UNIVERSIDADE FEDERAL DO RIO GRANDE - FURG
CENTRO DE CIÊNCIAS COMPUTACIONAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO
CURSO DE MESTRADO EM ENGENHARIA DE COMPUTAÇÃO

Dissertação de Mestrado

Detecção de Ataques de Injeção de Dados em Sistemas ROS via Anomalias no Tráfego de Rede

Rodrigo Abrantes Antunes

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal do Rio Grande - FURG, como requisito parcial para a obtenção do grau de Mestre em Engenharia de Computação

Orientador: Prof. Dr. Paulo Lilles Jorge Drews Junior
Co-orientador: Prof. Dr. Bruno Lopes Dalmazo

Rio Grande, 2023

Ficha Catalográfica

A636d Antunes, Rodrigo Abrantes.
Detecção de ataques de injeção de dados em sistemas ROS via anomalias no tráfego de rede / Rodrigo Abrantes Antunes. – 2023.
65 f.

Dissertação (mestrado) – Universidade Federal do Rio Grande – FURG, Programa de Pós-Graduação em Computação, Rio Grande/RS, 2023.
Orientador: Dr. Paulo Lilles Jorge Drews Junior.
Coorientador: Dr.Bruno Lopes Dalmazo.

1. Robotica 2. ROS 3. Segurança 4. Detecção de intrusão 5. Aprendizado de máquina I. Drews Junior, Paulo Lilles Jorge II. Dalmazo, Bruno Lopes III. Título.

CDU 004.896

Catalogação na Fonte: Bibliotecário José Paulo dos Santos CRB 10/2344

FURG

Dissertação de Mestrado

Detecção de Ataques de Injeção de Dados em Sistemas ROS via Anomalias no Tráfego de Rede

Rodrigo Abrantes Antunes

Banca examinadora:

gov.br Documento assinado digitalmente
FERNANDO SANTOS OSORIO
Data: 22/02/2023 16:48:48-0300
Verifique em https://verificador.iti.br

Prof. Dr. Fernando Santos Osório

Prof. Dr. Jéferson Campos Nobre

gov.br Documento assinado digitalmente
MARCELO RITA PIAS
Data: 22/02/2023 15:42:38-0300
Verifique em https://verificador.iti.br

Prof. Dr. Marcelo Rita Pias

Prof. Dr. Paulo Lilles Jorge Drews Junior (Orientador)

Prof. Dr. Bruno Lopes Dalmazo (Coorientador)

Dedico este trabalho à minha família, por propiciarem todas as condições necessárias para um dia eu chegar aqui e por sempre me apoiarem nas minhas decisões, em especial, ao meu irmão Rafael, por ser uma fonte constante de alegria, por me lembrar, todos os dias, de dar valor às pequenas e mais simples coisas da vida e por me proporcionar, a cada dia, novos e importantes aprendizados. Por fim, dedico a todos que se empenham para o progresso da ciência e para a manutenção dela como uma fonte de verdades.

AGRADECIMENTOS

Agradeço primeiramente aos meus orientadores, professor Dr. Paulo Lilles Jorge Drews-Jr e professor Dr. Bruno Lopes Dalmazo, por terem detectado esta oportunidade de pesquisa e me ajudado a fazer dela um trabalho de mestrado.

Aos demais professores do programa, que contribuíram para o meu aprendizado e crescimento intelectual e aos colegas de curso que, apesar de não termos tido o convívio por conta da pandemia, não mediram esforços para prestar ajuda naquelas disciplinas mais difíceis.

Aos membros da banca, professor Dr. Fernando Santos Osório, professor Dr. Jéferson Campos Nobre e professor Dr. Marcelo Rita Pias, pelas valiosas observações durante a qualificação, que foram de suma importância para o amadurecimento desta pesquisa.

Aos colegas de trabalho do IFSul, *campus* Pelotas, que foram compreensíveis ao abraçarem as minhas atribuições quando precisei me ausentar para desenvolver as atividades de pesquisa.

E por fim, agradeço à FURG e ao ensino público pela oportunidade de realizar mais este sonho.

O maior bem do homem é uma mente inquieta.
— ISAAC ASIMOV

RESUMO

ANTUNES, Rodrigo Abrantes. **Detecção de Ataques de Injeção de Dados em Sistemas ROS via Anomalias no Tráfego de Rede**. 2023. 64 f. Dissertação (Mestrado) – Programa de Pós-Graduação em Computação. Universidade Federal do Rio Grande - FURG, Rio Grande.

Nas últimas décadas, inúmeros avanços tecnológicos permitiram aos robôs compartilharem ainda mais espaço com os seres humanos. Os robôs estão cada vez mais inseridos na nossa realidade, portanto eles devem ser seguros e confiáveis. No entanto, esses sistemas são construídos sobre plataformas de computação tradicionais, sendo suscetíveis aos mesmos ataques cibernéticos. Além disso, também introduzem um novo conjunto de problemas de segurança que podem resultar em violação de privacidade ou até mesmo em danos físicos. Nesse contexto, uma nova geração de software de robótica ganhou força. O *Robot Operating System* (ROS) é um dos mais populares softwares para pesquisadores e desenvolvedores de robôs, contudo, vários estudos têm demonstrado que ele traz consigo diversas vulnerabilidades que podem comprometer sua segurança e confiabilidade, possibilitando, dentre outros, o ataque de injeção de dados não autorizados. Este trabalho tem por objetivo contribuir com a segurança de sistemas ROS por meio da aplicação de um modelo de detecção de anomalias para reconhecer esses ataques através do tráfego de rede, uma abordagem mais abrangente que as encontradas na literatura, que dispensa alterações no código do ROS ou da aplicação e que não interfere no desempenho de ambos. Foi proposto um método empregando o algoritmo SVM, o qual foi treinado a partir de características do tráfego de rede de uma aplicação ROS. Resultados obtidos por meio de experimentos em ambiente simulado demonstraram um grande potencial no uso de aprendizado de máquina no reconhecimento dos ataques, com o algoritmo SVM obtendo 99% de acurácia e 98% de precisão nos experimentos com anomalias menos complexas e 92% de acurácia e 65% de precisão nos experimentos com anomalias mais complexas. O método também foi comparado com outros modelos de aprendizado, merecendo destaque o modelo *Multilayer Perceptron*, que obteve precisão e especificidade levemente superiores ao SVM.

Palavras-chave: Robótica, ROS, Segurança, Detecção de Intrusão, Aprendizado de Máquina.

ABSTRACT

ANTUNES, Rodrigo Abrantes. **Detection of Data Injection Attacks in ROS Systems via Network Traffic Anomalies**. 2023. 64 f. Dissertação (Mestrado) – Programa de Pós-Graduação em Computação. Universidade Federal do Rio Grande - FURG, Rio Grande.

In recent decades, numerous technological advances have allowed robots to share even more space with humans. Robots are increasingly inserted into our reality, so they must be secure and reliable. However, these systems are built on top of traditional computing platforms and are susceptible to the same cyber-attacks. In addition, they also introduce a new set of security issues that can result in a breach of privacy or even physical harm. In this context, a new generation of robotics software has gained momentum. The Robot Operating System (ROS) is one of the most popular software for robot researchers and developers, yet several studies have shown that it brings vulnerabilities that can compromise its security and reliability. This work aims to contribute to ROS security by applying an anomaly-based intrusion detection model to recognize data injection attacks, a more comprehensive approach than those found in the literature, which does not require modification in the ROS or application code and does not interfere with its performance. A method using the SVM algorithm was proposed, which was trained from the network traffic characteristics of a ROS application. Results obtained through experiments in a simulated environment demonstrated great potential in the use of machine learning to recognize the attacks, with the SVM algorithm reaching 99% of accuracy and 98% of precision in experiments with less complex anomalies and 92% of accuracy and 65% of precision in experiments with more complex anomalies. The method was compared with other learning models, with emphasis on the *Multilayer Perceptron* model, which achieved slightly higher precision and specificity than the SVM.

Keywords: Robotics, ROS, Security, Intrusion Detection, Machine Learning.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABS	Anomaly-based System
AD	Anomaly Detection
API	Application Programming Interface
AS	Authentication Server
AUC	Area Under The Curve
BSD	Berkeley Software Distribution
CPS	Cyber-Physical System
DDS	Data Distribution Service
DoS	Denial Of Service
GnuPG	GNU Privacy Guard
IDPS	Intrusion Detection and Prevention System
IDS	Intrusion Detection System
IP	Internet Protocol
IPsec	Internet Protocol Security
M2M	Machine to Machine
MD5	Message Digest 5
ML	Machine Learning
NIDS	Network-based Intrusion Detection System
OMG	Object Management Group
P2P	Peer-to-peer
PGP	Pretty Good Privacy
QoS	Quality of Service
RBF	Radial Basis Function
ROC	Receiver Operating Characteristic
ROS	Robot Operating System
RPC	Remote Procedure Call

SBS	Signature-based System
SLAM	Simultaneous Localization and Mapping
SSH	Secure Shell
SSL	Secure Sockets Layer
SVM	Support Vector Machine
TCP	Transmission Control Protocol
TDNN	Time Delay Neural Network
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
VPN	Virtual Private Network
XML	Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

A robótica está cada vez mais inserida na nossa realidade, desde manipuladores industriais a aspiradores de pó eles se fazem cada vez mais presentes. Na última metade do século houve inúmeros avanços na robótica em resposta à evolução das necessidades sociais humanas, desde a robótica industrial que libertou o operador humano de tarefas perigosas ou arriscadas até a recente explosão da robótica de campo e de serviço para auxiliar o ser humano [30]. Os robôs estão se tornando populares. Em um futuro muito próximo, os robôs estarão por toda parte, em missões militares, realizando cirurgias, construindo arranha-céus, auxiliando clientes em lojas, como atendentes de saúde, como assistentes de negócios e interagindo intimamente com as pessoas em uma infinidade de maneiras [11].

Como muitas dessas máquinas são autônomas, é importante que estejam seguras, bem protegidas e não sejam fáceis de serem comprometidas. Caso contrário, em vez de recursos úteis, elas poderiam rapidamente se tornar ferramentas perigosas, capazes de causar estragos e danos substanciais aos arredores e aos humanos para os quais foram projetados para servir [11]. Estes sistemas robóticos são construídos sobre plataformas de computação tradicionais, sendo conectados a atuadores, sensores, câmeras e outros tipos de hardware, assim, não apenas os robôs se tornam vulneráveis aos mesmos ataques cibernéticos de sistemas de computação tradicionais à medida que se tornam conectados em rede, mas também revelam todo um novo conjunto de problemas de segurança, que podem resultar em violação de privacidade se forem atacados ou até mesmo em danos físicos [60]. Esta questão vem se tornando ainda mais crítica nos dias atuais, onde os robôs estão compartilhando cada vez mais ambientes com os humanos e ganhando ainda mais espaço na indústria [67].

Esse avanço das tecnologias de robótica nos últimos anos também deu forças a uma nova geração de softwares voltados à simplificação do desenvolvimento e pesquisa em robótica, sendo um dos mais populares o *Robot Operating System* (ROS) [61]. O ROS é um *middleware* gratuito e de código aberto voltado para robótica que permite aos desenvolvedores criarem aplicativos robóticos de maneira rápida e fácil. Ele proporciona diversas funcionalidades tais como abstração de hardware, controle de baixo nível de dispositivos, implementação de funcionalidades comumente usadas, troca de mensagens

entre processos e gerenciamento de pacotes. O ROS promove a reutilização de código com diferentes hardwares, fornecendo uma grande quantidade de bibliotecas disponíveis para a comunidade [44]. Atualmente, a maioria dos robôs de pesquisa executa ROS e sua adoção está crescendo continuamente. Acredita-se que os sistemas robóticos baseados em ROS constituirão a maioria de todos os sistemas robóticos nos próximos cinco anos, tanto em ambientes acadêmicos quanto comerciais [59].

Apesar de ser uma plataforma amplamente utilizada, existem vulnerabilidades significativas que não são abordadas na estrutura do ROS. Diversos trabalhos relacionados à segurança em robótica podem ser encontrados na literatura, sendo que muitos apontam várias ameaças à segurança do ROS tais como [40], [71] e [16]. A falta de recursos de segurança adequados expõe os sistemas a uma variedade de vetores de ataque preocupantes, incluindo a possibilidade de falsificação de dados de sensores (*sensor spoofing*), falsificação de nodos (*node impersonation*), ataques de negação de serviço (DoS), interceptação de dados (*man-in-the-middle*), dentre outros [63]. Essas falhas ameaçam a segurança dos sistemas construídos usando a estrutura do ROS e, portanto, coloca em risco os usuários e sistemas relacionados. Devido ao uso cada vez mais difundido o ROS se torna um alvo significativo e vulnerável para ataques maliciosos. A segunda versão do ROS, intitulada ROS2 [48], aborda alguns destes problemas, porém muitas vulnerabilidades de segurança permanecem. Além disso, atualmente, a maioria dos robôs em atividade utiliza ROS1 [51], o que faz da segurança destes sistemas uma questão ainda em evidência.

Embora haja o reconhecimento e o interesse da comunidade, este tópico ainda é relativamente recente, com poucas abordagens propostas, como será visto no capítulo de trabalhos relacionados. Há, portanto, muito espaço para aprimoramento e busca por soluções mais eficientes, e, dada a vasta utilização do *middleware* ROS e o número expressivo de vulnerabilidades que tem sido demostrado, fica evidente a urgência por mais pesquisas e estudos.

1.1 Descrição do Problema

Uma das possibilidade de ataque ao sistema ROS é a injeção de dados não autorizados. A fim de tratar este problema, a literatura traz, em sua maioria, abordagens relacionadas a criptografia ou autenticação que, apesar de serem eficazes no aumento da segurança, acabam demandando alterações no código do sistema ou das aplicações, além de introduzirem camadas que podem interferir no desempenho em tempo real do qual grande parte dos sistemas robóticos necessita. Este trabalho, por sua vez, faz uso de técnicas de detecção de anomalias para reconhecer esses ataques através do tráfego de rede de uma aplicação ROS. Esta abordagem, além de contribuir com a segurança e tomada de decisão, serve como complemento às outras abordagens. Esta pesquisa foi desenvolvida

dentro do grupo de pesquisa em Automação e Robótica Inteligente (NAUTEC), do Centro de Ciências Computacionais (C3), da Universidade Federal do Rio Grande (FURG), sendo a continuação da pesquisa iniciada por Teixeira *et al.* [67].

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo geral deste trabalho é contribuir com a segurança e confiabilidade de sistemas robóticos baseados em ROS empregando técnicas de detecção de anomalias para reconhecer ataques de injeção de dados não autorizados e, deste modo, auxiliar na tomada de decisão.

1.2.2 Objetivos Específicos

A fim de atender o objetivo geral foram estipulados os seguintes objetivos específicos:

- Realizar uma revisão bibliográfica a fim de inteirar-se do atual estado da arte.
- Propor uma metodologia de detecção de anomalias a ser empregada em ambiente ROS.
- Estipular e modelar um cenário de aplicação ROS para avaliação do método.
- Comparar a técnica empregada com outras técnicas conhecidas.

1.3 Contribuições

A principal contribuição deste trabalho é a aplicação e avaliação de técnicas de detecção de anomalias para reconhecer ataques de injeção de dados não autorizados em sistemas ROS, com ênfase no algoritmo de Máquina de Vetores de Suporte (*Support Vector Machine*), técnica amplamente adotada no contexto de detecção de anomalias. Como contribuições específicas relevantes, tem-se a metodologia de elaboração do cenário e das anomalias e a construção e modelagem dos *datasets* [1], os quais podem ser utilizados em outros trabalhos como meio de avaliar novas metodologias. Como frutos desta pesquisa, foram aceitos dois artigos em congressos, relacionados abaixo:

- Antunes, R., Dalmazo, B., and Drews, P. (2022). Detecção de Ataques de Injeção de Dados no Tráfego de Rede de Sistemas ROS. In Anais do XXII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais, pages 384–389, Porto Alegre, RS, Brasil. SBC.

[1] Disponíveis em: https://github.com/rodrigoaantunes/adanros

- Antunes, R., Dalmazo, B., and Drews, P. (2022). Detecting Data Injection Attacks in ROS Systems using Machine Learning. In 2022 Latin American Robotics Symposium (LARS), 2022 Brazilian Symposium on Robotics (SBR), and 2022 Workshop on Robotics in Education (WRE), pages 1–6. IEEE.

1.4 Estrutura do Trabalho

Este trabalho está organizado em 6 capítulos, sendo o primeiro deles esta introdução, que apresentou os avanços da robótica nas últimas décadas e os novos desafios e preocupações com segurança que este avanço trouxe, apresentou também o *middleware* ROS, destacando algumas de suas características e funcionalidades bem como alguns problemas de segurança relacionados à sua arquitetura, introduzindo assim a problemática da pesquisa. Por fim foram descritos os objetivos gerais e específicos deste trabalho e as contribuições realizadas.

No Capítulo 2 são apresentados alguns conceitos importantes e necessários ao pleno entendimento deste trabalho, são abordados detalhes do *middleware* ROS e das técnicas utilizadas no desenvolvimento do método proposto. Em seguida, no Capítulo 3, são analisados vários trabalhos relacionados à problemática da pesquisa, destacando-se as vulnerabilidades encontradas pelos autores no sistema ROS e as abordagens propostas por cada um. Também é feita uma discussão a respeito, onde é apresentada a oportunidade de pesquisa explorada por este trabalho.

Já no Capítulo 4, é demonstrada a metodologia adotada no desenvolvimento do método proposto. É apresentada a metodologia do ataque de injeção de dados não autorizados e proposto um modelo para reconhecer este ataque. No Capítulo 5 é feita a avaliação do método, onde são apresentados os experimentos realizados e os resultados obtidos. Por fim, no Capítulo 6, são feitas algumas considerações finais e apontadas direções para pesquisas futuras.

2 FUNDAMENTAÇÃO TEÓRICA

Por ser um trabalho multidisciplinar e que envolve diferentes áreas de conhecimento, a fim de nivelar leitores, serão apresentados neste capítulo alguns conceitos importantes e necessários para o pleno entendimento deste trabalho. Primeiramente, será feita uma breve apresentação do *middleware* ROS e destacados alguns aspectos específicos ao alvo da pesquisa e, em seguida, será feita uma introdução a respeito das técnicas empregadas no desenvolvimento do método proposto.

2.1 Robot Operating System

O ROS é descrito em sua documentação como um *meta-operating system* de código aberto voltado para robótica [19]. Ele compartilha diversas características com sistemas operacionais convencionais tais como abstração de hardware, controle de baixo nível de dispositivos, implementação de funcionalidades comumente usadas, comunicação entre processos, dentre outros, mas também implementa outras funcionalidades tais como o fornecimento de ferramentas e bibliotecas para obter, construir, escrever e executar códigos em diversos computadores. Por conta disso, ele também pode ser classificado como um *middleware*, uma vez que ele estende as funcionalidades de um sistema operacional fornecendo serviços para as aplicações, sendo assim, uma camada de software acima do sistema operacional mas abaixo da aplicação que fornece uma abstração de programação comum entre diversos sistemas [8].

2.1.1 Arquitetura

A representação fundamental de um sistema ROS é a de um grafo: ele é composto por vários programas diferentes, executados simultaneamente, que se comunicam entre si através da transmissão de mensagens. Visualizando isso como um grafo os programas são os nós e a comunicação entre eles são as arestas, ou seja, um nó representa um módulo de software que está enviando ou recebendo mensagens, e uma aresta representa um fluxo de mensagens entre dois nós [62]. O grafo do ROS representa uma rede ponto a ponto de processos ROS que processam dados conjuntamente. Este grafo é alimentado

de diferentes maneiras pelos componentes básicos da arquitetura ROS: nodos, ROS *master*, mensagens, tópicos, servidor de parâmetros, serviços e *bags*. No entanto, apenas os quatro primeiros componentes são relevantes para esta pesquisa.

Os nodos são processos que executam computação, um sistema de controle de robô geralmente compreende muitos nodos. Por exemplo, um nodo controla os motores das rodas, outro executa a localização, outro executa o planejamento do caminho, e assim por diante. Eles se comunicam entre si enviando mensagens para um tópico, podendo ser tanto assinantes de um tópico (*subscribers*) quanto anunciantes (*publishers*). Os nodos que assinam um tópico solicitarão conexões de nodos que publicam esse tópico e estabelecerão essa conexão por meio de um protocolo de conexão acordado. O protocolo mais comumente utilizado no ROS é chamado TCPROS, que usa soquetes TCP/IP padrão. Um nodo ROS é desenvolvido por meio de uma biblioteca cliente ROS, como roscpp ou rospy. [18].

Já o ROS *master* atua como um serviço de nomes na rede, sendo o núcleo central de comunicação que armazena informações sobre todos os tópicos, serviços e nodos disponíveis, ele mantém um mapeamento de quais nodos fornecem quais tópicos/serviços, quais nodos estão inscritos em quais tópicos e em quais portas esses tópicos e nodos estão localizados. O papel do ROS *master* é possibilitar que nodos ROS se localizem uns aos outros individualmente. Uma vez que esses nodos se localizam, eles passam a se comunicar de forma ponto a ponto (P2P). Sem ele os nodos não seriam capazes de se encontrar, trocar mensagens ou invocar serviços.

O ROS *master* fornece uma API baseada em XML-RPC, a qual é invocada pelas bibliotecas cliente do ROS (empregadas pelos nodos) para armazenar e recuperar informações. Quando um nodo deseja publicar mensagens em um tópico ele envia uma requisição XML-RPC ao ROS *master* registrando suas informações e intenção. Já quando a intenção é se inscrever em um tópico, o nodo envia uma requisição XML-RPC ao ROS *master* informando os seus dados e o tópico desejado. O ROS *master* retorna o endereço do nodo que anunciou este tópico, caso disponível. Em seguida o nodo assinante envia uma requisição XML-RPC para o endereço do nodo anunciante fornecido pelo ROS *master*, informando que deseja estabelecer uma conexão para começar a receber as mensagens. Assim é criado um canal de comunicação ponto a ponto entre os nodos (Figura 1). Detalhes dessa comunicação são abordados de forma mais aprofundada na seção 4.2, onde é descrita a metodologia do ataque de injeção de dados não autorizados em sistemas ROS.

2.1.2 ROS 2

O sistema ROS começou a ser desenvolvido em 2007, tendo como principal foco a pesquisa acadêmica, porém ao longo dos anos ele foi se tornando cada vez mais difundido, passando a ser adotado em domínios além da pesquisa [32]. Produtos baseados em ROS começaram a surgir no mercado, incluindo robôs de fabricação, robôs agrícolas,

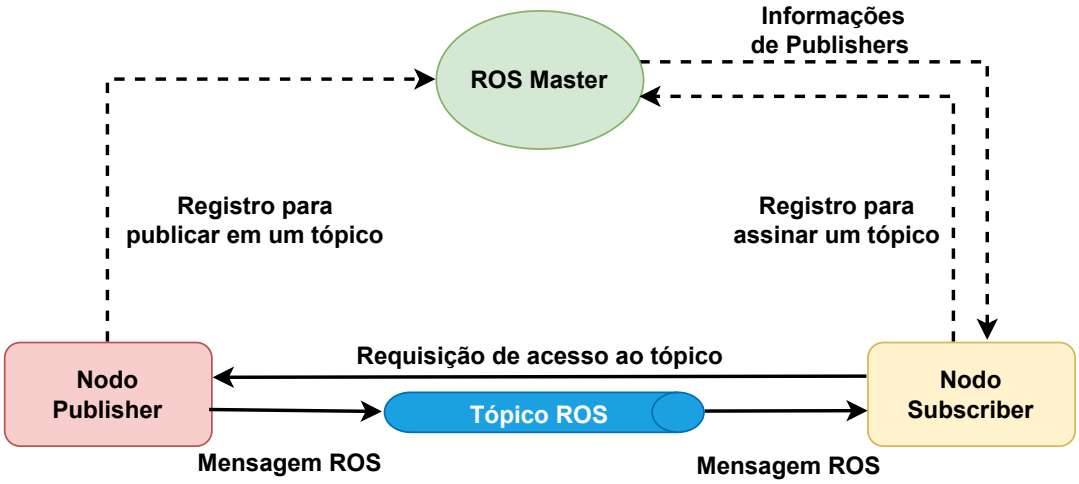

Figura 1: Modelo de comunicação entre nodos no ambiente ROS.

robôs de limpeza e outros. Agências governamentais também passaram a considerar o ROS para uso em seus sistemas de campo [7]. Muita coisa mudou na robótica e na comunidade ROS, e com o objetivo de se adaptar a estas mudanças uma nova versão do ROS, intitulada ROS2, foi anunciada em meados de 2014, tendo sua primeira versão de distribuição lançada no final de 2017.

Da mesma forma que a versão anterior, o ROS2 implementa um mecanismo anônimo de publicação/assinatura que permite a troca de mensagens entre diferentes processos ROS. No caso do ROS1, esse *middleware* de comunicação foi construído quase inteiramente do zero, foram desenvolvidos sistemas próprios para descoberta, definição de mensagens, serialização e transporte [32]. Porém dado o desenvolvimento, aprimoramento e/ou ampla adoção de várias novas tecnologias relevantes ao ROS nessas áreas, no ROS2 os desenvolvedores decidiram implementar estes conceitos de comunicação em cima de uma solução de*middleware* já existente conhecida como Data Distribution Service (DDS) [58]. O DDS é um padrão de comunicação *machine to machine* (M2M) especificado pelo *Object Management Group* (OMG) que unifica os componentes de um sistema, fornecendo conectividade de dados de baixa latência, extrema confiabilidade e uma arquitetura escalável, suportando a natureza distribuída dos dados de robótica e permitindo políticas baseadas em Qualidade de Serviço (QoS) [33].

Comparado ao ROS1, o ROS2 oferece avanços no campo de aplicações em tempo real, possibilita a aplicação de políticas de QoS para garantir comunicação eficiente entre processos e lidar com redes lentas ou com perdas, além disso apresenta uma arquitetura totalmente distribuída, não mais contando com o ROS *master* para descoberta (ponto único de falha), também proporciona suporte a novos casos de uso operando em ambientes distribuídos tais como veículos autônomos, sistemas multi-robôs, etc [63].

Essa nova versão trouxe inúmeras melhorias, inclusive no campo da segurança [28],

abordando alguns dos problemas mencionados anteriormente. Porém, muitas vulnerabilidades de segurança permanecem. Notavelmente, o padrão de segurança DDS foi introduzido para fornecer comunicação segura entre sistemas. No entanto, isso não aborda outras vulnerabilidades de segurança e deixa os sistemas ROS abertos a vários outros ataques. As soluções atuais para a segurança de ROS1 e ROS2 não fornecem proteção contra o comprometimento de nodos ou contra ataques de negação de serviço, não suportam políticas reativas que são atualizadas dinamicamente em tempo real e não podem impor políticas de rede de baixo nível [63]. Além disso, a maioria dos robôs em atividade atualmente utiliza ROS1, sendo que estudos apontam que o ROS2 está a anos de ser amplamente adotado para grandes tarefas de automação [51]. A segurança do ROS1 é uma questão ainda extremamente relevante. Apesar disso, o método apresentado pode ser estendido para suportar ROS2 em trabalhos futuros.

2.2 Sistemas de Detecção de Intrusão

Detecção de intrusão é o processo de monitorar os eventos que ocorrem em um sistema de computador ou rede, analisando-os em busca de sinais de problemas de segurança [5]. Basicamente, existem dois tipos principais de sistemas de detecção de intrusão (IDS - do inglês "*Intrusion Detection System*"): baseado em assinatura (SBS - do inglês "*Signature-based System*") e baseado em anomalias (ABS - do inglês "*Anomaly-based System*"). Sistemas SBS contam com técnicas de reconhecimento de padrões, mantendo um banco de dados de assinaturas de ataques já conhecidos e comparando-os com os dados analisados. Por outro lado, os sistemas ABS constroem um modelo estatístico descrevendo o tráfego normal da rede e qualquer comportamento anormal que se desvie do modelo é identificado. Uma vantagem de sistemas baseados em anomalias é poder detectar ataques novos imediatamente. No entanto, em comparação com SBSs, ABSs apresentam um maior número de falsos positivos na detecção de ataques [42].

2.3 Detecção de Anomalias

A detecção de anomalias (AD - do inglês "*Anomaly Detection*") refere-se ao problema de encontrar padrões nos dados que não estão de acordo com o comportamento esperado. Esses padrões desacordados são frequentemente referidos como anomalias, *outliers*, observações discordantes, exceções, aberrações, surpresas, peculiaridades ou contaminantes dependendo do domínio de aplicação. Destes, anomalias e *outliers* são os termos mais comumente utilizados no contexto de detecção de anomalias, sendo muitas vezes intercambiáveis. A detecção de anomalias é utilizada em uma ampla variedade de aplicações, tais como detecção de fraude para cartões de crédito, seguro ou saúde, detecção de intrusão para segurança cibernética, detecção de falha em sistemas críticos

de segurança e vigilância militar para atividades inimigas [12].

A importância da detecção de anomalias se deve ao fato de que as anomalias nos dados se traduzem em informações significativas (e frequentemente críticas) para tomadas de decisão em uma ampla variedade de domínios de aplicação. Por exemplo, um padrão de tráfego anômalo em uma rede de computadores pode significar que um computador hackeado está enviando dados confidenciais para um destino não autorizado [45]; do mesmo modo, um padrão de tráfego anômalo em uma rede de sistemas ciber-físicos (CPSs - do inglês "*Cyber-Physical Systems*") pode significar que um intruso está inserindo dados de imagem falsos [46] ou ainda tentando afetar a qualidade de produtos em uma linha de produção [57].

2.4 Máquinas de Vetores de Suporte

Fundamentada na Teoria da Aprendizagem Estatística, a Máquina de Vetores de Suporte (SVM - do inglês "*Support Vector Machine*"), é uma técnica de aprendizado baseada em métodos de aprendizado de máquina (ML - do inglês "*Machine Learning*"). Foi desenvolvida por Vapnik [69] com o intuito de resolver problemas de classificação. Esta técnica é amplamente utilizada em bioinformática, mineração de dados, reconhecimento de imagens, categorização de texto, reconhecimento de dígitos manuscritos, dentre outros [47]. Ela se encaixa no paradigma de aprendizado supervisionado, onde é fornecido um conhecimento prévio do ambiente na forma de dados separados/rotulados em classes, a partir dos quais o algoritmo é treinado a fim de aprender um modelo de predição. Cada instância no conjunto de treinamento contém um valor alvo (ou seja, os rótulos das classes) e vários atributos (ou seja, os recursos ou variáveis observadas). O objetivo é produzir um modelo (com base nos dados de treinamento) que prevê os valores alvo (as classes) de um conjunto de dados de teste dados apenas os seus atributos [38].

O conceito básico do SVM é encontrar uma linha, conhecida como hiperplano, que separe os dados representados em um espaço. Podem haver vários hiperplanos, por isso o objetivo é encontrar o melhor deles, aquele que possui a maior margem, ou seja, que fica mais distante dos vetores de suporte (dados mais próximos do hiperplano). Uma ilustração disso pode ser vista na Figura 2. Nela estão representadas duas classes de dados, "triângulo" e "círculo", e quatro possíveis hiperplanos separando as classes: H1, H2, H3 e H4; o hiperplano ideal, neste caso, é o H1 pois é o mais distante de ambas as classes.

Em muitos casos, porém, os dados não são linearmente separáveis, ou seja, não é possível traçar uma linha reta para separar os dados no espaço em que estão representados. A Figura 3 mostra um comparativo entre ambos os casos. No caso B a complexidade em separar os dados fica evidente, sendo assim, necessário o uso de uma técnica conhecida como *kernel trick*. Esta técnica consiste em um conjunto de funções matemáticas (*kernels*)

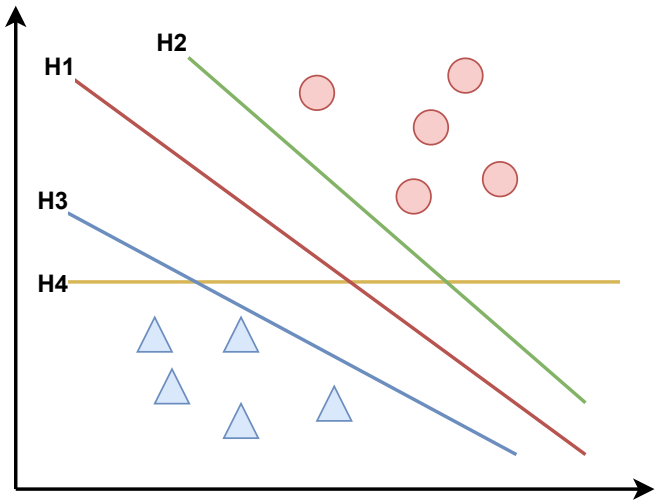

Figura 2: Possíveis hiperplanos de separação para um conjunto de dados bidimensional e binário.

cuja finalidade é mapear os dados em um espaço de maior dimensão no qual se torna possível obter um hiperplano que separe as classes corretamente. Para tal ela aplica uma transformação não linear do espaço, dessa forma, uma fronteira de separação não linear é visualizada quando se retorna ao espaço original (de menor dimensão) [53].

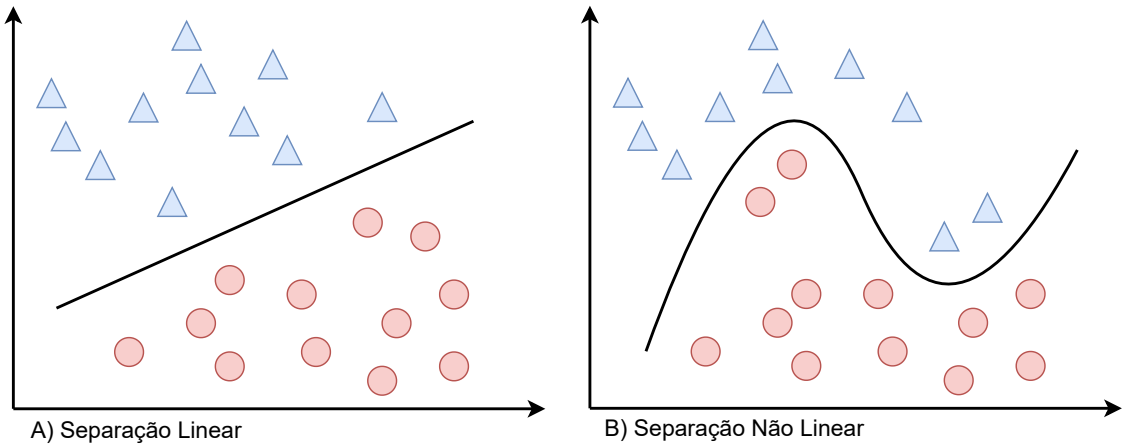

Figura 3: Separação linear e não linear no SVM.

Existem vários *kernels* propostos pela literatura, sendo que os mais utilizados são os *kernels* linear, polinomial, gaussiano e sigmoidal [38]. Neste trabalho, foi utilizado o *kernel* Gaussiano, também conhecido como *Radial Basis Function* (RBF). O *kernel* RBF é um dos mais utilizados para resolução de problemas de aprendizagem, inclusive é usado computacionalmente como padrão em muitas bibliotecas de linguagens de programação que utilizam o modelo SVM [41]. Além de ter uma boa performance em geral ele tem uma menor complexidade se comparado a outros *kernels* por conta da menor quantidade de parâmetros ajustáveis (hiperparâmetros). Este *kernel* mapeia amostras de forma não linear em um espaço dimensional superior para que, ao contrário do *kernel* linear, possa lidar

com casos em que a relação entre rótulos de classe e atributos é não linear. Existem dois parâmetros que podem ser ajustados na busca por um melhor resultado para o aprendizado do classificador, C (custo) e γ (gamma). O objetivo de ambos é trazer flexibilidade e versatilidade ao modelo, sendo que o C é relacionado à tolerância a erros de classificação e o γ ao peso das amostras que serão consideradas.

2.5 Considerações

Neste capítulo foram apresentados conceitos básicos do ROS e sua arquitetura, a fim de estabelecer uma base para o pleno entendimento do trabalho. Também foi discutido o conceito de detecção de intrusão, onde foram apresentados os principais tipos, sendo introduzida, em detalhes, a detecção de anomalias, abordagem utilizada neste trabalho. Por último, foi apresentada a técnica empregada no método proposto.

3 TRABALHOS RELACIONADOS

Neste capítulo, será discutido o atual estado da arte em segurança de sistemas robóticos baseados em ROS, serão discutidas as diversas vulnerabilidades e abordagens apresentadas pelos trabalhos bem como destacados alguns aspectos importantes de cada um. Também será feita uma discussão a respeito, onde será apresentada a oportunidade de pesquisa explorada por este trabalho.

Para realizar esta revisão foi feita uma pesquisa por palavras chave no repositório do Google Acadêmico, que é um mecanismo virtual de pesquisa que organiza e lista trabalhos da literatura acadêmica em uma extensa variedade de formatos de publicação. As palavras chave utilizadas foram *security*, *robotics*, *intrusion detection*, *vulnerabilities* e *robot operating system*. A Tabela 1 relaciona os artigos encontrados.

3.1 Abordagens empregando Autenticação ou Criptografia

Os trabalhos da literatura demonstram diversas vulnerabilidades existentes no ROS. O artigo *Secure communication for the Robot Operating System* [9] comenta que o padrão de publicação/assinatura [27] que o ROS implementa é muito útil para resolver problemas de compatibilidade e de dependências em um sistema robótico, fornecendo várias formas de abstração e transparência, porém é exatamente essa transparência, entretanto, que acaba também introduzindo vários riscos de segurança que não são suficientemente tratados no ROS [52]. Os *publishers* não podem controlar o consumo de seus dados e os *subscribers* não podem verificar facilmente a fonte e a integridade das informações recebidas. Esse ponto fraco pode ser usado para injetar informações maliciosas ou para espionar dados confidenciais. Nesse trabalho, foi proposto um canal de comunicação seguro para o ROS que lida com a comunicação entre dois nodos de maneira segura. Este canal estende os canais de comunicação TCP e UDP existentes implementando autenticação baseada em certificado, autorização, integridade e confidencialidade. O objetivo é estabelecer uma relação de confiança entre os nodos, protegendo a comunicação em nível ponto a ponto (P2P), nodos que não confiam uns nos outros não trocarão dados, excluindo efetivamente os nodos maliciosos de qualquer comunicação.

Tabela 1: Lista de trabalhos analisados.

Título do trabalho	Número de citações	Ano de publicação
Secure communication for the Robot Operating System [9]	73	*2017 (SysCon)*
The role of security in human-robot shared environments: A case study in ROS-based surveillance robots [60]	26	*2017 (RO-MAN)*
Application-level security for ROS-based applications [23]	79	*2016 (IROS)*
A preliminary cyber-physical security assessment of the Robot Operating System (ROS) [52]	99	*2013 (SPIE)*
Scanning the internet for ros: A view of security in robotics research [21]	77	*2019 (ICRA)*
Learning based anomaly detection for industrial arm applications [57]	30	*2018 (CPS-SPC)*
Intrusion detection on robot cameras using spatio-temporal autoencoders: A self-driving car application [3]	1	*2020 (VTC2020-Spring)*

No trabalho *The role of security in human-robot shared environments: A case study in ROS-based surveillance robots* [60], os autores destacam o fato de a comunicação entre nodos no ROS ser feita em texto plano via TCP e UDP. O ROS verifica apenas a soma de verificação MD5 (MD5 *checksum*) da estrutura da mensagem para garantir que as partes concordam com o leiaute da mensagem. Essa comunicação em texto plano permite que um ouvinte não autorizado intercepte e interprete facilmente a forma da mensagem, tenha acesso a informações críticas e possa introduzir mensagens falsas no sistema [52]. Também cita outras vulnerabilidades como o uso de portas TCP não seguras e desprotegidas e a ausência de mecanismos de autenticação. Merece destaque a observação de que em um sistema ROS vários serviços ficam centralizados no nodo mestre (ROS *master*), o que introduz um gargalo em potencial [35], inclusive em um evento de ataque de negação de serviço (DoS), pois um grande número de requisições falsas direcionadas a este nodo poderia fazer com que este deixasse de responder às requisições verdadeiras em tempo hábil. Para propor um método, o trabalho utilizou, como estudo de caso, um sistema de monitoramento e vigilância de edifícios e instalações [20] que consiste em múltiplos robôs de baixo custo que realizam o patrulhamento de infraestruturas de forma conjunta. Todos os robôs utilizam o sistema ROS, se comunicam entre si via wi-fi e são acessíveis remotamente através de uma interface web centralizada. A proposta consiste em um conjunto hierárquico de medidas de segurança nos vários níveis da arquitetura do sistema, desde o nível de hardware até o nível de rede. No que tange à comunicação entre os nodos ROS os autores propuseram um método semelhante à do artigo anterior, com adição de autenticação e autorização por meio de certificados, assinaturas digitais ou chaves privadas [25]. Também consideraram a possibilidade de utilizar criptografia para as mensagens, utilizando tecnologias como SSL, SSH, GnuPG, PGP e outras, além da possibilidade de integrar a extensão de segurança do protocolo IP (IPsec) no ROS.

Dieber et al. ressaltam em seu trabalho [23] que no futuro a produção industrial será feita em ambientes de rede que requerem a troca de dados entre diferentes localidades. Como a segurança não poderá mais ser fornecida isolando-se as redes do acesso externo, isso tornará os locais de produção acessíveis para ataques cibernéticos. Os processos de produção e o software que os controla não são suficientemente seguros para suportar tais ataques [14]. Assim como em [9], os autores dissertam sobre o padrão de publicação/assinatura que o ROS implementa também introduzir diversos riscos de segurança, salientando a incapacidade dos *publishers* e *subscribers* de verificar facilmente a fonte e a integridade das mensagens trafegadas. Uma menção interessante é sobre a possibilidade do uso de redes virtuais privadas (VPNs) para proteger toda a rede de produção, porém é observado que isso introduziria uma sobrecarga adicional ao sistema pois todos os dados não relevantes também seriam criptografados. Vale mencionar também a citação de um artigo [10] que propõe que uma abordagem multicamadas para segurança deve ser considerada em redes industriais. O trabalho propõe uma solução no nível de aplicação,

ou seja, sem alterações no código-fonte do ROS, somente nas aplicações a serem desenvolvidas. Ele propõe o uso de um servidor de autenticação (AS) dedicado para garantir que apenas nodos válidos façam parte da aplicação, os *publishers* e *subscribers* se registram no AS utilizando chaves públicas e privadas enquanto o AS autoriza ou desautoriza com base nesses registros.

Em *A Preliminary Cyber-Physical Security Assessment of the Robot Operating System (ROS)* [52], os autores concentraram-se na detecção de falhas de segurança em sistemas utilizando ROS ao invés de propor um método, para tal foi desenvolvido um protótipo de carro robô em pequena escala configurado como um *honeypot* ciber-físico. Um *honeypot* é um sistema computacional que é inserido para ser comprometido com o objetivo de obter informações sobre os atacantes, uma espécie de armadilha [50]. Como um *honeypot* convencional, o *honeypot* ciber-físico é projetado de forma a monitorar as tentativas de uso não autorizado de seus recursos ciber-físicos, a diferença é que este possui sensores e atuadores para permitir uma nova classe de vulnerabilidades e *exploits*. Este protótipo foi apresentado como um desafio em uma conferência de segurança cibernética, a DEF CON 20 [56], onde os participantes foram convidados a encontrar e explorar as vulnerabilidades de segurança do mesmo. Durante o desafio todo o tráfego de rede direcionado ao dispositivo foi monitorado e registrado. No decorrer do experimento houve vários incidentes, duas unidades de armazenamento falharam e seus dados foram permanentemente perdidos, um participante foi capaz de injetar mensagens ROS falsas no robô para acionar o acelerador, a interface web do dispositivo deixou de responder, as câmeras deixaram de enviar dados, dentre outros. Foi descoberto que é bastante fácil interceptar mensagens ROS utilizando o software de análise de protocolo de rede Wireshark [17], o que levanta a preocupante hipótese de que um indivíduo com experiência limitada na área de segurança cibernética poderia explorar com sucesso um sistema robótico baseado em ROS. Outra conclusão interessante dos autores é que a natureza complexa dos sistemas ciber-físicos torna muito difícil discernir se essas falhas foram ou não defeitos inerentes ao *honeypot* ou o se foram resultado de uma ação externa, o que implica que essa dificuldade pode ser explorada como um meio pelo qual os *exploits* podem ser disfarçados como simples *bugs*, embora, na verdade, o sistema esteja sob ataque.

O trabalho *Scanning the internet for ros: A view of security in robotics research* [21] teve uma abordagem interessante, o objetivo foi escanear a Internet em busca de dispositivos ROS desprotegidos e expostos publicamente, para tal foi feita uma busca por ROS *masters* conectados à Internet realizando-se uma varredura de todos os endereços públicos IPv4 (cerca de 3,7 bilhões de IPs) na porta TCP 11311, a porta ROS *master* padrão. Os resultados identificaram uma série de dispositivos totalmente desprotegidos espalhados pelo mundo, permitindo o acesso a sensores e atuadores robóticos por qualquer pessoa. Foi possível obter informações sobre a versão do sistema ROS, dados de sensores e atuadores, tópicos disponíveis, modelos de robôs, informações de simuladores, bibliotecas

usadas, entre outros. Foi possível também mover um robô de uma universidade nos Estados Unidos. Os autores fizeram algumas recomendações para evitar estes problemas, tais como o uso de VPN e *firewall*. Uma VPN fornece uma maneira abrangente de permitir que apenas *hosts* autorizados acessem um ROS *master*, já o *firewall* é uma maneira de proteger as instâncias ROS contra acesso não autorizado evitando a exposição dos serviços ROS à Internet pública.

3.2 Abordagens empregando Detecção de Anomalias

Uma outra abordagem possível é sugerida como trabalho futuro por Teixeira *et al.* [67]. Trata-se da aplicação de técnicas de detecção de intrusão. Propostas nesse sentido são apresentadas por Narayanan and Bobba [57] e por Amrouche *et al.* [3].

No trabalho de Narayanan and Bobba [57], o objetivo é detectar anomalias em sistemas de braços robóticos industriais através da análise das mensagens trafegadas no ambiente ROS. Foi proposto um sistema de detecção de anomalias empregando um modelo de aprendizado não supervisionado (*One-Class* SVM), o qual é treinado a partir do comportamento esperado, estabelecendo-se graus de tolerância para esse comportamento e o utilizando como base para classificar as ações do braço como anômalas ou benignas. Tudo que foge do comportamento esperado é considerado anômalo (*whitelisting*). As características utilizadas para treinamento são os estados das juntas, que são publicados em um tópico na forma de mensagens ROS, o sistema então se conecta nesse tópico e coleta e analisa estas mensagens. O autor conclui que a abordagem é promissora e destaca que melhor treinamento e ajuste do modelo são preocupações a serem trabalhadas no futuro.

Já Amrouche *et al.* [3] propuseram um sistema para detectar imagens anormais inseridas por um atacante em um fluxo de câmera. A metodologia consiste em analisar fluxos de vídeo vindos de carros autônomos com ROS utilizando um tipo de rede neural conhecido como *autoencoders*, e deste modo, reconstruir confiavelmente imagens de entrada levando em consideração imagens anteriores e prever imagens futuras dadas as imagens presentes e anteriores. Assim é feita uma comparação entre as imagens previstas e o fluxo de imagens atual a fim de detectar se há diferenças. Havendo diferenças entre as imagens, há o indicativo de uma intrusão em potencial.

3.3 Discussão

Conforme visto nas seções anteriores, há o reconhecimento e o interesse da comunidade em tornar o ROS mais seguro, porém este tópico ainda é relativamente recente, com poucas soluções apresentadas. Vários trabalhos demonstraram as diversas vulnerabilidades presentes no sistema ROS e algumas abordagens foram propostas para melhorar a segurança deste sistema. No entanto, a grande maioria foca em alterar o código do

sistema ou da aplicação a fim de aumentar a proteção contra possíveis ataques, ou seja, implementam camadas de segurança que agem na execução, podendo assim interferir no comportamento e, por consequência, no desempenho da aplicação. São aplicadas técnicas que envolvem algum tipo de criptografia ou autenticação, o que pode não ser viável em todos os casos. uma vez que isso pode comprometer o desempenho em tempo real do qual a maioria dos sistemas robóticos depende [67]. Apesar disso, o presente trabalho é complementar a essas abordagens e pode, de fato, se beneficiar dos mecanismos de segurança propostos nelas.

Uma outra abordagem possível é a detecção de anomalias, porém, abordagens desse tipo são ainda mais escassas na literatura. Os trabalhos encontrados focam em analisar o conteúdo das mensagens ROS (estado das juntas e fluxo de imagens), sendo aplicável a esses tipos de dados apenas. Dessa forma, tendo em vista o atual cenário, nota-se que a literatura carece de soluções mais abrangentes que não sejam dependentes de tipos de dados ou vetores de ataques. Nesse sentido, esse trabalho propõe uma avaliação do uso de técnicas de detecção de anomalias como meio para reconhecer, através do tráfego de rede, ataques de injeção de dados não autorizados em sistemas ROS, conforme será detalhado no capítulo seguinte.

4 MATERIAIS E MÉTODO

Neste capítulo, será apresentada a metodologia utilizada na pesquisa. Inicialmente serão abordados os principais vetores de ataque específicos ao *middleware* ROS e demonstrada, em detalhes, a metodologia do ataque explorado neste trabalho. Em seguida, será apresentado o método proposto para reconhecer esse ataque, onde serão discutidas a abordagem, técnica empregada e extração e processamento de *features*.

4.1 Vetores de Ataque ao Sistema ROS

Conforme visto anteriormente, um sistema ROS consiste em múltiplos processos computacionais, chamados de nodos, os quais constituem a representação fundamental do ROS, o grafo. Os nodos se comunicam por meio de canais, conhecidos como tópicos, e ao se comunicar assumem funções de *publisher* e/ou *subscriber*, enviando e/ou lendo mensagens por meio desses tópicos. Essa arquitetura de comunicação permite um baixo acoplamento entre os processos e uma maior independência e granularidade, mas ao mesmo tempo traz consigo diversos problemas de segurança [52].

Ao comunicar-se por meio de tópicos, os nodos não sabem com quem estão se comunicando e o nodo mestre é incapaz de impor um mapeamento entre tópicos e nodos [64]. O ROS faz uma distinção clara entre gerenciamento de aplicação (encontrar um *publisher* para um tópico ao qual se quer subscrever por exemplo) e comunicação de dados. O primeiro é tratado por meio de uma API XML-RPC, enquanto o segundo é comunicação baseada em TCP ou UDP. Em ambos, a segurança com relação à confidencialidade, integridade ou autenticidade não é levada em consideração, um nodo ROS não precisa se identificar ou se autenticar antes de executar qualquer ação e a API também não leva em consideração o que está acontecendo na rede [24].

Embora do ponto de vista da engenharia de software muitas dessas decisões de design pareçam elegantes, elas abrem várias alternativas de ataque ao sistema ROS [22] [52]. A literatura descreve três principais vetores de ataque específicos ao *middleware* ROS e seus componentes, sendo eles:

- Publicação Não Autorizada

- Acesso Não Autorizado a Dados
- Ataque de Negação de Serviço (DoS) a Nodos ROS

O ataque de publicação não autorizada consiste na injeção de dados falsos em um sistema ROS através da publicação destes em um tópico. Um nodo ROS pode publicar dados em um tópico qualquer sem autorização prévia. Isso pode ser utilizado indevidamente para injetar dados ou comandos em uma aplicação a fim de interferir em sua operação. Como exemplo, um robô pode receber comandos de movimento falsos causando movimentos imprevisíveis que podem machucar pessoas próximas ou danificar equipamentos. Além disso, dados falsos de sensor também podem ser injetados no sistema afim de falsificar um estado normal do sistema ou provocar uma determinada reação do robô.

Já o ataque de acesso não autorizado a dados corresponde ao monitoramento anônimo de um tópico. Qualquer nodo ROS pode se inscrever em qualquer tópico em uma determinada aplicação. Ao se inscrever ele passa a receber todas as mensagens publicadas neste tópico. Deste modo um nodo malicioso pode escutar um tópico visando coletar e extrair informações confidenciais sobre o sistema, interferindo na confidencialidade e privacidade dos dados robóticos. No entanto, não deve ser visível no ROS *graph* que há um *subscriber* adicional presente. Este ataque é especialmente difícil de ser detectado, pois um nodo malicioso pode não ter comunicação ROS de saída.

No ataque de negação de serviço a nodos ROS, um grande número de mensagens falsas é publicado em um tópico. O *subscriber* deste tipo de mensagem será inundado com estes dados. Isso leva a uma alta carga de processamento e potencialmente à incapacidade de realizar processamento significativo. Como não há controle sobre qual nodo pode publicar quais dados, qualquer nodo na rede pode ser usado para publicar dados em um tópico no qual o nodo alvo está inscrito. Isso pode ser usado para um ataque DoS direcionado a esse nodo.

Neste trabalho, é abordado apenas o ataque de injeção de dados não autorizados, apesar de a metodologia demonstrada também poder ser ampliada e aplicada em outros cenários. Detalhes deste ataque e sua metodologia são discutidos a seguir.

4.2 Injeção de Dados Não Autorizados em Sistemas ROS

Em seu trabalho, *Security for the Robot Operating System* [22], o autor denomina este ataque, no contexto de sistemas ROS, de "*Unauthorized Publishing*" (Publicação Não Autorizada), o qual consiste em, por meio da API XML-RPC, utilizar indevidamente os canais de comunicação do ROS a fim de inserir dados falsos na aplicação.

Um invasor, naturalmente, não deseja ser detectado durante o ataque. Desse modo, do ponto de vista dele, as alterações feitas no sistema devem ser difíceis de serem detectadas. Em um sistema ROS, isso significa que o invasor não deve ser visível no ROS

graph, ou seja, qualquer alteração não deve ser representada nele [22]. Conforme visto anteriormente, o ROS *graph* representa as conexões entre nodos construídas via relações de publicação/assinatura, ou seja, se um *subscriber* S assina um tópico T publicado por um *publisher* P, haverá uma conexão entre S e P representada no ROS *graph*. Esse grafo é construído a partir dos nodos conhecidos pelo ROS *master* e de suas conexões, e é facilmente visualizável via comandos específicos do sistema ROS. Partindo desse princípio o ataque utiliza a API XML-RPC para fazer mau uso dos canais de comunicação de modo a injetar dados sem que isso seja representado no ROS *graph*.

A sequência padrão de execução de uma aplicação ROS começa primeiramente com a inicialização do ROS *master*, o que permite aos *publishers* se registrarem no mesmo. Para se registrar os *publishers* invocam o procedimento *registerPublisher* da API XML-RPC no ROS *master*, informando, dentre outros, os nomes dos tópicos que querem publicar e a sua informação de URI, ou seja, seu endereço e identificação. Após a resposta eles estão registrados. Da mesma maneira, os *subscribers* invocam o procedimento *registerSubscriber* no ROS *master*, informando os nomes dos tópicos aos quais querem ter acesso e a sua informação de URI, obtendo assim uma resposta com as informações dos *publishers* de cada tópico requisitado. Assim, através da informação de URI obtida na resposta anterior, os *subscribers* invocam o procedimento *requestTopic* diretamente aos publishers, juntamente com uma lista de protocolos desejados para efetivar a conexão, os quais respondem com as informações necessárias para assim o canal de comunicação ponto a ponto poder ser efetivado pelo *subscriber*, dando início à troca de mensagens. Uma visualização deste processo pode ser vista na Figura 4.

Conhecendo o fluxo de dados, o atacante pode desenvolver a própria sequência de chamadas de procedimentos, fazendo mau uso da API e dos canais de comunicação para alterar o fluxo e possibilitar o ataque. O objetivo é isolar um *subscriber* de seus *publishers* sem que ele, os *publishers* e o ROS *master* percebam isso. O autor de [22] ressalta que a metodologia do ataque descrito em seu trabalho não faz uso de nenhuma ferramenta do ROS (como o *rostopic*) e, portanto, não necessita de acesso a um nodo ROS nem executa alterações visíveis no ROS *graph*.

Para este trabalho, é assumido que o atacante já violou a segurança da rede e que já tem conhecimento do ROS *graph* e, portanto, já sabe as informações de tópicos e as URIs dos componentes do sistema. Sendo assim, conforme descreve o autor de [22], o atacante pode então invocar o procedimento *publisherUpdate* diretamente no *subscriber*. Através deste procedimento é possível alterar as informações de *publishers* conhecidas pelo *subscriber*, informando uma nova lista de *publishers* de um tópico específico por exemplo, permitindo assim isolar o mesmo de seus *publishers* verdadeiros. Nessa nova lista estará o *publisher* falso criado pelo atacante, o qual estará injetando as mensagens falsas no tópico. A figura 5 demonstra detalhadamente o fluxo de chamadas de procedimentos XML-RPC durante este ataque, onde em vermelho se observam os pontos mais críticos

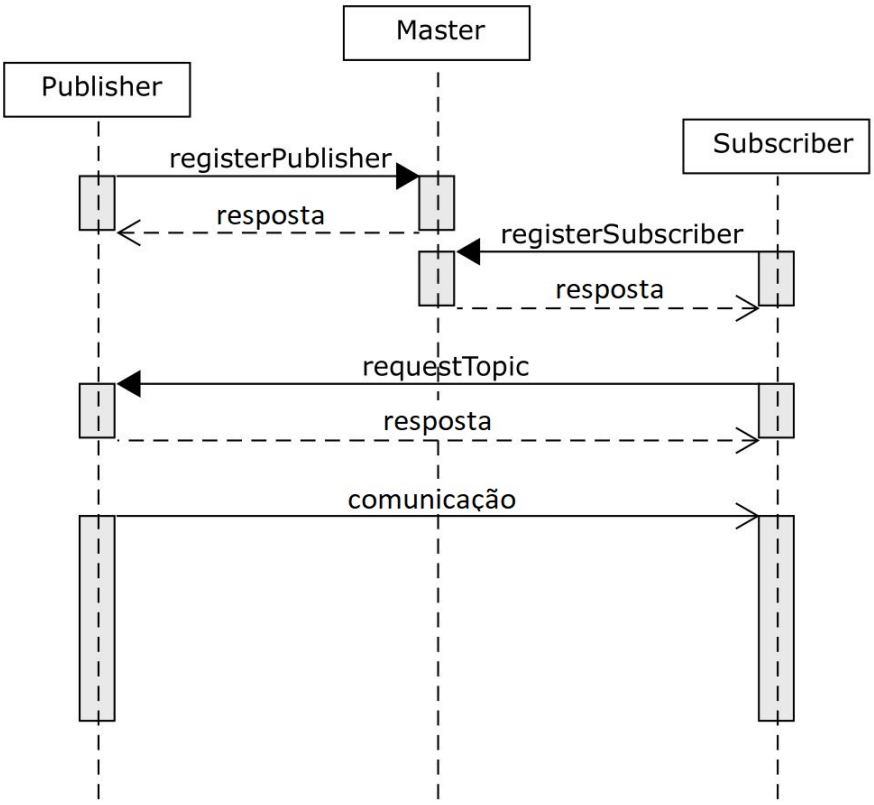

Figura 4: Sequência de chamadas de procedimentos XML-RPC durante a execução de uma aplicação ROS (adaptado de [22]).

em que acontece o isolamento do *subscriber*. É importante destacar que os *publishers* originais continuam ativos e visíveis no ROS *graph*, como se nada tivesse acontecido, porém suas mensagens não irão mais alcançar o *subscriber*, além disso, percebe-se que o ataque é facilmente reversível, bastando ao atacante invocar um novo procedimento *publisherUpdate* informando a lista original de *publishers*.

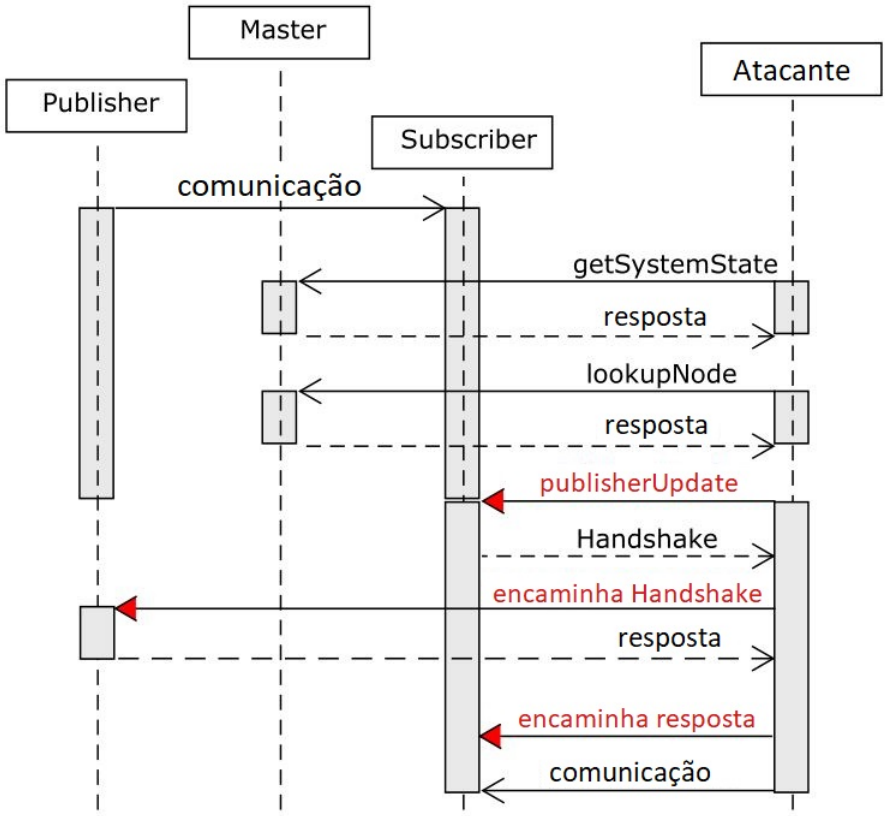

Figura 5: Sequência de chamadas de procedimentos XML-RPC durante um ataque de injeção de dados não autorizados em uma aplicação ROS (adaptado de [22]).

4.3 Método Proposto

Neste trabalho, é proposto um método empregando técnicas de detecção de anomalias. A detecção de anomalias é uma área de estudo cada vez mais importante no campo da robótica, uma vez que os sistemas robóticos tendem a níveis mais elevados de autonomia. Ser capaz de prever, identificar e corrigir essas anomalias é fundamental, especialmente quando os sistemas robóticos podem ter um impacto direto ou indireto na vida humana [34]. Este tipo de abordagem tem se mostrado bastante eficiente quando aplicado em sistemas ciber-físicos em geral [54], [70], sendo possível também sua aplicação em sistemas ROS. A Figura 6 ilustra uma visão arquitetural do método, cujos detalhes serão apresentados logo a seguir.

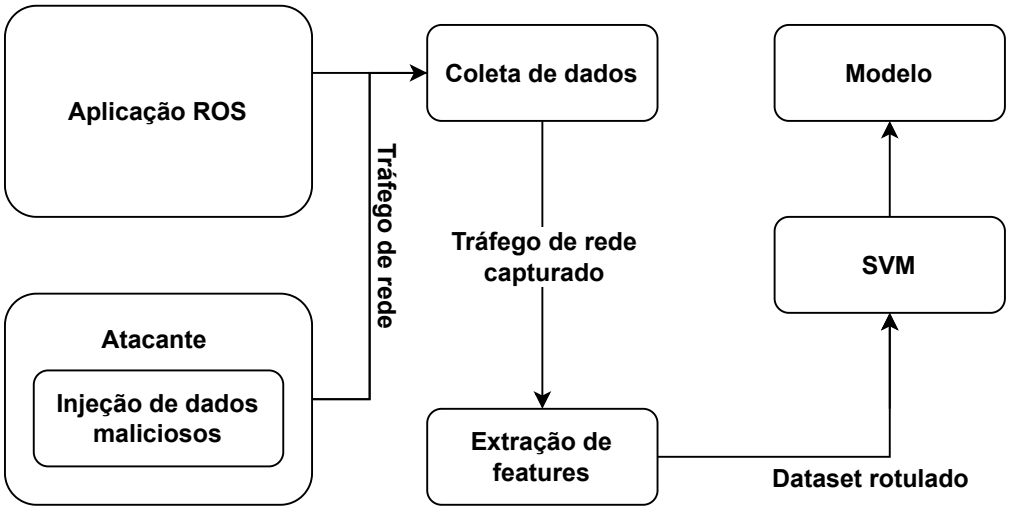

Figura 6: Método para Detecção de Ataques de Injeção de Dados Não Autorizados em Sistemas ROS.

4.3.1 Abordagem

A abordagem utilizada consiste no reconhecimento dos ataques através da detecção de anomalias no tráfego de rede de uma aplicação ROS. Sistemas de detecção de intrusão baseados em rede (NIDSs - do inglês "*Network-based Intrusion Detection Systems*") detectam ataques capturando e analisando pacotes de rede. Escutando em um segmento de rede ou *switch*, um NIDS pode monitorar o tráfego de rede de vários *hosts* conectados nele, protegendo assim esses *hosts* [6]. Esta abordagem é bastante conveniente no contexto de sistemas ROS, uma vez que estes executam sobre uma rede TCP/UDP. Além disso ela oferece mais neutralidade na detecção dos ataques, uma vez que analisar características da rede ao invés de características de um tipo de dado específico torna a abordagem mais abrangente e menos dependente. Outra vantagem é que a implantação de IDSs baseados em rede tem pouco impacto sobre uma rede existente. Eles geralmente são dispositivos passivos que monitoram e escutam sem interferir com a operação normal da mesma [6]. Isso é especialmente interessante no âmbito de sistemas robóticos, uma vez que, além de dispensar a necessidade de alterações no código-fonte do sistema ou da aplicação, também não interfere na execução nem no desempenho da aplicação como nas abordagens apresentadas anteriormente.

4.3.2 Técnica Empregada

Como técnica, foi utilizado o aprendizado de máquina aplicado em detecção de anomalia, mais especificamente o modelo de aprendizado supervisionado conhecido como "Máquina de Vetores de Suporte" (SVM). Ao contrário dos sistemas e redes de computação tradicionais, os sistemas robóticos tendem a exibir padrões e comportamentos bem definidos [57]. Trabalhos anteriores mostraram que soluções baseadas em técnicas de aprendizado de máquina são eficazes nestes cenários, tais como [2], [73]

e [49]. Além disso, em seu trabalho *Anomaly Detection in Cyber-Physical Systems Using Machine Learning* [55], o autor demonstrou, por meio de um estudo de caso com redes elétricas inteligentes (*Smart Grids*), a eficácia desse tipo de técnica na classificação de ataques de injeção de dados em sistemas ciber-físicos, onde o SVM obteve acurácia e *F1-score* significativamente melhores comparado a outros algoritmos de aprendizado de máquina.

O modelo SVM é normalmente utilizado em problemas de classificação, sendo que, neste trabalho, foi empregado como um classificador binário, ou seja, para identificar se determinada instância de um conjunto de dados (*dataset*) representa uma anomalia ou não. Uma tarefa de classificação envolve a separação dos dados em conjuntos de treinamento e conjuntos de teste. Cada instância no conjunto de treinamento contém um valor alvo (os rótulos das classes) e vários atributos ou características (*features*). O objetivo do SVM é produzir um modelo, com base nos dados de treinamento, que preveja os valores alvo dos dados de teste tendo apenas as características dos dados de teste [38].

A fim de se obter melhores resultados com sua utilização, Chang *et al.* recomenda a aplicação do método apresentado em *A Practical Guide to Support Vector Classification* [38], onde são descritas as seguintes etapas:

- Transformar os dados para o formato de um pacote SVM
- Diminuir a escala dos dados
- Considerar o uso do *kernel* RBF
- Utilizar validação cruzada para encontrar os melhores parâmetros C e γ
- Utilizar os melhores parâmetros C e γ para treinar o modelo SVM
- Testar o modelo

O método proposto adota todas essas etapas, utilizando o *kernel* RBF juntamente com as técnicas de *grid-search* e validação cruzada para encontrar os melhores parâmetros C e γ.

4.3.3 Extração e Processamento de *Features*

O método proposto utiliza como treinamento dados de rede de uma aplicação ROS contendo o comportamento esperado e o anômalo. Esses dados são processados e suas características extraídas para então serem utilizadas no treinamento do algoritmo SVM, o qual gera um modelo de previsão para reconhecer os ataques.

Diferentemente dos trabalhos de Narayanan and Bobba [57], Amrouche *et al.* [3] e Lagraa *et al.* [46], onde o foco é analisar o conteúdo das mensagens ROS (estado de

juntas ou imagens de câmera) em busca de alguma anomalia, o objetivo do modelo proposto é ser mais abrangente, ou seja, capaz de identificar anomalias independente do tipo de dado abordado pelo ataque, sejam dados de sensores laser, imagens de câmera, dados de movimentação, estados de juntas, etc. Deste modo, para tornar a abordagem o mais abrangente possível foi necessário selecionar características que sejam comuns aos mais variados tipos de dados, por isso optou-se por características do tráfego de rede da aplicação. O trabalho *One-class Support Vector Machine for Anomaly Network Traffic Detection* [68] sugere algumas características para aplicações de detecção de anomalias no tráfego de rede. Neste trabalho, foram utilizadas o conjunto de sete, conforme a Tabela 2.

Tabela 2: Lista de *features* utilizadas na elaboração dos *datasets*.

Índice	Feature
1	Número de bits por segundo
2	Número de pacotes por segundo
3	Número de pacotes TCP por segundo
4	Tamanho médio de pacote em bytes
5	Desvio padrão do tamanho de cada pacote em bytes
6	Utilização da rede durante a última janela observada
7	Tamanho máximo de pacote em bytes

Para a extração das *features*, o método utiliza o software Tcpstat [36]. O Tcpstat é uma ferramenta de monitoramento de rede que monitora e reporta estatísticas de interfaces de rede. Ele obtém suas informações monitorando uma interface específica ou lendo os dados de um arquivo de saída do Tcpdump. Ele foi escrito com desempenho e eficiência em mente sendo capaz de lidar com grandes quantidades de pacotes por segundo [68]. As estatísticas reportadas pelo Tcpstat se configuram nas *features* do tráfego de rede necessárias ao algoritmo SVM. Ele calcula vinte tipos diferentes de estatísticas, sendo que, neste trabalho, além das apresentadas na Tabela 2, também foi utilizada a estatística de *timestamp* para fins de rotulação dos dados, caso necessário. O uso do Tcpstat neste trabalho é bastante conveniente, uma vez que sua saída é em formato de vetor de linhas de texto, exatamente o formato de entrada necessário ao algoritmo SVM, restando apenas a devida rotulação dos dados.

A rotulação consiste em especificar a qual classe pertence cada amostra de um conjunto de dados, no *dataset* de treinamento esses rótulos são utilizados para ensinar o modelo SVM quais as características de cada classe, já no *dataset* de teste servem para

avaliar o desempenho do modelo gerado pelo algoritmo. O método proposto considera dados previamente rotulados, sendo necessária somente uma adequação ao padrão requerido pelo mesmo. Conforme mencionado anteriormente, o SVM foi empregado como um classificador binário, isso significa que foram utilizadas apenas duas classes, uma para representar a presença de anomalia (ataque), identificada com o número “1”, e outra para representar a ausência de anomalia, identificada com o número “0”.

5 AVALIAÇÃO

A fim de avaliar a viabilidade técnica do método proposto, foi modelado um cenário para a coleta de dados e a construção de *datasets*. Isso foi necessário pois os *datasets* ROS encontrados na literatura não atendem os requisitos do método proposto, ou seja, não contêm informações de rede de aplicações ROS, aspecto importante da abordagem utilizada. Este cenário, bem como a construção dos *datasets* e os resultados dos experimentos, serão apresentados em detalhes neste capítulo.

5.1 Modelagem do Cenário

Para trazer validade aos experimentos, é interessante abordar um cenário em que o robô execute tarefas úteis a uma aplicação real, diante disso decidiu-se por um cenário de localização e mapeamento simultâneos (SLAM [26]). A localização e o mapeamento simultâneos é um problema desafiador que tem despertado o interesse de cada vez mais pesquisadores na última década [74]. A auto-localização de robôs móveis é uma questão fundamental na navegação autônoma: o robô móvel deve ser capaz de estimar sua posição e orientação (*pose*) em um mapa do ambiente em que está navegando. No entanto, em muitas aplicações de relevância prática (como tarefas de exploração ou operações em ambientes hostis), um mapa não está disponível ou é altamente incerto. Portanto, em tais casos o robô deve utilizar as medidas fornecidas por seu equipamento sensorial para estimar um mapa do ambiente e, ao mesmo tempo, localizar-se dentro do mapa [31].

Os experimentos foram conduzidos em ambiente simulado, no simulador de robótica em 3D conhecido como Gazebo [37]. O Gazebo é uma ferramenta totalmente em código aberto e disponível gratuitamente, tem uma base ativa de colaboradores que o mantém em constante evolução. É uma ferramenta eficaz, escalonável e simples que expande o campo de pesquisa em robótica para uma comunidade mais ampla, demandando menos recursos, além de oferecer um ambiente rico para o desenvolvimento e teste de sistemas robóticos das mais diversas maneiras [43]. O Gazebo e o ROS são uma poderosa combinação, sendo amplamente utilizada pela comunidade de robótica em geral pela sua capacidade de simular ambientes e de realizar cálculos de física com credibilidade [65].

O ambiente consistiu em três máquinas virtualizadas e independentes entre si, cada uma executando como base o sistema operacional Linux Ubuntu Focal Fossa 20.04 LTS [66] e o *middleware* ROS Noetic Ninjemys, todas na mesma rede local. Na primeira máquina foi iniciado um mundo virtual no simulador Gazebo (versão 11.9.0) no qual foi instanciado um robô móvel Turtlebot3 Burger [29] equipado com um sensor laser (*lidar*) para desvio de obstáculos e mapeamento (Figura 7). Na Figura 8 é apresentada a interface do simulador, onde é possível visualizar o mundo em questão juntamente com o robô. Destaque para as linhas em azul que representam o alcance das várias amostras do *lidar* e para os objetos brancos que representam os obstáculos. Já na Figura 9, é possível visualizar o processo de mapeamento sendo executado nesse mesmo mundo virtual. A área em cinza claro representa a área já mapeada.

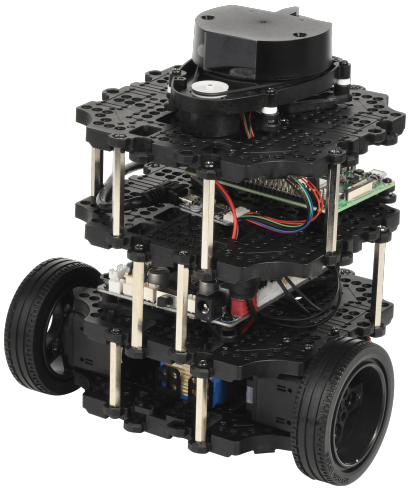

Figura 7: Robô móvel Turtlebot3 Burger [29].

Na segunda máquina virtual foi iniciada a aplicação de controle do robô, responsável pelo deslocamento, desvio de obstáculos, mapeamento, dentre outros. Já na terceira máquina, foi inicializado um ambiente ROS à parte, onde foi implementado o conjunto de ferramentas utilizadas na simulação dos ataques.

O ROS *graph* do cenário estudado pode ser visto na Figura 10, onde as elipses correspondem aos nodos e os retângulos aos tópicos. O robô Turtlebot3 abrange o ROS *master* e o nodo *publisher/subscriber* denominado "/gazebo". O *publisher* do nodo "/gazebo" publica, dentre outros, as informações do sensor laser e da odometria nos tópicos "/scan" e "/odom", respectivamente, os quais são acessados pelos nodos "/turtlebot3_slam_gmapping" e "/turtlebot3_drive" da aplicação de controle a fim de realizar a localização e o mapeamento e decidir a movimentação. Já o *subscriber* do nodo "/gazebo" lê do tópico "/cmd_vel" as mensagens de movimentação publicadas pelo nodo "/turtlebot3_drive" pertencente à aplicação de controle.

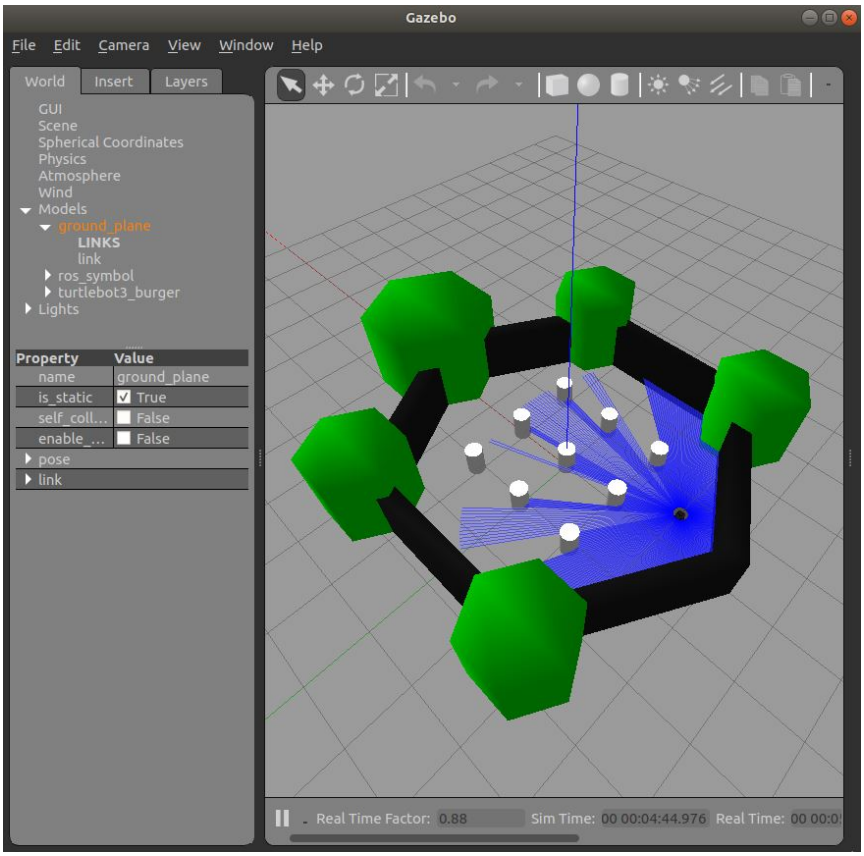

Figura 8: Interface do simulador Gazebo e mundo virtual inicializado.

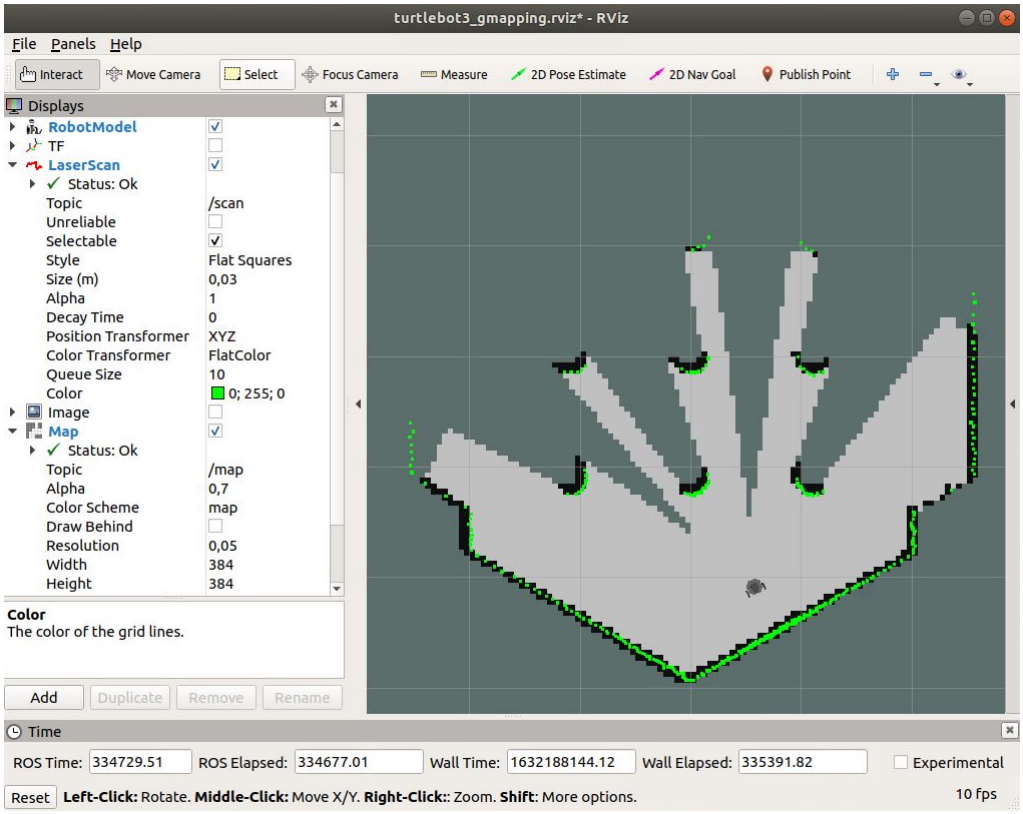

Figura 9: Robô executando o mapeamento no mundo virtual.

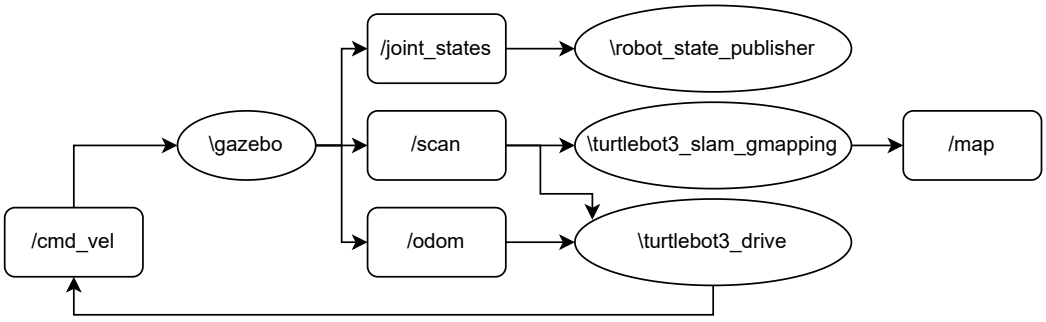

Figura 10: ROS *graph* correspondente ao cenário dos experimentos.

5.2 Modelagem das Anomalias

O método proposto utiliza o comportamento esperado e o anômalo para o treinamento do modelo SVM, logo, além do cenário esperado, foi necessário modelar as anomalias a serem injetadas. Importante observar que, para este trabalho, é assumido que o objetivo do atacante é tornar o robô inseguro sem que isso seja percebido de imediato, ou seja, são ataques mais discretos e difíceis de ser identificados, mas que ainda assim perturbam a confiabilidade e segurança do sistema. Ataques mais agressivos e destrutivos costumam causar efeito físico imediato, tal como o *worm* Stuxnet no Irã [15]. Esse tipo de cenário não se enquadra no contexto deste trabalho, uma vez que as consequências deste tipo de ataque acontecem no ato, não fazendo sentido o uso de técnicas de detecção [57].

Para esta pesquisa optou-se por realizar os experimentos com três variantes do ataque descrito em 4.2. A primeira delas consiste em apenas isolar o *subscriber* alvo, nesse caso, o *subscriber* "/gazebo", de modo que o mesmo não receba as mensagens de movimentação publicadas pelo nodo "/turtlebot3_drive", o que, por si só, já causa mal funcionamento no robô. Para tal, utilizando a metodologia descrita em 4.2, é invocado o procedimento *publisherUpdate* no *subscriber* "/gazebo" informando-se uma lista nula de *publishers* para o tópico "/cmd_vel", assim o nodo "/gazebo", mesmo estando inscrito no tópico, não receberá nenhuma mensagem. Na Figura 11, é possível visualizar um diagrama de como fica o cenário modelado durante este ataque, percebe-se que o *subscriber* "/gazebo" não lê as mensagens publicadas no tópico "/cmd_vel",

A segunda variante consiste em injetar mensagens de movimento incorretas no tópico "/cmd_vel", paralelamente ao nodo verdadeiro ("/turtlebot3_drive"), fazendo com que o robô assuma um comportamento imprevisível, podendo não desviar de um obstáculo quando deveria por exemplo, tornando-o suscetível a colisões. Para tal, é invocado o procedimento *publisherUpdate* no *subscriber* "/gazebo" informando-se uma nova lista de *publishers* para o tópico "/cmd_vel' que inclui o nodo original ("/turtlebot3_drive") e um nodo falso. Esse nodo falso, denominado "/turtlebot3_falso", é hospedado no sistema ROS da terceira máquina virtual e publica mensagens de movimento no tópico "/cmd_vel" na mesma frequência do nodo verdadeiro (100hz).

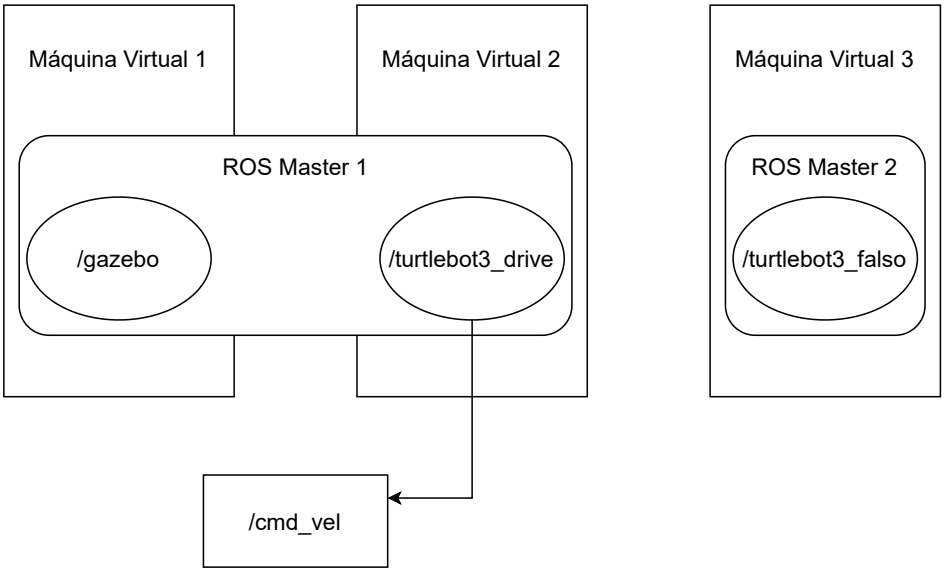

Figura 11: Cenário durante ataque de isolamento de *subscriber*.

Já na terceira variante, foi empregada a mesma metodologia acima, mas desta vez utilizando um nodo falso que publica mensagens de movimento no tópico "/cmd_vel" numa frequência de 10hz. Trata-se de um ataque bem mais sutil e difícil de ser detectado, mas que ainda sim pode oferecer riscos. Na Figura 12, é possível visualizar um diagrama do cenário durante esse ataque.

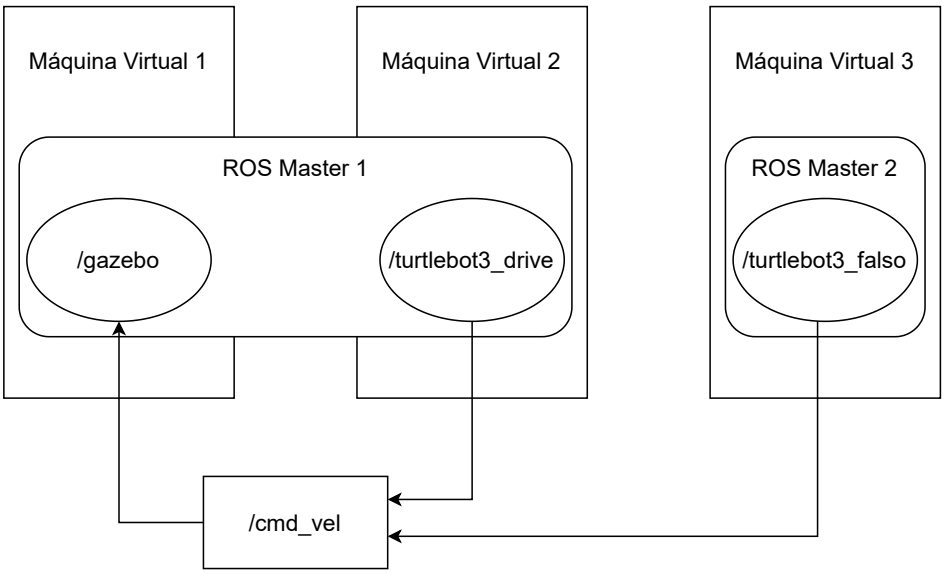

Figura 12: Cenário durante ataque de injeção de dados não autorizados.

A fim de gerar as anomalias, foi desenvolvido e implementado um conjunto de ferramentas capazes de realizar esta tarefa de forma automatizada. Primeiramente, foi implementado o RosPenTo [22], uma ferramenta para execução de testes de intrusão desenvolvida especificamente para o ROS. Ele explora as inconsistências na API XML-RPC do ROS empregando a metodologia descrita na seção 4.2, possibilitando a execução de ata-

ques de injeção de dados não autorizados e o isolamento de *publishers* e *subscribers*. Em seguida foi desenvolvido um programa que, em conjunto com a ferramenta RosPenTo, possibilita gerar as anomalias de forma automática e customizável. O mesmo é adaptável ao tamanho em minutos do *dataset*, podendo ser utilizado com *datasets* de curta ou longa duração. Também permite gerar os ataques com ordem, duração e intervalo aleatórios, além de possibilitar a definição, através de parâmetros, da duração mínima e máxima de cada ataque e de cada intervalo. O programa também faz um registo das *timestamps* de cada ataque em formato Unix, de algumas características de cada ataque (tipo, duração, detalhes internos) e do total de ataques realizados durante sua execução.

5.3 Construção dos Datasets

Os *datasets* foram construídos a partir do tráfego de rede da aplicação ROS, o qual foi capturado por meio do software conhecido como Tcpdump [39]. O Tcpdump é um analisador de pacotes de rede executado via linha de comando. Ele permite ao usuário interceptar e exibir pacotes TCP/IP e outros sendo transmitidos ou recebidos em uma rede à qual o computador está conectado. Distribuído sob a licença BSD, o Tcpdump é um software livre [4]. As *features* foram então extraídas com o Tcpstat, o qual gerou um arquivo no formato de vetor de linhas de texto. Conforme mencionado em 4.3.3, o método considera dados previamente rotulados, portanto foi necessário realizar a rotulação dos dados extraídos, já no padrão requerido pelo método.

Para efetivar a rotulação, foi desenvolvido um programa que lê o arquivo não rotulado criado pelo Tcpstat e compara as *timestamps* de cada amostra com o registro de *timestamps* de ataques proporcionado pelo programa que gera as anomalias, marcando então as linhas que contém ataques com “1” e as que não contém com “0”. Uma amostra de um dos *datasets*, devidamente rotulado, pode ser visualizada na Figura 13, na primeira coluna estão os rótulos das classes e nas outras estão as *features* acompanhadas de seus respectivos índices, na mesma ordem apresentada na Tabela 4.3.3.

5.4 Implementação do Modelo SVM

O modelo SVM foi implementado utilizando a biblioteca LIBSVM (versão 3.3), uma das implementações do algoritmo de máquinas de vetores de suporte mais utilizadas atualmente [13]. Ela foi desenvolvida por Chang *et al.* [13] e tem por objetivo promover o SVM como uma ferramenta conveniente e de fácil utilização, além de apresentar resultados promissores quando empregada em detecção de anomalias [47]. A biblioteca LIBSVM inclui diversas ferramentas que possibilitam a aplicação do método descrito em 4.3.2 de forma simples e eficaz.

```
0 1:4416611.20 2:8218.60 3:41048 4:67.17 5:135.20 6:0.95 7:2997
0 1:4405539.20 2:8199.20 3:40951 4:67.16 5:135.29 6:0.94 7:2997
0 1:4408438.40 2:8199.60 3:40953 4:67.21 5:135.45 6:0.94 7:2997
0 1:4410560.00 2:8207.20 3:40991 4:67.18 5:135.25 6:0.94 7:2997
0 1:4402168.00 2:8187.80 3:40894 4:67.21 5:135.41 6:0.94 7:2997
0 1:4409702.40 2:8199.40 3:40952 4:67.23 5:135.50 6:0.94 7:2997
1 1:4409574.40 2:8204.60 3:40978 4:67.18 5:135.27 6:0.94 7:2997
1 1:4411886.40 2:8208.80 3:40999 4:67.18 5:135.37 6:0.94 7:2997
1 1:4404011.20 2:8195.40 3:40932 4:67.17 5:135.33 6:0.94 7:2997
1 1:4654800.00 2:8384.00 3:41875 4:69.40 5:150.90 6:0.94 7:7292
1 1:4507555.20 2:8360.40 3:41757 4:67.39 5:134.20 6:0.94 7:2997
1 1:4516065.60 2:8377.80 3:41844 4:67.38 5:133.97 6:0.94 7:2997
1 1:4490432.00 2:8337.00 3:41640 4:67.33 5:134.24 6:0.94 7:2997
1 1:4510299.20 2:8387.20 3:41891 4:67.22 5:131.72 6:0.94 7:2997
1 1:4520331.20 2:8383.60 3:41873 4:67.40 5:133.88 6:0.94 7:2997
1 1:4522032.00 2:8380.00 3:41855 4:67.45 5:134.05 6:0.94 7:2997
1 1:4543572.80 2:8422.00 3:42065 4:67.44 5:133.55 6:0.94 7:2997
1 1:4515443.20 2:8371.00 3:41810 4:67.43 5:134.16 6:0.94 7:2997
0 1:4543120.00 2:8424.40 3:42077 4:67.41 5:133.52 6:0.94 7:2997
0 1:4520081.60 2:8378.20 3:41846 4:67.44 5:134.11 6:0.95 7:2997
0 1:4539516.80 2:8407.60 3:41993 4:67.49 5:133.84 6:0.95 7:2997
0 1:4500320.00 2:8346.60 3:41688 4:67.40 5:134.17 6:0.95 7:2997
0 1:4492619.20 2:8335.40 3:41632 4:67.37 5:134.26 6:0.95 7:2997
0 1:4535920.00 2:8406.00 3:41985 4:67.45 5:133.85 6:0.95 7:2997
```

Figura 13: Amostra de *dataset* rotulado.

5.5 Resultados Experimentais

A fim de avaliar a eficiência do método em diferentes situações, foram realizados experimentos com múltiplos *datasets*, variando o tempo de captura, janela de tempo entre amostras, percentual de anomalia injetada e características das anomalias. Uma lista com todos os *datasets* e suas respectivas particularidades pode ser visualizada na Tabela 3, onde, na primeira coluna, estão seus rótulos, que representam o número do experimento seguido da finalidade do *dataset* (“A” para treinamento e “B” para teste).

Tabela 3: Lista de *datasets* utilizados nos experimentos.

Dataset	Duração	Janela	Total de amostras	Amostras com anomalia	Percentual de anomalia	Duração de cada ataque
Exp1-A	24h	30s	2880	600	20,83%	1-2 min
Exp1-B	24h	30s	2880	322	11,18%	1-2 min
Exp2-A	24h	30s	2880	581	20,17%	1-2 min
Exp2-B	24h	30s	2880	312	10,83%	1-2 min

Os *datasets* Exp1-A e Exp1-B foram construídos tendo como anomalias a primeira e a segunda variante do ataque de injeção de dados não autorizados, conforme seção 5.2, sendo capturados por 24 horas com janela de 30 segundos, tanto para o treinamento quanto para o teste. A duração dos ataques variou entre 1 e 2 minutos e o percentual de

anomalia injetada foi de 20,83% para o treinamento e 11,18% para o teste. Nas Figuras 14, 15 e 16, é possível visualizar um trecho destes *datasets*, onde é demostrado o comportamento do tráfego de rede com relação à média de pacotes por segundo, taxa média de transferência e tamanho médio de pacote, respectivamente. Em vermelho é representado o momento em que ocorrem as anomalias. Percebe-se que há uma alteração significativa no tráfego normal da rede.

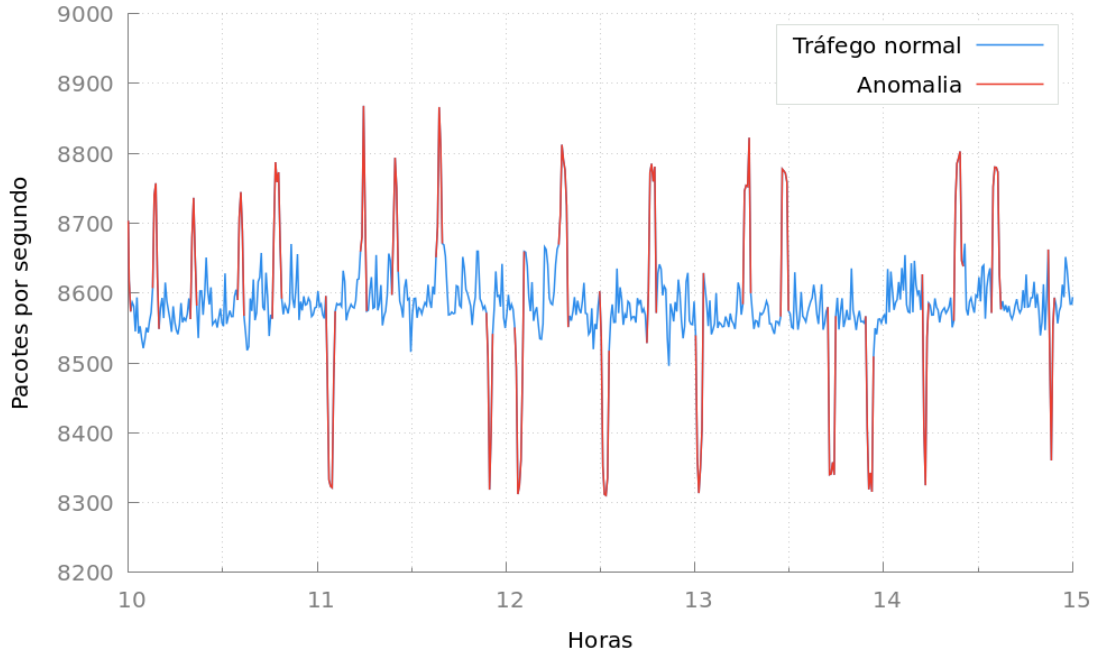

Figura 14: Média de pacotes por segundo (Exp1-A e Exp1-B).

Já os *datasets* Exp2-A e Exp2-B foram construídos tendo como anomalia apenas a terceira variante do ataque de injeção de dados não autorizados, sendo capturados por 24 horas com janela de 30 segundos, tanto para o treinamento quanto para o teste. A duração dos ataques variou entre 1 e 2 minutos e o percentual de anomalia injetada foi de 20,17% para o treinamento e 10,83% para o teste. Nas Figuras 17, 18 e 19, é possível visualizar um trecho destes *datasets*, onde é demostrado o comportamento do tráfego de rede com relação à média de pacotes por segundo, taxa média de transferência e tamanho médio de pacote, respectivamente. Percebe-se que, neste caso, as anomalias são bem mais discretas, pouco alterando o tráfego de rede.

5.5.1 Métricas Avaliadas

Em todos os experimentos, a eficiência do método foi avaliada com relação a métricas amplamente utilizadas na avaliação de modelos de aprendizado de máquina, as quais foram obtidas por meio da matriz de confusão. A matriz de confusão é uma tabela que fornece alguns dados sobre o desempenho do modelo, a partir dos quais é possível obter diversas métricas úteis à avaliação, sendo avaliadas, neste trabalho, a acurácia, precisão,

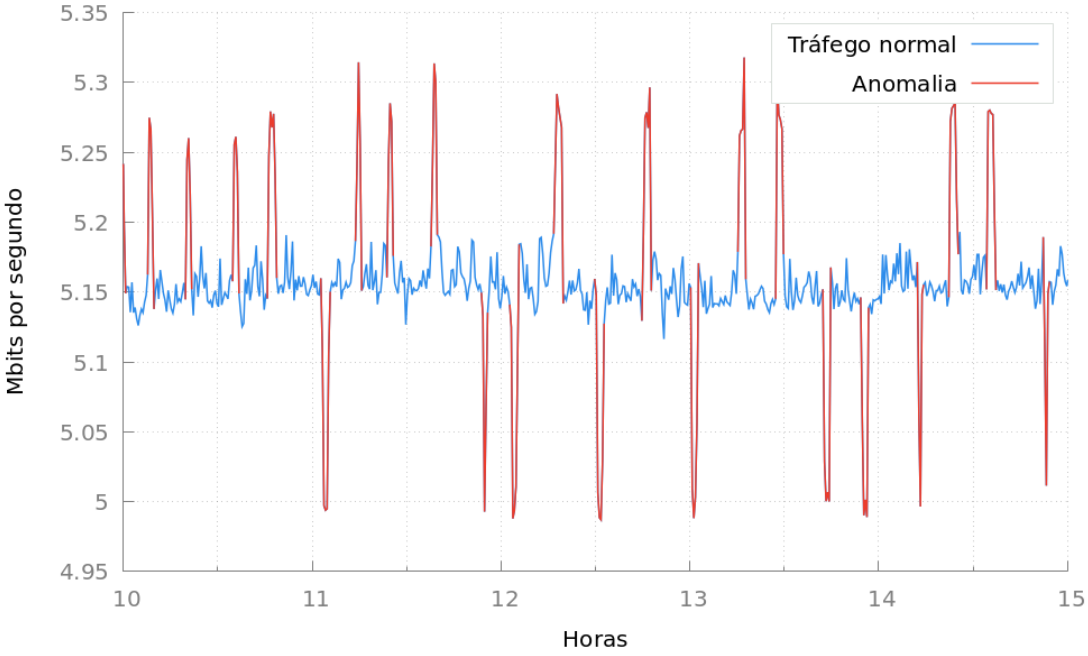

Figura 15: Taxa média de transferência (Exp1-A e Exp1-B).

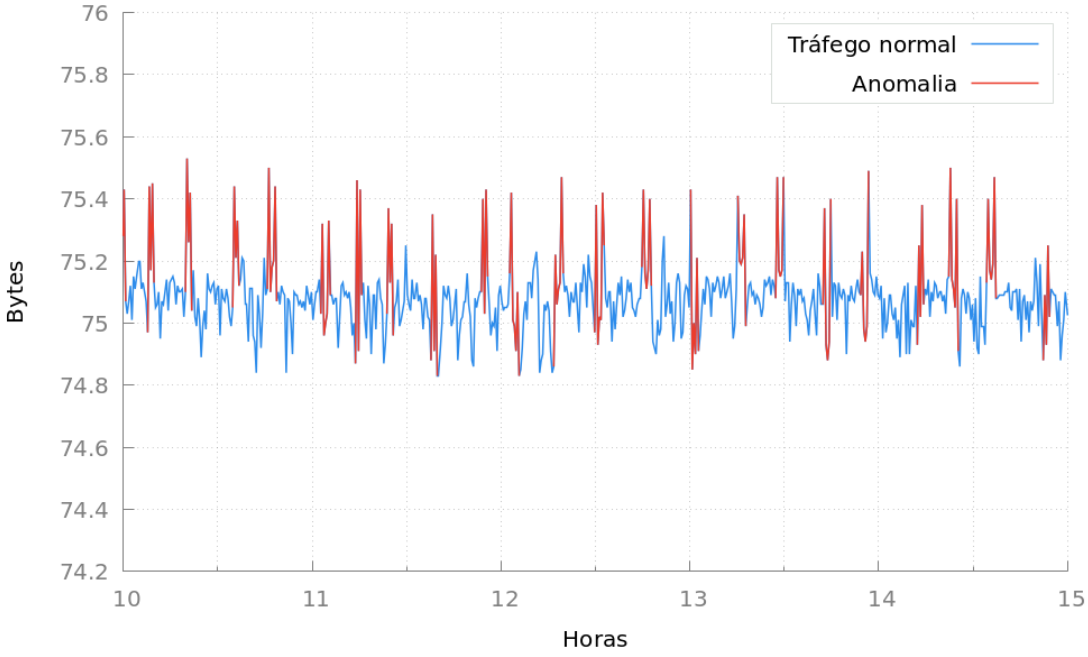

Figura 16: Tamanho médio de pacote (Exp1-A e Exp1-B).

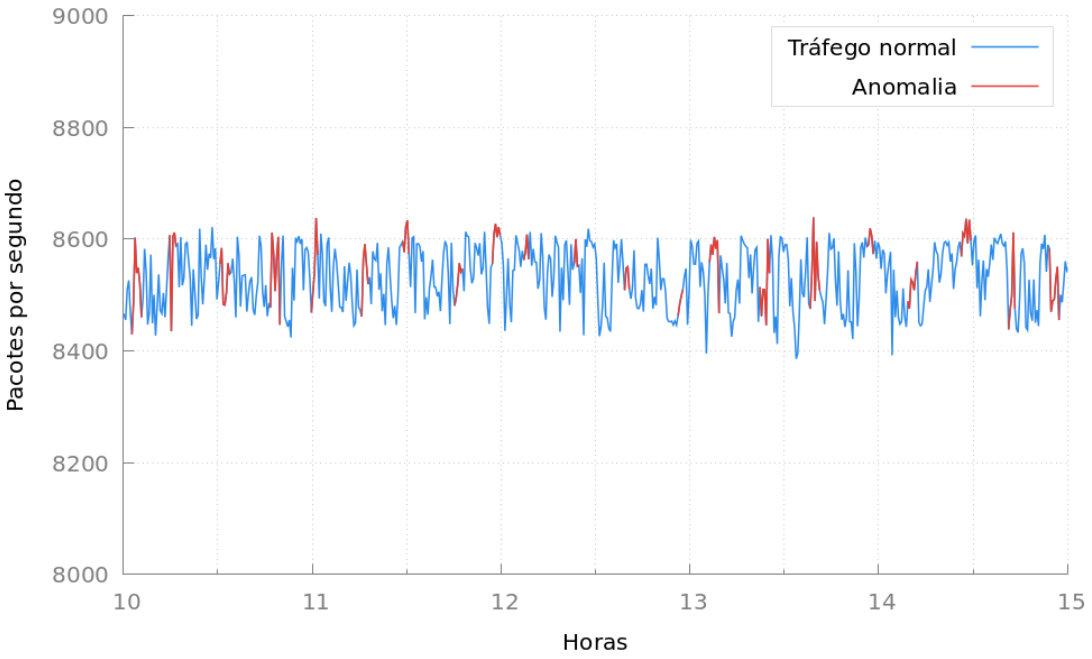

Figura 17: Média de pacotes por segundo (Exp2-A e Exp2-B).

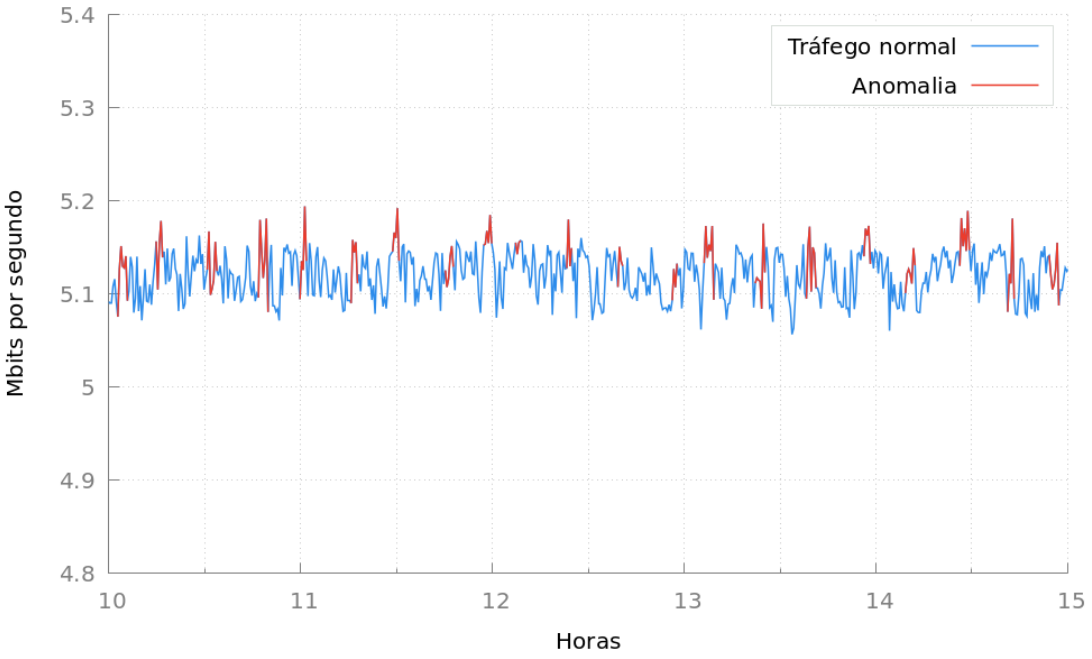

Figura 18: Taxa média de transferência (Exp2-A e Exp2-B).

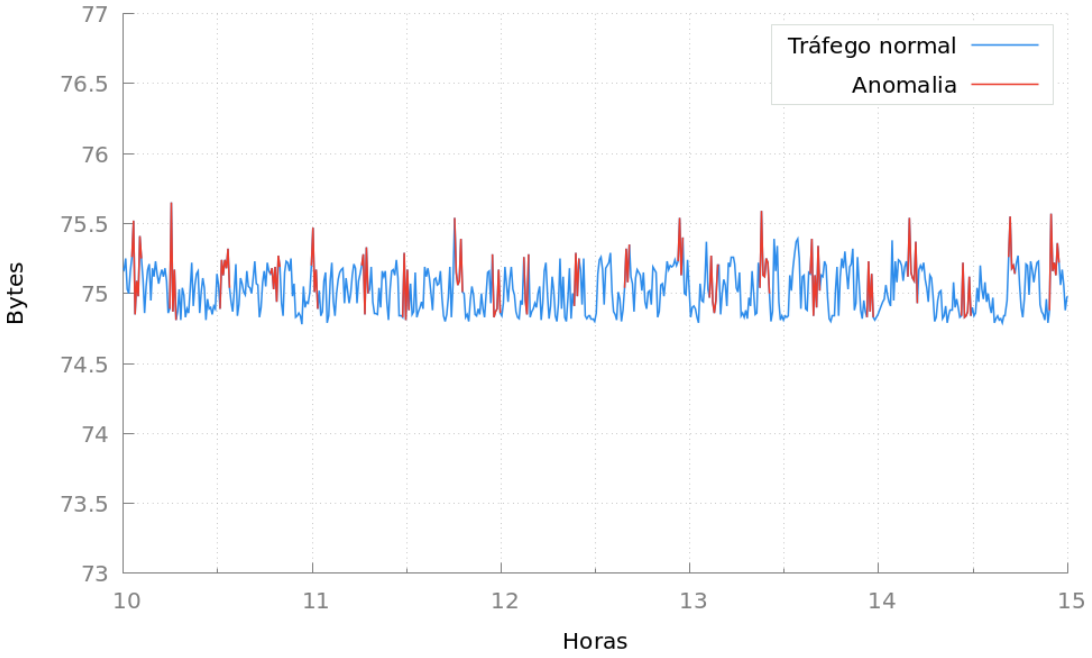

Figura 19: Tamanho médio de pacote (Exp2-A e Exp2-B).

taxa de verdadeiros positivos (também conhecida como sensibilidade ou *recall*), taxa de verdadeiros negativos (também conhecida como especificidade) e *F1-score*.

A acurácia avalia o percentual total de acertos, ou seja, ela pode ser obtida pela razão entre a quantidade de acertos e o total de entradas, de acordo com a fórmula (1). Já a precisão, é uma métrica que avalia a proporção de verdadeiros positivos com relação à soma de todos os valores detectados como positivos, conforme a fórmula (2), sendo utilizada para avaliar a qualidade da previsão positiva do método. A taxa de verdadeiros positivos (*recall*) avalia a proporção de verdadeiros positivos com relação à soma de todos os valores realmente positivos, ou seja, avalia a capacidade do método de detectar resultados positivos corretamente, sendo dada pela fórmula (3). Da mesma maneira, a taxa de verdadeiros negativos (especificidade) avalia a capacidade do método de detectar resultados negativos corretamente, sendo dada pela fórmula (4). O *F1-score*, por sua vez, é um balanço entre a precisão e o *recall*, de modo a fornecer um número que determine a qualidade geral do modelo, conforme a fórmula (5).

$$Acurácia = \frac{VP + VN}{VP + FN + VN + FP} \tag{1}$$

$$Precisão = \frac{VP}{VP + FP} \tag{2}$$

$$Recall = \frac{VP}{VP + FN} \tag{3}$$

$$Especificidade = \frac{VN}{VN + FP} \tag{4}$$

$$F1 = 2 * \frac{Precisão * Recall}{Precisão + Recall} \tag{5}$$

$$TFP = 1 - TVN \tag{6}$$

5.5.2 Experimentos

Foram realizados três experimentos, nos dois primeiros o modelo SVM foi avaliado quanto à detecção de diferentes anomalias. No terceiro experimento, o desempenho do modelo foi comparado ao de mais três modelos de aprendizado, sendo eles *Random Forest*, *Naive Bayes* e *Multilayer Perceptron*.

5.5.2.1 Experimento 1

O primeiro experimento foi realizado utilizando os *datasets* Exp1-A e Exp1-B, onde foram injetadas a primeira e a segunda variante do ataque de injeção de dados não autorizados. Conforme descrito em 4.3.2, foi utilizado o *kernel* RBF juntamente com as técnicas de *grid-search* e validação cruzada para encontrar os melhores parâmetros *C* e γ, sendo estes 32768,0 e 0,03125, respectivamente. Aplicando o método, obteve-se a matriz de confusão, conforme Tabela 4, a partir da qual foram extraídas as métricas apresentadas na Tabela 5. Percebe-se que o método teve um desempenho excelente em todas as métricas avaliadas, atingindo 99,79% de acerto no geral (acurácia), 99,69% de acerto na detecção de anomalias (*recall*) e 99,80% na detecção de tráfego normal (especificidade), o que indica uma baixa complexidade das anomalias avaliadas neste experimento.

Tabela 4: Matriz de confusão (Experimento 1).

		Classe prevista	
		0	1
Classe esperada	0	VN: 2553	FP: 5
	1	FN: 1	VP: 321

5.5.2.2 Experimento 2

Uma vez que, para este trabalho, é considerado que o objetivo do atacante é executar ataques mais discretos e difíceis de serem detectados, o segundo experimento foi realizado utilizando os *datasets* Exp2-A e Exp2-B, onde foi injetada apenas a terceira variante do ataque de injeção de dados não autorizados. Conforme descrito em 5.2, trata-se de

Tabela 5: Métricas (Experimento 1).

Métrica	Valor
Acurácia	0,9979
Precisão	0,9847
Recall	0,9969
Especificidade	0,9980
F1-score	0,9909

um ataque bem mais sutil, mas que também oferece riscos. O objetivo foi aumentar a complexidade da anomalia de modo a tornar sua detecção mais desafiadora ao algoritmo SVM.

Novamente, foi utilizado o *kernel* RBF juntamente com as técnicas de *grid-search* e validação cruzada para encontrar os melhores parâmetros C e γ, sendo estes 32768,0 e 8,0, respectivamente. Aplicando o método, obteve-se a matriz de confusão, conforme Tabela 6, a partir da qual foram extraídas as métricas apresentadas na Tabela 7. Neste caso, percebe-se que, apesar de o método ter obtido acurácia de 92%, o mesmo obteve especificidade de 96% e *recall* de 65% indicando uma menor taxa de acerto na detecção da classe 1 e demonstrando uma maior complexidade na detecção destas anomalias. Na Figura 20, é possível visualizar uma amostra do método em operação, onde em verde é representado o que foi detectado como anomalia.

Tabela 6: Matriz de confusão (Experimento 2).

		Classe prevista	
		0	1
Classe esperada	0	VN: 2463	FP: 105
	1	FN: 110	VP: 202

5.5.2.3 Experimento 3

A fim de comparar o desempenho do SVM com outros modelos de aprendizado de máquina, no terceiro experimento, foram avaliados os algoritmos de *Random Forest*, *Naive Bayes* e *Multilayer Perceptron*, sendo utilizados os mesmos *datasests* do segundo experimento. Todos os algoritmos foram empregados utilizando o *framework* Weka (versão 3.8.6) [72]. O Weka é um software livre e de código aberto que implementa uma coleção de algoritmos de aprendizado de máquina para tarefas de mineração de da-

Tabela 7: Métricas (Experimento 2).

Métrica	Valor
Acurácia	0,9253
Precisão	0,6580
Recall	0,6474
Especificidade	0,9591
F1-score	0,6527

Figura 20: Amostra do método em operação.

dos. Ele contém ferramentas para preparação de dados, classificação, regressão, agrupamento, regras de associação e visualização. Os modelos foram empregados utilizando as configurações padrão do Weka, obtendo-se assim as matrizes de confusão apresentadas nas Tabelas 8, 9 e 10.

Tabela 8: Matriz de confusão (Experimento 3 - *Random Forest*).

		Classe prevista	
		0	1
Classe esperada	0	VN: 2458	FP: 110
	1	FN: 167	VP: 145

Tabela 9: Matriz de confusão (Experimento 3 - *Naive Bayes*).

		Classe prevista	
		0	1
Classe esperada	0	VN: 2467	FP: 101
	1	FN: 198	VP: 114

Tabela 10: Matriz de confusão (Experimento 3 - *Multilayer Perceptron*).

		Classe prevista	
		0	1
Classe esperada	0	VN: 2483	FP: 85
	1	FN: 146	VP: 166

A partir das matrizes de confusão, foram obtidas as métricas demonstradas na Tabela 11. Analisando os resultados, percebe-se uma nítida vantagem da rede neural *Multilayer Perceptron* com relação aos outros algoritmos de aprendizado de máquina, porém, apresentando um desempenho inferior ao SVM quanto a acurácia, *recall* e *F1-score*, mas sendo mais eficiente em especificidade e precisão.

Por serem *datasets* desbalanceados, onde existe uma incidência significativamente maior de uma classe com relação a outra, a precisão é uma métrica que merece atenção especial, pois ela indica a qualidade das previsões da classe minoritária que, neste caso, corresponde às anomalias. Uma maior precisão denota um possível potencial no uso de redes neurais para este problema específico, sendo necessários mais estudos a fim de evidenciá-lo.

Tabela 11: Métricas (Experimento 3).

Métrica	*Random Forest*	*Naive Bayes*	*Multilayer Perceptron*	*Support Vector Machine*
Acurácia	0,9038	0,8962	0,9198	**0,9253**
Precisão	0,5686	0,5302	**0,6614**	0,6580
Recall	0,4647	0,3654	0,5321	**0,6474**
Especificidade	0,9572	0,9607	**0,9669**	0,9591
F1-score	0,5114	0,4326	0,5897	**0,6527**

6 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Neste trabalho, foi abordado o problema de segurança presente em sistemas robóticos baseados em ROS, sendo avaliado o uso de técnicas de detecção de anomalias como meio de detectar atividades maliciosas nestes sistemas e assim dar suporte na tomada de decisão e prover maior segurança.

Primeiramente, foi feita uma introdução ao sistema ROS e às técnicas empregadas nesta pesquisa, a fim de nivelar leitores de diferentes áreas de conhecimento. Em seguida, foi realizada um revisão da literatura, onde foram apresentados diversos trabalhos apontando vulnerabilidades no sistema e propondo algumas abordagens, sendo discutido como tais abordagens diferem daquela proposta neste trabalho. Foram demonstrados os principais vetores de ataque e apresentada, em detalhes, a metodologia do ataque abordado, sendo enfim proposto um método para reconhecer este ataque.

O método empregou o algoritmo de aprendizado supervisionado *Support Vector Machine*, técnica amplamente adotada no contexto de detecção de anomalias, tendo esta sido utilizada para detectar ataques de injeção de dados não autorizados na forma de anomalias no tráfego de rede de uma aplicação ROS. Também foram avaliados, para fins de comparação, os algoritmos de *Random Forest*, *Naive Bayes* e *Multilayer Perceptron*. Foram conduzidos experimentos em simulação, cujos resultados foram avaliados por meio de métricas amplamente conhecidas.

A partir dos resultados obtidos percebe-se um grande potencial no uso de aprendizado de máquina para detecção de ataques de injeção de dados não autorizados em sistemas ROS, atingindo alta acurácia e especificidade em todos os experimentos. O método obteve excelentes resultados com anomalias menos complexas. Já com anomalias mais complexas o método teve maior dificuldade, mas ainda assim atingindo bons resultados, principalmente com os algoritmos SVM e *Multilayer Perceptron*. Um dado interessante sobre o *Multilayer Perceptron* é que o mesmo atingiu uma precisão levemente maior que os outros algoritmos, sem quaisquer calibração de parâmetros. Conforme mencionado anteriormente, a precisão é uma métrica importante em *datasets* desbalanceados, sendo que um melhor resultado nos experimentos indica um possível potencial de redes neurais nesta aplicação.

A avaliação explora um cenário em simulação, portanto, entende-se que o próximo passo lógico é a avaliação em um cenário real. Também entende-se que o método proposto pode ser aplicado em outros cenários. Nesse sentido, possibilidades de trabalhos futuros incluem a avaliação em aplicações mais complexas, tais como condução autônoma e execução de tarefas domésticas. Também é interessante avaliar o método quanto à injeção de outros tipos de dados maliciosos, como laser, imagem e coordenadas GPS; e quanto a anomalias de diferentes características e causas raiz (má configuração, falha de hardware, ataques de DoS, etc.).

Este trabalho utilizou a seleção manual de características, portanto, uma direção possível seria o aprendizado de características, a fim de otimizar o método e obter resultados ainda melhores. Outra direção interessante seria explorar, com maior ênfase, a capacidade de redes neurais nesta aplicação, experimentando também com outros tipos de redes neurais, tais como a TDNN (*Time Delay Neural Network*) [1]. Também merece atenção a avaliação de outras técnicas empregadas em detecção de anomalias, tais como algoritmos não supervisionados. Por último, é importante também a adaptação do método para suportar sistemas ROS2.

REFERÊNCIAS

[1] Al-Jarrah, O. and Arafat, A. (2015). Network intrusion detection system using neural network classification of attack behavior. *Journal of Advances in Information Technology Vol*, 6(1).

[2] Ali, M. Q. and Al-Shaer, E. (2013). Probabilistic model checking for ami intrusion detection. In *2013 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pages 468–473. IEEE.

[3] Amrouche, F., Lagraa, S., Frank, R., and State, R. (2020). Intrusion detection on robot cameras using spatio-temporal autoencoders: A self-driving car application. In *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, pages 1–5. IEEE.

[4] Asrodia, P. and Patel, H. (2012). Network traffic analysis using packet sniffer. *International journal of engineering research and applications*, 2(3):854–856.

[5] Bace, R. G. (2000). *Intrusion detection*. Sams Publishing.

[6] Bace, R. G., Mell, P., et al. (2001). Intrusion detection systems.

[7] Badger, J., Gooding, D., Ensley, K., Hambuchen, K., and Thackston, A. (2016). Ros in space: A case study on robonaut 2. In *Robot Operating System (ROS)*, pages 343–373. Springer.

[8] Bakken, D. (2001). Middleware. *Encyclopedia of Distributed Computing*, 11.

[9] Breiling, B., Dieber, B., and Schartner, P. (2017). Secure communication for the robot operating system. In *2017 annual IEEE international systems conference (SysCon)*, pages 1–6. IEEE.

[10] Byres, E. and Lowe, J. (2004). The myths and facts behind cyber security risks for industrial control systems. In *Proceedings of the VDE Kongress*, volume 116, pages 213–218. Citeseer.

[11] Cerrudo, C. and Apa, L. (2017). Hacking robots before skynet. *IOActive Website*, pages 1–17.

[12] Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58.

[13] Chang, C.-C. and Lin, C.-J. (2011). Libsvm: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):1–27.

[14] Cheminod, M., Durante, L., and Valenzano, A. (2012). Review of security issues in industrial networks. *IEEE transactions on industrial informatics*, 9(1):277–293.

[15] Chen, T. M. and Abu-Nimeh, S. (2011). Lessons from stuxnet. *Computer*, 44(4):91–93.

[16] Clark, G. W., Doran, M. V., and Andel, T. R. (2017). Cybersecurity issues in robotics. In *2017 IEEE conference on cognitive and computational aspects of situation management (CogSIMA)*, pages 1–5. IEEE.

[17] Combs, G. et al. (2006). Wireshark. URL `https://www.wireshark.org`. Accesso em: 22/01/2021.

[18] Conley, K. et al. (2009a). Ros concepts. URL `http://wiki.ros.org/ROS/Concepts`. Accesso em: 18/01/2021.

[19] Conley, K. et al. (2009b). Ros introduction - what is ros? URL `http://wiki.ros.org/ROS/Introduction`. Accesso em: 18/01/2021.

[20] Couceiro, M. et al. (2016). Projeto STOP - seguranças robóticos cooperativos. URL `http://stop.ingeniarius.pt`. Accesso em: 20/01/2021.

[21] DeMarinis, N., Tellex, S., Kemerlis, V. P., Konidaris, G., and Fonseca, R. (2019). Scanning the internet for ros: A view of security in robotics research. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8514–8521. IEEE.

[22] Dieber, B., Breiling, B., Taurer, S., Kacianka, S., Rass, S., and Schartner, P. (2017). Security for the robot operating system. *Robotics and Autonomous Systems*, 98:192–203.

[23] Dieber, B., Kacianka, S., Rass, S., and Schartner, P. (2016). Application-level security for ros-based applications. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4477–4482. IEEE.

[24] Dieber, B., White, R., Taurer, S., Breiling, B., Caiazza, G., Christensen, H., and Cortesi, A. (2020). Penetration testing ros. In *Robot Operating System (ROS)*, pages 183–225. Springer.

[25] Dóczi, R., Kis, F., Sütő, B., Póser, V., Kronreif, G., Jósvai, E., and Kozlovszky, M. (2016). Increasing ros 1. x communication security for medical surgery robot. In *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 004444–004449. IEEE.

[26] Durrant-Whyte, H. and Bailey, T. (2006). Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110.

[27] Eugster, P. T., Felber, P. A., Guerraoui, R., and Kermarrec, A.-M. (2003). The many faces of publish/subscribe. *ACM computing surveys (CSUR)*, 35(2):114–131.

[28] Fazzari, K. (2019). Ros 2 dds-security integration. URL `https://design.ros2.org/articles/ros2_dds_security.html`. Accesso em: 05/03/2022.

[29] Garage, W., Wise, M., and Foote, T. (2010). Turtlebot3 - what is a turtlebot? URL `https://www.turtlebot.com`. Accesso em: 09/04/2021.

[30] Garcia, E., Jiménez, M. A., de Santos, P. G., and Armada, M. A. (2007). The evolution of robotics research. *IEEE Robotics Autom. Mag*, 14(1):90–103.

[31] Garulli, A., Giannitrapani, A., Rossi, A., and Vicino, A. (2005). Mobile robot slam for line-based environment representation. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 2041–2046. IEEE.

[32] Gerkey, B. (2015). Why ros 2? URL `http://design.ros2.org/articles/why_ros2.html`. Accesso em: 05/02/2022.

[33] Group, O. M. (2021). What is dds? URL `https://www.dds-foundation.org/what-is-dds-3/`. Accesso em: 05/03/2022.

[34] Gupta, R., Kurtz, Z. T., Scherer, S., and Smereka, J. M. (2018). Open problems in robotic anomaly detection. *arXiv preprint arXiv:1809.03565*.

[35] Hax, V. A., Duarte Filho, N. L., da Costa Botelho, S. S., and Mendizabal, O. M. (2012). Ros as a middleware to internet of things. *Journal of Applied Computing Research*, 2(2):91–97.

[36] Herman, P. (2000). Tcpstat. URL `https://www.frenchfries.net/paul/tcpstat/`. Accesso em: 10/03/2021.

[37] Howard, A. et al. (2002). Gazebo. URL `http://gazebosim.org`. Accesso em: 05/07/2021.

[38] Hsu, C.-W., Chang, C.-C., Lin, C.-J., et al. (2003). A practical guide to support vector classification.

[39] Jacobson, V. et al. (1988). Tcpdump. URL `https://www.tcpdump.org/`. Accesso em: 10/03/2021.

[40] Jeong, S.-Y., Choi, I.-J., Kim, Y.-J., Shin, Y.-M., Han, J.-H., Jung, G.-H., and Kim, K.-G. (2017). A study on ros vulnerabilities and countermeasure. In *Proceedings of the Companion of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, pages 147–148.

[41] Junior, G. (2010). Máquina de vetores suporte: estudo e análise de parâmetros para otimização de resultado. *Ciência da Computação, Universidade Federal de Pernambuco*.

[42] Jyothsna, V., Prasad, R., and Prasad, K. M. (2011). A review of anomaly based intrusion detection systems. *International Journal of Computer Applications*, 28(7):26–35.

[43] Koenig, N. and Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. IEEE.

[44] Koubâa, A. et al. (2017). *Robot Operating System (ROS).*, volume 1. Springer.

[45] Kumar, V. (2005). Parallel and distributed computing for cybersecurity. *IEEE Distributed Systems Online*, 6(10).

[46] Lagraa, S., Cailac, M., Rivera, S., Beck, F., and State, R. (2019). Real-time attack detection on robot cameras: A self-driving car application. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 102–109. IEEE.

[47] Lin, C.-H., Liu, J.-C., and Ho, C.-H. (2008). Anomaly detection using libsvm training tools. In *2008 international conference on information security and assurance (isa 2008)*, pages 166–171. IEEE.

[48] Macenski, S., Foote, T., Gerkey, B., Lalancette, C., and Woodall, W. (2022). Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074.

[49] Maglaras, L. A. and Jiang, J. (2014). Intrusion detection in scada systems using machine learning techniques. In *2014 Science and Information Conference*, pages 626–631. IEEE.

[50] Mairh, A., Barik, D., Verma, K., and Jena, D. (2011). Honeypot in network security: a survey. In *Proceedings of the 2011 international conference on communication, computing & security*, pages 600–605.

[51] Mayoral-Vilches, V., Pinzger, M., Rass, S., Dieber, B., and Gil-Uriarte, E. (2020). Can ros be used securely in industry? red teaming ros-industrial. *arXiv preprint arXiv:2009.08211*.

[52] McClean, J., Stull, C., Farrar, C., and Mascarenas, D. (2013). A preliminary cyber-physical security assessment of the robot operating system (ros). In *Unmanned Systems Technology XV*, volume 8741, page 874110. International Society for Optics and Photonics.

[53] Meyer, D. and Wien, F. T. (2015). Support vector machines. *The Interface to libsvm in package e1071*, 28.

[54] Mitchell, R. and Chen, I.-R. (2014). A survey of intrusion detection techniques for cyber-physical systems. *ACM Computing Surveys (CSUR)*, 46(4):1–29.

[55] Mohammadi Rouzbahani, H., Karimipour, H., Rahimnejad, A., Dehghantanha, A., and Srivastava, G. (2020). *Anomaly Detection in Cyber-Physical Systems Using Machine Learning*, pages 219–235. Springer International Publishing, Cham.

[56] Moss, J. et al. (2012). Def con 20. URL `https://www.defcon.org/html/defcon-20/dc-20-index.html`. Accesso em: 22/01/2021.

[57] Narayanan, V. and Bobba, R. B. (2018). Learning based anomaly detection for industrial arm applications. In *Proceedings of the 2018 Workshop on Cyber-Physical Systems Security and PrivaCy*, pages 13–23.

[58] Pardo-Castellote, G. (2003). Omg data-distribution service: Architectural overview. In *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.*, pages 200–206. IEEE.

[59] Petrara, D. (2019). The rise of ros: Nearly 55% of total commercial robots shipped in 2024 will have at least one robot operating system package.(2019). *URL: https://www. bloomberg. com/press-releases/2019-05-16/the-rise-of-ros-nearly-55-of-total-commercial-robots-shipped-in-2024-will*, 20.

[60] Portugal, D., Pereira, S., and Couceiro, M. S. (2017). The role of security in human-robot shared environments: A case study in ros-based surveillance robots. In *2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 981–986. IEEE.

[61] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A. Y., et al. (2009). Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan.

[62] Quigley, M., Gerkey, B., and Smart, W. D. (2015). *Programming Robots with ROS: a practical introduction to the Robot Operating System*. "O'Reilly Media, Inc.".

[63] Rivera, S. (2021). *Securing Robots: An Integrated Approach for Security Challenges adn Monitoring for the Robotic Operating System*. PhD thesis, University of Luxembourg, Luxembourg.

[64] Rivera, S., Iannillo, A. K., et al. (2020). Ros-immunity: Integrated approach for the security of ros-enabled robotic systems.

[65] Rivera, Z. B., De Simone, M. C., and Guida, D. (2019). Unmanned ground vehicle modelling in gazebo/ros-based environments. *Machines*, 7(2):42.

[66] Shuttleworth, M. et al. (2004). The story of ubuntu. URL `https://ubuntu.com/about`. Accesso em: 10/04/2021.

[67] Teixeira, R. R., Maurell, I. P., and Drews, P. L. (2020). Security on ros: analyzing and exploiting vulnerabilities of ros-based systems. In *2020 Latin American Robotics Symposium (LARS), 2020 Brazilian Symposium on Robotics (SBR) and 2020 Workshop on Robotics in Education (WRE)*, pages 1–6. IEEE.

[68] Tran, Q.-A., Duan, H., and Li, X. (2004). One-class support vector machine for anomaly network traffic detection. *China Education and Research Network (CERNET), Tsinghua University, Main Building*, 310.

[69] Vapnik, V. (1999). *The nature of statistical learning theory*. Springer science & business media.

[70] Vasquez, G., Miani, R. S., and Zarpelao, B. B. (2017). Flow-based intrusion detection for scada networks using supervised learning. *XVII Simpósio Brasileiro em Seguranca da Informacao e de Sistemas*, pages 168–181.

[71] White, R., Christensen, D., Henrik, I., Quigley, D., et al. (2016). Sros: Securing ros over the wire, in the graph, and through the kernel. *arXiv preprint arXiv:1611.07060*.

[72] Witten, I. H., Frank, E., Trigg, L. E., Hall, M. A., Holmes, G., and Cunningham, S. J. (1999). Weka: Practical machine learning tools and techniques with java implementations.

[73] Yoon, M.-K. and Ciocarlie, G. F. (2014). Communication pattern monitoring: Improving the utility of anomaly detection for industrial control systems. In *NDSS Workshop on Security of Emerging Networking Technologies*.

[74] Zamora, E. and Yu, W. (2013). Recent advances on simultaneous localization and mapping for mobile robots. *IETE Technical Review*, 30(6):490–496.