

MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

1º Ten TIAGO OLIVEIRA SALDANHA
ANTÔNIO LUÍS SOMBRA DE MEDEIROS

DESENVOLVIMENTO DE APLICATIVO EM ANDROID PARA
SNIFFER DE REDE

Rio de Janeiro
2016

INSTITUTO MILITAR DE ENGENHARIA

**1º Ten TIAGO OLIVEIRA SALDANHA
ANTÔNIO LUÍS SOMBRA DE MEDEIROS**

**DESENVOLVIMENTO DE APLICATIVO EM ANDROID
PARA SNIFFER DE REDE**

Projeto de Fim de Curso apresentado ao Curso de Graduação em Engenharia de Computação do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Engenheiro de Computação.

Orientador: Maj Anderson Fernandes Pereira dos Santos - D.Sc.

Rio de Janeiro
2016

INSTITUTO MILITAR DE ENGENHARIA
Praça General Tibúrcio, 80 - Praia Vermelha
Rio de Janeiro - RJ CEP 22290-270

Este exemplar é de propriedade do Instituto Militar de Engenharia, que poderá incluí-lo em base de dados, armazenar em computador, microfilmar ou adotar qualquer forma de arquivamento.

É permitida a menção, reprodução parcial ou integral e a transmissão entre bibliotecas deste trabalho, sem modificação de seu texto, em qualquer meio que esteja ou venha a ser fixado, para pesquisa acadêmica, comentários e citações, desde que sem finalidade comercial e que seja feita a referência bibliográfica completa.

Os conceitos expressos neste trabalho são de responsabilidade do(s) autor(es) e do(s) orientador(es).

005.268 Saldanha, Tiago Oliveira
S162d Desenvolvimento de Aplicativo em Android para sniffer de rede / Tiago Oliveira Saldanha, Antônio Luís Sombra de Medeiros, orientados por Anderson Fernandes Pereira dos Santos - Rio de Janeiro: Instituto Militar de Engenharia, 2016.

61p. : il.

Projeto de Fim de Curso (PROFIC) - Instituto Militar de Engenharia, Rio de Janeiro, 2016.

1. Curso de Engenharia de Computação - Projeto de Fim de Curso. 2. Aplicativo Android. 3. Wireless I. Medeiros, Antônio Luís Sombra de. II. Santos, Anderson Fernandes Pereira dos. III. Título. IV. Instituto Militar de Engenharia.

INSTITUTO MILITAR DE ENGENHARIA

**1º Ten TIAGO OLIVEIRA SALDANHA
ANTÔNIO LUIS SOMBRA DE MEDEIROS**

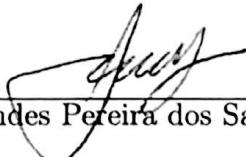
**DESENVOLVIMENTO DE APLICATIVO EM ANDROID
PARA SNIFFER DE REDE**

Projeto de Fim de Curso apresentado ao Curso de Graduação em Engenharia de Computação do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Engenheiro de Computação.

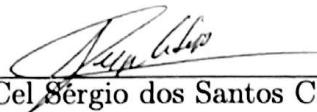
Orientador: Maj Anderson Fernandes Pereira dos Santos - D.Sc.

Aprovado em 29 de Setembro de 2016 pela seguinte Banca Examinadora:

Maj Anderson Fernandes Pereira dos Santos - D.Sc. do IME - Presidente



Ten Cel Sérgio dos Santos Cardoso Silva - M.Sc. do IME



Prof. Ricardo Choren Noya - D.Sc. do IME



Rio de Janeiro
2016

Dedicamos este trabalho a Deus e às nossas famílias, que nos apoiaram em todos os momentos.

Ao meu Tio Tourinho, que veio a falecer esse ano, um senhor que sempre estava de bom humor e que trazia alegria a todos em sua volta.

AGRADECIMENTOS

Agradecemos a todas as pessoas que nos incentivaram, apoiaram e possibilitaram esta oportunidade de ampliar horizontes.

Familiares, mestres e amigos que apoiaram o projeto.

Em especial ao meu Professor Orientador Dr. Anderson Fernandes Pereira dos Santos, por sua disponibilidade e atenção.

“Não é o conhecimento, mas o ato de aprender, não a posse mas o ato de chegar lá, que concede a maior satisfação ”

CARL FRIEDRICH GAUSS

SUMÁRIO

LISTA DE ILUSTRAÇÕES	9
LISTA DE SIGLAS	10
1 INTRODUÇÃO	13
1.1 Motivação	13
1.2 Objetivo	14
1.3 Justificativa	14
1.4 Metodologia	14
1.5 Organizaçāo da Monografia	15
2 FUNDAMENTAÇÃO TEÓRICA	16
2.1 A Plataforma Android	16
2.1.1 A Arquitetura Android	16
2.1.2 Componentes de uma Aplicação	18
2.1.3 Android Manifest.xml	18
2.1.4 Intents e filtros Intent	18
2.1.5 Android Marshmallow	19
2.2 Conceitos de Segurança da Computação	19
2.3 Arquitetura ISO para segurança	21
2.4 Ataques e Vulnerabilidades	21
2.4.1 Ataques Passivos	22
2.4.2 Ataques Ativos	22
2.4.3 Sniffing	22
2.5 WiFi	23
2.5.1 IEEE 802.11 Wireless LAN	23
2.5.2 Conceitos Importantes	24
2.5.2.1 SSID - Service Set Identifier	24
2.5.2.2 BSSID	24
2.5.2.3 Hotspot	24
2.5.2.4 Ponto de Acesso	24
2.5.2.5 Provedor de Serviço de Internet	24

3	MODELAGEM	26
3.1	Diagrama de Funcionamento	26
3.2	Levantamento de Requisitos	27
3.2.1	Escopo de Obtenção do SSID e do BSSID	27
3.2.2	Escopo do Hotspot	27
3.2.3	Escopo do Sniffer	27
3.3	Storyboard	27
4	DESENVOLVIMENTO	29
4.1	Etapa 1: Módulo App WiFi Watcher	29
4.1.1	Descrição do Módulo	30
4.1.2	Justificativa e Importância	30
4.1.3	Implementação	31
4.2	Etapa 2: Módulo App Hotspot Creator	31
4.2.1	Descrição do Módulo	31
4.2.2	Importância e posição no aplicativo final	32
4.2.3	Implementação	32
4.3	Etapa 3: Módulo App WiFi Sniffer	32
4.3.1	Descrição do Módulo	33
4.3.2	Importância e posição no aplicativo final	33
4.3.3	Implementação	34
4.3.3.1	LocalVPN	34
4.3.3.2	Sniffer	35
4.4	Etapa 4: Unificação dos aplicativos	35
4.4.1	Unificação dos aplicativos WiFi Watcher e Hotspot Creator	36
4.4.2	Unificação com o módulo Sniffer final	36
4.5	Dificuldades Gerais Encontradas	36
5	CONCLUSÃO	39
6	REFERÊNCIAS BIBLIOGRÁFICAS	40
7	APÊNDICES	42
7.1	APÊNDICE 1: Código das Classes Principais	43
7.1.1	CLASSE WIFI WATCHER	43
7.1.2	CLASSE HOTSPOT CREATOR	47

7.1.3 CLASSE LOCAL VPN SERVICE	52
--	----

LISTA DE ILUSTRAÇÕES

FIG.2.1	Arquitetura de Camadas Android	17
FIG.2.2	Tríade CID	20
FIG.3.1	Diagrama de Funcionamento	26
FIG.3.2	Storyboard	28
FIG.4.1	Versão inicial - WW	30
FIG.4.2	Módulo no aplicativo Final - WW	30
FIG.4.3	Versão inicial - HC	32
FIG.4.4	Módulo no aplicativo Final - HC	32
FIG.4.5	Versão inicial - Sniffer	35
FIG.4.6	Módulo no aplicativo Final - Sniffer	35
FIG.4.7	Todos os Módulos	37
FIG.4.8	Tela inicial da Versão Final	37

LISTA DE SIGLAS

BSSID	Basic Service Set Identifier
CIA	Confidentiality, Integrity and Availability
CID	Confidencialidade, Integridade e Disponibilidade
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
NIST	National Institute of Standards and Technology
OSI	Open Systems Interconnection
SSID	Service Set Identifier
VM	Virtual Machine
WLAN	Wireless local area network

RESUMO

Este projeto tem por finalidade propor, desenvolver e analisar um aplicativo de *sniffer* de rede, fazendo uma análise da sua utilização bem como da extensão das informações possíveis de serem obtidas através de um aplicativo básico desse tipo.

O aplicativo permite a coleta de pacotes de rede a partir do momento em que algum indivíduo se conecte ao *hotspot* criado por ele, o qual é configurado de modo a se assemelhar a alguma rede *wireless* existente no ambiente.

Optou-se pela implementação para Android por ser a plataforma móvel de maior abertura atualmente. Já existem bibliotecas e projetos em linguagem JAVA que podem ser utilizadas para facilitar a captura de pacotes de rede e algumas APIs do próprio Android já facilitam o processo. O aplicativo construído foi sujeito a um experimento prático realizado dentro do Instituto Militar de Engenharia e foi verificado a captura de pacotes de usuários conectados ao *hotspot* criado.

ABSTRACT

The purpose of this project is the suggestion, development and analysis of a packet-sniffing app, also doing an analysis of the app usage, as well as of the extension of the information able to be captured by a basic app of this type.

The app allows the packet sniff as soon as the subject connects his device to the hotspot, created and set up by the app to mimic an already existing local wireless network.

It was chosen to implement the app for Android because it is the most open mobile platform nowadays. There are JAVA libraries and projects that can be used to help the implementation of a packet sniffer and some Android APIs really help the process. The application was built and submitted to an experiment held within the Instituto Militar de Engenharia, where was verified that the application successfully capture packets from users connected to the created hotspot.

1 INTRODUÇÃO

Durante séculos reis, rainhas e generais tiveram que confiar na eficiência de seus meios de comunicação para que assim pudessem governar seus países ou comandar suas tropas. Sabiam da importância desses meios e principalmente das consequências que trariam caso suas mensagens não fossem entregues ou fossem interceptadas por pessoas não autorizadas. Foi esse risco e medo das consequências que motivaram os desenvolvimentos de técnicas de criptografia e segurança da informação.

Numa era de conexão eletrônica universal, uma era de vírus e *hackers*, de invasão e fraude eletrônica, não existe mais a situação de não ter tempo para a segurança ou em que a segurança não é tão importante.

Esses fatores junto com a explosão de crescimento de sistemas computacionais interligados formando redes que gerou dependências de organizações e indivíduos para protegerem suas informações e dados, motivam o constante estudo da área da segurança da informação, tanto na parte de defesa como ataque.

Sabe-se que o estudo da segurança não deve somente se focar na defesa, pois o estudo do ataque cibernético além de ajudar no entendimento da área e assim agregar na formulação da defesa para os diversos ataques possíveis, deve servir como poder dissuasório para inimigos, auxiliando na proteção, principalmente, de entidades governamentais.

Chegamos assim ao foco desse projeto, que é a criação de um aplicativo de *sniffer*, um coletor de informações que pode ser considerado um ataque tanto passivo como ativo, dependendo de como implementado e utilizado, para produzir conhecimento nessa área que tanto cresce no mundo moderno em quantidade e importância da proteção do indivíduo e da pátria (STALLINGS, 2014), (SINGH, 2002).

1.1 MOTIVAÇÃO

O projeto tem por motivação o exercício e maior estudo da técnica de obtenção de informações através do *sniffer* de redes. O produto possui aplicações na área de ensino de segurança da informação, servindo como ferramenta para testes de segurança e maiores estudos na área, além de, futuramente, poder ser otimizado para melhorar as suas funcionalidades, como um aumento de detalhes e informações capturadas de cada pacote e até a possibilidade de reconstrução de páginas acessadas.

1.2 OBJETIVO

Este projeto tem como objetivo propor, desenvolver e analisar um aplicativo de *sniffer* de redes que permita o usuário criar um *hotspot* que simule uma rede *WiFi* local, além de monitorar e gravar as informações dos pacotes trocados na rede para cada usuário conectado a esse *hotspot*.

1.3 JUSTIFICATIVA

O desenvolvimento desse aplicativo serve para expor e estudar os riscos aos quais um sistema está sujeito ao utilizar uma rede *WiFi* de forma displicente. O projeto também possui grande valor acadêmico para a pesquisa na área de defesa cibernética, grande foco do Exército Brasileiro na atualidade, visando a crescente importância do nosso país no cenário mundial e na recepção de grandes eventos de caráter internacional, como a copa do mundo de futebol, realizada em 2014, e as olimpíadas de verão realizadas nesse ano de 2016 na cidade do Rio de Janeiro.

Visa também motivar o estudo e pesquisa de acadêmicos, estudantes e professores, para as áreas da segurança da informação, redes de computadores, criptografia entre outras.

1.4 METODOLOGIA

Inicialmente foi realizado um estudo referente aos conceitos relacionados à Segurança da Informação e programação em Android, dando enfoque especial a *sniffer* de redes *WiFi*, com o objetivo de, além de conhecer o que já existe de produto semelhante no mercado, conhecer os parâmetros necessários para a implementação do aplicativo, juntamente com as suas limitações.

Uma vez feito um estudo sobre a plataforma e determinado os limites da utilização de bibliotecas JAVA para *sniffer* de redes, como a *pcap4j* (KAITOY, 2016) e semelhantes, na programação Android, foi feita a implementação do aplicativo, a qual ocorre em 4 etapas.

As quatro etapas da implementação do aplicativo, que serão posteriormente detalhadas, consistem na divisão do aplicativo em 3 módulos, WiFi Watcher, Hotspot Creator e o Sniffer, que são implementados separadamente e posteriormente na quarta etapa é feita a junção de todos os módulos em um aplicativo único.

Após a implementação, foram realizados testes práticos para mostrar o uso do aplicativo em redes *WiFi* dentro do IME.

1.5 ORGANIZAÇÃO DA MONOGRAFIA

No capítulo 2 deste trabalho são introduzidos conceitos de Segurança de Informação e da plataforma utilizada. Neste capítulo, é realizado uma introdução à plataforma Android, com uma breve descrição do modelo de camadas e suas características, e em seguida uma introdução aos princípios da Segurança da Informação, com enfoque em *sniffer* de redes.

No capítulo 3 é apresentada a modelagem do produto, com um diagrama de funcionamento que explica de forma didática como o aplicativo funciona e o storyboard tomado como modelo para a implementação.

No capítulo seguinte tem-se a descrição do desenvolvimento do aplicativo em que é explicado a abordagem utilizada com o desenvolvimento de três aplicativos diferentes que foram, posteriormente, agregados resultando no produto final.

No capítulo 5 são apresentadas as conclusões do projeto. Nas seções seguintes tem-se as referências das fontes utilizadas para a realização desse documento e o apêndice com o código das principais classes do produto final, representando cada módulo.

2 FUNDAMENTAÇÃO TEÓRICA

Nessa seção é feita uma breve introdução da plataforma Android e aos conceitos de Segurança da Informação, com enfoque em *sniffer* de rede, para ambientar o leitor na área em que o projeto proposto se posiciona. Também são abordados conceitos importantes sobre redes *wireless*.

2.1 A PLATAFORMA ANDROID

A plataforma Android atualmente é considerada a plataforma para dispositivos móveis mais difundida do mundo, devido a sua ampla utilização nos mais variados dispositivos móveis. Suas aplicações podem ser escritas em JAVA e as ferramentas Android SDK compilam o código em um pacote APK (sufixo .apk). Os arquivos de APK contêm todo o conteúdo de um aplicativo do Android e são os arquivos que os dispositivos desenvolvidos para Android usam para instalar o aplicativo.

O sistema Android funciona em um sistema de Sandbox, o que, de forma resumida, significa que cada aplicativo do Android é ativado em sua própria área de segurança. O sistema operacional Android é baseado em um sistema Linux multiusuário em que cada aplicativo é um usuário diferente, cada processo tem sua própria máquina virtual (VM), portanto o código de um aplicativo é executado isoladamente de outros aplicativos. Por padrão, cada aplicativo é executado em seu próprio processo Linux (DEVELOPER, 2016a).

2.1.1 A ARQUITETURA ANDROID

Camada de Aplicações: É onde se localizam todos os aplicativos que são executados sobre o sistema operacional, tais como, cliente de SMS e MMS, cliente de e-mail, navegador, mapas, calculadora, dentre outros.

Camada de Framework de Aplicação: Camada onde se instancia a máquina virtual Dalvik, criada para cada aplicação executada no Android.

Camada de Bibliotecas: É a camada que possui bibliotecas C/C++ que são utilizadas pelo sistema e também bibliotecas de multimídia, visualização de camadas 2D e 3D, fun-

ções para navegadores web, funções de aceleradores de *hardware*, fontes *bitmap* e funções de acesso a banco de dados SQLite.

Camada de Runtime: Camada onde se instancia a máquina virtual Dalvik, desenvolvida para cada aplicação executada no Android.

Camada de Kernel Linux: O núcleo do sistema operacional Android. Parte importante do Linux utilizada na concepção do Google Android é o de controle de processos, gerenciarem memória, *threads*, protocolos de rede, modelos de *drivers* e a segurança dos arquivos.

Na Figura 2.1 pode-se ter uma visualização da separação das camadas da arquitetura Android.

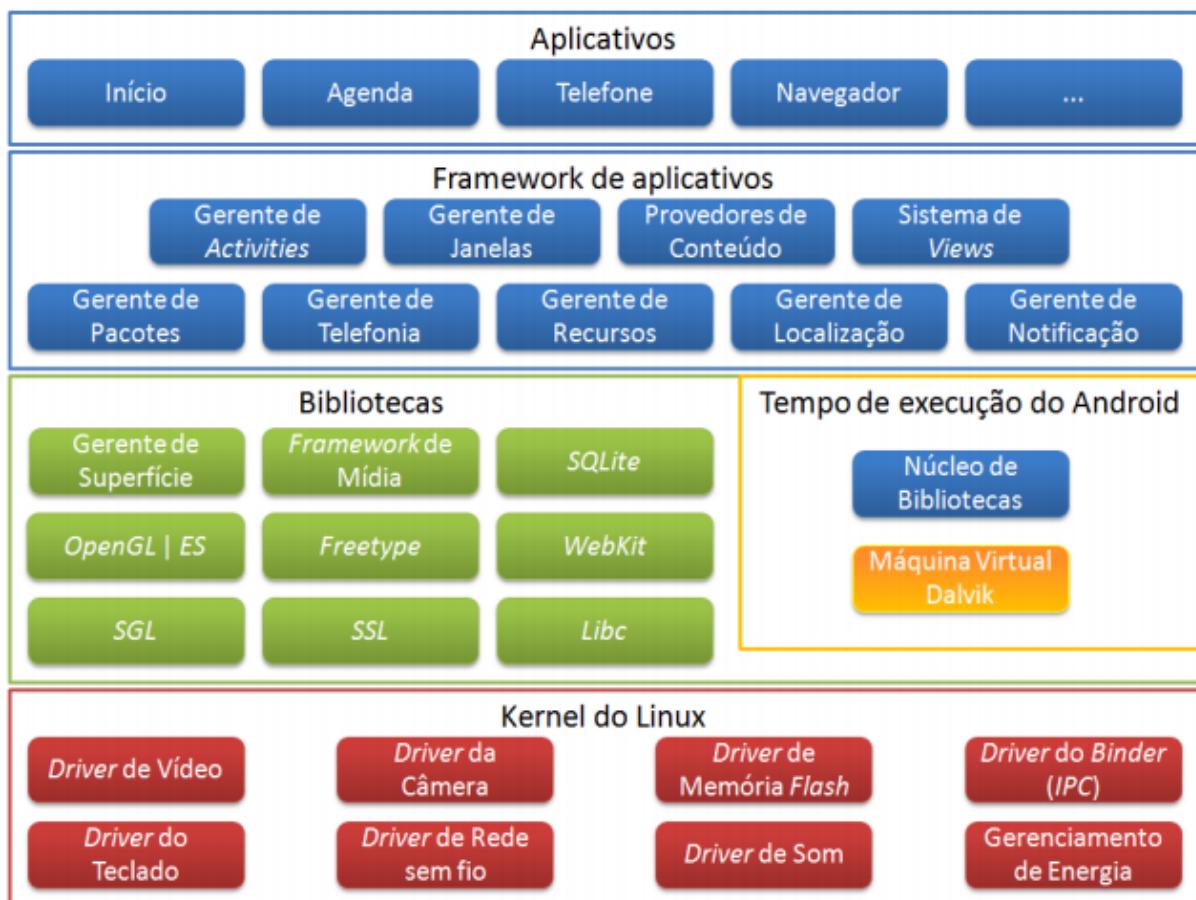


FIG. 2.1: Arquitetura de Camadas Android

2.1.2 COMPONENTES DE UMA APLICAÇÃO

Atividades: As atividades representam uma tela única na aplicação. Possui interface do usuário composta por Views, componentes gráficos, eventos etc.

Serviços: São componentes executados em segundo plano para realizar operações de execução longa ou para realizar trabalho para processos remotos. Eles não apresentam uma interface do usuário.

Provedores de conteúdo: Gerenciam um conjunto compartilhado de dados do aplicativo. É possível armazenar os dados no sistema de arquivos, em um banco de dados SQLite ou em qualquer local de armazenamento persistente que o aplicativo possa acessar. Por meio do provedor de conteúdo, outros aplicativos podem consultar ou até modificar os dados (se o provedor de conteúdo permitir).

Receptores de transmissão: Ou receptores de *broadcast*, são componentes que respondem a anúncios de transmissão por todo o sistema. Muitas transmissões se originam do sistema, como por exemplo, uma transmissão que anuncia que uma tela foi desligada, a bateria está baixa ou uma tela foi capturada (SHARMA, 2016).

2.1.3 ANDROID MANIFEST.XML

O arquivo *Android Manifest.xml* é um arquivo que deve estar na pasta raiz do projeto da aplicação e que contém declarações dos componentes da aplicação, os recursos de *hardware* e *software* usados, permissões e o menor API level requerido para rodar a aplicação, assim como as bibliotecas usadas entre outras informações.

2.1.4 INTENTS E FILTROS INTENT

Activities, *services* e *broadcast receivers* são componentes de aplicação que podem ser ativados por uma mensagem assíncrona chamada *intent* (intenção). *Intents* fazem a requisição de determinadas ações à componentes, estas podendo pertencer a outras aplicações além da própria. O *Intent* carrega o nome da ação a ser realizada, no caso de *services* e *activities*, ou o nome do evento sendo anunciado, no caso de *broadcast receivers*. *Intents* podem ser filtrados por uma aplicação para especificar quais *intents* podem ser processados pelos componentes da aplicação. A lista de filtros é definida no arquivo

manifest, assim o Android pode determinar quais *intents* são permitidos antes de iniciar a aplicação.

2.1.5 ANDROID MARSHMALLOW

A Google anunciou oficialmente o Android 6.0 Marshmallow em Outubro de 2015(NEWS; NOTES FROM ANDROID TEAM, 2016). Dentre as várias novidades oferecidas na nova versão, como um recurso de economia de energia chamado de *Doze* e opção de formatar o cartão de memória para funcionar como memória interna, o que possivelmente chamou mais a atenção dos desenvolvedores foi a mudança no sistema de permissões de aplicativos.

No Android Lollipop, versão anterior, todas as permissões de aplicativos deveriam ser aceitas no momento da instalação do mesmo para a sua utilização. Com o Android Marshmallow, as permissões são requeridas ao usuário no momento que o aplicativo precisa executar a função determinada (como acessar o microfone ou a localização), sendo disponível também, nas configurações do Android, a opção de bloquear permissões individuais de um app.

Essas mudanças na segurança dos aplicativos mudaram a forma de programação das permissões em um projeto, ou seja, todos os projetos que utilizavam permissões até a versão anterior tiveram que ser atualizados para se adequar ao novo sistema de permissões.

2.2 CONCEITOS DE SEGURANÇA DA COMPUTAÇÃO

Primeiramente é apresentado para análise a definição dada pela NIST (*National Institute of Standards and Technology*) para o termo segurança da computação:

“É a proteção conferida a um sistema de informação automatizado, a fim de atingir os objetivos aplicáveis de preservação da integridade, disponibilidade e confidencialidade dos recursos do sistema de informação (inclui hardware, software, firmware, informação/dados e de telecomunicações).” [NIST Computer Security Handbook (NIST95)]

Nessa breve definição observa-se a introdução de três conceitos muito importantes na área de segurança, integridade, disponibilidade e confidencialidade. Esses três conceitos formam juntos a tríade da CIA (*Confidentiality, Integrity and Availability*) que fundamentam os objetivos de segurança, tanto para informações como para serviços.

- **Confidencialidade:** Dois conceitos

Confidencialidade de dados: Garante que a informações privadas ou confidenciais não são disponibilizadas ou divulgadas a pessoas não autorizadas.

Privacidade: Garante que indivíduos controlem ou influenciem as informações relacionadas a eles que podem ser recolhidas e armazenadas e por quem ou para quem que as informações podem ser divulgadas.

- **Integridade**: Dois conceitos

Integridade dos dados: Garante que a informação e os programas são alterados somente de uma forma especificada e autorizada.

Integridade do sistema: Garante que um sistema execute a sua função prevista em uma forma perfeita, livre de manipulações deliberadas ou inadvertidas não autorizada do sistema.

- **Disponibilidade**: Garante que os sistemas funcionem prontamente e serviços não sejam negados a usuários autorizados.



FIG. 2.2: Tríade CID

Apesar de bem estabelecidos, os conceitos da CID em geral também são acompanhados por outros dois conceitos importantes, autenticidade e responsabilidade.

- Autenticidade: A propriedade de ser genuína e ser capaz de ser verificável e confiável; confiança na validade de uma transmissão, de uma mensagem, ou no originador da mensagem. Isto significa verificar que os usuários são quem eles dizem que são e que cada insumo que chega ao sistema veio de uma fonte confiável.

- Responsabilidade: A meta de segurança que gera a necessidade de ações de uma entidade serem traçadas exclusivamente a essa entidade. Isto suporta não-repúdio, dissu-

asão, isolamento de falhas, detecção de intrusão e prevenção, e recuperação após ação e ação legal. Como os sistemas verdadeiramente seguros não são ainda uma meta alcançável, deve existir a capacidade de rastrear uma falha de segurança para um responsável. Os sistemas devem manter registros de suas atividades a fim de permitir análise forense para rastrear violações de segurança ou para ajudar nas disputas de transações.

2.3 ARQUITETURA ISO PARA SEGURANÇA

Para ter uma noção de como a segurança do sistema é estabelecida, devemos conhecer a arquitetura geralmente aceita de configurações de segurança cibernética. A arquitetura de segurança para o *Open System Interconnect* (OSI) foi designado pelo ITU-T (União Internacional de Telecomunicações - Telecomunicações). Esta arquitetura de segurança é útil para os gestores como forma de organizar a tarefa de fornecer segurança. A ITU-T decidiu que seu padrão 'X.800' seria a arquitetura de segurança da ISO, organização internacional de padrões.

Essa arquitetura padronizada define requisitos de segurança e especifica os meios pelos quais estes requisitos podem ser satisfeitos. A Arquitetura OSI possui foco nos seguintes campos e definições:

Ataque de Segurança: Qualquer ação que comprometa a segurança das propriedades de informações de uma organização.

Mecanismo de segurança: Processo que é projetado para detectar, prevenir ou recuperar de um ataque de segurança.

Serviço de segurança: Um serviço de processamento ou de comunicação que melhora a segurança dos sistemas de processamento de dados e informações de transferência de uma organização. Os serviços são destinados a combater os ataques de segurança, fazendo o uso de um ou mais mecanismos de segurança para fornecer o serviço (STALLINGS, 2014),(TELECOMMUNICATION, 2016).

2.4 ATAQUES E VULNERABILIDADES

Um meio útil de classificação de ataques de segurança é em termos de ataques passivos e ativos. Um ataque passivo tenta aprender ou fazer uso de informações do sistema, mas não afeta os recursos do sistema. O ataque ativo tenta alterar recursos do sistema ou afetar o seu funcionamento.

2.4.1 ATAQUES PASSIVOS

Têm a natureza de espionagem ou monitorização de transmissões. O objetivo do adversário é a obtenção de informação que está sendo transmitida. Dois tipos de ataques passivos são a liberação do conteúdo das mensagens e análise de tráfego.

A liberação do conteúdo das mensagens é facilmente compreendida. A conversa por telefone, uma mensagem de correio eletrônico ou um arquivo transferido podem conter informações sensíveis. Desse modo a segurança nesse caso teria como objetivo impedir que o atacante tivesse acesso ao conteúdo das informações transmitidas.

O segundo tipo de ataque passivo é mais sutil. A análise de tráfego consiste na captura das mensagens, mesmo criptografadas, impedindo a extração de informação, podem apresentar algum padrão de comunicação, como determinar a localização dos envolvidos, frequência e duração das mensagens trocadas.

Ataques do tipo análise de tráfego, ou do tipo passivo em geral, são de difícil detecção, de modo que os mecanismos de segurança focam em lidar na prevenção do ataque ao invés da detecção.

2.4.2 ATAQUES ATIVOS

Envolvem alguma modificação do fluxo de dados ou na criação de um falso fluxo e pode ser subdividido em quatro categorias: mascarado, *replay*, modificação de mensagens, e negação de serviço (DoS).

O mascarado ocorre quando uma entidade pretende se passar por uma entidade diferente. Um ataque de máscaras normalmente inclui uma das outras formas de ataques ativos.

Replay envolve a captura passiva de uma unidade de dados e sua retransmissão subsequente para produzir um efeito não autorizado.

Modificação de mensagens significa simplesmente que alguma parte de uma mensagem legítima é alterada, ou que as mensagens estão atrasadas ou reordenadas, para produzir um efeito não autorizado.

A negação de serviço (*DoS – Denial of Service*) impede ou inibe a utilização normal ou gestão de serviços de comunicação (STALLINGS, 2014).

2.4.3 SNIFFING

A técnica de *sniffing* pode ou não ser considerada um ataque, nesse caso do tipo passivo. Um *sniffer* de rede se baseia no monitoramento do fluxo de dados de uma determinada

rede de computadores, fazendo uma cópia dos pacotes sem redirecioná-los ou alterá-los.

Sabendo dessa propriedade do *sniffer*, considerado um ataque, seria um ataque passivo e dependendo da segurança em si dos pacotes, se possuem criptografia muito forte impedindo a extração de informação ou não, pode ser uma liberação de conteúdo das mensagens ou simplesmente um ataque do tipo análise de tráfego. Desse modo o *sniffer* estaria atacando a tríade CID na parte de confidencialidade, mas também podendo ser usada como um mecanismo de segurança ajudando nos conceitos de responsabilidade e autenticidade (MITCHELL, 2016),(KEVIN, 2003).

2.5 WIFI

Redes sem fio, e os dispositivos que as usam, introduzem uma série de problemas com segurança muito além dos encontrados em redes com fio. Alguns dos fatores chaves que contribuem para esses novos problemas incluem o canal de comunicação e recursos disponíveis no aparelho.

O canal de comunicação que, tipicamente, em redes sem fio envolvem comunicações *broadcast*, tornam a rede muito mais suscetível a espionagem e interferências, o que acaba sendo um grande problema, visto que muitos dispositivos sem fio, como *Tablets* e *smartphones*, possuem recursos limitados de processamento e memória, o que torna o uso de ferramentas de defesa difícil.

2.5.1 IEEE 802.11 WIRELESS LAN

IEEE 802.11 é um comitê que desenvolve padrões para uma grande gama de redes locais (*LANs – local area networks*). Em 1990, o comitê IEEE 802 formou um novo grupo, IEEE 802.11, com o objetivo de desenvolver um protocolo e especificações para LANs sem fio (*Wireless LAN – WLAN*). Desde então a demanda para WLANs em diferentes frequências e taxas de transmissão de dados vem se expandindo. Para manter o ritmo dessa crescente demanda, o grupo IEEE 802.11 emite uma lista de padrões que sempre se mantêm em expansão.

A tecnologia WiFi é baseada no padrão IEEE 802.11, no entanto, isso não quer dizer que todo produto que trabalhe com estas especificações seja também WiFi. Para que um determinado produto receba um selo com esta marca, é necessário que ele seja avaliado e certificado pela *Wi-Fi Alliance* (WIFI ALIANCE, 2016). Esta é uma maneira de garantir ao usuário que todos os produtos com o selo *WiFi Certified* seguem normas de funcionalidade que garantem a interoperabilidade com outros equipamentos. A *Wi-*

Fi Alliance preocupa-se com uma gama de áreas de mercado para WLANs, incluindo empresas, casas e *hotspots*.

Mais recentemente, a *Wi-Fi Alliance* tem desenvolvido procedimentos de certificação para normas de segurança IEEE 802.11, referidos como *Wi-Fi Protected Access (WPA)*. A versão mais recente do WPA, conhecido como WPA2, incorpora todas as características do especificação de segurança IEEE 802.11i WLAN (STALLINGS, 2014).

2.5.2 CONCEITOS IMPORTANTES

2.5.2.1 SSID - SERVICE SET IDENTIFIER

O SSID é o nome público de uma rede *wireless*. Todos os dispositivos wireless de uma WLAN devem conter o mesmo SSID para poder comunicar entre si.

2.5.2.2 BSSID

BSSID é o *Mac Address* do ponto de acesso *wireless*. Mac Address (*Media Access Control Address*) é o endereço único que individualiza o *hardware* de rede, seja ele uma placa Ethernet ou um Ponto de Acesso.

2.5.2.3 HOTSPOT

Um *hotspot* é um local onde as pessoas podem obter acesso à internet, tipicamente utilizando tecnologia *WiFi*, através de uma rede sem fio local, usando um roteador conectado a um provedor de internet. São encontrados geralmente em locais públicos como cafés, restaurantes, hotéis e aeroportos onde é possível conectar-se à Internet utilizando qualquer aparelho portátil que esteja preparado para se comunicar em uma rede sem fio do tipo WiFi, como *smartphones*, *laptops* e *tablets*.

2.5.2.4 PONTO DE ACESSO

É um dispositivo em uma rede sem fio que realiza a interconexão entre todos os dispositivos móveis. Em geral se conecta a uma rede cabeadas servindo de ponto de acesso para uma outra rede, como por exemplo a Internet. Está ligado a camada de Enlace.

2.5.2.5 PROVEDOR DE SERVIÇO DE INTERNET

São empresas que oferecem a usuários individuais a possibilidade de acessar um de seus equipamentos e se conectar à Internet, obtendo assim acesso a todos os serviços da in-

ternet. Essas empresas geralmente cobram um valor mensal de seus clientes que varia de acordo com a velocidade de transferência de dados contratada.

3 MODELAGEM

3.1 DIAGRAMA DE FUNCIONAMENTO

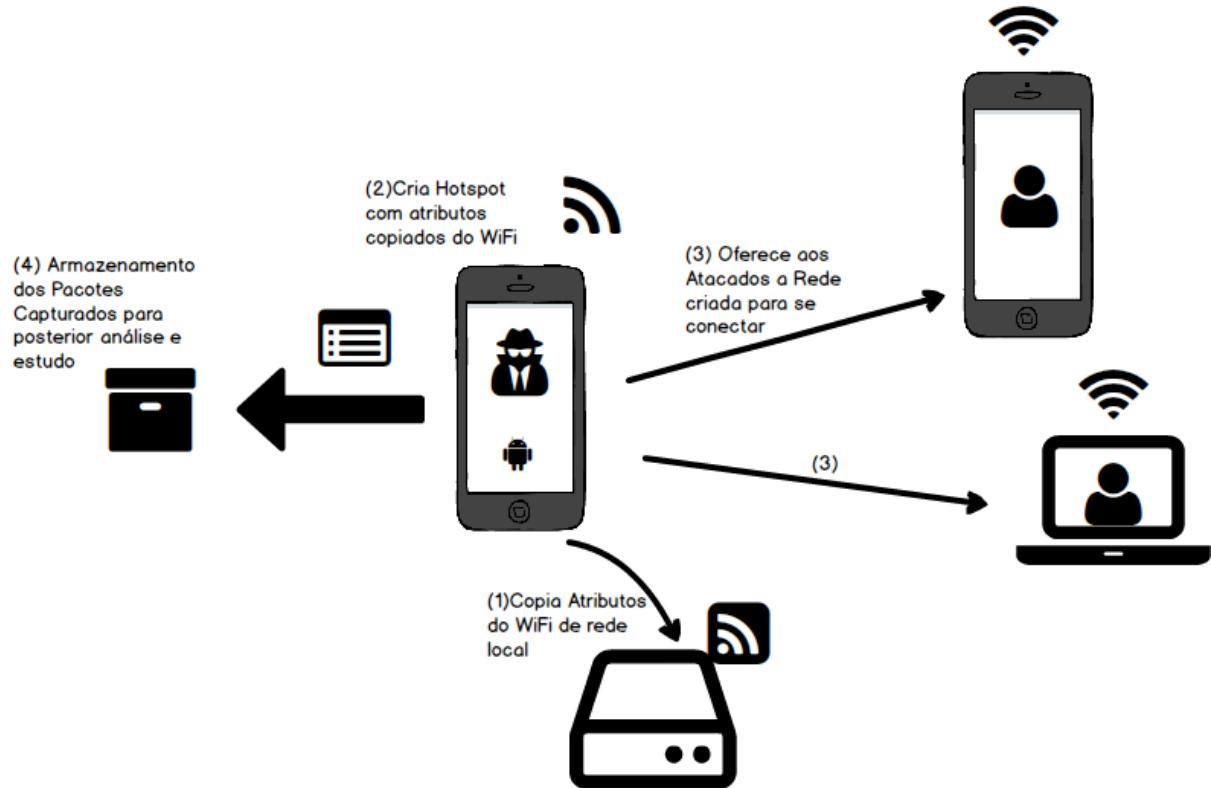


FIG. 3.1: Diagrama de Funcionamento

Na Figura 3.1 tem-se um diagrama didático do funcionamento e objetivo do aplicativo. No primeiro passo (1) tem-se o dispositivo que possui o aplicativo copiando atributos (SSID, BSSID, etc.) da rede *WiFi* desejada presente no ambiente, em seguida (2) o dispositivo cria um *hotspot* com esses atributos fazendo se passar pela rede WiFi original, desse modo pode (3) oferecer o acesso a qualquer dispositivo presente que tenha como objetivo se conectar à rede original, mas irá ao final se conectar ao dispositivo com o aplicativo.

O dispositivo poderá se conectar ao aparelho com o aplicativo por causa do sinal mais intenso criado pelo *hotspot*, por questão de proximidade, juntamente com a negligência do usuário do dispositivo atacado. Finalmente (4) o dispositivo com o aplicativo faz a captura de pacotes acessados pelo(s) dispositivo(s) conectado(s) para posterior análise.

3.2 LEVANTAMENTO DE REQUISITOS

Para esse aplicativo tem-se um breve levantamento de requisitos, que será separado em três escopos, modo análogo ao desenvolvimento do aplicativo que será melhor detalhado na seção seguinte.

3.2.1 ESCOPO DE OBTENÇÃO DO SSID E DO BSSID

Nesse escopo tem-se como objetivo a obtenção do SSID e do BSSID de alguma rede local. Para isso existe como requisitos que o dispositivo que possui o aplicativo esteja com a funcionalidade de WiFi funcionando corretamente e que ele também se encontre em alguma área que possuam redes WiFi disponíveis. Essas redes devem estar disponíveis para serem descobertas, mas não necessariamente abertas para o acesso, apenas a descoberta de sua existência deve ser aberta.

3.2.2 ESCOPO DO HOTSPOT

Para a criação de um Hotspot o dispositivo deve possuir alguma fonte de internet própria para que seja capaz de permitir o acesso de outros dispositivos à internet através do celular.

No momento de criação do Hotspot, deve existir a possibilidade do usuário realizar algumas alterações em suas configurações para que estejam de acordo com o objetivo do usuário.

3.2.3 ESCOPO DO SNIFFER

Deve existir a possibilidade de seleção apenas dos dispositivos alvos para a realização do sniffer, da identificação dos dispositivos conectados, da identificação dos diferentes protocolos assim como a sequência temporal que esses pacotes trafegaram e foram capturados para posterior análise.

3.3 STORYBOARD

Em seguida, na figura 3.2, tem-se o *storyboard* do aplicativo. Nele, pode-se observar a sequência para a criação de um *hotspot*, copiando o SSID e o BSSID de alguma rede local, nas segunda e terceira telas, oferecendo o *hotspot* para que dispositivos possam se conectar, na quarta tela, e então fazendo a captura de pacotes dos dispositivos conectados selecionados, na quinta e última tela.

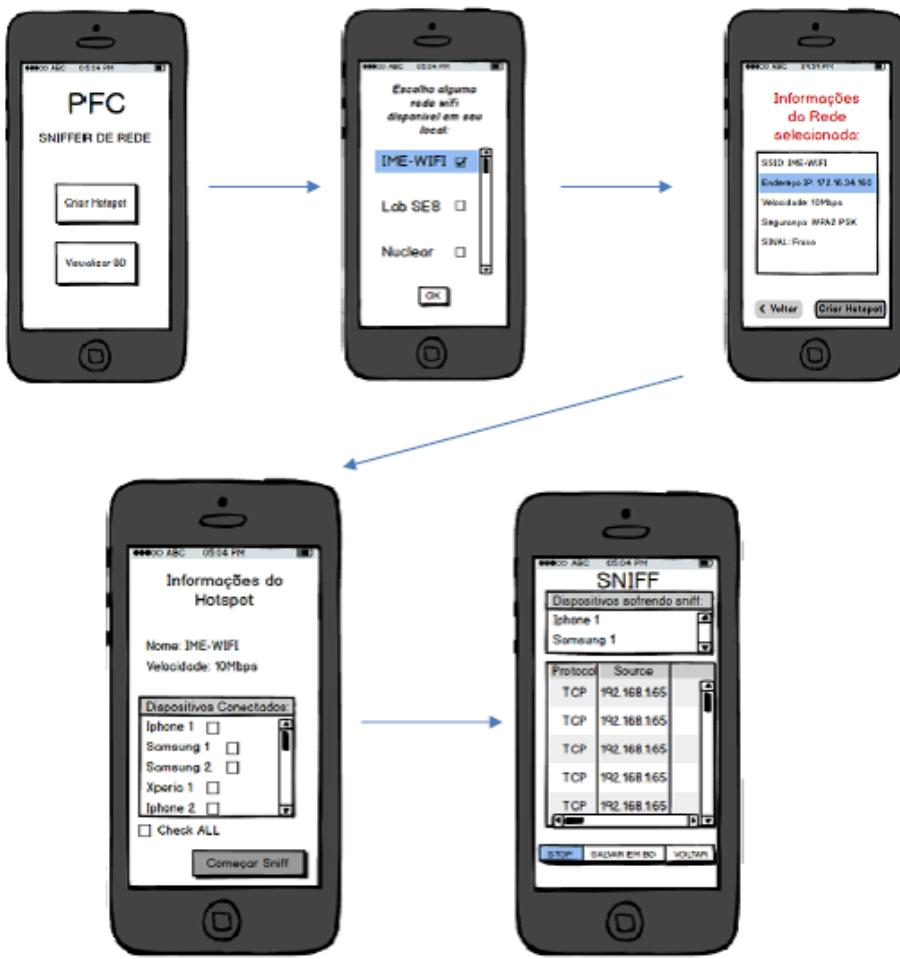


FIG. 3.2: Storyboard

Importante notar que a captura de pacotes será focada em dispositivos alvos que estão conectados ao usuário, e que primeiramente não existe nenhuma análise dos pacotes, apenas a captura, já separada por protocolos, fonte, destino, etc. A análise dos pacotes deve ser realizada em momento posterior com a utilização de outras ferramentas que não fazem parte do aplicativo. Este apenas irá capturar e armazenar o arquivo com o histórico da captura em seu banco de dados.

4 DESENVOLVIMENTO

Devido à complexidade do projeto, decidiu-se pela abordagem da implementação do aplicativo em formato de módulos, os quais representam aplicações independentes capazes de reproduzir, de forma individual, as principais funcionalidades exigidas pelo projeto.

O desenvolvimento do projeto é dividido em 4 etapas, sendo nas 3 primeiras implementado um módulo (aplicativo) em cada e, na quarta e última etapa, é gerado o aplicativo final através da união dos aplicativos implementados nas etapas anteriores.

A etapa 1 é a etapa em que é implementada a funcionalidade de obtenção e exibição dos dados principais das redes sem fio disponíveis no local. Foi então desenvolvido um aplicativo com a única funcionalidade de retornar na tela, um texto simples, o número de conexões WiFi disponíveis nas proximidades juntamente com uma lista com as informações de SSID e BSSID de cada rede.

A etapa 2 é a etapa responsável pela implementação da funcionalidade de Hotspot. O aplicativo desenvolvido nesta fase tem a única função de criar um ponto de acesso WiFi aberto (sem senha) com o SSID definido previamente pelo usuário, nesse momento ainda não é importante a definição do BSSID pois essa rede criada não é a cópia de nenhuma rede, essa funcionalidade de definição do BSSID será implementada na etapa de fusão dos aplicativos.

A etapa 3 é a etapa em que se implementa o aplicativo com a funcionalidade de *sniffer*. Nesta etapa é verificada a possibilidade de obtenção de pacotes trocados entre o próprio celular e a Internet. São definidos os requisitos para tal funcionalidade e finalmente foi realizada a implementação de tal.

Por fim, na etapa 4 é onde é feita a implementação do aplicativo final, o qual representa a união de todos os módulos implementados anteriormente.

4.1 ETAPA 1: MÓDULO APP WIFI WATCHER

Nessa primeira etapa é realizado a implementação do primeiro módulo, o aplicativo WiFi Watcher. Tal aplicativo tem como única função obter e exibir na tela todas as informações de SSID e BSSID das conexões locais WiFi disponíveis. O aplicativo foi feito utilizando-se da *api* nativa do Android *android.net.wifi.WifiManager* (DEVELOPERS, 2016c).

4.1.1 DESCRIÇÃO DO MÓDULO

O módulo dessa etapa tem como objetivo a implementação da funcionalidade de captura e impressão das informações de redes WiFi locais.

O aplicativo é composto de uma tela com um botão do tipo *switch*, para ligar ou desligar a WiFi, e um botão com o nome "Procurar redes WiFi", que ao ser pressionado retorna uma lista na mesma tela com o número total de redes disponíveis e uma descrição de cada rede com informações como SSID e BSSID das mesmas. Caso a funcionalidade de WiFi esteja desligada, uma mensagem TOAST avisará o usuário para ligar a WiFi, o que pode ser feito clicando no botão de *switch*.

4.1.2 JUSTIFICATIVA E IMPORTÂNCIA

Essa implementação é justificada pela necessidade dessa funcionalidade de obtenção e exibição de informações de uma rede WiFi local que o aplicativo final possuirá pois, em sua versão completa, o aplicativo do projeto primeiramente exibirá as informações de rede WiFi locais, em seguida copiando as informações de SSID e BSSID da rede escolhida pelo usuário para utilizar na configuração de um ponto de acesso (etapa seguinte).



FIG. 4.1: Versão inicial - WW

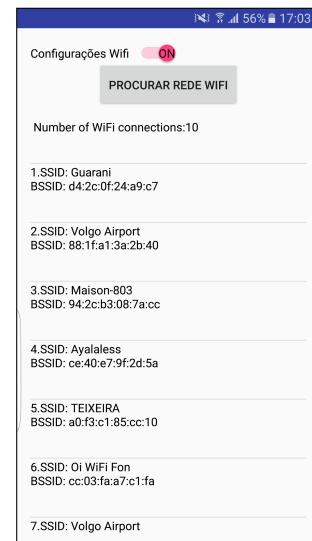


FIG. 4.2: Módulo no aplicativo Final - WW

4.1.3 IMPLEMENTAÇÃO

Para a implementação dessa funcionalidade foi utilizado um *broadcast receiver*, que verifica mudanças no número de conexões Wifi disponíveis, e foram exploradas classes como *android.net.wifi.WifiManager* (DEVELOPER, 2016b), que possui métodos para ligar e desligar a função WiFi do aparelho, e *android.net.wifi.ScanResult*, classe com métodos que permitem a pesquisa das conexões(redes) disponíveis no local assim como captura as informações necessárias para serem impressas na tela.

4.2 ETAPA 2: MÓDULO APP HOTSPOT CREATOR

Nessa etapa foi realizada a implementação da funcionalidade de criação de *hotspot* com SSID definido pelo usuário.

O aplicativo resultante da implementação desse módulo, denominado Hotspot Creator, possui um botão e um espaço para texto onde se pode definir o nome do ponto de acesso que se deseja criar. Ao se clicar no botão para criar o *hotspot*, em seguida a conexão WiFi atual é desligada e é iniciado o novo ponto de acesso WiFi com o SSID definido pelo usuário.

Para se implementar as funcionalidades básicas de inicialização e configuração do *hotspot*, foram utilizadas a *api android.net.wifi* e *android.net.ConnectivityManager*, ambas nativas, onde foram mais exploradas as classes *WifiManager* e *WiFiConfiguration*, assim como a *ConnectivityManager* para a conexão com a rede móvel.

4.2.1 DESCRIÇÃO DO MÓDULO

O módulo dessa etapa tem como objetivo permitir o usuário criar um Hotspot definindo o seu SSID com antecedência.

O aplicativo consiste de uma tela com um caixa de texto editável, para o usuário escrever o nome (SSID) da rede que quer criar (nome do *hotspot*), e um botão de nome "Criar Hotspot". Se o usuário não escrever nada na caixa de texto e clicar no botão, o Hotspot criado terá por padrão o nome "IME-WIFI". Se a função WiFi estiver ativa no aparelho, o aplicativo automaticamente irá desativá-la para a criação do Hotspot.

Por fim, uma breve mensagem é impressa na tela após a criação do Hotspot confirmado que este foi criado com êxito.

4.2.2 IMPORTÂNCIA E POSIÇÃO NO APLICATIVO FINAL

Esse módulo é parte crucial do projeto, pois facilitará a criação do *hotspot* a partir das configurações de uma rede WiFi local e permitirá que outros usuários se conectem à essa rede para que possa ser feito o *sniffing*.

Tomando-se o módulo anterior e o integrando com este, bastará apenas implementar o envio das informações da rede selecionada (no módulo anterior) para o módulo atual e, em vez deste receber as configurações por texto inserido pelo usuário como o módulo individualmente funciona, este receberá as configurações automaticamente e funcionará da mesma forma quando o usuário clicar no botão "Criar Hotspot".

4.2.3 IMPLEMENTAÇÃO

O aplicativo foi inicialmente implementado utilizando as funcionalidades básicas da *api android.net.wifi* (DEVELOPERS, 2016b), onde foram mais exploradas as classes *WifiManager* e *WiFiConfiguration* para a criação e configuração do Hotspot. Para habilitar a conexão da rede móvel usou-se o objeto *ConnectivityManager*

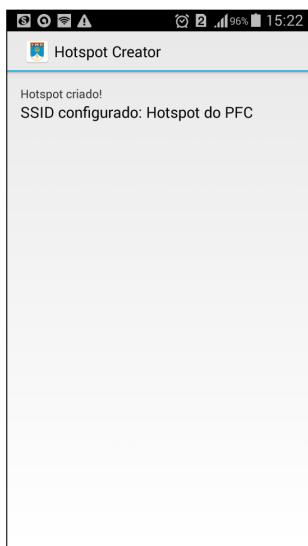


FIG. 4.3: Versão inicial - HC



FIG. 4.4: Módulo no aplicativo Final - HC

4.3 ETAPA 3: MÓDULO APP WIFI SNIFFER

Nesta etapa o objetivo foi implementar um aplicativo que fizesse a captura e leitura do fluxo de pacotes acessados pelo próprio usuário atacado ao acessar a Internet. Os pacotes recebidos e enviados são exibidos na tela em tempo de execução e salvos em

arquivos de texto. Com este módulo implementado, concluem-se todas as funcionalidades imprescindíveis para um *sniffer* de rede.

Duas possibilidades de implementação foram exploradas, uma utilizando a biblioteca libpcap em um aparelho Android que possui *root* e um método que não necessita de *root* e usa uma VPN. Optou-se pela segunda opção e o aplicativo foi desenvolvido a partir de um projeto aberto chamado LocalVPN encontrado no *github*. O aplicativo final conseguiu capturar pacotes e mostra-los na tela, sendo possível capturar pacotes apenas na forma de bytes puros, que foram transformados para string na tela. Informações como IP de origem, IP de destino e protocolo utilizado em cada pacote não foram identificáveis, mas pode-se identificar partes do cabeçalho dos pacotes ao se analisar os arquivos .txt dos resultados do sniffer salvos na memória interna.

4.3.1 DESCRIÇÃO DO MÓDULO

O aplicativo implementado nesse módulo chama-se Sniffer. Nesse módulo o aplicativo a ser implementado possui apenas uma tela com o botão "Start sniffer" e, quando o usuário selecionar nesse botão, é pedido permissão para o usuário para a criação da VPN e, quando concedida, uma mensagem será exibida na tela avisando que o *sniffer* foi inicializado. A partir desse momento um serviço é inicializado e o aplicativo ficará realizando o *sniffing* em *background* e gravando essas informações em duas listas em arquivos de texto no diretório de *Downloads* na memória interna do aparelho.

4.3.2 IMPORTÂNCIA E POSIÇÃO NO APLICATIVO FINAL

Esse módulo é o que de maior importância dentre os três, sendo a sua implementação bem sucedida fator decisivo para a viabilidade do projeto.

Com a sua implementação, tornou-se possível a realização de testes para confirmar o sucesso do projeto final da forma que já pode-se criar de forma independente um *hotspot* no *smartphone* e, quando algum usuário externo se conectar a este, pode-se utilizar o aplicativo recém criado para criar o VPN e iniciar o *sniffing* de pacotes na rede e assim, capturar pacotes trocados entre os usuários conectados ao *hotspot*.

A partir dessa etapa bastou realizar a integração com os dois módulos anteriores e realizar as adaptações necessárias para executar a captura dos pacotes da rede criada (*hotspot*) e de algum cliente específico conectado à essa rede.

4.3.3 IMPLEMENTAÇÃO

Na implementação desse módulo, pensou-se inicialmente na utilização da biblioteca *pcap4j*, uma biblioteca em Java para obtenção de pacotes (KAITOY, 2016). Porém, a utilização dessa biblioteca necessitaria a compilação da biblioteca libpcap no core de bibliotecas Android, o que necessitaria de um aparelho que possuísse *root* e a manipulação de código nativo. Uma outra solução encontrada para a implementação de um aplicativo android que realizasse *sniffing* foi utilizando uma VPN, solução utilizada por diversos aplicativos na *Google Play Store*, como o *PacketCapture* (SHIRTS, 2016) e o *tPacketCapture* (CO., 2016). Essa abordagem funciona sem a necessidade de *root* a medida que se cria uma camada a mais de VPN a qual se pode utilizar para realizar a captura e pacotes, visto que os fluxos de dados do VPNService se encontram na camada de Rede.

A abordagem final para esse módulo foi a de utilizar uma VPN e, assim, o *sniffer* foi desenvolvido a partir de um pequeno projeto chamado LocalVPN (NAUFAL, 2016) que explora a API VPNService do próprio Android. Desenvolvido por Mohamed Naufal, o aplicativo LocalVPN é um protótipo de VPN para Android sem autenticação mútua ou tratamento para lidar com pacotes inesperados. O projeto não usa NDK ou código nativo e foi escolhido justamente por se implementado inteiramente em Java e por sua facilidade de compreensão e modificação de código.

4.3.3.1 LOCALVPN

O aplicativo LocalVPN funciona de forma simples. Este possui apenas uma *activity* com um botão "*Start VPN*". Quando o usuário seleciona o botão este recebe uma mensagem pedindo permissão para a criação do VPN e quando esta é dada o serviço do VPN começa a executar em *background*. Na barra de notificação pode-se visualizar apenas o total em MB de pacotes recebidos e enviados, há um botão com a opção para se desconectar.

Quando um aplicativo faz uma requisição de algum recurso externo (requisição do Android para algum servidor na Internet) a requisição passa pela classe *LocalVNPService.class*. Na *LocalVPNService* os pacotes TCP/UDP são investigados e os IP's de origem e destino são extraídos (a classe *Packet.class* é usada para tal). Basicamente o LocalVPN roteia todo o tráfego em uma outra interface criada no smartphone e envia todos os dados para a internet.

4.3.3.2 SNIFFER

Para implementar o *Sniffer* inicialmente foi identificado o *ByteBuffer* utilizado no momento de troca de pacotes e enviado o fluxo de *bytes* para a *activity* principal para assim tornar visível para o usuário os pacotes recebidos e enviados. O envio de informações de um *thread* (no *LocalVPNService*) para uma *activity* foi feito utilizando um *Handler*, mas essa funcionalidade apenas mostrava informações ilegíveis devido ao fato da extração das informações dos pacotes ter sido feita no formato binário. Foi decidido então por não mostrar a captura em tela e foi implementada um comando para salvar os dados dos pacotes recebidos e enviados em arquivos de textos separados na pasta *Downloads*, por esse motivo foi implementada uma requisição de permissão em tempo de execução para escrever na memória interna.



FIG. 4.5: Versão inicial - Sniffer



FIG. 4.6: Módulo no aplicativo Final - Sniffer

4.4 ETAPA 4: UNIFICAÇÃO DOS APLICATIVOS

Nesta etapa é realizada a unificação dos três aplicativos desenvolvidos nos módulos anteriores. O produto final dessa etapa tem como objetivo se aproximar o máximo possível do aplicativo idealizado e modelado durante todo esse documento.

O aplicativo resultante da finalização desse módulo, denominado PFC Sniffer, possui uma tela inicial com as opções “Iniciar Sniffer” e “Visualizar WiFi e Criar Hotspot”. Ao escolher a primeira opção inicia-se o aplicativo WiFi Sniffer, mesmo sem a inicialização de um *hotspot* do aplicativo, assim fazendo o *sniffing* apenas do tráfego no próprio aparelho. Ao escolher a segunda opção será iniciado o primeiro aplicativo, WiFi Watcher, que ao

selecionar alguma rede da lista apresentada irá iniciar automaticamente o segundo aplicativo, Hotspot Creator, que irá receber informações do aplicativo anterior para realizar as configurações do SSID e BSSID do Hotspot que será em seguida inicializado ao selecionar “Criar Hotspot”.

4.4.1 UNIFICAÇÃO DOS APLICATIVOS WIFI WATCHER E HOTSPOT CREATOR

A unificação desses aplicativos foi simples. Foi utilizado um ListView em que eram exibidos as redes WiFi disponíveis no local, e, ao selecionar uma rede da lista, é realizado uma chamada para a segunda atividade do aplicativo, Hotspot Creator, através de um objeto “Intent” que recebe atributos extras como parâmetros a serem usados na segunda atividade, no caso o SSID e BSSID da rede selecionada. Assim é possível a inicialização do Hotspot com os atributos SSID e BSSID iguais à rede selecionada, como modelado no capítulo anterior.

4.4.2 UNIFICAÇÃO COM O MÓDULO SNIFFER FINAL

Após feita a unificação dos dois primeiros módulos resultando em um aplicativo funcional que proporcionava ao usuário a capacidade de escanear as redes WiFi locais e selecionar uma para copiar as informações e criar um *hotspot* com tais, restou apenas unir esse aplicativo ao aplicativo implementado na etapa 3.

Como os aplicativos foram desenvolvidos de forma independente, a união foi feita de tal forma que o projeto do aplicativo resultante da unificação dos dois primeiros módulos foi inteiramente incluído no projeto do aplicativo de *sniffing* desenvolvido na etapa 3. A inclusão foi realizada copiando todo código do projeto para um novo pacote no projeto da etapa 3, importando o projeto como se fosse uma biblioteca e executando toda a configuração através do Gradle.

O resultado final constitui então de um aplicativo com todas as funcionalidades requisitadas no projeto.

4.5 DIFICULDADES GERAIS ENCONTRADAS

Mesmo havendo materiais na comunidade com exemplos relacionados ao tema e às funcionalidades desejadas, principalmente para os dois primeiros módulos, foram observadas algumas adversidades durante a construção do aplicativo.

Durante o desenvolvimento do aplicativo a versão do sistema operacional do dispositivo utilizado foi atualizada para Android 6.0. Essa atualização automática do *software*

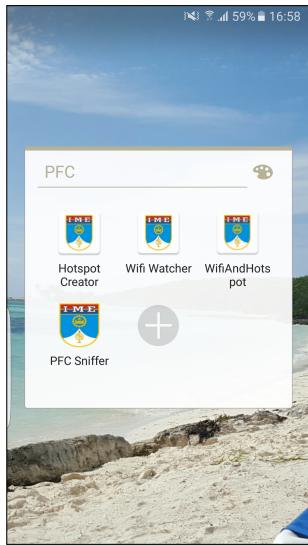


FIG. 4.7: Todos os Módulos

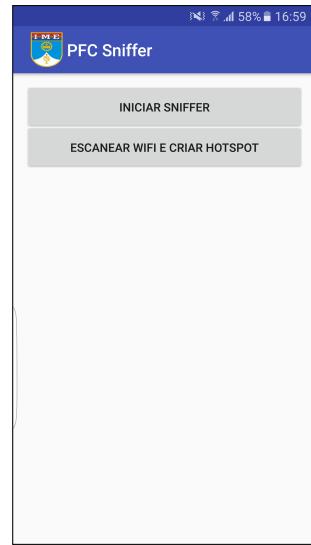


FIG. 4.8: Tela inicial da Versão Final

fez com que o aplicativo WiFi Watcher e Hotspot Creator parassem de funcionar. Isso ocorreu porque a nova versão do sistema operacional exigia que permissões, que antes não eram necessárias, fossem declaradas e tivessem tratamento especial. Essas permissões, chamadas permissões de tempo de execução (DEVELOPERS, 2016a), foram apresentadas somente na atualização Android 6.0. Marshmallow.

Como esses problemas eram relativamente novos, pouca documentação na comunidade era acessível, mas posteriormente foi encontrada a solução. Para o aplicativo WiFi Watcher foram necessárias as permissões ACCESS_COARSE_LOCATION e ACCESS_FINE_LOCATION, sendo que a primeira necessita que seja requisitada para o usuário através da função “*requestPermission()*” e manipulada pela função “*onRequestPermissionsResult*” que é invocada após a resposta do usuário.

Para a resolução da permissão do Hotspot Creator uma permissão especial era necessária, WRITE_SETTINGS, que não funciona como permissões normais ou do tipo *dangerous*, essa permissão junto com SYSTEM_ALERT_WINDOW são particularmente sensíveis e devem sempre ser evitadas. Logo, além de terem que ser declaradas no “*manifest.xml*” do aplicativo, devem também invocar um objeto Intent requisitando autorização do usuário. O sistema responde ao objeto Intent, mostrando uma tela de gerenciamento detalhado para o usuário (DEVELOPER, 2016c).

Por fim, no terceiro módulo, foram encontradas dificuldades na leitura dos pacotes devido à captura por fluxo de bytes. Apenas o fluxo de bytes pode ser exibido na tela, portanto não houve como identificar informações claras de pacotes, como Ip de origem e destino. Contudo foi possível identificar parcialmente informações dos pacotes quando se

abriu o arquivo de texto.

Mesmo com a dificuldade de interpretação das informações presentes nos pacotes capturados pelo *Sniffer*, ainda foi possível identificar informações de aplicativos e protocolos utilizados em partes dos bytes, que apareciam em forma de texto claro, capturados e salvos nos arquivos de texto na memória interna.

5 CONCLUSÃO

Este relatório buscou apresentar o projeto e construção de um aplicativo de *sniffer* de redes, juntamente com conceitos iniciais da plataforma Android e de Segurança da Informação, com enfoque em captura de pacotes de redes. O projeto foi realizado seguindo uma estrutura de desenvolvimento definida por módulos, mantendo uma implementação por etapas bem definida do produto final.

Como resultado final deste Projeto de Final de Curso foi implementado o aplicativo "PFC Sniffer", aplicativo completo que permite a criação de um ponto de acesso para se passar por uma rede WiFi já existente e realizar a captura de pacotes dos usuários que se conectarem a esse *Hotspot*. Como sugestões de trabalhos futuros ou continuação desse projeto, é sugerido um melhor tratamento dos pacotes capturados para que seja possível a visualização de informações com IP de origem e saída de cada pacote, um possível filtro de pacotes baseados no IP de origem ou de acesso e outras melhorias na estabilidade do VPN criado.

Existe também a possibilidade de criação do Hotspot sem que seja uma rede totalmente aberta, com a requisição de uma senha para o usuário atacado que deseje se conectar, porém aceitando qualquer entrada no campo. Com esse novo recurso o aplicativo estaria criando um ponto de acesso de maior similaridade com a rede tomada como referência, que possui senha, e também uma possível descoberta da senha dessa rede, considerando a negligência de um usuário usual da rede que estaria passando para o aplicativo a senha correta que seria capturada em texto simples.

6 REFERÊNCIAS BIBLIOGRÁFICAS

- TAOSOFTWARE CO. tPacketCapture. Disponível em: <<https://play.google.com/store/apps/details?id=jp.co.taosoftware.android.packetcapture>>. Acesso em: 23 de julho de 2016.
- ANDROID DEVELOPER. Fundamentos de Aplicativos. Disponível em: <<https://developer.android.com/intl/pt-br/guide/components/fundamentals.html>>. Acesso em: 10 maio de 2016.
- ANDROID DEVELOPER. Fundamentos de Aplicativos. Disponível em: <<https://developer.android.com/reference/android/net/wifi/package-summary.html>>. Acesso em: 23 de julho de 2016.
- ANDROID DEVELOPER. System Permissions. Disponível em: <<https://developer.android.com/guide/topics/security/permissions.html>>. Acesso em: 8 de setembro de 2016.
- ANDROID DEVELOPERS. RUN TIME PERMISSIONS. Disponível em: <<https://developer.android.com/training/permissions/requesting.html>>. Acesso em: 19 de setembro de 2016.
- ANDROID DEVELOPERS. Wifi API. Disponível em: <<https://developer.android.com/reference/android/net/wifi/package-summary.html>>. Acesso em: 19 de setembro de 2016.
- ANDROID DEVELOPERS. Wifi Manager. Disponível em: <<https://developer.android.com/reference/android/net/wifi/WifiManager.html>>. Acesso em: 19 de setembro de 2016.
- KAITOY. Pcap4j. Disponível em: <<https://github.com/kaitoy/pcap4j>>. Acesso em: 23 de julho de 2016.
- KEVIN, J. C. **Law of Internet Security and Privacy**. [S.l.]: Aspen Publisher, 2003.
- BRADLEY MITCHELL. Sniffer:What is a sniffer in computer network?. Disponível em: <http://compnetworking.about.com/od/networksecurityprivacy/g/bldef_sniffer.htm>. Acesso em: 10 de maio de 2016.

MOHAMED NAUFAL. LocalVPN. Disponível em: <<https://github.com/hexene/LocalVPN>>. Acesso em: 15 de setembro de 2016.

OFFICIAL ANDROID BLOG - NEWS AND NOTES FROM ANDROID TEAM. Get ready for the sweet taste of Android 6.0 Marshmallow. Disponível em: <<https://android.googleblog.com/2015/10/get-ready-for-sweet-taste-of-android-60.html>>. Acesso em: 19 de setembro de 2016.

NEHA SHARMA. The Beginner's Guide to Android: Android Architecture. Disponível em: <<http://www.edureka.co/blog/beginners-guide-android-architecture/>>. Acesso em: 9 maio de 2016.

GREY SHIRTS. PacketCapture. Disponível em: <<https://play.google.com/store/apps/details?id=app.greyshirts.sslcapture>>. Acesso em: 23 de julho de 2016.

SINGH, S. **The Code Book: How to make it, break it, hack it, crack it.** [S.l.]: Delacorte, 2002.

STALLINGS, W. **Cryptography and Network Security: Principles and Practice.** 6. ed. [S.l.]: Pearson, 2014.

INTERNATIONAL TELECOMMUNICATION UNION - TELECOMMUNICATION. Computer Security The OSI security architecture. Disponível em: <https://en.wikibooks.org/wiki/Computer_Security/The_OSI_security_architecture>. Acesso em: 10 maio de 2016.

WIFI ALIANCE. WiFi Aliance. Disponível em: <<http://www.wi-fi.org>>. Acesso em: 22 de julho de 2016.

7 APÊNDICES

APÊNDICE 1: CÓDIGO DAS CLASSES PRINCIPAIS

7.1.1 CLASSE WIFI WATCHER

```
public class MainActivity_WifiWatcher extends Activity {  
    Switch aSwitch;  
    WifiManager wifiManager;  
    Button btn;  
    MyBroadCastReceiver myBroadCastReceiver = new MyBroadCastReceiver();  
    StringBuilder sb = new StringBuilder();  
    private static final String ROW_ID = "row_id";  
    private List<ScanResult> wifiList;  
    private ListView wifiListView;  
    private ArrayList<String> wifiArray = new ArrayList<String>();  
    private ArrayAdapter wifiArrayAdapter;  
    private static final int PERMISSIONS_REQUEST_CODE_ACCESS_COARSE_LOCATION =  
    1001;  
    /**  
     * ATTENTION: This was auto-generated to implement the App Indexing API.  
     * See https://g.co/AppIndexing/AndroidStudio for more information.  
     */  
    private GoogleApiClient client;  
  
    @TargetApi(Build.VERSION_CODES.M)  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
  
        if(!Settings.System.canWrite(this))  
        {  
            Intent intent = new Intent(Settings.ACTION_MANAGE_WRITE_SETTINGS);  
            startActivity(intent);  
        }  
    }  
}
```

```

        Log.i("NetworkUtil", "PERMISSION TO WRITE_SETTINGS : " +
        Settings.System.canWrite(this));

    }

    setContentView(R.layout.activity_main_activity__wifi_watcher);
    wifiListView = (ListView) findViewById(R.id.wifi_list_view);
    wifiListView.setOnItemClickListener(viewWifiListener);

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
        if (checkSelfPermission(Manifest.permission.ACCESS_COARSE_LOCATION)
        != PackageManager.PERMISSION_GRANTED) {
            if (Build.VERSION.SDK_INT >= M) {
                requestPermissions(new String[]
                {Manifest.permission.ACCESS_COARSE_LOCATION},
                PERMISSIONS_REQUEST_CODE_ACCESS_COARSE_LOCATION);
            }
            //After this point you wait for callback in
            onRequestPermissionsResult(int, String[], int[])
            overriden
            method
        }
    }

    wifiManager = (WifiManager) getSystemService(WIFI_SERVICE);
    aSwitch = (Switch) findViewById(R.id.myswitch);
    btn = (Button) findViewById(R.id.btn);

    //register the switch for event handling
    aSwitch.setOnCheckedChangeListener(new
    CompoundButton.OnCheckedChangeListener() {
        @Override
        public void onCheckedChanged(CompoundButton compoundButton, boolean
        isChecked) {
            if (isChecked && !wifiManager.isWifiEnabled()) {

```

```

        //to switch on WiFi
        wifiManager.setWifiEnabled(true);
    }

    //to switch off WiFi
    else if (!isChecked && wifiManager.isWifiEnabled()) {
        wifiManager.setWifiEnabled(false);
    }

}

});

//



btn.setOnClickListener(new View.OnClickListener() { //melhorar
verificação
com o swtich do wifi
@Override
//On click function
public void onClick(View view) {
    if (!wifiManager.isWifiEnabled()) {
        Toast.makeText(getApplicationContext(), "Turn On Wifi",
        Toast.LENGTH_SHORT).show();
    } else {
        wifiManager.startScan();
    }
}
});



//register the broadcast receiver
// Broacast receiver will automatically call when number of wifi
connections
changed
registerReceiver(myBroadCastReceiver, new
IntentFilter(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION));





// ATTENTION: This was auto-generated to implement the App Indexing
API.

```

```

// See https://g.co/AppIndexing/AndroidStudio for more information.
client = new
GoogleApiClient.Builder(this).addApi(AppIndex.API).build();
}

@Override
public void onRequestPermissionsResult(int requestCode, String[]
permissions,
int[] grantResults) {
    if (requestCode == PERMISSIONS_REQUEST_CODE_ACCESS_COARSE_LOCATION
        && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
        // TODO: What you want to do when it works or maybe
        .PERMISSION_DENIED
        if it works better
    }
}

private class MyBroadCastReceiver extends BroadcastReceiver
{

    @Override
    public void onReceive(Context context, Intent intent) {
        wifiList = wifiManager.getScanResults();

        sb.append("\n Number of WiFi connections:" + wifiList.size() +
        "\n\n");
        wifiArray.add(sb.toString());
        sb.setLength(0);
        for (int i = 0; i < wifiList.size(); i++) {
            sb.append(new Integer(i + 1).toString() + ".");
            sb.append("SSID: ");
            sb.append((wifiList.get(i)).SSID.toString());
            sb.append("\n");
            sb.append("BSSID: ");
        }
    }
}

```

```

        sb.append((wifiList.get(i)).BSSID.toString());
        sb.append("\n\n");
        wifiArray.add(sb.toString());
        sb.setLength(0);
    }

    wifiArrayAdapter = new ArrayAdapter(MainActivity_WifiWatcher.this,
        R.layout.wifi_list_item, R.id.wifi_list_item_text_view, wifiArray);
    wifiListView.setAdapter(wifiArrayAdapter);
}

}

ListView.OnItemClickListener viewWifiListener = new
AdapterView.OnItemClickListener()
{
    @Override
    public void onItemClick(AdapterView<?> adapterView, View view, int i,
    long l)
    {
        Intent viewWiFi = new Intent(MainActivity_WifiWatcher.this,
SecondActivity_HotspotCreator.class);

        viewWiFi.putExtra(ROW_ID,l);
        viewWiFi.putExtra("network_ssid",wifiList.get((int) (l-1)).SSID);
        viewWiFi.putExtra("network_bssid",wifiList.get((int) (l-1)).BSSID);
        viewWiFi.putExtra("network_capabilities",wifiList.get((int) (l-
1)).capabilities);
        startActivity(viewWiFi);
    }
};

}//END OF CLASS WIFI WATCHER

```

7.1.2 CLASSE HOTSPOT CREATOR

```
public class SecondActivity_HotspotCreator extends Activity {
```

```

/**
 * ATTENTION: This was auto-generated to implement the App Indexing API.
 * See https://g.co/AppIndexing/AndroidStudio for more information.
 */

private GoogleApiClient client;
private long rowID;
private TextView ssidTextView;
private TextView bssidTextView;
private TextView smallTextView;
private TextView capabilitiesTextView;
private Button createHotpotButton;
private WifiManager wifiManager;
private String ssidString;
private String bssidString;
private String capabilitiesString;

@TargetApi(Build.VERSION_CODES.M)
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_second_hotspot_creator);

    ssidTextView = (TextView) findViewById(R.id ssid_textView);
    bssidTextView= (TextView) findViewById(R.id bssid_textView);
    smallTextView= (TextView) findViewById(R.id small_textView);
    capabilitiesTextView = (TextView)
        findViewById(R.id capabilities_textView);
    createHotpotButton= (Button) findViewById(R.id hotspot_button);

    Bundle extras=getIntent().getExtras();
    rowID= extras.getLong("row_id");
    ssidString=extras.getString("network_ssid");
    bssidString=extras.getString("network_bssid");
    capabilitiesString=extras.getString("network_capabilities");
}

```

```

ssidTextView.setText("Network SSID: "+ssidString+"\n");
bssidTextView.setText("Network BSSID: "+bssidString+"\n");
capabilitiesTextView.setText("Network capabilities:
"+capabilitiesString+"\n");
Log.i("NetworkUtil","VALUE OF ROW_ID : "+ rowID);

createHotpotButton.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View view)
    {
        Toast.makeText(getApplicationContext(), "CRIANDO HOTSPOT...",
        Toast.LENGTH_SHORT).show();
        wifiManager =
        (WifiManager) getSystemService(Context.WIFI_SERVICE);
        createWifiAccessPoint();

        setMobileDataEnabled (true);
        Toast.makeText(getApplicationContext(), "Hotspot Criado",
        Toast.LENGTH_SHORT).show();
        smallTextView.setText("HOTSPOT CRIADO");
    }
});

// ATTENTION: This was auto-generated to implement the App Indexing
// API.
// See https://g.co/AppIndexing/AndroidStudio for more information.
client = new
GoogleApiClient.Builder(this).addApi(AppIndex.API).build();
}

private void createWifiAccessPoint()

```

```

{
    if (wifiManager.isWifiEnabled())
    {
        wifiManager.setWifiEnabled(false);
    }

    Method[] wmMethods = wifiManager.getClass().getDeclaredMethods();

    boolean methodFound = false;
    for (Method method : wmMethods)
    {
        if (method.getName().equals("setWifiApEnabled"))
        {
            methodFound = true;
            WifiConfiguration netConfig = new WifiConfiguration();
            netConfig.SSID = ssidString;
            netConfig.BSSID= bssidString;
            netConfig.hiddenSSID = false;
            netConfig.allowedAuthAlgorithms.set(WifiConfiguration.AuthAlgorithm.OPEN);

            try
            {
                boolean apstatus = (Boolean) method.invoke(
                    wifiManager, netConfig, true);
                for (Method isWifiApEnabledmethod : wmMethods)
                {
                    if (isWifiApEnabledmethod.getName().equals(
                        "isWifiApEnabled")) {
                        while (!(Boolean) isWifiApEnabledmethod.invoke(
                            wifiManager)) {
                            }
                        ;
                    for (Method method1 : wmMethods)
                    {
                        if (method1.getName().equals(
                            "getWifiApState")) {

```

```

        }
    }
}
}

if (apstatus)
{
    Log.d("Splash Activity",
          "Access Point created");
}
else
{
    Log.d("Splash Activity",
          "Access Point creation failed");
}

} catch (IllegalArgumentException e) {
    e.printStackTrace();
} catch (IllegalAccessException e) {
    e.printStackTrace();
} catch (InvocationTargetException e) {
    e.printStackTrace();
}
}

if (!methodFound) {
    Log.d("Splash Activity", "cannot configure an access point");
}
}

/**
 * Switching On data
 */
@SuppressLint({"rawtypes", "unchecked"})
private void setMobileDataEnabled (boolean enabled){
    Log.i("NetworkUtil", "Mobile data enabling: " + enabled);
    final ConnectivityManager conman =

```

```

(ConnectivityManager)getSystemService(Context.CONNECTIVITY_SERVICE);

try {
    Class conmanClass = Class.forName(conman.getClass().getName());
    final Field connectivityManagerField =
    conmanClass.getDeclaredField("mService");
    connectivityManagerField.setAccessible(true);
    final Object connectivityManager =
    connectivityManagerField.get(conman);
    final Class connectivityManagerClass =
    Class.forName(connectivityManager.getClass().getName());

    final Method setMobileDataEnabledMethod =
    connectivityManagerClass.getDeclaredMethod("setMobileDataEnabled",
    Boolean.TYPE);
    setMobileDataEnabledMethod.setAccessible(true);
    setMobileDataEnabledMethod.invoke(connectivityManager, enabled);
} catch (Exception e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
}

}//END OF CLASS HOTSPOT CREATOR

```

7.1.3 CLASSE LOCAL VPN SERVICE

```

public class LocalVPNService extends VpnService
{
    private static final String TAG = LocalVPNService.class.getSimpleName();
    private static final String VPN_ADDRESS = "10.0.0.2"; // Only IPv4 support
    for now
    private static final String VPN_ROUTE = "0.0.0.0"; // Intercept everything
    public static final String BROADCAST_VPN_STATE =
    "xyz.hexene.localvpn.VPN_STATE";

```

```

private static boolean isRunning = false;
private ParcelFileDescriptor vpnInterface = null;
private PendingIntent pendingIntent;
private final IBinder mIBinder = new LocalBinder();
private static Handler mHandler;

public class LocalBinder extends Binder
{
    public LocalVPNService getInstance()
    {
        return LocalVPNService.this;
    }
}

private static void saveData(byte[] data, String filename){
    File path =Environment.getExternalStoragePublicDirectory
    (Environment.DIRECTORY_DOWNLOADS);
    File file = new File(path, filename);
    try {
        FileOutputStream stream = new FileOutputStream(file, true);
        stream.write(data);
        stream.close();
        Log.i("saveData", "Data Saved");
    } catch (IOException e) {
        Log.e("SAVE DATA", "Could not write file " + e.getMessage());
    }
}

private ConcurrentLinkedQueue<Packet> deviceToNetworkUDPQueue;
private ConcurrentLinkedQueue<Packet> deviceToNetworkTCPQueue;
private ConcurrentLinkedQueue<ByteBuffer> networkToDeviceQueue;
private ExecutorService executorService;

private Selector udpSelector;
private Selector tcpSelector;

```

```

@Override
public void onCreate()
{
    super.onCreate();
    isRunning = true;
    setupVPN();
    try
    {
        udpSelector = Selector.open();
        tcpSelector = Selector.open();
        deviceToNetworkUDPQueue = new ConcurrentLinkedQueue<>();
        deviceToNetworkTCPQueue = new ConcurrentLinkedQueue<>();
        networkToDeviceQueue = new ConcurrentLinkedQueue<>();

        executorService = Executors.newFixedThreadPool(5);
        executorService.submit(new UDPInput(networkToDeviceQueue,
        udpSelector));
        executorService.submit(new UDPOutput(deviceToNetworkUDPQueue,
        udpSelector, this));
        executorService.submit(new TCPIInput(networkToDeviceQueue,
        tcpSelector));
        executorService.submit(new TCPOutput(deviceToNetworkTCPQueue,
        networkToDeviceQueue, tcpSelector, this));
        executorService.submit(new
        VPNRunnable(vpnInterface.getFileDescriptor(),
                    deviceToNetworkUDPQueue, deviceToNetworkTCPQueue,
                    networkToDeviceQueue));
        LocalBroadcastManager.getInstance(this).sendBroadcast(new
        Intent(BROADCAST_VPN_STATE).putExtra("running", true));
        Log.i(TAG, "Started");
    }
    catch (IOException e)
    {
        // TODO: Here and elsewhere, we should explicitly notify the user
        // of any errors
    }
}

```

```

        // and suggest that they stop the service, since we can't do it
        // ourselves
        Log.e(TAG, "Error starting service", e);
        cleanup();
    }

}

private void setupVPN()
{
    if (vpnInterface == null)
    {
        Builder builder = new Builder();
        builder.addAddress(VPN_ADDRESS, 32);
        builder.addRoute(VPN_ROUTE, 0);
        vpnInterface =builder.setSession(getString(R.string.app_name)).
        setConfigureIntent(pendingIntent).establish();
    }
}

@Override
public IBinder onBind(Intent intent)
{
    Log.d(TAG, "onBind");
    return mIBinder;
}

@Override
public int onStartCommand(Intent intent, int flags, int startId)
{
    return START_STICKY;
}

public static boolean isRunning()
{
    return isRunning;
}

```

```

    }

    public void setHandler(Handler handler)
    {
        Log.d(TAG, "Setting handler...");
        mHandler = handler;
    }

    @Override
    public void onDestroy()
    {
        super.onDestroy();
        isRunning = false;
        executorService.shutdownNow();
        cleanup();
        Log.i(TAG, "Stopped");
    }

    private void cleanup()
    {
        deviceToNetworkTCPQueue = null;
        deviceToNetworkUDPQueue = null;
        networkToDeviceQueue = null;
        ByteBufferPool.clear();
        closeResources(udpSelector, tcpSelector, vpnInterface);
    }

    // TODO: Move this to a "utils" class for reuse
    private static void closeResources(Closeable... resources)
    {
        for (Closeable resource : resources)
        {
            try
            {
                resource.close();
            }
        }
    }
}

```

```

        catch (IOException e)
        {
            // Ignore
        }
    }

private static class VPNRunnable implements Runnable
{
    private static final String TAG = VPNRunnable.class.getSimpleName();
    private FileDescriptor vpnFileDescriptor;
    private ConcurrentLinkedQueue<Packet> deviceToNetworkUDPQueue;
    private ConcurrentLinkedQueue<Packet> deviceToNetworkTCPQueue;
    private ConcurrentLinkedQueue<ByteBuffer> networkToDeviceQueue;
    private Message msg;

    public VPNRunnable(FileDescriptor vpnFileDescriptor,
                       ConcurrentLinkedQueue<Packet>
                           deviceToNetworkUDPQueue,
                       ConcurrentLinkedQueue<Packet>
                           deviceToNetworkTCPQueue,
                       ConcurrentLinkedQueue<ByteBuffer>
                           networkToDeviceQueue)
    {
        this.vpnFileDescriptor = vpnFileDescriptor;
        this.deviceToNetworkUDPQueue = deviceToNetworkUDPQueue;
        this.deviceToNetworkTCPQueue = deviceToNetworkTCPQueue;
        this.networkToDeviceQueue = networkToDeviceQueue;
    }

    @Override
    public void run()
    {
        Log.i(TAG, "Started");
    }
}

```

```

FileChannel vpnInput = new
FileInputStream(vpnFileDescriptor).getChannel();
FileChannel vpnOutput = new
FileOutputStream(vpnFileDescriptor).getChannel();

try
{
    ByteBuffer bufferToNetwork = null;
    boolean dataSent = true;
    boolean dataReceived;
    while (!Thread.interrupted())
    {
        if (dataSent)
            bufferToNetwork = ByteBufferPool.acquire();
        else
            bufferToNetwork.clear();
        // TODO: Block when not connected
        int readBytes = vpnInput.read(bufferToNetwork);
        if (readBytes > 0)
        {
            dataSent = true;
            bufferToNetwork.flip();
            ByteBuffer bufferCopy = bufferToNetwork.duplicate();
            byte[] packetData = new byte[bufferCopy.limit()];
            bufferCopy.get(packetData);
            Log.d(TAG, "PACKAGE DATA SENDING: " + packetData);
            msg =
mHandler.obtainMessage(LocalVPN.PACKET_MESSAGE_SENDING,-1,-1,packetData);
//                      stream.write(packetData);
            Log.d(TAG, "Mandando para o target");
            msg.sendToTarget();
            saveData(packetData, "SendingPackets.txt"); //Salva o
pacote no arquivo em memória
            Log.d(TAG, "Salvou os pacotes enviados no arquivo em
memória");
    }
}

```

```

//SimpleDateFormat dateFormat = new
SimpleDateFormat("dd-MM-yy HH-mm-ss");
//Date date = new Date();
// new BufferedWriter(new
FileWriter(dateFormat.format(date) +
"SendingPackets.txt", true));

Log.d(TAG, "Salvou os pacotes recebidos no arquivo na
memória o pacote ");
Packet packet = new Packet(bufferToNetwork);
if (packet.isUDP())
{
    deviceToNetworkUDPQueue.offer(packet);
}
else if (packet.isTCP())
{
    deviceToNetworkTCPQueue.offer(packet);
}
else
{
    Log.w(TAG, "Unknown packet type");
    Log.w(TAG, packet.ip4Header.toString());
    dataSent = false;
}
else
{
    dataSent = false;
}
ByteBuffer bufferFromNetwork =
networkToDeviceQueue.poll();
if (bufferFromNetwork != null)
{
    bufferFromNetwork.flip();
    ByteBuffer bufferCopy = bufferFromNetwork.duplicate();

```

```

        byte[] packetData = new byte[bufferCopy.limit()];
        bufferCopy.get(packetData);
        Log.d(TAG, "PACKAGE DATA COMING: " + packetData);
        msg =
mHandler.obtainMessage(LocalVPN.PACKET_MESSAGE_COMING, -1,-1,packetData);
//                      stream.write(packetData);
        Log.d(TAG, "Enviando para o target");
        saveData(packetData, "ReceivingPackets.txt"); //Salva
o pacote no arquivo em memória
//SimpleDateFormat dateFormat = new
SimpleDateFormat("dd-MM-yy HH-mm-ss");
//Date date = new Date();
//new BufferedWriter(new
FileWriter(dateFormat.format(date) +
"ReceivingPackets.txt", true));
        Log.d(TAG, "Salvou os pacotes recebidos no arquivo na
memória o pacote ");
        msg.sendToTarget();
        while (bufferFromNetwork.hasRemaining())
            vpnOutput.write(bufferFromNetwork);
        dataReceived = true;

        ByteBufferPool.release(bufferFromNetwork);
    }
else
{
    dataReceived = false;
}
// TODO: Sleep-looping is not very battery-friendly,
consider blocking instead
// Confirm if throughput with ConcurrentQueue is really
higher compared to BlockingQueue
if (!dataSent && !dataReceived)
    Thread.sleep(10);
}

```

```
    }

    catch (InterruptedException e)
    {
        Log.i(TAG, "Stopping");
    }

    catch (IOException e)
    {
        Log.w(TAG, e.toString(), e);
    }

    catch (Exception e){
        Log.e(TAG, e.toString(), e);
    }

    finally
    {
        closeResources(vpnInput, vpnOutput);
    }
}

}

}

} \\END OF CLASS LOCAL VPN SERVICE
```