IoTSafe: Detecção de Ataques Baseada em Redes Neurais Profundas

Fábio Coutinho dos Santos[1], Aldri Luiz dos Santos[2], Fátima Duarte-Figueiredo[1]

[1]Departamento de Ciência da Computação – PUC Minas – Belo Horizonte – MG – Brasil

[2]Departamento de Ciência da Computação – UFMG – Belo Horizonte – MG – Brasil

fabio.coutinho@sga.pucminas.br, aldri@dcc.ufmg.br, fatimafig@pucminas.br

Abstract. *The massive use of the IoT (Internet of Things) in urban scenarios exposes several systems of different natures to attacks. Although proposals for authentication and encryption or attack detection systems, IDS (Intrusion Detection System) are recurrent in the literature, there is a lack of solutions that can efficiently unite these two security tactics, especially if inserted in computing architectures at the edge of the network. This article proposes an attack detection module as a complement to IoTSafe's encryption and authentication mechanisms, presented in a previous work. This new module is based on deep learning and implements a deep neural network for attack identification. Specialized in communication via MQTT (Message Queuing Telemetry Transport), the module is implemented in the fog layer of the architecture. To evaluate the module and the complete IoTSafe architecture, tests were performed in three different environments, with comparative indications of resource consumption in each one. A case study was also done, showing that IoTSafe works with all proposed mechanisms, modules and layers. The results obtained, for the new module, show the effectiveness of the proposal, adding an accuracy of 99.57% and a precision of 99.66% in detecting attacks.*

Resumo. *O uso massivo da Internet das Coisas (IoT), em ambientes urbanos, expõe sistemas de naturezas diversas a ataques. São recorrentes, na literatura, propostas de autenticação e criptografia ou sistemas de detecção de ataques, IDS (Intrusion Detection System). Entretanto, faltam soluções que consigam unir, eficientemente, essas duas táticas de segurança, principalmente se inseridas em arquiteturas de computação de borda. Este artigo propõe uma solução para detecção de ataques como complemento aos mecanismos de criptografia e autenticação da IoTSafe, apresentados em um trabalho anterior. O novo modulo, de detecção, é baseado em deep learning e implementa uma rede neural profunda para identificação de ataques. Especializado em comunicação via MQTT (Message Queuing Telemetry Transport), o módulo é implantado na camada fog da arquitetura. Para avaliar o módulo e a arquitetura IoTSafe completa, testes em três ambientes foram realizados, com indicativos de consumo de recursos em cada um. Um estudo de caso mostrou que a IoTSafe funciona com todos os mecanismos, módulos e camadas propostos. O desempenho do novo módulo evidencia a sua eficácia, tendo obtido uma acurácia de 99,57% e precisão de 99,66% na detecção de ataques.*

1. Introdução

Ambientes urbanos estão, cada vez mais, imersos em soluções tecnológicas de controle e operação de serviços e infraestrutura. Atualmente, há mais de 20 bilhões de dispositivos de Internet das

Coisas (IoT, *Internet of Things*) conectados e com expectativa de superar os 30 bilhões, em 2024 [Lombardi et al. 2021]. Embora essa expansão de aplicações de IoT implique em inúmeros benefícios para o aumento de qualidade de vida, o alto número de objetos conectados, trafegando massiva quantidade de dados, expõe cada vez mais os sistemas a ataques. Podem ser citados ataques a sistemas de transportes, médicos e infraestruturas críticas [Zarpelão et al. 2017], que podem trazer prejuízos inestimáveis. O problema de segurança em IoT é acentuado pelo fato de não haver padrões especificamente projetados para dispositivos com recursos computacionais limitados, o que resulta em elevada exposição das redes, tornando-as suscetíveis a diversas modalidades de ataques [Frustaci et al. 2017].

Duas das principais táticas para reforçar a segurança do tráfego de informações, em sistemas IoT, são a autenticação dos dispositivos e a criptografia dos dados trocados. Entretanto, as limitações computacionais relativas à capacidade de processamento, memória e armazenamento dos dispositivos periféricos (sensores, atuadores, microcontroladores) tornam soluções clássicas e robustas de criptografia inviáveis na IoT [McCormack et al. 2020]. Na literatura atual, há uma busca por soluções que forneçam privacidade e integridade das informações nos sistemas, com baixo consumo de recursos e baixa sobrecarga na rede. Em geral, os trabalhos possuem algumas lacunas, como a falta de técnicas de computação de borda e a ausência de mecanismos de detecção de ataques na rede. Em trabalho prévio, foi apresentada a arquitetura IoTSafe, com seus mecanismos de autenticação e criptografia, empregando *fog computing* [dos Santos and de Freitas Mini 2021]. Apesar de apresentar técnicas suficientes de segurança, faltava, na arquitetura IoTSafe, métodos de detecção de ataques na rede para elevar os níveis de proteção das aplicações e preencher uma importante lacuna observada nos trabalhos na literatura.

Este trabalho apresenta um módulo de detecção de ataques implementado sob *fog computing* e especializado em comunicação via MQTT. Empregando *deep learning* em conjunto com os mecanismos de autenticação e criptografia, o módulo proposto aumenta consideravelmente o nível de segurança da arquitetura IotSafe. A IoTSafe foi estruturada em 4 níveis sendo focada em baixo consumo de recursos computacionais e baixa latência, empregando protocolos de comunicação otimizados e implementando um *gateway* na borda da rede. O módulo de detecção de ataques foi adicionado ao *gateway* e funciona em conjunto com os mecanismos de autenticação e criptografia já existentes. Resultados experimentais obtidos evidenciam um ganho de precisão e acurácia na detecção de ataques, além de tempos de respostas satisfatórios, quando todos os procedimentos de segurança são acionados, pela IoTSafe, em ambientes computacionais com recursos limitados. Além da avaliação da eficácia e do desempenho, um estudo de caso real foi conduzido para mostrar validação de uso real da arquitetura IoTSafe.

O restante do trabalho está organizado da seguinte maneira: Na Seção 2 são discutidos os trabalhos relacionados. A Seção 3 descreve arquitetura IoTSafe e seus componentes A Seção 4 apresenta uma avaliação de desempenho da arquitetura, obtida por meio de experimentos práticos. A Seção 5 apresenta a conclusão e os trabalhos futuros.

2. Trabalhos Relacionados

Alguns trabalhos da literatura propõem arquiteturas que utilizam *middlewares* de autenticação e criptografia em uma camada entre os dispositivos periféricos e os elementos de gerenciamento do sistema, como [Garg and Dave 2019] e [dos Santos and de Freitas Mini 2021] . Trabalhos que apresentam sistemas de detecção de ataques são amplamente encontrados na literatura, mas, geralmente, focam apenas nas métricas obtidas pelos algoritmos e não possuem associação com mecanismos de detecção e criptografia. Esses IDS (*Intrusion Detection System*) implementam algoritmos de aprendizado para detectar tráfego malicioso na rede. E em geral empregam bases de dados públicas como UNSW-NB15 e NSL-KDD, entre outras, para realizar o treinamento dos algorit-

mos de ML (*Machine Learning*) e DL (*Deep Learnig*). Em [dos Santos and de Freitas Mini 2021], apesar de ser proposta uma solução arquitetural eficiente, não há métodos de detecção de ataques. Em [Ullah and Mahmoud 2020] os autores sugerem um sistema IDS de dois níveis, implementado em uma camada *fog* que estaria entre as camadas de rede e de aplicação. No primeiro nível, o tráfego é classificado como normal ou malicioso. Caso seja malicioso, ele é enviado ao segundo nível onde é detectado qual tipo de ataque está ocorrendo. Os algoritmos utilizados para realizar estas tarefas são *Decision Tree* no primeiro nível e *Random Forest* no segundo. As bases usadas para implementar a solução foram UNSW-NB15 e CICIDS2017. A falta de preocupação com consumo de recursos computacionais é recorrente nas soluções de sistemas IDS encontrados na literatura. Os trabalhos geralmente são focados nos métodos de extração de *features* e de classificação das bases para gerar melhor desempenho dos algoritmos na detecção do tráfego malicioso, como pode ser observado também em [Shafi et al. 2018], [Soukup et al. 2019], [Islam et al. 2021] e [Sarhan et al. 2022]. Outros pontos negativos da utilização de algoritmos de aprendizado de máquina como solução é que eles produzem uma alta quantidade de falsos positivos e não possuem a capacidade de detectar ataques que não estão previamente listados nas bases de treinamento. Para mitigar o problema de não reconhecer novos comportamentos maliciosos que não estão contidos nas bases de dados, alguns trabalhos sugerem soluções utilizando *Extreme Learning* e *Deep Leaning*. [Prabavathy et al. 2018] propõem uma solução baseada em *Online Sequential Extreme Learning Machine* (OS-ELM) utilizando *fog computing*.

Outras soluções capazes de detectar ataques que não constam nas bases de dados de treinamento são apresentadas em [Ashiku and Dagli 2021] e [Abou El Houda et al. 2022], todas empregam *Deep Learning*. Um exemplo de sistema IDS focado no baixo consumo de recurso é o proposto por [Kirupakar and Shalinie 2019], no qual os autores argumentam que soluções clássicas para detecção de ataques não funcionam em ambientes IoT com restrição de recursos computacionais. Embora sejam soluções eficientes, a utilização das bases de dados mais tradicionais em conjunto com o protocolo MQTT traz alguns problemas específicos. A base UNSW-NB15, embora seja utilizada na elaboração de sistemas IDS para redes IoT, como apresentado em [Alrashdi et al. 2019] e [Ullah and Mahmoud 2020], possui todos os dados extraídos da rede baseados apenas no protocolo TCP, não incluindo dados que são especificamente do protocolo MQTT. Este problema também ocorre com a base de dados NSL-KDD que, embora seja usada para prevenir ataques em sistemas IoT que utilizam o protocolo MQTT, como apresentado em [Shalaginov et al. 2019], também não inclui os dados relativos ao tráfego sob MQTT gerado pelos dispositivos. A base de dados BoT-IoT sugerida por [Koroniotis et al. 2019] também é utilizada na construção de sistemas que utilizam *deep learning* para realizar a detecção de intrusos na rede, como no estudo apresentado por [Ge et al. 2019]. Para obter o melhor ajuste possível entre as características da camada de percepção e rede da arquitetura com o módulo de detecção de ataques, é adequada a base de dados MQTT-Set, apresentada em [Vaccari et al. 2020].

3. Arquitetura IoTSafe

Esta seção descreve a arquitetura IoTSafe, proposta em [dos Santos and de Freitas Mini 2021], que recebe o módulo de detecção de ataques baseado em redes neurais profundas proposto neste artigo. A IoTSafe é composta por uma camada de percepção, uma de rede, uma *fog*, e uma de aplicação, como mostra a Figura 1. A camada de percepção compreende os dispositivos IoT que tem o papel de coletar informações do mundo físico e enviá-las aos elementos contidos no núcleo da rede. A camada de rede coordena o transporte dessas informações até a camada subsequente denominada *fog*. Na camada *fog* estão os elementos essenciais da arquitetura IoTSafe, responsáveis por todo o processamento das informações, e pela aplicação dos métodos de segurança. A camada de aplicação é responsável pelo gerenciamento da rede e pelo armazenamento dos dados.

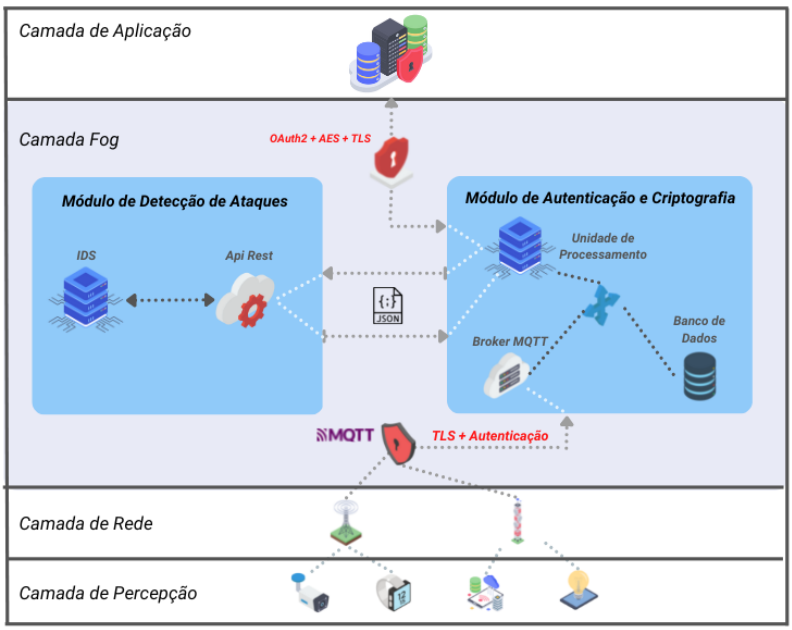

Figura 1. Arquitetura da IoTSafe

Na camada fog, o módulo de autenticação e criptografia é responsável por gerenciar e executar todas as tarefas relativas à autenticação dos dispositivos, autenticação do próprio *gateway* e criptografia dos dados trafegados, sendo empregados todos os mecanismos de segurança apresentados em [dos Santos and de Freitas Mini 2021].

O módulo de detecção de ataques, apresentado neste artigo, consiste em um sistema detecção de intrusos (IDS) que é acionado logo após a criação da requisição REST realizada pelo módulo de autenticação e criptografia. Dessa forma, o sistema IDS recebe esta requisição e realiza a análise do tráfego para identificar se ela é um ataque ou requisição normal. Caso seja considerada normal, a requisição segue para a nuvem. Caso contrário, ações para interromper esse tráfego são desencadeadas. O Algoritmo 1 descreve o processo de envio das mensagens pelos dispositivos IoT, em IoTSafe. Como mostra o algoritmo, os dispositivos IoT devem ser conectados, através de login com usuário e senha, com o *broker* MQTT instalado no *gateway fog* sob criptografia TLS. Após as conexões com o *broker*, os dispositivos estão aptos a enviar os dados coletados. O envio dos dados é realizado por meio da função *EnviaDado*, que recebe o id e a chave do dispositivo IoT, cria o objeto JSON e, em seguida, publica essa mensagem no tópico do *broker*, no qual o *gateway fog* está inscrito e recebe a mensagem. O algoritmo 2 mostra os processos realizados após a mensagem chegar no *gateway fog*.

Inicialmente, o *gateway fog* precisa se conectar com o *broker* MQTT e, para isso, também necessita de um certificado TLS, além das credenciais de autenticação, usuário e senha. A partir da conexão estabelecida com o *broker*, o *gateway* se inscreve no tópico que os dispositivos IoT enviam as mensagens por meio da função *SubscribeMqtt*, passando o tópico como parâmetro. Assim, fica estabelecida a função *ChegouMensagemMQTT*, que funciona como uma função de *callback* do protocolo MQTT, sendo que todas as mensagens que são publicadas, no *broker*, chegam nesta função.

Ao chegar uma mensagem, no tópico ao qual o *gateway* está inscrito, é chamada

Algoritmo 1: Algoritmo dos dispositivos IoT

```
Entradas: dado - dado coletado pelo dispositivo IoT
          idDispositivo - id do dispositivo IoT
          chaveDispositivo - KEY do dispositivo IoT
          cert - certificado TLS
          brokerUser - usuário do broker MQTT
          brokerSenha - senha do broker MQTT
          topicoPublish - tópico do broker MQTT

while Broker MQTT desconectado do
    if Conectado com o Broker MQTT then
        EnviaDado(Dado, idDispositivo, chaveDispositivo)
    else
        ConectarMQTT(cert, brokerUser, brokerSenha)
    end
end

EnviaDado (Dado, idDispositivo, chaveDispositivo)
    mensagemJson = CriarMensagemJson(Dado, idDispositivo,
    chaveDispositivo)
    publishMqtt(mensagemJson,topicoPublish)
```

a função *ExisteTokenJwt*, que confere se há um *token* gravado no banco de dados do *gateway*, associado com o id do dispositivo que enviou a mensagem. Caso exista o *token*, é realizada a criptografia do dado enviado pelo dispositivo e, em seguida, é criado o objeto JSON que é enviado ao módulo de detecção de ataques via requisição HTTP, pela função *EnviaRequisiçãoIDS*. Por outro lado, caso não exista o *token* gravado no banco de dados, o algoritmo autentica o dispositivo que enviou a mensagem, por meio da função *AutenticaDispositivo*. Caso a autenticação seja bem sucedida, o algoritmo realiza a autenticação do *gateway fog*, enviando seu id e a chave para o servidor que hospeda o banco de dados que está na nuvem, chamando a função *AutenticaGateway*. Caso a autenticação do *gateway* seja bem sucedida, essa função retorna o *token JWT* que, ao chegar no *gateway*, é gravado no banco de dados associado ao dispositivo IoT. Em seguida, é realizada a criptografia do dado enviado pelo dispositivo IoT e é criado o objeto JSON que é enviado para o módulo de detecção de ataques pela função *EnviaRequisiçãoIDS*.

O sistema IDS contido no módulo de detecção de ataques é construído através de *deep learnig* em conjunto com a base de dados MQTTset. A MQTTset disponibiliza um total de 33 *features* que podem ser empregadas para a implementação dos algoritmos de detecção. Além disso, ela classifica o tráfego em 6 tipos de classes distintas, sendo uma como tráfego legítimo e as outras cinco como tipos de ataques. Os cinco tipos de ataques classificados por ela são:

- *Brute Force Authentication*: Este ataque é caracterizado pela tentativa sistemática de recuperar as informações de autenticação, usuário e senha do *broker* MQTT.
- *Malformed Data*: Este ataque se caracteriza pela geração e pelo envio, para o *broker*, de vários pacotes mal formados, com o objetivo de aumentar as exceções causadas no *broker*.
- *SlowITe*: Este ataque consiste em um modelo de ataque de negação de serviço, em que

Algoritmo 2: Algoritmo do *gateway fog*

```
Entradas: idGateway - id do gateway
          chaveGateway - KEY do gateway
          cert - certificado TLS
          brokerUser - usuário do broker MQTT
          brokerSenha - senha do broker MQTT
          topicoSubscribe - tópico subscribe do broker MQTT

while Broker MQTT desconectado do
    if Conectado com o Broker MQTT then
        SubscribeMqtt(topicoSubscribe)
    else
        ConectarMQTT(cert, brokerUser, brokerSenha)
    end
end

ChegouMensagemMQTT (mensagem, topico)
if topico == topicoSubscribe then
    if ExisteTokenJwt(mensagem.idDispositivo) then
        dadoCriptografado = CriptografaDado(mensagem.dado)
        jsonDado = CriaRequisiçãoHTTP(dadoCriptografado)
        EnviaRequisiçãoIDS(jsonDado)
    else
        AutenticaDispositivo(mensagem.idDispositivo, mensagem.chaveDispositivo)
        token = AutenticaGateway(idGateway, chaveGateway)
        GravaTokenRecebido(token)
        dadoCriptografado = CriptografaDado(mensagem.dado)
        jsonDado = CriaRequisiçãoHTTP(dadoCriptografado)
        EnviaRequisiçãoIDS(jsonDado)
    end
end
```

ocorrem diversas conexões simultâneas com o *broker* MQTT, com o objetivo de acabar com as conexões disponíveis.

- *MQTT Publish Flood*: Este ataque consiste no envio de enorme quantidade de mensagens realizadas por dispositivos IoT maliciosos, fazendo com que se esgote a capacidade de processamento do *broker*.

- *Flooding Denial of Service*: Este ataque é caracterizado por dispositivos maliciosos conseguirem abrir grandes quantidades de conexões simultâneas com o *broker* e, concomitantemente a isso, enviarem grande quantidade de mensagens em cada uma dessas conexões.

Em [Chockwanich and Visoottiviseth 2019], os autores argumentam que, ao trabalhar com grande quantidade de dados, o tempo de treinamento e a chance de *overfitting* de uma rede neural profunda pode ser altamente influenciado pela quantidade de *features*. Logo, a redução da quantidade de *features* é extremamente importante para construir um modelo com alta dimensão de dados sem perda de informação. Neste contexto, para obter alta taxa de detecção, consumindo a menor quantidade de recurso computacional possível, a rede neural profunda da IoTSafe realiza uma classificação binária do fluxo. Testes foram aplicados para que esse objetivo fosse atingido com o menor número de *features*

possível. Essa estratégia é relevante, tendo em vista que a IoTSafe tem como objetivo ser eficiente em ambientes que possuam poucos recursos computacionais, se adequando aos novos padrões de computação de borda que, atualmente, fundamentam as aplicações IoT. Após a aplicação dos testes, foram selecionadas as oito *features* de maior relevância para a proposta da arquitetura, listadas a seguir.

- **tcp_flags**: é referente a uma *flag* do protocolo TCP que acompanha as mensagens publicadas no *broker*.
- **tcp.len**: refere-se ao tamanho total da mensagem enviada, incluindo o cabeçalho TCP.
- **mqtt_msg**: refere-se a mensagem literal que foi publicada no *broker*
- **mqtt.conflag.passwd**: refere-se a uma *flag* que indica se a conexão foi realizada utilizando a credencial de *password* do *broker*
- **mqtt.conflag.uname**: refere-se a uma *flag* que indica se a conexão foi realizada utilizando a credencial de usuário do *broker*
- **mqtt.len**: refere-se ao tamanho da mensagem literal publicada no *broker*, descartando o cabeçalho e outros detalhes do protocolo MQTT.
- **mqtt.qos**: refere-se ao nível de QoS que foi empregado na mensagem, 0,1 ou 2.
- **mqtt.retain**: refere-se a uma *flag* que pode ser acionada no envio de uma mensagem, fazendo com ela persista no *broker*, a *flag* de *retain*

Com o objetivo de reduzir a complexidade e consequentemente o processamento da rede neural para realizar a detecção, após a seleção das *features* foi necessário transformar a classificação dos cinco tipos de ataques que a base oferece, para uma classificação binária. Dessa forma, para todos os registros de classificação de tráfego normal, foi atribuído o valor 0. Para todos os registros que eram classificados como ataque, independentemente de qual tipo, foi atribuído o valor 1. Após empregar os métodos descritos para seleção das *fatures* mais adequadas e para o processamento dos dados, foi realizado o processo de implementação da rede neural profunda. A Figura 2 ilustra os processos de implementação da rede neural profunda, após a conclusão da preparação dos dados.

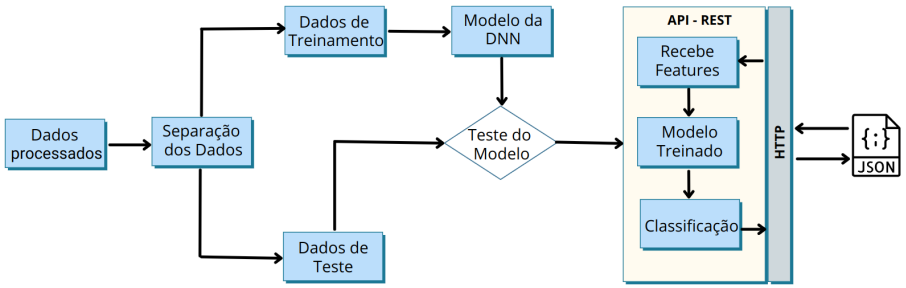

Figura 2. Módulo de Detecção de Ataques

Após o processamento dos dados, o algoritmo, implementado através da linguagem Python, associado com a biblioteca de inteligência artificial Tensor Flow, carrega o

arquivo .csv que contém a base de dados com um total de 1.527.378 registros. Em seguida, há o processo de separação desses registros em dados de treinamento e de teste da rede neural profunda. Assim, a base de dados é dividida em 70% dos registros para realização do treinamento e 30% para o teste. Após essa divisão, é iniciado o treinamento da rede neural profunda. A rede neural profunda, implementada para a IoTSafe, possui duas camadas ocultas contendo 128 neurônios em cada uma, sendo que o algoritmo classificador aplicado foi o *DNN Classifier*. Esse algoritmo classificador foi aplicado com o otimizador Adagrad e a função de ativação Relu, assim como em [Vaccari et al. 2020] e [Al-Garadi et al. 2020], sendo executada em 10000 passos. Todos esses parâmetros foram selecionados na aplicação do treinamento e do teste da rede neural.

Como resultado dos processos de treinamento e de teste da rede neural profunda, foi obtido o modelo treinado. Após o algoritmo produzir o modelo treinado, foi elaborada um API REST, criando um servidor local por meio da biblioteca Flask para Python, como em [Bonney et al. 2022]. Essa API é capaz de receber requisições via POST HTTP contendo os dados relacionados às *features* que foram empregadas na construção da rede neural profunda. Dessa forma, ao receber essa requisição com os dados em formato JSON, a API realiza a classificação da mensagem e envia essa classificação, também em formato JSON, como resposta da requisição. Com o resultado dessa classificação, a unidade de processamento do módulo de criptografia e autenticação pode descartar essa mensagem e iniciar ações para conter um possível ataque. A Figura 3 mostra todas as ferramentas e tecnologias que foram utilizadas na construção de cada uma das camadas da arquitetura, assim como foram aplicadas na execução dos experimentos práticos.

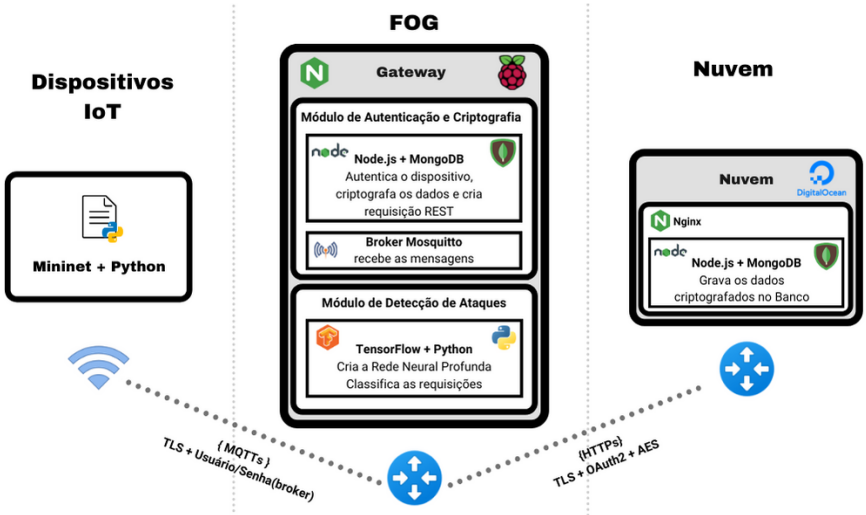

Figura 3. Ferramentas Utilizadas

4. Resultados

Para experimentos práticos no módulo de detecção de ataques, foram implementadas duas aplicações *back-end*. A primeira foi uma *API (Application Programming Interface)*, desenvolvida em Python, que, por meio da biblioteca do *Tensorflow*, realiza o treinamento da rede neural artificial, usando a base de dados MQTTset com todo o processo de extração de *features* e classificação. Após o treinamento da rede neural profunda, foi implementado um servidor, através da biblioteca Flask para Python. Ele fica em execução para realizar

o processo de predição da requisição, classificando-a em ataque ou normal. Os resultados obtidos, após a obtenção do modelo da rede neural profunda, foram de uma acurácia de 99,57% com precisão de 99.66%. Nesta avaliação, a acurácia apresenta a porcentagem de classificação correta dos dados, ou seja, tanto para os registros da base de teste que foram classificados como ataque, quanto para os que forma classificados como normal. A precisão representa a taxa de acerto na classificação dos ataques. A Tabela 1 mostra uma comparação entre as taxas alcançadas pela rede neural profunda proposta para a IoTSafe e algumas referências da literatura.

Tabela 1. Comparação dos valores alcançados pelas métricas

Proposta	Acurácia	Precisão
Pacheco et al. (2017)	-	96,20%
Prabavathy et al. (2018)	97,36%	97,72%
Sohal et al. (2018)	92%	-
Alrashdi et al. (2019)	-	79%
Soukup et al. (2019)	77%	-
Nagisetty e Gupta (2019)	99,24%	-
Ashiku e Dagli (2021)	95,40%	95,60%
IoTSafe	**99.57%**	**99.66%**

Para avaliar o desempenho da IotSafe, a camada *fog* foi implementada em três ambientes diferentes, com capacidades computacionais distintas. O primeiro ambiente contém um RaspberryPi 3 mobelo B, gerido por um sistema operacional Raspbian, possui 1 GB de memória RAM e um processador Broadcom Cortex-A53, com 4 núcleos de processamento de 1.2 GHz de *clock*. O segundo contém um computador pessoal gerido por um sistema operacional Ubuntu 20.04 com 8 GB de memória RAM e processador Intel(R) Core(TM) i7-3612QM com 8 núcleos de processamento de 2.10GHz de *clock*. O terceiro, contém um servidor gerido por um sistema operacional Ubuntu 16.04, com 32 GB de memória RAM e processador Intel(R) Xeon(R) CPU E3-1220 V2 com 4 núcleos de processamento de 3.10Ghz de *clock*. Dessa forma, nos três ambientes, foram aplicados os experimentos para avaliar a latência e consumo de recursos. Inicialmente, foram feitos experimentos para avaliar apenas o módulo de detecção de ataques. Para realizar a avaliação do tempo de resposta, a aplicação Nodejs envia mensagens com dados simulados para a API Python. Com o objetivo de mensurar o tempo de resposta da API Python, após construir o objeto JSON com os dados simulados, a aplicação Node.js captura um *timestamp* e, em seguida, envia o JSON via POST HTTP para a API Python. Dessa forma, assim que a resposta da requisição chega para a aplicação Node.js com a classificação realizada pela Api python, um novo *timestamp* é capturado e é calculada a diferença entre este e o primeiro *timestamp* capturado no envio da mensagem. Assim, é possível capturar quanto tempo foi gasto pela API Python para realizar a classificação. O fluxo de envio das mensagens foi realizado variando a frequência pela aplicação Node.js em 10, 20, 60, 100, 120, 150 e 200 mensagens por minuto (msg/min), sendo enviadas 100 mensagens em cada uma das frequências. Além disso, como o módulo de detecção de ataques está contido na camada *fog* da IotSafe, além de ter a variação da frequência de envio das mensagens, este experimento foi realizado com a API Python e a aplicação Node.js implementados nos três ambientes computacionais citados anteriormente. Assim, pode-se avaliar como o poder computacional pode influenciar no tempo de resposta médio e, consequentemente,

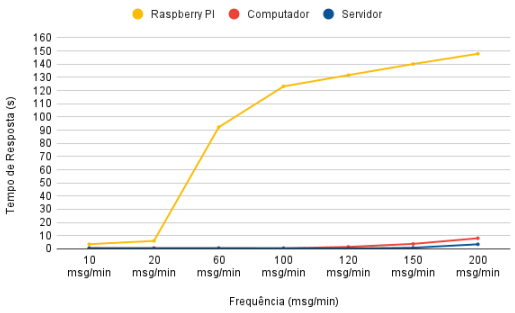

Figura 4. Latência Média da Api Python

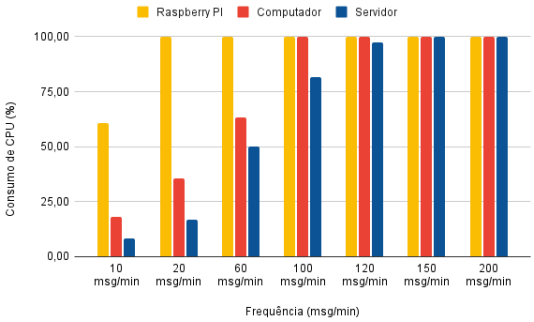

Figura 5. Consumo de CPU da API Python

na eficiência do módulo de detecção de ataques. O gráfico da Figura 6 apresenta o gráfico comparativo do tempo de resposta nos três ambientes experimentados.

Para estimar a eficiência da API, em [Stejskal 2018], os autores argumentam que o tempo de resposta em torno de 0,1 segundo é o ideal, até 1 segundo é satisfatório e acima de 10 segundos é inaceitável. Utilizando a menor frequência de mensagens, de 10 msg/min, no computador pessoal e no servidor da PUC Minas, a API obtém um tempo de resposta médio próximo a 0,5 s. Essa média se mantém em ambos os ambientes até 120 msg/min. Nesta frequência, no servidor, a API consegue manter o tempo de resposta próximo aos 0,5 s. No computador pessoal a média sobe para próximo de 1,5 s. Enviando 150 msg/min, o servidor ainda consegue manter em 0,8 s, enquanto, no computador pessoal, a média sobe para 3,7 s. Na frequência de 200 mgs/min o tempo de resposta do servidor alcança 3,5 s e do computador pessoal atinge 14,2 s. O Raspberry PI obtém um desempenho inferior aos outros ambientes: na frequência de 10 msg/min, o tempo de resposta atinge 3,5 s e, com 20 msg/min, ele é de 6,0 s. A partir de 60 msg/min, o desempenho piora consideravelmente, superando a marca dos 10 segundos, chegando a 150 s utilizando a frequência de 200 msg/min.

Para medir o consumo de recursos, também foram utilizadas as estruturas e aplicações Node.js e API python. O método aplicado para obter o consumo de CPU e de memória RAM foi diferente. Para capturar as referidas métricas foi utilizado o *framework* desenvolvido Python chamado psrecord. Foram identificados e monitorados, durante 1 minuto, os processos do sistema operacional que realizavam a execução das aplicações contidas na API Python. Neste 1 minuto, a aplicação Node.js enviou requisições POST HTTP contento um JSON com os dados simulados para que fosse feita a classificação pela API Python. Para mensurar o comportamento da API Python, nos ambientes computacionais diferentes. De forma análoga aos experimentos do tempo de resposta, a frequência das mensagens enviadas pela aplicação Node.js foi variada entre 10, 20, 60, 100, 120, 150 e 200 mensagens por minuto (msg/min). O gráfico da Figura 5 apresenta os resultados obtidos. Utilizando as frequências até 60 msg/min, o computador pessoal e o servidor utilizam até próximo de 50% de CPU. Por outro lado, o Raspberry PI atinge 65% de utilização de CPU com a frequência de 10 msg/min e, com 20 msg/min, atinge os 100%. O computador pessoal atinge 100% de CPU com 100 msg/min e o servidor da PUC apenas em 150 msg/min. Esses valores estão diretamente relacionados ao desempenho no tempo de resposta da API pois, observando os limites de eficiência discutidos no teste de

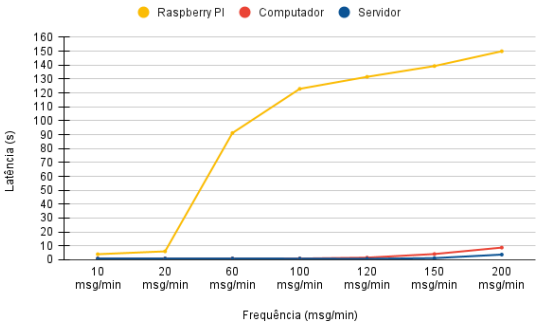

Figura 6. Latência Média IoTSafe

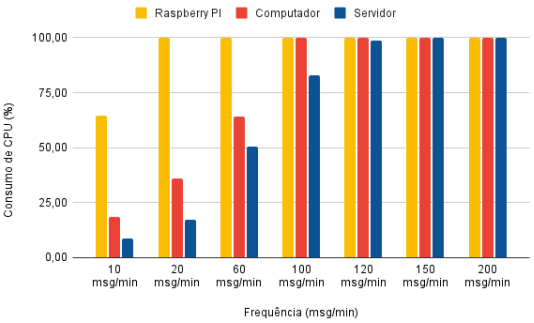

Figura 7. Consumo de CPU IotSafe

tempo de resposta, a API se torna ineficiente a partir do momento em que atinge 100% de utilização de CPU, nos três ambientes. Por outro lado, a utilização de memória RAM não possui um impacto significativo no desempenho da API, uma vez que nos 3 ambientes, o valor médio utilizado se manteve constante com a variação das frequências de envio das mensagens. Considerando o que cada ambiente disponibiliza de memória RAM, o Raspberry Py utilizou 67,9%, o computador pessoal utilizou 12,4% e o servidor 3,0%. Esses percentuais se mantiveram praticamente constantes em todas as frequências de envio.

Para mensurar a latência, foram realizados procedimentos de captura dos *timestamps* entre o envio e a gravação da requisição MQTT. O envio foi feito pelos dispositivos periféricos, que foram simulados por meio do emulador de rede Mininet. Este procedimento de captura da latência foi aplicado com as mesmas frequências de mensagens 10, 20, 60, 100, 120, 150 e 200 mensagens por minuto (msg/min), sendo enviadas 100 mensagens em cada frequência. O gráfico da Figura 6 mostra os resultados de latência do fluxo completo da IoTSafe processando 10 msg/min no Raspberry pi possui uma latência média total de 4,0 s, valor que sobe para 6,5 s processando 20 msg/min e alcança os 150 s com 200 msg/min. Por outro lado, quando executada no computador pessoal, com até 100 msg/min, a latência fica próxima a 0,9 s, passando para 1,55 s em 120 msg/min e alcançando 8,7 s processando 200 msg/min. No servidor, a latência fica em torno de 0,8 s até a frequência de 120 msg/min, passando para 1,18 s na frequência de 150 msg/min e atingindo 3,72 s processando 200 msg/min. Com estes resultados e, levando em conta os parâmetros estabelecidos em [Stejskal 2018] a IoTSafe entrega resultados satisfatórios no computador pessoal, com até 100 msg/min, e no servidor, com até 120 msg/min, tendo desempenho aceitável no Raspberry pi, com até 20 msg/min.

4.1. Estudo de Caso

Para comprovar a viabilidade e eficiência da arquitetura proposta foi implementada uma aplicação simples que emprega todos os elementos e mecanismos de segurança explicitados nas seções anteriores. Para tanto, foi produzido um *firmware* que coleta a quantidade de corrente elétrica que está passando em um fio por meio de um sensor de corrente SCT-013-000 integrado a um microcontrolador esp8266. A coleta é enviada para um *broker* MQTT que, ao receber a mensagem, executa todos os procedimentos de segurança da IoTSafe, descritos anteriormente, e, após a análise do módulo de detecção de ataques, envia o valor medido pelo sensor, de forma criptografada, para um banco de dados na nuvem. Além do *firmware*, também foi elaborada uma aplicação *front-end* que consome esses

dados gravados criptografados, descriptografa e produz um gráfico com as informações de consumo de energia elétrica em um determinado período de tempo. As Figuras 8 e 9 ilustram a aplicação *front-end* desenvolvida.

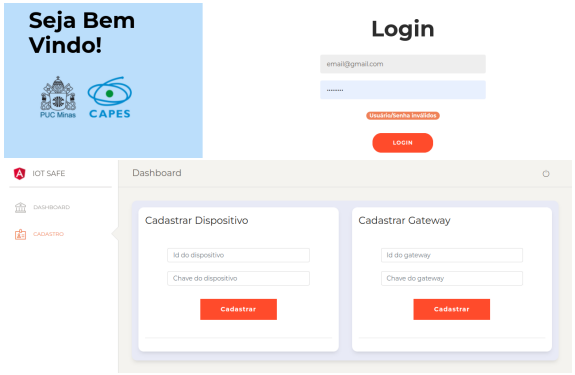

Figura 8. Tela de Login

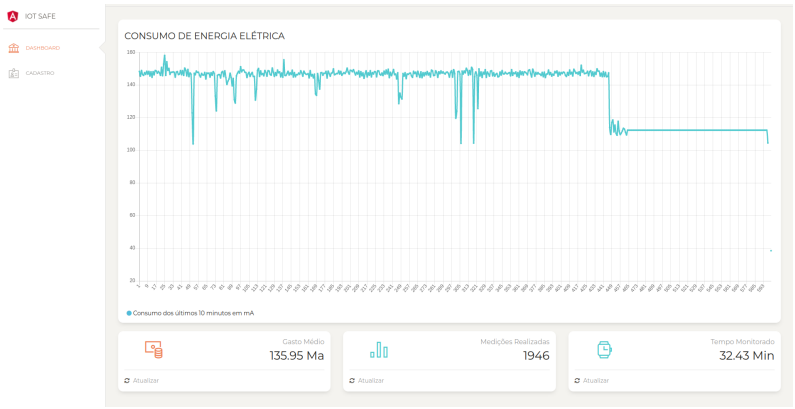

Figura 9. Tela Dashboard

5. Conclusões e Trabalhos Futuros

Este trabalho acrescentou um módulo de detecção de ataques na arquitetura IoTSafe aumentando a segurança contra ataques de diversas naturezas e que podem acontecer, principalmente, nas camadas de rede, *fog* e aplicação de um sistema IoT. Ao empregar *Fog Computing* aliada a um IDS e aos mecanismos de criptografia e autenticação, a IoTSafe atualizada consegue preencher uma importante lacuna encontrada em diversos trabalhos da literatura recente, além de adequar as aplicações à padrões arquiteturais que utilizam computação de borda de forma segura.

As principais contribuições deste trabalho consistiu na elaboração de um sistema IDS baseado em *deep learning* que consegue classificar, como ataques, padrões que não são aplicados no treinamento. Ele é adequado para ambientes IoT, pois é especializado em comunicação sob protocolo MQTT, alcança altas taxas de detecção empregando poucas *features*, reduzindo seu custo computacional, tornando-o, também, adequado para ambientes com pouco poder computacional. Implementou-se e avaliou-se uma arquitetura baseada em *fog computing*. Ela entrega autenticação e criptografia de ponta a ponta, possui um sistema IDS especializado em MQTT, pode ser distribuída em ambientes com

recursos computacionais limitados na camada *fog*, mantendo a eficiência e a eficácia das aplicações. As implementações da IoTSafe encontram-se em *https://gitlab.com/masters-coutinho*. Como trabalho futuro, pode-se empregar a IoTSafe em outros ambientes de baixo custo na camada *fog* da arquitetura e avaliar o desempenho, principalmente em relação à latência, para torna-lá ainda mais adequada às aplicações com requisitos mais rígidos, como sistemas de tempo real. Além disso, pode-se utilizar outros algoritmos, *frameworks* e linguagens na implementação do sistema IDS, visto que ele é o elemento mais impactante no consumo de recursos computacionais e na latência da arquitetura.

Referências

Abou El Houda, Z., Brik, B., and Khoukhi, L. (2022). "why should i trust your ids?": An explainable deep learning framework for intrusion detection systems in internet of things networks. *IEEE Open Journal of the Communications Society*, 3:1164–1176.

Al-Garadi, M. A., Mohamed, A., Al-Ali, A. K., Du, X., Ali, I., and Guizani, M. (2020). A survey of machine and deep learning methods for internet of things (iot) security. *IEEE Communications Surveys & Tutorials*, 22(3):1646–1685.

Alrashdi, I., Alqazzaz, A., Aloufi, E., Alharthi, R., Zohdy, M., and Ming, H. (2019). Ad-iot: Anomaly detection of iot cyberattacks in smart city using machine learning. In *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0305–0310.

Ashiku, L. and Dagli, C. (2021). Network intrusion detection system using deep learning. *Procedia Computer Science*, 185:239–247.

Bonney, M. S., De Angelis, M., Dal Borgo, M., Andrade, L., Beregi, S., Jamia, N., and Wagg, D. J. (2022). Development of a digital twin operational platform using python flask. *Data-Centric Engineering*, 3.

Chockwanich, N. and Visoottiviseth, V. (2019). Intrusion detection by deep learning with tensorflow. In *2019 21st international conference on advanced communication technology (ICACT)*, pages 654–659. IEEE.

dos Santos, F. C. and de Freitas Mini, R. A. (2021). Iotsafe-uma arquitetura baseada em fog computing para prover segurança em iot. In *Anais do XIX Workshop de Computação em Clouds e Aplicações*, pages 15–28. SBC.

Frustaci, M., Pace, P., Aloi, G., and Fortino, G. (2017). Evaluating critical security issues of the iot world: Present and future challenges. *IEEE Internet of things journal*, 5(4):2483–2495.

Garg, H. and Dave, M. (2019). Securing iot devices and securelyconnecting the dots using rest api and middleware. In *2019 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU)*, pages 1–6. IEEE.

Ge, M., Fu, X., Syed, N., Baig, Z., Teo, G., and Robles-Kelly, A. (2019). Deep learning-based intrusion detection for iot networks. In *2019 IEEE 24th pacific rim international symposium on dependable computing (PRDC)*, pages 256–25609. IEEE.

Islam, N., Farhin, F., Sultana, I., Kaiser, M. S., Rahman, M. S., Mahmud, M., Hosen, A., and Cho, G. H. (2021). Towards machine learning based intrusion detection in iot networks. *Comput. Mater. Contin*, 69(2):1801–1821.

Kirupakar, J. and Shalinie, S. M. (2019). Situation aware intrusion detection system design for industrial iot gateways. In *2019 International Conference on Computational Intelligence in Data Science (ICCIDS)*, pages 1–6. IEEE.

Koroniotis, N., Moustafa, N., Sitnikova, E., and Turnbull, B. (2019). Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset. *Future Generation Computer Systems*, 100:779–796.

Lombardi, M., Pascale, F., and Santaniello, D. (2021). Internet of things: A general overview between architectures, protocols and applications. *Information*, 12(2):87.

McCormack, M., Vasudevan, A., Liu, G., Echeverría, S., O'Meara, K., Lewis, G., and Sekar, V. (2020). Towards an architecture for trusted edge iot security gateways. In *3rd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 20)*. USENIX Association.

Prabavathy, S., Sundarakantham, K., and Shalinie, S. M. (2018). Design of cognitive fog computing for intrusion detection in internet of things. *Journal of Communications and Networks*, 20(3):291–298.

Sarhan, M., Layeghy, S., Moustafa, N., Gallagher, M., and Portmann, M. (2022). Feature extraction for machine learning-based intrusion detection in iot networks. *Digital Communications and Networks*.

Shafi, Q., Basit, A., Qaisar, S., Koay, A., and Welch, I. (2018). Fog-assisted sdn controlled framework for enduring anomaly detection in an iot network. *IEEE Access*, 6:73713–73723.

Shalaginov, A., Semeniuta, O., and Alazab, M. (2019). Meml: Resource-aware mqtt-based machine learning for network attacks detection on iot edge devices. In *Proceedings of the 12th IEEE/ACM Int. Conference on Utility and Cloud Computing Companion*, pages 123–128.

Soukup, D., Hujňák, O., Štefunko, S., Krejčí, R., and Grešák, E. (2019). Security framework for iot and fog computing networks. In *2019 Third International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC)*, pages 87–92. IEEE.

Stejskal, B. J. (2018). Performance testing and analysis of qpid dispatch router.

Ullah, I. and Mahmoud, Q. H. (2020). A two-level flow-based anomalous activity detection system for iot networks. *Electronics*, 9(3):530.

Vaccari, I., Chiola, G., Aiello, M., Mongelli, M., and Cambiaso, E. (2020). Mqttset, a new dataset for machine learning techniques on mqtt. *Sensors*, 20(22):6578.

Zarpelão, B. B., Miani, R. S., Kawakani, C. T., and de Alvarenga, S. C. (2017). A survey of intrusion detection in internet of things. *Journal of Net. and Computer Applications*, 84:25–37.