

Análise de Vulnerabilidades no Sistema Supervisório Industrial SCADA-LTS sob a Perspectiva do OWASP

Francisco Santana de Sousa Júnior, Marcelo Diniz Leite e Iguatemi E. Fonseca

Resumo— Este artigo apresenta um estudo de vulnerabilidades no sistema supervisório industrial de código aberto SCADA-LTS, utilizando o catálogo OWASP como guia de referência. A investigação identificou falhas de *Cross-Site Request Forgery* (CSRF), de consumo irrestrito de recursos e de armazenamento inadequado de credenciais, além da exposição de funcionalidades administrativas sensíveis. Tais falhas, se exploradas, podem comprometer a integridade de processos industriais e afetar serviços essenciais como água e energia. Para cada vulnerabilidade, apresentam-se os métodos de análise, as evidências obtidas e as mitigações recomendadas.

Abstract— This article presents a study of vulnerabilities in the open-source industrial supervisory system SCADA-LTS, using the OWASP catalog as a reference guide. The investigation identified Cross-Site Request Forgery (CSRF) flaws, unrestricted resource consumption flaws, and improper credential storage flaws, as well as the exposure of sensitive administrative functionalities. Such flaws, if exploited, could compromise the integrity of industrial processes and affect essential services such as water and energy. For each vulnerability, the analysis methods, the evidence obtained, and the recommended mitigations are presented.

Palavras-Chave— SCADA-LTS, cibersegurança industrial, redes de computadores, teste de penetração.

Keywords— SCADA-LTS, industrial cybersecurity, computer networks, penetration testing.

I. INTRODUÇÃO

O SCADA-LTS (*Supervisory Control and Data Acquisition – Long Term Support*) é uma plataforma supervisória de código aberto amplamente utilizada em Sistemas de Controle Industrial (ICS) de empresas que gerenciam serviços de água e energia. Por meio de uma interface Web, a plataforma permite o gerenciamento remoto de pontos físicos, como caldeiras e válvulas. A descoberta de vulnerabilidades nesses sistemas é de extrema importância, visto que ataques bem-sucedidos podem impactar negativamente a sociedade [1], [2]. A temperatura de uma caldeira, por exemplo, pode ser remotamente alterada para um valor crítico, causando acidentes; ou uma válvula pode ser indevidamente acionada, ocasionando danos materiais [3].

O SCADA-LTS é de código aberto e multiplataforma, com o projeto disponível publicamente em um repositório no GitHub [4]. Seguindo as instruções de seu arquivo README.md, é possível instanciar rapidamente o sistema em uma máquina

Linux. Por essas características, o SCADA-LTS foi escolhido como objeto de estudo deste trabalho. Seu núcleo (*backend*) é implementado em Java, executando sobre o servidor de aplicação Apache Tomcat e apoiado no Spring Framework, enquanto a interface (*front-end*) utiliza o framework JavaScript Vue.js. Como se trata de um sistema Web, decidiu-se utilizar a metodologia OWASP (*Open Worldwide Application Security Project*) como guia para mapeamento da superfície de ataque do sistema, com o objetivo de identificar brechas passíveis de exploração por atacantes. Recentemente, alguns trabalhos têm estudado o contexto de cibersegurança em sistemas industriais com SCADA, mas também levando-se em conta as vulnerabilidades presentes no OWASP [5], [6], [7].

O OWASP é uma fundação sem fins lucrativos voltada à melhoria da segurança de software. Entre seus projetos, destaca-se o OWASP TOP 10, um documento de referência que, a cada 2, 3 ou 4 anos, consolida os 10 riscos de segurança mais críticos para aplicações Web, a partir de dados reais coletados por milhares de aplicações testadas. Como referência, adotaram-se neste trabalho dois catálogos OWASP complementares: o OWASP Top 10:2025 [8], voltado a aplicações Web, e o OWASP API Security Top 10:2023 [10], voltado a interfaces de programação. Essa combinação justifica-se pelo fato de o SCADA-LTS expor simultaneamente uma interface Web tradicional e endpoints de API consumidos por seu frontend Vue.js, exigindo análise sob ambas as perspectivas.

Por se tratar de um sistema de código aberto, adotou-se uma abordagem *white-box*, na qual o código fonte está disponível para análise. Isso permitiu estudar cada componente do sistema, seus papéis de usuário e suas implicações de segurança tendo o OWASP Top 10 como referência. A partir dessa análise, identificaram-se três vulnerabilidades principais do SCADA-LTS, detalhadas nas seções subsequentes: a recuperação de credenciais via funcionalidade SQL administrativa combinada a hash fraco, a ausência de proteção contra consumo irrestrito de recursos e a manipulação remota de estado via CSRF por requisições GET.

O restante deste artigo está organizado da seguinte forma. A Seção II apresenta os conceitos fundamentais para o trabalho. A Seção III descreve a metodologia adotada, incluindo o ambiente de testes e o processo de mapeamento da superfície de ataque. As Seções IV, V e VI detalham, respectivamente, a funcionalidade SQL administrativa e o armazenamento de credenciais, a vulnerabilidade de CSRF e o consumo irrestrito de recursos. A Seção VII reúne as mitigações recomendadas e a Seção IX apresenta as conclusões e os trabalhos futuros.

II. FUNDAMENTAÇÃO TEÓRICA

A. Sistemas SCADA e o SCADA-LTS

Esta seção apresenta os conceitos que fundamentam o trabalho: os sistemas SCADA e o SCADA-LTS, o catálogo OWASP e as classes de vulnerabilidade investigadas. Sistemas SCADA realizam controle industrial remoto e supervisão centralizada por meio de sensores, atuadores, controladores (PLCs e PACs) e Interface Homem-Máquina (IHM). O SCADA-LTS [4] implementa esse modelo como plataforma Web de código aberto, com *backend* em Java sobre Apache Tomcat e *frontend* disponível em duas versões uma legada em JSP e outra moderna em Vue.js. Seu domínio é organizado em torno de *DataSources* (fontes de dados), *DataPoints* (variáveis individuais monitoradas ou controladas), *Watchlists* (agrupamentos de *DataPoints*), Event Detectors (gatilhos de eventos) e Event Handlers (ações associadas). Historicamente, protocolos industriais como o Modbus (1979) operavam em redes isoladas e não previam autenticação ou criptografia [11]. Ao expor uma interface Web, o SCADA-LTS herda a superfície de ataque típica de aplicações Web, incluindo as classes listadas pelo OWASP Top 10, agora com potencial de consequências físicas.

B. OWASP Top 10 como Referência de Avaliação

O OWASP mantém catálogos de vulnerabilidades que representam os maiores riscos em aplicações modernas. O OWASP TOP 10 trata de aplicações Web como um todo, sendo sua versão mais recente a do ano de 2025 e a anterior de 2021. No entanto, a organização possui uma abrangência maior do que apenas no âmbito Web exclusivamente, mantendo catálogos que envolvem IoT (*Internet of Things*) e API (*Application Programming Interface*), no caso deste último, o TOP 10 OWASP API Security, lançado em 2023 [10].

A análise conduzida neste trabalho partiu de uma estratégia exploratória inspirada em trabalho anterior que investigou vulnerabilidades em APIs REST de controladores SDN (*Software-Defined Networking*) ONOS (*Open Network Operating System*) [9]. Partindo então dessa análise anterior, aplicou-se o mesmo padrão de investigação sobre o SCADA-LTS, ou seja, buscar vulnerabilidades envolvendo credenciais padrão e a partir do acesso obtido examinar progressivamente os componentes do sistema. A investigação contemplou ainda a busca por vulnerabilidades fora do escopo abordado no trabalho anterior sobre o ONOS, ampliando a visão sobre as brechas passíveis de exploração.

Esse processo exploratório conduziu a achados mapeados em categorias dos dois catálogos. Do OWASP Top 10:2025 [8], foram identificadas falhas de *Identification and Authentication Failures* (credenciais padrão), *Security Misconfiguration* (funcionalidade administrativa de execução de SQL exposta e proteção CSRF desabilitada) e *Cryptographic Failures* (armazenamento de senhas em SHA-1 sem salt). Do OWASP API Security Top 10:2023 [10], foi identificada a falha de *Unrestricted Resource Consumption nos endpoints de autenticação*.

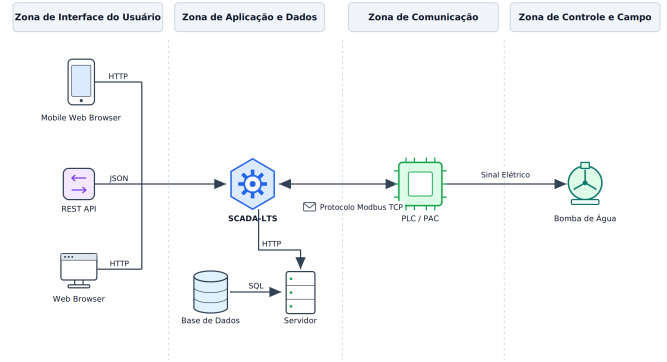


Fig. 1: Acesso ao SCADA-LTS via interface Web.

C. Categorias de Vulnerabilidade Investigadas

A análise resultou em achados mapeados em três frentes, cada uma associada a categorias distintas do catálogo OWASP e selecionada por seu impacto direto sobre a operação do sistema, detalhadas em seção específica.

Broken Access Control, ou Controle de acesso quebrado, diz respeito à aplicação inadequada de restrições sobre o que o usuário pode fazer, bem como à fragilidade dos mecanismos de identificação de usuários e aos modelos de permissões dos mesmos. No SCADA-LTS, essa categoria se manifesta na adoção de credenciais administrativas padrão básicas (usuário *admin* e senha *admin*) que são recuperáveis por script de ataque de dicionário, conforme detalhado na Seção IV.

Security Misconfiguration refere-se a configurações inseguras em qualquer camada da aplicação, a proteções desativadas ou a recursos indevidamente expostos. No SCADA-LTS, essa categoria manifesta-se em dois pontos: a exposição de uma funcionalidade administrativa de execução de SQL que permite visualizar as senhas dos usuários, armazenadas por meio de um *hash* fraco e a desativação da proteção contra CSRF, que possibilita a manipulação remota de operações sensíveis. Esses pontos são detalhados nas Seções IV e V, respectivamente.

Unrestricted Resource Consumption corresponde à ausência de limitação do número de requisições ou sobre o volume de recursos consumidos por uma mesma origem. No SCADA-LTS, a ausência de qualquer mecanismo de limitação de requisições permite que o sistema fique indisponível quando instanciado em uma única máquina, conforme analisado na Seção VI.

III. METODOLOGIA

Por ser um sistema de código aberto, o código-fonte do SCADA-LTS está publicamente disponível, o que viabilizou a adoção de uma abordagem *white-box*. Os experimentos foram conduzidos sobre a Versão 2.8.0 do SCADA-LTS, instanciada localmente por meio de contêineres Docker a partir da configuração oficial do projeto. O ambiente é composto por dois contêineres principais: a aplicação SCADA-LTS, executada sobre o servidor Apache Tomcat, e o banco de dados MySQL 8, ambos hospedados em uma máquina Linux com a distribuição Kali. A Figura 1 ilustra a estratégia usada para acesso ao SCADA-LTS via interface Web.

Os scripts de exploração ofensiva, desenvolvidos em Python, foram executados a partir de uma máquina distinta, que acessava a instância-alvo por meio de encaminhamento de porta. Essa separação entre a origem do ataque e o sistema atacado aproxima o cenário de teste de uma situação real. Tendo os catálogos OWASP como guia, a investigação combinou a inspeção do código-fonte, a análise do tráfego HTTP com as ferramentas Wireshark [13] e Burp Suite [14] e a manipulação prática das funcionalidades da plataforma.

IV. ACESSO INICIAL E RECUPERAÇÃO DE CREDENCIAIS

Esta seção descreve a obtenção do acesso inicial pela credencial padrão e a subsequente recuperação das senhas armazenadas no sistema.

A. Força Bruta sobre a credencial padrão

Desenvolveu-se um script em python, mostrado na Listagem 1, que automatiza tentativas de login contra o endpoint *Scada-LTS/login.htm*. O script percorre uma *wordlist* de senhas facilmente encontrada na internet, a *password-list.txt*, e para cada entrada submete-se uma requisição POST com usuário *admin*.

Diferentemente de processos de autenticação Web mais modernos, no qual é possível verificar se o login deu certo ou não pelo código 200 (*Success*) ou 401 (*Unauthorized*), no SCADA-LTS, caso a requisição seja mal sucedida, o sistema redireciona para *login.htm?error* e tentativas bem-sucedidas para o *dashboard* administrativo. Após cerca de quinze minutos, a senha é descoberta. Esse tempo decorre de dois fatores: a senha encontra-se em uma posição avançada da lista utilizada, e cada tentativa corresponde a uma requisição HTTP individual, sujeita à latência de rede e ao tempo de resposta do servidor. Observou-se então um padrão em relação ao trabalho anterior, no qual o usuário e senha padrão também eram fáceis de serem descobertos seguindo essa metodologia [9].

Listagem 1: Verificação de sucesso de login pela URL de resposta.

```
res = session.post(f'{HOST}/Scada-LTS/login.htm',
                  data=form_data)
if res.url == SUCCESS_LOCATION:
    return True # senha correta
```

B. Funcionalidade SQL em produção e Armazenamento de senhas

A investigação dos componentes e funcionalidades do SCADA-LTS verificou a existência de uma seção no sistema na qual usuários administradores conseguem executar código SQL, retornando informações sobre o banco de dados. Verificou-se que apenas *SELECT* é possível de ser feito, quando executa-se *UPDATE* retorna erro. Investigando o repositório no *GitHub*, verificou-se que a restrição a *SELECT* decorreu de questões de performance, e não de segurança, havendo indícios de que versões anteriores permitiam *UPDATE*.

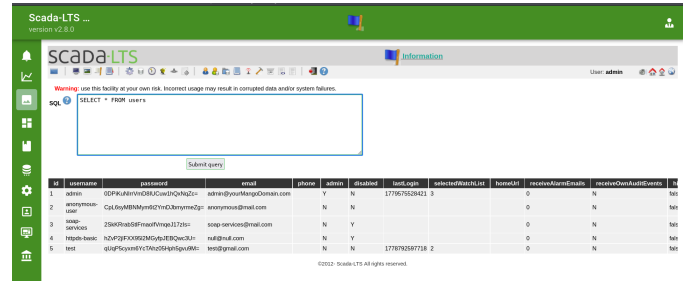


Fig. 2: Execução da *query* de usuários no SCADA-LTS.

De qualquer forma, percebeu-se que ao executar *SELECT * FROM users* no campo de execução de SQL, era possível ver a coluna de senha armazenada como hash, conforme a Figura 2.

A análise do valor armazenado revelou características como: cadeia de 28 caracteres no alfabeto base64, terminada pelo caractere de preenchimento *=*. A decodificação desse valor retorna um valor de 20 bytes, exatamente o tamanho de saída do algoritmo SHA-1. A partir dessa observação, levantou-se a hipótese que as senhas são armazenadas em *base64(SHA-1(senha))*. Dessa forma, criou-se um script em python, mostrado na Listagem 2, também usando uma *wordlist*, que aplica o mesmo procedimento que o sistema faz para armazenar suas senhas. Se o valor comparado naquela iteração é igual, então encontrou-se a senha. O script então conseguiu recuperar com sucesso a senha correspondente ao hash analisado. O uso de SHA-1 hoje é considerado inadequado para o armazenamento de senhas [16], o que é pior quando não há salt. Com isso, um atacante com acesso ao sistema poderia recuperar credenciais de usuários por meio de ataques dicionário, sem qualquer limite de tentativas ao acessar a funcionalidade de SQL nativa do sistema.

Listagem 2: Recuperação da senha por comparação com a *wordlist*.

```
target = "0DPiKuNIrrVmD8IUCuw1hQxNqZc="

for p in open('password-wordlist.txt'):
    senha = p.strip()
    h = hashlib.sha1(senha.encode()).digest()
    if base64.b64encode(h).decode() == target:
        print(f'senha encontrada: {senha}')
        break
```

V. MANIPULAÇÃO DE ESTADO VIA CSRF.

Esta seção analisa como a ausência de proteção contra CSRF, aliada ao uso de GET para operações de escrita, permite manipular remotamente o estado do sistema.

A. Configuração de segurança e Anti-Padrão GET

Após análise de configuração de segurança do SCADA-LTS, verificou-se que a proteção contra CSRF, oferecida nativamente pelo *Spring Security* [15], encontra-se explicitamente desabilitada. Essa diretiva pode ser desabilitada no arquivo *spring-security.xml*, no qual ficam declarações de segurança sobre o framework da seguinte maneira: *<csrf disabled="true"/>*. A remoção dessa proteção aparece em

todas as seis cadeias de filtros de segurança http definidas no arquivo.

No *Spring Security*, cada bloco http define uma cadeia de filtros de segurança aplicada a um conjunto de URL, determinado pelo atributo *pattern*. No SCADA-LTS, *patterns* envolvendo serviços SOAP (*/services/***), rotas administrativas (*/api/secure/***) e a cadeia padrão (cobrindo toda a aplicação), aparecem com a diretiva de proteção contra CSRF desabilitada. A esse cenário soma-se um agravante identificado no exame dos *endpoints*: embora exista uma rota com método POST para operações de alteração de valores, diversos *endpoints* GET equivalentes permanecem acessíveis para as mesmas operações. O caso mais crítico é a requisição GET para alteração de *datapoint*: *api/pointvalue/setValue/xid/type/value*. Dessa forma, alterações significativas de valores podem ser feitas com simples requisições GET, o que não é mitigado apenas reabilitando o Spring Security, já que a validação por token do CSRF não se aplica a GET, sendo necessário migrar as operações de escrita para métodos como POST, PUT, PATCH ou DELETE.

B. Exploração e Impacto

Usando a instância do SCADA-LTS local, criou-se um *datapoint* e um *datasource* para teste. Uma requisição maliciosa foi criada, apontando para o respectivo *datapoint*, usando a url da seção anterior e um valor fora da faixa operacional esperada. Essa requisição inicialmente foi colocada em uma *tag image* em um html. No entanto, a vulnerabilidade só foi explorada ao usar a função *window.location* para redirecionamento automático. Com isso, o navegador anexa o cookie JSESSIONID, e se ele corresponde a um usuário administrador logado, a requisição é realizada com êxito.

Dessa forma, um atacante pode hospedar uma página html com esse comportamento (supondo conhecimento prévio do identificador do *datapoint*) em um endereço aparentemente inofensivo e enviar para um administrador por e-mail. No instante em que o administrador autenticado abre o email com o link e clica no mesmo, a requisição é disparada pelo seu próprio navegador e o valor do *datapoint* é alterado. Em navegadores padrão, o envio de cookies é bloqueado em requisições POST de origem cruzada, o fato dessa requisição ser aceita como GET contorna isso e mantém o ataque viável.

Como *datapoints* representam não só leituras, como também comandam atuadores, a manipulação pode alterar o valor de referência de temperatura de uma caldeira ou interromper o funcionamento de uma bomba, como mostra a Figura 1. Devido ao fato da ação ter sido feita com êxito por um administrador, o rastreamento da origem da requisição pode ser difícil. Pela mesma técnica, o *endpoint* */api/datasource/toggle?id=* permite desligar uma fonte de dados inteira, fazendo com que a interface de supervisão exiba valores desatualizados, sem que os *event detectors* avaliem novas condições ou disparem alarmes.

VI. CONSUMO IRRESTRITO DE RECURSOS

A partir de uma única máquina, avaliou-se a resiliência do SCADA-LTS diante de um volume elevado de requisições

simultâneas. Para isso, desenvolveu-se um script em Python baseado na biblioteca *asyncio*, capaz de disparar mil tarefas concorrentes de autenticação contra o *endpoint* de login, sem qualquer intervalo entre as requisições. A escolha de um modelo assíncrono permitiu gerar carga expressiva a partir de um único processo, dispensando a infraestrutura distribuída normalmente associada a ataques de negação de serviço.

Constatou-se que o SCADA-LTS não impõe qualquer limitação de taxa (*rate limiting*) sobre o número de requisições aceitas por origem. Sob a carga gerada, o tempo de resposta do servidor degradou-se progressivamente, e parte das requisições passou a ser descartada por esgotamento de *timeout*, conforme registrado nos *logs* da execução. A ausência desse mecanismo de proteção caracteriza a falha de *Unrestricted Resource Consumption*: um único atacante é capaz de comprometer a disponibilidade da plataforma. Em um sistema supervisorio, essa indisponibilidade é particularmente grave, pois priva o operador da capacidade de monitorar o processo industrial e de reagir a eventos críticos durante o período do ataque.

VII. MITIGAÇÕES RECOMENDADAS

Para a questão da autenticação, após um primeiro acesso, forçar a alteração de senha e usuário para fugir da senha padrão. Para impedir consumo irrestrito de recursos, recomenda-se implementar limitação após um certo número de tentativas de login. Para não só essa, mas também para outras requisições, limitar, para um mesmo IP, o número possível de requisições que podem ser feitas por minuto e, se passar desse limiar, bloquear mais requisições deste IP.

Para o armazenamento de senhas, substituir o esquema *base64(SHA-1(senha))* por uma função de hash própria para senhas como bcrypt ou Argon2 [12] com salt único por usuário. A migração pode ser feita aos poucos, restaurando a senha do usuário após login bem-sucedido, sem precisar resetar tudo de uma vez.

A funcionalidade de rodar SQL diretamente na plataforma deveria, idealmente, ser removida, visto que sua mera presença pode ser uma porta aberta para futuros ataques. Como foi o caso do problema descrito em relação ao armazenamento de senhas.

Sobre o CSRF, seria necessário habilitar novamente a proteção do Spring Security, além de corrigir o anti-padrão GET em que operações de escrita estão sendo realizadas como *setValue* e *toggle*, que deveriam, na verdade, usar apenas PUT, DELETE, PATCH, POST.

VIII. TRABALHOS RELACIONADOS

A segurança de sistemas de controle industrial é objeto de atenção crescente na literatura. O documento NIST SP 800-82 [11] consolida diretrizes para a proteção de sistemas de tecnologia operacional, destacando a fragilidade de protocolos historicamente concebidos para operar em redes isoladas, sem mecanismos de autenticação ou criptografia. Tais diretrizes, contudo, tratam predominantemente da arquitetura e dos protocolos de campo, dedicando menor atenção à camada de supervisão exposta via Web.

Krishnan e Wei [17] propõem um laboratório SCADA de baixo custo voltado a testes de penetração, análise de vulnerabilidades e perícia de incidentes; o trabalho concentra-se em hardware e protocolos industriais com finalidade didática, não tratando da exploração de uma plataforma de supervisão específica. Rashid et al. [5] propõem uma arquitetura de referência unificada para a convergência entre pipelines de IA e infraestrutura em nuvem em sistemas ciber-físicos. Articula uma taxonomia de ameaças por estágios de ciclo de vida e mapeando controles contra *frameworks* como NIST AI RMF, MITRE ATLAS, OWASP AI Exchange, e NERC CIP, com ênfase na proteção da cadeia de suprimentos do modelo e na governança em tempo de execução. Chowdhury et al. [6] conduzem uma análise empírica de larga escala sobre 14.133 registros de ataques distribuídos em 63 categorias de segurança. Com base em testes estatísticos de associação entre categorias e vetores, propõem dois *frameworks* adaptativos (CADF e MPI) para a geração de *playbooks* de mitigação ajustados ao perfil de ameaça observado. Tomando o OWASP Top 10 e o MITRE ATT&CK como referenciais de partida. Algül et al. [7], por sua vez, apresentam o SCADANet, *dataset* construído em ambiente virtual SCADA com tráfego Modbus/TCP rotulado para treze cenários de ataque e empregado no treinamento de sistemas de detecção de intrusão baseados em aprendizado profundo, atuando sobre a camada de rede e do protocolo industrial. Tais investigações abordam, respectivamente, a governança de IA, a seleção adaptativa de controles e a detecção em nível de rede, sem instanciar a análise sobre a interface Web de uma plataforma supervisória específica — lacuna endereçada por este trabalho.

No que se refere à avaliação de segurança guiada por catálogos OWASP, trabalho anterior dos autores aplicou o OWASP API Security à análise de controladores de Redes Definidas por Software, identificando falhas de autenticação e de consumo irrestrito de recursos no controlador ONOS [9]. O presente trabalho dá continuidade a essa linha de investigação, transferindo a abordagem para o domínio dos sistemas supervisórios industriais e concentrando-se especificamente na interface Web do SCADA-LTS uma camada, cujo comprometimento, diferentemente de uma aplicação Web convencional, pode acarretar sérias consequências físicas.

IX. CONCLUSÃO

Este artigo apresentou uma análise de vulnerabilidades no sistema supervisório de código aberto SCADA-LTS, conduzida sob uma abordagem *white-box* e guiada pelo catálogo OWASP. A investigação identificou falhas em três frentes. A primeira, refere-se ao acesso inicial e à recuperação de credenciais, viabilizada pela adoção de senhas administrativas padrão e pelo armazenamento de senhas por meio de um esquema de hash fraco. A segunda, corresponde à ausência de proteção contra *Cross-Site Request Forgery*, agravada pela exposição de operações de escrita por meio de requisições GET. A terceira consiste na inexistência de limitação de consumo de recursos, que permite levar a plataforma à indisponibilidade a partir de uma única máquina.

Os resultados evidenciam que vulnerabilidades típicas de aplicações Web, quando presentes em um sistema que su-

pervisiona processos industriais, podem ter consequências que extrapolam o domínio digital, afetando a integridade do controle e a confiabilidade da supervisão. Cabe destacar que parte significativa da superfície de ataque identificada decorre de configurações que são na verdade oferecidas pela plataforma, mas que se encontram desativadas, o que indica que uma fração relevante dos riscos poderia ser mitigada sem desenvolvimento adicional.

Como trabalhos futuros, deseja-se a incorporação de agentes de Inteligência Artificial à análise de segurança desses sistemas em duas frentes complementares, tanto na parte ofensiva, para mapeamento de *endpoints* e verificação de riscos, quanto na defensiva, para detecção de anomalias. Adicionalmente, buscar novas vulnerabilidades, usando como base o OWASP.

REFERÊNCIAS

- [1] G. Singh Gaba, A. Sari, I. Butun, I., P. Singh, A. Gurtov, and M. Liyanage, "A survey on security and privacy of industry 4.0 and beyond: Technical aspects, use cases, challenges, and research directions," *IEEE Open Journal of the Communications Society*, pp: 8865–8929, 2025.
- [2] M. S. Alvarez-Alvarado, "Quantum Computing for Reliability Assessment of SCADA With Continuous Penetration Testing and High Penetration of Cyber-Physical Attacks," *IEEE Transactions on Smart Grid*, vol. 16, no. 6, pp. 5392–5403, Nov. 2025.
- [3] W. Li, Y. Yao, N. Zhang, C. Sheng, Z. Ran and W. Yang, "IMADP: Imputation-Based Anomaly Detection in SCADA Systems via Adversarial Diffusion Process," *IEEE Transactions on Network and Service Management*, vol. 23, pp. 3166–3179, 2026.
- [4] SCADA-LTS Project, "SCADA-LTS: Open Source SCADA — Long Term Support." Disponível em: <https://github.com/SCADA-LTS/Scada-LTS>. Acessado em maio de 2026.
- [5] S. M. Z. U. Rashid, D. Gurung, S. R. Gupta and S. Rath, "Lifecycle-Integrated Security for AI-Cloud Convergence in Cyber-Physical Infrastructure," 2026 14th International Symposium on Digital Forensics and Security (ISDFS), Boston, MA, USA, 2026, pp. 1–6.
- [6] I. Chowdhury, S. Sinha and B. Dutta, "Systematic Analysis of Cybersecurity Attacks and Evaluation of Mitigation Strategies," *IEEE 15th International Conference on Communication Systems and Network Technologies (CSNT)*, Al-Khobar, Saudi Arabia, 2026, pp. 656–661.
- [7] E. ALGUL, F. DOĞAN, A. Ayid Ahmad, O. POLAT, "SCADANet: A novel dataset for SCADA cybersecurity and intrusion detection," *Computer Networks*, Volume 278, pp: 112087, 2026.
- [8] OWASP Foundation, "OWASP Top 10:2025," 2025. Disponível em: <https://owasp.org/Top10/>. Acessado em maio de 2026.
- [9] M. D. S. Leite, F. S. de Sousa Júnior e I. E. Fonseca, "Análise de vulnerabilidades em aplicações que usam API REST," em *XLIII Simpósio Brasileiro de Telecomunicações e Processamento de Sinais (SBt)*, Natal, RN, 2025.
- [10] OWASP Foundation, "OWASP API Security Top 10," 2023. Disponível em: <https://owasp.org/www-project-api-security/>. Acessado em maio de 2026.
- [11] K. Stouffer et al., *Guide to Operational Technology (OT) Security*, NIST Special Publication 800-82, Revisão 3, National Institute of Standards and Technology, 2023.
- [12] A. Biryukov, D. Dinu e D. Khovratovich, "Argon2: the Memory-Hard Function for Password Hashing and Other Applications," *Password Hashing Competition*, 2015.
- [13] Wireshark Foundation, "Wireshark." Disponível em: <https://www.wireshark.org/>. Acessado em maio de 2026.
- [14] PortSwigger, "Burp Suite." Disponível em: <https://portswigger.net/burp>. Acessado em maio de 2026.
- [15] VMware, "Spring Security Reference — CSRF Protection." Disponível em: <https://docs.spring.io/spring-security/reference/>. Acessado em maio de 2026.
- [16] M. Stevens, E. Bursztein, P. Karpman, A. Albertini e Y. Markov, "The First Collision for Full SHA-1," em *Advances in Cryptology — CRYPTO 2017*, Springer, 2017, pp. 570–596.
- [17] S. Krishnan e M. Wei, "SCADA Testbed for Vulnerability Assessments, Penetration Testing and Incident Forensics," em *7th International Symposium on Digital Forensics and Security (ISDFS)*, Barcelos, Portugal, 2019, pp. 1–6.