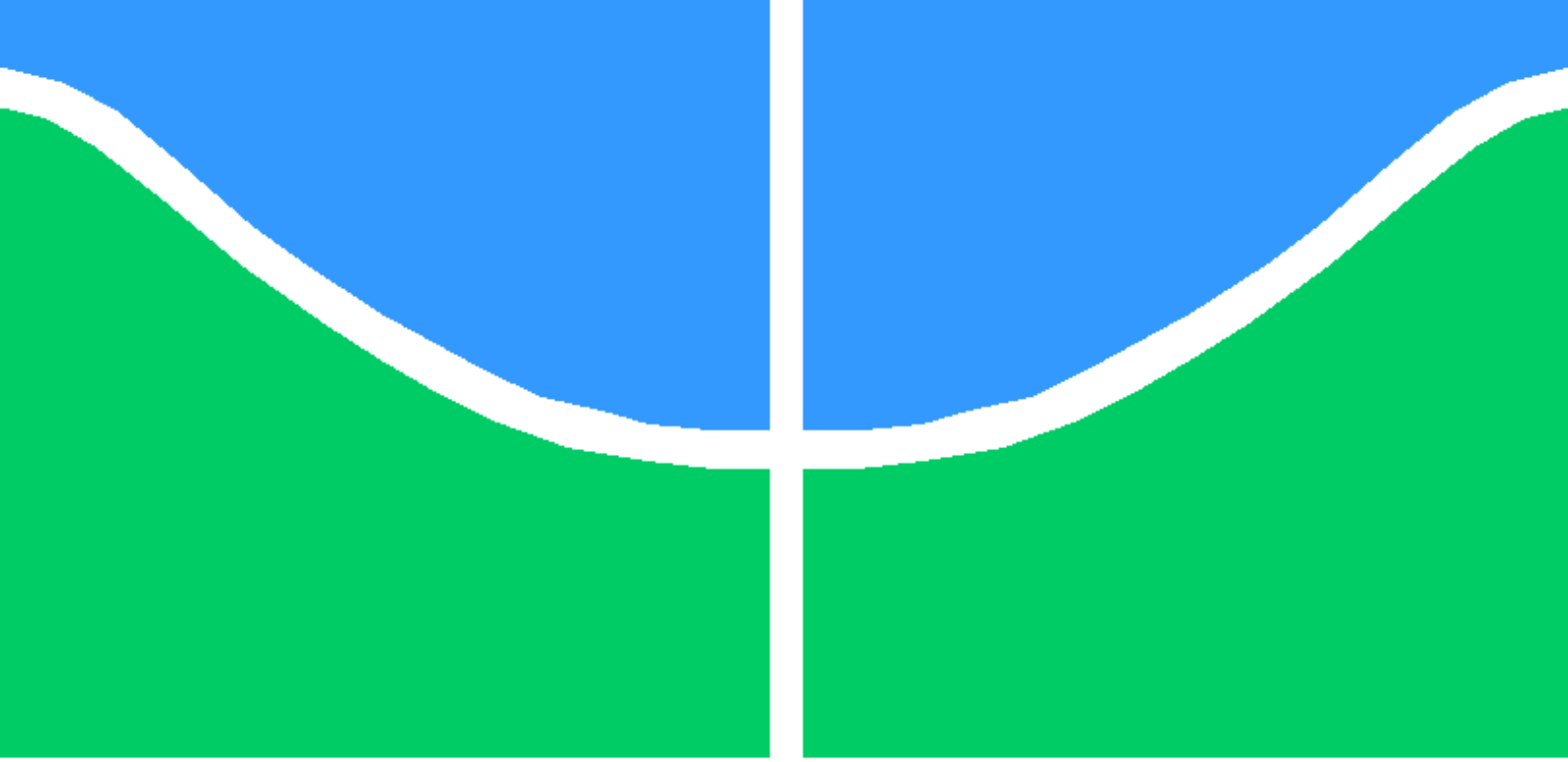


Trabalho Final de Graduação

Arquitetura híbrida para detecção e prevenção de intrusão em dispositivos IoT

Rafael Zerbini Alves da Mata

Brasília, Novembro de 2021

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia

Trabalho Final de Graduação

Arquitetura híbrida para detecção e prevenção de intrusão em dispositivos IoT

Rafael Zerbini Alves da Mata

Relatório submetido ao Departamento de Engenharia
Elétrica como requisito parcial para obtenção
do grau de Engenheiro de Redes de Comunicação

Banca Examinadora

Prof. PhD. Rafael Timóteo de Sousa Júnior, _____
ENE/UnB
Orientador

MSc. Francisco Lopes de Caldas Filho, _____
ENE/UnB
Co-orientador

Msc. Juliano Barbosa Prettz, ENE/UnB _____
Examinador

Agradecimentos

Com o final desse ciclo extenso regado de aprendizados, experiências e sentimentos, devo lembrar de todos aqueles que estiveram ao meu lado em todos esses momentos e que me ajudaram de forma única. Agradeço à minha família por todo apoio e por sempre estar ao meu lado durante essa jornada, em especial aos meus pais Daniela Zerbini e Sinval Alves da Mata. Jamais esquecerei os momentos durante essa jornada em que precisei de suporte e encontrei junto deles um ponto de apoio. Agradeço também a minha irmã, Gabriela Zerbini, que me ajudou de formas inimagináveis durante meu processo de ingresso na Universidade de Brasília.

É com imensa gratidão que agradeço todos os amigos que fiz durante essa caminhada, por todos os momentos de fraternidade e auxílio que foram divididos, em especial agradeço Mirella Sales, Ricardo Dias, Giampaolo Lepore e Larissa Nobre pela amizade construída durante esse período, obrigado por tudo! Agradeço ao amigos que estiveram ao meu lado durante o desenvolvimento desse trabalho por todo auxílio e aprendizado, em especial ao Helio Adson, Felipe Ayres, Vinícius Campos, Carlos André, Flávio Scorpione, Lucas Perin, Lucas Nobre e Henrique Pasquarelli, valeu Vencedores! Agradeço também aos meus amigos de longa data, que sempre estiveram ao meu lado em momentos importantes, em especial Pedro Henrique e Camila de Moura.

Agradeço a todos cujos caminhos já se encontraram com o meu, a quem me ajudou direta ou indiretamente nesta pesquisa, em especial ao meu co-orientador Francisco Lopes e ao meu professor orientador Rafael Timóteo por me instruírem durante essa pesquisa e em todos projetos realizados no UIoT, sou eternamente grato pelos ensinamentos e apoio oferecido.

Agradeço o apoio técnico e computacional do Laboratório de Tecnologias para Tomada de Decisão - LATITUDE, da Universidade de Brasília, que conta com apoio do CNPq - Conselho Nacional de Pesquisa (Outorgas 312180/2019-5 PQ-2, BRICS2017-591 LargEWiN e 465741/2014-2 INCT em Cibersegurança), da CAPES - Coordenação de Aperfeiçoamento do Pessoal de Nível Superior (Outorgas PROAP PPGEE/UnB, 23038.007604/2014-69 FORTE, e 88887.144009/2017-00 PROBRAL), da FAP-DF - Fundação de Amparo à Pesquisa do Distrito Federal (Outorgas 0193.001366/2016 UIoT e 0193.001365/2016 SSDDC), do Ministério da Economia (Outorgas 005/2016 DIPLA e 083/2016 ENAP), da Secretaria de Segurança Institucional da Presidência da República do Brasil (Outorga ABIN 002/2017), do Conselho Administrativo de Defesa Econômica (Outorga CADE 08700.000047/2019-14), da Advocacia Geral da União (Outorga AGU 697.935/2019),do Departamento Nacional de Auditoria do SUS (Outorga 23106.118410/2020-85), e da Procuradoria Geral da Fazenda Nacional (Outorga 23106.148934/2019-67).

Rafael Zerbini Alves da Mata

Resumo

Este ano, o número de dispositivos IoT já ultrapassou 8 bilhões e a expectativa é de um crescimento exponencial para os próximos anos. No entanto, a instalação e configuração desses dispositivos apresente em muitos casos vulnerabilidades de não se preocupar necessariamente com segurança. Isso favorece o aumento de ataques DDoS contra redes distribuídas de dispositivos e aplicativos IoT. Nesse contexto, soluções de segurança específicas para dispositivos IoT são atraentes. Este trabalho propõe um sistema de detecção e prevenção de intrusão para ser executados em dispositivos IoT através de uma arquitetura híbrida para prover regras ao sistema. A arquitetura faz uso de sistemas de mensagem distribuídas e o registro distribuído de transações Tangle para compartilhamento de dados com os dispositivos IoT. O Tangle permite um sistema federado no qual comportamentos anômalos identificados em uma rede externa podem ser repassados para outros dispositivos. Para validar a proposta, este artigo apresenta uma simulação onde dispositivos estão infectados com a botnet Mirai e uma análise do tempo de atualização de regras no sistema.

Palavras-chaves: Segurança em IoT, detecção de intrusão, prevenção de intrusão, Tangle, Blockchain, streaming de eventos.

Abstract

This year, the number of IoT devices has already exceeded 8 billion and exponential growth is expected for the coming years. However, the focus of these embedded devices is to provide the proposed services with easy configurability, necessarily worrying about security. This favors the rise of DDoS attacks against distributed device networks and IoT applications are increasing. In this context, specific security solutions for IoT devices are attractive. This work proposes an intrusion detection and prevention system for IoT devices through a hybrid architecture to provide rules for the system. The architecture makes use of distributed messaging systems and Tangle to share data with IoT devices. Tangle allows a federated system without qualifying anomalous behavior identified in an external network can be passed on to other devices. To validate the proposal, this article presents a simulation where they are infected with a Mirai botnet and an analysis of the rules update time in the system.

Keywords: IoT Security, intrusion detection, intrusion prevention, Tangle, blockchain, event streaming.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

Acrônimos

HIDS	Host Intrusion Detection System
HIDPS	Host Intrusion Detection and Prevention System
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
IDPS	Intrusion Detection and Prevention System
NIDS	Network Intrusion Detection System
UnB	Universidade de Brasília
IoT	Internet of Things
DLT	Distributed Ledger Technologies
DDoS	Distributed Denial of Service
ABS	Anomaly Based System
SBS	Signature Based System
P2P	Peer-to-Peer
JSON	JavaScript Object Notation
YAML	Yet Another Markup Language
ELK	ElasticSearch, Logstash e Kibana
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
IP	Internet Protocol
MAC	Medium Access Control
API	Application Programming Interface
WSN	Wireless Sensors Network
SSH	Secure Shell

1 Introdução

Ao longo dos anos a humanidade continuou a buscar formas de tornar a vida confortável através de tecnologias. Nas últimas décadas a sociedade passou por uma jornada de diversas transformações tecnológicas explorando novas fronteiras. Tais fronteiras tem interagido com a sociedade e realizado trabalhos em um menor período de tempo e com maior acurácia. Com o advento dessas novas tecnologias o mundo está se tornando cada vez mais conectado. Essas tecnologias criam um ecossistema de dispositivos onde a principal função deles é enviar e receber dados. A Internet das Coisas (IoT) é uma tecnologia dominante que engloba dispositivos inteligentes trazendo a tona aplicações de ficção para a realidade habilitando a quarta revolução industrial [34].

A Internet das Coisas (IoT) [17] refere-se à rede de dispositivos de detecção de baixa potência e capacidade de processamento limitada, que podem enviar / receber dados de / para outros dispositivos usando tecnologias sem fio, como RFID (identificação por radiofrequência), Zigbee, WiFi, Bluetooth, 3G/4G etc. Dispositivos IoT estão sendo implantados em uma série de aplicações como automação residencial, monitoramento ambiental, gerenciamento de infraestrutura, automação industrial, automação agrícola, saúde e cidades inteligentes.

Os números evidenciam o impacto do paradigma de IoT sobre a sociedade, uma vez que quantidade de dispositivos IoT vem crescendo de maneira exponencial ao longo dos anos. Segundo o levantamento apresentado por [42], em 2020 haviam 8,74 bilhões de dispositivos IoT conectados e a previsão é que o numero de dispositivos conectados em 2025 quase triplique para 25.4 bilhões.

Dispositivos IoT são projetados para permitir a instalação em ambientes domésticos e empresariais de forma fácil e descomplicada, realizando o acompanhamento de variáveis ambientais como temperatura e umidade do ambiente além da monitoração de espaços utilizando câmeras inteligentes levando os dados para a nuvem [36]. Para facilitar a configuração destes dispositivos para o usuário final, muitas vezes os fabricantes utilizam configurações padrões e com baixo teor de segurança vulneráveis a acesso por terceiros [23]. Nesse contexto, destacam-se os Ataques Distribuídos de Negação de Serviço (DDoS, do inglês Distributed Denial of Service), que exploram vulnerabilidades nos dispositivos IoT para criar *botnets*, redes de dispositivos infectados com o objetivo de realizar um ataque, que serão utilizados para realizar requisições simultâneas ao mesmo alvo, podendo assim sobrecarregar seu sistema e impedir que conexões legitimas sejam efetivadas [19].

Existem diversos mecanismos de segurança para identificar se dispositivos IoT estão fazendo parte de uma *botnet* e para realizar ações de mitigação do tráfego malicioso [14, 18, 22]. A mitigação do tráfego malicioso pode ocorrer diretamente no dispositivo, que identifica situações anormais no seu funcionamento e toma ações para realizar a correção, ou na camada de redes com o uso de *firewalls* e IPS com capacidade de reconhecer situações anormais e bloquear o trafego. A mitigação diretamente no dispositivo envolve a aplicação de sistemas *Host-based intrusion detection and prevention system*, HIDPS, onde uma aplicação executada pelo próprio dispositivo identifica

situações anormais nos seus processos, em portas de rede ou em arquivos do sistema e toma ações para corrigir a falha [44].

Este método interrompe o ataque na origem, evitando sobrecarregar a infraestrutura de rede. Porém, ele tem as suas limitações quando aplicado a um cenário de IoT:

- dispositivos IoT são dotados de baixo poder computacional, dificultando a execução de programas de verificação;
- a quantidade de dispositivos IoT cresce exponencialmente e aplicar regras de segurança em cada um deles individualmente seria um trabalho árduo e inviável;
- manter registro e gerência das regras de segurança aplicadas em uma rede com vários dispositivos se torna complexo;
- regras de uma rede podem não ser heterogêneas o suficiente, o que pode levar uma rede a não estar preparada o suficiente para diferentes tipos de ataques;

Para que as limitações apresentadas acima sejam resolvidas é necessário que um HIDPS de baixo consumo voltado para dispositivos IoT seja desenvolvido e que uma arquitetura para provisionamento e monitoramento do HIDS que tenha escalabilidade frente o crescente número também seja construída. Além disso uma possível solução para a heterogeneidade das regras é o compartilhamento de regras entre diferentes redes, utilizando um canal seguro e auditável.

1.1 Objetivos

O presente trabalho apresenta os objetivos gerais e específicos descritos a seguir.

1.1.1 Objetivo Geral

Desenvolver e analisar um sistema de detecção e prevenção de intrusão para ser executado em *gateways* IoT, bem como uma arquitetura escalável que permita a gerência e provisionamento de regras para esse sistema. Como parte fundamental desse trabalho, tem-se o estudo de uma arquitetura híbrida, entre cliente-servidor e *peer-to-peer*. Para a construção da arquitetura será utilizado uma arquitetura baseada em eventos [16] para gerenciamento e provisionamento dos *gateways* IoT, bem como o cliente HIDPS. Já para o peer-to-peer será utilizada uma solução baseada em DLTs (do inglês, Distributed Ledger Technologies) [43], com o objetivo de permitir a distribuição de regras entre dispositivos em diferentes redes e sobre jurisdições.

1.1.2 Objetivos específicos

- Avaliar o desempenho do sistema de detecção de intrusão;
- Desenvolver um orquestrador capaz de provisionar e gerenciar gateways IoT;

- Desenvolver e avaliar mecanismos de segurança para regras oriundas de diferentes redes e diferentes jurisdições;
- Análise e criação de assinaturas para prevenção e detecção das vulnerabilidades mais comuns e das botnets mais famosas;

1.2 Justificativa

Dado o aumento na quantidade de ataques distribuídos utilizando dispositivos IoT torna-se necessário soluções de segurança distribuída e escaláveis. Dessa forma o proposto nesse trabalho é um sistema de detecção de intrusão executado em gateways IoT com assistência de uma arquitetura híbrida utilizando um sistema baseado em eventos em nuvem e uma DLT descentralizada. Durante o processo é pretendido buscar compreender os detalhes envolvidos na implementação da arquitetura e do HIDPS, bem como se os resultados atingidos pela proposta são favoráveis.

Se tratando de um sistema de compartilhamento de regras, uma arquitetura totalmente distribuída enfraquece a arquitetura geral da rede, uma vez que se a maioria dos nós forem corrompidos dados que implementam vulnerabilidades nos dispositivos podem ser implementados [47]. Utilizando uma arquitetura híbrida fortalecemos a rede, uma vez que existe uma entidade responsável por gerenciar os dispositivos, assim permitindo que exista uma validação dos dados trafegados pela rede peer-to-peer. O uso de uma arquitetura híbrida permite que os dispositivos sejam gerenciados pelo administrador de redes através do orquestrador e também permite que as quantidade de regras fornecidas aos dispositivos seja escalada, recebendo dados de servidores externos através da rede peer-to-peer.

Dessa forma, uma arquitetura de detecção de intrusão e prevenção de intrusão para dispositivos IoT que seja escalável torna-se significante frente ao aumento exponencial na quantidade de dispositivos e ao crescente número de ataques por meio desses dispositivos. Esse trabalho visa contribuir com um sistema que seja capaz de realizar a mitigação de vulnerabilidades nesses dispositivos de forma escalável e eficaz.

1.3 Trabalhos Publicados

Durante o desenvolvimento deste trabalho, foram publicados artigos científicos propondo uma solução IoT interoperável e segura, onde dispositivos IoT podem se comunicar de forma autônoma e utilizando diferentes semânticas. A pesquisa, que também foi direcionada para aperfeiçoar a segurança de dispositivos e redes IoT, resultou assim nas seguintes publicações:

1. DA MATA, Rafael Zerbini Alves; RODRIGUES, Carlo Kleber da Silva. Uma Análise Competitiva entre as Tecnologias Blockchain e Tangle para o Projeto de Aplicações IoT. In: WORKSHOP EM DESEMPENHO DE SISTEMAS COMPUTACIONAIS E DE COMUNICAÇÃO (WPERFORMANCE), 17. , 2018, Natal. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2018 . ISSN 2595-6167. DOI: https://doi.org/10.5753/wperformance.2018.3339.

2. DA MATA, Rafael ZA et al. 01 Distributed Architecture for Intrusion Detection in IoT Networks using Smart Contracts. 2021. N.º 34: Investigación en Ciberseguridad. Jornadas Nacionales de Investigación en Ciberseguridad I.S.B.N.: 978-84-9044-463-4;

3. Dias, B. S. D., e Mello, I. P., de Caldas Filho, F. L., da Mata, R. Z. A., Mendonça, F. L. L., Junior, R. T. S. (in press). Sistema Monitor De Aglomerações Baseado Em Reconhecimento De Padrões E Cálculos De Distanciamento Social Operante Em Rede Iot Estruturada Em Fog Computing. (Prevision Screen –November/2021)

4. DA MATA, Rafael ZA, et al. "Hybrid Architecture for Intrusion Prevention and Detection in IoT Networks"2021 Workshop on Communication Networks and Power Systems (WCNPS). IEEE, 2021. (Prevision Screen – November/2021)

2 Revisão Bibliografica e Marco Teórico

2.1 Marco Teórico

2.1.1 Gateways IoT

Os Gateways IoT têm como principal função a interconexão de redes de sensores com a rede tradicional, agindo de forma semelhante a um *proxy* [10]. Os Gateways IoT geralmente possuem interfaces que suportam diferentes protocolos de comunicação, permitindo assim que sensores sem suporte a comunicação TCP / IP como Zigbee, Bluetooth e LoRA possam enviar dados e receber comandos remotos [6].

Os requerimentos de um gateway IoT podem ser resumidos como [50]:

1. Encaminhamento de dados: A função básica de um Gateway IOT é receber dados de terminais de rede de sensores ou terminais de Internet e, em seguida, transferir dados para a outra redes de forma transparente e correta.

2. Conversão de protocolos: Os protocolos utilizados em redes de sensores geralmente diferem da rede tradicional, por exemplo redes de sensores costumam utilizar *IEEE 802.15.4/Zigbee* enquanto a internet tradicional utiliza TCP/IP. Assim, Gateways IoT recebem dados através de uma rede de sensores sem fio, *WSN* (do inglês, Wireless Sensors Network) e encapsula os dados recebidos em protocolos da internet tradicional.

3. Gerenciamento e controle: além de receber e enviar dados de sensores, o Gateway IoT também deve oferecer gerenciamento e controle dos sensores. Por exemplo, quando o gateway recebe os comandos do servidor remoto, o mesmo deve processar os comandos e, em seguida, despachá-los para os nós sensores para que o servidor remoto possa gerenciar e controlar a rede de sensores através do Gateway IoT.

Dado as funções de um gateway IoT especificadas acima podemos observar que é extremamente importante garantir que esses dispositivos sejam seguros. Principalmente dado as capacidades de gerenciamento e controle de um gateway IoT, no caso de comprometimento de um gateway os sensores também podem ser comprometidos e utilizados em botnets por exemplo [19]. Uma forma de monitorar por vulnerabilidades e comprometimentos de dispositivos e/ou da rede é através de sistemas de detecção e prevenção de intrusão, tópico que será abordado a seguir.

2.1.2 Sistemas de detecção e prevenção de intrusão

A detecção de intrusão é o processo de monitoramento e análise de eventos que ocorrem em um sistema ou em uma rede em busca de possíveis indícios de incidentes de segurança. Incidentes podem ter muitas causas, como *malware*, *spyware*, invasores obtendo acesso não autorizado e

usuários autorizados que abusam de seus privilégios ou tentam obter privilégios para os quais não estão autorizados.

Um sistema de detecção de intrusão, do inglês *Intrusion Detection System* (IDS), é um software que automatiza o processo de detecção de intrusão. Sistemas de prevensão de intrusão, do inglês *Intrusion Prevention System* (IPS), é um software que tem por objetivo evitar que intrusões ocorram. Sistemas de detecção e prevenção de intrusão (IDPS) é um software que realiza ambas as tarefas.

IDPSs tem como objetivo identificar possíveis incidentes de segurança. IDPSs podem detectar quando um invasor consegue acesso ao sistema através de alguma vulnerabilidade, assim o IDPS pode gerar um relatório para o administrador ou realizar alguma ação para mitigar o ataque. Também pode ser configurado para reconhecer violações das politicas de segurança, com um conjunto de regras semelhante as de um *firewall* [40]. A Figura 2.1 mostra o funcionamento geral de sistemas de detecção e prevenção de intrusão.

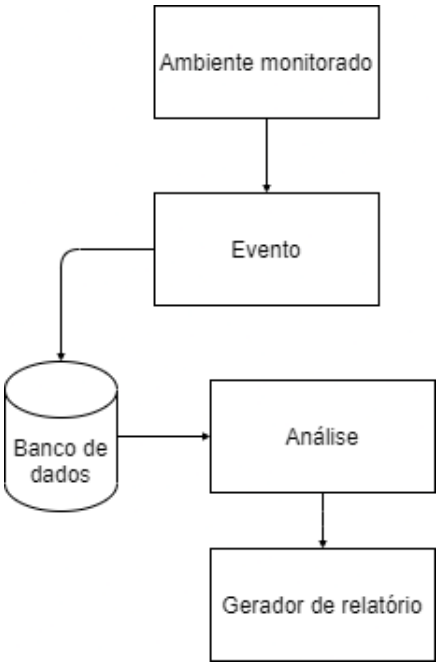


Figura 2.1 – Arquitetura geral de IDPSs. Fonte: adaptado de Mudzingwa [32]

A Figura 2.2 mostra de forma resumida classificações importantes para um IDPS [28]. Os intrusos podem ter origem tanto externa ou interna a rede. Os principais tipos de intrusão são:

- Tentativa de acesso: quando algum agente tenta conseguir acesso ao hospedeiro de alguma forma. Geralmente esse tipo de ataque é detectado através de violações nas políticas de segurança
- Vazamento: quando o atacante consegue retirar dados da aplicação que sofreu o ataque. É detectado através da análise de uso dos recursos do sistema.

- Uso malicioso: quando um atacante usa o sistema que foi explorado para atividades que o mesmo não deveria exercer. Detectado através de violações nas políticas de segurança, uso de privilégios especiais e mudança das atividades do sistema.
- Negação de serviço: é a tentativa de tornar os recursos de um sistema indisponíveis para os seus utilizadores. Detectado através de uso dos recursos do sistema

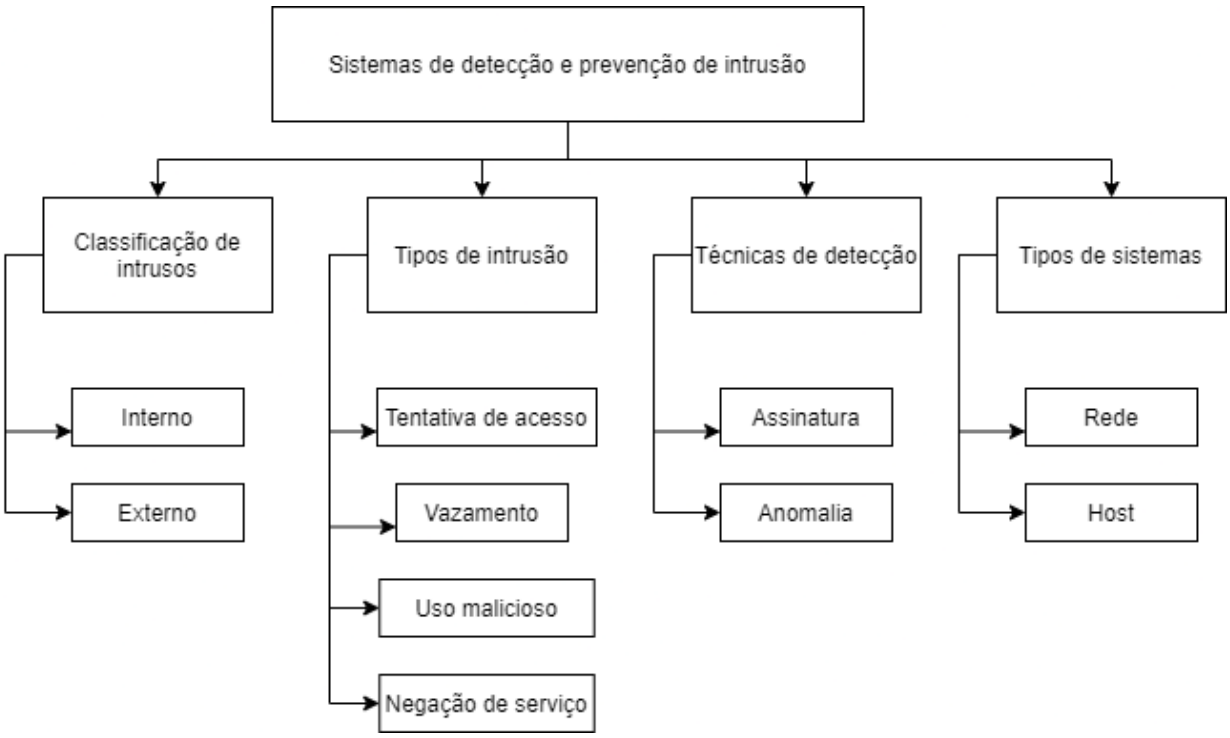


Figura 2.2 – Classificação dos sistemas de detecção e prevenção de intrusão
Fonte: Adaptado de Letou [28]

Existem dois principais tipos de sistemas de detecção e prevenção de intrusão: baseado em assinatura (SBS) e baseado em anomalia (ABS). SBSs utilizam técnicas de detecção de padrões, eles contém um banco de dados com assinaturas de ataques conhecidos e verificam essas assinaturas contra o dado a ser analisado. Já ABSs criam um modelo estatístico utilizando *Machine Learning* que representa o estado normal do sistema, então, qualquer comportamento diferente do considerado esperado para a rede é alarmado.

A metodologia baseada em assinatura funciona comparando assinaturas observadas com assinaturas arquivadas. Qualquer assinatura observada no ambiente monitorado que corresponde à assinaturas arquivadas são sinalizadas como uma violação da segurança, política ou como um ataque. O IDPS baseado em assinatura tem pouca sobrecarga, uma vez que não inspeciona todas as atividades ou tráfego no ambiente monitorado. Em vez disso, apenas realiza uma pesquisa para assinaturas conhecidas no banco de dados. SBSs, em geral, se apresentam como uma solução muito eficaz contra ataques conhecidos, porém não pode detectar novos ataques até que seja atualizado com novas assinaturas [32]. A Figura 2.3 mostra o processo geral de funcionamento para SBSs.

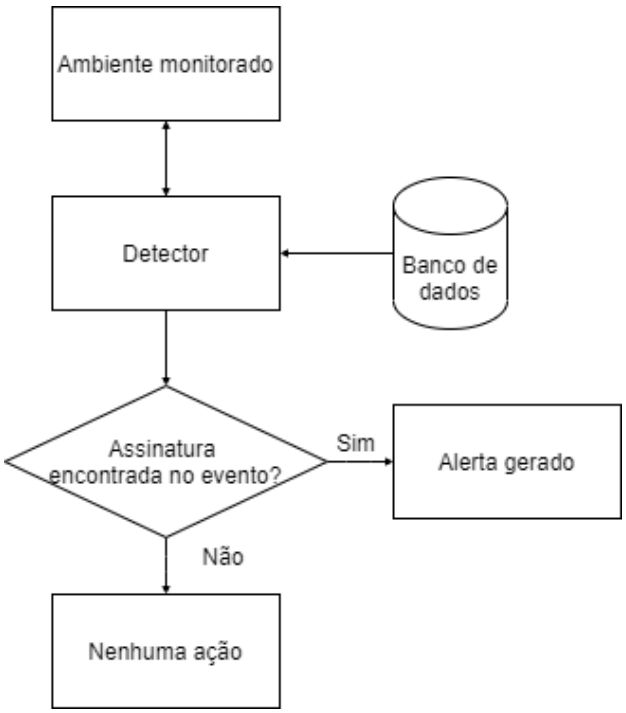


Figura 2.3 – Arquitetura geral de SBSs. Fonte: adaptado de Mudzingwa [32]

A metodologia baseada em anomalias funciona através da comparação da atividade observada em relação a um perfil considerado como o padrão normal. O perfil pode ser fixo ou dinâmico. Um perfil fixo não muda uma vez estabelecido, enquanto um perfil dinâmico muda à medida que os sistemas monitorados evoluem. Um perfil dinâmico traz mais sobrecarga ao sistema, uma vez que o IDPS continua atualizando o perfil. Um atacante pode evitar um IDPS que usa um perfil dinâmico espalhando o ataque por um longo período de tempo. Com isso, o ataque se torna parte do perfil a medida que o IDPS atualiza o comportamento [32]. A partir de um limite predefinido, quaisquer desvios que caiam fora do limite são relatados como violações. Um perfil fixo é muito eficaz na detecção de novos ataques, uma vez que qualquer comportamento diferente do padrão é categorizado como uma anomalia. A Figura 2.4 apresenta o processo geral de ABSs.

Metodologias baseadas em anomalias podem detectar ataques de dia zero ao ambiente, sem nenhuma atualização do sistema. Porém IDPSs baseados em anomalias frequentemente produzem muitos falsos positivos, por causa da atividade benigna que se desvia significativamente dos perfis, especialmente em ambientes mais diversos ou dinâmicos. Outro problema digno de nota com o uso de sistemas com técnicas de detecção baseados em anomalias é que muitas vezes é difícil para determinar o que causou um alerta, devido à complexidade do eventos e número de eventos que podem ter gerado o alerta [40].

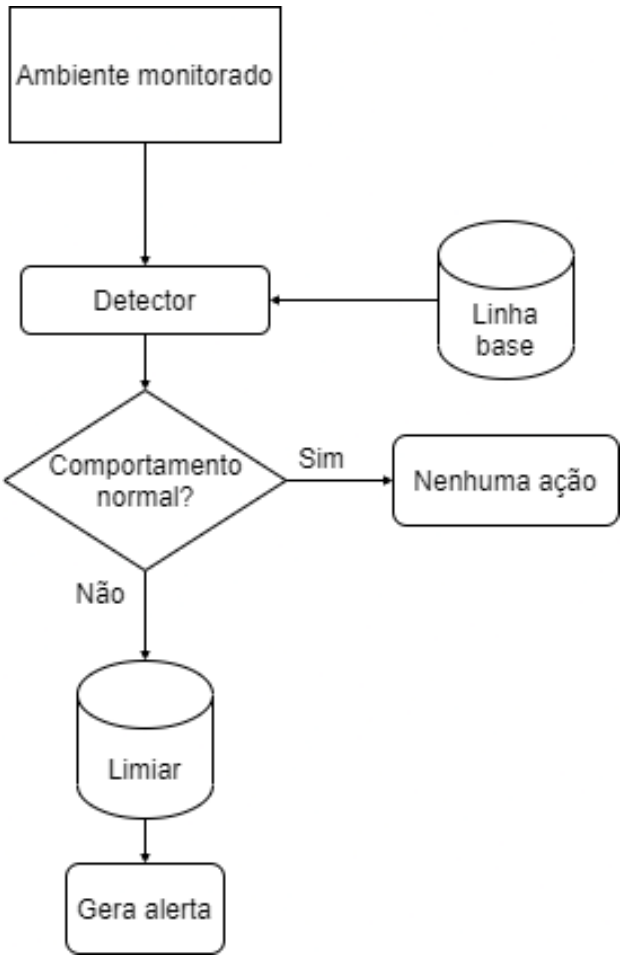


Figura 2.4 – Arquitetura geral de ABSs. Fonte: adaptado de Mudzingwa [32]

Também podemos dividir os sistemas de detecção de intrusão baseado em onde atuam, sistemas que atuam diretamente na rede (NIDSs) ou em um dispositivo (HIDSs). Os NIDSs analizam o trafégo da rede como um todo, utilizando técnicas de assinatura ou de anomalia. Já o HIDSs são executados em apenas um dispositivo e monitoram e analisam arquivos, processos, memória, entrada e saída do dispositivo. Nesse trabalho temos a construção de um HIDPS para dispositivos IoT, portanto o HIDPS será mais explicado a seguir.

A principal função de um HIDPS é rastrear e verificar informação relativas a logs, registros, eventos, permissões, além de outros eventos que são considerados objetos importantes para detecção de intrusão [41].

Outra abordagem [45] considera que as intrusões podem ser detectadas analisando diferentes comportamentos em processos, chamadas de sistema e outros eventos que são considerados anomalias de comportamento anormal. Por exemplo, se uma aplicação de e-mail começa a abrir e escrever arquivos, criar *sockets* e ouvir em diferentes portas do que são determinadas pela aplicação, pode-se concluir que o cliente foi comprometido.

Um aspecto importante de qualquer sistema IDPS é a atualização de regras para o sistema. Em geral sistemas IDPS possuem um banco de dados para armazenamento de regras ou perfis,

como visto anteriormente. Para clientes IDPS que não possuem banco de dados devido a recursos limitados é extremamente importante um sistema de compartilhamento de regras em tempo real com garantia de entrega. Duas possíveis soluções serão abordadas à seguir: a primeira através de sistemas de mensagens distribuídas, voltada para comunicação entre cliente e servidor e a segunda utilizando DLTs, voltada para comunicação *peer-to-peer*.

2.1.3 Sistemas de mensagens distribuídas

Atualmente, as informações são continuamente geradas em tempo real por aplicações e precisam de maneiras fáceis de serem roteado de forma confiável e rápida para vários receptores. Na maioria das vezes, os aplicativos que estão produzindo informações e aplicativos que estão consumindo este as informações estão separados e inacessíveis um dos outros. Para evitar a perda e o reenvio de informações entre os produtor e consumidor uma solução distribuída de mensagens se torna extremamente atraente [25].

Dado que em um cenário com dispositivos IoT, que possuem recursos limitados, os mesmos podem não ser capazes de processar as mensagens no mesmo momento em que foram recebidas [37]. Portanto, em um cenário onde perda de pacotes não é aceitável, uma solução de envio de mensagens distribuída é importante para evitar que os produtores de dados fiquem bloqueados no caso dos dispositivos finais não conseguirem receber dados devido ao alto uso dos seus recursos. Dado os requerimentos apresentados, o Apache Kafka é apresentado como uma solução viável [4]. O Apache Kafka é uma solução para lidar com grande volumes de dados em tempo real de informações e encaminhá-las para vários consumidores rapidamente. O Apache Kafka é distribuído e facilmente escalonável, além de oferecer alto *throughput*. Além disso o Apache Kafka possui como foco armazenamento de mensagens, o que o torna extremamente atrativo para a arquitetura proposta neste trabalho.

O Apache Kafka mantém as mensagens armazenadas em categorias chamadas de tópicos. Os processos que publicam mensagens para um tópico Kafka são denominados produtores. E os processos que ficam aguardando novas mensagens serem publicadas em um tópico são chamada de consumidores. O Kafka é executado como um *cluster*, composto por um ou mais servidores, cada um dos quais é chamado de *broker*. Em alto nível, os produtores enviam mensagens por meio da rede para o cluster Kafka, que por sua vez serve as mensagens para os consumidores. A Figura 2.5 apresenta a arquitetura Kafka em alto nível.

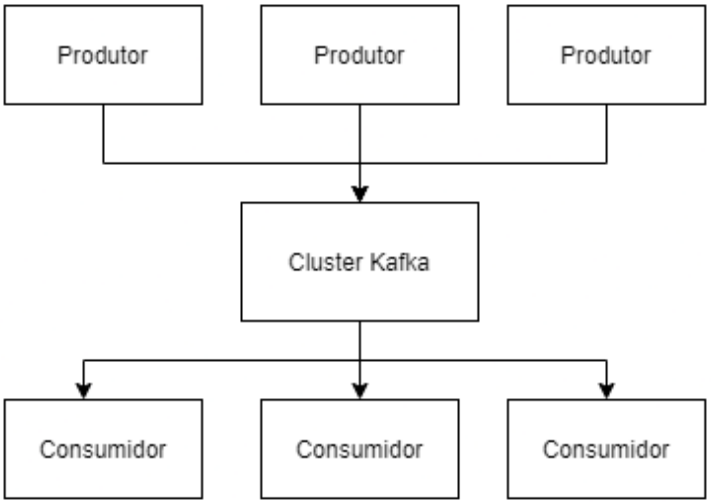


Figura 2.5 – Arquitetura Kafka em alto nível

Para cada tópico, o cluster Kafka mantém uma partição para dimensionamento, paralelismo e tolerância a falhas [25]. Cada partição é uma sequência ordenada e imutável de mensagens que é continuamente anexado a um log de confirmação. As mensagens em cada partição recebe um número de identificação sequencial chamado de *offset*. A anatomia de um tópico pode ser visto na Figura 2.6.

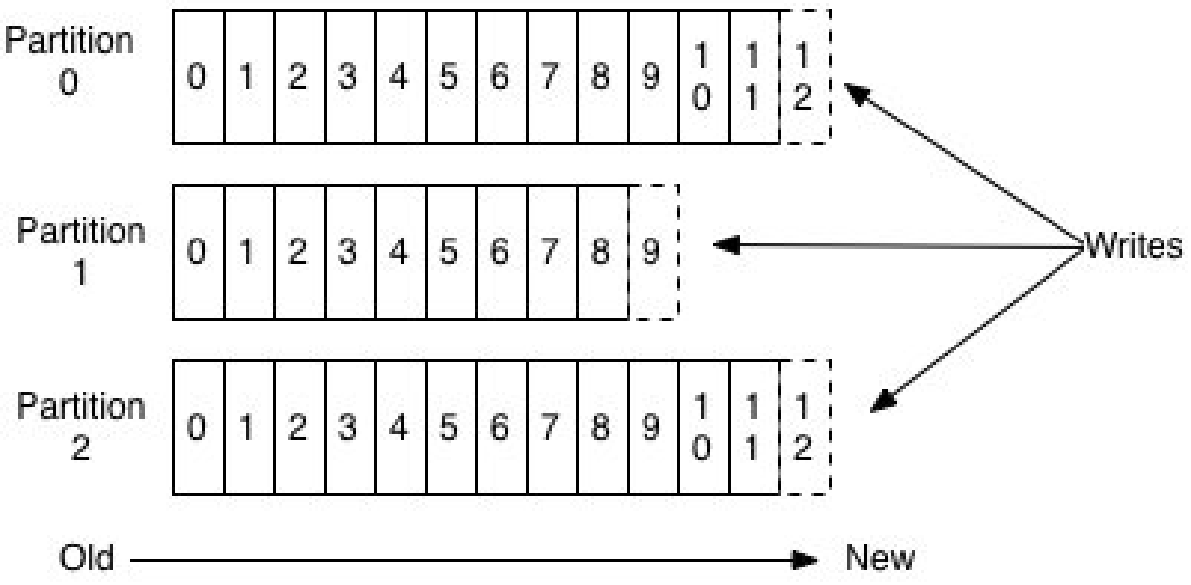


Figura 2.6 – Anatomia de um tópico no Apache Kafka
Fonte: Apache Kafka [4]

O offset é controlado pelo consumidor, permitindo que o consumidor decida se deseja ou não receber mensagens que já foram compartilhadas anteriormente. Um consumidor sempre consome mensagens de um determinado partição sequencialmente, o ponto de começo para consumir pode

ser escolhido uma vez que o cluster Kafka retém todas as mensagens publicadas dentro um período de tempo configurado. Os produtores escolhem qual tópico, e qual partição dentro do tópico, vão publicar a mensagem. Os consumidores atribuem a si mesmos um nome dentro do grupo de consumidores, e cada mensagem é entregue a um consumidor dentro de cada grupo de consumidores assinantes. Se todos o os consumidores têm o nome do grupo de consumidores diferentes, então as mensagens são transmitidos a cada consumidor. A Figura 2.7 apresenta os produtores enviando mensagens para todos os brokers de um cluster Kafka e temos dois grupos de consumidores, onde um grupo está consumindo de dois brokers e um grupo apenas de um broker.

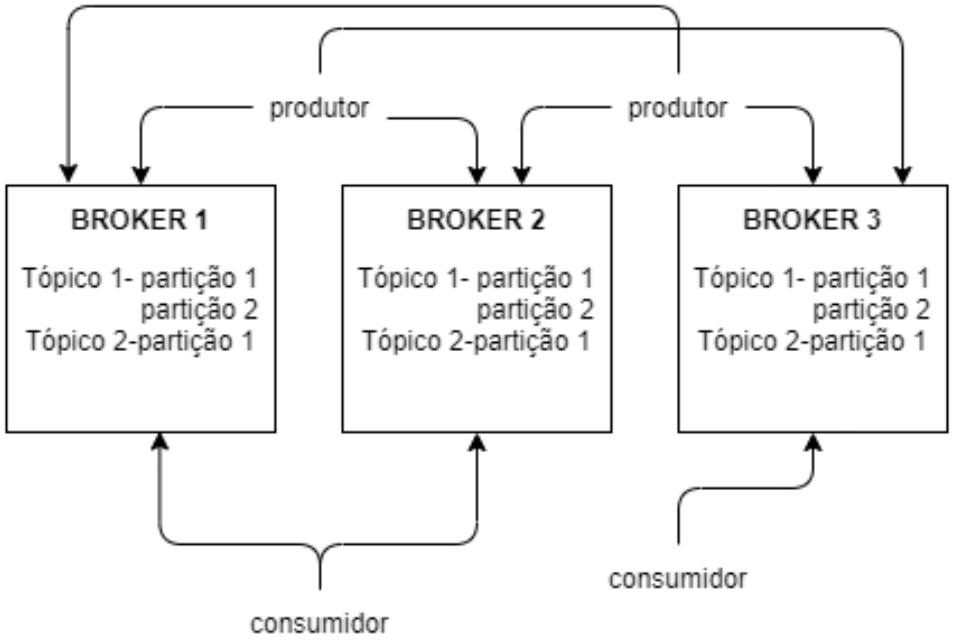


Figura 2.7 – Arquitetura Kafka apresentando os brokers
Fonte: Adaptado de [25]

O Kafka pode ser usado como um broker de mensagens tradicional. Possui alto rendimento, integrado particionamento, replicação e tolerância a falhas, o que o torna uma boa solução para processamento de mensagens em aplicações de grande escala. O Kafka também pode ser usado para rastreamento de atividades em alto volume. Os logs podem ser publicados em um tópico e podem ser processados em tempo real ou carregados em um sistema de armazenamento, como por exemplo o Logstash [7]. Dessa forma o Apache Kafka apresenta as seguintes vantagens:

- Persistência de mensagens: Para sistemas de troca de dados em tempo real, perdas de informação não podem ocorrer. O Kafka é projetado com estruturas de armazenamento de complexidade O(1), oferencendo alto desempenho em tempo constante, mesmo com grandes volumes de mensagens armazenadas.
- Escalabilidade: O Kafka permite o particionamento de tópicos entre diversos brokers, assim permite que o cluster escale e tenha um alto throughput.

- Replicação: Os tópicos são replicados automaticamente entre os brokers.
- Mensagens ordenados: Cada consumidor recebe informações em ordem devido à arquitetura de log particionada.
- Múltiplos consumidores: Vários consumidores podem se inscrever no mesmo tópico, porque o Kafka permite que a mesma mensagem seja reproduzida por um determinado período de tempo.

2.1.4 Blockchain

O conceito de *Blockchain* foi introduzido juntamente com o Bitcoin no artigo de Satoshi Nakamoto [33], o texto apresenta uma alternativa ao sistema de transações monetárias tradicionais sem a necessidade de intermediários. Propostas de um sistema monetário descentralizado já existiam previamente ao Bitcoin mas encontravam um sério problema conhecido como gasto duplo [11], o qual trata a possibilidade de se gastar duas vezes a mesma moeda, problema que foi solucionado no Bitcoin através do conceito de Blockchain.

A solução proposta por Nakamoto [33] consiste em uma lista encadeada de blocos de transações, que se apresentam em uma linha de tempo, onde as transações são confirmadas por agentes anônimos, denominados mineradores, através de computação distribuída que utiliza de criptografia para gerar a cadeia de blocos e a autenticidade das transações realizadas. Os mineradores confirmam *hashs* gerados das novas transações em troca de um valor monetário resultante do processo, esse mecanismo é denominado de mineração [26].

Cada bloco está conectado a apenas um bloco anterior a ele. Cada bloco possui um cabeçalho com metadados, além dos dados das transações. A identificação de cada bloco é feita pelo *hash* criptográfico de seu cabeçalho. Cada bloco faz referência a apenas um bloco anterior, chamado de bloco pai. A referência ao bloco pai é feita com o *hash* do cabeçalho do bloco pai. Ou seja, cada bloco contém o *hash* do bloco pai em seu próprio cabeçalho [3] além das seguintes informações:

- hora de criação (timestamp);
- *nonce*, explicado com mais detalhes em seguida;
- hash da *Merke Tree root* formada a partir das transações do bloco;

A sequência de hashes que liga cada bloco ao seu pai cria uma cadeia de blocos que faz o caminho de volta até o primeiro bloco do sistema, denominado bloco gênese. A mudança da identidade de um bloco geraria um efeito cascata da mudança da identidade de todos os blocos subsequentes a ele. Esse efeito cascata constitui a base da segurança do sistema com relação à inviolabilidade das informações registrada [3].

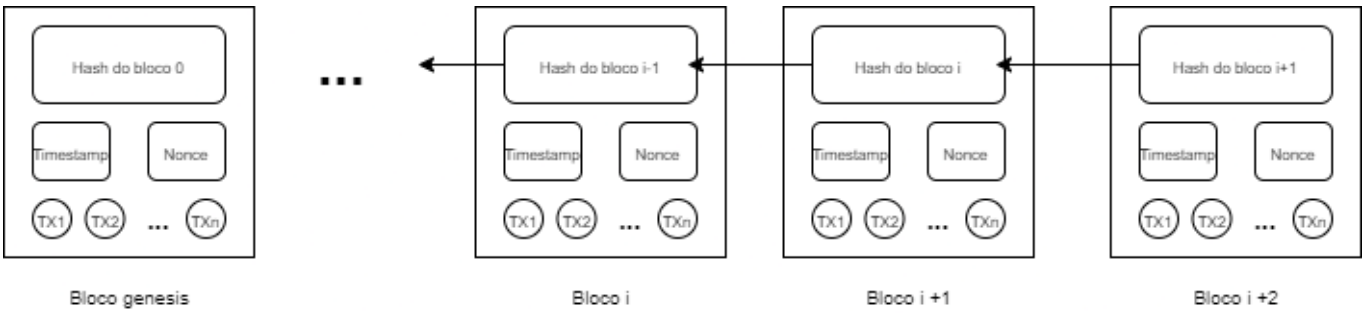


Figura 2.8 – Cadeia de blocos. Observamos um encadeamento temporal dos blocos dada a utilização do *hash* do bloco anterior para a elaboração do bloco posterior.
Fonte: Adaptado de Zeng [49].

Por se tratar de uma arquitetura distribuída e descentralizada chegar a um consenso entre todos os nós é um trabalho árduo. Na Blockchain consenso é visto como uma variação do Problema dos Generais Bizantinos [27]. Nesse problema um grupo de generais que comandam o exército bizantino circundam a cidade. O ataque falharia se apenas parte dos generais atacar a cidade. Os generais precisam se comunicar para alcançar um acordo sobre o ataque ou não. No entanto, pode haver traidores em generais, assim o traidor poderia informar diferentes decisões para diferentes generais. Nesse caso temos um ambiente sem confiança, semelhante a Blockchain. Assim atingir consenso se torna um desafio. Não existe nenhum nó central para validar as informações e os nós não devem confiar uns aos outros. Portanto, protocolos são necessários para garantir que as cadeias em diferentes nós são consistentes, afim de formar uma fonte única de verdade.

A solução proposta por Nakamoto [33] para o consenso é chamada de *Proof-of-work* (PoW). O PoW consiste na realização de uma tarefa, previamente definida, para que o bloco possa ser acrescentado na cadeia. Os agentes responsáveis por realizar essa tarefa são chamados de mineradores. A tarefa deve ser de difícil execução, mas de fácil comprovação de sua realização [46]. Na proposta de Nakamoto [33], a tarefa consiste em produzir um nonce que faça que um determinado número de bits inicias do hash tenha valor igual a zero. Assim a sua comprovação é imediata, sendo necessário apenas verificar a quantidade de zeros iniciais do hash e o esforço além de não ser impossível é facilmente controlado, modificando apenas a quantidade de zeros inicias necessárias [24]. Assim que o nonce é encontrado os outros nós devem mutuamente verificar que o valor está correto. Após a validação o bloco é adicionado na cadeia. Esse processo é chamado de mineração.

O trabalho computacional para validação dos blocos inseridos na cadeia Blockchain é recompensado com uma transferência de recursos aos mineradores. Esse recompensa é formada por taxas de transação pagas pelo remetente da transação e por moedas geradas pela adição de um novo bloco na cadeia [33]. Esse procedimento torna atrativo utilizar os recursos computacionais disponíveis para participar da validação dos blocos, e não para promover ataques à cadeia [39]. O consenso estabelecido pelo Bitcoin poderia ser fraudado caso algum dos validadores controlasse mais da metade dos recursos computacionais da rede, esse processo é conhecido como ataque de cinquenta e um porcento [48].

O corpo do bloco é composto de um contador de transações e das transações em si. A quanti-

dade máxima de transações em bloco depende do tamanho máximo permitido para o bloco e do tamanho das transações. Para validar a autenticidade das transações a blockchain utiliza criptografia assimétrica. A chave privada é utilizada para assinar as transações. As transações assinadas são disseminadas pela rede e são verificadas através da chave pública [49].

Ao longo do tempo o tamanho da cadeia Blockchain tornaria impossível o armazenamento em todos os nós da cadeia. Para resolver o problema de armazenamento da Blockchain foi proposto uma simplificação para a validação de transações. Isso é feito tornando necessário apenas armazenar os cabeçalhos dos blocos e não todas as transações. Para isso é utilizado uma estrutura denominada Árvore de Merkle, conhecida como árvore binária de hash, é construída a partir dos seus vértices folhas. Cada folha guarda o hash de uma das transações existentes no bloco. Os hashes de pares de vértices são computados recursivamente, a partir dos vértices folhas em direção à raiz, até que se obtenha um único hash, o qual constitui o resumo de todas as transações existentes no bloco. Esse hash final é então guardado no cabeçalho do bloco, denominado raiz da árvore de Merkle. No caso de o número de transações ser ímpar, a última transação é duplicada, obtendo sempre uma árvore binária cheia [12].

Apesar da Blockchain se apresentar como uma boa solução para sistemas descentralizados ela apresenta alguns problemas para ser utilizada em dispositivos IoT. Um dos principais problemas é que a mineração da Blockchain requer uma grande quantidade de poder de processamento. Muitos dispositivos IoT não têm a energia necessária para isso. A seguir será apresentado uma solução descentralizada específica para ambientes IoT que promete superar os problemas apresentados para a Blockchain.

2.1.5 Tangle

A tecnologia Tangle surgiu para implementação da criptomoeda Iota [35], então projetada em 2014 para a indústria de IoT. Sua aplicabilidade, no entanto, não se restringe a pagamentos eletrônicos, mas envolve um escopo bem mais abrangente, tal como ocorre com a tecnologia Blockchain. Diferentemente da Blockchain, o Tangle utiliza um Grafo Acíclico Direcionado (do inglês, Directed Acyclic Graph DAG), onde cada nó do grafo representa uma transação.

O Tangle é considerado uma melhor abordagem para aplicações IoT devido a alguns aspectos comentados a seguir [29]. O primeiro aspecto é a inexistência de recompensas de mineração, sendo extremamente útil para micro transações entre dispositivos IoT [15]. Isso é possível porque o processo de adição de uma transação na rede é diferente do Blockchain. Para que uma nova aresta seja criada no grafo o nó emissor da transação precisa primeiro aprovar duas transações emitidas previamente. Essas aprovações são representadas por arestas direcionadas, como mostrado na Figura 2.9. Se não houver um vértice adjacente entre a transação A e a transação B, mas há um caminho de comprimento pelo menos dois de A à B, dizemos que A indiretamente aprova B. Quanto maior é o número de validações que uma transação possui, mais confiável é a validade dessa transação. O bloco gênese é aprovado por todas as transações, diretamente ou indiretamente, como visto na Figura 2.10.

O processo de escolha de qual transação será verificado é chamado de *tip selection algorithm*.

Para resistir a vários tipos de ataques [35], três tipos de algoritmos são propostos em [35]: *Uniform Random*, *Unweighted*, *Random Walk* e *Weighted Random Walk*. A partir da análise proposta em [35] a seleção de ponta mais avançada algoritmo é o Weighted Random Walk.

O algortimo de Weighted Random Walk é uma aplicação de *Markov Chain Monte Carlo* (MCMC) [8]. A ideia principal é que passeios multi-aleatórios são iniciados no meio do grafo do Tangle. Um caminho aleatório irá em direção às novas transações seguindo a direção oposta no DAG. Como uma transação no Tangle poderia ter várias transações de verificação direta, o peso cumulativo de uma transação v no Tangle é número total de transações verificando v direta ou indiretamente. Quanto mais peso cumulativo uma próxima transação tiver, maior probabilidade de que o caminho aleatório leve até ele [5].

De forma resumida o processo do algoritmo seguem os seguintes passos:

1. Escolhe um intervalo no Tangle, [W; 2W], onde W representa a profundidade ou o comprimento do grafo.
2. Adiciona N partículas para executar caminhos aleatórios nas transações dentro do intervalo definido
3. As N partículas executam o caminho aleatório discreto direto ponderado em direção as novas transações.
4. Escolha as primeiras duas transações encontradas pelo algoritmo

O Tangle se apresenta como uma solução mais escalável do que a Blockchain. Como todos os dispositivos são iguais na rede e como o processo de adição de uma transação na rede requer a validação de duas transações pré-existentes, quanto mais dispositivos na rede, mais rápidas as transações são aprovadas e adicionadas ao gráfico [35]. Além disso, como um nó aprova duas transações antes de sua própria transação ser validada, as transações anteriores tem um peso maior em relação a validade, sendo ainda mais difícil modificar a identidade de uma transação quando comparada com a Blockchain [29].

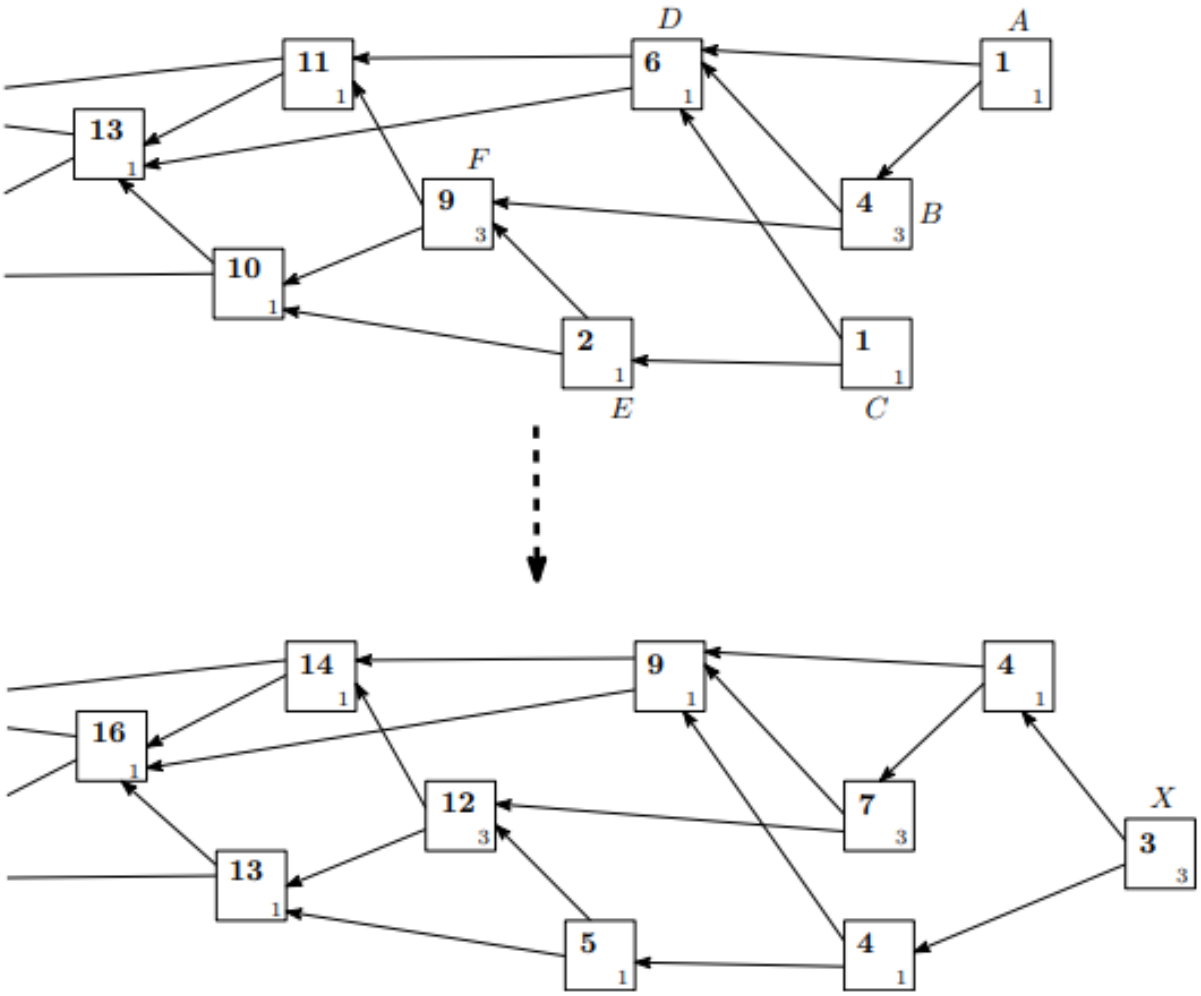


Figura 2.9 – DAG com atribuições de peso antes e depois de uma nova transação emitida, X. As caixas representam as transações, o pequeno número no canto SE de cada a caixa denota o próprio peso e o número em negrito denota o peso cumulativo.
Fonte: [35].

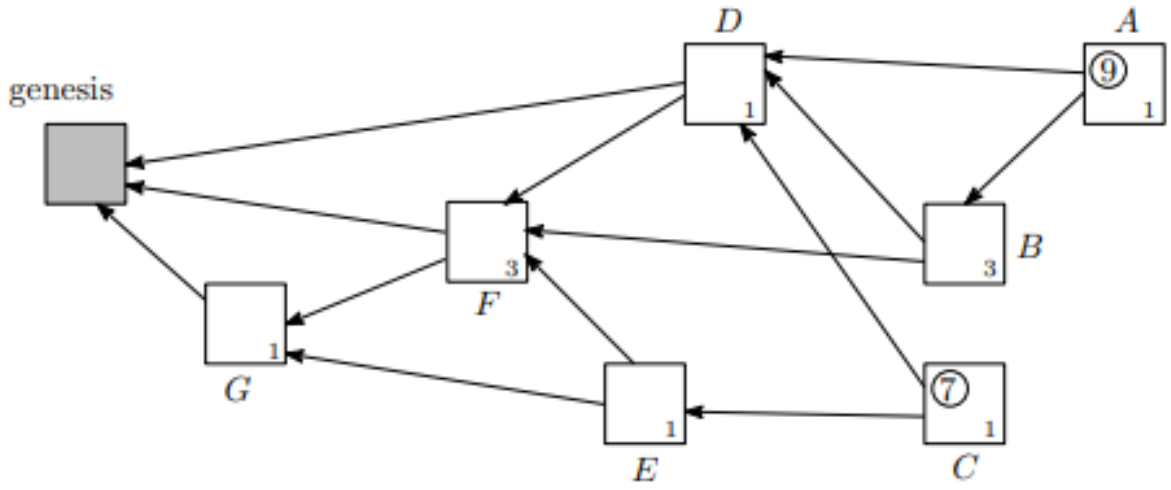


Figura 2.10 – DAG com pesos próprios atribuídos a cada aresta e pontuações calculadas para vértices A e C.
Fonte: Popov [35].

2.2 Revisão Bibliográfica.

Dada a expansão das redes IoT e a necessidade contínua de prover segurança a essas redes e seus sistemas, diversas soluções utilizando sistemas de detecção de intrusão em diferentes arquiteturas já foram propostas. A revisão feita por [31] aborda os principais desafios envolvidos na detecção de intrusão, em especial:

- a problemática do compartilhamento de dados por conta da desconfiança entre as partes envolvidas e também a possibilidade da redução da quantidade de informações compartilhadas pelas mesmas partes devido a preocupação com privacidade, diminuindo assim a otimização do treinamento dos algoritmos de detecção;
- a utilização de um servidor central para prover gerenciamento de confiança e o fato dessa metodologia se tornar cada vez mais complexa a medida que a organização cresce, aumentando assim a possibilidade de ataques internos onde nós que pertencem a organização autorizam acesso à rede para atacantes externos.

Visando a solução dos problemas apresentados por [31], o projeto desenvolvido neste artigo faz uso de uma arquitetura híbrida como forma de apresentar uma resolução aos problemas. O Tangle é utilizado para intensificar a confiança mútua entre as partes envolvidas no compartilhamento de informações e também para favorecer a auditoria e a inviolabilidade das mesmas, além de ser implementado em uma rede distribuída para prover uma gestão de confiança mais robusta, do ponto de vista das controladoras, pelo fato de não possuir um único servidor central como ponto único de falha. No cenário apresentado no artigo as referidas informações faz referência à troca de regras de segurança entre controladoras e nós.

No contexto de infraestruturas IoT tradicionais, [1] apresenta a desconfiança entre os nós do sistema e a existência ponto único de falha como os dois principais empecilhos para a gestão contínua da segurança. Os autores sugerem, como solução, a utilização de uma arquitetura distribuída utilizando *Blockchain* para armazenar as interações do usuário com o ambiente IoT de forma que essas interações sejam persistidas na forma de blocos representando o histórico do nó ou usuário e para isto é utilizado um *token* criptografado que garante a legitimidade do usuário em questão.

Diferentemente da proposta de [1], que utiliza Blockchain para armazenar o histórico dos nós, esta proposta utiliza o Tangle [35], um grafo acíclico dirigido, para endereçar os problemas de escalabilidade da Blockchain [9].

Em seu trabalho [22] mostra a grande frequência de ataques do tipo DDoS em redes IoT e cita a necessidade da implantação de contramedidas que sejam distribuídas na rede com intuito de oferecer coordenação mais eficaz dos nós IoT. Na pesquisa citada a solução desenvolvida é um sistema de detecção de intrusão baseado em *host* (HIDS) que tem como função principal oferecer proteção para o *backbone* da rede IoT. O HIDS proposto é projetado com funcionalidades convencionais como o sistema de autenticação utilizando usuário e senha, busca por assinaturas de ataques conhecidos, verificação de alocação de recursos, processos ativos, porta de serviços abertas e conexões ativas, e além disso oferece como diferencial a possibilidade de interagir com o seu Controlador HIDS para gerir as ações de detecção de intrusão de forma distribuída pela rede em situações de ataque DDoS. No caso da da proposta deste artigo há mais de um fornecedor de regras, diferente do cenário proposto por [22]. A principal vantagem em ter mais de um fornecedor de regras é a velocidade em que elas são difundidas pela rede oferecendo assim uma reposta mais rápidas aos incidentes de segurança como o próprio DDoS e os casos *zero day*.

Em [14], é feita uma proposta de um sistema HIDS por assinatura para detectar ataques e vulnerabilidades nos dispositivos que queiram se conectar a redes IoT. A arquitetura possui agentes HIDS para dispositivos IoT que são responsáveis por realizar os testes determinados pelas regras persistidas no banco de dados do dispositivo em questão. O resultado dos testes é comparado a um resultado esperado e, caso esteja fora dos destes parâmetros, a regra de segurança é executada e a controladora HIDS notificada. A controladora HIDS, por sua vez, é responsável por gerenciar os agentes HIDS no que diz respeito a distribuição e atualização de regras, análise dos alertas, tratamento de ameaças e definição de premissas.

O modelo proposto neste trabalho pretende resolver algumas deficiências apresentadas em [14] utilizando uma proposta híbrida, onde dispositivos IoT trocam informações entre si e com a controladora, onde a controladora pode gerenciar os dispositivos também. A atualização das regras que serão replicadas para os dispositivos não ocorre apenas em nuvem mas ponto a ponto, permitindo que regras criadas e validadas com sucesso em outros ambientes com controladora possam ser replicadas, montando um ambiente federado sempre atualizado.

No trabalho [30] é proposta uma arquitetura para troca de regras entre diferentes redes utilizando contratos inteligentes. Nessa proposta o compartilhamento de regras acontece entre as controladoras de cada rede, uma forma de consenso é adicionado ao contrato inteligente para garantir segurança e legitimidade das regras propagadas pelos nós. O proposto neste trabalho

tem como objetivo garantir segurança e legitimidade das regras também a nível dos dispositivos IoT, implementado políticas de segurança e permitindo o compartilhamento do servidor externo diretamente com os dispositivos IoT através de um canal criptografado que é auditado por uma DLT.

3 Arquitetura Proposta.

O objetivo da solução proposta nesse trabalho é prover um mecanismo de prevenção e detecção de intrusão para gateways, bem como uma arquitetura escalável que permita a gerência e provisionamento de regras para esse sistema. Como parte fundamental desse sistema tem-se uma arquitetura híbrida entre cliente servidor e *peer-to-peer* que permite atualizações de regras nos dispositivos. As regras podem ser oriundas da mesma rede do dispositivo ou diferentes redes sobre diferentes jurisdições, como pode ser visto na Figura 3.1.

A arquitetura é composta por um orquestrador que tem como objetivo gerenciar e provisionar os dispositivos sob sua jurisdição. O sistema de detecção e prevenção de intrusão é executado diretamente nos gateways IoT e é gerenciado pelo orquestrador. O cliente HIDPS não possui banco de dados para armazenar as regras, uma vez que é executado em dispositivos com recursos limitados. Assim um sistema de mensagens em tempo real é necessário para otimização do agente. A comunicação entre orquestrador e gateways IoT é realizada através de uma solução de streaming de eventos em tempo real. Já no contexto peer-to-peer a comunicação entre dispositivos e controladoras externas é realizado através de um canal criptografado no Tangle. A utilização de uma DLT na arquitetura é importante para criar um registro de dados descentralizado em um ambiente onde não é necessário estabelecer confiança entre os participantes.

Dado os requisitos do sistema descritos acima esse trabalho propõe uma arquitetura composta por: gateways IoT executando o cliente HIDPS, um orquestrador para gerenciar os dispositivos e as regras do sistema de detecção de intrusão, um sistema de *streaming* de eventos e um canal de mensagens descentralizado para atualização de regras e controladoras externas que compartilham regras de suas redes com outras. As controladoras externas precisam primeiro serem autorizadas à transmitir novas regras para os dispositivos sobre a jurisdição do orquestrador, permitindo um sistema federado no qual comportamentos anômalos identificados em uma rede externa podem ser repassados para outros dispositivos, permitindo que eles possam reagir a falhas de segurança de forma coordenada e descentralizada conforme as regras são propagadas.

Para uma fácil compreensão da solução proposta, cada parte da arquitetura será explicada separadamente à seguir.

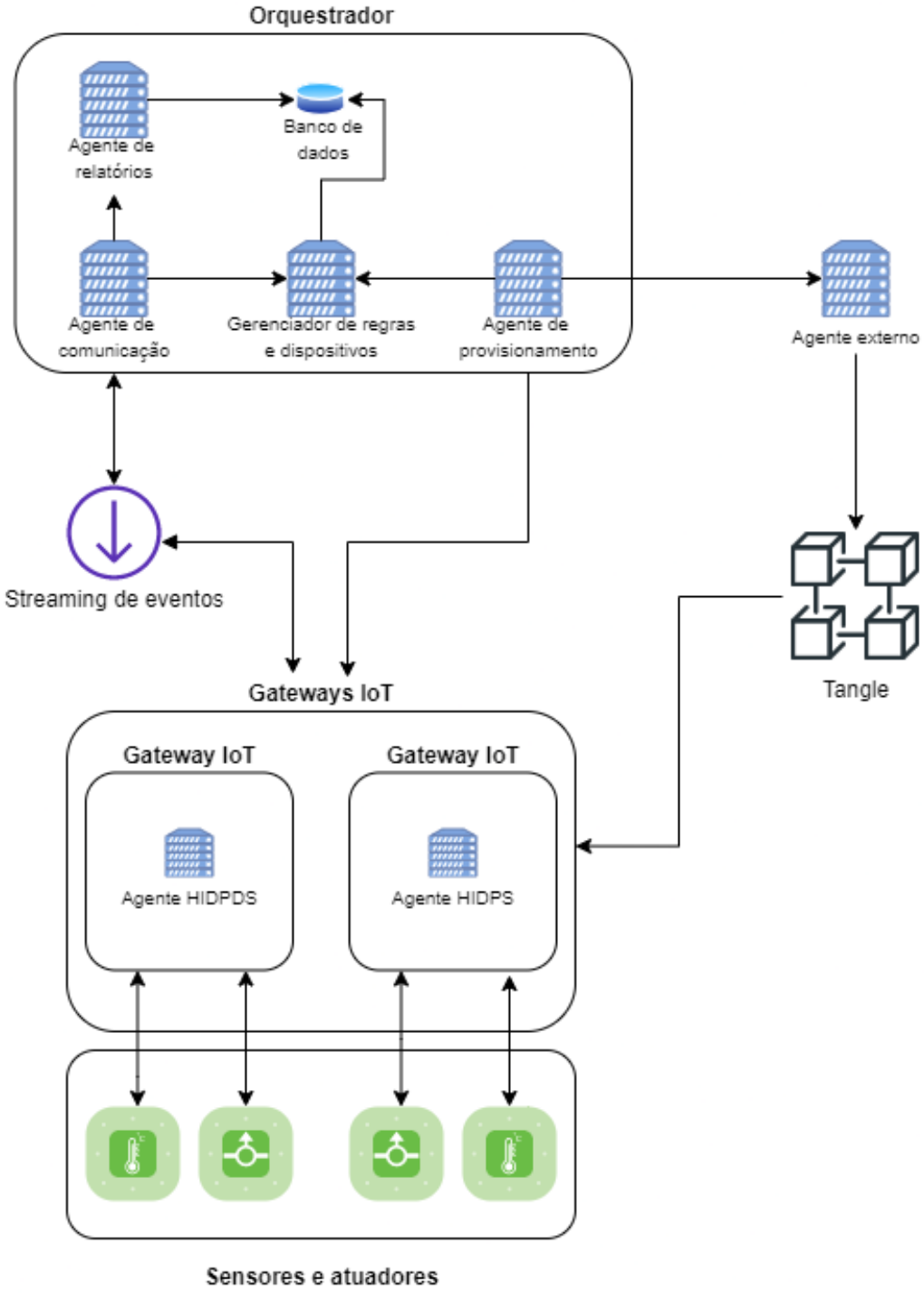


Figura 3.1 – Arquitetura proposta

3.1 Cliente HIDPS

O cliente HIDPS é composto por uma série de *scripts*, que são executados pelo gateway IoT. A sua função é detectar vulnerabilidades presentes no dispositivo. Para que isso ocorra o cliente HIDPS recebe uma série de regras fornecidas pelo orquestrador, cada regra tem como objetivo

verificar se o dispositivo apresenta uma vulnerabilidade especifica e resolve-lá.

Para que a regra tenha a capacidade de detectar e corrigir vulnerabilidades ela é organizada em duas etapas: a primeira é o teste de caso, onde o cliente executa um ou vários scripts em busca da vulnerabilidade e a segunda é uma ação, que é executada caso a vulnerabilidade seja descoberta pelo teste de caso. Semelhante ao teste de caso uma ação é composta de um ou vários scripts para mitigar a vulnerabilidade. Porém, antes de uma regra ser executada é necessário verificar se o dispositivo tem permissão para executar os scripts associados à ela. Todo dispositivo possui uma política criada e gerenciada pelo orquestrador, nessa política o orquestrador indica quais os scripts cada dispositivo tem permissão para executar. Essa solução foi criada para evitar que possíveis scripts maliciosos sejam executados pelo dispositivo. Caso o dispositivo não tenha permissão para executar determinado script, o orquestrador é notificado e decide se a política do dispositivo deve ser alterada para permitir a execução de determinado script. Uma regra pode ser executada apenas uma vez ou de forma periódica. A periodicidade é definida pelo orquestrador e no momento que é recebida pelo dispositivo é verificado se existe periodicidade, caso exista é criado um CRON Job com a periodicidade fornecida. No final do ciclo de execução da regra, um relatório é gerado e enviado ao orquestrador. Na Figura 3.2 é apresentado o fluxo completo do agente HIDPS. É importante notar que o cliente HIDPS não possui um banco de dados para armazenar regras. Dado que HIDPS proposto nesse trabalho é executado em Gateways IoT, que geralmente são dispositivos com restrição de recursos, o cliente é simplificado para evitar sobrecarregar os recursos do sistema.

Na Figura 3.3 é apresentado um exemplo de uma regra. Nesse caso a regra tem como objetivo verificar se a conexão *Telnet* é permitida no dispositivo, e caso seja detectado a existência do processo telnet ele é interrompido pela ação. Nesse caso, a regra possui apenas um script tanto para o teste de caso quanto para ação. A regra possui periodicidade definida para ser executada todos os dias as quatro da manhã. A sintaxe da periodicidade definida na regra é a mesma usada pelo CRON. Caso a periodicidade definida fosse zero, a regra seria executada apenas no momento que for recebida pelo dispositivo. É importante notar também a presença do campo *name* na regra, é uma forma de sumarizar para o administrador qual o objetivo da regra. As regras são criadas no formato *YAML* (*Yet Another Markup Language*) ou *JSON* (*JavaScript Object Notation*).

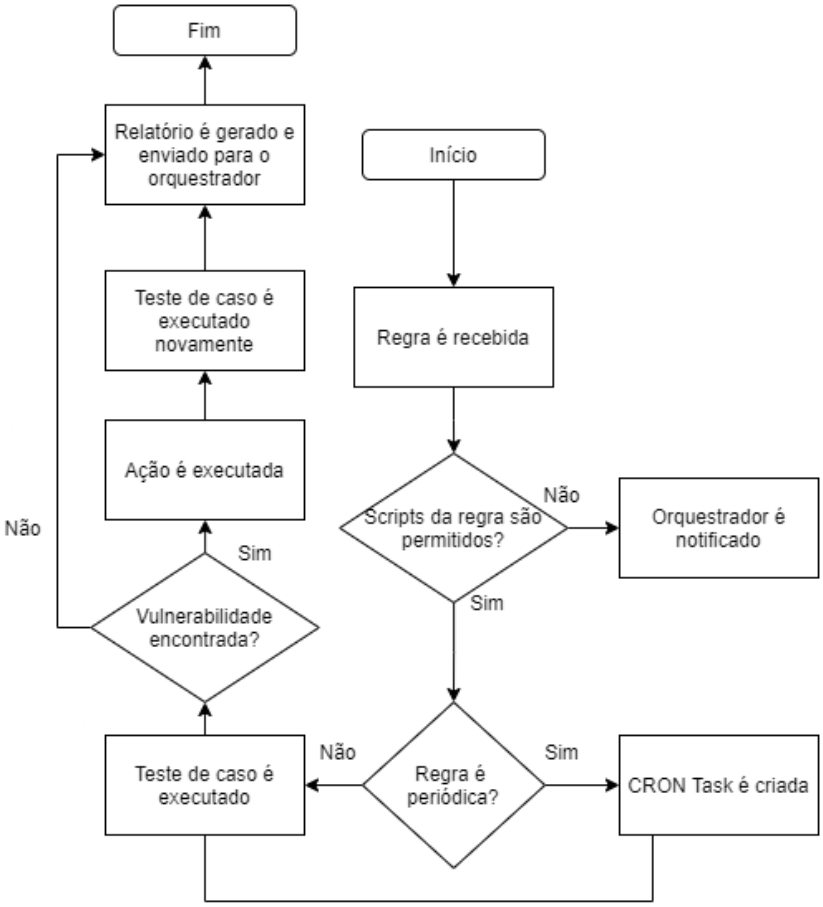


Figura 3.2 – Fluxograma apresentando o processo de execução de uma regra pelo cliente HIDPS.

```
telnet_rule.yaml U hids_client/telnet_rule.yaml/...
1   name: "Check and stop Telnet process"
2   # Executes at 4:00 every day
3   periodicity: '0 4 * * *'
4
5   test_case:
6     scripts:
7       - check_telnet.sh
8
9   action:
10    scripts:
11      - stop_telnet.sh
12
```

Figura 3.3 – Regra para verificar processo de Telnet

Na Figura 3.4 temos o script que verifica se o processo de Telnet está sendo executado. É possível observar que o script retorna um objeto contendo algumas informações importantes:

- mensagem: usada no relatório para o orquestrador
- status: indica se o teste de caso foi executado com sucesso ou não, sucesso indica que a vulnerabilidade foi encontrada;

Caso o status retornado pelo script do teste de caso seja positivo, a ação é executada. É importante salientar que no cenário em que um teste de caso é composto por vários scripts, todos precisam ter um status positivo para que a ação seja executada. Na Figura 3.5 temos o script com o objetivo de parar o processo de Telnet. Depois que a ação é executada, os testes de caso são executados novamente, esperando um resultado negativo, para comprovar que a vulnerabilidade foi mitigada. Ao final o relatório de execução da regra é gerado e enviado ao orquestrador. Caso após a ação ser executada o teste de caso continue acusando positivo, é incluindo no relatório para o orquestrador e cabe ao administrador investigar e tomar as ações necessárias.

check_telnet.sh U hids_client/scripts/test_cases/check_telnet.sh

```
1  #!/bin/bash
2
3  # Check if telnet.d is running
4  if /etc/init.d/inetd  status | grep -q 'not running'; then
5    echo '{"message": "telnet not running", "status": "false"}'
6
7    exit 1
8  else
9    echo '{"message": "telnet running", "status": "true"}'
10 fi
11
```

Figura 3.4 – Teste de caso para verificar se processo de Telnet está sendo executado

stop_telnet.sh U hids_client/scripts/actions/stop_telnet.sh

```
1  #!/bin/bash
2
3  # Stop telnet service
4  /etc/init.d/inetd stop
```

Figura 3.5 – Ação para encerrar processo de Telnet

A política de segurança que indica quais scripts podem ser executados por um dispositivo é especificada através de um arquivo YAML, onde cada regra tem duas entradas:

- o nome do script para fácil identificação da regra;
- hash do script, para garantir que caso haja alterações no script o mesmo não seja executado.

A Figura 3.6 traz um exemplo de política de um dispositivo, onde ele tem permissão de executar quatro scripts:

- *check sshd*;
- *check telnet*;
- *stop ssh*;
- *stop telnet*.

Antes de cada script ser executado é gerado o hash do mesmo e verificado se ele existe na política. Caso não exista, o script não é executado e o orquestrador é acionado.

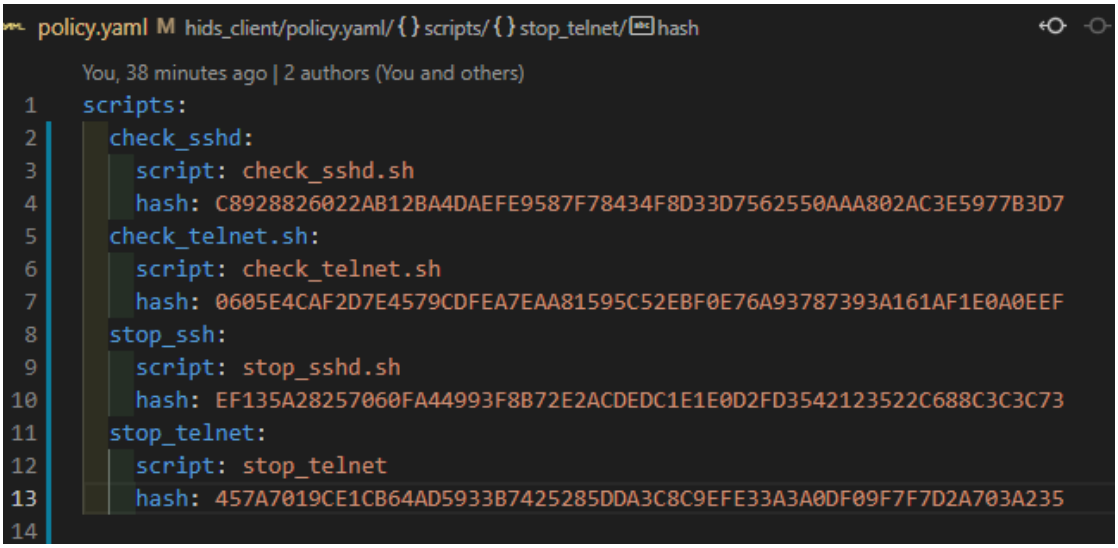


Figura 3.6 – Exemplo da política de segurança para o dispositivo

3.2 Orquestrador

Na arquitetura proposta o orquestrador tem a função de gerenciar e provisionar os dispositivos. O orquestrador é composto por vários agentes trabalhando juntos para cumprir seus objetivos, cada um desses agentes será abordado em detalhes em seguida.

3.2.1 Agente de provisionamento

Este agente é responsável por provisionar Gateways IoT com o Agente HIDPS e atualizar suas políticas de segurança. Este agente automatiza este processo através do Ansible [2]. A automação no processo de provisionamento do cliente HIDPS traz facilidades nos momentos em que se torna

necessário provisionar novamente os dispositivos, por exemplo, no caso de atualizações da política de segurança do cliente HIDPS.

Os dispositivos são provisionados e atualizados em grupos gerenciados pelo administrador por meio do agente Gerenciador de Regras e Dispositivos. Os dispositivos são separados em grupos para que diferentes políticas e regras de segurança possam ser aplicadas a diferentes grupos, com o intuito de manter consistência entre dispositivos que possuem funções similares. O provisionamento ocorre via *SSH*, todas as dependências necessárias são instaladas nos dispositivos e o Agente HIDPS é copiado para os dispositivos. A transferência de arquivos ocorre através do *rsync*, utilizado para sincronizar duas pastas remotas. No momento em que o cliente HIDPS é executado pelos dispositivos, os mesmos recebem as regras de acordo com a política do grupo através do Agente de Comunicação.

A Figura 3.7 apresenta o fluxo de provisionamento de um grupo de dispositivos. Primeiramente, o administrador registra novos dispositivos no orquestrador, se eles ainda não existirem. Em seguida, os dispositivos que devem compartilhar as mesmas regras e política de segurança são acoplados ao mesmo grupo. Um script para provisionar um grupo é chamado, reunindo todas as informações de dispositivos do banco de dados do orquestrador e criando um inventário para o Ansible.

Em seguida, o *playbook* Ansible é executado. A primeira etapa do playbook é instalar todas as dependências necessárias nos dispositivos, depois que o cliente HIDPS é copiado do orquestrador para os dispositivos e executado. E por último, todas as regras que devem ser aplicadas a este grupo são enviadas para os dispositivos. O código do playbook Ansible pode ser visto no Anexo A.

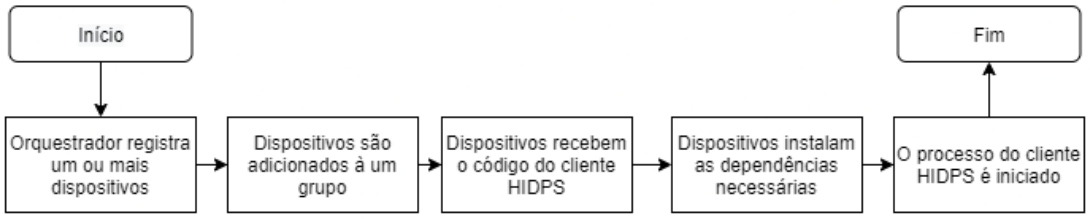


Figura 3.7 – Fluxograma de provisionamento de um grupo de dispositivos utilizando o Ansible

Assim, este agente consegue cumprir sua função de provisionar novos dispositivos com o cliente HIDPS, bem como realizar atualizações das políticas de segurança dos dispositivos.

3.2.2 Gerenciador de regras e dispositivos

Esse agente é responsável por gerenciar as informações sobre todos os dispositivos sob jurisdição do orquestrador e todas as regras da rede. O orquestrador tem um banco de dados no qual as informações sobre o dispositivo e sobre as regras são armazenadas.

A Figura 3.8 traz as tabelas existentes no banco de dados do orquestrador a referência entre elas. A tabela dos dispositivos, *devices*, armazena o IP de cada dispositivo e o grupo a qual ele pertence. Uma tabela intermediária, *device_groups* integra a tabela de dispositivos com a tabela

que representa as regras aplicadas a um determinado grupo, *device_group_rules* e a tabela que representa a política de segurança de um grupo de dispositivos, *devices_groups_scripts*. As regras são controladas através da tabela *rules*, nela é armazenado o nome de cada regra, a periodicidade, se a regra foi compartilhada com outras redes e a referência para a ação e o teste de caso. Os scripts são salvos na tabela *scripts*, onde é armazenado o nome, o endereço onde está armazenado o script e o hash do script. Tabelas intermediarias, *scripts_test_cases* e *scripts_actions*, associam vários scripts há várias ações e testes de caso.

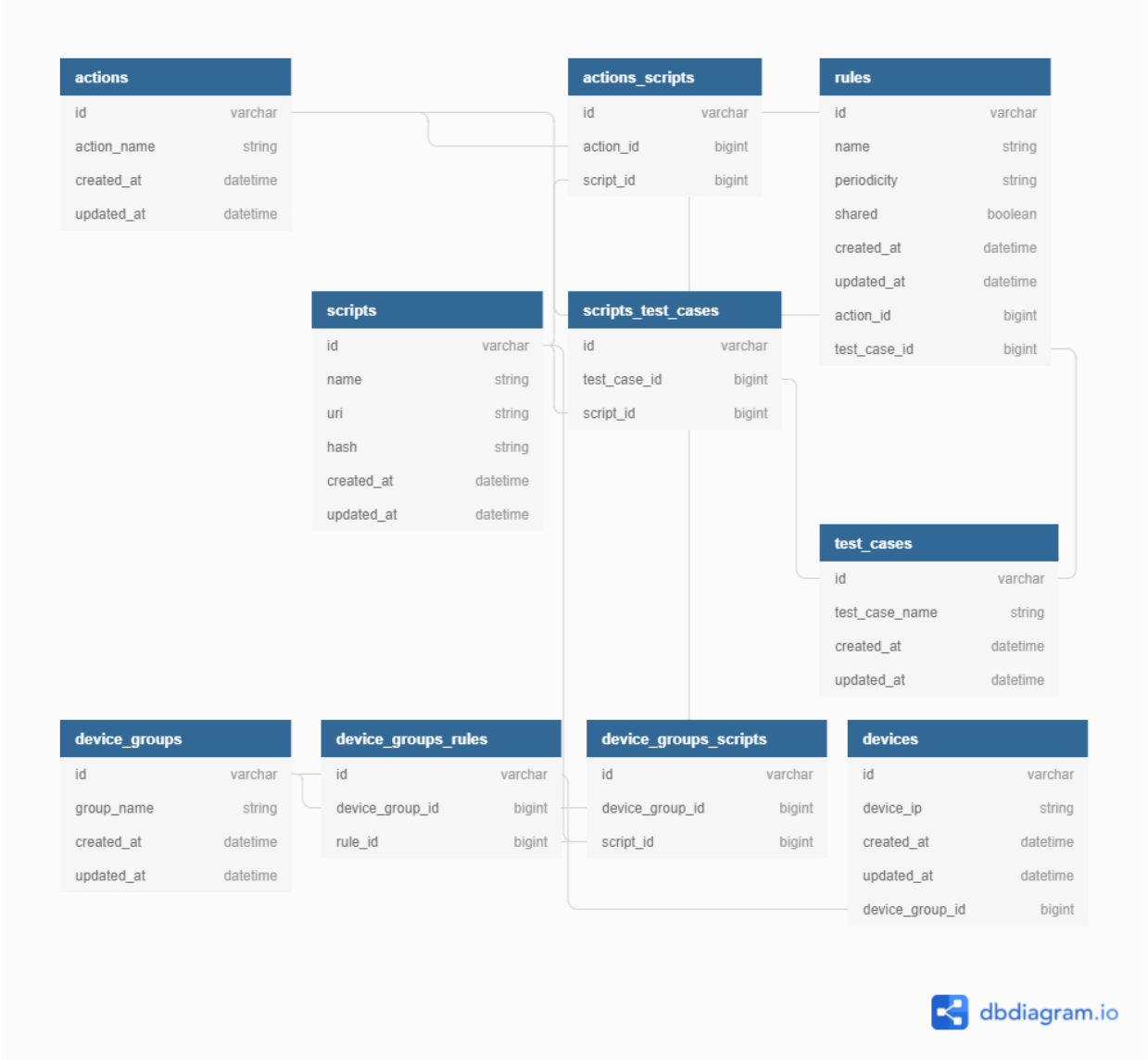


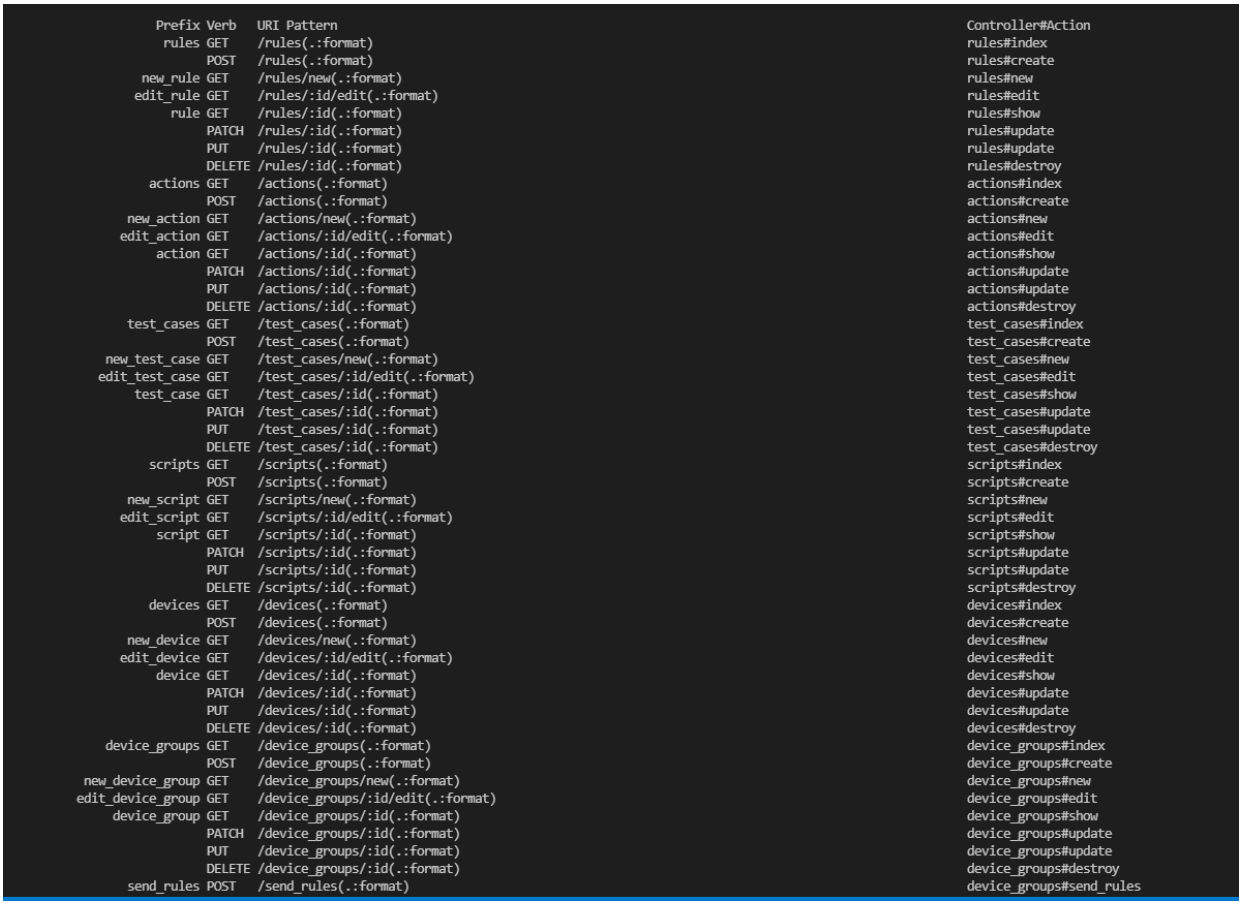
Figura 3.8 – Banco de dados do orquestrador

A interação com este agente acontece através de uma interface *Restful* [38]. A Figura 3.9 mostra as rotas existentes na API. Através de chamadas na API o agente de provisionamento consegue montar um inventário dos dispositivos à serem provisionados, bem como a política de segurança para os mesmos.

Esse agente também conta uma chamada para enviar as regras aos dispositivos, ela é acionada fornecendo o grupo de dispositivos que irá receber as regras, assim como quais regras serão enviadas.

Assim que o grupo e as regras são selecionadas no banco de dados, as informações são enviadas para o agente de comunicação, que envia a regra para os dispositivos.

Assim, este agente consegue cumprir sua função de permitir o gerenciamento de regras e de dispositivos no orquestrador. A sua natureza *Restful* permite que o serviço seja de fácil interação, seja através de chamadas de API ou através de uma interface gráfica.



Prefix	Verb	URI Pattern	Controller#Action
rules	GET	/rules(.:format)	rules#index
	POST	/rules(.:format)	rules#create
new_rule	GET	/rules/new(.:format)	rules#new
edit_rule	GET	/rules/:id/edit(.:format)	rules#edit
rule	GET	/rules/:id(.:format)	rules#show
	PATCH	/rules/:id(.:format)	rules#update
	PUT	/rules/:id(.:format)	rules#update
	DELETE	/rules/:id(.:format)	rules#destroy
actions	GET	/actions(.:format)	actions#index
	POST	/actions(.:format)	actions#create
new_action	GET	/actions/new(.:format)	actions#new
edit_action	GET	/actions/:id/edit(.:format)	actions#edit
action	GET	/actions/:id(.:format)	actions#show
	PATCH	/actions/:id(.:format)	actions#update
	PUT	/actions/:id(.:format)	actions#update
	DELETE	/actions/:id(.:format)	actions#destroy
test_cases	GET	/test_cases(.:format)	test_cases#index
	POST	/test_cases(.:format)	test_cases#create
new_test_case	GET	/test_cases/new(.:format)	test_cases#new
edit_test_case	GET	/test_cases/:id/edit(.:format)	test_cases#edit
test_case	GET	/test_cases/:id(.:format)	test_cases#show
	PATCH	/test_cases/:id(.:format)	test_cases#update
	PUT	/test_cases/:id(.:format)	test_cases#update
	DELETE	/test_cases/:id(.:format)	test_cases#destroy
scripts	GET	/scripts(.:format)	scripts#index
	POST	/scripts(.:format)	scripts#create
new_script	GET	/scripts/new(.:format)	scripts#new
edit_script	GET	/scripts/:id/edit(.:format)	scripts#edit
script	GET	/scripts/:id(.:format)	scripts#show
	PATCH	/scripts/:id(.:format)	scripts#update
	PUT	/scripts/:id(.:format)	scripts#update
	DELETE	/scripts/:id(.:format)	scripts#destroy
devices	GET	/devices(.:format)	devices#index
	POST	/devices(.:format)	devices#create
new_device	GET	/devices/new(.:format)	devices#new
edit_device	GET	/devices/:id/edit(.:format)	devices#edit
device	GET	/devices/:id(.:format)	devices#show
	PATCH	/devices/:id(.:format)	devices#update
	PUT	/devices/:id(.:format)	devices#update
	DELETE	/devices/:id(.:format)	devices#destroy
device_groups	GET	/device_groups(.:format)	device_groups#index
	POST	/device_groups(.:format)	device_groups#create
new_device_group	GET	/device_groups/new(.:format)	device_groups#new
edit_device_group	GET	/device_groups/:id/edit(.:format)	device_groups#edit
device_group	GET	/device_groups/:id(.:format)	device_groups#show
	PATCH	/device_groups/:id(.:format)	device_groups#update
	PUT	/device_groups/:id(.:format)	device_groups#update
	DELETE	/device_groups/:id(.:format)	device_groups#destroy
send_rules	POST	/send_rules(.:format)	device_groups#send_rules

Figura 3.9 – Rotas da API do gerenciador de regras e dispositivos.

3.2.3 Agente de comunicação

O agente de comunicação é responsável por enviar regras para dispositivos e receber relatórios de dispositivos. O agente de comunicação precisa ser capaz de enviar e receber dados sem perda de pacotes para vários dispositivos. Este agente precisa ser capaz de enviar e receber dados sem perda de pacotes para vários dispositivos. Como o HIDPS é executado em dispositivos com recursos limitados, eles podem não ser capaz de receber mensagens exatamente da mesma forma que são enviadas. Como resultado, nossa solução proposta utiliza o Apache Kafka como o meio de comunicação que possui armazenamento, permitindo que os dispositivos busquem as regras quando puderem, garantindo a integridade dos dados e apresenta a possibilidade de múltiplos leitores para um tópico.

O Kafka também permite que a arquitetura proposta escale horizontalmente com facilidade, sendo necessário aumentar o tamanho do cluster Kafka a medida que se torna necessária. Nos

casos. No Kafka, escalar horizontal significa adicionar mais brokers a um cluster Kafka existente. Isso permite compartilhar a carga no cluster com um número maior de nós individuais, permitindo que o cluster atenda a mais solicitações como um todo. A escala horizontal também torna o cluster mais resiliente a falhas em um único nó: se um broker em um cluster de três nós falhar, os dois nós restantes obterão um aumento de carga de 50%, o que pode causar problemas de disponibilidade no cluster. Se um broker em um cluster de 9 nós falhar, os 8 nós restantes verão apenas um aumento de carga de aproximadamente 13%.

O Kafka também permite o uso de grupos de consumidores, um grupo de consumidores é um mecanismo de agrupamento de vários consumidores em um grupo. Os dados são divididos igualmente entre todos os consumidores de um grupo, sem que dois consumidores de um grupo recebam os mesmos dados. A utilização de um grupo de consumidores permite que o orquestrador consiga escalar horizontalmente para receber relatórios de cada vez mais dispositivos. Os gateways IoT não podem fazer parte de grupos de consumidores, uma vez que todos precisam receber as mensagens referentes à novas regras.

A Figura 3.10 mostra uma *dashboard* do Kafka utilizado na arquitetura. Temos um cluster, *hids-iot*, composto por dois brokers com cinquenta e quatro partições. A troca de dados entre orquestrador e os dispositivos IoT acontece por meio de tópicos específicos. O envio de regras para os dispositivos acontece através de um tópico específico para grupo de dispositivos. Dessa forma as regras podem ser provisionados para cada grupo separadamente. O nome do tópico para os grupos é *rules_group_ <group_id>* onde <group_id é o identificador para cada grupo, por exemplo para o grupo com identificador igual a um o tópico seria *rules_group_1*. Os dispositivos consomem as mensagens do tópico referente ao grupo que pertencem.

Já os relatórios de execução das regras são enviados pelos dispositivos para o orquestrador através de um tópico próprio. Esse tópico é chamado de *devices_rules_report*. Cada dispositivo produz uma mensagem para o tópico com o relatório gerado e o Agente de Relatórios fica constantemente escutando esse tópico. A Figura 3.11 apresenta o fluxo descrito.

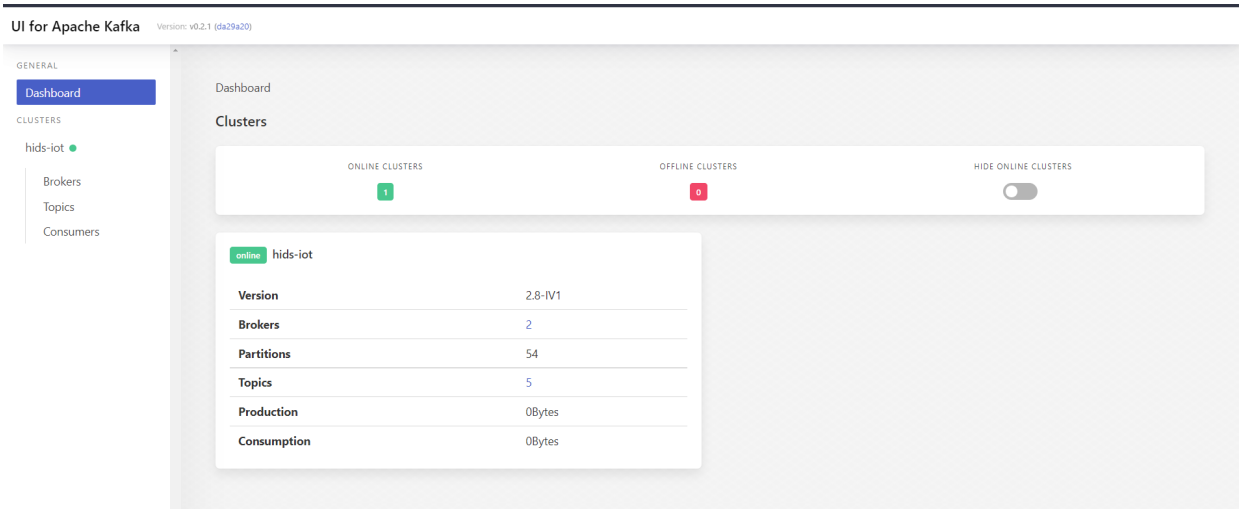


Figura 3.10 – Dashboard do cluster Kafka

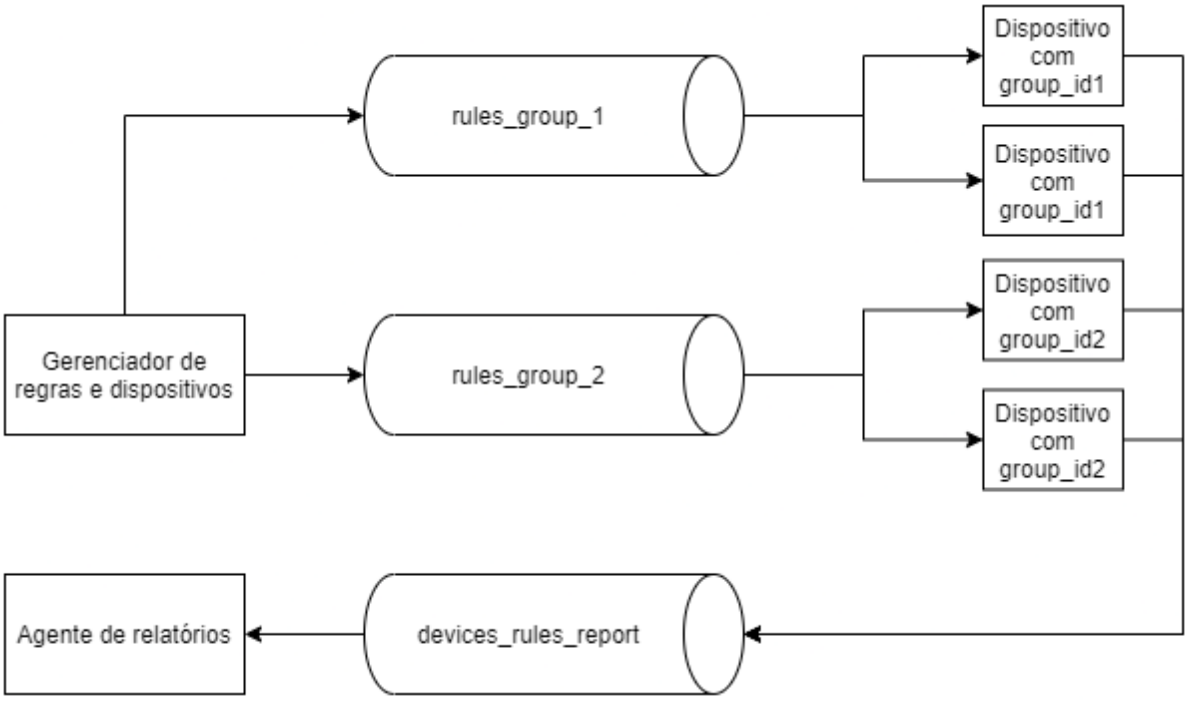


Figura 3.11 – Fluxo do agente de comunicação através de tópicos do Apache Kafka

Na Figura 3.12 temos o exemplo de um relatório gerado pelos dispositivos ao fim da execução de uma regra. Esse relatório é enviado para o agente de relatórios através do tópico *device_rules_report*. Na Figura 3.13 temos um exemplo de um grupo de consumidores do tópico de relatórios, *device_rules_report*. Esse grupo é formado por dois consumidores sob jurisdição do orquestrador. Podemos observar que cada consumidor está consumindo de uma partição diferente, partição zero para o *orchestrator_consumer1* e partição um para o *orchestrator_consumer2* Esse comportamento é o padrão do Kafka, cada consumidor recebe um grupo de partições e as mensagens não são replicadas entre as partições, dessa forma uma mensagem nunca vai ser recebida por dois consumidores. A Figura 3.14 apresenta a atribuição de partições para os consumidores em um grupo.

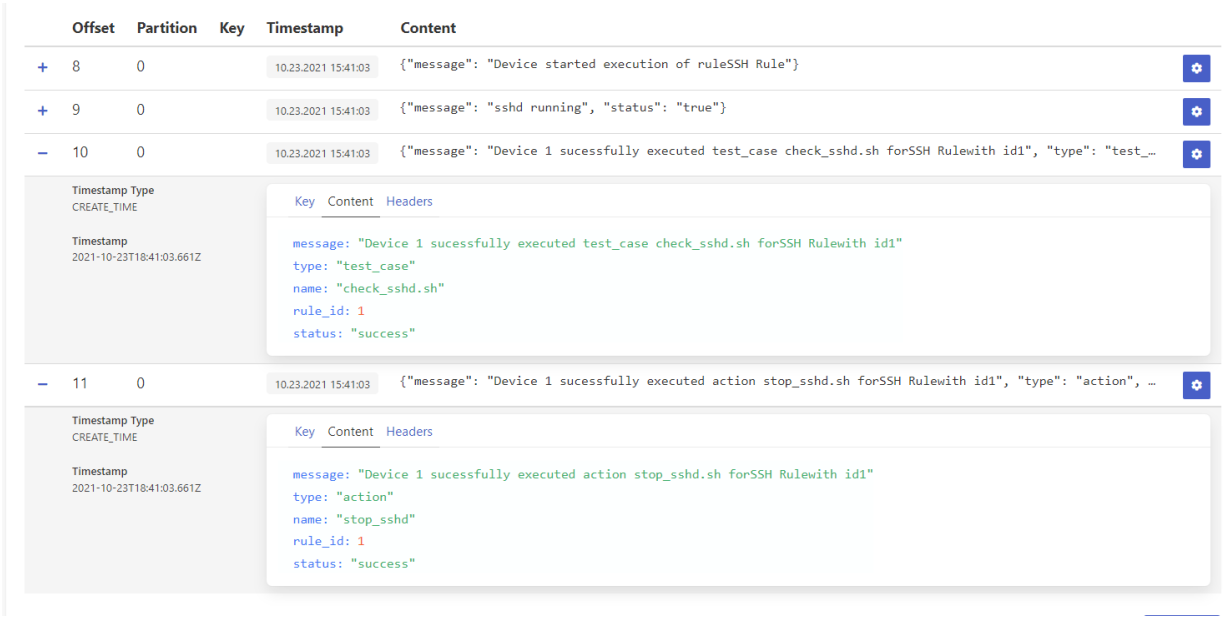


Figura 3.12 – Exemplo de relatório enviado pelos dispositivos após execução de uma regra

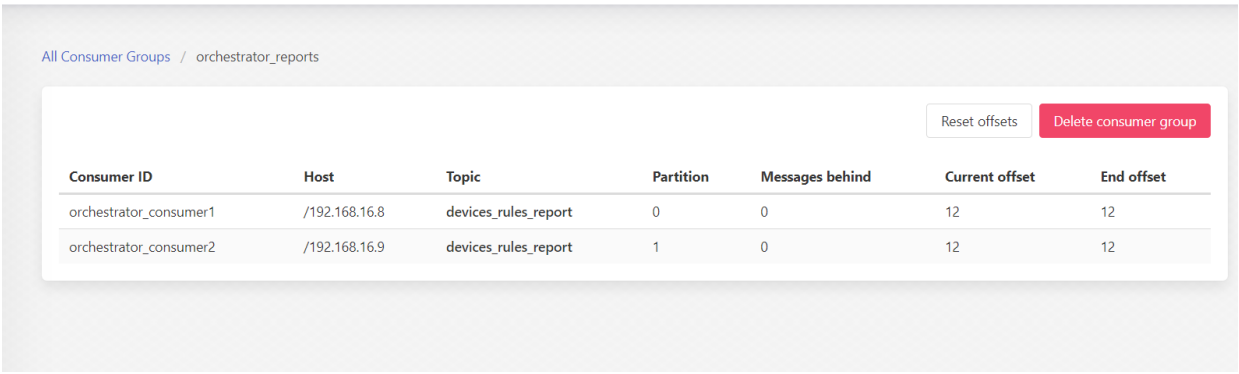


Figura 3.13 – Exemplo de um grupo de consumidores para o tópico de recepção de relatórios

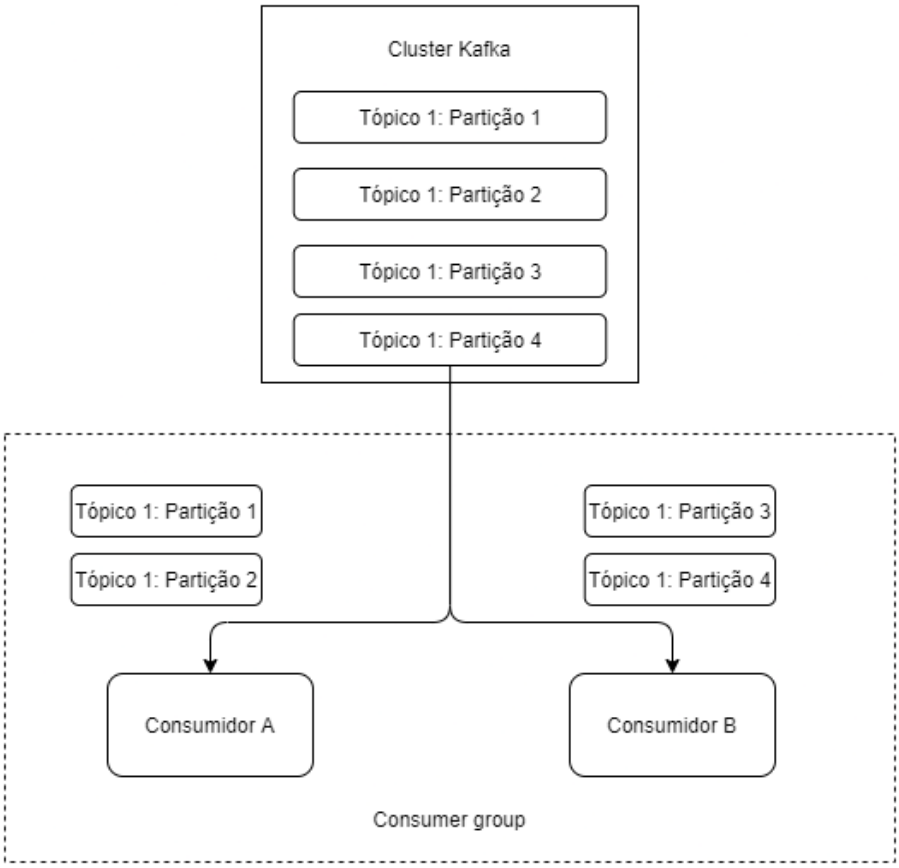


Figura 3.14 – Atribuição de partições de um tópico para um grupo de consumidores

Na Figura 3.16 temos um exemplo de mensagem produzida pelo orquestrador para compartilhamento de regras para um grupo de dispositivos. Podemos observar que todas as informações necessárias para execução da regra pelo cliente HIDPS são fornecidas na mensagem: nome da regra, periodicidade, scripts para os testes de caso e scripts para as ações. Já na Figura 3.15 temos o dispositivo consumindo a mensagem produzida pelo orquestrador.

Com isso o agente de comunicação consegue cumprir sua função de permitir troca de mensagens entre múltiplos dispositivos de forma rápida, com persistência de mensagens e que seja facilmente escalável.

```
bash-4.4# python3 rules_consumer.py
Starting consumer for topic rules_group_1
Consumer started
Received message: ConsumerRecord(topic='rules_group_1', partition=0, offset=9, timestamp=1635012983344, timestamp_type=0, key=None, value=b'{"rule":[{"id":1,"name":"Check ssh service","periodicity":"0","shared":true,"created_at":"2021-10-03T22:53:08.580Z","updated_at":"2021-10-03T22:53:08.580Z","action_id":1,"test_case_id":1}],"action_scripts":[{"id":1,"name":"sshd","uri":null,"created_at":"2021-10-03T22:49:03.540Z","updated_at":"2021-10-03T22:49:03.540Z"}],"test_case_scripts":[{"id":1,"name":"sshd","uri":null,"created_at":"2021-10-03T22:49:03.540Z","updated_at":"2021-10-03T22:49:03.540Z"}]}', headers=[], checksum=None, serialized_key_size=-1, serialized_value_size=458, serialized_header_size=-1)
```

Figura 3.15 – Exemplo de regra sendo consumida pelo dispositivo depois de ter sido publicado pelo orquestrador para um tópico Kafka

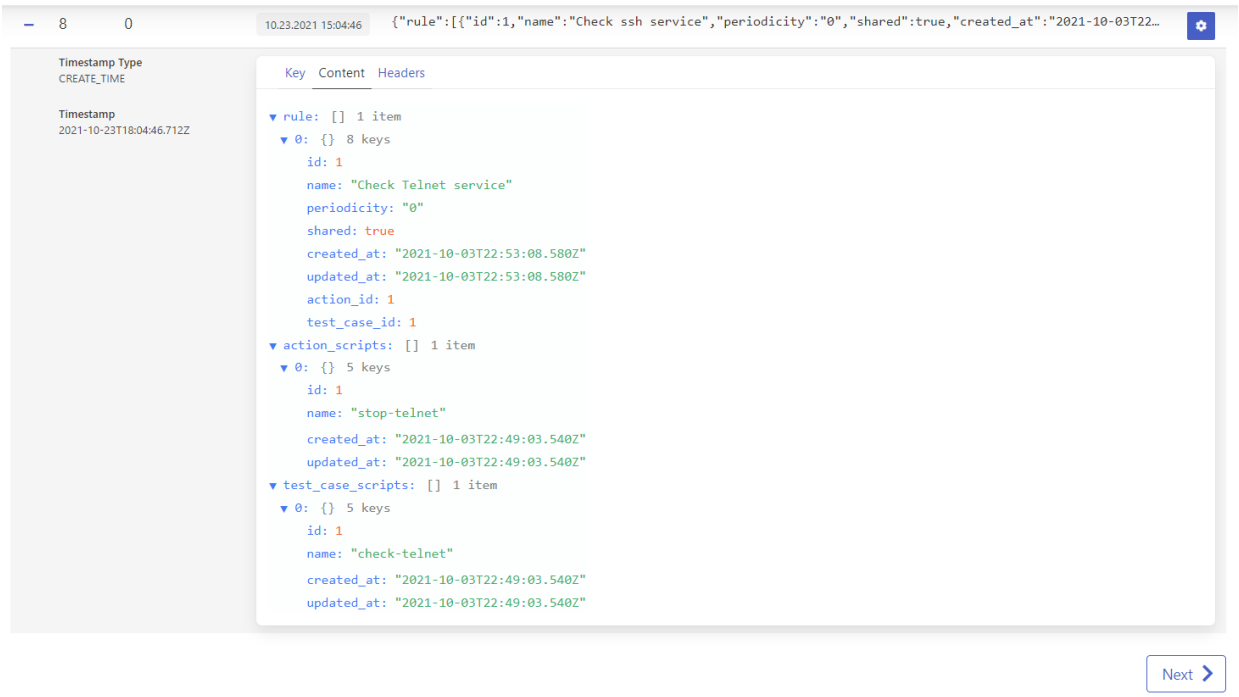


Figura 3.16 – Exemplo de regra publicada pelo orquestrador para um tópico Kafka consumido por um grupo de dispositivos

3.2.3.1 Agente de relatórios

O agente de relatório tem como função receber os relatórios gerados pelos dispositivos e apresentar de uma forma visual para o administrador da rede. Para isso este agente utiliza o *stack* ELK. O ELK consiste em vários componentes de código aberto (ElasticSearch, Logstash e Kibana) e pode ser executado em ambiente de hardware virtual.

O ElasticSearch é uma ferramenta baseada no Apache Lucene para busca e análise de dados distribuído. Logstash é uma pipeline de coleta dinâmica de logs, os dados são enviados para o ElasticSearch. Kibana é um componente para visualização em vários tipos como gráfico, tabela e mapa, etc. Assim, a stack se apresenta como uma solução interessante para visualização e computação de métricas a partir dos relatórios gerados pelos dispositivos IoT.

Na arquitetura proposta, o Logstash é configurado para o tópico do Apache Kafka referente aos relatórios enviados pelo dispositivo, a configuração pode ser vista no Anexo C. A Figura 3.17 apresenta as mensagens que são coletadas pelo Logstash e fornecidas para o ElasticSearch. Cada mensagem indica um momento da execução da regra. Em geral, relatórios são compostos por duas etapas: dispositivo reporta o resultado do teste de caso após execução, caso o teste de caso seja verdadeiro, dispositivo executará a ação e em sequência reportará também o resultado da ação realizada. Todo relatório contém os seguintes campos:

- *message*: Uma mensagem geral que informa o que aconteceu.
- *type*: Identifica qual execução está sendo reportada, teste de caso ou ação.

- *status*: Identifica o status de execução para um teste de caso ou ação, *success* no caso do teste de caso indica que a vulnerabilidade foi encontrada, e para ação significa que os scripts propostos pela ação mitigaram a vulnerabilidade. O status *not found* é usado apenas para o teste de caso e indica que a vulnerabilidade não foi encontrada no dispositivo. O status *error* informa que algum erro aconteceu na execução do teste de caso ou script. E por último o status *not mitigated* indica que a ação foi executada com sucesso, mas na validação após sua execução a vulnerabilidade ainda foi detectada.
- *device_ id*: Identifica qual o dispositivo está executando a regra.
- *rule_ id*: Identifica qual regra está sendo executada através do seu identificador.
- *rule_ name*: Identifica qual regra está sendo executada através do seu nome.

Na Figura 3.18 temos um exemplo de um relatório de um teste de caso que foi bem sucedido. É possível verificar que se trata de um teste de caso através do campo *type* e o campo status indica que foi um sucesso, ou seja, uma vulnerabilidade foi encontrada. A mensagem fornecida no campo *message* sumariza essas informações.

Na Figura 3.19 temos um relatório para execução da ação decorrente do sucesso do teste de caso anterior. É importante relembrar que após a execução de uma ação, o teste de caso é acionado novamente para garantir que a vulnerabilidade foi mitigada. O relatório da ação só indica sucesso para o caso em que foi confirmado a mitigação da vulnerabilidade. Caso a ação tenha sido executada com sucesso mas o teste de caso ainda encontrou a vulnerabilidade, o relatório indica esse cenário através do status *not mitigated*, um exemplo pode ser visto na Figura 3.20.

Figura 3.17 – Exemplo de mensagens no relatório apresentado pelo Kibana

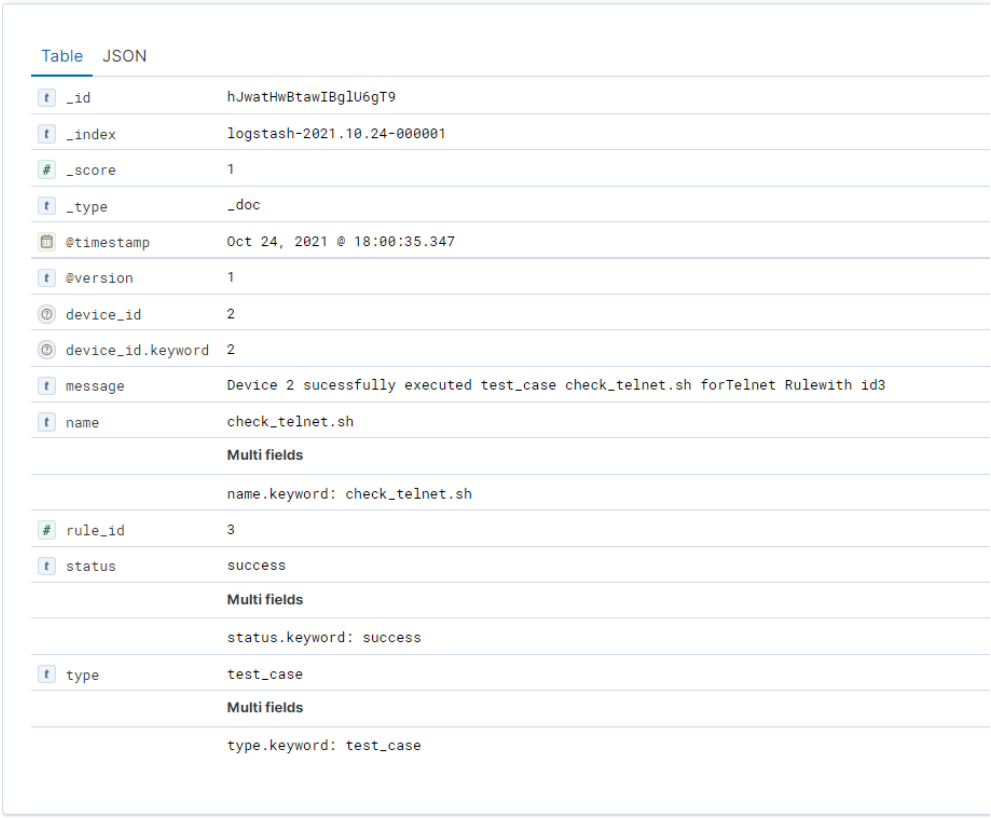


Table JSON

t _id	hJwatHwBtawIBglU6gT9
t _index	logstash-2021.10.24-000001
# _score	1
t _type	_doc
@timestamp	Oct 24, 2021 @ 18:00:35.347
t @version	1
device_id	2
device_id.keyword	2
t message	Device 2 sucessfully executed test_case check_telnet.sh forTelnet Rulewith id3
t name	check_telnet.sh
	Multi fields
	name.keyword: check_telnet.sh
# rule_id	3
t status	success
	Multi fields
	status.keyword: success
t type	test_case
	Multi fields
	type.keyword: test_case

Figura 3.18 – Exemplo de relatório para um teste de caso no qual uma vulnerabilidade foi detectada

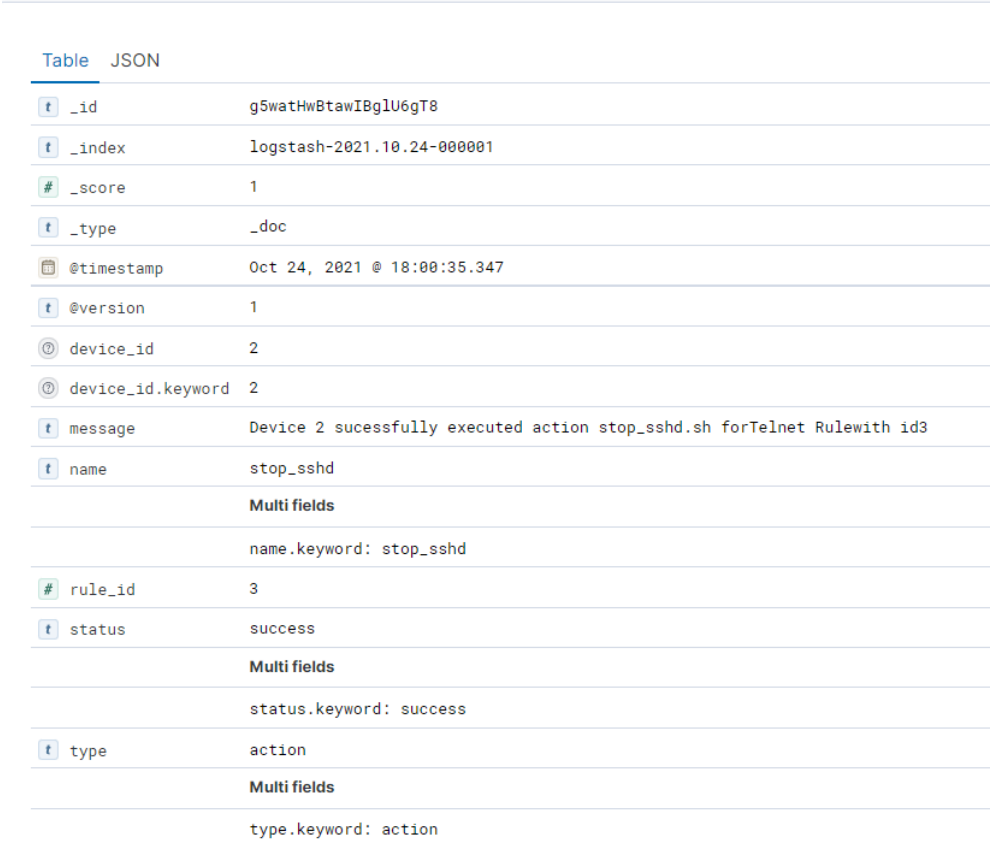


Table JSON

t _id	g5watHwBtawIBglU6gT8
t _index	logstash-2021.10.24-000001
# _score	1
t _type	_doc
@timestamp	Oct 24, 2021 @ 18:00:35.347
t @version	1
device_id	2
device_id.keyword	2
t message	Device 2 sucessfully executed action stop_sshd.sh forTelnet Rulewith id3
t name	stop_sshd
	Multi fields
	name.keyword: stop_sshd
# rule_id	3
t status	success
	Multi fields
	status.keyword: success
t type	action
	Multi fields
	type.keyword: action

Figura 3.19 – Exemplo de relatório para uma ação que foi bem sucedida em mitigar a vulnerabilidade

Table JSON

t _id	iJxJtHwBtawIBglU-wQZ
t _index	logstash-2021.10.24-000001
# _score	1
t _type	_doc
@timestamp	Oct 24, 2021 @ 18:51:59.626
t @version	1
t device_id	2
	Multi fields
	device_id.keyword: 2
t message	Device 2 executed action stop_telnet executed sucessfully but vulnerability was still found. For ruleTelnet Rulewith id3
t name	stop_telnet
	Multi fields
	name.keyword: stop_telnet
# rule_id	3
t status	not mitigated
	Multi fields
	status.keyword: not mitigated
t type	action
	Multi fields
	type.keyword: action

Figura 3.20 – Exemplo de relatório para uma ação que foi executada mas vulnerabilidade não foi mitigada

Através dos relatórios recebidos pelo Logstash, é possível gerar *dashboards* no Kibana para uma fácil visualização dos dados pelo administrador. Dado as informações fornecidas no relatório, o administrador consegue facilmente acompanhar e gerenciar os dispositivos na rede. Por exemplo, a Figura 3.21 apresenta um gráfico de contagem da quantidade de mensagens com um status específico para cada regra. O administrador também tem a possibilidade de criar gráficos para analisar como os dispositivos estão executando as regras ao longo do tempo, por exemplo a Figura 3.22 apresenta um gráfico da execução de cada regra ao longo do tempo.

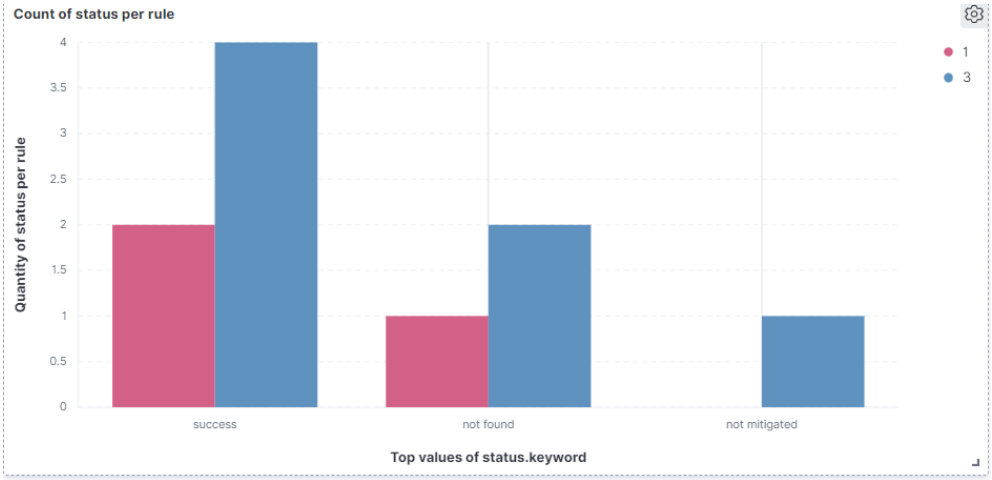


Figura 3.21 – Gráfico no Kibana apresentando uma contagem da quantidade de status de cada regra para todos os dispositivos que executaram a regra.

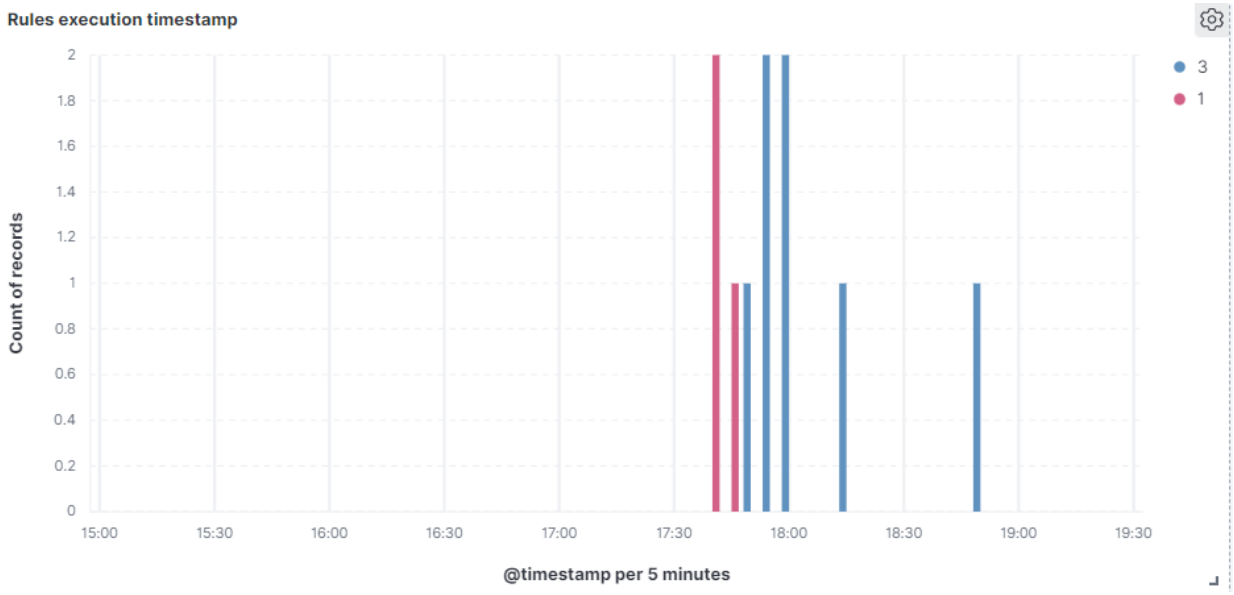


Figura 3.22 – Gráfico no Kibana apresentando as regras executadas pelos dispositivos ao longo do tempo. Na imagem destacam-se as regras com id 1 e 3.

Em suma, o agente de relatórios tem como objetivo oferecer uma maneira do administrador visualizar o estado do cliente HIDPS na rede. A utilização do stack ELK permite que as mensagens sejam recebidas através do Apache Kafka, solução utilizada pelo agente de comunicação, e sejam organizadas como mensagens puras ou como gráficos e tabelas através das dashboards oferecidas pelo Kibana.

3.2.4 Agente Externo

O agente externo é um servidor fora da jurisdição do orquestrador e seu objetivo é compartilhar e receber regras de diferentes redes. O compartilhamento de regras entre os sistemas IDS em diferentes cenários é importante para aumentar o escopo de possíveis vulnerabilidades, em vez de ter uma única fonte de verdade, aumentando a precisão [51]. Porém é necessário algumas restrições e gerenciamento das controladoras externas, como autorização para que a controladora externa se comunique com os dispositivo, um canal entre dispositivos e controladora e algumas políticas sobre o que o servidor externo tem jurisdição sobre.

Para entrar em uma rede, o Agente Externo precisa primeiro ser autorizado pelo orquestrador da rede. Esse processo é dado da seguinte forma. Primeiro o agente externo requisita entrada na rede para o orquestrador, cabe ao administrador aceitar a requisição, no caso de aceite o orquestrador cria as permissões para o agente externo e fornece credencias de acesso para o mesmo. As permissões são declaradas a partir de uma política, existem duas políticas:

- restritiva, onde o servidor externo não pode criar novos scripts no dispositivo;
- completa, em que o servidor pode agir de forma semelhante ao orquestrador, criando novas políticas e alterando a permissão de execução de scripts dos dispositivos.

A autenticação do agente externo pelo orquestrador ocorre da seguinte forma. Primeiramente, o orquestrador adiciona os dados sobre o agente externo no seu banco dados, além de criar uma chave de API, *api_key* e uma segredo, *api_secret*, que é utilizado para criptografar as mensagens trocadas entre o orquestrador e o agente externo. Assim que o orquestrador salva as informações ele as compartilha com o servidor externo. O servidor externo então armazena a chave e o segredo da API.

Em toda requisição o agente externo fornece a chave de API, que é validada pelo orquestrador. Após confirmar que a chave de API está correta, o orquestrador criptografa a mensagem utilizando o segredo, quando o agente externo recebe a mensagem ele descriptografa a mensagem utilizando o mesmo segredo e assim consegue ter acesso aos dados. O fluxo descrito pode ser visualizado na Figura 3.23 e na Figura 3.24 temos um exemplo de mensagem do orquestrador para o agente externo no momento de cadastro, podemos observar o chave e o segredo da API sendo fornecidos ao agente externo.

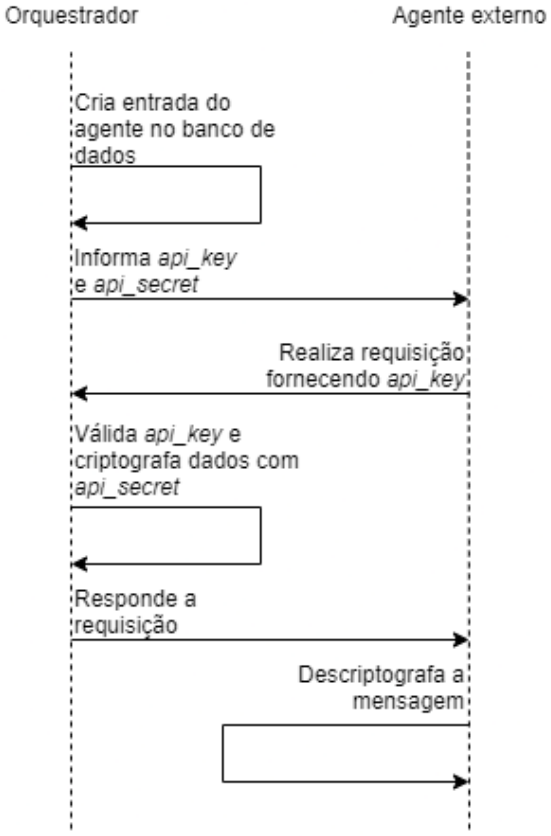


Figura 3.23 – Fluxograma de autenticação e troca de mensagens de um agente externo com o orquestrador.

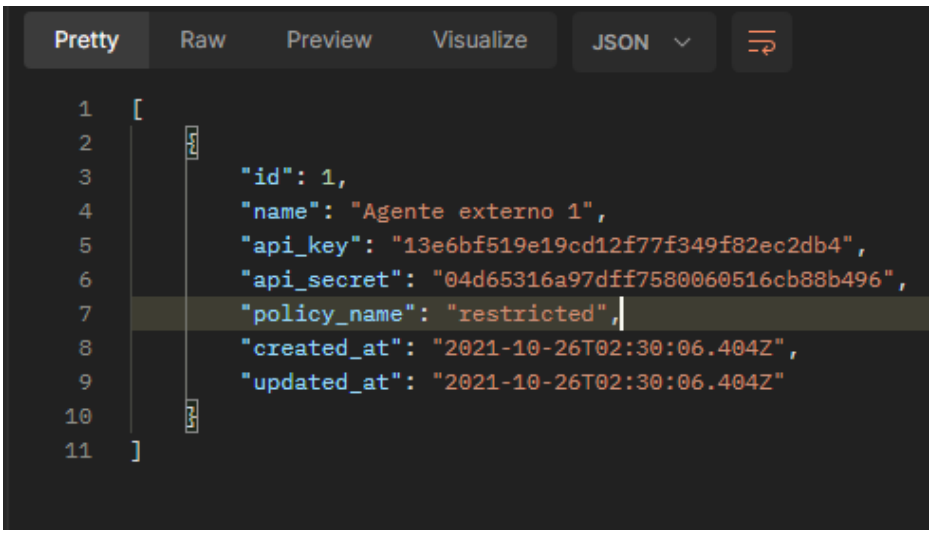


Figura 3.24 – Exemplo de mensagem para o agente externo no momento de cadastro no orquestrador.

Com as credencias o agente externo requisita as regras existentes no orquestrador, baseado nas permissões que o mesmo possui, assim o agente externo pode compartilhar as regras dentro da sua própria rede e conhecer o estado da rede em que está ingressando. Em seguida o servidor requisita credenciais do canal no Tangle. O orquestrador compartilha as informações com os dispositivos que farão parte do canal e com o agente externo. Nesse momento a política do agente externo é passada aos dispositivos pertencentes ao canal, ela funciona como um filtro sobre mudanças da política de segurança do dispositivo.

O agente externo cria um canal no Tangle e pode começar a compartilhar regras. O orquestrador também ingressa no canal para conseguir ter acesso as mensagens produzidas pelo agente externo. Esse fluxo completo é evidenciado na Figura 3.25.

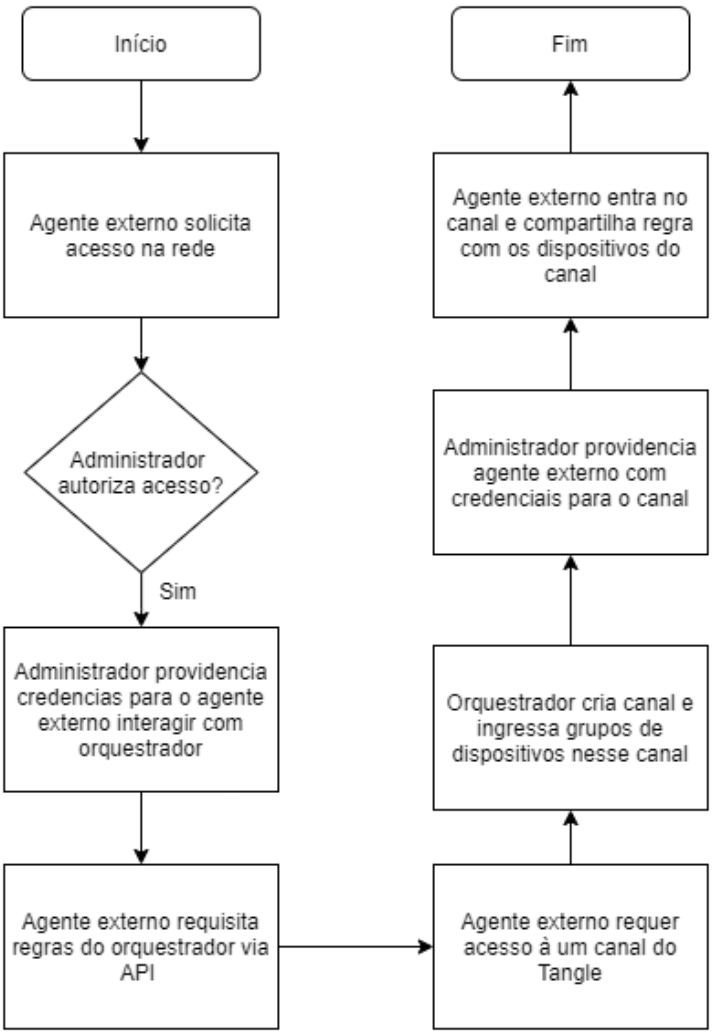


Figura 3.25 – Fluxograma de provisionamento de credencias e acesso do agente externo à uma nova rede pelo orquestrador.

Com isso temos a estrutura de rede entre orquestrador, servidor externo e dispositivos. Importante ressaltar que toda a comunicação entre agente externo e dispositivos é realizada através de um canal no Tangle, garantindo auditoria, privacidade e autorização para toda a comunicação.

3.2.5 Tangle

Para o canal de comunicação entre servidor externo e dispositivos foi escolhido o Tangle [35]. Uma solução semelhante a blockchain específica para dispositivos IoT. As principais vantagens de se utilizar o Tangle são:

- não existência de taxas na rede;
- escalabilidade da rede, incluindo um aumento significativo na segurança da rede à medida que escala [29];
- possibilidade de criar um canal criptografado privado.

Para a solução proposta utilizamos o Iota Streams, que possibilita a criação um canal de comunicação criptografado com um autor e vários assinantes [20].

Na arquitetura proposta o autor é representado pelo agente externo e tanto os dispositivos quanto o orquestrador são assinantes do canal. No Iota Streams o autor pode enviar mensagens criptografadas em *broadcast* para todos os assinantes, os assinantes também podem publicar mensagens no canal porém não criptografadas.

O Iota Streams também permite controle de acesso dentro de um canal, onde é possível aplicar um mecanismo criptográfico diferente a cada mensagem com base no tipo da mensagem [20].

Na arquitetura proposta apenas o autor precisa publicar mensagens para a rede e todas elas são criptografadas. Como o Iota Streams é executado sobre o Tangle, temos auditoria dos dados trocados entre servidor externo e dispositivos.

O canal no Iota Streams pode ser de ramificação única ou de múltiplas ramificações. Na ramificação única cada vez que um autor posta uma mensagem na cadeia, todos os participantes incrementam o estado da sequência uniformemente. Isso permite a fácil transmissão de dados entre dispositivos conectados diretamente, onde todos os dados postados são relevantes para todas as partes envolvidas. Ao procurar a próxima mensagem, um identificador de mensagem pode ser gerado para cada chave pública usando o próximo estado de sequência antecipado. Se a mensagem for encontrada e verificada, o estado da sequência é atualizado e a próxima mensagem é pesquisada. Na Figura 3.26 é apresentado um diagrama ilustrando esse processo.

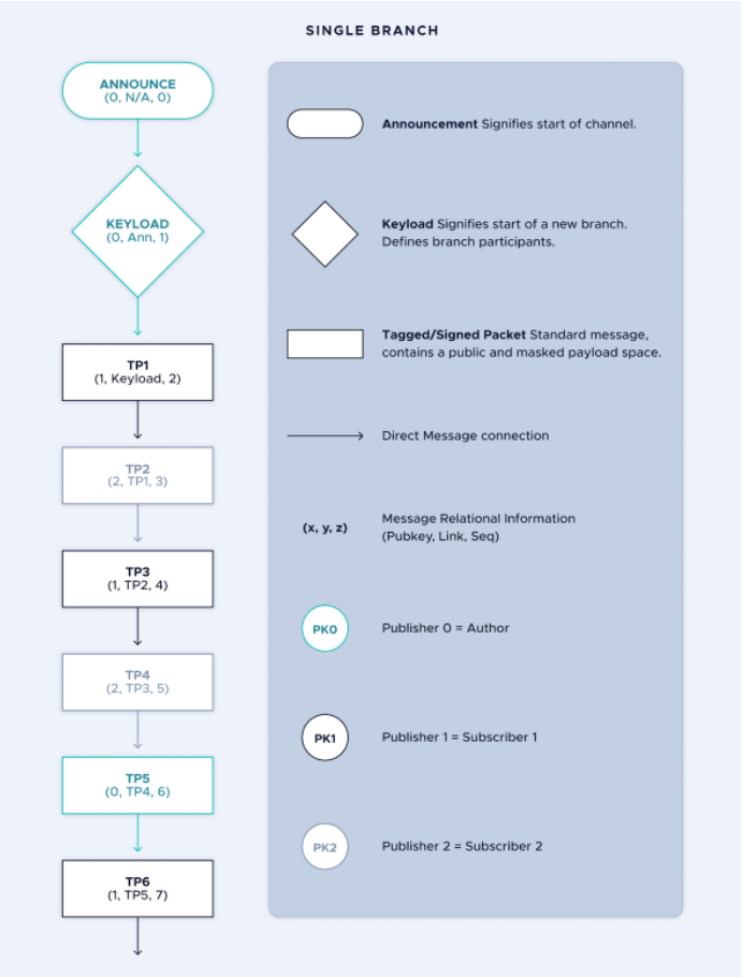


Figura 3.26 – Diagrama de ramificação única de um canal no Iota Streams. Fonte [21]

Já o canal de múltiplas ramificações utiliza uma ramificação como ponto de referência, chamada ramificação de sequenciamento, para mapear autores individuais para uma ou mais ramificações de mensagens encontradas no canal. Isso permite que autores e assinantes usem a ramificação de sequenciamento como uma referência para navegar nas ramificações de mensagens para encontrar facilmente as mensagens de um editor específico. A Figura 3.27 apresenta um diagrama ilustrando o processo.

O autor do canal é responsável pela geração de um novo canal junto com a configuração da estrutura pretendida daquele canal. O autor possui a capacidade de definir as restrições de acesso a ramos dentro de uma estrutura de canal, bem como aceitar e gerenciar mensagens de inscrição do usuário.

Um assinante é qualquer usuário em um canal que não seja o autor. Um assinante pode ser gerado de forma independente sem verificação por um autor, mas para escrever em uma ramificação, eles serão obrigados a se inscrever no canal.

Na solução proposta é utilizado um canal de ramificação única. O agente externo tem a função de autor no canal, enviando mensagens criptografadas para todos os assinantes. Os assinantes

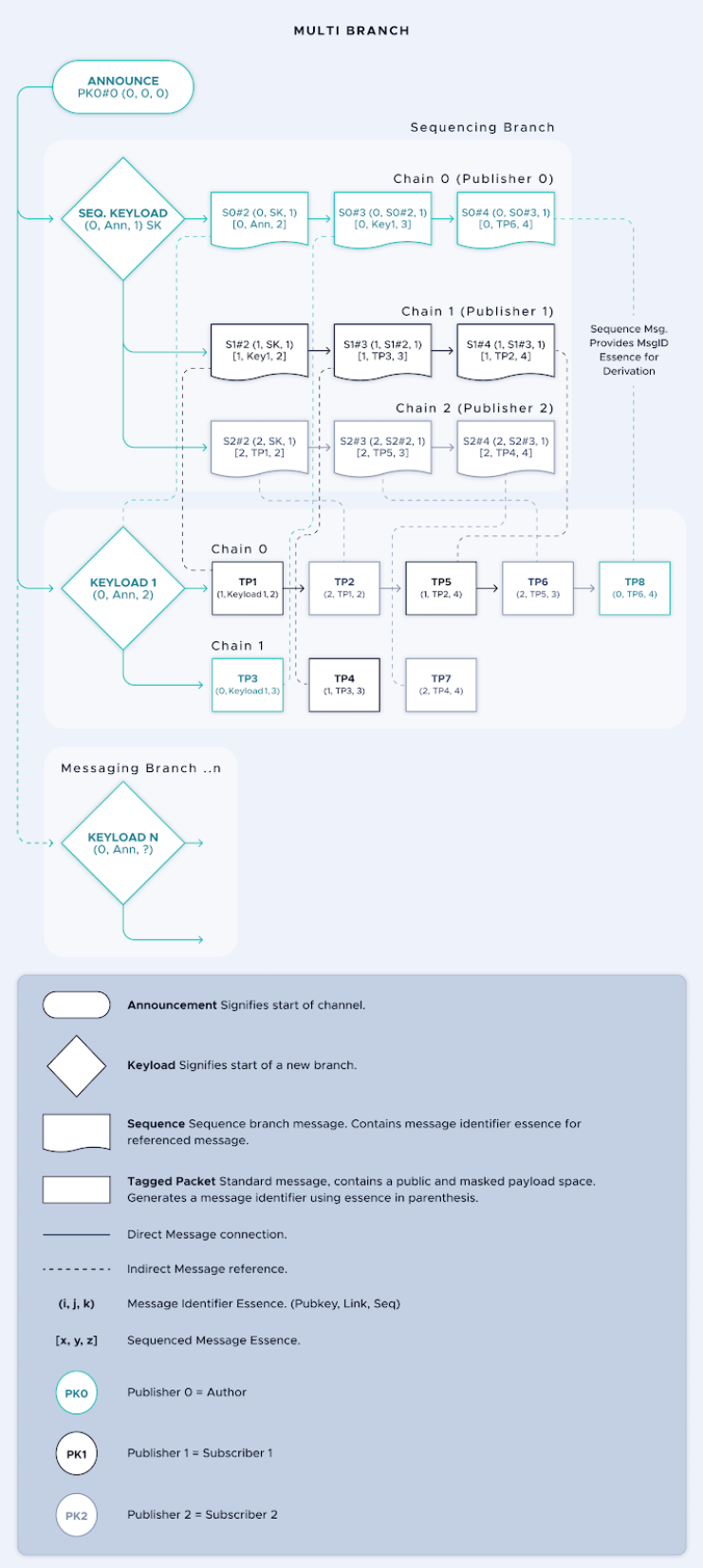


Figura 3.27 – Diagrama de múltiplas ramificações de um canal no Iota Streams. Fonte [21]

são os dispositivos IoT e o orquestrador. Os assinantes não publicam mensagens na rede, apenas escutam as mensagens oriundas do agente externo, assim os assinantes não precisam da verificação do autor do canal.

As mensagens publicados pelo controlador externo podem ser de dois tipos: publicação de novas regras ou atualização da política de segurança dos dispositivos. É importante relembrar que para alterar a política de segurança dos dispositivos é necessário que o controlador externo tenha as permissões necessárias.

A Figura 3.28 apresenta um exemplo do uso do canal no arquitetura proposta. Primeiro o autor cria o canal e anuncia a existência desse canal para os dispositivos, fornecendo um link de acesso ao canal. Quando os dispositivos recebem o anúncio do canal eles realizam a assinatura ao canal para receber mensagens. Em seguida o autor envia o *keyload* aos dispositivos que fazem parte do canal.

O keyload é uma mensagem de controle de acesso que permite ao autor especificar quem deve ser capaz de descriptografar quaisquer mensagens anexadas após ela. O keyload pode ser especificado de duas formas. Através de chaves públicas de cada assinante do canal, assim o autor armazena as chaves e através dela verifica os nós que acessarão a mensagem. A segunda possibilidade é através de chaves pré compartilhadas (PSK), uma chave que é compartilhada antes da criação do canal com todos os nós que devem ser capazes de ler o conteúdo das mensagens. Na solução proposta é utilizado chaves pré-compartilhadas. As chaves são criadas e distribuídas para os nós pelo orquestrador no momento em que o agente externo é autorizado a compartilhar mensagens com os dispositivos.

A Figura 3.29 apresenta o fluxograma completo de compartilhamento de regras pelo servidor externo, desde o fornecimento de credenciais para o agente externo até o recebimento de mensagens pelos dispositivos.

Figura 3.28 – Exemplo de processo de criação do canal e compartilhamento de mensagens criptografadas através do Tangle.

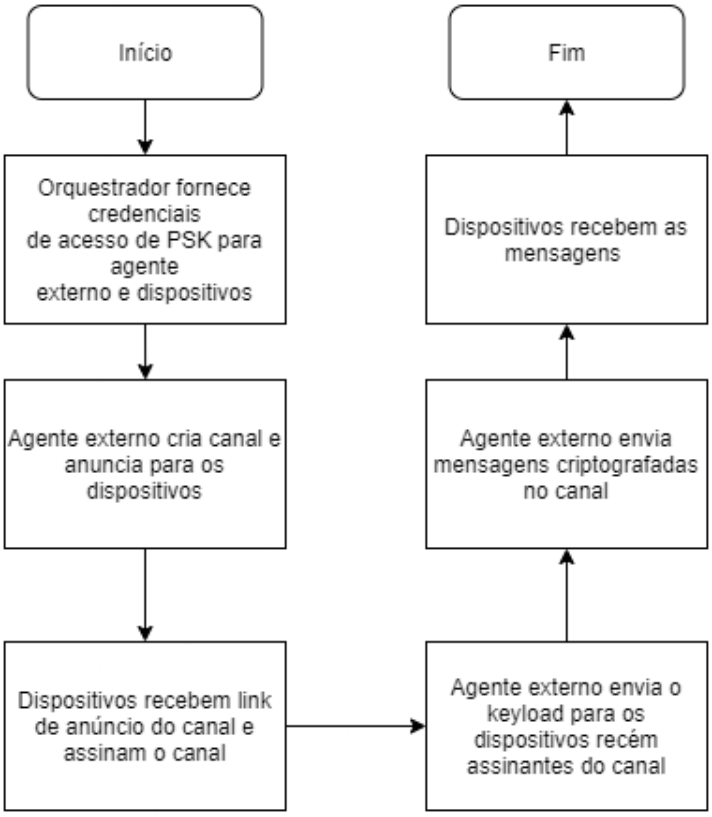


Figura 3.29 – Fluxo de criação e configuração do canal Tangle para envio de regras pelo agente externo

Com isso a solução provê um canal de comunicação criptografado entre agente externo e gateways IoT. A utilização desse canal permite que servidores externos compartilhem regras para o cliente HIDPS com interferência mínima do orquestrador.

4 Resultados e Análise

Este capítulo realiza uma análise da solução proposta e está divido em duas sub-seções. A primeira sub-seção tratará da validação do sistema como um todo. Uma regra será criada no orquestrador e compartilhada com os dispositivos que buscará indícios de infecção por uma botnet. O objetivo dessa análise é observar toda a arquitetura funcionando em conjunto, desde o provisionamento dos dispositivos com o cliente HIDPS, o gerenciamento de dispositivos e regras pelo orquestrador, o envio de regras tanto pelo orquestrador quanto pelo agente externo e a execução das regras pelo cliente HIDPS.

A segunda sub-seção tratará de analisar o tempo médio para que regras publicadas sejam consumidas pelos dispositivos, tanto pelo orquestrador quanto por agentes externos.

As simulações foram realizadas utilizando *containers*. No Anexo D temos a configuração do container para simular o cliente HIDPS. No Anexo B temos o arquivo de configuração dos agentes do orquestrador em containers. Podemos ver nesse arquivo a configuração da rede *hids_iot_network*, todos os containers declarados nesse arquivo já se encontram na mesma rede e os clientes HIDPS são adicionados a esse rede também, para permitir comunicação com o orquestrador.

Na Figura 4.1 é possível observar todos os containers utilizados nas simulações. Os containers estão nomeados de acordo com o nome do serviço que representam. Já para o Tangle, foi utilizado uma rede local simulada. Tanto os dispositivos quanto os agentes externos se conectam a essa rede para acessar o canal.

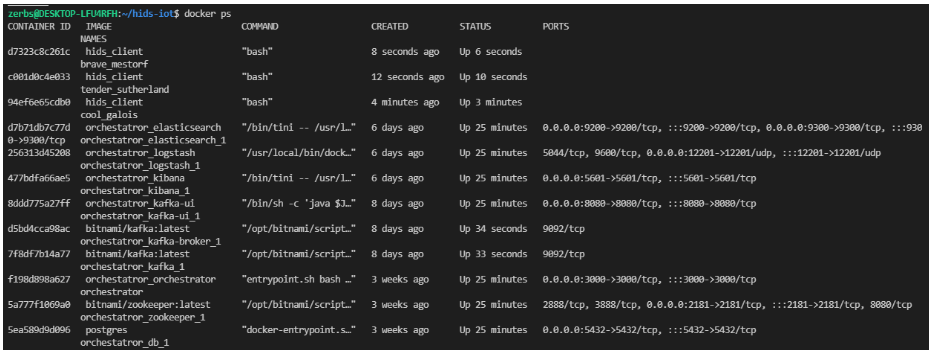


Figura 4.1 – Lista de container utilizados nas simulações. Os containers estão nomeadoss de acordo com o nome do serviço que representam.

4.1 Validação da solução proposta

Para realizar a validação da solução proposta será criado regras na rede para detectar se os dispositivos estão infectados pela botnet Mirai [13]. Todo o processo será demonstrado em seguida, desde o provisionamento dos dispositivos até a execução da regra.

Primeiramente o administrador precisa criar regras para a rede. Para criar uma regra, primeiro

é necessário cadastrar os scripts que serão utilizados pela regra. Na Figura 4.2 temos o cadastro do script para checar se a porta 48101 está sendo utilizada, dispositivos infectados pelo Mirai utilizam essa porta para reportar informações encontradas sobre novos dispositivos na rede [13]. O código para o script pode ser visto na Figura 4.3.

Em seguida é cadastrado o script para interromper o processo sendo executado na porta utilizada pelo Mirai. Encerrar o processo na porta 48101 apenas impede que o dispositivo reporte à controladora do Mirai sobre novos dispositivos. Para que o dispositivo seja desinfectado é necessário que ele seja reinicializado [13]. É importante ressaltar que o dispositivo pode ser reinfectado novamente após a reinicialização se a vulnerabilidade que permitiu a infecção originalmente não for resolvida. Então o script utilizado nessa análise é composto por três etapas:

1. encerrar o processo que está sendo executado na porta 48101;
2. alterar a senha do dispositivo, a nova senha é composta por um hash do ip do dispositivo, do seu endereço MAC e de uma entrada aleatória com dez caracteres;
3. reinicializar o dispositivo;

Esse script pode ser visto na Figura 4.4 e na Figura 4.5 o script é cadastrado no orquestrador. É importante observar que o reinicialização do dispositivo é realizada apenas dez minutos depois da execução da ação. Esse tempo é necessário para que o dispositivo execute novamente o teste de caso para verificar se a vulnerabilidade foi mitigada.

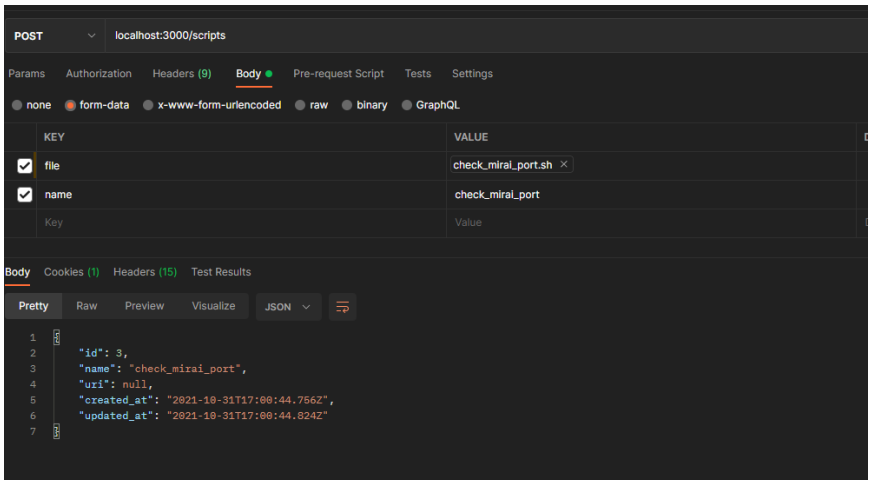


Figura 4.2 – Criação do script para verificar se a porta utilizada pelo Mirai está em uso.

check_mirai_port.sh U hids_client/scripts/test_cases/check_mirai_port.sh

```
#!/bin/sh

# Check if Mirai controller port is being used
if lsof -nP -iTCP:48101 -sTCP:LISTEN | grep '48101'; then
  echo '{"message": "Mirai controller port found", "status": "true"}'
else
  echo '{"message": "Mirai controller port not found", "status": "false"}'
  exit 1
fi
```

Figura 4.3 – Script para verificar se a porta 48101 está sendo utilizada pelo dispositivo.

kill_mirai_port.sh U hids_client/scripts/actions/kill_mirai_port.sh

```
#!/bin/sh

# Kill mirai port
kill -9 $(lsof -t -i tcp:48101)

# Change password
MAC=`ip a | grep link/ether | awk -F " " '{print $2}'`
IP=`hostname -i`
SALT=`cat /dev/urandom | tr -dc 'a-zA-Z0-9' | fold -w 10 | head -n 1`
PWD=`echo -n ${MAC}:${IP}:${SALT} | md5sum`


echo `pi:${PWD}` | chpasswd

# Reboot device
reboot now

```

Figura 4.4 – Script para encerrar processo na porta 48101, mudar a senha e reinicializar o dispositivo

kill_mirai_port.sh U hids_client/scripts/actions/kill_mirai_port.sh

```
#!/bin/sh

# Kill mirai port
kill -9 $(lsof -t -i tcp:48101)

# Change password
MAC=`ip a | grep link/ether | awk -F " " '{print $2}'`
IP=`hostname -i`
SALT=`cat /dev/urandom | tr -dc 'a-zA-Z0-9' | fold -w 10 | head -n 1`
PWD=`echo -n ${MAC}:${IP}:${SALT} | md5sum`


echo `pi:${PWD}` | chpasswd

# Reboot device after 10 minutes, gives time for the test_case to be reran
shutdown -r +10


```

Figura 4.5 – Criação do script para encerrar processo na porta utilizada pelo Mirai.

Com os scripts cadastrados no orquestrador, o próximo passo é associa-los à ações ou testes de caso. Nas Figuras 4.6 e 4.7 é apresentado a associação do script responsável por checar se a porta utilizada pelo Mirai à um teste de caso e o script para terminar o processo nessa porta à uma ação, respectivamente.

Com ação e teste de caso criados, o administrador consegue então criar uma regra, definindo uma ação e um teste de caso para a mesma. Na Figura 4.8 é apresentado a criação da regra com

a ação e teste caso cadastrados anteriormente.

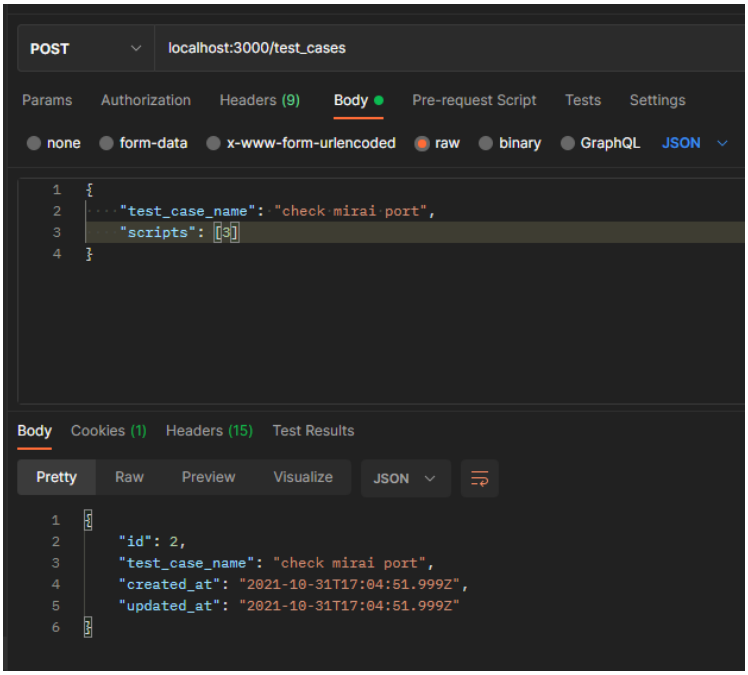


Figura 4.6 – Criação do teste de caso para checar se a porta utilizada pela controladora do Mirai está sendo utilizada.

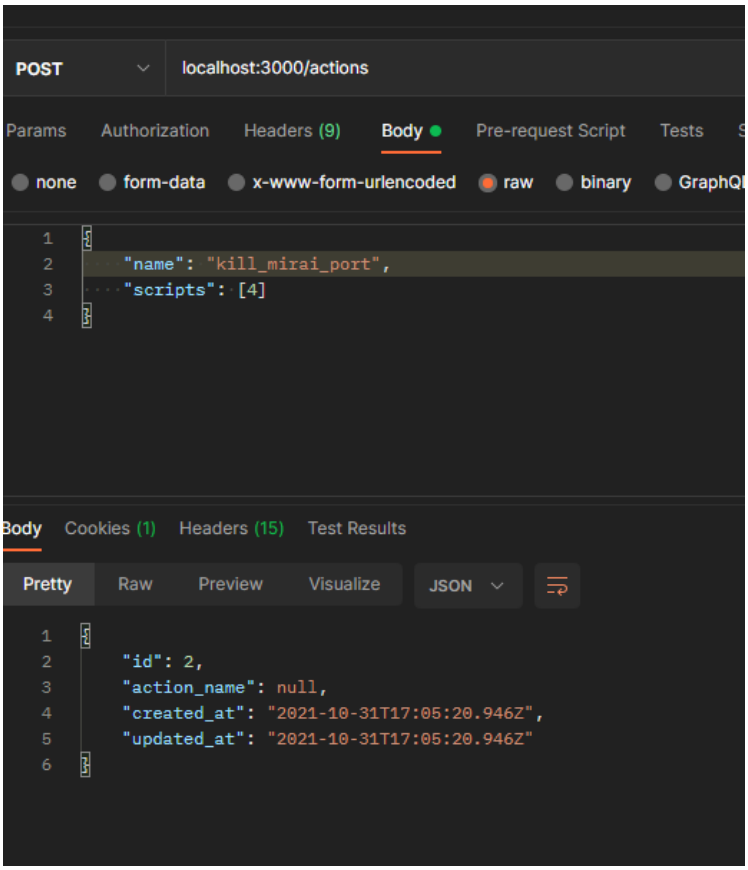


Figura 4.7 – Criação ação para encerrar processo na porta utilizada pela controladora do Mirai.

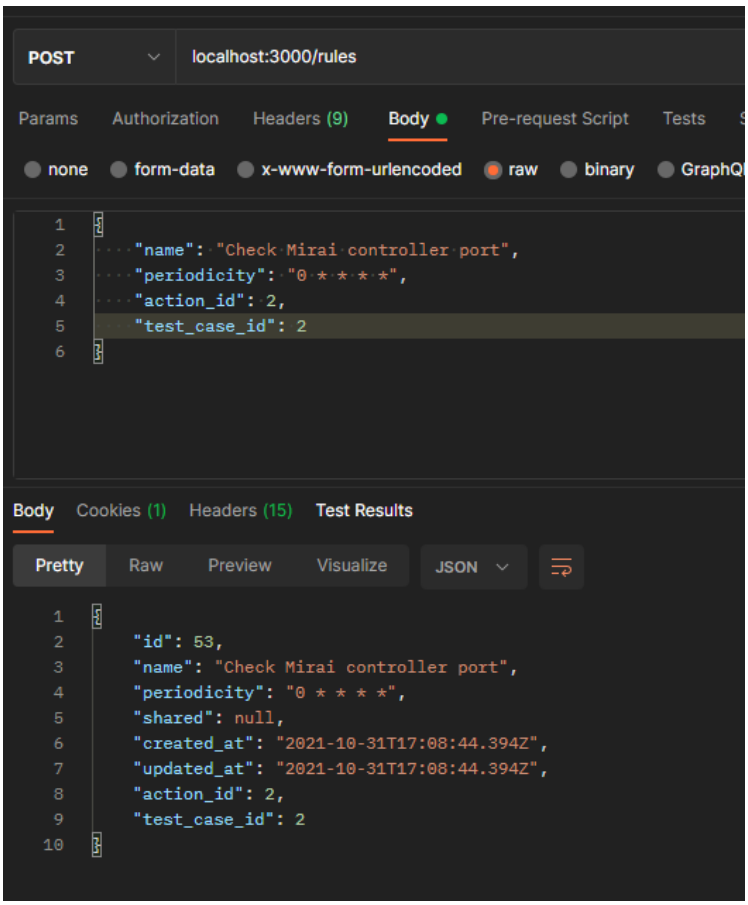


Figura 4.8 – Criação da regra para verificar se dispositivo está infectado pelo Mirai

O próximo passo realizado pelo administrador é a criação de um grupo de dispositivos que receberão as regras e que compartilham a mesma política de segurança. Primeiramente o administrador cria um novo grupo informando quais os scripts poderão ser executados pelos dispositivos daquele grupo e também informa um nome para o grupo, como pode ser visto na Figura 4.9.

Após a criação do grupo, o administrador registra dispositivos ao grupo, informando o grupo e o IP do dispositivo, como pode ser observado na Figura 4.10. Após o registro dos dispositivos, os mesmos são provisionados com o cliente HIDPS, a política de segurança e os scripts que a política de segurança permite através do agente de provisionamento.

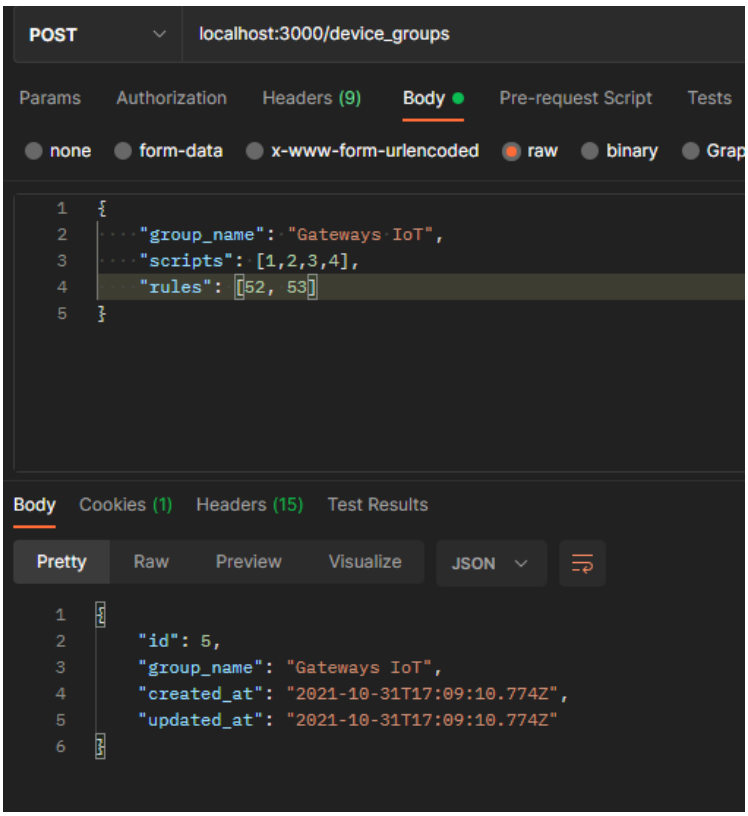


Figura 4.9 – Criação de um grupo de dispositivos fornecendo uma política de segurança e um nome.

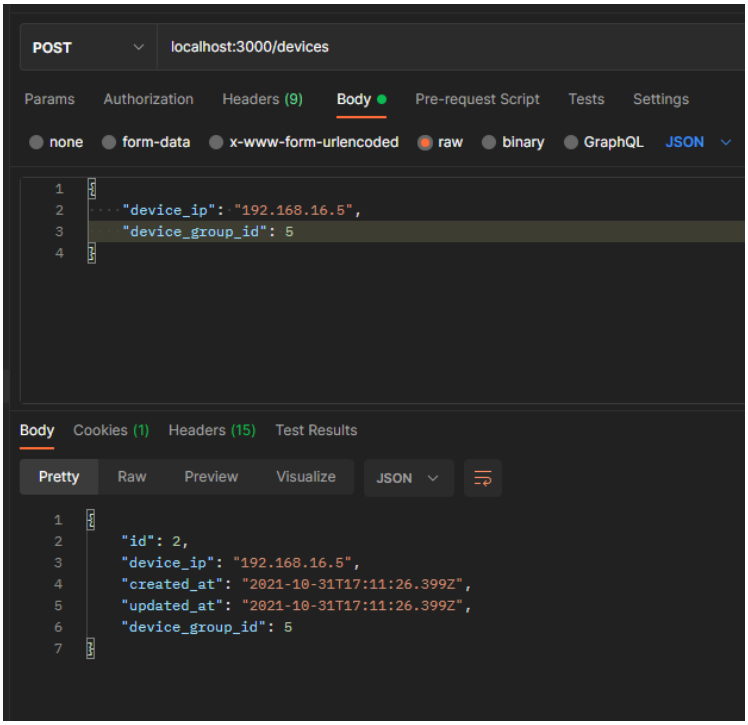


Figura 4.10 – Registro de um dispositivo à um grupo de dispositivos fornecendo o identificador do grupo e o IP do dispositivo.

Os dispositivos que já foram provisionados com o cliente HIDPS ficam aguardam novas regras

no tópico de comunicação utilizado pelo seu grupo. O administrador publica as regras para um grupo em qualquer momento. Assim que as regras são publicadas para o agente de comunicação os dispositivos tem acesso as mesmas.

Na Figura 4.11 temos a publicação da regra criada anteriormente para detecção do Mirai pelo orquestrador. Podemos observar que a regra é publicada no tópico *rules_group_5*, ou seja, os dispositivos que pertencem ao grupo cinco terão acesso a essa regra. Na Figura 4.12 é possível observar um dispositivo pertencente ao grupo cinco recebendo a regra enviada pelo orquestrador.

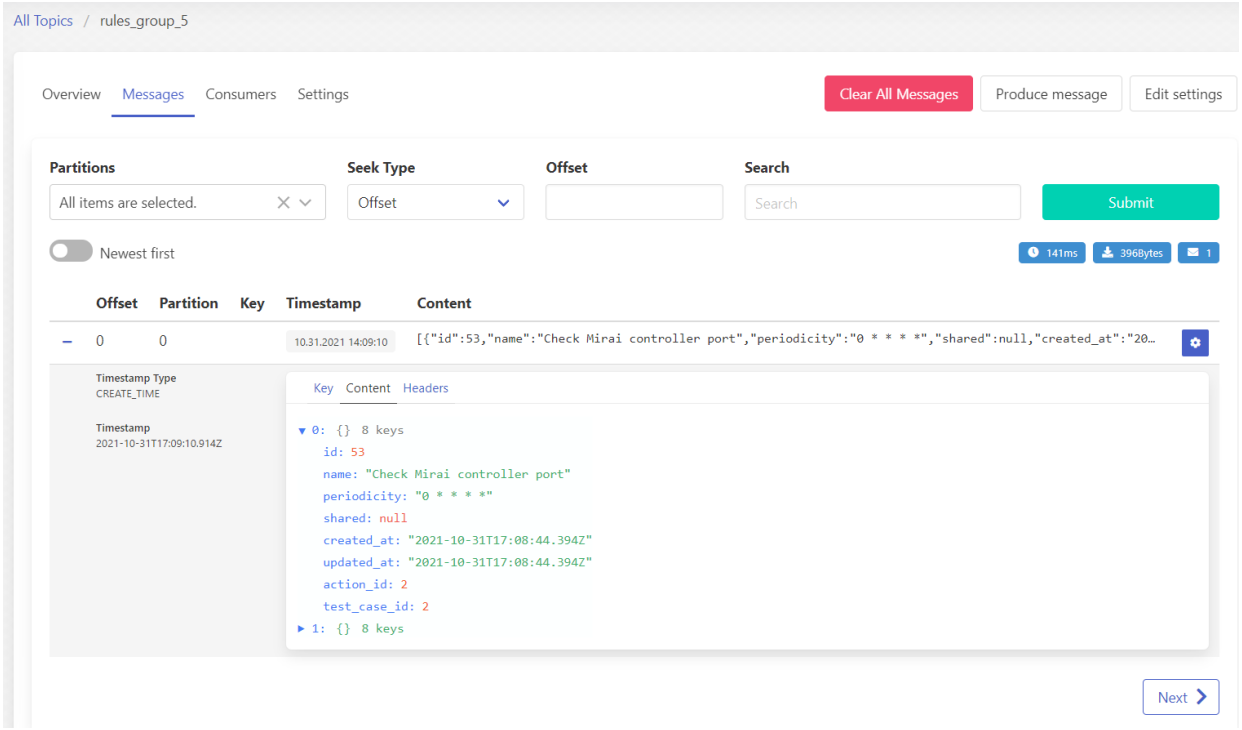


Figura 4.11 – Regra para detecção do Mirai enviada pelo orquestrador para o agente de comunicação.

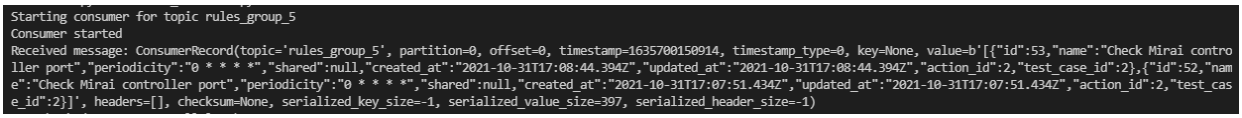


Figura 4.12 – Dispositivo recebe regra para detecção do Mirai enviada pelo orquestrador.

É importante observar que a regra para detecção do Mirai é executada de forma periódica, assim ela é configurada como uma tarefa no CRON assim que recebida. Na Figura 4.13 é possível oberservar a *crontab*, um programa do Unix que edita o arquivo onde são especificados os comandos a serem executados e a hora e dia de execução pelo cron, um serviço que executa comandos agendados. A execução de regra de detecção do Mirai acontece a cada hora.

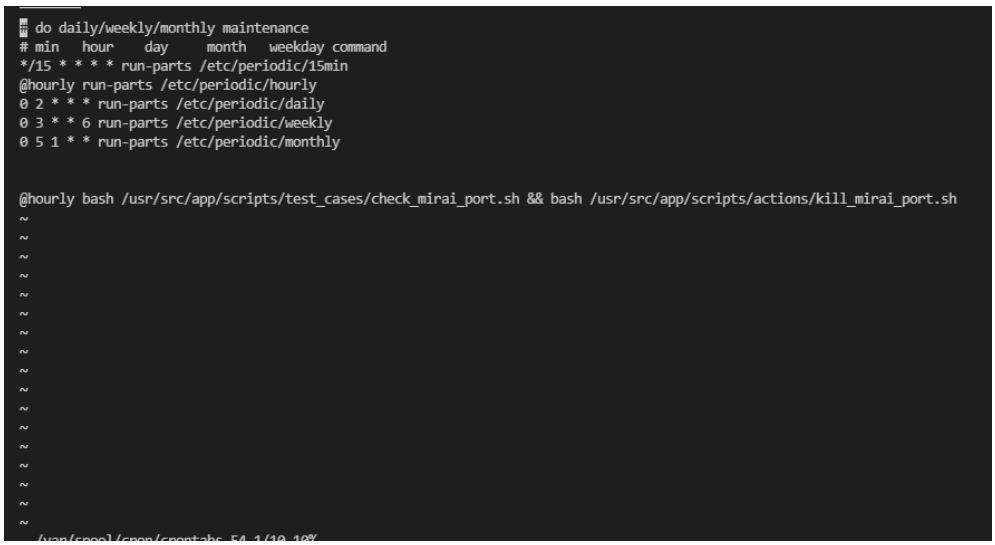


Figura 4.13 – Crontab evidenciando regra de detecção do Mirai configurada para ser executada a cada hora.

A regra de detecção do Mirai foi executada em dois dispositivos do mesmo grupo. Em um dos dispositivos foi simulado um processo sendo executado na porta 48101, para que o teste de caso encontrasse a vulnerabilidade e a ação fosse executada. No segundo dispositivo a vulnerabilidade não foi simulada, logo apenas o teste de caso é executado.

Na Figura 4.14 temos o relatório gerado para o dispositivo em que a vulnerabilidade foi simulada. Na primeira mensagem do relatório é possível observar o momento em que a regra começa a ser executada bem como qual regra está sendo executada e por qual dispositivo. Nesse caso, o dispositivo com identificador três está executando a regra de detecção do Mira. Na segunda mensagem o dispositivo informa que o teste caso foi executado e a vulnerabilidade foi encontrada, assim a ação é executada em seguida. Já na terceira mensagem, o dispositivo informa que ação foi executada com sucesso e que a vulnerabilidade foi mitigada. Para verificar que a vulnerabilidade não existe mais, o dispositivo executa o teste de caso novamente após a finalização da ação.

Já na Figura 4.15 temos o relatório de execução da regra pelo dispositivo com identificador dois, onde a vulnerabilidade não foi simulada. Nesse caso, a vulnerabilidade não foi encontrada, o que é evidenciado pela segunda mensagem que informa que o dispositivo não encontrou a vulnerabilidade especificada no teste de caso, e portanto, a ação não será executada.

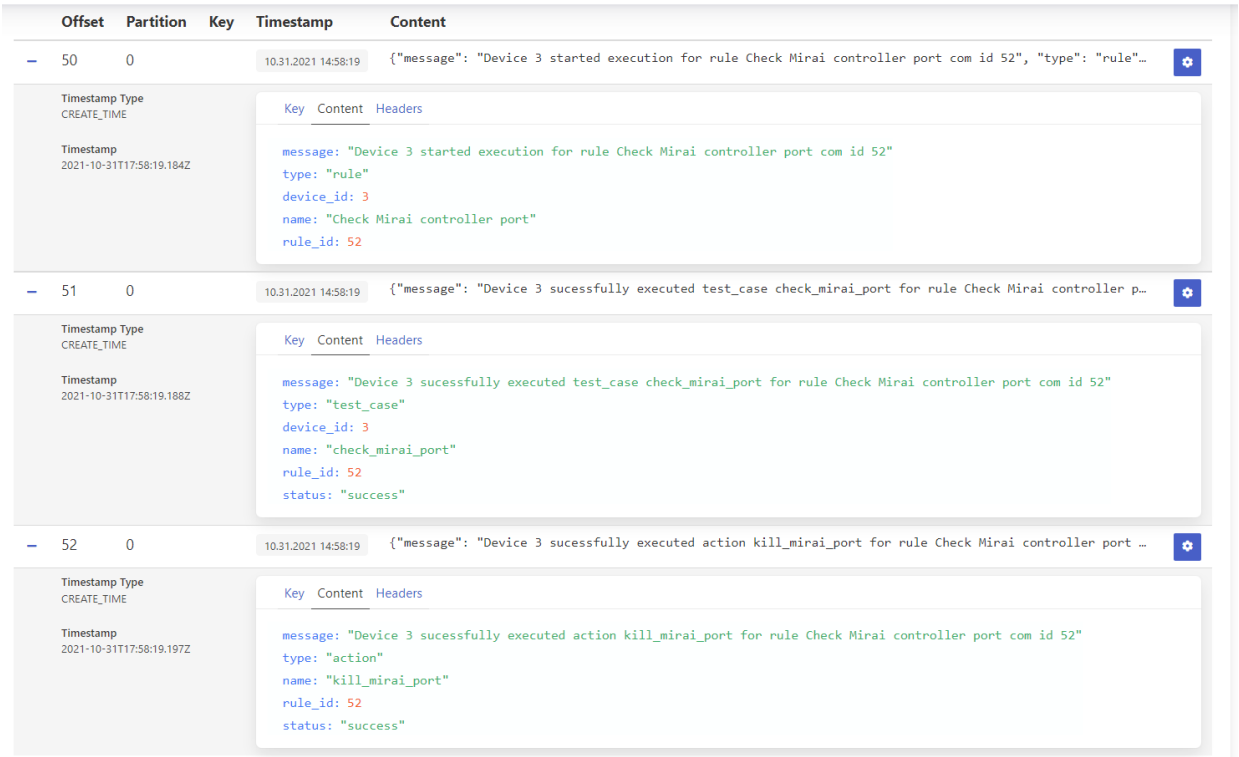


Figura 4.14 – Relatório de execução da regra de detecção do Mirai onde vulnerabilidade foi encontrada e mitigada.

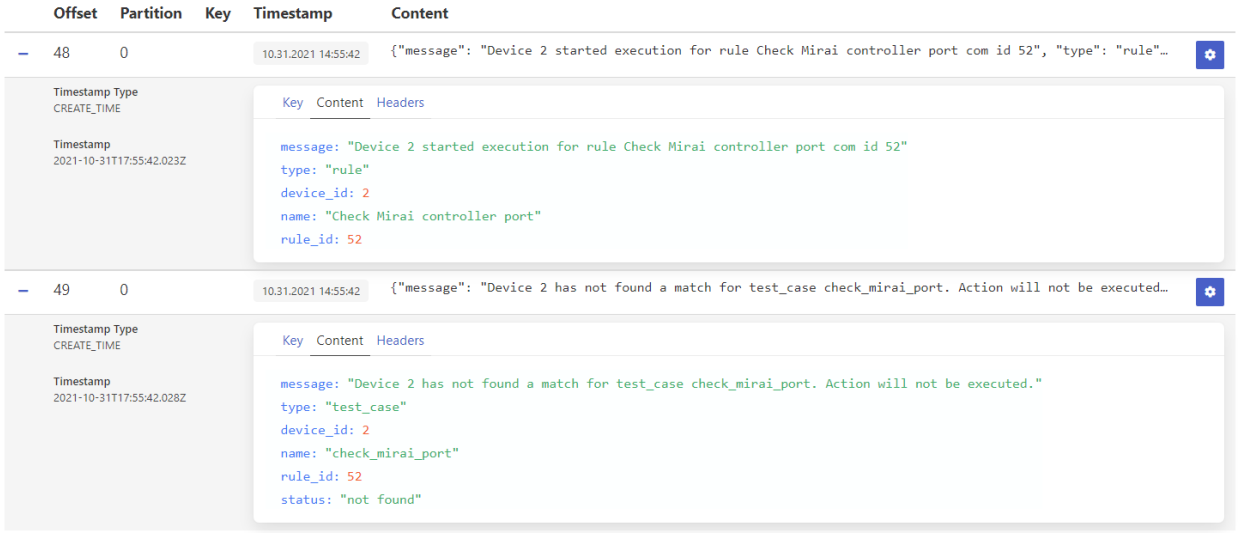


Figura 4.15 – Relatório de execução da regra de detecção do Mirai onde vulnerabilidade não foi encontrada.

Como último aspecto a se analisar da arquitetura temos a interação do agente externo com os dispositivos através do Tangle. Primeiramente um agente externo foi autorizado pelo administrador e recebeu as credencias necessárias do orquestrador. Na Figura 4.16 é possível observar as credenciais fornecidas. A chave de API, *api_key* e o segredo da API, *api_secret* são utilizados para a comunicação entre servidor externo e orquestrado, através da API do orquestrador. Já a

chave pré-compartilhada, *psk*, é utilizada para criptografar as mensagens que serão compartilhadas com os dispositivos.

No momento em que o orquestrador provê as credenciais para o agente externo ele aciona o agente de provisionamento para provisionar os dispositivos que receberão regras do agente externo com as credencias necessárias também. Os dispositivos são provisionados com o código para se conectar ao canal do Tangle e também recebem a chave pré-compartilhada para que possam descriptografar as mensagens enviadas pelo agente externo.

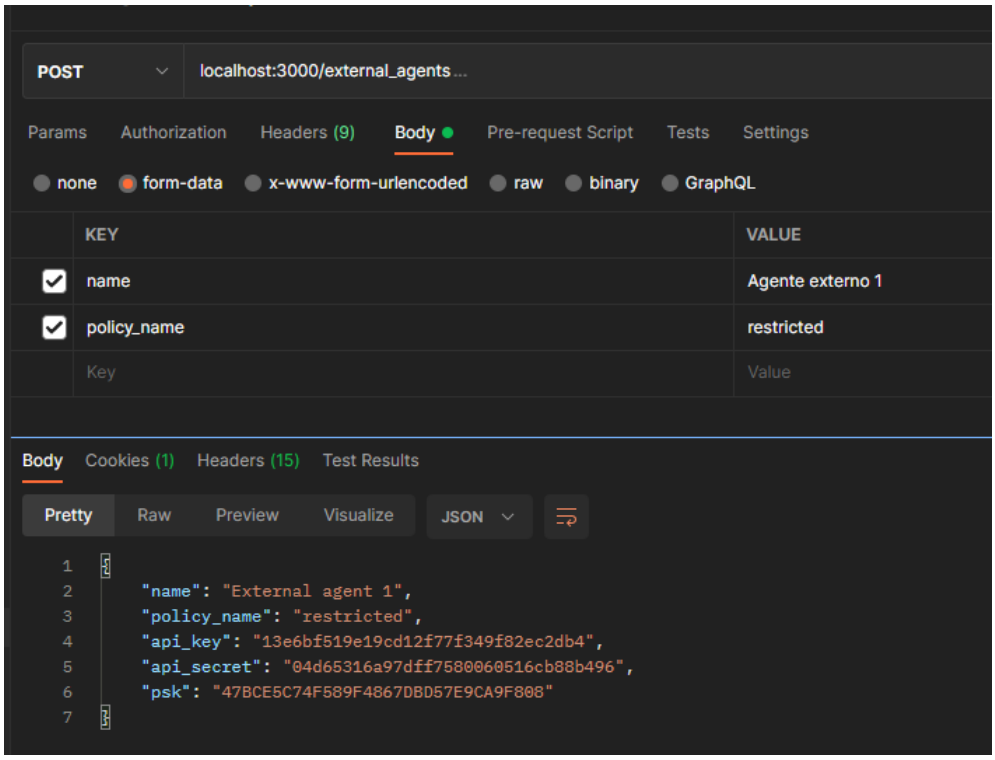


Figura 4.16 – Credencias fornecidas ao agente externo pelo orquestrador.

Assim que o agente externo recebe as credencias do orquestrador, o canal no Tangle é criado e regras podem ser compartilhadas com o dispositivos. A regra compartilhada pelo agente externo nessa análise também é para detecção do Mirai nos dispositivos. A regra é compartilhada com um grupo diferente de dispositivos dos utilizados anteriormente.

A configuração da regra é diferente da utilizada anteriormente, a verificação de existência de um processo na porta 48101 ainda é realizada, porém, a regra fornecida pelo agente externo também verifica se o Telnet está habilitado no dispositivo. O Telnet é utilizado pelo Mirai para acessar e infectar dispositivos, geralmente utilizando ataques de dicionário com usuários e senhas padrões [13]. Na Figura 4.17 é apresentado a configuração da regra gerada pelo agente externo.

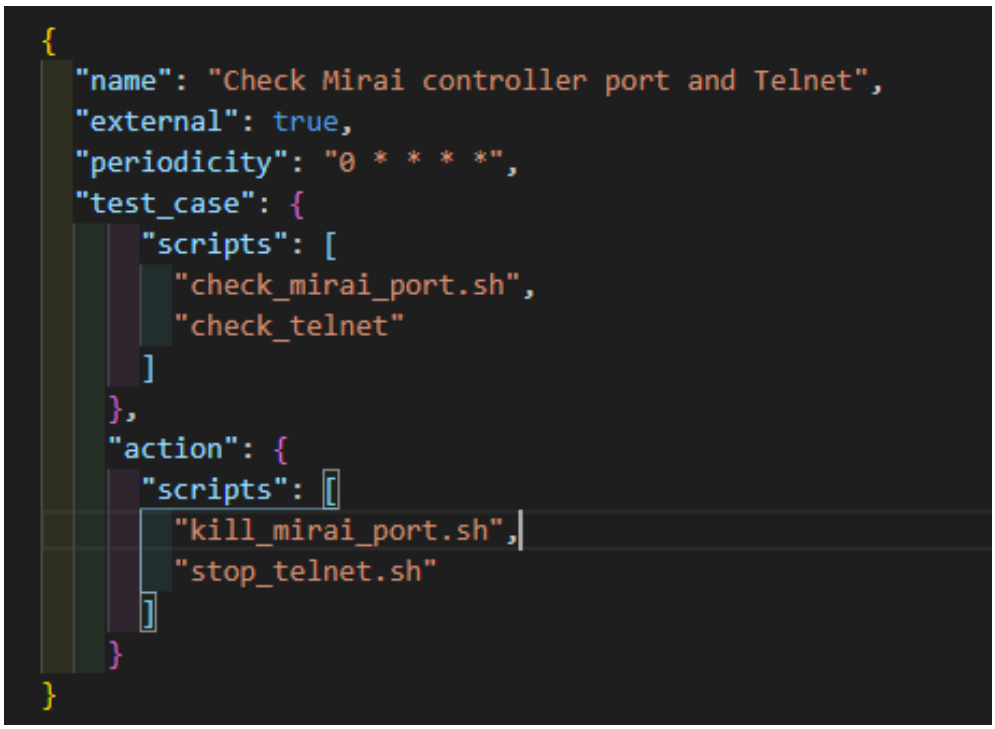


Figura 4.17 – Regra para detecção do Mirai fornecida pelo agente externo.

A regra fornecida pelo agente externo também é periódica, então assim que é recebida pelos dispositivos ela é configurada como uma tarefa no CRON, como visto na Figura 4.18. A regra é executada da forma tradicional, primeiro os testes de casos são executados e caso eles encontrem as vulnerabilidades, as ações são executadas.

Para regras oriundas de agentes externos, o relatório apresenta qual o canal em que a regra foi recebida. Na Figura 4.19 é apresentado o relatório de execução do teste de caso da regra pelo dispositivo com identificador cinco. É possível observar que o identificador do canal é fornecido no relatório, o que facilita o trabalho do orquestrador para identificar a regra, uma vez que o mesmo é um assinante do canal.

Na Figura 4.20 é apresentado o relatório para a ação executada a partir da regra fornecida pelo agente externa. Os relatórios das regras providas pelo orquestrador e por agentes externo são semelhantes, tendo como única diferença a identificação da regra.Para as regras oriundas do orquestrador a identificação é realizada através do identificador da regra no banco de dados do orquestrador. Já para as regras compartilhadas por agentes externos, a identificação é realizada apenas pelo canal.

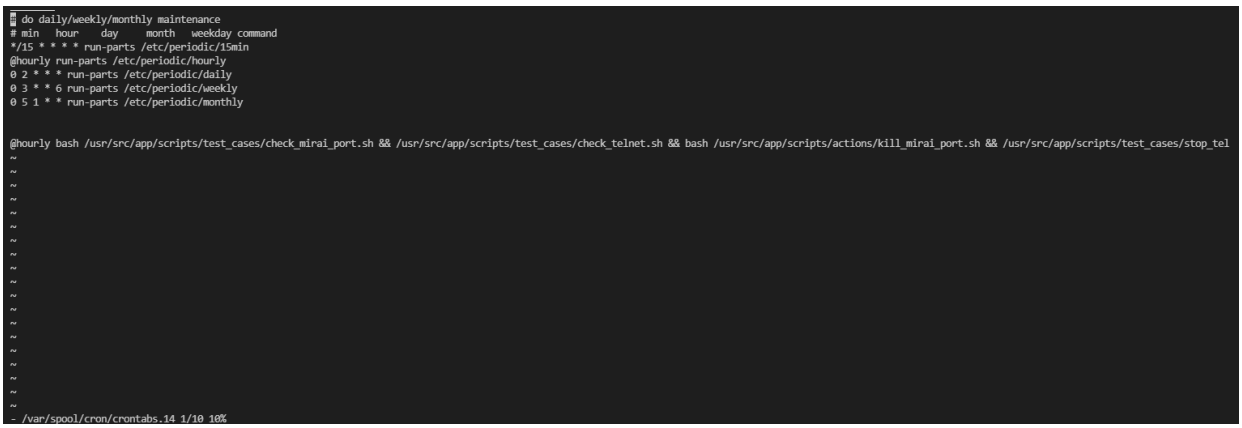


Figura 4.18 – Crontab evidenciando regra fornecida pelo agente externo com execução a cada hora.

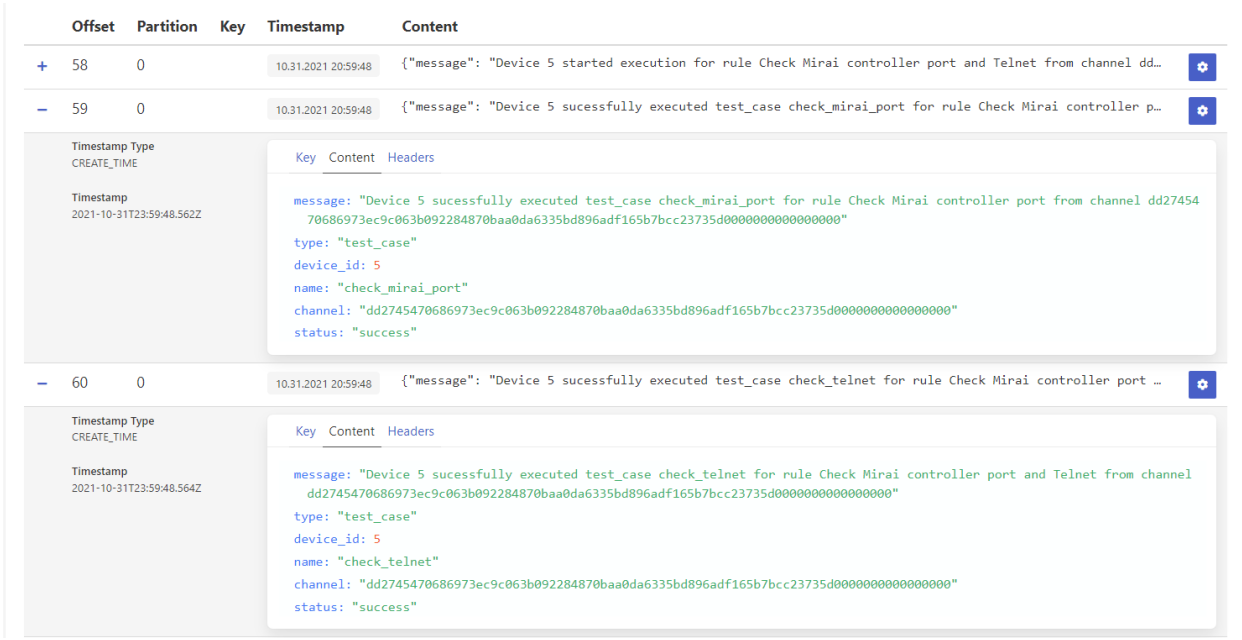


Figura 4.19 – Relatório de execução do teste de caso da regra para detecção do Mirai fornecida pelo agente externo

Figura 4.20 – Relatório de execução da ação da regra para detecção do Mirai fornecida pelo agente externo

Com isso temos a validação da arquitetura proposta nesse trabalho. Desde o provisionamento do dispositivos, o gerenciamento de regras e dispositivos pelo orquestrador, a comunicação entre orquestrador e dispositivos, a execução de regras pelo cliente HIDPS e o compartilhamento de regras de diferentes redes através do agente externo.

4.2 Tempo de atualização de regras

Uma métrica importante em sistemas de detecção e prevenção de intrusão baseados em assinatura é o tempo de atualização de regras. Em seguida, avaliamos o tempo para que os gateways IoT recebam novas regras. A avaliação é realizada com as regras oriundas do orquestrador, através do Apache Kafka e de agentes externos, através do Tangle.

A simulação foi realizada cinco vezes variando o número de regras providas. O tempo para a regra enviada pelo orquestrador ser recebida pelos dispositivos foi mensurado e é apresentado nas Tabelas 4.1 e 4.2 para a comunicação através Kafka e do Tangle, respectivamente. E as Tabelas 4.3 e 4.4 apresentam o valor médio das métricas apresentadas anteriormente.

Tabela 4.1 – Tempo por simulação para mensagens enviadas pelo orquestrador através do Apache Kafka

Uma Regra	Cinco Regras	Cinquenta Regras
05.81s	07.70s	16.53s
05.87s	05.81s	16.64s
04.52s	08.89s	19.97s
04.94s	05.87s	19.06s
04.24s	09.52s	18.08s

Tabela 4.2 – Tempo por simulação para mensagens enviadas pelo agente externo através do Tangle

Uma Regra	Cinco Regras	Cinquenta Regras
62.193s	118.191s	364.225s
57.832s	64.883s	359.432s
118.232s	138.432s	422.10s
71.110s	91.538s	320.23s
61.830s	88.380s	344.332s

Tabela 4.3 – Tempo médio para mensagens enviadas pelo orquestrador através do Apache Kafka

Uma Regra	Cinco Regras	Cinquenta Regras
05.076s	07.558s	18.056s

Tabela 4.4 – Tempo médio para mensagens enviadas pelo agente externo através do Tangle

Uma Regra	Cinco Regras	Cinquenta Regras
74.239s	100.28s	361.861s

A partir dos resultados da simulação é possível concluir que a comunicação direta entre os dispositivos e o orquestrador, através do Apache Kafka, apresenta melhores resultados em relação ao tempo para propagação de mensagens. Para os teste com uma e cinco regras enviadas, o Apache Kafka alcança resultados até quatorze vezes melhor em comparação ao Tangle e até vinte vezes com cinquenta regras transmitidas.

Esperava-se que o Tangle se mostrasse mais lento quando comparado ao Apache Kafka, à sua natureza de consenso descentralizado. Mas é importante ressaltar que o principal objetivo do Tangle na solução proposta não é se apresentar como um oponente ao Apache Kafka. Seu objetivo é permitir a comunicação de agentes externos com dispositivos em redes sobre diferentes jurisdições garantindo garantindo autorização, auditoria, transparência na troca de dados.

5 Conclusões e Trabalhos Futuros.

Este trabalho teve como objetivo desenvolver um sistema de prevenção e detecção de intrusões para gateways de IoT. Ao longo do trabalho foram apresentados os conceitos teóricos acerca de sistemas de detecção e prevenção de intrusão, bem como os desafios da utilização desses sistemas em dispositivos com recursos limitados.

Para resolver os problemas de utilização em dispositivos IoT este trabalho desenvolveu um Host Intrusion Prevention and Detection System (HIDPS) executado em gateways IoT foi desenvolvido juntamente com uma arquitetura completa composta por um orquestrador e servidores externos para gerenciar regras e provisionar dispositivos.

Na solução proposta, os agentes externos são responsáveis por compartilhar regras de suas redes com diretamente com os dispositivos de outras redes. Para a solução proposta atingir níveis aceitáveis de escalabilidade e segurança, um canal criptografado no Tangle foi utilizado para realizar a comunicação entre agentes externos e dispositivos e o Apache Kafka, uma solução de streaming de eventos, foi usado para a comunicação entre os dispositivos e orquestrador.

Para validar a arquitetura proposta duas análises foram realizadas. A primeira análise tratou de validar o funcionamento da solução como um todo. Para isso, dispositivos foram provisionados com o cliente HIDPS e regras forma criadas para verificar a presença do malware do Mirai nos dispositivos tanto no orquestrador quanto no agente externo. As regras foram publicadas nas respectivas redes e executadas pelo dispositivo, assim foi possível acompanhar o processo de execução da regra e a geração de relatórios para o orquestrador. Essa análise permitiu acompanhar o funcionamento da arquitetura como um todo, desde a criação de uma regra e provisionamento dos dispositivos até a execução das regras pelos dispositivos e o informe dos resultados ao orquestrador.

A segunda análise foi realizada através de uma simulação prática foi realizada para avaliar o tempo para transmissão de regras para os dispositivos pelo orquestrador e por agentes externos. Este estudo concluiu que as regras provenientes do orquestrador são recebidas mais rapidamente do que as de agentes externos. Esse comportamento era esperado devido à natureza do canal para troca de dados. A solução de streaming de eventos é mais rápida para encaminhar mensagens do que a abordagem descentralizada.

Dessa forma esse trabalho atingiu seu objetivo inicial de desenvolver e analisar um sistema de detecção e prevenção de intrusão para ser executado em gateways IoT, bem como uma arquitetura híbrida escalável que permita a gerência e provisionamento de regras para esse sistema.

Como sugestões de trabalhos futuros tem-se adicionar detecção de anomalias ao sistema de prevenção e detecção de intrusão proposto, realização de análises de carga com vários dispositivos e executando o orquestrador em nuvem, análise da escalabilidade do Tangle adicionando mais nós e análise da solução proposta em frente a botnets mais recentes na literatura.

Bibliografia

[1] Rahul Agrawal et al. “Continuous Security in IoT Using Blockchain”. Em: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Calgary, AB, Canada, 2018, pp. 6423–6427.

[2] *Ansible*. URL: <https://www.ansible.com/> (acesso em 17/10/2021).

[3] Andreas M Antonopoulos. “Mastering bitcoin”. Em: (2019).

[4] *Apache Kafka*. URL: <https://kafka.apache.org/> (acesso em 17/10/2021).

[5] Gewu Bu, Önder Gürcan e Maria Potop-Butucaru. “G-IOTA: Fair and confidence aware tangle”. Em: *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE. 2019, pp. 644–649.

[6] FL d Caldas Filho et al. “Gerenci-amento de Serviços IoT com Gateway Semântico”. Em: *Atas das Con-ferências IADIS Ibero-Americanas WWW/Internet 2017 e Computação Aplicada 2017. Vilamoura*. IADIS Press, 2017, pp. 199–206.

[7] Rodrigo Campiolo et al. “Uma arquitetura para detecçao de ameaças cibernéticas baseada na análise de grandes volumes de dados”. Em: *Anais do I Workshop de Segurança Cibernética em Dispositivos Conectados*. SBC. 2018.

[8] Chain Monte Carlo. “Markov chain monte carlo and gibbs sampling”. Em: *Lecture notes for EEB* 581 (2004), p. 540.

[9] Anamika Chauhan et al. “Blockchain and Scalability”. Em: *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. 2018, pp. 122–128. DOI: <10.1109/QRS-C.2018.00034>.

[10] Hao Chen, Xueqin Jia e Heng Li. “A brief introduction to IoT gateway”. Em: *IET International Conference on Communication Technology and Application (ICCTA 2011)*. 2011, pp. 610–613. DOI: <10.1049/cp.2011.0740>.

[11] Usman W Chohan. “The double spending problem and cryptocurrencies”. Em: *Available at SSRN 3090174* (2017).

[12] Thomas H Cormen et al. *Introduction to algorithms*. MIT press, 2009.

[13] Michele De Donno et al. “DDoS-Capable IoT Malwares: Comparative Analysis and Mirai Investigation”. Em: *Security and Communication Networks* (2018). DOI: <10.1155/2018/7178164>.

[14] Bruno V. Dutra et al. “HIDS by signature for embedded devices in IoT networks”. en. Em: *Actas de las V Jornadas Nacionales de Investigación en Ciberseguridad (JNIC 2019)*. Universidad de Extremadura. Cáceres, Spain, jun. de 2019, pp. 53–61. ISBN: 978-84-09-12121-2.

[15] Alice Ensor, Sigrid Schefer-Wenzl e Igor Miladinovic. “Blockchains for iot payments: A survey”. Em: *2018 IEEE Globecom Workshops (GC Wkshps)*. IEEE. 2018, pp. 1–6.

[16] "Event-Driven Architecture". Em: *Enterprise Service Oriented Architectures*. Springer-Verlag, pp. 317–355. DOI: <10.1007/1-4020-3705-8_8>. URL: <http://dx.doi.org/10.1007/1-4020-3705-8_8>.

[17] Ala Al-Fuqaha et al. "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications". Em: *IEEE Communications Surveys Tutorials* 17.4 (2015), pp. 2347–2376. DOI: <10.1109/COMST.2015.2444095>.

[18] Daniel G. V. Gonçalves et al. "IPS architecture for IoT networks overlaid on SDN [Arquitetura de IPS para redes IoT sobrepostas em SDN]". Em: *XIX Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*. São Paulo-SP, 2019.

[19] Nazrul Hoque, Dhruba K. Bhattacharyya e Jugal K. Kalita. "Botnet in DDoS Attacks: Trends and Challenges". Em: *IEEE Communications Surveys Tutorials* 17.4 (2015), pp. 2242–2270. DOI: <10.1109/COMST.2015.2457491>.

[20] *Iota Streams*. URL: <https://wiki.iota.org/streams/welcome> (acesso em 17/10/2021).

[21] *Iota Streams*. URL: <https://blog.iota.org/iota-streams-update-september-2020-c3b8668e231e> (acesso em 17/10/2021).

[22] Guilherme de O Kfouri et al. "Design of a Distributed HIDS for IoT Backbone Components". Em: *FedCSIS (Communication Papers)*. Leipzig, Germany, 2019, pp. 81–88. DOI: <10.15439/2019F329>.

[23] Constantinos Kolias et al. "DDoS in the IoT: Mirai and Other Botnets". Em: *Computer* 50.7 (2017), pp. 80–84. DOI: <10.1109/MC.2017.201>.

[24] Daniel Kraft. "Difficulty control for blockchain-based consensus systems". Em: *Peer-to-peer Networking and Applications* 9.2 (2016), pp. 397–413.

[25] Jay Kreps, Neha Narkhede, Jun Rao et al. "Kafka: A distributed messaging system for log processing". Em: *Proceedings of the NetDB*. Vol. 11. 2011, pp. 1–7.

[26] Joshua A Kroll, Ian C Davey e Edward W Felten. "The economics of Bitcoin mining, or Bitcoin in the presence of adversaries". Em: *Proceedings of WEIS*. Vol. 2013. 2013, p. 11.

[27] Leslie Lamport, Robert Shostak e Marshall Pease. "The Byzantine generals problem". Em: *Concurrency: the Works of Leslie Lamport*. 2019, pp. 203–226.

[28] Kopelo Letou, Dhruwajita Devi e Yumnam Jayanta. "Host-based Intrusion Detection and Prevention System (HIDPS)". Em: *International Journal of Computer Applications* 69 (mai. de 2013), pp. 30–31. DOI: <10.5120/12136-8419>.

[29] Rafael Z. A. da Mata et al. *Uma Análise Competitiva entre as Tecnologias Blockchain e Tangle para o Projeto de Aplicações IoT ST and Education: Algorithms, Systems and Higher Education View project Multimedia Content Distribution over the Internet View project*.

[30] Rafael Z. A. da Mata et al. "Distributed Architecture for Intrusion Detection in IoT Networks using Smart Contracts". Em: *Actas de las V Jornadas Nacionales de Investigación en Ciberseguridad (JNIC 2021)*. Cáceres, Spain, 2021. DOI: <10.18239/jornadas_2021.34.01>.

[31] Weizhi Meng et al. “When Intrusion Detection Meets Blockchain Technology: A Review”. Em: *IEEE Access* 6 (2018), pp. 10179–10188. DOI: <10.1109/ACCESS.2018.2799854>.

[32] David Mudzingwa e Rajeev Agrawal. “A study of methodologies used in intrusion detection and prevention systems (IDPS)”. Em: *2012 Proceedings of IEEE Southeastcon*. IEEE. 2012, pp. 1–6.

[33] Satoshi Nakamoto. “Bitcoin: A peer-to-peer electronic cash system”. Em: *Decentralized Business Review* (2008), p. 21260.

[34] Thomas Philbeck e Nicholas Davis. “THE FOURTH INDUSTRIAL REVOLUTION: SHAPING A NEW ERA”. Em: *Journal of International Affairs* 72.1 (2018), pp. 17–22. ISSN: 0022197X. URL: <https://www.jstor.org/stable/26588339>.

[35] Serguei Popov. *The Tangle*. Rel. técn. 2018.

[36] Bruno J. G. Praciano et al. “Segurança do Ambiente Usando Dispositivo IoT com Processamento Distribuído”. Em: *Atas das Conferências Ibero-Americanas WWW/Internet 2019 e Computação Aplicada 2019*. Lisbon, Portugal, 2019, pp. 163–170.

[37] Miguel A Prada et al. “Communication with resource-constrained devices through MQTT for control education”. Em: *IFAC-PapersOnLine* 49.6 (2016), pp. 150–155.

[38] Leonard Richardson e Sam Ruby. *RESTful web services*. "O’Reilly Media, Inc.", 2008.

[39] Muhammad Saad et al. “Mempool optimization for defending against ddos attacks in pow-based blockchain systems”. Em: *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE. 2019, pp. 285–292.

[40] Karen Scarfone, Peter Mell et al. “Guide to intrusion detection and prevention systems (idps)”. Em: *NIST special publication* 800.2007 (2007), p. 94.

[41] Steven R Snapp et al. “DIDS (distributed intrusion detection system)-motivation, architecture, and an early prototype”. Em: (2017).

[42] Statista. *Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2030*. Jan. de 2021. URL: <https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide>.

[43] Ali Sunyaev. “Distributed Ledger Technology”. Em: *Internet Computing*. Springer International Publishing, 2020, pp. 265–299. DOI: <10.1007/978-3-030-34957-8_9>. URL: <http://dx.doi.org/10.1007/978-3-030-34957-8_9>.

[44] Lionel N. Tidjon, Marc Frappier e Amel Mammar. “Intrusion Detection Systems: A Cross-Domain Overview”. Em: *IEEE Communications Surveys Tutorials* 21.4 (2019), pp. 3639–3681. DOI: <10.1109/COMST.2019.2922584>.

[45] David Wagner e Paolo Soto. “Mimicry attacks on host-based intrusion detection systems”. Em: *Proceedings of the 9th ACM Conference on Computer and Communications Security*. ACM. 2002, pp. 255–264.

[46] Dylan Yaga et al. “Blockchain technology overview”. Em: *arXiv preprint arXiv:1906.11078* (2019).

[47] Congcong Ye et al. “Analysis of Security in Blockchain: Case Study in 51% Attack Detecting”. Em: *2018 5th International Conference on Dependable Systems and Their Applications (DSA)*. 2018, pp. 15–24. DOI: <10.1109/DSA.2018.00015>.

[48] Congcong Ye et al. “Analysis of security in blockchain: Case study in 51%-attack detecting”. Em: *2018 5th International Conference on Dependable Systems and Their Applications (DSA)*. IEEE. 2018, pp. 15–24.

[49] Zibin Zheng et al. “Blockchain challenges and opportunities: A survey”. Em: *International Journal of Web and Grid Services* 14.4 (2018), pp. 352–375.

[50] Qian Zhu et al. “Iot gateway: Bridgingwireless sensor networks into internet of things”. Em: *2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*. Ieee. 2010, pp. 347–352.

[51] Quanyan Zhu et al. “A game-theoretic approach to rule sharing mechanism in networked intrusion detection systems: Robustness, incentives and security”. Em: *2011 50th IEEE Conference on Decision and Control and European Control Conference*. 2011, pp. 243–248. DOI: <10.1109/CDC.2011.6161171>.

ANEXO A – Playbook Ansible

```
---
---
# YAML documents begin with the document separator ---

# The minus in YAML this indicates a list item.
# The playbook contains a list
# of plays, with each play being a dictionary
-

  # Hosts: where our play will run and options it will run with
  hosts: gateways
  user: root
  become: true

  # Vars: variables that will apply to the play, on all target systems

  # Tasks: the list of tasks that will be executed within the playbook
  tasks:
    # This can be changed to a script command
    # if the hids_client is fetched from a remote
    - name: Copy HIDS Client
      synchronize:
        src: ./hids_client/
        dest: /home/hids_client
        rsync_opts:
          - "--exclude=.git"

    - name: Install HIDS dependencies
      command: bash /home/hids_client/install.sh
      async: 10000


    - name: Run HIDS
      command: bash /home/hids_client/run.sh
      async: 10000
```

```
  # Handlers: the list of handlers that are
  # executed as a notify key from a task

  # Roles: list of roles to be imported into the play

# Three dots indicate the end of a YAML document
...
```

ANEXO B – Docker Compose

```
version: "3"
services:
  db:
    image: postgres
    volumes:
      - postgresdata:/var/lib/postgresql/data
    ports:
      - "5432:5432"
    environment:
      POSTGRES_DB: orchestrator
      POSTGRES_PASSWORD: postgres

  orchestrator:
    build: .
    container_name: orchestrator
    logging:
      driver: gelf
      options:
        gelf-address: 'udp://localhost:12201'
    volumes:
      - .:/myapp
    ports:
      - "3000:3000"
    depends_on:
      - db

  elasticsearch:
    build:
      context: elasticsearch/
    volumes:
      - ./elasticsearch/config/elasticsearch.yml:
        /usr/share/elasticsearch/config/elasticsearch.yml:ro
      - ./elasticsearch/esdata:/usr/share/elasticsearch/data
    ports:
      - "9200:9200"
      - "9300:9300"
    environment:
```

```
    ES_JAVA_OPTS: "-Xmx256m -Xms256m"

kafka-ui:
  build:
    context: kafka-ui/
  ports:
    - "8080:8080"
  depends_on:
    - kafka

logstash:
  build:
    context: logstash/
  volumes:
    - ./logstash/config/logstash.yml:
      /usr/share/logstash/config/logstash.yml:ro
    - ./logstash/pipeline:/usr/share/logstash/pipeline:ro
  ports:
    - "12201:12201/udp"
  environment:
    LS_JAVA_OPTS: "-Xmx256m -Xms256m"
  depends_on:
    - elasticsearch
    - kafka

kibana:
  build:
    context: kibana/
  volumes:
    - ./kibana/config/:/usr/share/kibana/config:ro
  ports:
    - "5601:5601"
  depends_on:
    - elasticsearch

zookeeper:
  image: 'bitnami/zookeeper:latest'
  ports:
    - '2181:2181'
  environment:
    - ALLOW_ANONYMOUS_LOGIN=yes
kafka:
```

```
    image: 'bitnami/kafka:latest'
    environment:
      - KAFKA_BROKER_ID=1
      - KAFKA_CFG_LISTENERS=CLIENT://:9092,EXTERNAL://:9093
      - KAFKA_CFG_ADVERTISED_LISTENERS=CLIENT://kafka:9092,
                                                    EXTERNAL://localhost:9093
      - KAFKA_CFG_ZOOKEEPER_CONNECT=zookeeper:2181
      - ALLOW_PLAINTEXT_LISTENER=yes
      - KAFKA_CFG_LISTENER_SECURITY_PROTOCOL_MAP=CLIENT:PLAINTEXT,
                                                          EXTERNAL:PLAINTEXT
      - KAFKA_INTER_BROKER_LISTENER_NAME=CLIENT
    depends_on:
      - zookeeper

  kafka-broker:
    image: 'bitnami/kafka:latest'
    environment:
      - KAFKA_BROKER_ID=2
      - KAFKA_CFG_LISTENERS=CLIENT://:9092
      - KAFKA_CFG_ADVERTISED_LISTENERS=CLIENT://kafka-broker:9092
      - KAFKA_CFG_ZOOKEEPER_CONNECT=zookeeper:2181
      - ALLOW_PLAINTEXT_LISTENER=yes
      - KAFKA_CFG_LISTENER_SECURITY_PROTOCOL_MAP=CLIENT:PLAINTEXT,
                                                          EXTERNAL:PLAINTEXT
      - KAFKA_INTER_BROKER_LISTENER_NAME=CLIENT
    depends_on:
      - zookeeper

networks:
  default:
    name: 'hids_iot_network'

volumes:
  postgresdata:
    driver: local
```

ANEXO C – Configuração Logstash

```
input {
  kafka {
    bootstrap_servers  => "kafka:9092, kafka-broker:9092"
    topics => "devices_rules_report"
  }
}

output {
  elasticsearch {
    hosts => "elasticsearch:9200"
  }
}

filter {
  json {
    source => "message"
  }
}
```

ANEXO D – Dockerfile cliente HIDPS

```
FROM alpine:3.7

WORKDIR /usr/src/app
COPY . .

RUN apk add python3 py-pip
RUN apk add --no-cache bash
RUN pip3 install -r requirements.txt

CMD ["/entry.sh"]
```

D.1 entry.sh file

```
#!/bin/sh

# start cron
/usr/sbin/crond -f -l 8

# start Kafka consumer
export RULES_GROUP_ID=1 # Usually provided in device provisioning
python3 /usr/src/app/rules_consumer.py
```