Arquitetura de IPS para redes IoT sobrepostas em SDN

Daniel G. V. Gonçalves[1], Guilherme de O. Kfouri[1], Bruno V. Dutra[1], João F. de Alencastro[1], Francisco L. de Caldas Filho[1], Lucas M. C. e Martins[1], Robson de O. Albuquerque[1], Rafael T. de Sousa Jr.[1]

[1] Instituto Nacional de Ciência e Tecnologia em Segurança Cibernética
Departamento de Engenharia Elétrica – Universidade de Brasília (UnB)
Caixa Postal 4466 – 70.910-900 – Brasília – DF – Brasil

daniel.goncalves@redes.unb.br, guiokfouri@gmail.com,

{bruno.dutra,joao.alencastro,francisco.lopes,lucas.martins,robson,

rafael.desousa}@redes.unb.br

Abstract. *The programmability resulting from the Software Defined Networking (SDN) approach facilitates the integration of the functionalities of firewalls, Intrusion Prevention Systems (IPS) and switching gear, allowing fast reconfiguration of the network in case of anomaly detection. In this paper, the proposed architecture aims to structure a distributed security measure integrating firewall, IPS, switches and a controller entity to support Internet of Things (IoT) instances, allowing the identification of anomalous behavior of IoT devices by the IPS, thus leading the SDN to block the attacks as near as possible to the sources, reducing the volume of malicious traffic and isolating the infected device from the rest of the network.*

Resumo. *A programabilidade resultante da abordagem Software Defined Networking (SDN) facilita a integração das funcionalidades de firewalls, sistemas de prevenção de intrusão (IPS) e switches, permitindo a rápida reconfiguração da rede em caso de detecção de anomalias. Neste artigo, a arquitetura proposta visa estruturar uma medida de segurança distribuída que integra firewall, IPS, switches e uma entidade controladora para suportar instâncias de Internet das Coisas (IoT), permitindo a identificação de comportamento anômalo de dispositivos IoT pelo IPS, levando então a SDN a bloquear os ataques o mais próximo possível das fontes, reduzindo o volume de tráfego malicioso e isolando o dispositivo infectado do resto da rede..*

1. Introdução

A presença cada vez maior de redes redes móveis e de instâncias de Internet das Coisas (do Inglês, *Internet of Things* – IoT) em todo o mundo, concomitante ao intenso aumento da quantidade de dispositivos conectados, tem várias consequências como o aumento do volume de dados processados e enviados pela rede, e a complexidade do gerenciamento de fluxos de dados heterogêneos para múltiplas aplicações simultâneas e paralelas. Outra consequência é o aumento do número de dispositivos que podem ser explorados para

fins maléficos [Kolias et al. 2017], uma vez que a grande maioria possui sérias vulnerabilidades de configuração de fábrica e carências quanto a medidas e configurações de segurança. Em tal situação, a tarefa de gerenciar a rede e a infraestrutura de segurança necessária para abrigar tais dispositivos necessita de novas abordagens.

Por outro lado, problemas de complexidade e escalabilidade de redes vêm motivando a introdução de um novo paradigma para gerência e controle na forma de redes definidas por software (do Inglês, *Software Defined Networking* – SDN) [Kreutz et al. 2015]. Tal paradigma, que objetiva, entre outros benefícios, aumentar a funcionalidade da rede, reduzir gastos e reduzir a complexidade de *hardware*, propõe a centralização de atividades de controle, usando uma entidade (*controller*) que lida com as decisões de encaminhamento do tráfego por meio de uma abstração chamada de fluxo (*flows*), realizando a configuração de equipamentos (*switches*) que comutam os pacotes que representam os diversos fluxos.

Já no que se refere à segurança das redes, um importante desenvolvimento é aquele relativo aos sistemas de prevenção/detecção de intrusão (do Inglês, *Intrusion Prevention/Detection System* – IPS/IDS) que constituem medidas preventivas e de reação a incidentes que impactam as redes. Assim como vem ocorrendo desde sua introdução [Roesch 1999], em instâncias de redes IoT tais sistemas continuam se mostrando de grande interesse, pois ataques como o da *botnet* Mirai [Kolias et al. 2017] mostram que a segurança em dispositivos IoT ainda possui necessidades que devem ser sanadas para o correto funcionamento das aplicações que dependem dos dados de tais dispositivos. Porém, sanar os problemas de segurança de instâncias IoT não é algo simples de se realizar, uma vez que a quantidade de entidades e a capacidade de processamento tornam isso uma tarefa difícil [Kolias et al. 2017].

Para a configuração de instâncias de redes IoT, tecnologias como *Cloud Computing* [Mell and Grance 2011] são utilizadas exatamente para suprir a falta de poder computacional necessário para o processamento dos dados advindos dos dispositivos IoT [Gubbi et al. 2013]. Entretanto, por mais que a *Cloud Computing* seja grande facilitadora por disponibilizar recursos computacionais sob demanda, ela se torna um problema para aplicações que precisam de baixa latência na transmissão de dados [Bonomi et al. 2012], como é o caso de serviços que dependem de dados advindos de sensores IoT. Para atacar tal problema, surgiram as ideias de *Edge* e *Fog Computing*, definindo a existência de entidades de processamento/tomada de decisões que estejam mais próximas geograficamente das fontes e destinos finais [Bonomi et al. 2012].

Observando tal evolução da SDN para tratar dos problemas de escalabilidade e gerenciamento, além da contribuição da SDN para a segurança da computação em nuvem [de Jesus et al. 2014], bem como considerando os requisitos do processamento de dados com baixa latência de transmissão e as necessidades de segurança nas redes IoT [Ferreira and de Sousa Jr 2017], este artigo propõe uma medida distribuída de segurança para mitigação de ataques a instâncias de IoT, contando com a infraestrutura de rede SDN e capacitada a operar com baixa latência para tomada de decisão no caso de eventos de segurança.

Esse trabalho é dividido da seguinte maneira: a seção 2 traz trabalhos relacionados de segurança em redes IoT que associam paradigmas de SDN e IDS. A seção 3 apresenta

a proposta deste artigo, incluindo a arquitetura de segurança, metodologia de aplicação de políticas de contenção de ataques e interação das entidades da solução. A seção 4 apresenta e discute os resultados obtidos no ambiente montado para testes da proposta. Por fim, na seção 5, é apresentada a conclusão e indicações para trabalhos futuros.

2. Trabalhos Relacionados

Visando acompanhar o progresso nas tecnologias de redes de computadores, várias soluções de segurança para IoT utilizando a arquitetura SDN e seus mecanismos de gerenciamento já foram propostas.

Em [Jin and Wang 2013] apresenta-se uma arquitetura de IPS para dispositivos mobile que utiliza a flexibilidade de configuração da infraestrutura de rede SDN para realizar a configuração de políticas de segurança. Em tal arquitetura é proposta uma topologia onde os dispositivos estão conectados a um *switch* SDN que por sua vez é gerenciado por uma Controller onde são executados os módulos de detecção de *malware*. Tais módulos de detecção são algoritmos que identificam comportamentos maliciosos nos pacotes ou conexões da rede. Tais algoritmos trabalham com as ideias de: *blacklisting* de IP's maliciosos, análise do percentual de sucesso de conexão de um dispositivo, quantidade de conexões requisitadas por um dispositivo e análise de comportamento de dispositivos para enfim identificar na rede um dispositivo comprometido e realizar a configuração de regras no *switch* que mitiguem a propagação do ataque. Por mais que as análises feitas pelos algoritmos estejam bem fundamentadas no comportamento de dispositivos comprometidos tal solução é limitada na identificação de ataques específicos por trabalhar com métricas de identificação muito amplas. O Snort, IDS utilizado como mecanismo de detecção de nossa solução proposta, possui um amplo leque de assinaturas disponíveis e uma comunidade ativa que disponibiliza regras para ataques atuais, promovendo assim a detecção de ataques ou tráfego indesejado cujo comportamento não é necessariamente padrão.

Em [Bull et al. 2016], é proposto um sistema de segurança de IPS para redes IoT onde o *gateway* IoT é substituído por um *gateway* SDN. O sistema realiza a análise da anomalia nos *flows* criados para lidar com o tráfego advindo dos dispositivos, visando a detecção de tráfego anômalo/malicioso e realização das ações necessárias para a mitigação do ataque (bloqueio, encaminhamento, aplicação de *QoS* (*Quality of Service*)). Tal solução por sua vez é capaz de atuar na infraestrutura da rede, mitigando a expansão do ataque. Porém, não possui maneira de proteger as entidades finais/*hosts* da topologia, e depende do bom funcionamento do mecanismo de detecção de anomalia para evitar a não detecção como também os falsos positivos, falsos positivos não são um problema muito frequente em ferramentas baseadas em detecção por assinatura, tal método de detecção é um dos pontos centrais de nossa proposta.

Em redes IoT existem vários dispositivos que realizam as mais diversas funcionalidades que por sua vez se comunicam da maneira que o fabricante achou mais apropriado, o que acaba por gerar dificuldades na integração dos mesmos com o restante da rede. Para isso o IETF (*Internet Engineering Task Force*) definiu o MUD (*Manufacturer Usage Description*) [Lear et al. 2019], um padrão para softwares embarcados anunciarem os parâmetros de sua comunicação entre outras informações de seu funcionamento. Visando trabalhar em conjunto com esse novo paradigma [Hamza et al. 2018] propôs uma solução de segurança IPS para redes IoT que trabalha diretamente com os perfis MUD de

dispositivos inteligentes para realizar a tradução de suas ACE's (*Access Control Entries*) para regras de *flow* em *switches* SDN. Dessa maneira a infraestrutura de rede SDN se adéqua ao comportamento previamente definido do dispositivo IoT visando conter o tráfego malicioso endereçado a ele ou até mesmo vindo dele. Tal solução visa proteger os dispositivos IoT da rede e não é capaz de manter a integridade de outras entidades (*Gateway* IoT, *Hosts* Linux e etc) pertencentes a mesma, algo que será mostrado adiante como uma característica de nossa proposta.

Em [Cho et al. 2009], foi proposta uma solução para uma arquitetura que realiza a análise de pacotes que passam pelo roteador de borda. O trabalho tem como foco a mitigação de ataques de negação de serviços utilizando *botnets* em instâncias IoT [Kolias et al. 2017]. Para a correta implementação de tal solução é necessário que algumas premissas sejam satisfeitas: 1) Os dispositivos utilizem TCP para disponibilizar seus serviços/dados; 2) Um dispositivo 6LoWPAN ofereça somente um serviço; 3) Que o procedimento de comunicação entre o sensor e o usuário seja típico. Com tais condições atendidas, dispositivos de coleta IDS são espalhados pela instância IoT para realizar o *sniffing* de tráfego para envio posterior a uma entidade de IDS central para análise. A solução proposta necessita enviar as detecções de IDSs espalhados pela rede para uma entidade central de IDS gerar alertas, enquanto a nossa solução proposta envia os alertas da detecção para uma entidade capaz de realizar a prevenção do ataque na infraestrutura e nos *hosts* da rede.

Uma maneira de implementar IDSs de topologia híbrida em redes IoT consiste do posicionamento de IDSs tanto no roteador de borda quanto em outros nós da rede. A principal diferença nessa maneira de implementação é a presença de um componente central, com o IDS no roteador de borda realizando a análise de situações mais exigentes e com os dispositivos da rede tendo uma implementação leve da detecção. Uma realização híbrida utilizando um IDS chamado SVELTE foi proposta em [Raza et al. 2013]. Nesse trabalho, os roteadores de borda executam módulos do IDS que precisam de uma quantidade elevada de recursos, enquanto os dispositivos realizam operações leves como o envio do tráfego dos dispositivos para o roteador de borda. Assim como em [Bull et al. 2016], a solução proposta por [Raza et al. 2013] realiza a proteção da rede por meio da análise do tráfego, mas não impõe políticas de segurança automaticamente, deixando como responsabilidade do administrador a configuração das mesmas. Dependendo do tempo de resposta do administrador ou entidade terceira a integridade da rede e de seus serviços pode ser comprometida. Em nossa proposta a detecção de tráfego malicioso implica na aplicação de políticas de proteção tanto na infraestrutura da rede quanto nos *hosts* finais da mesma, proporcionando assim o bloqueio da disseminação do ataque na rede como também dificulta o comprometimento dos dispositivos finais da mesma, para que assim eles não se tornem propagadores do ataque.

Em [Kasinathan et al. 2013], foi proposta uma solução que realiza a integração de IDSs baseados em detecção por assinatura em redes IoT. O foco da solução é a detecção de ataques de negação de serviço em redes 6LoWPAN, com um IDS desenvolvido pelos autores adaptando o Suricata para tal. O software adaptado envia os alertas de detecção de ataques de negação de serviço para uma entidade denominada gerente de proteção que realiza uma série de análises adicionais para confirmar a existência de uma ataque. Tais passos adicionais no gerente de proteção são feitos para diminuir o número de falsos posi-

tivos gerados pela solução. Esta foi desenhada para permitir a utilização em computadores com sistema operacional Linux, para que assim sejam evitados problemas relacionados com a limitação de recursos computacionais. Tal solução por sua vez se encontra limitada à detecção de ataques de negação de serviço. Mais a frente, mostramos que a nossa solução utiliza o Snort IDS que possui um amplo leque de regras de assinatura disponibilizadas pela comunidade para detecção dos mais diversos ataques na rede. O Snort também provê a capacidade de o administrador escrever as próprias regras para definir o comportamento que não espera de sua rede, tal programabilidade da detecção é incorporada em nossa solução.

3. Proposta de Arquitetura de IPS para redes IoT sobrepostas em SDN

Levando em consideração os trabalhos correlatos, este trabalho propõe um IPS para redes IoT, integrado com o controlador SDN e os *switches* que estruturam a rede. Conforme mostra a Figura 1, arquitetura aqui proposta abrange a mitigação de eventos de segurança em cada rede local da seguinte forma:

1. Na infraestrutura de rede SDN: quando o tráfego malicioso for identificado na rede local, originado por um dispositivo IoT infectado, então serão criadas políticas de bloqueio nos dispositivos gerenciados (*switches* e roteadores) para impedir a proliferação do ataque;
2. Nos *gateways* IoT: quando tráfego malicioso for identificado na rede, serão criadas regras da *firewall* iptables no sistema do *gateway* IoT/*hosts* Linux da rede.

Tais medidas visam atenuar ou até mesmo parar ataques na rede local, uma vez que a ação dos dispositivos comprometidos é bloqueada por regras aplicadas nos *hosts* Linux e *gateways* IoT que poderiam ser utilizados para propagar o ataque de dentro da rede.

Para prover tais funcionalidades, a arquitetura proposta é composta por dois grandes componentes: o módulo de detecção e o módulo de proteção. O módulo de detecção é responsável pela identificação do tráfego malicioso utilizando o IDS *Snort* que dispõe de um grande número de regras previamente criadas pela comunidade que são capazes de identificar um leque amplo de tráfego malicioso, além de possuir a capacidade de implementação de novas regras que estejam de acordo com as necessidades da rede do usuário [Roesch 1999]. O módulo de proteção é formado pelo *Controller*, *SystemAPI* e *SnortAPI*. Estes componentes e a sua interação são descritos nas demais subseções a seguir.

A proposta atua no ambiente de redes locais tanto no que se refere à elementos de instância de rede IoT quanto a dispositivos de rede SDN. Considerando a topologia da Figura 1 como a ideal para utilização da arquitetura proposta, pode-se imaginar que tal figura representa a situação de uma empresa que possua uma série de sensores para automação e um ou mais *switches* gerenciáveis, afim de facilitar a exposição nós assumimos apenas um *switch* gerenciável em funcionamento na rede.

Nesse contexto, o pressuposto é que a rede pode ter variados tipos de tráfego, incluindo consultas *web*, requisições DNS, etc. O Snort deverá estar conectado ao *switch* de distribuição ou núcleo, de modo a receber todo o fluxo de dados da rede por espelhamento de portas. Com base nesses dados, serão feitas análises por meio de regras de

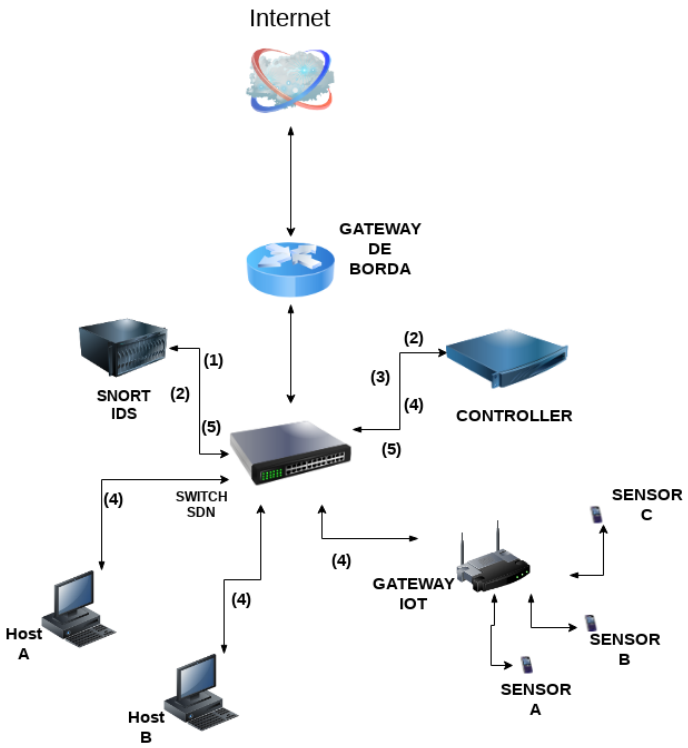

Figura 1. Topologia proposta

assinatura previamente cadastradas. A aplicação de controle da nossa proposta poderá solicitar o bloqueio do fluxo de dados ou portas para qualquer *switch* da rede, independente da camada em que ele esteja operando.

3.1. Interação entre as Entidades

A estrutura distribuída da proposta é formada por três componentes:

1. *Controller*: responsável pelo gerenciamento e manutenção do ecossistema de segurança, bem como o controle dos dispositivos gerenciáveis da rede local;
2. *SystemAPI*: API REST responsável pela configuração de *firewall* sob demanda no *host* em que se encontra (*host* Linux ou *gateway* IoT);
3. *SnortAPI*: API REST responsável pela configuração de regras de detecção sob demanda no Snort da rede.

As entidades *Controller*, *SystemAPI* e *SnortAPI* devem se comunicar para manter suas configurações a par da situação atual da rede. As aplicações *SystemAPI* e *SnortAPI* não conversam diretamente entre si, de forma que o nó central de toda comunicação e gerenciamento da solução é o *Controller*, por quem passam todas as informações e configurações para rede.

A Figura 1 ilustra as comunicações entre as entidades, ou seja: (1) Espelhamento de tráfego para análise no Snort. (2) Envio de alertas de segurança para o *Controller*. (3) Envio pelo *Controller* das configurações de bloqueio para dispositivos gerenciados SDN. (4) Envio de configurações de *firewall* para *hosts*/*gateways* IoT cadastrados. (5) Criação de novas regras no Snort via API.

A comunicação entre as entidades requer inicialmente o cadastro de todas as instâncias de *SystemAPIs* e *SnortAPIs* junto ao *Controller*, sendo o processo de cadastro destinado à autenticação entre as entidades. Nesse processo, as *SystemAPIs* e *SnortAPIs* comunicam-se com a API REST de registro existente no *Controller*, de modo que este último repasse as configurações necessárias e dê provimentos para autenticação da interação entre entidades, etapas que são detalhadas na subseção 3.2.

Já a identificação de tráfego malicioso será a responsabilidade do IDS que em nossa primeira escolha é o sistema Snort. Conforme ilustra a Figura 1, tendo identificado tráfego malicioso, o Snort envia um alerta *syslog* para o *Controller* com as informações sobre o atacante (IP de origem, IP destino, porta destino). Com essas informações, o *Controller* cadastra o evento em sua base de dados e inicializa o repasse de configurações de segurança aos dispositivos cadastrados, sendo então para os *hosts*/*gateways* IoT adicionadas regras de bloqueio no *iptables*, enquanto que para os dispositivos SDN são configuradas novas políticas de repasse para o tráfego com as informações do atacante.

À medida que novos ataques são identificados pelos módulo de IDS da rede, as configurações de segurança são impostas pelo *Controller* e como consequência a segurança em cada rede local se adapta a situação atual no que se refere à proteção das entidades e informações que fluam por tais entidades.

3.2. Autenticação entre entidades

Conforme já comentado, é necessário realizar o cadastro das *SystemAPIs* e *SnortAPIs* junto ao *Controller*, processo que ocorre durante a configuração de tais entidades.

A interoperação do *Controller* e das *SystemAPIs* e *SnortAPIs* tem base em APIs REST que se encontram em execução nessas entidades. Para configurar tais APIs REST, é necessário especificar informações como: endereço em que a API aguarda conexão, porta a qual o serviço estará associado, um nome associado à API e por fim uma interface de rede existente na máquina em que a API se encontra. No momento em que a interface de rede associada a API é configurada, um processo de criação de chave de autenticação é iniciado. Tal processo parte do endereço MAC da interface cadastrada e a partir dele realiza uma série de processos pseudo-aleatórios de embaralhamento de caracteres para gerar uma *string* que por sua vez é passada a uma função de *hash*, o resultado é salvo no banco de dados local da aplicação e é utilizado como chave estática de autenticação para comunicação com aquela entidade. Esse processo de criação de chaves de autenticação existe em todas as entidades da arquitetura de segurança: *Controller*, *SystemAPI* e *SnortAPI*.

A autenticação portanto é feita com a chave de confiança gerada durante o processo de configuração inicial das aplicações. Assim, ao utilizar tais chaves de confiança, as *SystemAPIs* e *SnortAPIs* só responderão a requisições da Controller, impedindo configurações maliciosas de dispositivos não confiáveis. Na Seção 3.5 será descrita a troca de chaves entre entidades.

3.3. SystemAPI

A SystemAPI é uma API REST em execução nos *hosts* Linux e *gateways* IoT existentes em cada rede local. Tal API é responsável pela configuração de regras do *firewall* iptables nos *hosts* em que se encontra, a partir de requisições autenticadas do *Controller*.

Afim de realizar tais configurações de *firewall*, a SystemAPI precisa receber informações sobre o que deve configurar. A Tabela 1 mostra os quatro campos que devem necessariamente ser passados pelo *Controller*. O campo *Rule* define as informações de tráfego que a regra aborda, possui por sua vez várias opções que estão descritas na Tabela 2]. Esse conjunto de informações para criação da regra é assim utilizado para identificar e tratar o tráfego malicioso que passa pela rede local.

Tabela 1. Parâmetros Principais para SystemAPI

Table	Determina a tabela em que a configuração será feita.
Chain	Determina a cadeia em que a configuração será feita.
Action	Determina a ação realizada para com a regra.
Rule	Informações sobre o que a regra ira abordar.

Tabela 2. Opções de Regras para a SystemAPI

Destination	Endereço IP destino do pacote.
Source	Endereço IP origem do pacote.
Interface IN	Interface entrante do pacote.
Interface OUT	Interface sainte do pacote.
Protocol	Protocolo associado ao pacote.
Porta Origem	Porta origem associada ao pacote.
Porta Destino	Porta destino associada ao pacote.
SYN	Identificação TCP SYN.
TCP Flags	*Flags* do TCP associadas ao pacote.
Jump	Ação tomada para um pacote que corresponde às opções.

Após receber do *Controller* uma requisição de segurança com os parâmetros apresentados acima, a configuração de uma regra na SystemAPI será realizada. A fim de manter as configurações de segurança em sintonia com a situação atual da rede, a SystemAPI executa em paralelo um programa que realiza a remoção das regras expiradas. Quando uma regra é cadastrada na SystemAPI, ela é inserida com um valor de tempo em *Unix Epoch Milliseconds* que representa o horário em que ela chegou na aplicação. Um processo separado verifica se uma regra está expirada e a remove do sistema, considerando que a duração padrão de uma regra cadastrada é de duas horas. Quando tal período passar, a regra será excluída tanto da *iptables* quando da base de dados local da SystemAPI.

3.4. SnortAPI

A SnortAPI é uma API REST em execução na máquina onde o IDS (Snort) se encontra na rede local, sendo responsável pela criação de regras de identificação de tráfego por assinatura no Snort, a partir de requisições autenticadas do *Controller*. Para realizar essa configuração, a SnortAPI possui dois campos principais: "*rule*" e "*action*", conforme apresentado na Tabela 3.

Após receber do *Controller* uma requisição de configuração com os parâmetros mostrados acima, a configuração de uma regra no Snort pela SnortAPI será realizada,

Tabela 3. Parâmetros Principais para a SnortAPI

Rule	A regra literal que se deseja cadastrar.
Action	Determina a ação realizada para com a regra.

cabendo observar que é um procedimento essencial para evitar tipos de tráfego que o administrador não deseja permitir em sua rede. A maleabilidade que a SnortAPI provê proporciona um ambiente de rede onde o tipo de tráfego malicioso pode ser definido sob demanda do administrador.

3.5. *Controller*

O *Controller* é a entidade central na arquitetura de segurança proposta, sendo responsável por gerenciar as SystemAPIs por meio da criação de regras de *firewall* nos *hosts* finais/*gateways* IoT, e as SnortAPIs através da configuração de assinaturas de detecção no IDS da rede. O gerenciamento de tais entidades presentes na rede local faz com que elas se adaptem dinamicamente diante dos eventos de segurança que são detectados. Considerando, conforme a subseção 3.2, que as entidades distribuídas SystemAPIs e SnortAPIs se autenticam com o *Controller*, é interessante detalhar a API de cadastro e as demais funcionalidades desta entidade central.

Afim de realizar o cadastro das entidades presentes na rede, o *Controller* necessita receber informações para proceder com a autenticação. Uma requisição de criação de relação de confiança recebida pelo *Controller* possui os quatro campos descritos na Tabela 4. Com tais campos, o *Controller* realiza na base de dados local o cadastro de cada API distribuída e envia para cada uma sua chave de autenticação, tornando cada uma um objetivo de configurações de segurança/regras de detecção de tráfego.

Tabela 4. Parâmetros para autenticação com o *Controller*

API Description	Informações a cerca da API.
API Port	Porta que a API está associada.
API Register Key	Chave de autenticação da API.
Type	Tipo da API (System ou Snort)

Da mesma maneira que uma SystemAPI ou SnortAPI pode realizar autenticação junto ao *Controller*, também a pode encerrar. Nesse caso, a API estará deixando de confiar no *Controller* e assim para de executar suas configurações. A fim de deixar de confiar em uma API, o *Controller* necessita receber as informações necessárias para começar o processo de encerramento da confiança. Para tanto, uma requisição recebida pelo *Controller* deve possui os três campos descrito na Tabela 5. Recebendo esse tipo de requisição, o *Controller* remove de sua base de dados todas as informações acerca da API solicitante, incluindo sua chave de confiança. Da mesma maneira, quando a API recebe a confirmação de que o procedimento foi realizado com sucesso ela apaga as informações do *Controller*.

O *Controller* é uma entidade composta por quatro programas, além das APIs de registro, que são inicializados em paralelo, cada um realizando tarefas necessárias para a manutenção do funcionamento da arquitetura de segurança na rede local. Em referência à Figura 2, os citados programas são:

Tabela 5. Parâmetros para desautenticação com o *Controller*

API Description	Informações acerca da API.
Controller Key	Chave de autenticação do Controller.
API Register Key	Chave de autenticação da API.

1. *Snort Listener*: Servidor de *syslog* para receber e processar alertas do Snort.
2. SDN *Controller*: Gerenciamento de dispositivos SDN.
3. *Health Agent*: *Daemon* de imposição de assinaturas para as SystemAPIs.
4. *Rule Cleaner*: *Daemon* de exclusão de regras expiradas.

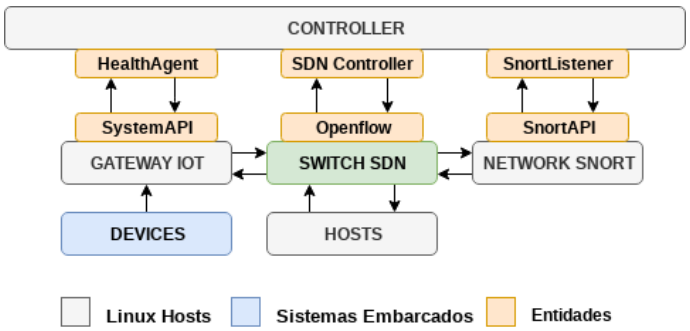

Figura 2. Comunicação entre as entidades da solução

Como descrito na subseção 3.1, o *Snort* será a aplicação responsável por identificação de tráfego malicioso na rede. Quando um ataque estiver ocorrendo, o Snort enviará via *syslog* para o *Controller* um alerta com as informações do incidente. Tal pacote de alerta é recebido pelo Snort *Listener*, que é um *daemon* que realiza a interpretação do *syslog* para determinar informações como: protocolo, endereço de origem, endereço destino, porta origem e porta destino. Assim que conseguir tais informações, o *daemon* as salva na base de dados local do *Controller* como identificadoras de um atacante.

As configurações de segurança começam a ser aplicadas a medida que novas informações sobre o tráfego de um atacante são cadastradas. No caso dos *switches* gerenciáveis, regras de repasse padrões na controladora SDN do *Controller* possuem tempo de vida de cinco minutos. Assim, quando determinada regra que realiza o repasse de um tipo de tráfego passa de cinco minutos, ela é removida e o *switch* deve verificar novamente com a controladora SDN qual política deve ser aplicada ao tráfego [Kreutz et al. 2015]. Se o tráfego que o *switch* envia para a controladora SDN pedindo instruções coincide com algum identificador de atacante na base de dados local, a controladora cria uma regra de bloqueio para esse tráfego com período de duração de duas horas. Dessa maneira o atacante não poderá mais transmitir dados na rede.

Com os identificadores de atacantes cadastrados na base de dados local do *Controller*, o *Health Agent* inicializa a criação de regras de *firewall* para as SystemAPIs cadastradas no *Controller*. Utilizando as informações obtidas pelo *Snort Listener*, o *Health Agent* realiza o cadastro de regras para bloqueio de tráfego que possua os mesmos identificadores dos atacantes cadastrados. Com isso os *hosts* Linux/*gateways* IoT existentes da rede local se tornam mais seguros frente a ataques, dificultando também a propagação

de ataques por sistemas comprometidos. A Figura 2 mostra a interação proposta entre as APIs e os *daemons* do *Controller*.

Da mesma maneira que ocorre na *SystemAPI*, o *Controller* também possui um *daemon* para exclusão de regras expiradas. O *daemon* da *SystemAPI* faz com que regras que passaram de duas horas de existência sejam removidas da base. O funcionamento é semelhante para o *Controller*, removendo as regras da base de dados que tenham tempo de vida superior a duas horas. As regras que correspondem com as assinaturas cadastradas serão aplicadas nos dispositivos SDN e *SystemAPIs*, resultando em um ambiente de segurança que evolui concomitantemente com a atual situação da rede.

Pode-se observar que o *Controller* é a peça principal para o ecossistema proposto, uma vez que ele correlaciona as informações recebidas acerca da rede e define políticas de segurança para mitigar possíveis ataques.

4. Resultados

As entidades SystemAPI, SnortAPI e Controller descritas na arquitetura foram implementadas em absoluto para realização dos testes aqui presentes, todas as funcionalidades das mesmas estão de acordo com o descrito na Seção 3 afim de testar a maneira com que a arquitetura proposta se comporta frente a tráfego malicioso. A seguir será descrito o ambiente onde os testes foram realizados, execução dos testes e a análise dos resultados obtidos.

4.1. Descrição do ambiente de teste

Afim de testar as funcionalidades da arquitetura proposta neste artigo, foi desenvolvido um ambiente para a execução da solução proposta em uma máquina virtual Ubuntu com 2GB de memória RAM e um núcleo de processador. Dentro desse ambiente de teste, foi criada a topologia exposta na Figura 3, nela o *Gateway* de borda representa a interface em NAT (*Network Address Translation*) da máquina virtual com a rede externa. O *firewall* desta topologia por sua vez é o próprio Iptables do sistema Ubuntu em execução, para o tráfego sair da rede interna da máquina virtual é necessário passar por ele. O Snort foi inicializado em modo *Network Intrusion Detection System* (NIDS) para realizar a detecção de tráfego malicioso passando pela topologia, o Snort por sua vez foi configurado para realizar detecção nos pacotes que passam pela interface de NAT da máquina virtual, ou seja todos os pacotes que são endereçados tanto para topologia interna quanto para a rede externa. Para simular uma rede definida por software foi utilizado o software Mininet para realizar a emulação de uma rede virtual com dois *hosts* e um *switch* OVS (*Open Virtual Switch*) em topologia *simple*, ou seja, os *hosts* estão conectados diretamente ao *switch*. Os endereços MAC dos dois *hosts* da topologia foram gerados no momento de inicialização do Mininet e a controladora SDN foi definida como *remote* para que a entidade SDN Controller implementada funcione como controladora SDN do *switch* da topologia.

As entidades que compõe a arquitetura de segurança também estão em execução no sistema da máquina virtual. A SystemAPI será a responsável por criar regras Iptables no *firewall* do sistema para impedir propagação de tráfego malicioso para fora da rede. A SnortAPI realizará o cadastro da regra de assinatura para detecção do ataque que será feito como teste. A Controller por sua vez atuara como a entidade central de processamento de alertas do Snort para criação de políticas de bloqueia no *switch* SDN e no *firewall* do sistema.

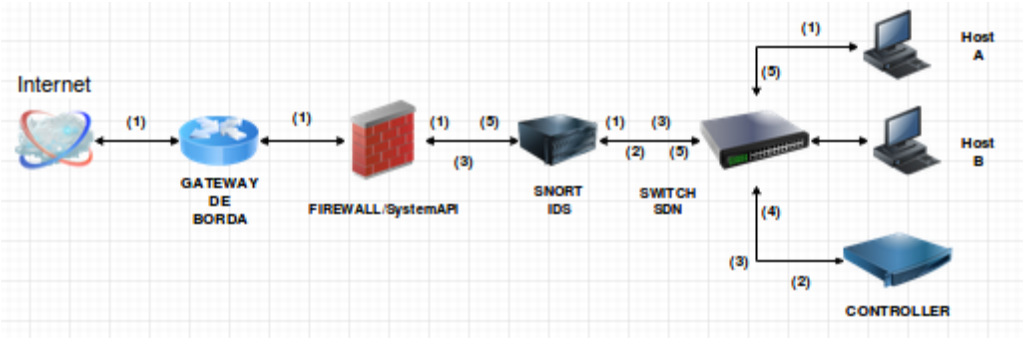

Figura 3. Topologia montada dentro da máquina virtual de testes

4.2. Descrição do teste

Para simular tráfego malicioso na topologia representada pela Figura 3 foi realizado um ataque de negação de serviço na porta 80 de uma máquina pertencente a rede externa. Para simular tal ataque foi utilizado o software *hping3* a partir do Host A, o software de *pentesting* foi utilizado com configurações de funcionamento *–fast* para aumentar a taxa de envio de pacotes e a opção *–random-source* para modificar o endereço de origem em cada pacote visando dificultar a detecção e aplicação de políticas de segurança. Na Figura 3 podemos ver o fluxo de acontecimentos na topologia, em (1) o ataque foi inicializado a partir do Host A tendo como destino a máquina da rede externa. No momento em que os pacotes passaram pela interface de NAT da máquina virtual o Snort identificou o ataque utilizando a regra previamente cadastrada e enviou o alerta para o Controller (2), o SnortListener do Controller realizou a interpretação do alerta para identificar as características do atacante e a partir delas o HealthAgent e a SDN Controller realizaram as configuração de políticas de segurança no *switch* SDN (4) e no *firewall* Iptables do sistema via SystemAPI (3). Dependendo da quantidade de pacotes o tempo de criação de regras pode aumentar pela quantidade das mesmas, em (5) podemos ver o caso em que os pacotes do ataque ainda saem da rede pois a regra que os bloqueie não foi ainda configurada. Discutiremos o tempo de criação de regras na próxima subseção.

4.3. Análise de resultados

Durante os testes de ataque foram armazenados metadados acerca do tráfego malicioso, quando o atacante foi cadastrado na Controller um *timestamp* em Unix epochmilis é armazenado para representar o momento em que o atacante foi identificado pela arquitetura. De maneira análoga foi armazenado um *timestamp* em unix epochmilis para representar o momento de cadastro das políticas de segurança tanto na SystemAPI quanto no *switch* SDN. A partir do tempo de cadastro de atacante e do tempo de cadastro de regra de segurança podemos determinar quanto tempo a arquitetura leva para mitigar o tráfego malicioso. Podemos ver a representação gráfica dos resultados de tempo de bloqueio nas Figuras 4a e 4b. No gráfico 4a pode-se observar que com o aumento do número de pacotes no ataque o tempo de bloqueio aumenta linearmente, há variações presentes no gráfico advindas da maneira com que o sistema de detecção e de aplicação foram feitos, se um pacote já tiver sido cadastrado como atacante o próximo pacote demora um pouco mais para ser bloqueado, resultando nos picos e vales presentes na reta. Conforme pode ser visto na Figura 4b o tempo que o *switch* levou para bloquear o ataque teve como valor máximo 770,867 ms sendo que o terceiro quartil das medições ficou em 728 ms.

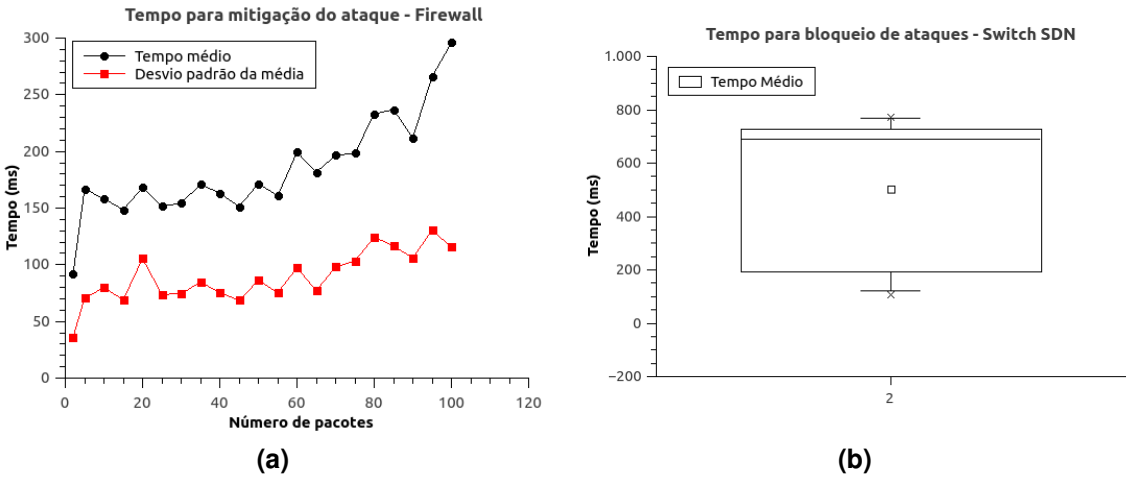

Figura 4. (a) Tempo para bloqueio de ataques no *firewall* (b) Tempo para bloqueio de ataques no *switch* SDN

5. Conclusão e trabalhos futuros

Este artigo apresenta um IPS que realiza configurações de proteção em elementos de infraestrutura de rede a partir da identificação de atacantes por um IDS. Após tal identificação, alertas são enviados para a entidade central da arquitetura proposta, o *Controller*, que por sua vez filtra tais alertas para reunir as informações acerca do atacante: endereços de origem e destino, portas de origem e de destino. Após a filtragem, as informações reunidas são salvas na base de dados local do *Controller* e propagadas às entidades cadastradas que possuem SystemAPI e aos *switches* SDN gerenciados pela solução.

Em trabalhos futuros, evoluções da arquitetura visarão a eficiência na criação de políticas de bloqueio quando estiver lidando com ataques de negação de serviço. Levando em consideração a evolução no relacionamento entre vulnerabilidades e ataques, uma abordagem de detecção por anomalia utilizando *Machine Learning* para substituir a utilização do Snort está sendo considerada para promover a execução autônoma da solução. Outra evolução que está sendo estudada é a noção de um *Controller* Global em nuvem para atuar como orquestrador de *Controllers* espalhados pela rede, de modo que ataques detectados por um *Controller* sejam anunciados para o *Controller* Global para que o mesmo propague as políticas de segurança para outras redes aderentes à proposta.

Agradecimentos

Os autores agradecem o apoio das Agências brasileiras de pesquisa, desenvolvimento e inovação CNPq (Projeto INCT SegCiber 465741/2014-2), CAPES (Projeto FORTE 23038.007604/2014-69) e FAPDF (Projetos UIoT 0193.001366/2016 e SSDDC 0193. 001365/2016), bem como o suporte do Laboratório LATITUDE/UnB (Projeto SDN 23106. 099441/2016-43), e a cooperação com o Ministério da Economia (TEDs DIPLA 005/2016 e ENAP 083/2016) e o Gabinete de Segurança Institucional da Presidência da República (TED 002/2017).

Referências

Bonomi, F., Milito, R., Zhu, J., and Addepalli, S. (2012). Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on*

Mobile cloud computing, pages 13–16, Helsinki, Finland. ACM.

Bull, P., Austin, R., Popov, E., Sharma, M., and Watson, R. (2016). Flow based security for IoT devices using an SDN gateway. In *2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 157–163, Vienna, Austria. IEEE.

Cho, E. J., Kim, J. H., and Hong, C. S. (2009). Attack model and detection scheme for botnet on 6LoWPAN. In *Asia-Pacific Network Operations and Management Symposium*, pages 515–518, Jeju, South Korea. Springer.

de Jesus, W. P., da Silva, D. A., de Sousa, Jr., R. T., and da Frota, F. V. L. (2014). Analysis of SDN contributions for cloud computing security. In *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, pages 922–927.

Ferreira, H. G. C. and de Sousa Jr, R. T. (2017). Security analysis of a proposed internet of things middleware. *Cluster Computing*, 20(1):651–660.

Gubbi, J., Buyya, R., Marusic, S., and Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660.

Hamza, A., Gharakheili, H. H., and Sivaraman, V. (2018). Combining mud policies with sdn for iot intrusion detection. In *Proceedings of the 2018 Workshop on IoT Security and Privacy*, pages 1–7. ACM.

Jin, R. and Wang, B. (2013). Malware detection for mobile devices using software-defined networking. In *2013 Second GENI Research and Educational Experiment Workshop*, pages 81–88. IEEE.

Kasinathan, P., Pastrone, C., Spirito, M. A., and Vinkovits, M. (2013). Denial-of-service detection in 6LoWPAN based internet of things. In *2013 IEEE 9th international conference on wireless and mobile computing, networking and communications (WiMob)*, pages 600–607, Lyon, France. IEEE.

Kolias, C., Kambourakis, G., Stavrou, A., and Voas, J. (2017). DDoS in the IoT: Mirai and other botnets. *Computer*, 50(7):80–84.

Kreutz, D., Ramos, F. M., Verissimo, P., Rothenberg, C. E., Azodolmolky, S., and Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76.

Lear, E., Romascanu, D., and Droms, R. (2019). Manufacturer usage description specification.

Mell, P. and Grance, T. (2011). The NIST Definition of Cloud Computing. Technical report, National Institute of Standards and Technology, Gaithersburg, MD, USA.

Raza, S., Wallgren, L., and Voigt, T. (2013). SVELTE: Real-time intrusion detection in the internet of things. *Ad hoc networks*, 11(8):2661–2674.

Roesch, M. (1999). Snort: Lightweight intrusion detection for networks. In *Proceedings of LISA '99: 13th Systems Administration Conference*, pages 229–238, Seattle, WA, USA.