que simulam assinaturas de sistemas computacionais, conhecidas como *Fingerprint*. O desenvolvimento de um *Honeypot* para o sistema operacional *Android* preenche esta lacuna, já que até o presente momento não tem um *Fingerprint* para o mesmo.

5 Testes e Resultados

Cenário Principal

Neste cenário genérico composto de uma rede interna, servidor de banco de dados do NIDIA, Access Point (AP), *Firewall*, um Computadores (Desktop) (PCs) conectado através de uma rede *Ethernet*, vários *Smarthphones* conectados através do AP usando rede *wireless*, todos tendo acesso a rede externa (Internet). É um cenário padrão e controlado (monitorado) pois estão sendo usado IP's internos ao LABSAC. Nos *Smartphones* e no *Tablet* foram instalado o *HoneypotLabsac* e no PCs foi instalado o *software* virtualBox para virtualizar o sistema operacional *Android* x86 onde também foi instalado o *HoneypotLabsac*.

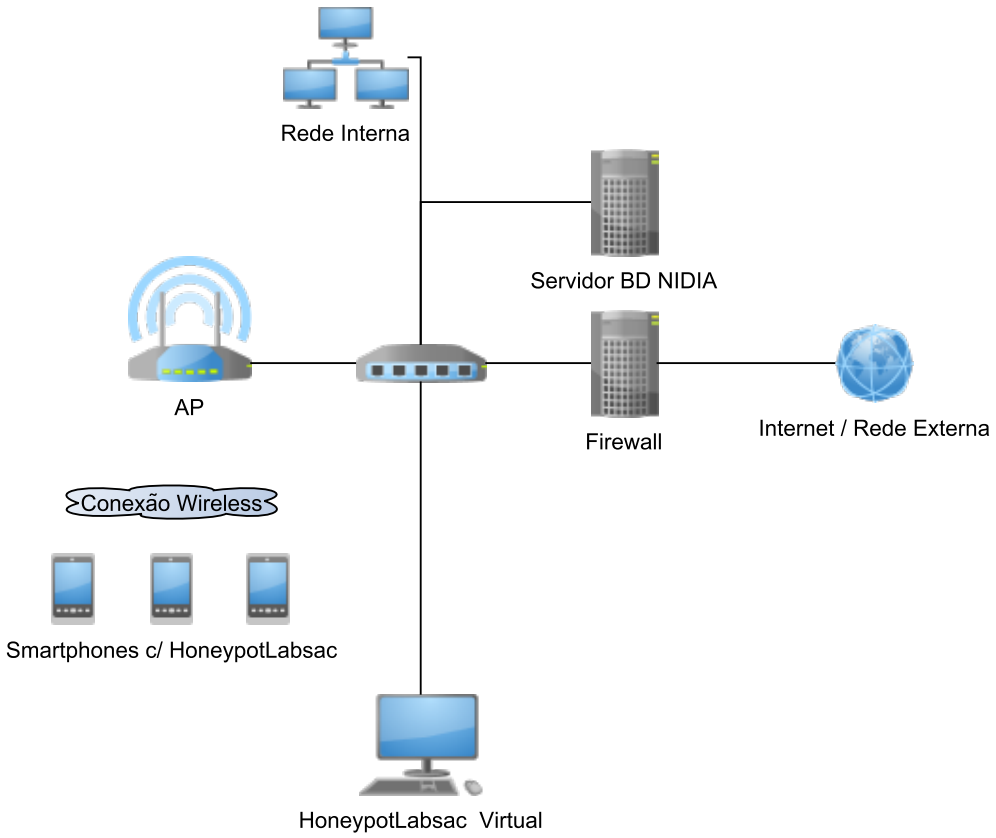

Figura 5.1: Cenário Proposto

Primeiro Teste *HoneypotLabsac* Instalado em um *Smartphone* Real

A Figura 5.2 mostra o cenário onde o *HoneypotLabsac* foi instalado em um *Smartphone* real de marca Samsung, com a seguinte configuração:

- Marca – Samsung;
- Modelo – Galaxy 5;
- Número do Modelo – GT-I5500B;
- Versão de firmware – 2.2.2;
- Versão kernel – 2.6.32.9-perf-se.infra@set-46#1;
- Número de compilação – FROYO.VJJP9;
- Processador – 600Mhz

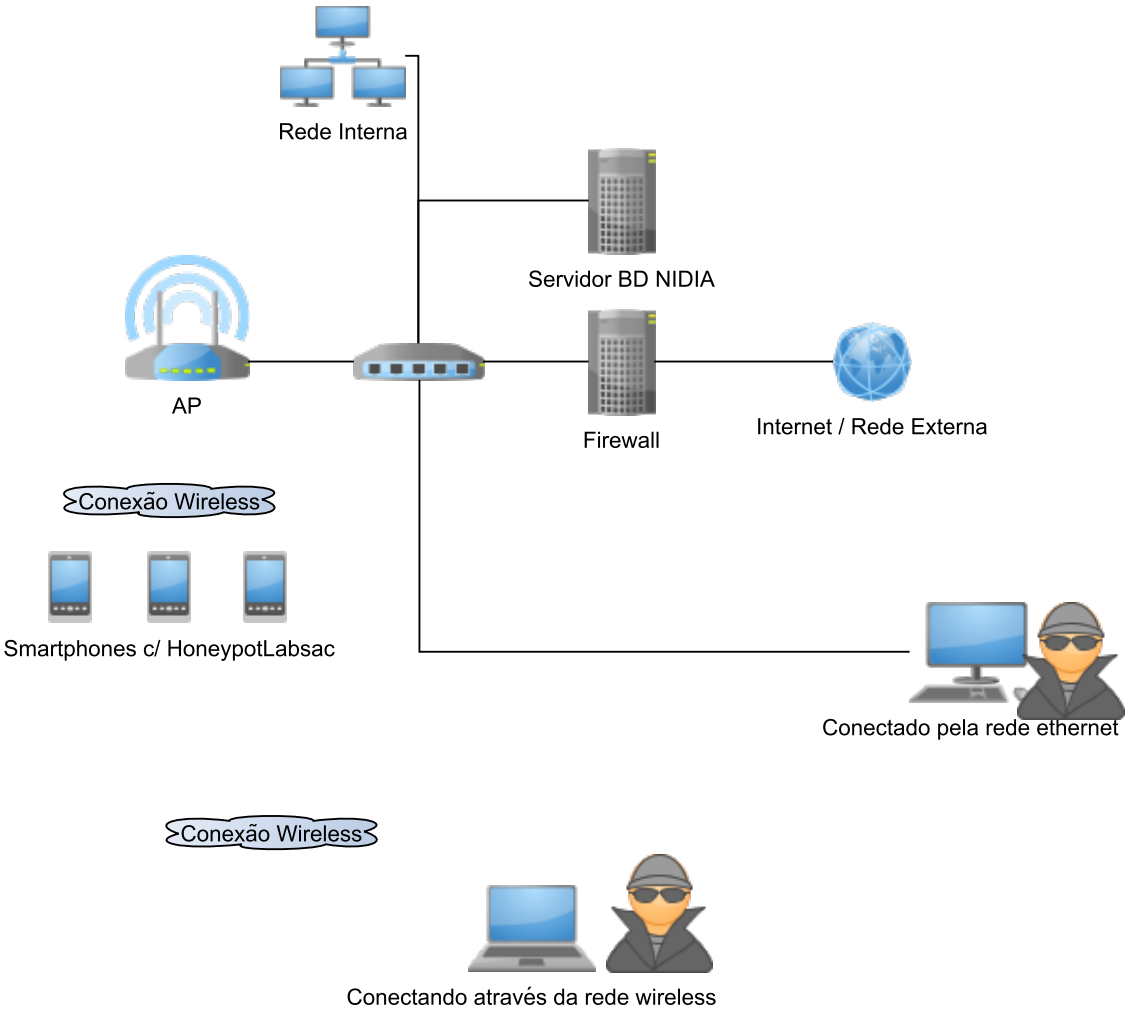

Figura 5.2: Cenário de Teste com Smartphone

Os testes feitos usando o cenário da Figura 5.2 foram realizados em ambiente controlado, este cenário é típico de quem usa um *Smartphone*. Esse tipo de usuário sempre que possível está conectado a Internet através de uma rede *wireless*, já que se torna mais viável financeiramente, pois não tem custo adicional (essas redes *wireless* na maioria das vezes são do trabalho, de sua residência ou de ambientes de convivência), ao contrário de quando se usa a conexão 3G. O *HoneypotLabsac* foi instalado no dispositivo e ativado todos os serviços disponíveis (*Telnet*, *HTTP*, *SMS*). Desta forma, o aplicativo está apto a ser invadido e gerar arquivo de log. Nas Figuras 5.3, 5.4, 5.5 é apresentado os arquivos de log gerado de cada serviço emulado, bem como dos SMS recebidos.

Analisando o log da Figura 5.3 vimos que o atacante com o IP 192.168.1.100 conecta-se ao *HoneypotLabsac*, que está usando o IP 192.168.1.104 e a porta 5000, após a conexão percorre o serviço *Telnet* achando que está interagindo com o serviço de *Telnet* do sistema operacional *Android*.

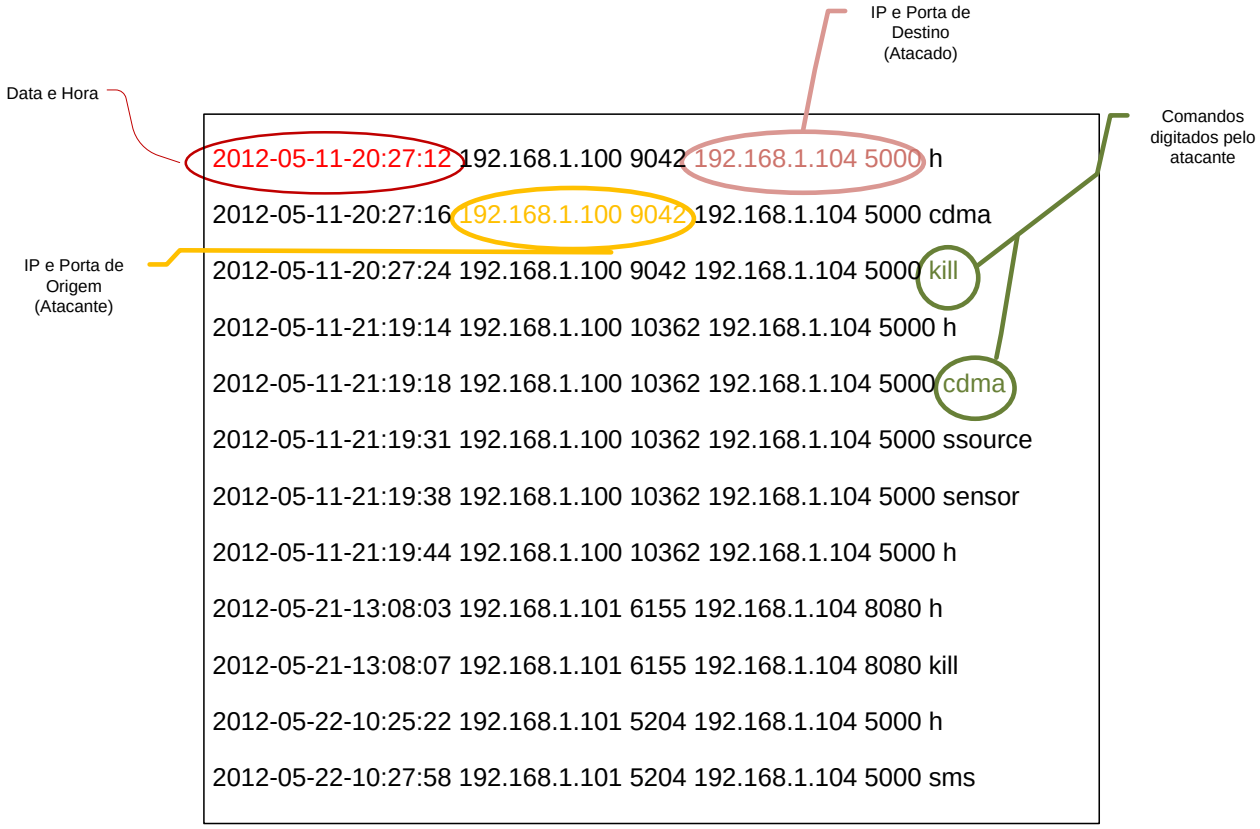

Figura 5.3: Log do Serviço Telnet

O log da Figura 5.4 mostra o atacante com o IP 192.168.1.100 interagindo com o *HoneypotLabsac*, através do IP 192.168.1.104 na porta 8080. Continuando analisando o log da Figura 5.4, mostra-se uma outra conexão de um atacante com o IP 192.168.1.101 interagindo com o *HoneypotLabsac*, através do IP 192.168.1.103 na porta 8080.

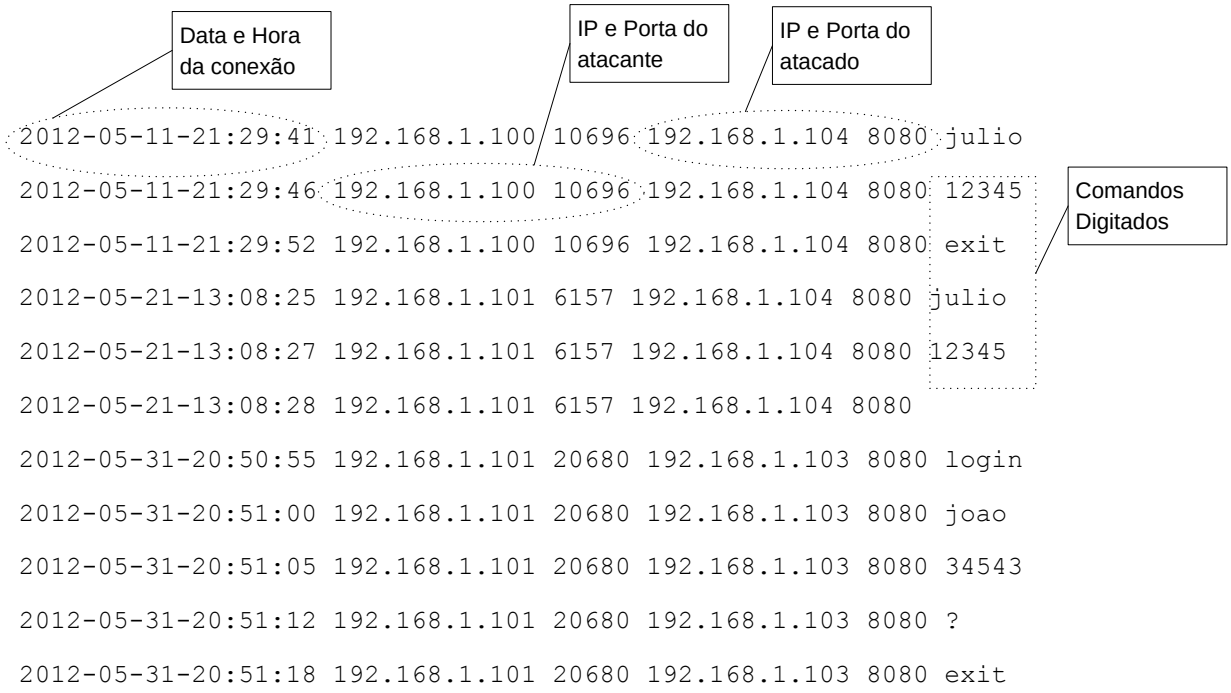

Figura 5.4: Log do Serviço *HTTP*

A Figura 5.5 mostra um log de uma mensagem de texto enviada pela operadora de celular.

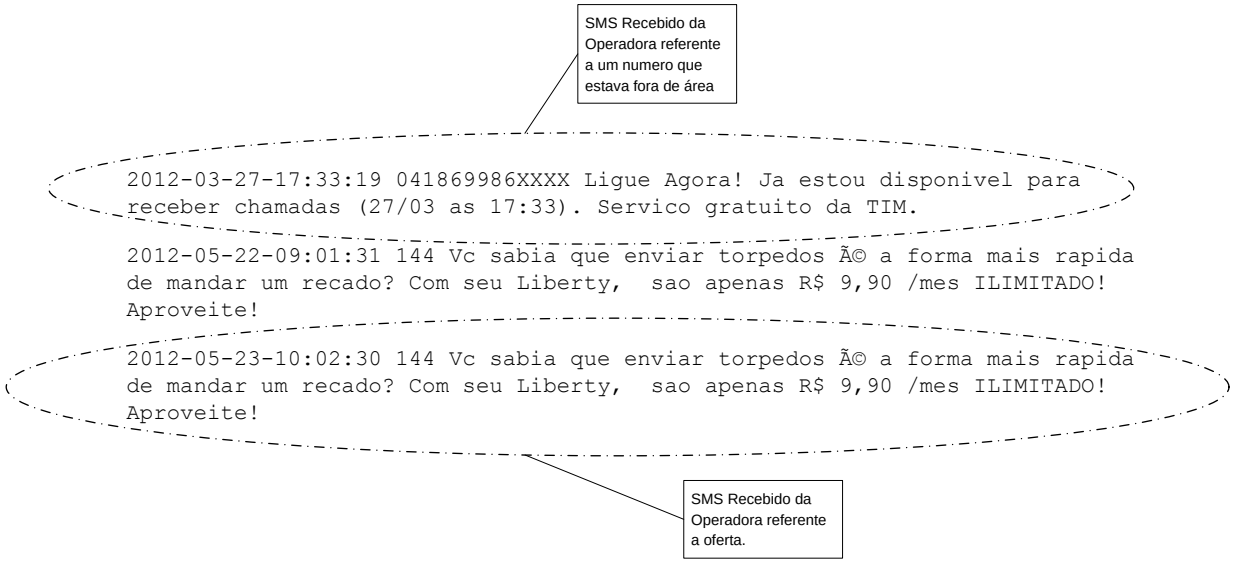

Figura 5.5: Log do SMS recebido pela Operadora de Celular

Segundo Teste *HoneypotLabsac* Instalado em uma máquina virtual usando o sistema operacional *Android* x86

Os testes feitos usando o cenário da Figura 5.6 foram realizados em ambiente controlado, usando um PCs com sistema operacional Windows e um *software* de virtualização, o VirtualBox, criando uma máquina virtual emulando o sistema operacional *Android* arquitetura x86. Neste cenário optou-se em virtualizar o sistema operacional *Android* pela praticidade de configuração.

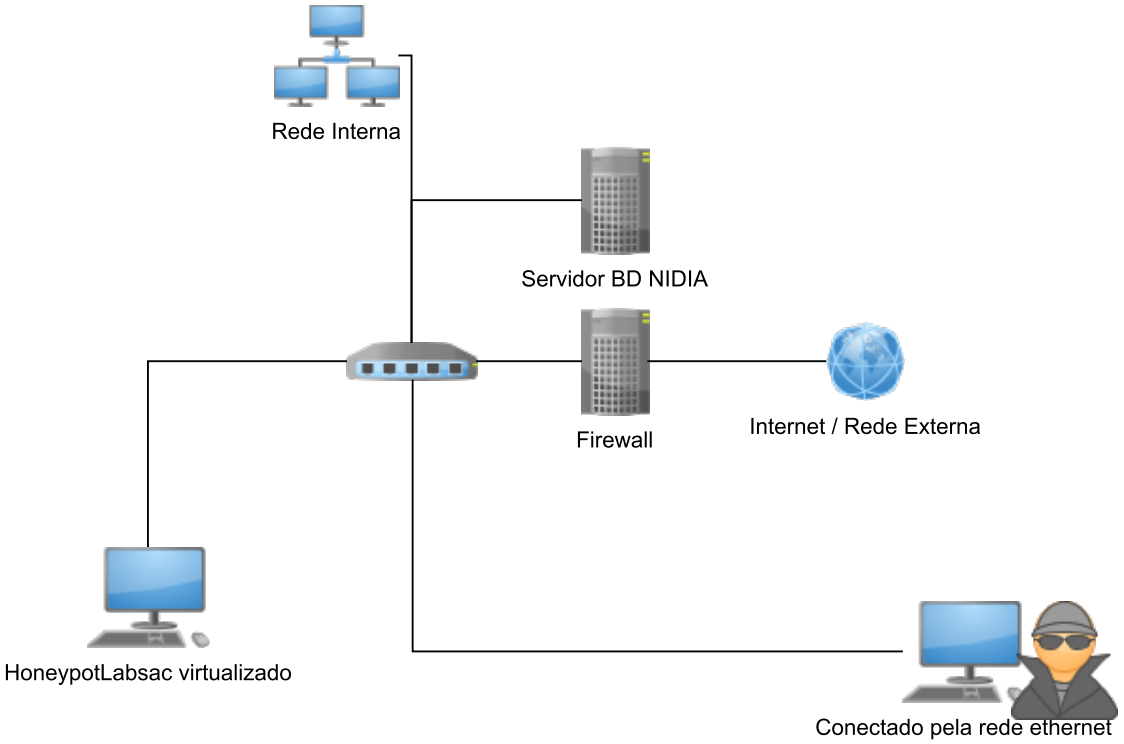

Figura 5.6: Cenário de Teste *HoneypotLabsac* Instalado em uma máquina virtual usando o sistema operacional *Android* x86

Para representar o cenário 5.6 foram usados *hardware* e *software*:

- PCs – de marca DELL, Modelo Inspiron 580, com sistema operacional Windows seven home Premium;
- VirtualBox – *software* de virtualização, versão 4.1.4r74291 da Oracle (VM).
- Android x86 – Sistema Operacional *Android* para ser usado na arquitetura x86 (intel), versão 2.2

Com o objetivo de transformar o *HoneypotLabsac* em uma ferramenta de aprendizado em ataques nos dispositivos móveis, optou-se por trabalhar em nível de aplicativo para o sistema operacional *Android*.

Após a simulação dos testes, coletas de dados através dos arquivos de log e apresentação dos resultados obtidos, apresenta-se agora uma comparação de forma sintética, através da Tabela 5.1, entre o *HoneypotLabsac* e outros trabalhos que foram citados no Capítulo 3, nas Seções 3.2 *Honeypots* para dispositivos móveis e 3.3 *Honeypots*. Para comparar os trabalhos foram levadas em conta as seguintes características:

- *Framework* – Objetivando reuso de software.
- Nível de Aplicação – São aplicativos que são instalados nos sistemas operacionais móveis.
- Nível de Sistema Operacional – Simulam os sistemas operacionais conhecidos como *Fingerprints*.
- Nível de Kernel – Trabalha na camada abaixo do sistema operacional.
- Virtualização de *Hardware* – Trabalha com hardware virtualizado.
- Baixa Interatividade – Provê serviços e assinaturas de sistemas operacionais (*Fingerprints*) falsos.
- Alta Interatividade – Provê sistemas operacionais, aplicações ou serviços reais.
- Sistema Operacional *Android* – Trabalha diretamente com o sistema operacional *Android*.
- Outros Sistemas Operacionais – Não trabalha com sistema operacional *Android*.
- Sistema Operacional Móvel – Trabalha com sistema operacional móvel.

Características	Honeydroid	HoneyM	Honeyd	HoneypotLabsac
Framework	Não	Sim	Sim	Sim
Nível de Aplicação	Não	Não	Não	Sim
Nível de Sistema Operacional	Não	Não	Sim	Não
Nível de Kernel	Sim	Não	Não	Não
Virtualização de *Hardware*	Sim	Não	Não	Não
Baixa Interatividade	Não	Sim	Sim	Sim
Alta Interatividade	Sim	Não	Não	Não
Sistema Operacional *Android*	Sim	Não	Não	Sim
Outros Sistemas Operacionais	Não	Sim	Sim	Não
Sistema Operacional Móvel	Sim	Sim	Não	Sim

Tabela 5.1: Comparação Qualitativa

Após a comparação chega-se a conclusão que o *HoneypotLabsac* é o primeiro *Honeypot* para o sistema operacional *Android* em nível de aplicação. Em conclusão, este trabalho demonstrou como o HoneypotLabsac poderia ser usado para detectar novos ataques contra dispositivos móveis localizado na periferia de uma rede.

5.1 Integração com o NIDIA

Esta Seção apresenta de forma sintética como acontece a integração do *HoneypotLabsac* com o NIDIA.

O funcionamento do sistema NIDIA pode ser demonstrado através da interação entre os agentes que compõem suas camadas. Cada camada dentro do NIDIA possui um objetivo bem definido. Este objetivo é alcançado através da interação e troca de informações entre as camadas. Para um melhor entendimento, o funcionamento do IDS-NIDIA está divido em três partes: funcionamento da captura, funcionamento da análise e funcionamento das contramedidas. Para a integração entre o *HoneypotLabsac* e o NIDIA mostra-se funcionamento da captura.

No funcionamento da captura, os agentes sensores de *host* e rede são os responsáveis por capturar a ocorrência de eventos e fornecer informações sobre os mesmos para o resto do sistema. Os agentes sensores de host fazem leitura dos logs de

segurança de servidores específicos e repassam para o agente *SMA* de *host* para que o mesmo formate os dados.

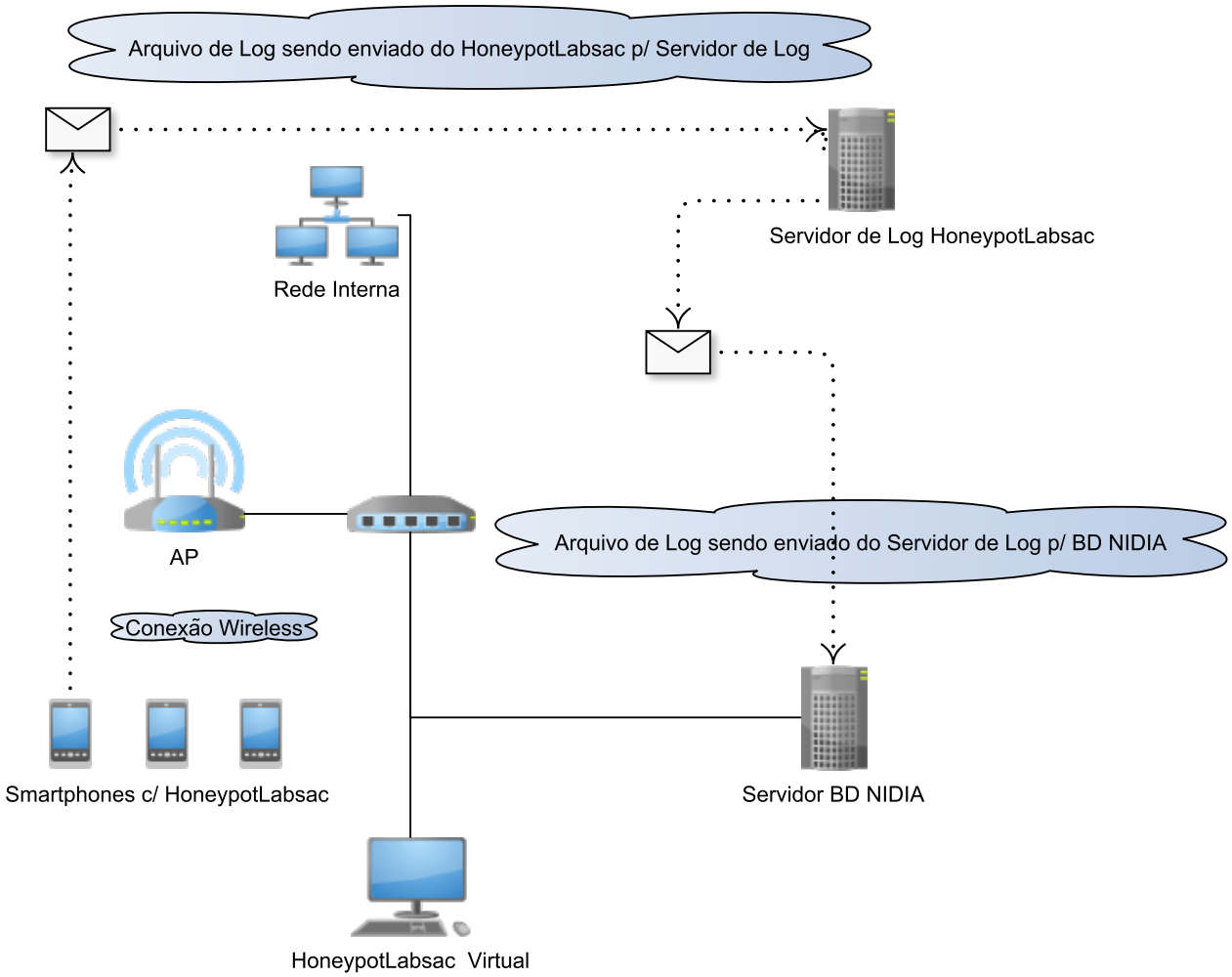

Figura 5.7: Integração entre o *HoneypotLabsac* e o NIDIA

6 Conclusões

Historicamente, os *Honeypots* estão presentes na segurança de redes desde 1988 até os dias atuais, com a finalidade de colher informações dos atacantes, através de monitoramento. Os dispositivos móveis, destacando os *Smartphones*, com suas funcionalidades e facilidades têm conquistados a cada dia mais usuários. Uma dessas facilidades é a conexão a rede wireless e ou 3G, permitindo a seus usuários acesso a Internet para baixar aplicativos, acessar redes sociais. No entanto, essa facilidade pode ser bastante prejudicial, pois através desse acesso pode ser infectado por *malwares*. Diante deste cenário de risco, há uma necessidade em conhecer melhor esses ataques ao dispositivos móveis.

A ideia por trás dessa abordagem é desenvolver um *Framework* para instanciar *Honeypot* em dispositivos móveis a nível de aplicativo no sistema operacional *Android*.

Contribuições

Nesse contexto, as principais contribuições desse trabalho são:

- A proposta de um *Framework* para gerar *Honeypot* para dispositivos móveis que usam sistema operacional *Android*. De forma a ser usado como uma ferramenta em segurança de redes baseados em decepção.
- O *HoneypotLabsac*, um *Honeypot* em nível de aplicação para o sistema operacional *Android*.
- O *HoneypotLabsac* é o primeiro *Honeypot* virtual a nível de aplicação para o sistema operacional *Android*.

Lições Aprendidas

No decorrer da pesquisa e desenvolvimento do trabalho destacam-se algumas lições, as quais listamos a seguir:

- A teoria de *Honeypots* com mais de 20 anos, ainda é atual e o seu principal objetivo é aprender ao ser atacado, (aprender com o atacante), através deste aprendizado pode-se evitar maiores danos em uma rede de computadores.

- Os Honeypots aliados à redes de computadores não vem substituir o papel do administrador de redes ou do gerente de segurança ou, de maneira geral, do responsável por essas áreas no segmento de Tecnologia da Informação das organizações, vem para contribuir para a segurança da rede de computadores e aos profissionais citados, com suas informações geradas através dos arquivos de log.

Desafios

Na Seção 2.3, do Capítulo 2, na Seção 2.3, percebe-se alguns desafios a serem superados quando se trabalha com *Honeypots*, são eles:

- Monitoramento – o arquivo de log é armazenado em uma pasta do sistema onde o usuário e o atacante não tem permissão para acessá-la;

- Log para auditoria – a integridade dos dados foi garantida com a transferência do arquivo de log gerado pelo dispositivo móvel para o servidor de log, de tempo em tempo, onde esse tempo é programado;

- Contenção – por ser um *Honeypot* de baixa interatividade se torna mais facil evitar que o *Honeypot* sirva de trampolim para ataque na rede.

- Visibilidade – A visibilidade é um grande problema. Não adianta implementar o melhor monitoramento se o *Honeypot* não é atacado. Precisa-se aumentar a visibilidade através de algumas ações, como por exemplo, ter um endereço IP público para o *HoneypotLabsac*.

Limitações

Em nossa proposta identificamos as seguintes limitações:

- O *FrameworkLabsac* foi desenvolvido em lingugem de programação Java e com SDK *Android*, o que limita a instanciar *Honeypots* apenas ao sistema operacional *Android*.

- Da mesma forma, há limitação ao *HoneypotLabsac*.

Trabalhos Futuros

A partir deste trabalho inicial, identificamos algumas possibilidades de trabalhos futuros que poderiam ser desenvolvidos, tais como:

- Neste trabalho o foco foi o sistema operacional *Android*, nada impede que o projeto seja adaptado para ser executado em outros sistemas operacionais móveis.
- Poderiam ser implementados novos serviços a serem emulados nos dispositivos móveis.

Referências Bibliográficas

[1] OSENÃ, M. A. *TrendLabs Annual Security Roundup A Look Back at 2011 Information is Currency*. Dez 2011. 7 p.

[2] ANDROID. *Sistema Operacional Android*. Disponível em: <http://www.android.com>. Acesso em: 01 mar. 2012.

[3] BARRERA, D.; OORSCHOT, V. P. Secure software installation on smartphones. *Security Privacy, IEEE*, v. 9, n. 3, p. 42 –48, may-june 2011. ISSN 1540-7993.

[4] COLLIN MULLINER, S. L.; LANGE, M. Poster: Honeydroid - creating a smartphone honeypot. In: *IEEE Symposium on Privacy and Security, Poster*. Oakland California: IEEE, 2011.

[5] O'CONNOR, T. J.; SANGSTER, B. honeym: a framework for implementing virtual honeyclients for mobile devices. In: *WiSec '10: Proceedings of the third ACM conference on Wireless network security*. New York, NY, USA: ACM, 2010. (WiSec '10), p. 129–138. ISBN 978-1-60558-923-7.

[6] PROVOS, N.; HOLZ, T. *Virtual honeypots: from botnet tracking to intrusion detection*. Four. Boston: Addison-Wesley Professional, 2010. ISBN 9780321336323.

[7] SIQUEIRA, L.; ABDELOUAHAB, Z. A fault tolerance mechanism for network intrusion detection system based on intelligent agents (nidia). In: *Proceedings of the The Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, and the Second International Workshop on Collaborative Computing, Integration, and Assurance (SEUS-WCCIA'06)*. Washington, DC, USA: IEEE Computer Society, 2006. (SEUS-WCCIA '06), p. 49–54. ISBN 0-7695-2560-1.

[8] LIMA, C. F. L. *Agentes Inteligentes para Detecção de Intrusos em Redes de Computadores*. Dissertação (Mestrado) — PPGEE / UFMA, Maio 2002.

[9] COMSCORE. *Comscore*. Disponível em: <http://www.comscore.com/por/Industry_Solutions/Mobile>. Acesso em: 05 mar. 2012.

[10] SPITZNER, L. *Honeypots: Tracking Hackers*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN 0321108957.

[11] OLIVER, E. A survey of platforms for mobile networks research. *SIGMOBILE Mob. Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 12, p. 56–63, February 2009. ISSN 1559-1662.

[12] MICROSOFT. *Sistema Operacional Windows Mobile*. Disponível em: <http://http://www.microsoft.com/brasil/windowsmobile/>. Acesso em: 01 mar. 2012.

[13] DWIVEDI, H.; CLARK, C.; THIEL, D. *Mobile Application Security*. San Francisco: McGraw-Hill, 2010. (IT Pro). ISBN 9780071633567.

[14] SYMBIAN. *Sistema Operacional Symbian*. Disponível em: <http://licensing.symbian.org/>. Acesso em: 01 mar. 2012.

[15] LINUX, M. *Sistema Operacional Mobile Linux*. Disponível em: <https://wiki.linuxfoundation.org/en/Mobile_Linux>. Acesso em: 02 fev. 2012.

[16] RIM. *Sistema Operacional RIM (BlackBerry OS)*. Disponível em: <http://us.blackberry.com/software.html>. Acesso em: 01 mar. 2012.

[17] APPLE. *Sistema Operacional iOS*. Disponível em: <http://www.apple.com/ios/>. Acesso em: 02 mar. 2012.

[18] ENCK, W.; ONGTANG, M.; MCDANIEL, P. On lightweight mobile phone application certification. In: *Proceedings of the 16th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2009. (CCS '09), p. 235–245. ISBN 978-1-60558-894-0.

[19] HOEPERS C.; JESSEN, K. S. e. C. M. H. P. C. *Honeypots e Honeynets: definições e aplicações.* Disponível em: <http://www.cert.br>. Acesso em: 01 mar. 2012.

[20] PEREIRA L. C. O E SILVA, M. L. *Android para desenvolvedores*. Rio de Janeiro: Brasport, 2009.

[21] LECHETA, R. *Google Android: aprenda a criar aplicações para dispositivos móveis com o Android SDK*. Sâo Paulo: Novatec, 2010. ISBN 9788575221860.

[22] OHA. *Open Handset Alliance*. Disponível em: <http://www.openhandsetalliance.com/>. Acesso em: 02 fev. 2012.

[23] MEIER, R. *Professional Android 2 Application Development*. 1st. ed. Birmingham, UK: Wrox Press Ltd., 2010. ISBN 0470565527, 9780470565520.

[24] HOOG, A. *Android Forensics: Investigation, Analysis and Mobile Security for Google Android*. Boston: Elsevier Science, 2011. (Android Forensics: Investigation, Analysis and Mobile Security for Google Android). ISBN 9781597496513.

[25] X86, A. *Sistema Operacional Android Arquitetura x86*. Disponível em: <www.android-x86.org>. Acesso em: 02 fev. 2012.

[26] HARRINGTON, J. L. *A Practical Approach*. San Francisco: U.S.A: Elsevier, 2005.

[27] MARCELO A.; E PITANGA, M. *Honeypots: A arte de iludir hackers*. São Paulo: Brasport, 2003. ISBN 9788574521480.

[28] ASSUNCAO, M. F. A. . *Honeypots e Honeynets: Aprenda a detectar e enganar os invasores*. Rio de Janeiro: Visual Books, 2009. ISBN 9788575022511.

[29] HERRERA, R. P. Um estudo sobre o uso de honeypots. In: . Londrina: Universidade Estadual de Londrina, 2009.

[30] AL-GHARABALLY, N. et al. Wireless honeypots: survey and assessment. In: *Proceedings of the 2009 conference on Information Science, Technology and Applications*. New York, NY, USA: ACM, 2009. (ISTA '09), p. 45–52. ISBN 978-1-60558-478-2.

[31] MURALEEDHARAN, R.; OSADCIW, L. A. An intrusion detection framework for sensor networks using honeypot and swarm intelligence. In: *In Proceedings of the 6th Annual International Conference on Mobile and Ubiquitous Systems: Networking & Services, MobiQuitous.* Toronto, Canada: IEEE, 2009.

[32] VERENDEL, V. et al. An approach to using honeypots in in-vehicle networks. In: *VTC Fall*. Singapore: IEEE, 2008. p. 1–5.

[33] MAIRH, A. et al. Honeypot in network security: a survey. In: *Proceedings of the 2011 International Conference on Communication, Computing & Security*. New York, NY, USA: Proceeding ICCCS '11 Proceedings of the 2011 International Conference

on Communication, Computing & Security, 2011. (ICCCS '11), p. 600–605. ISBN 978-1-4503-0464-1.

[34] GUERRA, P. H. C. et al. Caracterização de encadeamento de conexões para envio de spams. In: *27o Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. Recife, PE: SBRC, 2009.

[35] DUNHAM, K. *Mobile Malware Attacks and Defense*. United States of America: Burlington: Elsevier, 2009.

[36] JOHNSON, R. E. Frameworks = (components + patterns). *Commun. ACM*, ACM, New York, NY, USA, v. 40, n. 10, p. 39–42, out. 1997. ISSN 0001-0782.

[37] ADRIANO, C. *Explorando novos modelos de informação hipermídia em ambientes de aprendizagem.* Dissertação (Mestrado) — DCAIFEE - UNICAMP, Maio 2000.

[38] HALL, S. P.; ANDERSON, E. Operating systems for mobile computing. *J. Comput. Sci. Coll.*, Consortium for Computing Sciences in Colleges, USA, v. 25, n. 2, p. 64–71, dez. 2009. ISSN 1937-4771.

[39] WANG, K. *Honeyclient development project*. Disponível em: <http://www.honeyclient.org/>. Acesso em: 02 fev. 2012.

[40] SILVA, E. C. C. *Gerenciamento e Integração das Bases de Dados de Sistemas de Detecção de Intrusões*. Dissertação (Mestrado) — PPGEE/UFMA, December 2006.

[41] JÚNIOR, F. A. P. *Proposta de Atualização Automática dos Sistemas de Detecção de Intrusão por meio de WEB Services*. Dissertação (Mestrado) — PPGEE/UFMA, December 2005.

[42] SILVA, A. L. da. *Modelo IDS para usuários de Dispositivos Movéis*. Dissertação (Mestrado) — PPGEE/UFMA, June 2008.

[43] ECLIPSE. *Eclipse*. Disponível em: <http://www.eclipse.org.>. Acesso em: 01 mar. 2012.

[44] DEITEL, H. M.; DEITEL, P. J. *Java como programar*. 4. ed. Porto Alegre: Bookman, 2003. Tradução: Edson FurnanKiewicz.