UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Antonio Carlos Campos da Silva Junior

Aplicativo Mirai Scanner: modificações e uma análise preliminar das varreduras

Uberlândia, Brasil

2019

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Antonio Carlos Campos da Silva Junior

Aplicativo Mirai Scanner: modificações e uma análise preliminar das varreduras

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Rodrigo Sanches Miani

Universidade Federal de Uberlândia – UFU

Faculdade de Ciência da Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2019

Dedico essa vitória a todos meus amigos e familiares, em especial aos meus pais, que sempre me incentivaram e me apoiaram em todos os momentos da minha caminhada, que por meio de seus ensinamentos me passaram valores os quais levarei sempre comigo, me ensinaram a importância da educação, estudo e aprendizado no crescimento pessoal e profissional de qualquer pessoa. Foram minha base.

AGRADECIMENTOS

Primeiramente gostaria de agradecer a Deus por me permitir levantar todos os dias e ir atrás dos meus objetivos.

Feito isso gostaria de agradecer a todos os professores que me ajudaram durante a graduação, sendo por meio de uma aula, de uma monitoria ou um horário de atendimento.

Não poderei me esquecer dos meus colegas de turma e além destes meus colegas de graduação que de alguma forma contribuíram na minha jornada.

Entre meus colegas de curso alguns se destacaram, e se tornaram amigos. e não poderia deixar de mencioná los de forma individual, sendo eles: Neliton Rafael, Henrique Araújo, Felipe Gomes, Patrick Finzi, Lorena Rodrigues, Marcelo Filho e Maxwell Oliveira.

Sempre gosto de lembrar de todos meus amigos de infância que sempre estiveram comigo mesmo antes da faculdade, e da minha namorada Tayna Tamires, que esteve ao meu lado em momentos ruins e bons me apoio, incentivou e ajudou em tudo que estivesse ao seu alcance.

Aos meus pais e irmãos o sentimento é gratidão.

"O sucesso nasce do querer, da determinação e persistência em se chegar a um objetivo. Mesmo não atingindo o alvo, quem busca e vence obstáculos, no mínimo fará coisas admiráveis." – José de Alencar

RESUMO

O desenvolvimento das redes de computadores acarretou diversas possibilidades de conexões entre distintos dispositivos, uma delas envolve o paradigma da Internet das Coisas. A partir do desenvolvimento da Internet das Coisas, concomitantemente, houve a progressão de *botnets*, um conjunto de dispositivos infectados por um *malware*. A crescente interação humano máquina possibilitou novas abordagens no uso de ações maliciosas que utilizam as *botnet*. Assim, esse trabalho objetiva o levantamento de informações sobre vulnerabilidades existentes em dispositivos IoT conectados à rede sem fio. Para isso, foram realizadas modificações no aplicativo Mirai Scanner com o intuito de registrar as varreduras dos usuários em um banco de dados remoto e permitir a análise das vulnerabilidades encontradas nos dispositivos verificados. Os resultados demonstraram a confirmação da existência de vulnerabilidades desconhecidas entre os usuários finais, além da robustez do sistema. A partir disso, os resultados encontrados são benéficos para o desenvolvimento de possíveis meios de conscientização do uso de dispositivos IoT.

Palavras-Chave: Segurança da informação, Vulnerabilidades, *Botnet*, Internet das Coisas, *Malware*, Mirai.

LISTA DE ILUSTRAÇÕES

Figura 1 – Estrutura de uma *botnet*. Extraído de Prado (2018) 16
Figura 2 – Registro de ataques usando Mirai no mundo todo. Extraído de Farinaccio, Rafael (2017) 19
Figura 3 – Comunicação e operação da botnet Mirai. Extraído de Kolias et al. (2017a). 21
Figura 4 – Tela inicial após execução do aplicativo. 25
Figura 5 – Lista de dispositivos encontrados na rede em que o aplicativo se encontra conectado. 26
Figura 6 – Dispositivo sem vulnerabilidades. 27
Figura 7 – Dispositivo sem vulnerável. 28
Figura 8 – Aplicativo antes das modificações 28
Figura 9 – Aplicativo depois das modificações 29
Figura 10 – Gráfico que apresenta os resultados coletados 37
Figura 11 – Tabela com os dados dos dispositivos com vulnerabilidades 38
Figura 12 – Número de instalações do dispositivo 39
Figura 13 – Número de desinstalações do dispositivo 39
Figura 14 – Avaliação do dispositivo 40

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Language
CCTV	Closed-Circuit Television Camera
C&C	Command and Control
DVR	Digital Video Recorder
DDoS	Distributed Denial of Service
DNS	Domain Name System
DoS	Denial of Service
FTP	File Transfer Protocol
GRE	Generic Routing Encapsulation
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IoT	Internet of Things
IP	Internet Protocol
P2P	Peer-to-Peer
SSH	Secure Shell
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
JSON	JavaScript Object Notation
WSNs	Wireless Sensor Networks
M2M	Machine to Machine
CPS	CyberPhysical Systems
IRC	Internet Relay Chat

SUMÁRIO

1 INTRODUÇÃO . . . 10

2 REVISÃO BIBLIOGRÁFICA . . . 13
2.1 Segurança da Informação . . . 13
2.2 IoT . . . 14
2.3 *Botnets* . . . 15
2.3.1 Mirai . . . 18
2.3.1.1 Histórico . . . 18
2.3.1.2 Funcionamento . . . 19
2.4 Trabalhos correlatos . . . 21

3 DESENVOLVIMENTO . . . 24
3.1 Mirai Scanner . . . 24
3.2 Modificações no Mirai Scanner . . . 26
3.3 Construção da API . . . 27
3.4 Adaptando o aplicativo para se comunicar com a API . . . 32
3.4.1 Criação do objeto *Post* . . . 32
3.4.2 Criação da Instância de *Retrofit* . . . 33
3.4.3 Criação da interface da API . . . 33
3.4.4 Criação do *ApiUtils* . . . 34
3.4.5 Execução da solicitação *POST* . . . 34

4 RESULTADOS . . . 36
4.1 Google Play . . . 36
4.2 Análise das varreduras . . . 37

5 CONCLUSÃO . . . 41

REFERÊNCIAS . . . 43

ANEXOS 45

ANEXO A – CÓDIGO FONTE DA API . . . 46
A.1 Produto . . . 46
A.2 application.properties . . . 49
A.3 ProdutoRepository Interface . . . 49

A.4 ProdutoRepository 49

ANEXO B – CÓDIGO FONTE DAS CLASSES ADICIONADAS/-MODIFICADAS DO APLICATIVO MIRAI SCANNER 52

B.1 Post - Adicionada 52

B.2 RetrofitClient - Adicionada 55

B.3 ApiUtils - Adicioanda 56

B.4 Interface APIService - Adicionada 56

B.5 DispositivoActivity - Modificada 57

1 INTRODUÇÃO

Segundo França et al. (2011), há um grande progresso na área de dispositivos embarcados. Esses objetos físicos podem ser diversos, como eletrodomésticos, máquinas, sensores entre outros, e todos podem se conectar à Internet. A partir disso, em um futuro próximo, muitos aparelhos do dia a dia poderão estar conectados diretamente a Internet. A conexão desses inúmeros objetos cotidianos com a Internet é conhecida como Internet das Coisas (do inglês *Internet of Things* – IoT).

A utilização da Internet das Coisas está modificando a forma como nós, humanos, nos relacionamos com os objetos ao nosso redor, a partir disso há mudanças nos setores de segurança, energia, meio ambiente e outros (OLIVEIRA, 2017).

Além disso, por volta de 1999, a partir da popularização da Internet já se pensava sobre as possibilidades de conexão de itens utilizados no dia a dia com a Internet (OLIVEIRA, 2017). Foi o desenvolvimento de tecnologias nos últimos anos que tornaram a comunicação entre coisas e Internet possível, assim os dispositivos necessários para a conexão dos objetos adquiriram um preço compatível com os dispositivos no qual são conectados, e em alguns anos, será realidade em todo o mundo (OLIVEIRA, 2017).

De acordo com o Bohrer, Magalhães e Rocha (2014) houve um crescimento no número de usuários da Internet logo após a criação da Web, entre 2000 e 2011 a rede cresceu 528%. Segundo Monteiro (2001) a Internet está inserida em 150 países e reúne cerca de 300 milhões de computadores e mais de 400 milhões de usuários.

O avanço da Internet pode ser considerado de extrema relevância para a sociedade, porém, juntamente com esse avanço, os riscos de invasão de privacidade também aumentam, uma vez que dados podem ser coletados sem que as pessoas percebam que isso está ocorrendo (BARBOSA et al., 2014).

Essa crescente no número de aparelhos conectados proporciona um aumento significativo no tráfego de informações, e boa parte desses dados podem ser considerados como valiosos. Há uma grande quantidade de dados sigilosos que são armazenados em computadores, como transações bancárias, dados pessoais, senhas, segredos industriais e outros. Considerando o valor dessas informações que permeiam essa tecnologia faz se necessário um estudo sobre tais fatores, estudo esse que seja capaz de proporcionar uma análise sobre esse fenômeno e os seus possíveis riscos.

A segurança da informação tem o papel de garantir que nenhum dos três pilares confidencialidade, integridade e disponibilidade sejam afetados por nenhuma ação maliciosa (STALLINGS, 2011). Diante disso, é importante que as organizações provedoras de

tecnologia IoT possuam atenção especial no desenvolvimento da segurança dos dispositivos. O aperfeiçoamento de mecanismos que contribuem com essa perspectiva, assim como análises e pesquisas que auxiliem em um melhor entendimento de possíveis vulnerabilidades são de extrema importância.

Um exemplo que ameaça a segurança da informação são os códigos maliciosos ou *malwares* como vírus ou *worms*. Um *malware* pode comprometer o funcionamento de uma instituição e estar relacionado à várias vertentes, como invasões, negação de serviço, contaminações de sistemas, além de furtos de informações e outras atividades sem a devida permissão, que podem ser consideradas ilegais ou ilegítimas.

De acordo com Melo et al. (2011), os códigos maliciosos têm causado irregularidades e desordem em sistemas, e para a correção desses danos é necessário um grande empenho da instituição retentora da informação. O conhecimento sobre como se dá o funcionamento de um código malicioso é muito importante, uma vez que é um meio de desenvolvimento de produção de possíveis ferramentas de detecção e proteção eficientes (MELO et al., 2011).

O contexto tecnológico atual deu espaço para o crescimento de uma ameaça conhecida como *botnet*. Segundo Ceron (2010) *botnet* pode ser definido como uma rede de dispositivos infectados, que de forma similar a marionetes, podem ser controladas remotamente por um ou mais indivíduos. Quem detém o controle da cadeia é conhecido como *botmaster*, que representa uma figura humana e tem a capacidade de gerir toda a estrutura.

Ademais, cada aparelho infectado dentro da arquitetura manifesta um conjunto de funções anteriormente programadas que podem ser utilizadas pelo controlador (CERON, 2010). Na maioria dos casos tais funcionalidades capacitam a rede a fazer ataques maliciosos, comprometendo o alvo. Dessa forma pode-se relacionar *botnets* com códigos maliciosos.

Mirai Botnet, foi o nome escolhido para a *botnet* que emprega aparelhos IoT (principalmente câmeras de segurança) para criar sua rede maliciosa. O Mirai Botnet foi usado para realizar um ataque a rede de computadores em 2016, esse ataque consistiu no maior ataque DDoS (*Distributed Denial of Service*) visto até os dias de hoje. Segundo Favarin (2017), o Mirai gerou um ataque DDoS da ordem de 1.2 Tbps de tráfego contra o provedor de serviços *Dynamic Network Services* (*Dyn*).

Em Costa (2018) foi desenvolvida a primeira versão de um aplicativo para dispositivos móveis, Mirai Scanner. O intuito do aplicativo é varrer redes domésticas e encontrar dispositivos vulneráveis ao *malware* Mirai a partir da correlação de dados entre portas abertas e uso de senhas padrão.

O principal objetivo do presente trabalho é realizar modificações no aplicativo

Mirai Scanner para permitir que as varreduras conduzidas pelos usuários possam ser enviadas e armazenadas em um banco de dados remoto. Os objetivos secundários envolvem a publicação de tal aplicativo na loja da *Google Play Store* e uma análise preliminar das varreduras feitas pelos usuários que instalaram o aplicativo.

A pesquisa se deu a partir do desenvolvimento de modificações na primeira versão do aplicativo Mirai Scanner, foi construída uma *Application Programming Interface* (API) para a persistência dos dados dos usuários. Foi utilizado a plataforma Heroku para hospedar a API e fornecer um banco de dados para salvar as informações. Dessa forma, a partir de alterações no software o mesmo conseguiu se comunicar com a API.

Com isso, foi possível ter como resultado o aplicativo disponível na Google Play, além da possibilidade de análise dos dados coletados pelos usuários.

O trabalho em questão está organizado da seguinte forma. O capítulo 2 é responsável por retratar a Revisão Bibliográfica, trata-se de uma base teórica que aborda trabalhos que contribuem ou se assemelham com o que aqui se desenvolve. O capítulo 3 aborda o desenvolvimento do trabalho, ou seja, explicita o que foi realizado na pesquisa . O capítulo 4 é responsável por apresentar os resultados encontrados. Por fim, o capítulo 5 aborda a conclusão.

2 REVISÃO BIBLIOGRÁFICA

Este capítulo aborda a fundamentação teórica importante para a construção deste trabalho. Desse modo, há o levantamento bibliográfico de questões ligadas direta e indiretamente a pesquisa em questão. Inicialmente há a discussão acerca dos conceitos básicos para melhor entendimento e clareza do estudo, como conceituação de Segurança da Informação, *malwares*, *botnets*, ataques de negação de serviço, além da relação entre esses conceitos no contexto da Internet das Coisas. Há ainda uma análise dos trabalhos com maior relevância sobre a temática central da pesquisa, ou seja, sobre segurança da informação.

2.1 Segurança da Informação

Antes do advento da Internet e da popularização dos computadores pessoais o armazenamento das informações era realizado por meio de papéis e documentos físicos. A partir do século XXI, a informação passou a ser armazenada em dispositivos computacionais. Assim, o desenvolvimento de mecanismos para proteger as informações digitais, também conhecido como segurança da informação, se tornou uma importante área de pesquisa.

Em linhas gerais, a segurança da informação está pautada no fornecimento e manutenção de três fatores, sendo eles, a confidencialidade, a integridade e a disponibilidade da informação. Cada um desses fatores será apresentado a seguir (BENETTI, 2015).

1. Confidencialidade - trata da garantia de que a informação será conhecida apenas por quem tem permissão para conhecer a mesma, restringindo a visibilidade dos dados para futuras ameaças, ou seja, garantir que a informação seja sigilosa.

   Desse modo a confidencialidade é o ato de manter em confidência uma informação, independente de qual seja ela. Essa prática é custosa de ser mantida, uma vez que, muitos dados são utilizados por pessoas comuns, sucessíveis a engenharia social e que, além disso, podem estar em redes de computadores, sem possuir devidos cuidados de acesso e segurança. Dessa forma, a confidencialidade pode ser mantida a partir do uso de criptografias, isso é, a utilização de princípios e técnicas que visam tornar a informação ininteligível a indivíduos que não possuem acesso as combinações utilizadas para descriptografar os dados.

2. Integridade - diz respeito à garantia de que a informação não terá seu conteúdo comprometido por nenhuma transação, ou seja, os dados podem ser atualizados, re-

movidos, inseridos ou passar por qualquer outra operação similar e esses se manterão corretos quando consultados.

3. Disponibilidade - está relacionada a garantia de que a informação possa ser encontrada sempre que solicitada, ou seja, que esteja sempre disponível para quem precisar dela.

   Um tipo de ataque que pode vir a causar problemas na garantia desse fator, são ataques de negação de serviço (também conhecido como *DoS attack*), o objetivo dessa ação e fazer com que os recursos de um sistema se tornem indisponíveis para os seus usuários. Servidores *web* são as vítimas mais frequentes, não se tratando de uma invasão do sistema, mas sim uma invalidação por sobrecarga do mesmo (CANALTECH, 2019).

2.2 IoT

Segundo Santos et al. (2016), a Internet das Coisas (IoT) pode ser resumida em uma extensão da Internet atual. Ela propicia aos objetos do dia a dia, que possuam capacidade computacional e de comunicação, conexão à Internet. A conexão de aparelhos como TVs, laptops, automóveis, smartphones, consoles de jogos, webcams entre outros com a Internet, permitirá inicialmente que ocorra o controle remoto dos objetos, e após isso possibilitará que os próprios aparelhos sejam acessados como provedores de serviços. O autor retrata ainda que as capacidades adquiridas pelos objetos comuns são capazes de gerar um grande número de oportunidades acadêmicas e industriais, porém, essas possibilidades apresentam riscos e geram uma série de desafios tanto técnicos como sociais.

A partir da utilização dos objetos conectados torna-se possível o controle dos mesmos, a viabilização da troca de informações entre uma coisa e outra, o acesso a serviços da Internet e também a interação com pessoas. Considerando esse fator, surge novas possibilidades de aplicações, como por exemplo cidades inteligentes, casas inteligentes, e consequentemente manifesta também desafios relacionados a padronização, regulamentações e segurança (SANTOS et al., 2016).

De acordo com Khan e Salah (2018) houve um rápido crescimento e evolução das técnicas de hardware a partir da IoT. Esse desenvolvimento ocasionou melhorias da largura de banda, que incorporaram redes cognitivas baseadas em rádio. As redes sem fio de sensores (WSNs) e a *Machine-to-Machine* (M2M), ou sistemas *cibersfísicos* (CPS) também evoluíram como componentes integrais para o termo mais amplo IoT. Apesar dos avanços constantes, os problemas de segurança já existentes nas arquiteturas mencionadas se perpetuam no paradigma IoT.

A IoT utiliza-se de uma série de redes e dispositivos interligados, e herda assim

questões de segurança do sistema convencional de redes de computadores. A partir disso, o progresso desses itens apresenta desafios significativos para a segurança IoT, uma vez que os dispositivos envolvidos nessa tecnologia podem possuir potência e memória limitadas (KHAN; SALAH, 2018).

Os autores Khan e Salah (2018) descrevem alguns desafios para uma implantação segura dentro do paradigma IoT, sendo eles, confidencialidade e integridade dos dados. À medida que os dados trafegam, se faz necessário o uso de criptografia para garantir os requisitos mencionados, pois dispositivos comprometidos em uma rede IoT podem causar danos aos demais aparelhos da rede.

Além disso, há também os requisitos de autenticação, autorização e contabilidade. A autenticação se faz necessária entre os dois lados da comunicação, o fato de existir uma gama muito grande de arquiteturas e ambientes heterogêneo que suportam tecnologia IoT torna desafiador a definição de um protocolo padrão para autenticação, além disso os mecanismos de autorização garantem que o acesso aos sistemas ou informações seja fornecido aos autorizados e a contabilização do uso dos recursos por meio de relatórios é capaz de fornecer um mecanismo confiável para assegurar o gerenciamento da rede (KHAN; SALAH, 2018).

Outro requisito importante IoT é a disponibilidade de serviços, no geral os ataques relacionados com IoT exploram os componentes em diferentes camadas para deteriorar a qualidade do serviço fornecido aos seus usuários. Usualmente dispositivos IoT são limitados e caracterizados como aparelhos com baixo índice de energia e armazenamento, com tudo os ataques à arquitetura desses equipamentos podem elevar o consumo de energia dos mesmos, esgotando recursos dos equipamentos ligado a rede IoT prejudicando a disponibilidade dos serviços, por meio de solicitações de serviço redundantes ou falsificadas

2.3 *Botnets*

Segundo Nunes, Rodrigues e Mourão (2014), *botnets* são redes compostas por máquinas infectadas. Tais máquinas são chamadas de *bots* e controladas por um ou mais atacantes, denominados *botmasters*. Tal infecção se dá por meio de um *malware* qualquer instalado no dispositivo da vítima.

As *Botnets* são redes de dispositivos que possuem alguma capacidade de processamento como envio e recebimento de informações. Podem ser exemplos computadores, celulares, câmeras de segurança e outros, que de alguma forma foram infectados, e agora constituem uma espécie de rede zumbi. No século XXI uma das mais perigosas ameaças ao setor de segurança da informação são as *botnets* (WANG et al., 2010).

Segundo Ceron, Granville e Tarouco (2008), em geral as *botnets* podem danificar

a estrutura da Internet, através de ataques de negação de serviços distribuídos (DDoS), envio de *spams*, compartilhamento de *malwares* e roubo de informações.

A Figura 1 ilustra a estrutura de uma botnet.

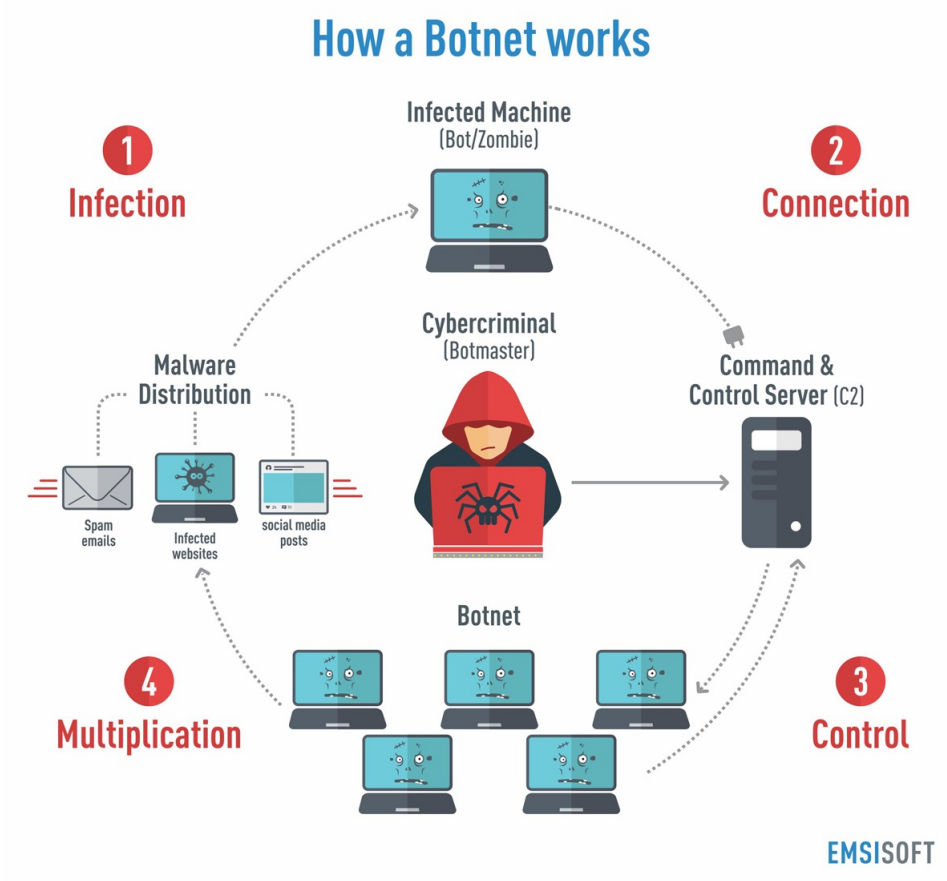

Figura 1 – Estrutura de uma *botnet*. Extraído de Prado (2018)

É possível distinguir três tipos de *botnets*: as centralizadas, as descentralizadas e as híbridas. As híbridas são as *botnets* constituídas por centralizadas e descentralizadas (NUNES; RODRIGUES; MOURÃO, 2014).

Na centralizada há a utilização de um centro de comando e controle (C&C). Esse centro compartilha informações com os *bots* da rede para atualizações e comandos, essa ordem é gerenciada pelo *botmaster*, ou seja, o operador humano. Em arquiteturas centralizadas os protocolos que geralmente são usados para a comunicação da *botnet* são o *Internet Relay Chat* (IRC) e o *Hypertext Transfer Protocol*(HTTP).

Botnets descentralizadas são mais difíceis de serem desarticuladas, ou seja, descobertas. Isso ocorre porque o reconhecimento dessas *botnets* não ocasiona, necessariamente, falhas na estrutura inteira, uma vez que não há C&C central para ser localizado e desativado (NUNES; RODRIGUES; MOURÃO, 2014).

As *botnets* descentralizadas comumente utilizam uma variedade de protocolos *Peer-to-peer* (P2P), essa é uma arquitetura de redes de computadores em que cada um dos

pontos ou nós da rede podem se desenvolver tanto como cliente quanto como servidor, assim há a permissão de compartilhamentos de serviços, além de dados sem a necessidade de um servidor central.

Nunes, Rodrigues e Mourão (2014) aborda que as *botnets* híbridas possuem características centralizadas e descentralizadas, assim é possível a utilização de uma *botnet* com arquitetura P2P com *superpeers*, e alguns *peers* atuarem por um período rotativo como C&C por exemplo. Nessa arquitetura, os *bots* pertencentes a ela são classificados como *bots* clientes e *bots* serventes. Os *bots* serventes possuem atuação dupla, ou seja, podem atuar como clientes e servidores, além disso, possuem IP estático e são os únicos que podem ter o endereço *Ip* na lista de *peers* para serem conectados. Já os *bots* clientes, possuem um IP dinâmico e não aceitam conexões.

O que difere uma *botnet* convencional de uma *botnet* IoT é a gama de itens que são usados para constituir a suas respectivas cadeias de aparelhos. As *botnets* comuns usam apenas computadores e similares, já as *botnets* IoT conseguem usar qualquer aparelho que tenha mínima capacidade computacional, como, por exemplo câmeras de segurança, geladeiras, impressoras e análogos.

A maior parte dos detentores dos dispositivos que constituem uma *botnet* IoT não possuem conhecimento de segurança ou incentivo para adotar métodos de prevenção para seus aparelhos (JERKINS, 2017). Dessa forma poucos usuários possuem o hábito de alterar configurações padrão, que envolve senhas e acessos, ou ajustar alguma chave de segurança quando não existe nenhuma pré-configurada para os seus equipamentos. Há ainda a falta de preocupação para atualizações voltadas para a correção de falhas de segurança, o que facilita a exploração destes itens por indivíduos com más intenções.

A partir da demanda do próprio mercado, para tornar os aparelhos competitivos nesse contexto, em sua produção vários itens, inclusive os de segurança, são removidos. Isso se dá para que esses dispositivos sejam lançados mais baratos no comércio (JERKINS, 2017).

Há alguns dispositivos IoT que podem ser considerados de baixa interação, é o caso de geladeiras e impressoras por exemplo. A baixa interação significa dizer que os usuários finais não interagem por um longo período de tempo com esses dispositivos, ou seja, os indivíduos não estão em contato com esses dispositivos em diversos momentos do dia, mas sim em momentos específicos, como ao imprimir algo ou buscar algum alimento.

No geral, os dispositivos, mesmo não sendo utilizados, estão conectados à redes de alta velocidade, o que possibilita um volume alto de tráfego de ataques DDoS por cada dispositivo comprometido.

2.3.1 Mirai

A seguir será apresentado um histórico sobre o Mirai e uma visão geral a respeito do seu funcionamento.

2.3.1.1 Histórico

Em outubro de 2016 um ataque de larga escala de negação de serviço foi presenciado na Internet. Tal ataque foi capaz de esgotar os recursos de serviços como Twitter, New York Times e Spotify. Ocorreu a geração de um enorme tráfego de rede que sobrecarregou os servidores que proviam estes serviços. A novidade nesse ataque de negação conhecido como DDoS foi de onde veio tamanho poder. Certa ação contou com centena de milhares de dispositivos IoT controlados por uma *botnet* cujo nome é Mirai.

De acordo com Edwards e Profetis (2016), a partir do primeiro ataque do Mirai, o *Security Research Group* (SRG) da *Rapidity Networks*, *Incorporation* (Inc), deu início a uma maior atenção a esse *malware*.

Iniciou-se exames do código fonte com o intuito de se obter mais informações sobre o seu funcionamento. Para essa verificação foi utilizada uma rede de *honeypots* de média interação, ou seja, computadores designados a atração de atividades suspeitas, com o intuito de adquirir informações através da imitação de um dispositivo IoT vulnerável. A finalidade era conseguir que o Mirai tentasse o recrutamento dos dispositivos forjado.

O Mirai é um dos *botnets* IoT com mais força de impacto (ANGRISHI, 2017). Em japonês, de acordo com o autor, Mirai significa futuro o que se relaciona com sua grande influência atual. Além disso, Angrishi (2017) revela que o Mirai pode ser considerado como o próximo passo em *malwares* do tipo *botnet* IoT, comparando-o com o *Remaiten*, que apesar de não ser tão sofisticado quanto esse, consegue ser mais eficaz.

O *Remaiten* possui como vítima dispositivos IoT, e combina capacidades dos malwares de linux Tsunami e Gafgyt. O Tsunami é um *software* (Bot Backdoor de IRC) usado para propagar ataques DDoS. Já o Gafgyt é utilizado como um *backdoor* que pode ser manipulado de forma remota, além de usado como um *Telnet* para escaneamento.

Os dispositivos infectados pelo Mirai, em sua maioria, são câmeras de circuito fechado ou circuito interno de televisão *closed-circuit television* (CCTV), câmeras digitais *Video Recorder Digital Video Recorder* (DVRs) e roteadores domésticos, que é a principal fragmentação do Mirai. A Figura 3, ilustra o poder de alcance do Mirai, mostrando as diversas regiões que o mesmo foi registrado.

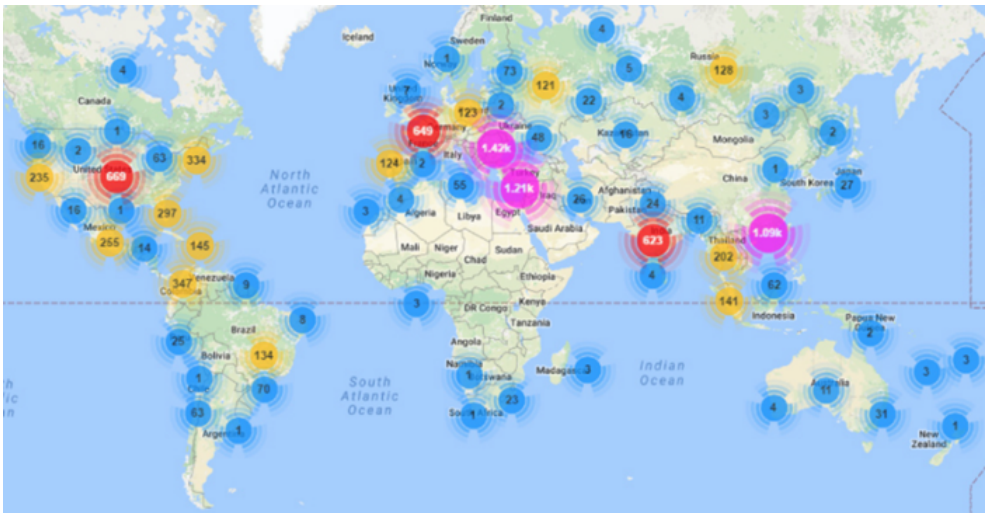

Figura 2 – Registro de ataques usando Mirai no mundo todo. Extraído de Farinaccio, Rafael (2017)

2.3.1.2 Funcionamento

Camargo (2018) retrata que o Mirai Botnet possui quatro componentes principais. Entre eles o *bot* que é o *malware* que realiza a infecção, propaga a contaminação para dispositivos mal configurados, e ataca o alvo assim que recebe o comando correspondente. Há também o servidor de comando e controle (C&C), que é responsável por fornecer ao *botmaster* uma interface de gerenciamento que verifica a situação da *botnet*, além de instrumentar novos ataques DDoS (KOLIAS et al., 2017b).

Kolias et al. (2017b) abordam que o *Loader* (carregador) é um facilitador da disseminação de executáveis direcionados a uma variedade de plataformas, entre elas é possível citar ARM, MIPS e x86, por exemplo, que se comunicam diretamente com as vítimas. Ele faz parte de um módulo desanexado do servidor C&C e dos *bots*, dessa forma a tarefa de enviar o binário do *malware* para infectar novos dispositivos não é realizada pelos equipamentos IoT, que por possuírem baixo poder computacional deixariam tal tarefa pouco eficiente.

Após a invasão, os bots remetem informações como, dispositivo e método de invasão usado para um Servidor. O servidor recebe essas informações e as repassa para o Loader, e esse é responsável por acessar novamente os dispositivos vulneráveis e infectá-los forçando-os a executar o código fonte de um *bot* (CAMARGO, 2018).

Kolias et al. (2017b) tratam ainda uma lista de passos para o funcionamento do Mirai sendo esses:

1. Passo 1 — força bruta. O Mirai possui 62 possíveis pares de usuários e senhas para tentar se autenticar nos dispositivos encontrados. O *malware* verifica endereços IP públicos gerados de forma aleatória, e assim realiza a busca por novos aparelhos IoT

suscetíveis à invasão pelas portas *Transmission Control Protocol* (TCP) 23 ou 2323. O serviço de Telnet é usado para testar a conexão nas portas mencionadas.

2. Passo 2 — credenciais corretas. Quando ocorre a descoberta das credenciais corretas e se obtém um *shell*, ou seja, uma interface de linha de comando ou gráfica, o *bot* direciona diversas características do dispositivo para o servidor de relatório por meio de uma porta diferente.

3. Passo 3 — servidor C&C. A partir do servidor C&C, o *botmaster* verifica frequentemente as novas vítimas-alvo em potencial, além do status da atual *botnet*, através da comunicação com o servidor de relatórios, geralmente por meio do *Tor*.

4. Passo 4 — Dados da infecção. Após a tomada de decisão de quais dispositivos vulneráveis serão infectados, o *botmaster* propaga um comando *infect* no *Loader* contendo os detalhes necessários. O endereço de IP e arquitetura de *hardware* são exemplos de detalhes.

5. Passo 5 — *Loader*. O *Loader* realiza o *login* no equipamento de destino e orienta o mesmo a baixar e executar a versão binária equivalente do *malware*, geralmente isso ocorre por meio do GNU Wget (www.gnu.org/software/wget/manual/wget.html) ou do *Trivial File Transport Protocol*. Após a execução, o *malware* irá se proteger de outros dispositivos maliciosos, desativando pontos de intrusão, como, por exemplo recursos de *Telnet* e *Secure Shell* (SSH).

   Nesse momento, o *bot* recém integrado consegue se comunicar com o servidor C&C, ou seja, o mesmo está apto a receber comandos de ataque. Isso é possível devido à resolução de um domínio codificado no executável (como referência essa entrada é cnc.changeme.com no código-fonte do Mirai) no lugar de um endereço IP estático. Assim, o *botmaster* tem a facilidade de conseguir mudar seu endereço IP ao longo do tempo sem modificar o binário e sem comunicação extra.

6. Passo 6 — *botmaster* O *botmaster* orienta todos os *bots* a começarem um ataque contra um alvo por meio de um simples comando do servidor C&C com os parâmetros correspondentes, sendo eles duração do ataque e os endereços IP das instâncias de *bot* e do servidor de destino.

7. Passo 7 — Ataque As instâncias de *bot* começarão a atacar o servidor de destino com uma das 10 variações de ataque disponíveis, como os ataques de *Generic Routing Encapsulation* (GRE), Protocolo de Controle de Transmissão (TCP) e Protocolo HTTP *flooding*.

A Figura 3, representa os passos do funcionamento do Mirai mencionados.

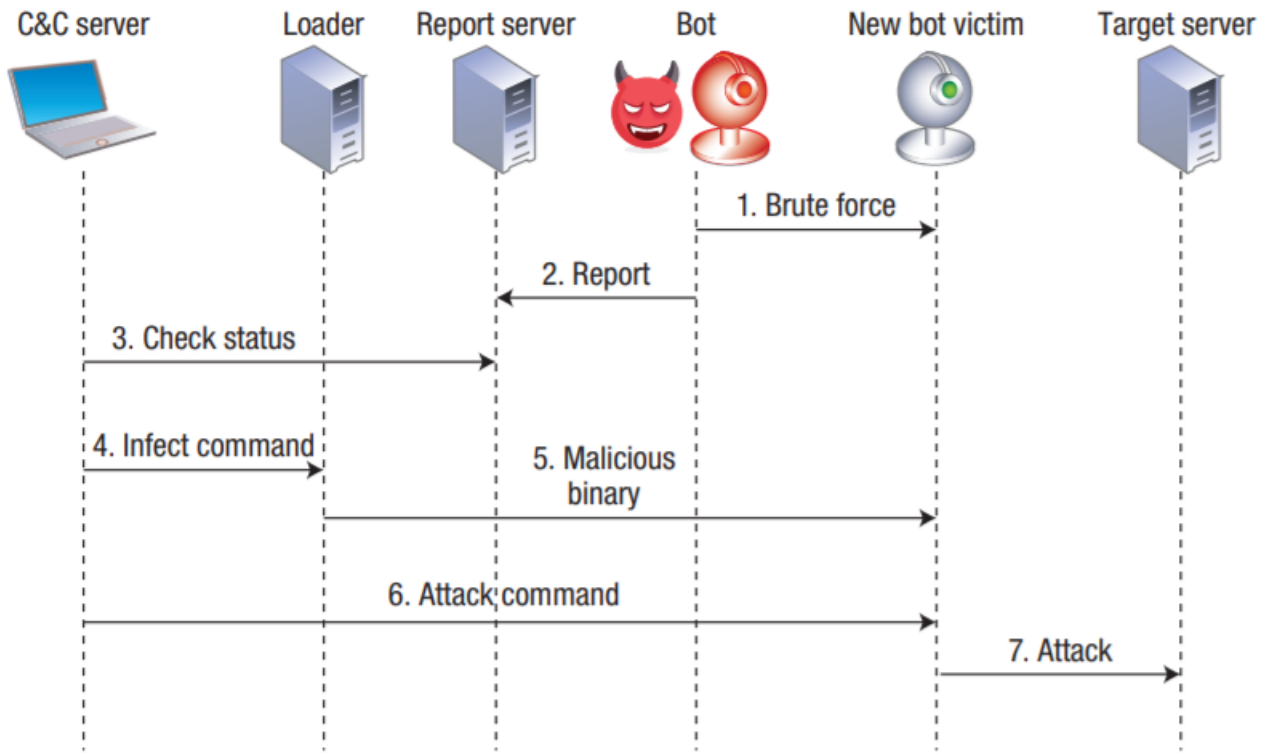

Figura 3 – Comunicação e operação da botnet Mirai. Extraído de Kolias et al. (2017a).

2.4 Trabalhos correlatos

Nesta seção será abordada uma série de trabalhos correlatos, ou seja, pesquisas que tratam de assuntos pertinentes ao trabalho em questão. Em particular serão discutidos trabalhos que analisam e identificam Botnets IoT.

Kolias et al. (2017b) discutem o aumento da popularidade da IoT e a ubiquidade que essa tecnologia possui. Considerando esses fatores, Kolias et al. (2017b) alertam sobre o proeminente risco desse desenvolvimento tecnológico. No geral os dispositivos estão frequentemente conectados à Internet e grande parte desses contém diversas vulnerabilidade, além de configurações de segurança facilmente burladas. A partir da grande disseminação de dispositivos IoT há uma quantidade valorosa desses na rede de Internet, com isso a ausência em poder computacional é compensada em números. Tais fragilidades na segurança atraem usuários maliciosos, principalmente aqueles adeptos à ataques DDoS.

Angrishi (2017) aborda que IoT é o próximo grande passo na evolução da Internet, porém o mesmo alerta sobre os perigos que esse avanço pode trazer. Ele argumenta que há dois grandes fatores com influências negativas. O primeiro deles, é a grande quantidade de dispositivos que é acessível por meio da Internet pública. O segundo fator de risco, se dá devido a reflexão tardia em relação a segurança na fabricação dos aparelhos, mais especificamente na arquitetura dos dispositivos IoT que são amplamente disseminados

no mercado. O autor também apresenta conceitos sobre *botnet* e explica o funcionamento dessas, abordando inclusive as vulnerabilidades exploradas por essas ameaças. Além disso, o trabalho fornece uma lista de práticas que podem evitar esse tipo de ataque, métodos que devem ser adotados por fabricantes e em alguns casos usuários finais.

O trabalho de Prado (2018) propõe a investigação do funcionamento do Mirai. O autor desenvolveu um ambiente de teste com o objetivo de analisar de forma mais detalhada o *malware* e conseguiu identificar as vulnerabilidades que foram exploradas para infectar os dispositivos. Os resultados desse trabalho foram úteis para criação de procedimentos capazes de detectar fragilidades em dispositivos IoT.

Dietz et al. (2018) abordam maneiras de detecção de *botnet* IoT no nível do acesso dos roteadores. O autor em questão desenvolveu uma abordagem de detecção e isolamento no nível de roteadores de acesso e, a partir dessa abordagem, conseguiu notar que a mesma possibilitou uma maior resistência aos dispositivos IoT contra ataques. A abordagem desenvolvida facilita a detecção automatizada de vulnerabilidades dos dispositivos IoT a partir do isolamento dos roteadores de acessos internos *firewall*, além disso o mecanismo de atualização é baseado em um serviço online. Outro fator apresentado no trabalho, sugere que a metodologia mostrada não exige conhecimentos técnicos aprofundados, tornando viável para usuários finais. Ademais a solução proposta se integra de forma fácil com a infraestrutura presente nos dispositivos, facilitando sua implantação.

Jerkins (2017) revisa o código fonte do Mirai, utilizando o mesmo mecanismo de catálogo de dispositivos IoT. O método adotado consistiu em modificar o código do Mirai mudando seu comportamento e utilizando o mesmo em um ambiente legal, para motivar proprietários e fabricantes a mudarem seu comportamento a respeito dessa tecnologia. Para isso, foi implementado o código modificado em um banco de testes privado para testar as modificações. As alterações possibilitaram o *scanner* identificar aparelhos vulneráveis com êxito. Assim, é importante visualizar o funcionamento do Mirai, uma vez que entender o modo como a infecção do Mirai se dá auxilia na construção de mecanismos de prevenção, como é o caso do aplicativo Mirai Scanner, proposto nesta pesquisa.

Meidan et al. (2018) propõem o uso de um laboratório de testes com duas *botnets*, Mirai e BASHLITE, o objetivo foi o desenvolvimento de novos métodos capazes de detectar ataques lançados a partir de dispositivos IoT vulneráveis, além de diferenciar ataques IoT. A abordagem consiste em um método que extrai comportamentos anômalos do tráfego de rede, e assim, realiza a detecção do tráfego anormal propagado por dispositivos IoT vulneráveis. O autor conseguiu notar que o método proposto realmente era capaz de detectar com precisão os ataques quando esses proviam de dispositivos IoT vulneráveis.

Por fim, Costa (2018) desenvolveu um aplicativo Android com o intuito de identificar dispositivos IoT que de algum modo pudessem apresentar possíveis vulnerabilidades

a serem exploradas pelo Mirai, e outros, além da conscientização do usuários dos distintos riscos envolvidos pela tecnologia IoT. O aplicativo desenvolvido possibilita varrer redes sem fio, encontrar dispositivos e executar a verificação das possíveis vulnerabilidades ao Mirai é o Mirai Scanner, aplicativo também utilizado neste trabalho para melhorias no seu desenvolvimento.

3 DESENVOLVIMENTO

O conteúdo deste capítulo tem como propósito demonstrar de forma detalhada o desenvolvimento da API usada para possibilitar a comunicação entre o aplicativo (Mirai Scanner) o e servidor, além de passar uma visão geral do funcionamento do software e das mudanças feitas no mesmo.

3.1 Mirai Scanner

O aplicativo em questão, Mirai Scanner, é derivado de pesquisas realizadas por alunos de graduação (Antonio Carlos Campos da Silva Junior, Gabriel Miranda Costa e Marcelo Alves Prado) da Faculdade de Computação (FACOM) — UFU e conduzidas pelo professor Dr. Rodrigo Sanches Miani. O Mirai Scanner é utilizado para verificação de dispositivos vulneráveis ao *malware* Mirai em redes sem-fio.

O aplicativo foi desenvolvido para celulares portadores do sistema operacional Android, e se encontra disponível na *Play Store* para download [1]. O *software* tem como intuito averiguar redes sem fio, varrendo e identificando dispositivos conectados as mesmas, além de trazer informações de forma individual para cada item encontrado na rede.

Após a instalação do *software*, para executá-lo, é necessário que o mesmo se mantenha conectado a uma rede sem fio, caso contrário o programa não permitirá a execução do escaneamento. Satisfeita essa condição o aplicativo busca informações dentro da rede que se encontra acionado, como a identificação e endereço de IP do roteador, dessa forma é possível localizar os demais dispositivos na rede.

Tendo como propósito escanear a rede em que se encontra rastreando possíveis vulnerabilidades de segurança a respeito do Mirai basta um clique para se abrir e começar o escaneamento. O usuário tem a opção de reiniciar a qualquer momento a busca ou inicializar uma nova varredura de forma manual por meio de um botão situado no canto superior direito da interface do *software*, além disso, o utilizador pode acompanhar o decorrer da busca por meio de uma barra de carregamento que evolui em forma de porcentagem de acordo com o progresso.

A tela inicial também conta com um espaço em branco, que é preenchido pela listagem de dispositivos encontrados na busca, caso o número de itens ultrapasse a capacidade da página, uma barra de rolagem lateral é criada, possibilitando a visualização de todos os aparelhos encontrados na varredura. A Figura 4 do *layout* inicial ilustra a tela inicial do aplicativo.

[1] https://play.google.com/store/apps/details?id=miraiscanner.facom.ufu.br.miraiscanner

Figura 4 – Tela inicial após execução do aplicativo.

Após o término da varredura, há a apresentação de uma lista de dispositivos ao usuário, esses dispositivos representam os aparelhos encontrados na rede. É apresentada também o nome da rede sem fio em que ocorreu a procura. A Figura 5 ilustra a tela do aplicativo durante o momento explicado anteriormente.

Em seguida o usuário pode clicar sobre qualquer item da listagem. Cada elemento possui as mesmas informações específicas, sendo elas o endereço de IP, fabricante e número de MAC, além de informações sobre as portas 23, 2323, 48101 e por fim uma mensagem trazendo a informação se o dispositivo está vulnerável ou não.

A seguir há a Figura 6 que ilustra a tela no instante tratado acima, considerando um dispositivo não vulnerável e uma Figura 7 demonstrando a tela de um dispositivo vulnerável.

O aplicativo em questão não difere dispositivos IoTs em sua lista, pois, o Mirai faz uso de qualquer mecanismo que demonstre ser vulnerável e consequentemente explorado. De forma geral computadores pessoais, celulares, e aparelhos mais sofisticados, tendem a

Figura 5 – Lista de dispositivos encontrados na rede em que o aplicativo se encontra conectado.

possuir protocolos de segurança capazes de garantir que os mesmos não sejam infectados pelo Mirai. Nas explicitações seguintes serão utilizadas algumas Figuras que ilustram o que foi explicado.

3.2 Modificações no Mirai Scanner

Esse capítulo apresenta informações referente as modificações que foram feitas no aplicativo Mirai Scanner. A seguir será apresentado duas Figuras 8 e 9 que ilustram o antes e o depois do funcionamento.

O aplicativo Mirai Scanner sofreu várias alterações, elas foram necessárias para possibilitar a persistência dos dados. As modificações se iniciaram pela necessidade de manter as informações que, anteriormente, eram perdidas. Essas informações são de grande valor para o desenvolvimento do trabalho, pois elas permitem a análise das varreduras. Outra limitação estava relacionada a disponibilidade do aplicativo. Inicialmente, a

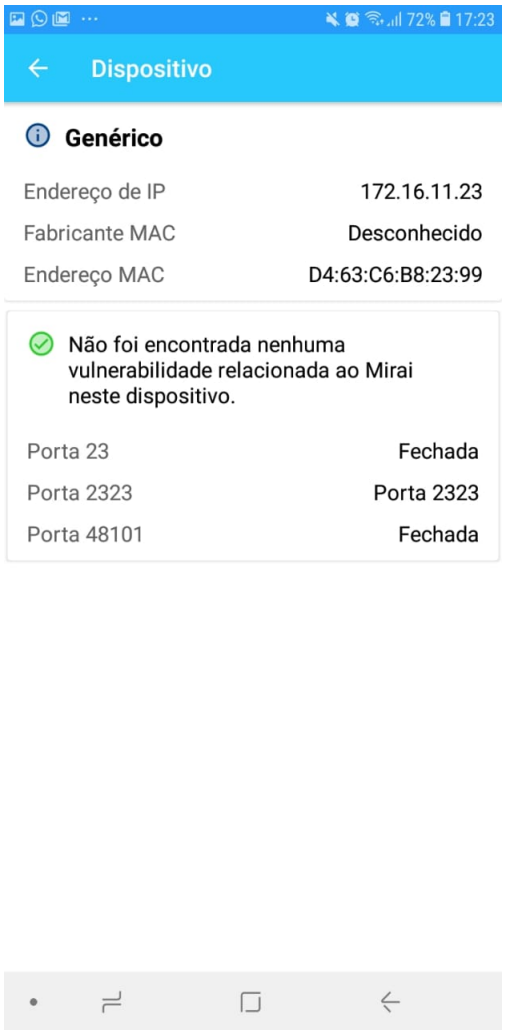

Figura 6 – Dispositivo sem vulnerabilidades.

sua disponibilidade era limitada apenas aqueles que possuíam o APK.

Considerando a necessidade de persistência dos dados gerados pelos usuários, foi utilizada a plataforma Heroku [2] para hospedar a API criada, para receber o *POST* feito pelo aplicativo e salvar as informações vindas do mesmo.

O propósito dessa seção se baseia em explicar de forma detalhada todo o desenvolvimento, desde a criação da API até as modificações no código fonte do *software* para comunicar-se com a mesma e o *deploy* dela no Heroku. Desse modo, a explicitação das modificações desenvolvidas se dará de forma linear aos acontecimentos.

3.3 Construção da API

A API foi construída baseada no *framework Spring boot* para a plataforma Java. O *framework Spring boot* é uma caixa de ferramentas que possibilita de forma mais prática

[2] https://www.heroku.com

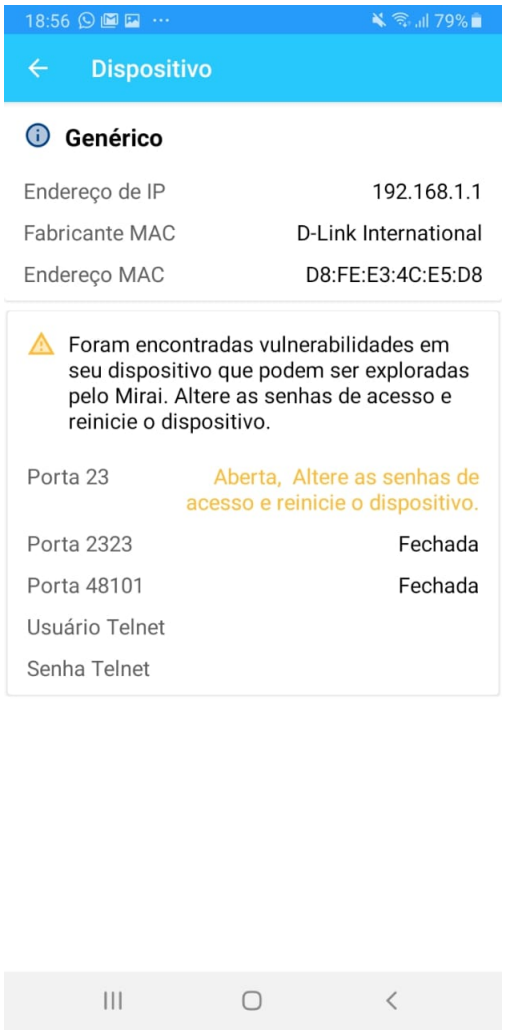

Figura 7 – Dispositivo sem vulnerável.

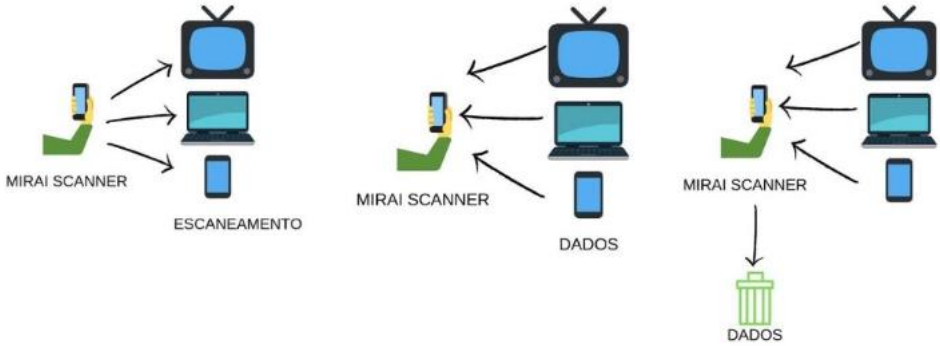

Figura 8 – Aplicativo antes das modificações

o uso de funções, entre elas, conexão e modelagem das informações no banco de dados.

O primeiro passo consiste em criar o projeto, isso se dá a partir da utilização do site https://start.spring.io/. Esse site gera o programa e ainda permite inserir as dependên-

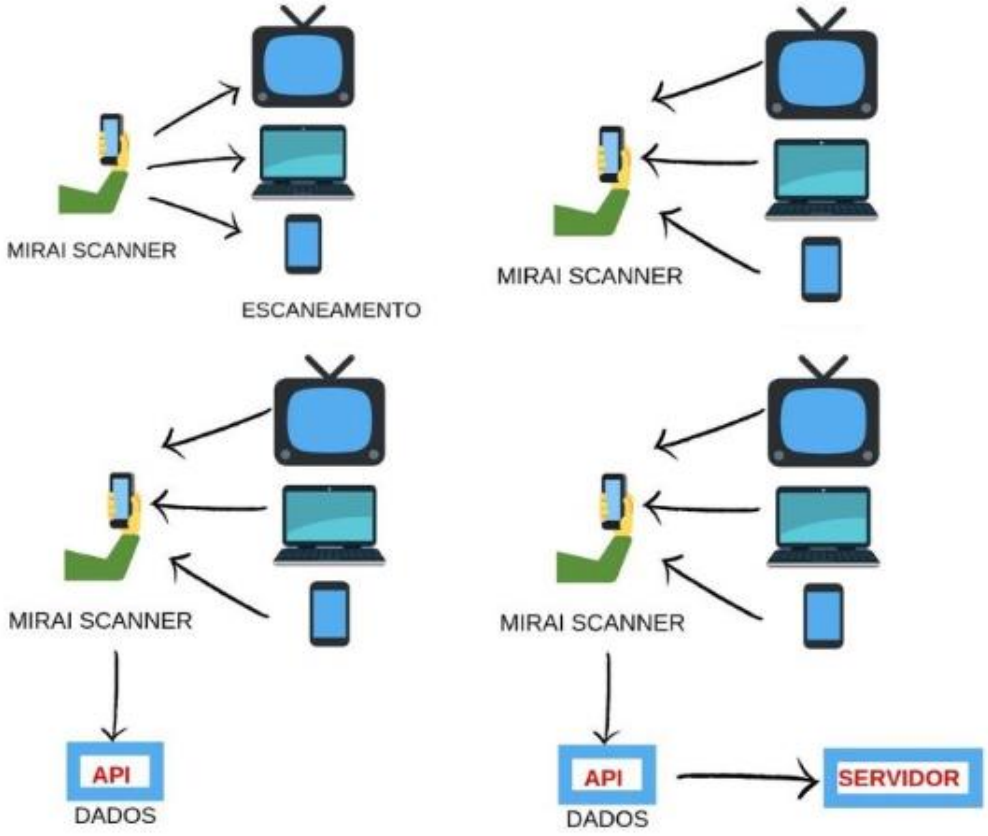

Figura 9 – Aplicativo depois das modificações

cias necessárias de forma automática, apenas selecionando as mesmas. As dependências usadas nesse projeto foram, *Spring Boot DevTools*, *Spring Data JPA*, *Spring Web Starter*, *PostgreSQL Driver*. Após gerar o projeto basta importá-lo para uma IDE de escolha qualquer, a IDE utilizada nesse trabalho foi a Eclipse.

Com o projeto criado e importado basta abrir a aba pom.xml para conferir todas as dependências, e na guia *Maven Dependencies* pode ser localizado todas as dependências baixadas pelo *Maven*.

Inicialmente foi configurado o banco de dados *Postgress* e criado o *Model*. Para esse projeto a classe refente ao *Model* serviu com o propósito de ser uma entidade para o banco de dados, sendo assim a classe do *Model*, Produto recebe a notação Entity e Table(name="MIRAI_APP"), para representar o nome da tabela no banco de dados. Além disso, cada propriedade do aplicativo que é salva foi representada por uma variável do tipo String, a seguir um trecho do código.

1

2

```
3  @Entity
4  @Table(name="MIRAI_APP")
5  public class Produto implements Serializable{
6
7          private String enderecoIp;
8
9          private String fabricanteMac;
10
11         @Id
12         private String enderecoMac;
13
14         private String porta23;
15
16         private String porta2323;
17
18         private String porta48101;
19
20         private String resposta;
```

Dentro do arquivo, que é padrão do projeto, *application.properties*, foi salvo as configurações do banco de dados. Para cada elemento mencionado até agora e os próximos que sucedem foi criado um pacote para melhor organização do projeto, por exemplo, foi criado um pacote para o *Model*, outro para o *Controller* e assim por diante.

Foi criada uma interface *ProdutoRepository*, essa estende da classe *JpaRepository* herdando todos os seus métodos e recebendo como argumento nossa entidade, o Produto e o tipo do Id. A interface tem como objetivo proporcionar o uso de métodos prontos facilitando a persistência com o banco de dados, pois, ela faz uso do *Jpa*. A seguir um trecho do código.

```
1
2  public interface ProdutoRepository extends JpaRepository<Produto, Long>{
3          Produto findById(long id);
4
5  }
```

Sabendo disso, foi realizada a construção do *Controller*, responsável por receber requisições HTTP, e criação do método *POST*, além de ser responsável por enviar as informações necessárias à base de dados.

A classe referente ao *Controller* recebe as seguintes notações *@RestController* e *RequestMapping*, sendo a primeira responsável por definir que essa classe será responsável por receber as requisições HTTP e a segunda respectivamente por definir uma URI padrão

da *API*. A seguir um trecho do código.

```

@RestController
@RequestMapping(value="/api")
public class ProdutoResource {

        @Autowired
        ProdutoRepository produtoRepository;

        @ApiOperation(value="Salva um produto")
        @PostMapping("/produto")
        public Produto salvaProduto(@RequestBody @Valid Produto produto) {
                return produtoRepository.save(produto);
        }

}
```

O método retorna o objeto produto (que contém as informações do usuário), esse foi salvo e possui dois parâmetros de entrada *@RequestBody* e Produto, o primeiro é responsável por trazer o corpo da requisição e o segundo é o que vai vir de fato nessa. Para esse exemplo Produto é passado para a função em forma de JSON. Exemplo de JSON usado para testar o funcionamento do recurso, o teste foi feito usando a ferramenta Postman, sendo o Postman uma ferramenta com o propósito de testar serviços *RESTfull* através de requisições HTTP e análise dos seus respectivos retornos..

```
{
"enderecoIp" : "teste1" ,
    "fabricanteMac" : "teste1",
    "enderecoMac" : "teste1" ,
    "porta23" : "teste1",
    "porta2323" : "teste1",
    "porta48101" : "teste1",
    "senha" : "teste1",
    "suporte" : "teste1",
    "resposta" : "teste1",
     "usuario" : "teste1"
     }
```

Posteriormente à construção da API foi feito o *deploy* na plataforma Heroku. O Heroku é uma plataforma em nuvem que disponibiliza serviços de forma gratuita e não gratuita, suportando diversas tecnologias. Para o desenvolvimento desse trabalho além

da linguagem Java foi adotado o Heroku Postgres que serviu como banco de dados para armazenamento das informações.

3.4 Adaptando o aplicativo para se comunicar com a API

Nessa seção é mostrado o modo que foi realizado a adaptação do aplicativo para se comunicar com a API. Assim, é demonstrado desde a criação do objeto até a etapa em que esse é salvo no banco de dados, além da identificação das ferramentas utilizadas. Para isso, foi utilizado quatro etapas, sendo a criação do objeto *Post*, a criação da instância do *retrofit*, a criação da interface da API e a criação da *ApiUtils*.

3.4.1 Criação do objeto *Post*

Classe responsável por serializar componentes para serem gravados no banco de dados, as informações contidas nessa classe devem coincidir com as do *Model* da API. A seguir um trecho do código.

```
1
2  public class Post {
3
4      @SerializedName("enderecoIp")
5      @Expose
6      private String enderecoIp;
7
8      @SerializedName("fabricanteMac")
9      @Expose
10     private String fabricanteMac;
11
12     @SerializedName("enderecoMac")
13     @Expose
14     private String enderecoMac;
15
16     @SerializedName("porta23")
17     @Expose
18     private String porta23;
19
20     @SerializedName("porta2323")
21     @Expose
22     private String porta2323;
23
24     @SerializedName("porta48101")
25     @Expose
```

```
26     private String porta48101;
```

3.4.2 Criação da Instância de *Retrofit*

Para emitir solicitações de rede para uma API com *Retrofit*, é necessário criar uma instância usando a classe *Retrofit Builder* e configurá-la com uma URL base. Passa-se uma URL ao chamar o método *RetrofitClient.getClient* (String baseUrl). Esta URL será então usada para construir a instância. Também especificamos o conversor JSON necessário (Gson). A seguir um trecho do código.

```
1
2  public class RetrofitClient {
3
4      private static Retrofit retrofit = null;
5
6      public static Retrofit getClient(String baseUrl) {
7          if (retrofit==null) {
8              retrofit = new Retrofit.Builder()
9                      .baseUrl(baseUrl)
10                     .addConverterFactory(GsonConverterFactory.create())
11                     .build();
12         }
13         return retrofit;
14     }
15 }
```

3.4.3 Criação da interface da API

A interface utilizada é chamada de *APIService*, utiliza-se essa para executar solicitações HTTP, como *POST*, *PUT* e *DELETE*. Para esse trabalho ficou-se restrito ao método *POST*, pois ele corresponde o objetivo proposto. A seguir um trecho do código.

```
1
2  public interface APIService {
3
4      //"/api/produto"
5
6      @POST("api/produto")
7          //@FormUrlEncoded
8      Call<Post> savePost(@Body Post post);
9
10 }
```

3.4.4 Criação do *ApiUtils*

A classe *ApiUtils* serve como uma classe auxiliar, ela é responsável por conter a URL de acesso (a URL em questão foi obtida na plataforma Heroku) como uma variável estática. Além disso, a classe também possui o método estático getAPIService() que auxilia no restante da aplicação. A seguir um trecho do código.

```
1
2  public class ApiUtils {
3
4      private ApiUtils() {}
5
6      public static final String BASE_URL = "https://api-miraiapp.herokuapp.com/" ;
7  //"https://api-miraiapp.herokuapp.com/"
8      public static APIService getAPIService() {
9
10         return RetrofitClient.getClient(BASE_URL).create(APIService.class);
11     }
12 }
```

3.4.5 Execução da solicitação *POST*

A solicitação acontece dentro da classe *DispositivoActivity.java*, e para que isso seja possível foi necessário salvar em variáveis do tipo *String* as informações que o aplicativo já proporcionava, como endereço IP, fabricante Mac, endereço Mac, informações sobre as portas, 23, 2323, 48101, se estão abertas ou fechadas, e se o dispositivo encontrado na busca possui alguma vulnerabilidade a respeito do Mirai.

Em seguida criou-se um objeto da classe *Retrofit* e outro da interface *APIService* que recebe como parâmetro um *.class* para que isso seja possível. Feito isso um objeto do tipo *Post* é inicializado contendo os valores recolhidos pelas variáveis mencionadas anteriormente, posterior a todos esses passos podemos chamar o método *Call* passando o objeto *Post* que será salvo no servidor. A seguir um trecho do código.

```
1
2          Call<Post> Post = service.savePost(post);
3          Post.enqueue(new Callback<miraiscanner.facom.ufu.br.
4          miraiscanner.Network.Post>() {
5              @Override
6              public void onResponse(Call<miraiscanner.facom.ufu.br.
7              miraiscanner.Network.Post> call, Response<miraiscanner.facom.ufu.br.
8              miraiscanner.Network.Post> response) {
9                  int status = response.code();
```

```
10
11                  Post resposta = response.body();
12
13                  Log.i("resposssta","On response" + status);
14              }
15
16              @Override
17              public void onFailure(Call<miraiscanner.facom.ufu.br.
18              miraiscanner.Network.Post> call, Throwable t) {
19
20              }
21          });
```

4 RESULTADOS

Esse capítulo contém os resultados da pesquisa, esses estão divididos em duas partes, sendo a primeira delas referente ao Google Play, 4.1, e a segunda relacionada à análise de varreduras, 4.2. A primeira conterá informações a respeito de como foi feito o processo de disponibilização do aplicativo na Google Play, tornando possível o *download* de qualquer usuário Android que contenha o mínimo de memória para uso do *software*. Respectivamente o segundo item contém os números desse estudo.

4.1 Google Play

O primeiro passo para a publicação do aplicativo na *Play Store* é ter o projeto compilado, posteriormente é necessário gerar uma chave assinada, esse processo pode ser feito pela própria IDE *Android Studio*.

Com o *Android Studio* aberto e o projeto devidamente carregado, deve-se ir em *Build*, *Generate Signed Bundle or APK*. Na guia que se abre é necessário selecionar a opção *APK* e clicar em *Next*, após essa sequência de ações precisa-se criar um repositório de chaves caso não haja nenhuma. Após criar o repositório mencionado deve se clicar em *Next* e gerar a *APK*.

O segundo passo consiste em criar uma conta de desenvolvedor na *Google Play*, para achar a plataforma basta digitar "*Google Play*" no *Google* e acessar o seguinte link, <https://play.google.com/apps/publish/?hl=pt-BR>. Acessando o link descrito pode-se usar uma conta *Google* para a autenticação, para continuar no processo é preciso um cartão de crédito internacional, pois, para a criação da conta de desenvolvedor é necessário pagar uma taxa de 25 dólares (taxa essa que é paga apenas uma vez).

Após a criação da conta de desenvolvedor, basta selecionar a opção criar *App*, e adicionar informações, como nome, descrição, e outros, além de adicionar imagens no formato indicado pela plataforma. Em seguida na aba versões de aplicativos, selecionar a opção versão e posteriormente gerenciar, e criar uma versão, será solicitado a *APK* do aplicativo, após é necessário inserir a mesma na plataforma. Feito isso os próximos passos são referentes a classificação do conteúdo do *software*, que devem ser feitos nas abas classificação de conteúdo e conteúdo do aplicativo. Ao término das etapas anteriores basta voltar na aba versões de aplicativos, clicar em revisar, verificar se existe algum erro ou requisito que não foi devidamente realizado e selecionar a opção iniciar lançamento para a produção, caso todas as exigências estiverem satisfeitas.

4.2 Análise das varreduras

Após o processo completo mencionado na seção anterior, o aplicativo torna-se disponível para *download* na plataforma *Google Play* para qualquer usuário *Android*. A partir da disponibilização do aplicativo Mirai Scanner na plataforma *Google Play*, foi possível obter o seguinte resultado: 71 downloads, 250 escaneamentos, três dispositivos vulneráveis e 247 não vulneráveis. A figura 10 traz essas informações de forma visual.

Figura 10 – Gráfico que apresenta os resultados coletados

A coleta dos dados apresentados na figura 10 se dá por meio do escaneamento, que de acordo com (COSTA, 2018), após o aplicativo estar conectado em uma rede de Internet doméstica é realizado o envio de um comando ping, que verifica a conectividade entre os dispositivos, para o endereço IP que está sob análise. Quando encontrados, os aparelhos ativos, passam por um processo de identificação do número MAC. Em seguida uma requisição HTTP do tipo GET é feita para uma API rest no endereço https://api.macvendors.com/, como parâmetro é passado o endereço MAC, e como resposta obtém-se o nome do fabricante. Para testar a ocorrência de vulnerabilidades é utilizado o conceito de sockets TCP/IP.

O aplicativo ficou disponível cerca de um mês para alcançar esses resultados. Deve-se considerar que a diferença entre *downloads* e escaneamentos se dá porque ao baixar o aplicativo o mesmo pode fazer a varredura de todos os dispositivos conectados na rede de Internet, por isso há apenas 71 *downloads*, mas 250 escaneamentos. Assim, há redes locais que podem ter mais ou menos dispositivos conectados.

Dos três dispositivos mencionados dois apresentaram a porta 23 aberta o que resultou em vulnerabilidades. O outro dispositivo que se mostrou vulnerável possuía a porta 48101 aberta. Nenhum aparelho escaneado possuía a porta 2323 aberta. Além desses

três dispositivos que revelaram dispor de no mínimo uma porta aberta nenhum outro foi encontrado em tais circunstâncias.

Deve-se considerar que dos três dispositivos vulneráveis dois são de mesmo endereço IP e um de endereço IP diferente. O IP que apresentou vulnerabilidades foi o 192.168.1.1. Além disso, é importante ressaltar que os IPs verificados foram somente IPs privados, dessa forma, não se pode obter conclusões detalhadas, mas devido os dados adquiridos há uma hipótese de que o IP com final um possa representar um roteador doméstico.

A seguir a Figura 11 contém informações a respeito dos equipamentos encontrados com vulnerabilidades.

Endereço MAC	Endereço IP	Fabricante MAC	Porta23	Porta2323	porta48101
10:02:B5:09:52:27	192.168.0.11	Intel Corporate	Fechada	Fechada	Aberta
D8:FE:E3:4C:E5:D8	192.168.1.1	D-Link International	Aberta	Fechada	Fechada
F8:8E:85:F2:89:79	192.168.1.1	Comtrend Corporation	Aberta	Fechada	Fechada

Figura 11 – Tabela com os dados dos dispositivos com vulnerabilidades

A porta 23 é uma porta conhecida, ela está no *Service Name and Transport Protocol Port Number Registry*. Essa faz identificação de serviços que podem utilizar os protocolos de transferência TCP e UDP, além disso é responsável pelo serviço de *Telnet*. Tal serviço permite uma comunicação cliente servidor, que possibilita a comunicação de computadores de forma remota.

De forma parecida com a porta 23, a porta 2323 também utiliza-se da identificação de serviços que utilizam os protocolos TCP, UDP e *Telnet* para a transferência de dados.

De acordo com Costa (2018) caso a porta 48101 esteja aberta existe um grande indício que o dispositivo esteja infectado, pois a mesma é usada no envio de comandos para o *bot*.

É necessário que o compromisso com a segurança dos dispositivos IoT seja desenvolvido tanto pelas empresas provedoras como pelos usuários finais. Faz se necessário uma conscientização em relação a essa proteção, e é determinante que as organizações mobilizem os usuários para que esses prossigam com a segurança oriunda das fábricas. Isso significa que é importante uma parceria entre os fabricantes e os consumidores finais para que a segurança IoT seja garantida.

Deve se considerar que os usuários são o elo mais fraco, portanto a informação é de extrema necessidade. Além disso, a partir da dimensão atingida pelo Mirai é indispensável que a compreensão dos riscos por trás de tais aparelhos ocorra, assim é estritamente necessário à prevenção para evitar episódios semelhantes.

Por fim, pode-se considerar que o aplicativo Mirai Scanner vem a ser uma alternativa válida para a constatação de vulnerabilidades em aparelhos IoT. Como dito anteriormente, houve 71 downloads, desses 35 ainda mantiveram o aplicativo em seus celulares durante toda a pesquisa. Além disso, na plataforma do Google Play o aplicativo obteve 5 estrelas, sendo 8 avaliações.

Desse modo, há uma valorização inicial dos usuários da relevância do aplicativo, assim como a importância de mecanismos que auxiliem na segurança de dispositivos IoT. As Figuras a seguir 12, 13 e 14, extraídas da plataforma Google Play demonstram os dados mencionados anteriormente.

Figura 12 – Número de instalações do dispositivo

Figura 13 – Número de desinstalações do dispositivo

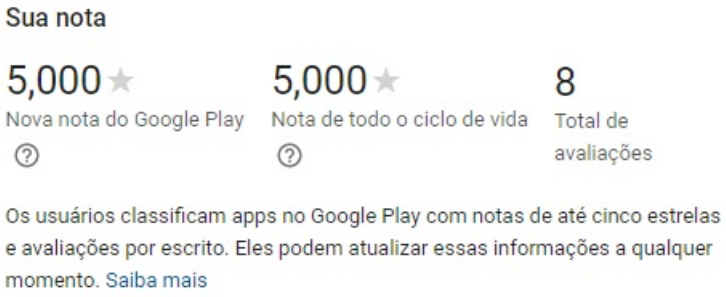

Figura 14 – Avaliação do dispositivo

5 CONCLUSÃO

A partir da análise foi possível concluir que a Mirai Botnet foi um marco no desenvolvimento de *malwares* para IoT. É possível notar que as vulnerabilidades presentes no dispositivo IoT estão sendo constantemente exploradas por usuários mal intencionados. Além disso, o Mirai abriu portas para novos tipos de ataques o que consequentemente influência no compromisso com a segurança.

Pode se considerar que os objetivos do trabalho foram cumpridos. As modificações no Mirai Scanner permitem a coleta e o armazenamento dos dados das varreduras realizadas pelos usuários. É importante mencionar que o armazenamento é feito de forma automática usando uma plataforma de computação em nuvem (Heroku). Tais dados são importantes para traçar um panorama acerca dos dispositivos vulneráveis ao Mirai.

Para o desenvolvimento adequado do trabalho devemos relembrar que diversas alterações foram realizadas no aplicativo Mirai Scanner. O aplicativo original não era capaz de salvar informações, e no geral as mudanças serviram para implementar essa funcionalidade e disponibilizar o aplicativo na plataforma *Play Store*, para que qualquer usuário detentor de um *smartphone android* fosse capaz de fazer o download do mesmo.

Para que o aplicativo atingisse essa nova versão foi necessário a construção de uma API, desenvolvida com o uso do *framework Spring boot* para a plataforma Java, que foi implantada na plataforma Heroku(plataforma em nuvem que oferece serviço para diversas linguagens e tecnologias). Posterior a construção da API, o aplicativo foi submetido a alterações, sendo a principal delas a adição da biblioteca Retrofit, que auxilia no processo de envio de requisições. Para o trabalho em questão foi utilizado o método *POST*, e para fins de teste foi utilizado a ferramenta *Postman*.

Para uma maior abrangência da pesquisa se faz necessário algumas mudanças em possíveis trabalhos futuros. É importante que nas próximas pesquisas alguns dados como a geolocalização, assim como os IPs reais possam ser coletados e armazenados.

Além disso, a mobilização de uma maior quantidade de usuários também se faz importante, porque a partir de um maior número de indivíduos utilizando o aplicativo, a amostra torna-se mais significativa.

Avaliar as varreduras levando em consideração os novos dados, assim como modificações do aplicativo para que outros *malwares*, além do Mirai, sejam verificados também é considerável, uma vez que a partir disso, torna-se é possível ter uma melhor noção de quais *malwares* estão se proliferando nas redes.

Por fim, o dispositivo Mirai Scanner está disponível apenas para usuários Android,

assim a pesquisa conseguiria um maior número de dados se o aplicativo fosse disponibilizado também para dispositivos IOS.

REFERÊNCIAS

ANGRISHI, K. Turning internet of things (iot) into internet of vulnerabilities (iov): Iot botnets. *arXiv preprint arXiv:1702.03681*, 2017. Citado 2 vezes nas páginas 18 e 21.

BARBOSA, A. S. et al. Relações humanas e privacidade na internet: implicações bioéticas. *Revista de bioética y derecho*, SciELO Espana, n. 30, p. 109–124, 2014. Citado na página 10.

BENETTI, T. *Segurança da Informação – Confidencialidade, Integridade e Disponibilidade (CID)*. 2015. Citado na página 13.

BOHRER; MAGALHÃES, F.; ROCHA, L. A. D. O processo de internacionalização de empresas de comércio eletrônico sob o olhar comportamental: Estudo longitudinal dos casos mercadolibre e ebay. *Revista Alcance*, Universidade do Vale do Itajaí, v. 21, n. 1, p. 126–151, 2014. Citado na página 10.

CAMARGO, C. I. Mirai: um estudo sobre botnets de dispositivos iot. Universidade Federal de Brasilia, 2018. 174 f., il. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação). Citado na página 19.

CANALTECH. *O que é DoS e DDoS?* 2019. Disponível em: <https://canaltech.com.br/produtos/O-que-e-DoS-e-DDoS/>. Acesso em: 2019-11-26. Citado na página 14.

CERON, J. M. *Arquitetura distribuída e automatizada para mitigação de botnet baseada em análise dinâmica de malwares*. Dissertação (Mestrado) — Universidade Federal do Rio Frande do Sul, 2010. Citado na página 11.

CERON, J. M.; GRANVILLE, L. Z.; TAROUCO, L. M. R. Uma arquitetura baseada em assinaturas para mitigaç ao de botnets. *X Simpósio Brasileiro em Seguranca da Informação e de Sistemas Computacionais (SBSeg)*, p. 105–118, 2008. Citado na página 15.

COSTA, G. M. Implementação de um aplicativo para detecção de botnets iot em ambientes domésticos. Universidade Federal de Uberlândia, 2018. 60 f. Trabalho de Conclusão de Curso (Graduação em Sistemas de Informação). Citado 4 vezes nas páginas 11, 22, 37 e 38.

DIETZ, C. et al. Iot-botnet detection and isolation by access routers. In: IEEE. *2018 9th International Conference on the Network of the Future (NOF)*. [S.l.], 2018. p. 88–95. Citado na página 22.

EDWARDS, S.; PROFETIS, I. Hajime: Analysis of a decentralized internet worm for iot devices. *Rapidity Networks*, v. 16, 2016. Citado na página 18.

Farinaccio, Rafael. *Registro de ataques usando Mirai no mundo todo*. [S.l.], 2017. Disponível em: <https://www.tecmundo.com.br/ataque-hacker/115425-ataque-ddos-nova-variante-malware-mirai-chega-durar-54-horas.htm>. Citado 2 vezes nas páginas 6 e 19.

FAVARIN, C. Data center-os paradigmas de segurança. *Datacenter: projeto, operação e serviços-Unisul Virtual*, 2017. Citado na página 11.

FRANÇA, T. C. de et al. *Web das coisas: conectando dispositivos físicos ao mundo digital*. [S.l.: s.n.], 2011. Citado na página 10.

JERKINS, J. A. Motivating a market or regulatory solution to iot insecurity with the mirai botnet code. In: IEEE. *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*. [S.l.], 2017. p. 1–5. Citado 2 vezes nas páginas 17 e 22.

KHAN, M. A.; SALAH, K. Iot security: Review, blockchain solutions, and open challenges. *Future Generation Computer Systems*, Elsevier, v. 82, p. 395–411, 2018. Citado 2 vezes nas páginas 14 e 15.

KOLIAS, C. et al. Ddos in the iot: Mirai and other botnets. *Computer*, v. 50, n. 7, p. 80–84, 2017. ISSN 0018-9162. Citado 2 vezes nas páginas 6 e 21.

KOLIAS, C. et al. Ddos in the iot: Mirai and other botnets. *Computer*, IEEE, v. 50, n. 7, p. 80–84, 2017. Citado 2 vezes nas páginas 19 e 21.

MEIDAN, Y. et al. N-baiot—network-based detection of iot botnet attacks using deep autoencoders. *IEEE Pervasive Computing*, IEEE, v. 17, n. 3, p. 12–22, 2018. Citado na página 22.

MELO, L. P. de et al. Análise de malware: Investigaçao de códigos ma-liciosos através de uma abordagem prática. *SBSeg*, v. 11, p. 9–52, 2011. Citado na página 11.

MONTEIRO, L. A internet como meio de comunicação: possibilidades e limitações. In: *Congresso Brasileiro de Comunicação*. [S.l.: s.n.], 2001. v. 24. Citado na página 10.

NUNES; RODRIGUES; MOURÃO. *MINISTÉRIO DA DEFESA EXÉRCITO BRASILEIRO DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA INSTITUTO MILITAR DE ENGENHARIA*. 2014. Citado 3 vezes nas páginas 15, 16 e 17.

OLIVEIRA, S. de. *Internet das Coisas com ESP8266, Arduino e Raspberry Pi*. [S.l.]: Novatec Editora, 2017. ISBN 8575225820, 9788575225820. Citado na página 10.

PRADO, M. A. Análise experimental da botnet iot mirai. Universidade Federal de Uberlândia, 2018. 68 f. Trabalho de Conclusão de Curso (Graduação em Sistemas de informação). Citado 3 vezes nas páginas 6, 16 e 22.

SANTOS, B. P. et al. Internet das coisas: da teoria à prática. *Minicursos SBRC-Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos*, 2016. Citado na página 14.

STALLINGS, W. *Network security essentials : applications and standards*. [S.l.: s.n.], 2011. xiii, 366 p. p. ISBN 0130160938. Citado na página 10.

WANG, P. et al. Honeypot detection in advanced botnet attacks. *International Journal of Information and Computer Security*, Inderscience Publishers, v. 4, n. 1, p. 30–51, 2010. Citado na página 15.

Anexos

ANEXO A – CÓDIGO FONTE DA API

A.1 Produto

```
1  package com.miraiScanner.apirest.models;
2
3  import java.io.Serializable;
4  import java.math.BigDecimal;
5
6  import javax.persistence.Entity;
7  import javax.persistence.GeneratedValue;
8  import javax.persistence.GenerationType;
9  import javax.persistence.Id;
10 import javax.persistence.Table;
11 import javax.validation.constraints.NotNull;
12
13
14 @Entity
15 @Table(name="MIRAI_APP")
16 public class Produto implements Serializable{
17
18
19
20         private String enderecoIp;
21
22         private String fabricanteMac;
23
24         @Id
25         private String enderecoMac;
26
27         private String porta23;
28
29         private String porta2323;
30
31         private String porta48101;
32
33         private String usuario;
34
35         private String senha;
```

```
36
37          private String resposta;
38
39          private String suporte;
40
41          public String getResposta() {
42                  return resposta;
43          }
44
45          public void setResposta(String resposta) {
46                  this.resposta = resposta;
47          }
48
49          public String getSuporte() {
50                  return suporte;
51          }
52
53          public void setSuporte(String suporte) {
54                  this.suporte = suporte;
55          }
56
57          public String getSenha() {
58                  return senha;
59          }
60
61          public void setSenha(String senha){
62                  this.senha = senha;
63          }
64
65          public String getUsuario() {
66                  return usuario;
67          }
68
69          public void setUsuario(String usuario) {
70                  this.usuario = usuario;
71          }
72
73          public String getEnderecoIp() {
74                  return enderecoIp;
75          }
76
```

```
77          public void setEnderecoIp(String enderecoIp) {
78                  this.enderecoIp = enderecoIp;
79          }
80
81          public String getFabricanteMac() {
82                  return fabricanteMac;
83          }
84
85          public void setFabricanteMac(String fabricanteMac) {
86                  this.fabricanteMac = fabricanteMac;
87          }
88
89          public String getEnderecoMac() {
90                  return enderecoMac;
91          }
92
93          public void setEnderecoMac(String enderecoMac) {
94                  this.enderecoMac = enderecoMac;
95          }
96
97          public String getPorta23() {
98                  return porta23;
99          }
100
101         public void setPorta23(String porta23) {
102                 this.porta23 = porta23;
103         }
104
105         public String getPorta2323() {
106                 return porta2323;
107         }
108
109         public void setPorta2323(String porta2323) {
110                 this.porta2323 = porta2323;
111         }
112
113         public String getPorta48101() {
114                 return porta48101;
115         }
116
117         public void setPorta48101(String porta48101) {
```

```
                this.porta48101 = porta48101;
        }

}
```

A.2 application.properties

```

spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true

##Banco local - Antonio
spring.datasource.url= jdbc:postgresql://localhost:5432/produtos-apirest
spring.datasource.username=postgres
spring.datasource.password=tilei123
spring.jpa.hibernate.ddl-auto=update
```

A.3 ProdutoRepository Interface

```

package com.miraiScanner.apirest.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import com.miraiScanner.apirest.models.Produto;

public interface ProdutoRepository extends JpaRepository<Produto, Long>{
        Produto findById(long id);

}
```

A.4 ProdutoRepository

```
package com.miraiScanner.apirest.resources;

import java.util.List;

import javax.validation.Valid;

import org.springframework.beans.factory.annotation.Autowired;
```

```
8  import org.springframework.web.bind.annotation.CrossOrigin;
9  import org.springframework.web.bind.annotation.DeleteMapping;
10 import org.springframework.web.bind.annotation.GetMapping;
11 import org.springframework.web.bind.annotation.PathVariable;
12 import org.springframework.web.bind.annotation.PostMapping;
13 import org.springframework.web.bind.annotation.PutMapping;
14 import org.springframework.web.bind.annotation.RequestBody;
15 import org.springframework.web.bind.annotation.RequestMapping;
16 import org.springframework.web.bind.annotation.ResponseBody;
17 import org.springframework.web.bind.annotation.RestController;
18
19 import com.miraiScanner.apirest.models.Produto;
20 import com.miraiScanner.apirest.repository.ProdutoRepository;
21
22 import io.swagger.annotations.Api;
23 import io.swagger.annotations.ApiOperation;
24
25 @CrossOrigin(origins = "*")
26 @RestController
27 @RequestMapping(value="/api")
28 @Api(value="API REST Produtos")
29 public class ProdutoResource {
30
31         @Autowired
32         ProdutoRepository produtoRepository;
33
34         @ApiOperation(value="Retorna uma lista de Produtos")
35         @GetMapping("/produtos")
36         public List<Produto> listaProdutos(){
37                 return produtoRepository.findAll();
38         }
39
40
41         @ApiOperation(value="Salva um produto")
42         @PostMapping("/produto")
43         public Produto salvaProduto(@RequestBody @Valid Produto produto) {
44                 return produtoRepository.save(produto);
45         }
46
47         @ApiOperation(value="Deleta um produto")
48         @DeleteMapping("/produto")
```

```
49          public void deletaProduto(@RequestBody @Valid Produto produto) {
50                  produtoRepository.delete(produto);
51          }
52
53          @ApiOperation(value="Atualiza um produto")
54          @PutMapping("/produto")
55          public Produto atualizaProduto(@RequestBody @Valid Produto produto) {
56                  return produtoRepository.save(produto);
57          }
58
59
60  }
```

ANEXO B – CÓDIGO FONTE DAS CLASSES ADICIONADAS/MODIFICADAS DO APLICATIVO MIRAI SCANNER

B.1 Post - Adicionada

```
1
2  package miraiscanner.facom.ufu.br.miraiscanner.Network;
3
4  // classe responsavel por searealizar componentes para serem gravados no banco de dados
5
6
7  import com.google.gson.annotations.Expose;
8  import com.google.gson.annotations.SerializedName;
9
10
11
12
13 public class Post {
14
15     @SerializedName("enderecoIp")
16     @Expose
17     private String enderecoIp;
18
19     @SerializedName("fabricanteMac")
20     @Expose
21     private String fabricanteMac;
22
23     @SerializedName("enderecoMac")
24     @Expose
25     private String enderecoMac;
26
27     @SerializedName("porta23")
28     @Expose
29     private String porta23;
30
31     @SerializedName("porta2323")
```

```
32      @Expose
33      private String porta2323;
34
35      @SerializedName("porta48101")
36      @Expose
37      private String porta48101;
38
39      @SerializedName("usuario")
40      @Expose
41      private String usuario;
42
43      public String getEnderecoIp() {
44          return enderecoIp;
45      }
46
47      public void setEnderecoIp(String enderecoIp) {
48          this.enderecoIp = enderecoIp;
49      }
50
51      public String getFabricanteMac() {
52          return fabricanteMac;
53      }
54
55      public void setFabricanteMac(String fabricanteMac) {
56          this.fabricanteMac = fabricanteMac;
57      }
58
59      public String getEnderecoMac() {
60          return enderecoMac;
61      }
62
63      public void setEnderecoMac(String enderecoMac) {
64          this.enderecoMac = enderecoMac;
65      }
66
67      public String getPorta23() {
68          return porta23;
69      }
70
71      public void setPorta23(String porta23) {
72          this.porta23 = porta23;
```

```
73      }
74
75      public String getPorta2323() {
76          return porta2323;
77      }
78
79      public void setPorta2323(String porta2323) {
80          this.porta2323 = porta2323;
81      }
82
83      public String getPorta48101() {
84          return porta48101;
85      }
86
87      public void setPorta48101(String porta48101) {
88          this.porta48101 = porta48101;
89      }
90
91      public String getUsuario() {
92          return usuario;
93      }
94
95      public void setUsuario(String usuario) {
96          this.usuario = usuario;
97      }
98
99      public String getSenha() {
100         return senha;
101     }
102
103     public void setSenha(String senha) {
104         this.senha = senha;
105     }
106
107     public String getResposta() {
108         return resposta;
109     }
110
111     public void setResposta(String resposta) {
112         this.resposta = resposta;
113     }
```

```
114
115     public String getSuporte() {
116         return suporte;
117     }
118
119     public void setSuporte(String suporte) {
120         this.suporte = suporte;
121     }
122
123     @SerializedName("senha")
124     @Expose
125     private String senha;
126
127
128     @SerializedName("resposta")
129     @Expose
130     private String resposta;
131
132     @SerializedName("suporte")
133     @Expose
134     private String suporte;
135
136 }
```

B.2 RetrofitClient - Adicionada

```
1
2 package miraiscanner.facom.ufu.br.miraiscanner.Network;
3
4 import retrofit2.Retrofit;
5 import retrofit2.converter.gson.GsonConverterFactory;
6
7 public class RetrofitClient {
8
9     private static Retrofit retrofit = null;
10
11     public static Retrofit getClient(String baseUrl) {
12         if (retrofit==null) {
13             retrofit = new Retrofit.Builder()
14                     .baseUrl(baseUrl)
15                     .addConverterFactory(GsonConverterFactory.create())
```

```
                        .build();
        }
        return retrofit;
    }
}
```

B.3 ApiUtils - Adicioanda

```

package miraiscanner.facom.ufu.br.miraiscanner.Network;

public class ApiUtils {

    private ApiUtils() {}

    public static final String BASE_URL = "https://api-miraiapp.herokuapp.com/" ;

    public static APIService getAPIService() {

        return RetrofitClient.getClient(BASE_URL).create(APIService.class);
    }
}
```

B.4 Interface APIService - Adicionada

```

package miraiscanner.facom.ufu.br.miraiscanner.Network;

import miraiscanner.facom.ufu.br.miraiscanner.Network.Post;
import retrofit2.Call;
import retrofit2.http.Body;
import retrofit2.http.Field;
import retrofit2.http.FormUrlEncoded;
import retrofit2.http.POST;

public interface APIService {

    //"/api/produto"


```

```
16      @POST("api/produto")
17          //@FormUrlEncoded
18      Call<Post> savePost(@Body Post post);
19
20  }
```

B.5 DispositivoActivity - Modificada

```
1
2  package miraiscanner.facom.ufu.br.miraiscanner.Activity;
3
4  import android.content.Context;
5  import android.content.Intent;
6  import android.content.res.ColorStateList;
7  import android.graphics.Color;
8  import android.support.v4.content.ContextCompat;
9  import android.support.v4.widget.ImageViewCompat;
10 import android.support.v7.app.ActionBar;
11 import android.support.v7.app.AppCompatActivity;
12 import android.os.Bundle;
13 import android.util.Log;
14 import android.view.MenuItem;
15 import android.view.View;
16 import android.widget.ImageView;
17 import android.widget.LinearLayout;
18 import android.widget.TextView;
19
20 import org.w3c.dom.Text;
21
22 import miraiscanner.facom.ufu.br.miraiscanner.Model.Dispositivo;
23 import miraiscanner.facom.ufu.br.miraiscanner.Model.DispositivoResponse;
24 import miraiscanner.facom.ufu.br.miraiscanner.Network.APIService;
25 import miraiscanner.facom.ufu.br.miraiscanner.Network.Post;
26 import miraiscanner.facom.ufu.br.miraiscanner.R;
27 import retrofit2.Call;
28 import retrofit2.Callback;
29 import retrofit2.Response;
30 import retrofit2.Retrofit;
31 import retrofit2.converter.gson.GsonConverterFactory;
32
33 public class DispositivoActivity extends AppCompatActivity {
```

```
34      private Dispositivo dispositivo;
35      private DispositivoResponse dispositivos;
36      private String nomeRede;
37
38      private String enderecoIp;
39
40      private String fabricanteMac;
41
42
43      private String enderecoMac;
44
45
46      private String porta23;
47
48
49      private String porta2323;
50
51
52      private String porta48101;
53
54
55      private String usuario;
56
57
58      private String senha;
59      private String suporte;
60      private String resposta;
61      private String testando;
62
63
64      @Override
65      protected void onCreate(Bundle savedInstanceState) {
66          super.onCreate(savedInstanceState);
67          setContentView(R.layout.activity_dispositivo);
68          Intent it = this.getIntent();
69
70
71          if(it.getSerializableExtra("dispositivos") != null) {
72              DispositivoResponse dispositivoResponse = (DispositivoResponse)
73                      it.getSerializableExtra("dispositivos");
74              this.dispositivos = dispositivoResponse;
```

```
75
76          }
77          if(it.getStringExtra("nome_rede") != null) {
78              this.nomeRede = it.getStringExtra("nome_rede");
79
80          }
81          if(it.getSerializableExtra("dispositivo") != null) {
82              Dispositivo disp = (Dispositivo) it.getSerializableExtra("dispositivo");
83              this.dispositivo = disp;
84
85          }
86          else{
87              dispositivo = new Dispositivo("0.0.0.0", "00:00:00:00:00:00");
88
89          }
90
91          //Painel de informações
92          TextView txt_nome = (TextView) findViewById(R.id.txt_nome);
93
94          TextView txt_ip = (TextView) findViewById(R.id.txt_ip);
95          TextView txt_fabricante = (TextView) findViewById(R.id.txt_fabricante);
96
97          TextView txt_mac = (TextView) findViewById(R.id.txt_mac);
98
99          txt_nome.setText(dispositivo.getNome());
100         txt_ip.setText(dispositivo.getIp());
101         txt_fabricante.setText(dispositivo.getFabricante());
102         txt_mac.setText(dispositivo.getMac());
103
104
105         enderecoIp = dispositivo.getIp();
106         fabricanteMac = dispositivo.getFabricante();
107         enderecoMac = dispositivo.getMac();
108
109         //Painel de ameaças
110         TextView txt_23 = (TextView) findViewById(R.id.txt_23);
111         TextView txt_2323 = (TextView) findViewById(R.id.txt_2323);
112         TextView txt_48101 = (TextView) findViewById(R.id.txt_48101);
113         TextView txt_usuario = (TextView) findViewById(R.id.telnet_usuario);
114         TextView txt_senha = (TextView) findViewById(R.id.telnet_senha);
115         LinearLayout layout_telnet = (LinearLayout) findViewById(R.id.layout_telnet);
```

```
116
117         boolean vulneravel = false;
118
119         if(dispositivo.getPorta23Aberta()){
120
121             porta23 = "\"Aberta, possiveis vulnerabilidades foram encontradas em seu dispositivo que podem esta
122                     " \"neste dispositivo.\"";
123
124
125             txt_23.setText("Aberta, Altere as senhas de acesso e reinicie o dispositivo.");
126             txt_23.setTextColor(Color.parseColor("#ffbb33"));
127             vulneravel = true;
128             txt_usuario.setText(dispositivo.getUsuario());
129             txt_senha.setText(dispositivo.getSenha());
130             layout_telnet.setVisibility(View.VISIBLE);
131         }
132         else{
133             txt_23.setText("Fechada");
134             porta23 = "fechada";
135         }
136
137         if(dispositivo.getPorta2323Aberta()){
138             porta2323 = "\"Aberta, possiveis vulnerabilidades foram encontradas em seu dispositivo que podem e
139                     " \"neste dispositivo.\"";
140             txt_2323.setText("Aberta, Altere as senhas de acesso e reinicie o dispositivo.");
141             txt_2323.setTextColor(Color.parseColor("#ffbb33"));
142             vulneravel = true;
143             if(!dispositivo.getPorta23Aberta()){
144                 txt_usuario.setText(dispositivo.getUsuario());
145                 txt_senha.setText(dispositivo.getSenha());
146                 layout_telnet.setVisibility(View.VISIBLE);
147             }
148         }
149         else{
150
151             txt_2323.setText("Fechada");
152             porta2323 = "Fechada";
153         }
154
155         if(dispositivo.getPorta48101Aberta()){
156             porta48101 = "\"Aberta, possiveis vulnerabilidades foram encontradas em seu dispositivo que podem
```

```
157                     " \"neste dispositivo.\"";
158             txt_48101.setText("Aberta, Altere as senhas de acesso e reinicie o dispositivo.");
159             txt_48101.setTextColor(Color.parseColor("#ffbb33"));
160             vulneravel = true;
161         }
162         else{
163             txt_48101.setText("Fechada");
164             porta48101 = "fechada";
165         }
166
167         TextView txt_ameacas = (TextView) findViewById(R.id.txt_ameacas);
168         ImageView img_ameacas = (ImageView) findViewById(R.id.img_ameacas);
169
170         if(vulneravel){
171             suporte = "\"Foram encontradas vulnerabilidades em seu dispositivo que podem \" +\n" +
172                     " \"ser exploradas pelo Mirai. Altere as senhas de acesso e reinicie o dispositivo.\"";
173             txt_ameacas.setText("Foram encontradas vulnerabilidades em seu dispositivo que podem " +
174                     "ser exploradas pelo Mirai. Altere as senhas de acesso e reinicie o dispositivo.");
175             img_ameacas.setImageResource(R.drawable.twotone_warning_24);
176
177         }
178         else{
179             suporte = "\"Não foi encontrada nenhuma vulnerabilidade relacionada ao Mirai \" +\n" +
180                     " \"neste dispositivo.\"";
181             txt_ameacas.setText("Não foi encontrada nenhuma vulnerabilidade relacionada ao Mirai " +
182                     ".neste dispositivo");
183             img_ameacas.setImageResource(R.drawable.twotone_check_circle_24);
184             ImageViewCompat.setImageTintList(img_ameacas, ColorStateList.valueOf(ContextCompat
185                     .getColor(DispositivoActivity.this, R.color.checked)));
186
187         }
188
189
190         //Botão voltar da barra superior
191         ActionBar ts = getSupportActionBar();
192         if(ts != null)
193             ts.setDisplayHomeAsUpEnabled(true);
194
195
196         Retrofit retrofit = new Retrofit.Builder()
197                 .baseUrl("https://api-miraiapp.herokuapp.com/")
```

```
198                 .addConverterFactory(GsonConverterFactory.create())
199                 .build();
200
201         APIService service = retrofit.create(APIService.class);
202
203
204         Post post = new Post();
205         post.setEnderecoIp(enderecoIp);
206         post.setEnderecoMac(enderecoMac);
207         post.setFabricanteMac(fabricanteMac);
208         post.setPorta23(porta23);
209         post.setPorta2323(porta2323);
210         post.setPorta48101(porta48101);
211         post.setSuporte(suporte);
212         post.setSenha(senha);
213         post.setUsuario(usuario);
214
215
216
217         Call<Post> Post = service.savePost(post);
218         Post.enqueue(new Callback<miraiscanner.facom.ufu.br.miraiscanner.Network.Post>() {
219             @Override
220             public void onResponse(Call<miraiscanner.facom.ufu.br.miraiscanner.Network.Post> call, Response<m
221                 int status = response.code();
222
223                 Post resposta = response.body();
224
225                 Log.i("resposssta","On response" + status);
226             }
227
228             @Override
229             public void onFailure(Call<miraiscanner.facom.ufu.br.miraiscanner.Network.Post> call, Throwable t) {
230
231             }
232         });
233
234
235     }
236
237
238     @Override
```

```
239     public boolean onOptionsItemSelected(MenuItem item) {
240         int id = item.getItemId();
241
242
243         System.out.printf("onOptionsItemSelected");
244         if (id == android.R.id.home) {
245
246
247             Intent intent = new Intent(DispositivoActivity.this, MainActivity.class);
248             intent.putExtra("dispositivos", this.dispositivos);
249             intent.putExtra("nome_rede", this.nomeRede);
250             startActivity(intent);
251
252
253             return true;
254         }
255
256         return super.onOptionsItemSelected(item);
257     }
258 }
```