DISSERTAÇÃO DE MESTRADO PROFISSIONAL

Proposta de Arquitetura para um Rastreador Veicular com Criptografia Ponta-a-ponta

Mauricio Madalozzo Bordini

Brasília, dezembro de 2024

Programa de Pós-Graduação Profissional em Engenharia Elétrica
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
FACULDADE DE TECNOLOGIA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

DISSERTAÇÃO DE MESTRADO PROFISSIONAL

Proposta de Arquitetura para um Rastreador Veicular com Criptografia Ponta-a-ponta

Mauricio Madalozzo Bordini

Brasília, dezembro de 2024

Dissertação de Mestrado Profissional submetida ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Mestre em Engenharia Elétrica

Banca Examinadora

Prof. Georges Daniel Amvame Nze, Ph.D, FT/UnB _____
Orientador

Prof. Robson de Oliveira Albuquerque, Ph.D, FT/UnB _____
Coorientador

Prof. Fábio Lúcio Lopes de Mendonça, Ph.D, FT/UnB _____
Examinador interno

Prof. Laerte Peotta de Melo, Ph.D, FT/UnB _____
Examinador externo

FICHA CATALOGRÁFICA

BORDINI, MAURICIO MADALOZZO
Proposta de Arquitetura para um Rastreador Veicular com Criptografia Ponta-a-ponta [Distrito Federal] 2024.
xvi, 74 p., 210 x 297 mm (ENE/FT/UnB, Mestre, Engenharia Elétrica, 2024).
Dissertação de Mestrado Profissional - Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica

1. Rastreador veicular	2. Criptografia
3. Segurança Cibernética	4. Proposta de Arquitetura
I. ENE/FT/UnB	II. Título (série)

PUBLICAÇÃO: PPEE.MP.075

REFERÊNCIA BIBLIOGRÁFICA

BORDINI, M.M. (2024). *Proposta de Arquitetura para um Rastreador Veicular com Criptografia Ponta-a-ponta*. Dissertação de Mestrado Profissional, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 74 p.

CESSÃO DE DIREITOS

AUTOR: Mauricio Madalozzo Bordini
TÍTULO: Proposta de Arquitetura para um Rastreador Veicular com Criptografia Ponta-a-ponta.
GRAU: Mestre em Engenharia Elétrica ANO: 2024
PUBLICAÇÃO: PPEE.MP.075

Mauricio Madalozzo Bordini
Depto. de Engenharia Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

DEDICATÓRIA

Dedico este trabalho primeiramente ao criador de todas as coisas, ao criador do universo: Deus. Também dedico este trabalho aos meus pais Gelma Salete Madalozzo e José Maria Dittert Bordini pelos valores repassados com relação aos estudos, em especial ao meu pai que foi o meu maior incentivador desde os primeiros passos na área de computação.

AGRADECIMENTOS

Agradeço imensamente a todos os professores do Programa de Pós-Graduação Profissional em Engenharia Elétrica (PPEE) da Universidade de Brasília (UnB), em especial ao meu orientador, Dr. Georges Daniel Amvame Nze, e ao meu coorientador Dr. Robson de Oliveira Albuquerque.

Agradeço também aos Professores Dr. Laerte Peotta de Melo e Fábio Lúcio Lopes de Mendonça, que juntos com o meu orientador e coorientador integraram minha banca de defesa de dissertação pelas críticas muito construtivas realizadas após a revisão do trabalho.

Agradeço também a todos os colegas que durante o curso contribuíram com os trabalhos nas mais diversas disciplinas e também a Secretaria do PPEE pelo apoio fundamental durante todo o curso.

Agradeço a minha namorada Eloiza Henequin pelo apoio incondicional durante os momentos mais difíceis durante o Mestrado.

RESUMO

Os rastreadores veiculares têm ampla utilização na sociedade moderna com diversas aplicações. Dentre essas o monitoramento de frota de veículos de empresas do ramo logístico, de viaturas das forças de segurança pública, de aplicativos de transporte e táxis, de veículos particulares. Apesar da existência de várias empresas que atuam provendo soluções para este propósito e mesmo da facilidade na obtenção de um rastreador para uso particular, grande parte dessas soluções não possuem foco na segurança, ficando sujeitas a ataques cibernéticos nos quais informações sensíveis podem ser vazadas ou interceptadas e até modificadas. Neste trabalho é proposto uma arquitetura para um rastreador veicular com criptografia ponta-a-ponta utilizando tecnologias abertas. Para comprovar os resultados foi criado um protótipo funcional com toda a infraestrutura necessária demonstrando a segurança na adoção da arquitetura. Além do foco na segurança proposto pela arquitetura, esta também foi pensada de forma que possa ser facilmente escalável para atender desde pequenas até grandes frotas de veículos.

Palavras-chaves: Rastreador veicular, Criptografia, Segurança cibernética, Proposta de arquitetura

ABSTRACT

Vehicle trackers are widely used in modern society for a variety of applications. These include monitoring the vehicle fleets of logistics companies, public security forces, transportation apps and cabs, and private vehicles. Despite the existence of several companies providing solutions for this purpose and even the ease of obtaining a tracker for private use, most of these solutions do not focus on security and are subject to cyber attacks in which sensitive information can be leaked or intercepted and even modified. This paper proposes an architecture for a vehicle tracker with end-to-end encryption using open technologies. To prove the results, a functional prototype was created with all the necessary infrastructure, demonstrating the security of adopting the architecture. In addition to the focus on security proposed by the architecture, it was also designed in such a way that it can be easily scaled to cater for small to large fleets of vehicles.

Keywords: Vehicle tracker, Cryptography, Cyber security, Architecture proposal

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE TERMOS E SIGLAS

ACL	*Access Control List*
AJAX	*Asynchronous JavaScript and XML*
AMQP	*Advanced Message Queuing Protocol*
APL	*Applied Physics Laboratory*
ARPA	*Advanced Research Projects Agency*
ASCII	*American Standard Code for Information Interchange*
BDS	*BeiDou Navigation Satellite System*
CLI	*Command Line Interface*
CoAP	*Constrained Application Protocol*
CSR	*Certificate Signing Request*
CSS	*Cascading Style Sheets*
CVE	*Common Vulnerabilities and Exposures*
DDL	*Data Definition Language*
DDoS	*Distributed Denial-of-Service*
DDS	*Data Distribution Service*
DML	*Data Manipulation Language*
GLONASS	*Global'naya Navigatsionnaya Sputnikovaya Sistema*
GND	*Ground*
GNSS	*Global Navigation Satellite System*
GPIO	*General-Purpose Input/Output*
GPS	*Global Positioning System*
GSM	*Global System for Mobile Communications*
HTTP	*Hypertext Transfer Protocol*
HTTPS	*Hyper Text Transfer Protocol Secure*
HSTS	*HTTP Strict Transport Security*
IEC	*International Electrotechnical Commission*
IoT	*Internet of Things*
ISO	*International Organization for Standardization*
ISP	*Internet Service Providers*
LCR	Lista de Certificados Revogados
M2M	*Machine to Machine*
MQTT	*Message Queuing Telemetry Transport*
NCSD	*National Cyber Security Division*
NMEA	*National Marine Electronics Association*
OASIS	*Organization for the Advancement of Structured Information Standards*
OCX	*Next Generation Operational Control System*
OWASP	*Open Web Application Security Project*
PHP	*PHP: Hypertext Preprocessor*

RFC	*Request for Comments*
RX	*Reception*
SA	*Selective Availability*
SAN	*Subject Alternative Name*
SMS	*Short Message Service*
SSL	*Secure Sockets Layer*
TLS	*Transport Layer Security*
TX	*Transmission*
UART	*Universal Asynchronous Receiver-Transmitter*
UTC	*Coordinated Universal Time*
VCC	*Voltage Common Collector*
XML	*Extensible Markup Language*
XMPP	*Extensible Messaging and Presence Protocol*

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

O rastreamento de veículos, principalmente pertencente a frotas para as mais diversas aplicações não é um fato recente. O rastreamento tem como propósito ajudar empresas de frota a recuperar veículos roubados, monitorar o diagnóstico dos veículos ou mesmo melhorar a eficiência dos veículos e reduzir os prazos de entrega. Além de empresas de frota, outros segmentos da sociedade também podem se beneficiar desta tecnologia, sendo um deles o setor de segurança pública.

A principal tecnologia utilizada, o *Global Positioning System (GPS) (1)* ou Sistema de Posicionamento Global, já foi pensada na década de 1960, sendo que muitas das primeiras implementações surgiram em meados e final da década de 1970. Em 1957, a Rússia lançou o Sputnik, o primeiro satélite a orbitar a Terra com sucesso. Enquanto o Sputnik orbitava o planeta, o satélite emitia um sinal de rádio. Um grupo de cientistas do Laboratório de Física Aplicada (*Applied Physics Laboratory - APL*) da Universidade Johns Hopkins observou um estranho fenômeno: A frequência dos sinais de rádio transmitidos pelo Sputnik aumentava à medida que o satélite se aproximava e diminuía à medida que se afastava. Esta mudança é conhecida em física como Efeito Doppler. A utilização do efeito Doppler do Sputnik permitiu aos cientistas utilizar sinais de rádio para seguir o movimento do satélite a partir do solo. Mais tarde, expandiram a ideia: se a localização de um satélite pudesse ser determinada a partir do solo através da mudança de frequência do seu sinal de rádio, então a localização de um receptor no solo poderia ser determinada pela sua distância a um satélite. Em 1958, a Agência de Projetos de Investigação Avançada (*Advanced Research Projects Agency - ARPA*) utilizou este princípio para desenvolver o Transit, o primeiro sistema global de navegação por satélite do mundo. O primeiro satélite do Transit foi lançado em 1960 e o conceito, desenvolvido pelo APL da Universidade John Hopkins, era capaz de fornecer navegação a utilizadores militares e comerciais, incluindo os submarinos de mísseis da Marinha. O programa foi transferido para a Marinha em meados da década de 1960 e, em 1968, uma constelação de 36 satélites estava totalmente operacional. A tecnologia do Transit permitia uma precisão de dezenas de metros e é considerada como tendo "melhorado a precisão dos mapas das áreas terrestres da Terra em quase duas ordens de grandeza", ajudando a aumentar a aceitação da navegação por satélite. O Transit funcionou durante 28 anos, até 1996, quando o Departamento de Defesa o substituiu pelo atual Sistema de Posicionamento Global (GPS) (2).

O Dr. Ivan Getting, presidente fundador da The Aerospace Corporation, imaginou um sistema mais potente e preciso, que ele via como "faróis no céu". Em 1963, a Aerospace começou a procurar maneiras de expandir e aprimorar um sistema de navegação por satélite. Um estudo da Aerospace de 1963, liderado por Phillip Diamond, recomendou um conceito chamado 621-B e, com a energia e a visão de Getting, a Força Aérea formou um novo programa de navegação por satélite chamado 621-B. Outros estudos de sistema realizados pelos engenheiros da Aerospace James Woodford e Hideyoshi Nakamura, concluídos em 1966, recomendaram uma arquitetura em que as medições de quatro satélites eliminariam a necessidade de relógios de alta precisão nos receptores. Isso serviu para reduzir significativamente os custos proibitivos, avançando assim a adoção da tecnologia ao torná-la economicamente mais viável (2).

Com base na arquitetura, cada satélite seria equipado com seu próprio relógio, atualizado periodicamente por meio de sinais de estações terrestres, que monitorariam as posições dos satélites GPS com um alto grau de precisão e exatidão. Posteriormente, a decisão de mover os relógios do receptor terrestre para o satélite teria implicações enormes: Sem a necessidade de incluir um relógio no solo, os dispositivos de GPS poderiam ser reduzidos - até mesmo pequenos o suficiente para caber dentro de um telefone celular (2).

Durante o restante da década de 1960, o desenvolvimento do GPS foi auxiliado por avanços tecnológicos, como microprocessadores de estado sólido, computadores e técnicas de utilização de largura de banda. O desenvolvimento de relógios atômicos no Centro Naval de Tecnologia Espacial do Laboratório de Pesquisa Naval (NRL) levou a avanços em um sistema de navegação por satélite conhecido como Timation (Time Navigation). Os dois primeiros satélites Timation, lançados em 1967 e 1968, foram equipados com relógios de oscilador de cristal. Um terceiro satélite, lançado em 1974, foi o primeiro equipado com um relógio atômico, o que melhorou muito a precisão e proporcionou uma cobertura de localização tridimensional (2).

Em fevereiro de 1978, foi lançado o primeiro satélite Navstar/GPS de desenvolvimento do Bloco I, com mais três satélites Navstar lançados até o final de 1978. Mais de 700 testes foram realizados entre 1977 e 1979, nos quais os engenheiros da Aerospace ajudaram a confirmar a precisão do sistema integrado de espaço/controle/usuário. Outros satélites de demonstração do GPS Bloco I foram lançados no início da década de 1980 (2).

Os satélites da constelação GPS são diferenciados por grupos chamados blocos, os quais representam diferentes gerações de satélites. A constelação GPS é uma mistura de satélites antigos e novos. Em 3 de julho de 2023, havia um total de 31 satélites operacionais na constelação GPS, sem incluir os sobressalentes desativados e em órbita. Nesta data faziam parte da constelação satélites do Bloco IIA (2ª geração, "Avançado"), do Bloco IIR ("Reposição"), do Bloco IIR-M ("Modernizado"), do Bloco IIF ("Acompanhamento"), do GPS III e do GPS IIIF ("Acompanhamento") (3). Os 11 satélites GPS lançados da Base Aérea de Vandenberg entre 1978 e 1985 eram conhecidos como satélites do Bloco I. O último satélite do Bloco I foi retirado no final de 1995. Seguiram-se os satélites do Bloco II. O primeiro deles foi lançado em 1989 e o último foi desativado em 2007 (4).

A utilização do GPS em veículos de frota começou em 1978, quando o satélite GPS experimental do Bloco I (criado pela Rockwell International) foi lançado ao espaço. Esta foi uma espécie de teste para a tecnologia GPS generalizada, e um sucesso. Esses satélites eram usados principalmente para fins militares, mas tinham alcance limitado – pois ainda não havia satélites GPS suficientes no espaço para o rastreamento generalizado de veículos (5). Para que um receptor possa ter sua localização tridimensional com os parâmetros de latitude, longitude e altitude em qualquer ponto do planeta Terra uma constelação com no mínimo 24 satélites é necessária (3).

Os satélites da constelação GPS estão dispostos em seis planos orbitais igualmente espaçados em torno da Terra. Cada plano contém quatro *slots* ocupados por satélites de linha de base. Essa disposição de 24 *slots* garante que os usuários possam visualizar pelo menos quatro satélites de praticamente qualquer ponto do planeta. A constelação GPS é ilustrada na Figura 1.1. A constelação GPS normalmente possui mais de 24 satélites em órbita para manter a cobertura mesmo que os satélites principais passem por manutenção ou

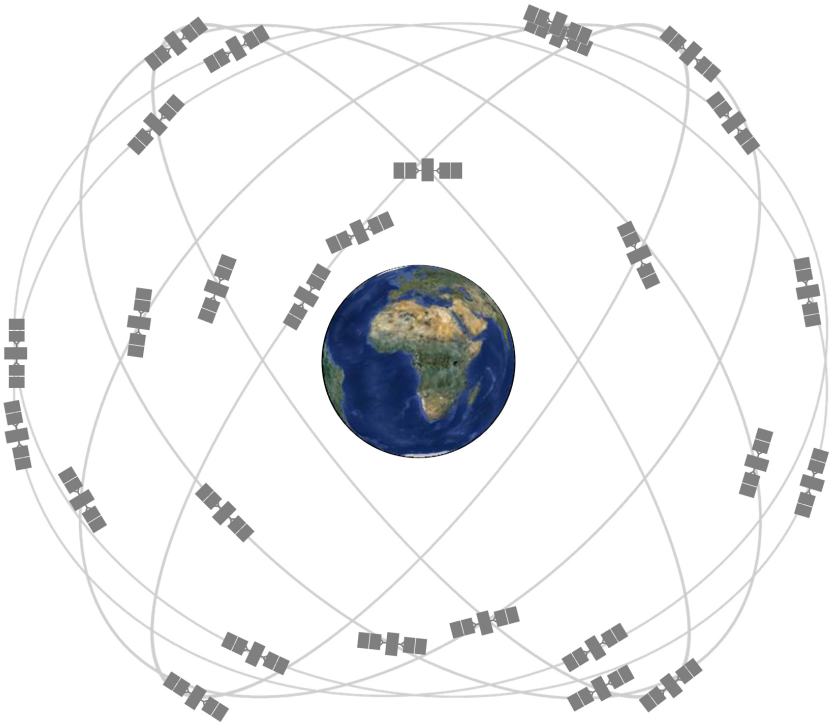

Figura 1.1: Constelação de satélites do Sistema de Posicionamento Global (GPS)
Fonte: U.S. Space Force - GPS.gov (3)

sejam desativados. Os satélites extras podem aumentar o desempenho do GPS, mas não são considerados parte da constelação principal (3).

Em junho de 2011, a Força Aérea dos Estados Unidos da América (EUA) concluiu com sucesso uma expansão da constelação GPS conhecida como configuração "*Expandable 24*". Três dos 24 *slots* foram expandidos e seis satélites foram reposicionados, de modo que três dos satélites extras passaram a fazer parte da linha de base da constelação. Como resultado, o GPS agora opera efetivamente como uma constelação de 27 *slots* com cobertura aprimorada na maior parte do mundo (3).

No início o GPS era usado para fins militares, ajudando as unidades militares a manter o controle de veículos importantes durante as missões. Foi só em 1996 que o presidente dos EUA, Bill Clinton, ajudou o público a ver o potencial do GPS para o mercado de massa - ele foi muito franco sobre as possibilidades do GPS tanto para militares como para civis, e emitiu uma diretiva política que desenvolveu um sistema de dupla utilização. Sistema de satélite GPS que poderá ser utilizado no interesse do público em geral. Esta mudança de política permitiu que o civil médio tivesse acesso à tecnologia GPS, incluindo os gestores de frota – que inicialmente viram os benefícios de poder manter o controle sobre a localização dos seus veículos (5).

Durante as décadas de 1980 e 1990, ocorreram duas grandes evoluções tecnológicas que deram origem aos sistemas de telemática de frota que temos hoje. Primeiro, a produção do computador pessoal permitiu que empresas de todos os tamanhos usassem essa tecnologia. Ela não era mais acessível apenas aos órgãos governamentais, mas as pequenas empresas agora podiam usar computadores para lidar com as informações da frota. Mais importante ainda, os computadores pessoais facilitaram o uso dessa tecnologia por pessoas com formação limitada em ciência da computação (6).

Outro grande avanço tecnológico na década de 1990 foi o uso da tecnologia GPS pelos consumidores. Isso, aliado ao uso da Internet, deu aos sistemas de telemática de frotas o maior impulso até hoje. Nos primórdios do rastreamento de frotas, para rastrear adequadamente uma frota, cada veículo tinha de ser habilitado com um dispositivo GPS de alto custo. A empresa era obrigada a pagar uma taxa mensal normalmente alta para usar o sistema de rastreamento por satélite. Embora úteis, esses primeiros sistemas eram difíceis de implementar, caros de usar e, às vezes, inconvenientes tanto para os motoristas quanto para o gerenciamento da frota. Assim, foram necessários vários anos para que o conceito se popularizasse. Nos primeiros dias, apenas as frotas grandes e ricas aproveitavam a tecnologia (6).

O governo dos EUA concedeu acesso à tecnologia GPS em 1993, o que significava que os motoristas comerciais poderiam finalmente usar esse método de mapeamento para o desenvolvimento de rotas. As frotas puderam implementar sistemas de navegação de veículos por GPS. Isso reduziu o impacto ambiental dos veículos graças à diminuição do tempo de direção e à melhoria das rotas para as frotas. Em meados dos anos 2000, a tecnologia de navegação por GPS evoluiu para sistemas de rastreamento. Isso se deveu, em parte, aos aprimoramentos das comunicações máquina a máquina (*Machine to Machine - M2M*), que é a predecessora da Internet das Coisas (*Internet of Things - IoT*). Juntamente com a tecnologia baseada em nuvem e os parâmetros do sensor, o rastreamento por GPS tornou-se cada vez mais preciso (6).

Os celulares e tablets progrediram junto com o uso da tecnologia de navegação GPS. Durante os anos 2000, os telefones celulares puderam executar processos de navegação e rastreamento por GPS usando aplicativos. Os veículos comerciais também foram equipados com sistemas computadorizados e plataformas de comunicação (6).

1.2 DEFINIÇÃO DO PROBLEMA

O processo de globalização mundial foi impulsionado em grande medida devido ao crescimento da Internet nas últimas décadas. Hoje existem mais de 5 bilhões de pessoas com acesso à Internet e junto com esse crescimento também houve o crescimento na quantidade de softwares e a consequente exploração de vulnerabilidades nestes.

Todo esse processo levou a necessidade de conectividade da população de forma geral, com forte dependência da utilização de aplicativos de celular para atendimento de uma série de serviços públicos e privados. Além disso, grande parte de dispositivos utilizados em residências, comércios ou indústrias também passou a ter conectividade à Internet.

Porém, apesar da disseminação e popularidade de tais sistemas, muitos não se utilizam de uma arquitetura robusta o suficiente com relação à segurança. O crescente número de ataques cibernéticos no mundo e as ameaças que podem decorrer através da exploração de vulnerabilidades em sistemas de informações, tornam muitos desses sistemas suscetíveis à ataques.

1.3 OBJETIVO

O objetivo geral deste trabalho é propor uma arquitetura para um rastreador veicular com criptografia de ponta-a-ponta utilizando tecnologias abertas, a fim de mitigar a exploração de vulnerabilidades e proteger informações sensíveis contra ataques cibernéticos.

Os objetivos específicos são discutir as vantagens e desvantagens de diferentes protocolos e arquiteturas utilizados na implementação de rastreadores veiculares comerciais, propor uma arquitetura para um rastreador veicular com foco na segurança que utilize tecnologias abertas, focando na segurança para mitigar a exploração de vulnerabilidades.

Além destes, com relação à gestão da infraestrutura, é prover toda a infraestrutura necessária para que o funcionamento do rastreador possa ser gerida de forma própria. No aspecto escalabilidade é planejar a arquitetura de modo que seja facilmente escalável, atendendo desde pequenas até grandes frotas de veículos.

1.4 ESTRUTURAÇÃO

A dissertação está estruturada no capítulo 2 pela fundamentação teórica sobre os rastreadores, sistema de posicionamento global (GPS), outros sistemas de posicionamento, protocolos comumente utilizados, segurança cibernética de forma geral e internet das coisas (IoT). No capítulo 3 são apresentados alguns trabalhos relacionados. No capítulo 4 é apresentada a proposta de uma arquitetura para um rastreador veicular com criptografia ponta-a-ponta, além do protótipo desenvolvido durante o trabalho. No capítulo 5 são apresentados alguns resultados e finalmente no capítulo 6 a conclusão com ideias para trabalhos futuros a partir deste.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 RASTREADORES

Os rastreadores são dispositivos utilizados para a localização de objetos, veículos ou pessoas em tempo real. Utilizam tecnologias como a *Global Positioning System (GPS)*, *Global System for Mobile Communications (GSM) – 2G, 3G, 4G ou 5G*, além de vários protocolos associados as comunicações da Internet. Estes tornam-se cada vez mais importantes em diversas áreas, incluindo a logística, o transporte de cargas, a segurança pessoal e automotiva. Além disso estão cada vez mais sendo utilizados pelas forças de segurança pública para otimização de atendimento a ocorrências. Integrados com sistemas de informação e apoio a decisão, estes rastreadores são fundamentais para maior eficiência nestes atendimentos.

Na área da logística, os dispositivos de rastreamento são empregados para acompanhar a posição das mercadorias em tempo real. Isso possibilita que as empresas de transporte e logística monitorem seus veículos e cargas. A utilização desses dispositivos permite às empresas otimizar o trajeto das entregas, diminuir despesas com manutenção, combustível, horas extras e penalidades, ao mesmo tempo em que reduz o risco de acidentes rodoviários.

Na esfera da segurança pessoal, dispositivos de rastreamento podem ser empregados para acompanhar a localização de crianças, idosos e indivíduos com necessidades especiais, proporcionando maior proteção para seus familiares e responsáveis.

Na área automotiva, o rastreamento começou a se destacar no Brasil nos anos 1990, quando empresas buscavam fornecer serviços para localizar e recuperar veículos roubados ou furtados. Naquela época, as tecnologias predominantes eram o rastreamento via satélite em órbita alta e a comunicação via *pager*. Poucas empresas operavam no mercado nacional, com a Autotrac sendo pioneira (7). Uma órbita alta é aquela com uma distância superior a 35.786 km da crosta terrestre (8). Um *pager* é um pequeno dispositivo eletrônico portátil que emite um som para avisar que alguém quer que você entre em contato. São dispositivos que no geral só recebiam informações de texto diretamente na tela, não possuindo características de comunicação bidirecional. Exemplos de aplicação dos *pagers* na década de 1990 incluem a utilização por empresas como forma de enviar mensagens a trabalhadores plantonistas quando precisavam ser acionados fora do horário comercial padrão. Estes dispositivos acabaram caindo em desuso a partir do surgimento dos primeiros aparelhos celulares.

Quando um satélite de órbita alta atinge exatamente 42.164 quilômetros do centro da Terra, ou seja, cerca de 36.000 quilômetros da superfície terrestre, ele entra em uma espécie de "ponto ideal" no qual sua órbita coincide com a rotação da Terra. Como o satélite orbita na mesma velocidade em que a Terra está girando, o satélite parece permanecer no lugar em uma única longitude, embora possa se deslocar de norte a sul. Essa órbita especial e elevada da Terra é chamada de geossíncrona. Um satélite em uma órbita geossíncrona circular diretamente sobre o equador (excentricidade e inclinação iguais a zero) terá uma órbita geoestacionária que não se move em relação ao solo. Ela está sempre diretamente sobre o mesmo lugar na superfície da Terra (8).

De forma diferente dos satélites de órbita alta, os satélites de órbita terrestre média se movem mais rapidamente, pois estão mais próximos da Terra. Duas órbitas terrestres médias são notáveis: a órbita semi-síncrona e a órbita Molniya. A órbita semi-síncrona é uma órbita quase circular (baixa excentricidade) a 26.560 quilômetros do centro da Terra ou cerca de 20.200 quilômetros acima da superfície terrestre. Um satélite a essa altura leva 12 horas para completar uma órbita. À medida que o satélite se move, a Terra gira sob ele. Em 24 horas, o satélite cruza os mesmos dois pontos no equador todos os dias. Essa órbita é consistente e altamente previsível. É a órbita usada pelos satélites do Sistema de Posicionamento Global (*GPS*) (8).

Durante os anos 2000, com a popularização de tecnologias como o *GPS* e a ampla expansão da rede *GSM* no Brasil, o mercado de rastreamento automotivo cresceu consideravelmente. Novas empresas surgiram oferecendo serviços de monitoramento e recuperação de veículos. O rastreamento passou a ser adotado não apenas por empresas de segurança e transporte, mas também por motoristas particulares preocupados com a segurança de seus veículos (7).

Um dos fatores mais importantes para a popularização do GPS no mundo foi a desativação da Disponibilidade Seletiva (*Selective Availability - SA*). O presidente dos EUA Ronald Reagan anunciou que o sistema GPS seria disponibilizado para uso civil a partir de 16 de setembro de 1983 (9); no entanto, inicialmente esse uso civil foi limitado a uma precisão média de 100 metros (cerca de 330 pés) pelo uso da Disponibilidade Seletiva (SA), um erro deliberado introduzido nos dados do GPS que os receptores militares podiam corrigir.

Com o aumento do uso civil do GPS, houve uma pressão crescente para remover esse erro. O sistema SA foi temporariamente desativado durante a Guerra do Golfo, pois a escassez de unidades militares de GPS significava que muitos soldados dos EUA estavam usando unidades civis de GPS enviadas de casa. Na década de 1990, os sistemas de GPS diferencial da Guarda Costeira dos EUA, da Administração Federal de Aviação e de agências semelhantes em outros países começaram a transmitir correções locais de GPS, reduzindo o efeito da degradação do SA e dos efeitos atmosféricos (que os receptores militares também corrigiam). As forças armadas dos EUA também desenvolveram métodos para realizar interferência local no GPS, o que significa que a capacidade de degradar globalmente o sistema não era mais necessária. Como resultado, o presidente dos Estados Unidos, Bill Clinton, assinou um projeto de lei determinando que a Disponibilidade Seletiva fosse desativada em 1º de maio de 2000 (10); e, em 2007, o governo dos EUA anunciou que a próxima geração de satélites GPS não incluiria esse recurso.

Os avanços tecnológicos e as novas demandas do sistema existente levaram a esforços para modernizar o GPS e implementar a próxima geração de satélites GPS do Bloco III e o Sistema de Controle Operacional de Próxima Geração (*Next Generation Operational Control System - OCX*) (11), que foi autorizado pelo Congresso dos EUA em 2000. Quando a disponibilidade seletiva foi descontinuada, o GPS tinha uma precisão de cerca de 5 metros (16 pés). Os receptores de GPS que usam a banda L5 têm uma precisão muito maior, de 30 centímetros (12 polegadas), enquanto os receptores para aplicações de ponta, como engenharia e levantamento topográfico, têm precisão de 2 cm (3⁄4 polegada) e podem até fornecer precisão submilimétrica com medições de longo prazo (10) (12). Dispositivos de consumo, como smartphones, podem ter precisão de 4,9 m (16 pés) ou melhor quando usados com serviços de assistência, como posicionamento por Wi-Fi (13). Em julho de 2023, 18 satélites GPS transmitiam sinais L5, que são considerados

pré-operacionais antes de serem transmitidos por um conjunto completo de 24 satélites em 2027.

Atualmente, o mercado de rastreamento automotivo no Brasil está consolidado, com diversas empresas fornecendo serviços de monitoramento, recuperação e gestão de frotas. As tecnologias de geolocalização e comunicação continuam avançando, com o uso de aprendizado de máquina para melhorar os sistemas de rastreamento e prevenir roubos e furtos de veículos, além de gerir frotas e ativos (7).

Em suma, os rastreadores são dispositivos que permitem monitorar a localização de objetos, veículos ou pessoas, proporcionando maior eficiência e segurança em várias áreas, como logística, segurança pessoal e automotiva. Com o avanço das tecnologias, o mercado de rastreamento tem expandido e oferecido soluções cada vez mais eficazes para empresas e usuários (7).

2.1.1 SISTEMAS GLOBAIS DE NAVEGAÇÃO POR SATÉLITE

Um sistema de navegação por satélite ou satnav é um sistema que usa satélites para fornecer posicionamento geo-espacial através do uso de satélites artificiais. Quando um sistema de navegação por satélite possui a capacidade de oferecer posicionamento em qualquer ponto da superfície terrestre, adota-se a nomenclatura de Sistema Global de Navegação por Satélite (*Global Navigation Satellite System - GNSS*). A partir de 2024, quatro sistemas globais estão operacionais: o Sistema de Posicionamento Global (*GPS*) dos Estados Unidos, o Sistema Global de Navegação por Satélite (*GLONASS*) da Rússia, o Sistema de Navegação por Satélite BeiDou (*BDS*) da China e o Galileo da União Europeia.

O Sistema de Posicionamento Global (*GPS*) é um sistema de propriedade dos EUA que fornece aos usuários serviços de posicionamento, navegação e cronometragem (PNT). Esse sistema consiste em três segmentos: o segmento espacial, o segmento de controle e o segmento do usuário. A Força Espacial dos EUA desenvolve, mantém e opera os segmentos espacial e de controle (14).

Assim como a Internet, o GPS é um elemento essencial da infraestrutura global de informações. A natureza gratuita, aberta e confiável do GPS levou ao desenvolvimento de centenas de aplicativos que afetam todos os aspectos da vida moderna. Atualmente, a tecnologia GPS está presente em tudo, desde telefones celulares e relógios de pulso até escavadeiras, contêineres de transporte e caixas eletrônicos (15).

O GPS aumenta a produtividade em vários setores da economia, incluindo agricultura, construção, mineração, levantamento topográfico, entrega de encomendas e gerenciamento logístico da cadeia de suprimentos. As principais redes de comunicação, sistemas bancários, mercados financeiros e redes de energia dependem muito do GPS para a sincronização precisa do tempo. Alguns serviços sem fio não podem operar sem ele (15).

O GPS salva vidas ao evitar acidentes de transporte, ajudar nos esforços de busca e resgate e acelerar a prestação de serviços de emergência e socorro em caso de desastres. O GPS é vital para o Sistema de Transporte Aéreo de Próxima Geração (*NextGen*), que aprimorará a segurança de voo e aumentará a capacidade do espaço aéreo. O GPS também avança em objetivos científicos, como previsão do tempo, monitoramento de terremotos e proteção ambiental (15).

2.1.2 PADRÃO NMEA-0183

NMEA é um acrônimo para *National Marine Electronics Association* (16). A Associação Nacional de Eletrônica Marítima é uma organização comercial mundial que define padrões de interface de eletrônica marítima, treinamento de instaladores de eletrônica marítima e promove sua conferência e exposição anual de eletrônica marítima. A NMEA e seus membros têm o compromisso de aprimorar a tecnologia e a segurança da eletrônica marítima por meio de treinamento de instaladores e padrões de interface. Os membros da NMEA promovem o profissionalismo no setor de eletrônicos marítimos. Os treinamentos e as certificações de instaladores da NMEA são reconhecidos por muitos dos principais fabricantes de eletrônicos para instalação, suporte e garantia.

O NMEA existia bem antes do GPS ser inventado. De acordo com o site da NMEA (16), a associação foi formada em 1957 por um grupo de revendedores eletrônicos para criar uma melhor comunicação com os fabricantes. Hoje, no mundo do GPS, o NMEA é um formato de dados padrão suportado por todos os fabricantes de GPS, assim como o ASCII é o padrão para caracteres de computador digital no mundo dos computadores (17).

O objetivo do NMEA é dar aos usuários de equipamentos a facilidade de comunicação entre *hardware* e *software*. Os dados GPS formatados em NMEA facilitam a vida dos desenvolvedores de software para escrever rotinas para uma ampla variedade de receptores GPS, ao invés vez de ter que escrever algo personalizada para cada equipamento (17).

O padrão de interface NMEA 0183 (16) é usado em todo o mundo em vários segmentos. O padrão define os requisitos de sinais elétricos, o protocolo e o tempo de transmissão de dados e os formatos específicos de sentenças para um barramento de dados serial de 4800 *baud*. Cada barramento pode ter apenas um locutor, mas muitos ouvintes. Esse padrão destina-se a oferecer suporte à transmissão de dados seriais unidirecionais de um único locutor para um ou mais ouvintes. Esses dados estão no formato ASCII imprimível e podem incluir informações como hora, posição, velocidade, profundidade da água, dentre outras. O padrão de interface NMEA 0183 é um documento protegido por direitos autorais e sua última versão é 4.30 publicada em dezembro de 2023 (16).

2.2 SEGURANÇA CIBERNÉTICA E INTERNET DAS COISAS

Os dispositivos do tipo Internet das Coisas (*Internet of Things - IoT*) são dispositivos que possuem conexão com uma rede, não necessariamente a Internet, e que sejam endereçáveis (acessíveis indistintamente) nesta rede. Exemplos de dispositivos IoT incluem relógios inteligentes (*smartwatches*), *drones*, sensores industriais, centrais multimídia de veículos, além do telefone celular, que provavelmente é o dispositivo mais utilizado diretamente pelas pessoas.

Segundo previsão do portal Statista, a quantidade tende a ultrapassar 29 bilhões até 2030 (18). Outras fontes citam a previsão de um número ainda maior com expectativa de 75 bilhões de dispositivos até 2025 (19). À medida que o número de dispositivos conectados aumenta, aumenta também a necessidade de proteção contra várias ameaças. Os primeiros ataques de malware em grande escala a dispositivos IoT

foram registados em 2008 e, desde então, este número só aumentou (20). Uma análise das ameaças à IoT em 2023 realizada por (20) mostrou que serviços para ataques a dispositivos IoT são oferecidos na *dark web*. Além disso também foram identificados os dois principais tipos de ataques à dispositivos IoT: a aplicação de força bruta para a quebra de senhas fracas e a exploração de vulnerabilidades nos serviços de rede. O Telnet, o protocolo de texto, nativamente sem segurança, foi o principal alvo da força bruta. Uma quebra de senha de acesso bem sucedida permite aos atacantes executar comandos remotamente além da possibilidade de injeção de *malware*. Ataques de força bruta a serviços que utilizam SSH, que possui criptografia nativamente, podem produzir resultados parecidos. Porém, são necessários mais recursos para atacar o SSH.

Para realizar uma análise mais detalhada dos ataques, a ***Kaspersky*** criou *honeypots* para verificar os tipos de tentativas de ataque e a distribuição percentual destas tentativas. No primeiro semestre de 2023, 97,91% das tentativas de força bruta registadas visavam o Telnet e apenas 2,09% o SSH. A maioria dos dispositivos infectados que realizaram estes ataques estavam localizados na China, na Índia e nos Estados Unidos, enquanto a China, o Paquistão e a Rússia foram os países que atacaram mais ativamente (21). O Brasil aparece como o quinto país do qual partiram os ataques conforme Figura 2.1.

Os ataques de força bruta são bastante comuns, uma vez que os serviços Telnet e SSH executados em dispositivos IoT utilizam normalmente senhas predefinidas amplamente conhecidas. Os desenvolvedores tendem a deixar estas senhas inalteradas. Além disso, muitos dispositivos IoT têm senhas principais inalteráveis definidas pelos fabricantes.

Outra forma de comprometer um dispositivo é aproveitar as vulnerabilidades dos serviços que nele são executados. A injeção de código malicioso nas requisições HTTP enviadas para a interface Web é a forma mais comum de explorar as vulnerabilidades. As consequências destes ataques podem gerar grande impacto, como no caso de uma vulnerabilidade na implementação do protocolo TR-064 utilizado pelos provedores de acesso à Internet (*Internet Service Providers - ISPs*) para automatizar a configuração de dispositivos na LAN. A falha de segurança permitiu a transmissão não autenticada de pacotes TR-064, resultando na proliferação do *malware* Mirai.

O *malware* Mirai, nome derivado da palavra japonesa para "futuro" é um *malware* que transforma dispositivos em rede que executam o Linux em *bots* controlados remotamente que podem ser usados como parte de uma *botnet* em ataques de rede em larga escala. A definição para *bot*, segundo o dicionário Oxford, é um programa autônomo na Internet ou em outra rede que pode interagir com sistemas ou usuários. A palavra *bot* é um encurtamento da palavra *robot* do inglês. Ele tem como alvo principal dispositivos *online*, como câmeras IP e roteadores domésticos (22). O botnet Mirai foi encontrado pela primeira vez em agosto de 2016 (23) pelo MalwareMustDie (24), um grupo de pesquisa de malware de chapéu branco, e foi usado em alguns dos maiores e mais perturbadores ataques de negação de serviço distribuído (DDoS).

Independentemente da técnica de comprometimento, os dispositivos IoT podem ser atacados tanto pelos próprios servidores infectados como por malware através da chamada auto-propagação, em que maliciosos procuram dispositivos vulneráveis online e implantam cópias nos mesmos através de diversos meios. Neste último caso, o ataque pode também ter origem num dispositivo IoT infectado anteriormente.

São diversos os objetivos e tipos de *malware* que atacam a IoT. Podem explorar o hardware infectado como uma ferramenta para lançar ciberataques, camuflar tráfego malicioso, aproveitar os recursos dos

dispositivos para mineração de criptomoedas ou exigir um resgate para restaurar o acesso ao dispositivo. Alguns podem atacar qualquer dispositivo IoT, enquanto outros atacam apenas certos tipos de *hardware* que são capazes de servir os seus objetivos. Alguns dos tipos de maliciosos são *botnets DDoS*, *ransomware*, mineradores de criptomoedas, alteradores de DNS e *proxy bots*.

Além de atacarem a IoT, os criminosos oferecem os seus serviços no mercado da *dark web*. Sendo assim, a maioria dos dispositivos ligados, incluindo os que se encontram em ambientes industriais, continuam a ser vulneráveis devido à utilização de senhas predefinidas e à presença de vulnerabilidades nos dispositivos, algumas das quais os fornecedores nunca chegam a corrigir. Os fornecedores de dispositivos IoT domésticos e industriais deveriam adotar uma abordagem responsável em relação à cibersegurança dos produtos e introduzir medidas de proteção na fase de concepção do produto. De modo geral, há recomendação para substituição de todas as senhas predefinidas por senhas fortes e distintas para cada dispositivo, além de atualizações de segurança regulares (21).

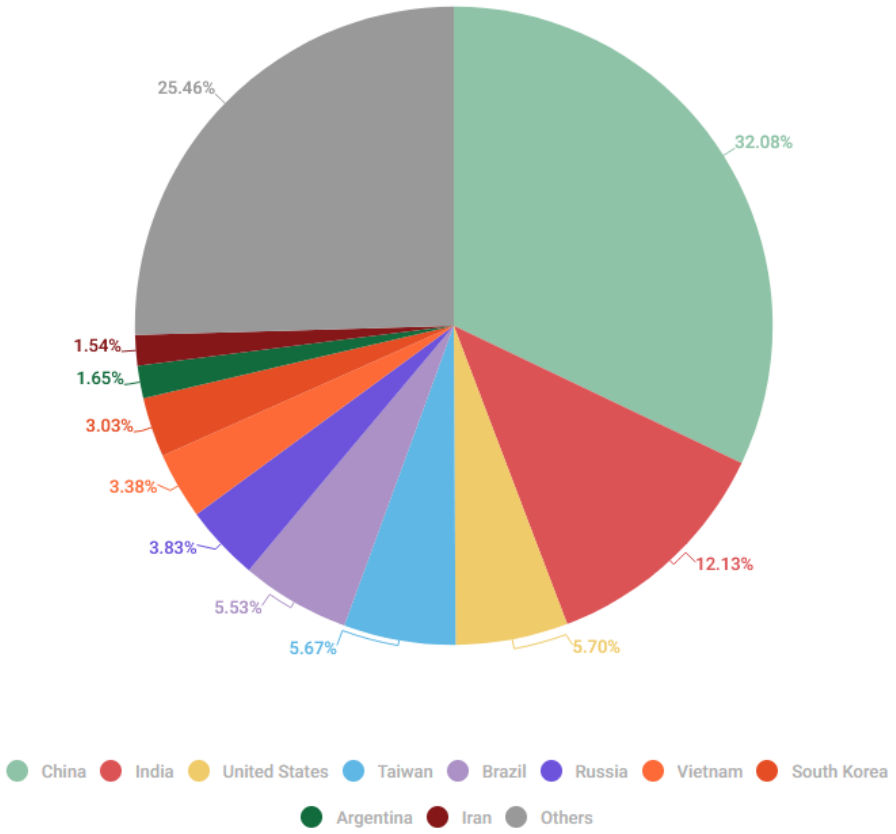

Figura 2.1: Origem dos ataques aos honeypots Kaspersky no 1º semestre de 2023
Fonte: SECURELIST by Kaspersky <https://securelist.com/iot-threat-report-2023/110644/>

A Fundação OWASP (*Open Web Application Security Project*), que é uma organização dedicada à análise e disseminação de temas, documentos e ferramentas relacionadas à segurança de software a nível global, publicou em 2018 uma lista (25) que indica as 10 principais vulnerabilidades associadas a dispositivos IoT. Estas devem ser sempre consideradas durante a fase de projeto pelos desenvolvedores que criam ou gerenciam sistemas IoT (26). As principais vulnerabilidades são:

I. **Senhas fracas, previsíveis ou dentro do código**

Uso de senhas que não podem ser modificadas e que estão publicamente disponíveis ou são fáceis de adivinhar através da utilização da técnica de força bruta, e até mesmo de *backdoors* em *firmware* ou software cliente que permitem obter acesso não autorizado a sistemas, aproveitando-se dessas senhas vulneráveis.

II. **Serviços de rede inseguros**

Serviços de rede inseguros e desnecessários em execução no próprio dispositivo, especialmente aqueles expostos à Internet, que comprometem a confidencialidade, autenticidade ou disponibilidade de informações ou permitem o controle não autorizado de forma remota.

III. **Interfaces de ecossitemas inseguras**

Problemas de segurança em interfaces web, móveis, na nuvem, ou *API* de *backend* em ecossistemas que estão fora dos dispositivos e que permitem que tanto os dispositivos como certos componentes relacionados possam ser comprometidos.

IV. **Falta de mecanismos de atualização seguros**

A falta de um sistema simples para atualizar o dispositivo de forma segura. Isso inclui: falta de validação do *firmware* no dispositivo, falta de segurança no envio (tráfego não criptografado), falta de mecanismos para evitar voltar atrás e falta de notificações sobre alterações de segurança devido às atualizações.

V. **Uso de componentes inseguros ou obsoletos**

Uso de componentes/bibliotecas de software obsoletos e/ou inseguros que podem permitir que o dispositivo seja comprometido. Isso inclui a personalização insegura da plataforma do sistema operacional e o uso de software de terceiros ou componentes de hardware de uma cadeia de suprimentos comprometida.

VI. **Proteção da privacidade insuficiente**

Informações pessoais do usuário armazenadas no dispositivo ou no ambiente ao qual o dispositivo está conectado, usadas de maneira insegura, inadequada ou sem permissão.

VII. **Transferência e armazenamento de dados de maneira insegura**

Falta de criptografia ou controle de acesso para dados confidenciais que estão dentro do ecossistema, incluindo dados em repouso, em trânsito ou em processamento.

VIII. **Falta de controles de gerenciamento**

Falta de suporte de segurança em dispositivos liberados para produção, incluindo gerenciamento de ativos, gerenciamento de atualizações, desarme seguro, monitoramento de sistemas e recursos de resposta.

IX. **Configuração insegura por padrão**

Dispositivos ou sistemas com configurações padrão pouco seguras ou sem a possibilidade de tornar o sistema mais seguro, aplicando restrições com base nas alterações de configuração.

X. **Falta de proteção física**

Falta de medidas de proteção física, permitindo que possíveis invasores obtenham informações confidenciais que possam ajudar em um futuro ataque remoto ou assumir o controle local do dispositivo.

2.3 PROTOCOLO MQTT

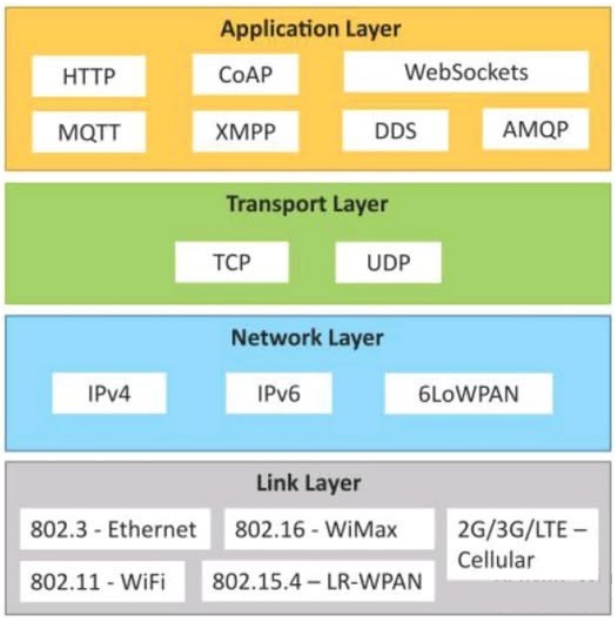

Figura 2.2: MQTT dentro da pilha de protocolos Internet
Fonte: Innokrea <https://www.innokrea.com/internet-of-things-iot-part-2/>

2.3.1 HISTÓRICO

O protocolo *Message Queuing Telemetry Transport* (MQTT) (27) foi criado em 1999 por Andy Stanford-Clark (IBM) e Arlen Nipper (Arcom, agora parte da Eurotech) como uma solução para a comunicação em sistemas de monitoramento de oleodutos na indústria do petróleo. O objetivo inicial era criar um protocolo leve e eficiente que pudesse funcionar em ambientes com baixa largura de banda e alta latência. Desde então, o MQTT foi adotado amplamente no campo da Internet das Coisas (IoT) devido à sua simplicidade e capacidade de operar em condições de rede com baixa largura de banda, alta latência, conexões intermitentes/instáveis ou mesmo com alta taxa de perda de pacotes. Ele deve ser executado sobre um protocolo de transporte que forneça conexões ordenadas, sem perdas e bidirecionais - normalmente o protocolo TCP/IP. É um protocolo que está na camada de aplicação da pilha de protocolos Internet, conforme Figura 2.2, que apresenta alguns protocolos Internet. A versão mais recente do protocolo atualmente é a versão 5.0 (28).

O MQTT é um padrão aberto da Organização para o Avanço dos Padrões de Informações Estruturadas (*Organization for the Advancement of Structured Information Standards - OASIS*) e uma recomendação da ISO (ISO/IEC 20922). A OASIS é um consórcio sem fins lucrativos que trabalha no desenvolvimento, na convergência e na adoção de projetos - tanto de padrões abertos quanto de código aberto - para segurança

de computadores, *blockchain*, Internet das Coisas (IoT), gerenciamento de emergências, computação em nuvem, intercâmbio de dados legais, energia, tecnologias de conteúdo e outras áreas (29).

O MQTT é um protocolo altamente eficiente e adequado para comunicação em tempo real em ambientes com recursos limitados. Sua flexibilidade e eficiência o tornaram um dos protocolos mais adotados em sistemas de IoT e automação, com uma ampla gama de aplicações que vão desde a automação residencial até o monitoramento industrial.

2.3.2 PRINCIPAIS CARACTERÍSTICAS

I. **Publicação (Pub) / Assinatura (Sub):** O MQTT usa um modelo de comunicação baseado em tópicos, onde os dispositivos podem se inscrever para receber informações (assinatura) ou publicar dados (publicação) em tópicos específicos.

II. **Leveza e eficiência:** O protocolo foi projetado para ser eficiente em termos de largura de banda e consumo de energia, sendo ideal para dispositivos com recursos limitados.

III. **Confiabilidade:** O MQTT suporta três níveis de Qualidade do Serviço (*Quality of Service - QoS*) para garantir a entrega das mensagens, mesmo em redes instáveis.

- QoS 0: Mensagem entregue no máximo uma vez.
- QoS 1: Mensagem entregue pelo menos uma vez.
- QoS 2: Mensagem entregue exatamente uma vez.

IV. **Manutenção de Conexão:** Usa um mecanismo de *keep-alive* para garantir que a conexão entre o cliente e o servidor esteja ativa.

V. **Mensagens pequenas:** As mensagens são compactas, o que as torna rápidas de transmitir e reduzir o consumo de banda.

2.3.3 PRINCIPAIS BENEFÍCIOS

I. **Eficiência em Redes de Baixa Largura de Banda:** O MQTT é otimizado para operar em condições de rede limitadas, com baixa latência e pequeno consumo de dados.

II. **Baixo Consumo de Energia:** Ideal para dispositivos IoT alimentados por baterias, pois as conexões são mantidas com pouca troca de dados, economizando energia.

III. **Escalabilidade:** É fácil escalar a infraestrutura MQTT para suportar grandes quantidades de dispositivos e comunicação em tempo real.

IV. **Segurança:** Embora o MQTT por padrão não forneça segurança, ele pode ser implementado sobre TLS/SSL para garantir a criptografia da comunicação.

V. **Desempenho:** O protocolo é rápido e leve, permitindo a troca de informações de forma eficiente e com baixa sobrecarga.

2.3.4 APLICAÇÕES DO MQTT

I. **Internet das Coisas (IoT):** É uma das soluções mais populares para conectar dispositivos IoT, como sensores, dispositivos domésticos inteligentes, *wearables* e automação industrial.

II. **Automação Residencial e Comercial:** Dispositivos como termostatos inteligentes, luzes e câmeras de segurança usam MQTT para se comunicar com *hubs* e aplicativos.

III. **Monitoramento e Controle Remoto:** Usado em sistemas de monitoramento industrial, como sensores de temperatura, umidade, pressão, e outros dispositivos conectados.

IV. **Veículos Conectados:** Na telemetria de carros e sistemas de transporte, MQTT pode ser utilizado para enviar dados em tempo real.

V. **Saúde e Bem-Estar:** Para monitoramento remoto de pacientes, como sensores de sinais vitais, onde a comunicação eficiente e confiável é crucial.

VI. **Sistemas de Energia Inteligente:** Usado para otimizar o monitoramento e controle de redes de energia elétrica, como em *smart grids* e sistemas de energia solar.

VII. **Cidades Inteligentes:** Para integração de sistemas como semáforos, câmeras de vigilância e sensores urbanos, permitindo a comunicação entre diversos dispositivos.

2.4 OUTROS PROTOCOLOS PARA IOT

Além do MQTT, existem outros protocolos que também são amplamente utilizados para comunicação em IoT. Cada um desses protocolos tem suas características, benefícios e desvantagens dependendo do tipo de aplicação, requisitos de rede, recursos do dispositivo, e outros fatores.

2.4.1 CoAP (*Constrained Application Protocol*)

O CoAP é um protocolo de aplicação projetado para dispositivos e redes com recursos limitados (dispositivos "*constrained*"), como aqueles encontrados em IoT. Ele segue o modelo de comunicação *request/-response* semelhante ao HTTP, mas é mais eficiente em termos de consumo de largura de banda e energia.

As principais desvantagens em relação ao MQTT são:

I. **Modelo de Comunicação:** O CoAP segue o modelo *request/response*, o que pode ser menos eficiente para aplicações de *pub/sub* em tempo real, como é o caso do MQTT. Isso significa que, em muitos casos, é necessário um dispositivo (cliente) solicitar dados do servidor, o que pode resultar em maior latência e uso de energia, especialmente em sistemas onde as atualizações são frequentes e em tempo real.

II. **Escalabilidade:** O MQTT é mais adequado para cenários com múltiplos dispositivos publicando dados em tópicos, o que é mais fácil de gerenciar em redes grandes e distribuídas. CoAP, por sua vez, pode ter dificuldades em escalar para cenários com muitos dispositivos interagindo simultaneamente.

III. **Mensagens com Confirmação:** O CoAP usa confirmações de mensagens, o que pode introduzir latência adicional e consumo de largura de banda, enquanto o MQTT oferece um controle mais flexível e eficiente com seus níveis de *QoS (Quality of Service)*.

2.4.2 HTTP (*Hypertext Transfer Protocol*)

O HTTP é um dos protocolos mais utilizados na comunicação web. No contexto de IoT, pode ser usado para enviar dados entre dispositivos e servidores. O HTTPS é a versão segura, com criptografia via SSL/TLS.

As principais desvantagens em relação ao MQTT são:

I. **Sobrecarregado:** O HTTP é mais pesado em comparação com o MQTT, especialmente em termos de *overhead* de comunicação. Ele requer a troca de cabeçalhos grandes e repetidos para cada requisição, o que não é eficiente para dispositivos IoT com recursos limitados.

II. **Sem Suporte nativo para Pub/Sub:** O HTTP não tem um modelo de *pub/sub* nativo, o que o torna ineficiente para aplicações que precisam de atualizações em tempo real (como no caso do MQTT). Para implementar um comportamento semelhante, seria necessário usar técnicas como *polling* (consultas regulares) ou *long polling*, que são mais ineficientes.

III. **Latência e Eficiência:** No HTTP, a comunicação é baseada em uma abordagem *request/response*, o que pode resultar em maior latência. Além disso, o HTTP exige o estabelecimento de uma conexão para cada requisição, o que consome mais recursos e tempo.

2.4.3 XMPP (*Extensible Messaging and Presence Protocol*)

O XMPP é um protocolo de comunicação baseado em XML que permite comunicação em tempo real. Originalmente foi projetado para mensagens instantâneas, mas também é usado em IoT para comunicação entre dispositivos.

As principais desvantagens em relação ao MQTT são:

I. **Complexidade e *Overhead***: O XMPP é mais complexo e envolve mais *overhead* em comparação ao MQTT, principalmente devido ao uso de XML, o que resulta em mensagens maiores e maior consumo de recursos (memória e largura de banda).

II. **Escalabilidade:** Embora o XMPP suporte comunicação em tempo real, a implementação e o gerenciamento de múltiplos tópicos (como no modelo *pub/sub* do MQTT) podem ser mais complexos.

III. **Latência e Eficiência:** O XMPP não foi projetado com a eficiência em redes com baixa largura de banda ou alta latência em mente, o que pode torná-lo menos eficiente para aplicações IoT com dispositivos limitados.

2.4.4 AMQP (*Advanced Message Queuing Protocol*)

O AMQP é um protocolo de mensagens robusto que oferece uma solução de troca de mensagens segura, confiável e eficiente. Ele é usado em sistemas de mensagens corporativas e é altamente escalável.

As principais desvantagens em relação ao MQTT são:

I. **Complexidade:** O AMQP é mais complexo e pesado em comparação com o MQTT, o que pode não ser ideal para dispositivos com recursos limitados, como em muitos sistemas IoT. Ele exige mais configurações e uma infraestrutura mais robusta.

II. **Sobrecarregado para IoT:** O AMQP é frequentemente excessivo para muitos cenários IoT, especialmente para dispositivos com baixa largura de banda e capacidade de processamento. O MQTT, por ser mais simples e leve, é preferido quando a eficiência de recursos é uma prioridade.

III. **Desempenho:** O AMQP pode ser mais lento em comparação ao MQTT, especialmente em redes com baixa latência e alta variação de tráfego.

2.4.5 DDS *(Data Distribution Service)*

O DDS é um protocolo de *middleware* em tempo real projetado para sistemas distribuídos de alto desempenho, como em aplicações críticas de IoT, automação industrial e sistemas embarcados.

As principais desvantagens em relação ao MQTT são:

I. **Complexidade e Custo:** O DDS é mais complexo e, geralmente, envolve custos mais elevados de implementação e manutenção. Ele é projetado para sistemas distribuídos de grande escala e não é tão eficiente para aplicações mais simples de IoT.

II. **Sobrecarregado para IoT Pequeno:** Em comparação com o MQTT, o DDS pode ser excessivo para dispositivos IoT pequenos ou com recursos limitados, pois exige maior capacidade de processamento e memória.

3 TRABALHOS RELACIONADOS

Diversas propostas de rastreadores veiculares foram apresentadas ao longo do tempo, conforme ilustrado em (30), (31) e (32). No entanto, essas soluções compartilham uma limitação significativa: elas não priorizam a segurança de maneira abrangente, especialmente nas camadas mais críticas da arquitetura de sistemas, como a proteção dos dados durante a transmissão, armazenamento e manipulação. Além disso, muitas dessas propostas utilizam protocolos desatualizados e vulneráveis, que podem ser facilmente explorados por agentes maliciosos. Embora os rastreadores veiculares em si atendam às funções básicas de monitoramento e localização, a falta de uma abordagem robusta de segurança torna essas soluções suscetíveis a diferentes tipos de ataques cibernéticos, como interceptação de dados e manipulação de informações.

Uma das questões mais preocupantes nas soluções de rastreamento veicular existentes é a utilização do *(Short Message Service - SMS)* como o principal método para a transferência de dados de localização entre os dispositivos de rastreamento e os servidores. De acordo com (33), o SMS apresenta falhas de segurança significativas, pois os dados são transmitidos em texto claro, sem qualquer forma de criptografia ou autenticação. Isso torna o sistema vulnerável a interceptações e alterações durante o processo de comunicação. Diversos estudos e autores, como (34), (35) e (36), ressaltam os riscos associados ao uso do SMS como meio de comunicação em sistemas que envolvem dados sensíveis. Embora o SMS tenha sido uma solução popular devido à sua simplicidade e baixo custo, ele não oferece a robustez necessária para proteger informações de rastreamento veicular, que muitas vezes podem envolver dados críticos sobre a localização e a segurança dos veículos e seus ocupantes.

Embora o SMS não deva ser completamente descartado como uma solução de transmissão de dados, especialmente em situações em que a implementação de uma camada de segurança adicional não seja viável, é fundamental que medidas de segurança complementares sejam adotadas. Autores como (37), (38) e (39) discutem abordagens para melhorar a segurança da comunicação via SMS, incluindo a utilização de protocolos de segurança adicionais, como a criptografia *end-to-end*, para proteger os dados em trânsito. No entanto, mesmo com essas medidas, a dependência do SMS em sistemas de rastreamento veicular representa uma fragilidade, principalmente considerando a evolução das ameaças cibernéticas.

Além disso, outra abordagem comum nas soluções de rastreamento veicular é o uso de microcontroladores, com destaque para as plataformas baseadas no Arduino, como observado em (40), (41) e (31). Embora essas soluções atendam aos requisitos funcionais básicos de monitoramento, elas frequentemente negligenciam aspectos essenciais de segurança, principalmente no que diz respeito à proteção dos dados durante a transmissão. Em todos os sistemas analisados, o Protocolo de Transferência de Hipertexto (HTTP) é utilizado para a comunicação entre o rastreador e os servidores. O HTTP, por ser um protocolo sem criptografia, torna a transmissão de dados vulnerável a ataques do tipo *Man-in-the-middle*, onde um atacante pode interceptar e modificar as mensagens trocadas entre o dispositivo e o servidor. A solução mais comum para esse problema seria a adoção do Protocolo de Transferência de Hipertexto Seguro (HTTPS), que utiliza uma camada adicional de segurança, empregando o protocolo *Transport Layer Security (TLS)* para criptografar os dados durante o trânsito.

Entretanto, apesar do HTTPS representar uma solução mais segura em relação ao HTTP, o protocolo

TLS é amplamente considerado a escolha preferencial, uma vez que o SSL *(Secure Sockets Layer)*, que era o protocolo anterior de segurança, já está obsoleto e apresenta várias vulnerabilidades. Conforme descrito em (42), o SSL não é mais recomendado devido à sua ineficácia em mitigar ataques modernos. O TLS, por outro lado, oferece uma segurança significativamente mais robusta e é amplamente adotado para proteger as comunicações em redes inseguras. Embora o uso do TLS seja viável e recomendado para soluções de rastreamento veicular, sua implementação em plataformas de microcontroladores como o Arduino pode ser desafiadora. O suporte a TLS em versões do Arduino é limitado, conforme discutido em (43), o que pode dificultar a implementação de uma solução segura, principalmente em versões mais antigas do hardware ou em plataformas com recursos limitados.

O suporte limitado ao TLS em dispositivos baseados em Arduino é um desafio técnico significativo, pois implica em um aumento no custo e na complexidade do desenvolvimento de soluções de rastreamento veicular seguras. Para implementar um sistema que utilize TLS adequadamente, é necessário garantir que o dispositivo tenha o processamento necessário para suportar a criptografia, além de depender de versões mais recentes do firmware e de bibliotecas específicas para comunicação segura. A falta de suporte total ao TLS em muitos dispositivos e a necessidade de configurações específicas exigem uma análise detalhada das limitações do hardware e software envolvidos, de modo a garantir que a implementação da segurança não comprometa o desempenho ou a funcionalidade do rastreador veicular.

Em resumo, embora existam diversas propostas para rastreadores veiculares, a grande maioria delas falha em adotar práticas de segurança adequadas para a proteção dos dados sensíveis durante o processo de transmissão. As soluções que utilizam SMS como meio de comunicação e HTTP como protocolo de comunicação carecem de medidas robustas de segurança, tornando-se vulneráveis a ataques. A implementação de TLS nas soluções de rastreamento veicular poderia melhorar significativamente a segurança, mas o suporte limitado a esse protocolo em plataformas como o Arduino representa um obstáculo importante. Assim, a evolução das soluções de rastreamento veicular deve levar em consideração não apenas os requisitos funcionais, mas também a implementação de segurança desde a concepção do sistema, utilizando protocolos modernos e práticas de segurança adequadas.

4 ARQUITETURA PROPOSTA

4.1 PROTÓTIPO DESENVOLVIDO

O protótipo do rastreador veicular seguro foi construído com base no *Raspberry PI* e em um conjunto de softwares livres. A arquitetura utiliza como base para comunicação o protocolo *Message Queuing Telemetry Transport* (*MQTT* - <https://mqtt.org/>), considerado ideal para comunicação entre dispositivos de uma rede de Internet das Coisas (*Internet of Things – IoT*), pois é leve e extremamente eficiente. O servidor utilizado para o MQTT foi o Mosquitto (<https://mosquitto.org/>). Na parte da aplicação, foi utilizada arquitetura Web, com servidor de aplicação PHP 7 (<https://www.php.net/>) rodando sob o *Apache HTTP Server* (<https://httpd.apache.org/>). Para a persistência das informações de rastreamento foi utilizado um servidor de banco de dados relacional *PostgreSQL Server* (<https://www.postgresql.org/>). Para a interface do operador para acompanhamento dos rastreadores foram utilizados mapas do *OpenStreetMap* (<https://www.openstreetmap.org/>) acessados no servidor de aplicação a partir da biblioteca *OpenLayers* (<https://openlayers.org/>).

Um microcomputador Compaq HP 8400 UltraSlim Desktop foi utilizado como hardware para instalação física do sistema operacional base do servidor. Foi utilizada a distribuição *Debian GNU/Linux versão 12 (bookworm)* (<https://www.debian.org/>) como sistema operacional instalado diretamente no computador sem a utilização de virtualização. Os servidores Apache HTTP Server, PostgreSQL Server e Mosquitto MQTT Broker foram instalados como serviços neste sistema operacional.

Na arquitetura proposta há preocupação com a segurança nas mais diversas camadas da aplicação, desde a segurança na infraestrutura de servidor de aplicação, servidor de banco de dados, no acesso da aplicação ao banco de dados, nos protocolos para troca de informações entre os rastreadores e o servidor, além de outras preocupações com relação à segurança muitas vezes não observadas pelos desenvolvedores de software.

HARDWARE E SOFTWARE UTILIZADOS

As Tabelas 4.1 e 4.2, resumem os principais componentes de hardware e software empregados no projeto. No hardware, destacam-se dispositivos como o Raspberry Pi 3 Model B, o módulo GPS ublox NEO-6M e o modem USB 3G/4G, que desempenham papéis essenciais na operação e comunicação do sistema. Além disso, o microcomputador Compaq/HP 8400 Ultraslim Desktop complementa a infraestrutura, fornecendo suporte como servidor. No software, são mencionados sistemas operacionais, linguagens e bibliotecas, como PHP, jQuery e OpenLayers, que suportam funcionalidades de rastreamento, armazenamento de dados e exibição de mapas na interface do usuário. O uso de extensões PHP como PDO e pdo_pgsql facilita o acesso e a manipulação de dados no banco de dados, enquanto a biblioteca Smarty contribui para a organização do código através da separação de lógica e apresentação. O Composer, gerenciador de dependências, também é utilizado para garantir a correta instalação das bibliotecas necessárias.

Tabela 4.1: Resumo do hardware utilizado.

Categoria	**Detalhes**
Raspberry Pi 3 Model B	- Quad Core 1.2GHz Broadcom BCM2837 64bit CPU - 1GB RAM
Módulo GPS	- ublox NEO-6M conectado via interface serial ao Raspberry Pi
Modem USB 3G/4G	- Para comunicação remota com o servidor/broker MQTT
Microcomputador	- Compaq/HP 8400 Ultraslim Desktop

Tabela 4.2: Resumo do software utilizado.

Categoria	**Detalhes**
Sistemas Operacionais	- Raspberry Pi OS v11 (bullseye) - Debian GNU/Linux v12 (bookworm)
PHP 7	- **Extensões:** * `dio`: captura de dados NMEA da porta serial do módulo GPS * `PDO` e `pdo_pgsql`: acesso ao banco de dados para armazenamento e leitura de informações - **Bibliotecas:** * `Smarty`: template engine para separação de lógica * `Composer`: gerenciador de dependências * `php-mqtt client`: cliente MQTT para conexão e envio de dados
Javascript	- **Bibliotecas:** * `jQuery`: requisições assíncronas (AJAX) para atualização de informações no mapa * `OpenLayers`: exibição de mapas do OpenStreetMap na interface

A Figura 4.1 mostra uma foto do protótipo desenvolvido. O item 1 da figura é uma *breakout board*, uma simples placa de circuito impresso que facilita o acesso a interface de entrada/saída de propósito geral *(General-Purpose Input/Output - GPIO* (44) e (45)). O item 2 é um *protoboard* ou placa de ensaio para prototipação de circuitos eletrônicos. Facilita o teste de circuitos eletrônicos em fase de desenvolvimento, pois a sua utilização elimina a necessidade de soldagem de componentes. O item 3 é a antena do módulo GPS ublox NEO-6M. O item 4 é o módulo GPS ublox NEO-6M. Este módulo está ligado ao Raspberry Pi indiretamente através do *protoboard* e *breakout board* e utiliza somente 4 fios para a ligação. Os fios branco e azul fornecem a alimentação para o módulo GPS através dos pinos 1 e 9. O fio branco é ligado ao módulo no pino VCC e ao Raspberry no pino 1 (3,3 volts). Já o fio azul está ligado no módulo no pino GND (*ground* - terra) e no Raspberry ao pino 9 (terra). Os fios amarelo e verde fazem são utilizados para a comunicação serial entre o Raspberry Pi e o módulo GPS. O fio verde é ligado ao pino TX (trasmissor) do módulo GPS e ao pino 10 (UART RX - receptor) do Raspberry. E o fio amarelo é ligado ao pino RX (receptor) do módulo GPS e ao pino 8 (UART TX - transmissor) do Raspberry. O item 5 da figura é um cabo de ligação de 40 vias que interliga a *breakout board* a interface GPIO de 40 pinos do Raspberry Pi. Por fim, o item 6 é o próprio Raspberry Pi Model 3B instalado dentro de uma *case*.

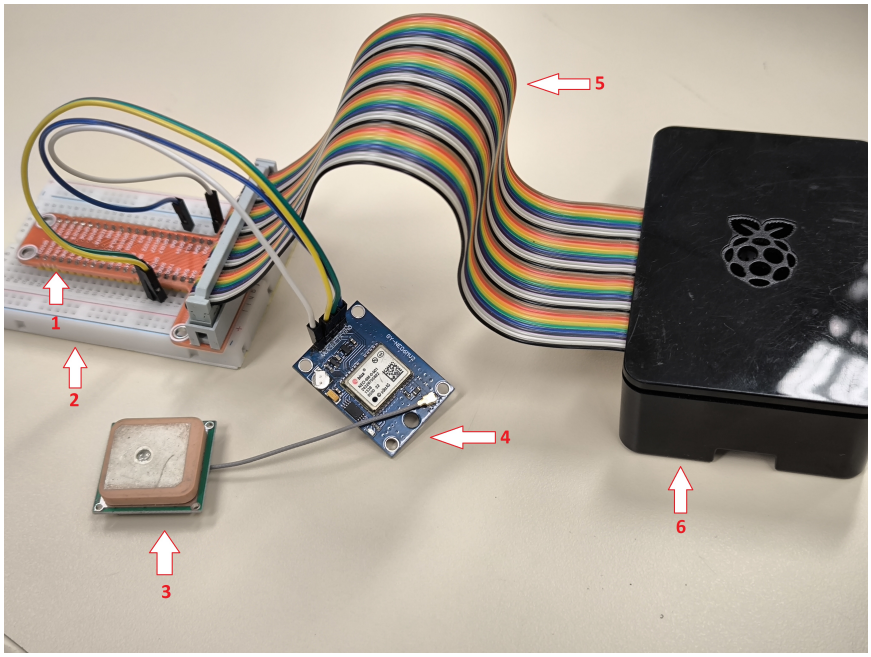

Figura 4.1: Protótipo desenvolvido

Para facilitar os testes durante o desenvolvimento, foi utilizado um telefone celular com a função de roteador portátil habilitada e o Raspberry Pi foi conectado a esta rede. Não foi possível conectar um módulo GSM/GPRS à esta versão do Raspberry, pois somente é possível a habilitação de uma única porta serial, a qual é utilizada para o módulo GPS. Para facilitar a prototipagem foi utilizada uma *breakout board*.

4.1.1 FUNCIONAMENTO BÁSICO

A arquitetura conforme Figura 4.2 prevê quantidade ilimitada de rastreadores, cada rastreador sendo identificado por um identificador único. Como identificador foi utilizado apenas um número inteiro sequencial. O esquema de banco de dados utilizado neste estudo está ilustrado na Figura 4.3, onde foi criada uma tabela **gps_tracker** que armazena a lista de todos os rastreadores com uma descrição para cada um. A tabela **tracking_data_simple** foi projetada para armazenar todos os dados de rastreamento provenientes de diferentes rastreadores, utilizando a data e hora capturadas diretamente pelo dispositivo. Esse registro segue o formato do Tempo Universal Coordenado (UTC), sem realizar transformações para o fuso horário local. As informações de latitude e longitude, obtidas pelo módulo GPS no formato NMEA (graus e minutos), são convertidas para graus decimais antes de serem armazenadas, facilitando seu posterior tratamento e análise. Além do UTC, essa tabela registra dados como latitude, longitude, altitude e a quantidade de satélites monitorados, estabelecendo uma relação com a tabela **gps_tracker** por meio de uma chave estrangeira, que garante a integridade referencial entre os rastreadores e seus respectivos dados de rastreamento.

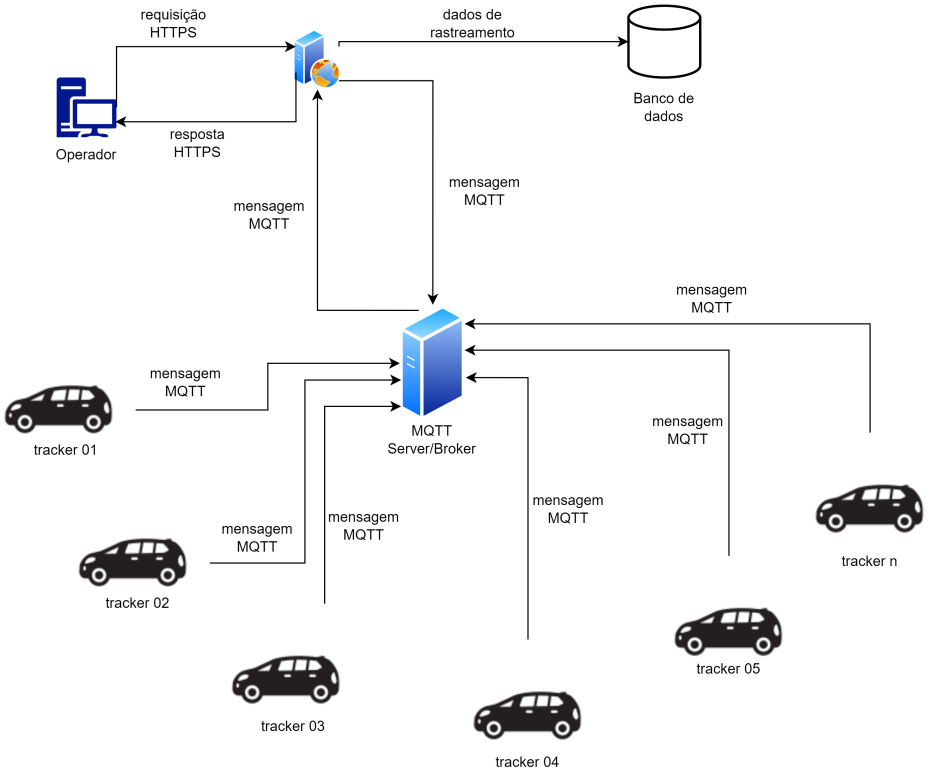

Figura 4.2: Arquitetura geral

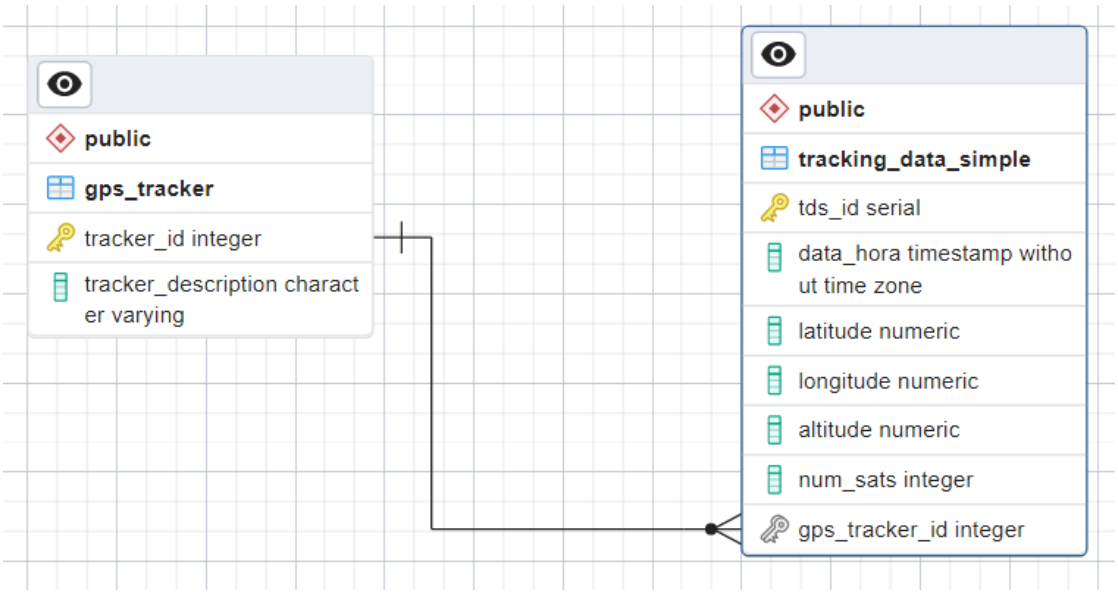

Figura 4.3: Diagrama entidade relacionamento do protótipo

Cada rastreador após ligado, inicia automaticamente o Raspberry Pi OS em modo console e depois inicia um script PHP Command Line Interface (PHP CLI) através de carregamento na crontab. Este script executa a seguinte sequência de ações:

I. Conecta ao servidor MQTT

II. Faz a leitura dos dados enviados pelo módulo GPS através da porta serial (/dev/ttyAMA0) utilizando a extensão dio

III. Armazena as informações capturadas como forma de backup para posterior envio nos casos onde não há rede de dados para o envio em tempo real das informações para o MQTT Server/Broker

IV. Filtra os dados recebidos e publica apenas a sentença GGA do padrão NMEA-0183, que contém as informações conforme demonstrado na Tabela 4.3.

Tabela 4.3: Sentença GPGGA

N	**Parâmetro**	**Descrição**
0	$GPGGA	Cabeçalho, sempre valor fixo para cada tipo de sentença (frame).
1	utc	Tempo no qual a leitura das informações foi realizada pelos satélites em formato horas/minutos/segundos/décimos de segundos. Tempo em UTC.
2	lat	Latitude (DDmm.mm). DD = Graus. mm.mm = Minutos decimais.
3	lat_dir	Direção da latitude (N = Norte, S = Sul).
4	lon	Longitude (DDDmm.mm). DD = Graus. mm.mm = Minutos decimais.
5	lon_dir	Direção da longitude (E = Leste, W = Oeste).
6	quality	Indicador de qualidade GPS. Ver tabela.
7	#sats	Número de satélites em uso. Pode ser diferente do número de satélites visualizados pelo módulo GPS.
8	hdop	Diluição da precisão horizontal.
9	alt	Altitude da antena acima/abaixo do nível médio do oceanos.
10	a-units	Unidade da altitude da antena (M = metros).
11	undulation	Ondulação - a relação entre o geóide e o elipsóide WGS84.
12	u-units	Unidades de ondulação (M = metros).
13	age	Idade dos dados de correção (em segundos). A idade máxima informada aqui é limitada a 99 segundos.
14	*xx	Checksum para verificação da integridade das informações.

Fonte: NovaTel <https://docs.novatel.com/OEM7/Content/Logs/GPGGA.htm>

No lado do servidor, executa-se uma sequência de ações que inclui o recebimento, validação e processamento das requisições, a execução de operações específicas, como consultas ou cálculos, e o envio da

resposta ao cliente, garantindo eficiência e segurança.

I. O servidor MQTT recebe as informações publicadas pelos rastreadores

II. Um script PHP CLI que roda no servidor faz a inscrição no tópico, recebe os dados de rastreamento que já estão no servidor MQTT e armazena esses dados no banco de dados

No lado do usuário (operador) da aplicação:

I. O usuário visualiza no mapa um ou mais rastreadores e tem a opção de visualização em tempo real e também de fazer consulta a histórico de rastreamentos

Na Figura 4.4 é apresentada a sequência de operações com os objetos envolvidos.

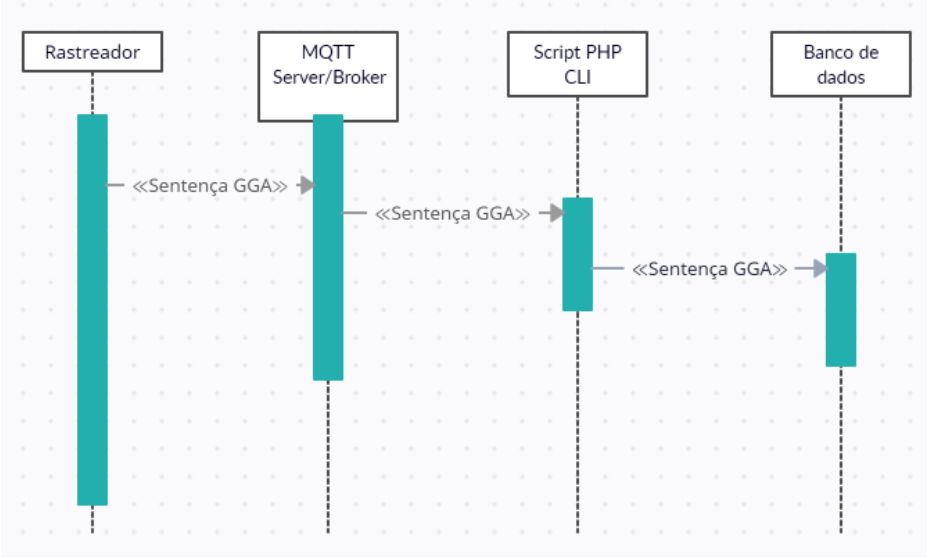

Figura 4.4: Diagrama de sequência básico

4.1.2 SEGURANÇA NA CAMADA DE BANCO DE DADOS

I. Proteção contra SQL Injection: toda a aplicação utiliza *prepared statements* para execução de SQL tanto para consulta quanto para alteração de dados no banco de dados. Esta utilização evita ataques deste tipo, pois as variáveis das queries são substituídas apenas no momento da execução efetiva. Além disso a substituição só é possível por variáveis cujo conteúdo sejam do mesmo tipo de dados da coluna da tabela do banco de dados que se está consultando ou alterando. Em um cenário no qual uma possível vulnerabilidade no nível da aplicação seja descoberta, ataques deste tipo se tornariam inviáveis. Este foco na segurança é reforçado diferentemente de outras soluções propostas como em (46) que não citam especificamente proteção nas operações relativas a linguagem de manipulação de dados (*Data Manipulation Language - DML*).

Algoritmo 1 Pseudocódigo SQL para obter a última localização do rastreador.

```
1: sql ← "SELECT latitude, longitude FROM public.tracking_data_simple WHERE
   gps_tracker_id = :gps_tracker_id ORDER BY tds_id DESC LIMIT 1"
2: sth ← prepare(sql)
3: sth.bindValue(':gps_tracker_id', trackerId, PDO::PARAM_INT)
4: sth.execute()
5: result ← sth.fetch(PDO::FETCH_ASSOC)
6: lat ← result['latitude']
7: lon ← result['longitude']
```

Esta *query* representada no Pseudocódigo 1 busca a última posição (latitude e longitude) de um rastreador específico. Primeiramente a *query* é preparada com valores de substituição (placeholders), neste caso ":gps_tracker_id" e somente antes da execução efetiva da *query* é que o valor é substituído por um número inteiro que representa um rastreador específico. Na substituição do valor específico é definido o tipo de dados da coluna específica do banco de dados que se está consultando. Neste caso o valor "PDO::PARAM_INT" indica ao banco de dados que somente serão aceitos valores inteiros.

I. Privilégios mínimos: o princípio de privilégios mínimos é utilizado tanto em ambiente de desenvolvimento quanto em ambiente de produção. Os diferentes perfis de usuários do banco de dados possuem apenas o conjunto mínimo de privilégios para consulta ou alteração dos dados necessários. Por exemplo, o usuário de banco de dados da aplicação que roda no servidor e recebe os dados dos diversos rastreadores (clientes MQTT) apenas tem privilégio para inclusão dos dados dos rastreadores na tabela específica do banco de dados. Portanto evita-se qualquer adulteração intencional (alteração ou exclusão) mesmo que houvesse vazamento das credenciais deste usuário de banco de dados. O operador da aplicação, que visualiza os rastreadores no mapa, tem apenas privilégio de consulta aos dados de rastreamento, impedindo adulterações. Nenhum usuário possui privilégios para comandos (*Data Definition Language – DDL*), garantindo a integridade da estrutura do banco de dados. Essa abordagem de segurança no banco de dados se diferencia de outras que não especificam essas restrições como em (31) e (47).

São 3 os usuários de banco de dados que possuem acesso aos objetos do banco de dados **secureVehicleTrackerDB**:

I. secureVehicleTrackerAdmin

II. secureVehicleTrackerOperator

III. secureVehicleTrackerWriterDBUser

Nos itens 4.1.2.1, 4.1.2.2 e 4.1.2.3 são apresentados os detalhes e a motivação para a criação de cada um destes usuários de banco de dados diferentes (perfis).

4.1.2.1 ADMINISTRADOR

O banco de dados **secureVehicleTrackerDB** é de propriedade do usuário **secureVehicleTrackerAdmin**, que é responsável pela sua administração e controle. O código DDL utilizado para a criação desse banco de dados, incluindo a definição de suas propriedades e configurações iniciais, é apresentado no Pseudocódigo 2. Esse código estabelece os parâmetros necessários, como o proprietário, a codificação, o idioma e outras configurações relevantes para o funcionamento adequado do banco de dados.

Algoritmo 2 Pseudocódigo DDL para Criação do Banco de Dados secureVehicleTrackerDB

```
1: - Database: secureVehicleTrackerDB
2: - DROP DATABASE IF EXISTS "secureVehicleTrackerDB";
3: CREATE DATABASE "secureVehicleTrackerDB"
4:   WITH OWNER = "secureVehicleTrackerAdmin"
5:   ENCODING = 'UTF8'
6:   LC_COLLATE = 'Portuguese_Brazil.1252'
7:   LC_CTYPE = 'Portuguese_Brazil.1252'
8:   TABLESPACE = pg_default
9:   CONNECTION LIMIT = -1
10:  IS_TEMPLATE = False;
```

O usuário **secureVehicleTrackerAdmin** é criado por meio do código DDL apresentado no Pseudocódigo 3, o qual define suas permissões e propriedades de acesso no banco de dados. Esse código estabelece a criação do usuário com os privilégios necessários para administrar e gerenciar o banco de dados **secureVehicleTrackerDB**, garantindo que ele tenha o controle adequado sobre a execução de operações e a manutenção do sistema de forma segura e eficiente. O código também pode incluir definições como a senha, o esquema de conexão e outras configurações de segurança que regulam o comportamento e a integridade do usuário dentro do banco de dados.

Algoritmo 3 Pseudocódigo DDL para Criação do Usuário secureVehicleTrackerAdmin

```
1: - Role: "secureVehicleTrackerAdmin"
2: - DROP ROLE IF EXISTS "secureVehicleTrackerAdmin";
3: CREATE ROLE "secureVehicleTrackerAdmin"WITH
4:   LOGIN
5:   NOSUPERUSER
6:   INHERIT
7:   NOCREATEDB
8:   NOCREATEROLE
9:   NOREPLICATION
10:  ENCRYPTED PASSWORD 'SCRAM-SHA-256$4096:**********************';
11: COMMENT ON ROLE "secureVehicleTrackerAdmin"IS 'Usuário administrador do banco de
    dados secureVehicleTrackerDB.';
```

Esse usuário, como proprietário do banco de dados, automaticamente possui todos os privilégios para manipulação de todos os objetos dentro deste banco de dados. Contudo, seguindo o princípio de privilégios mínimos, ele deve ser utilizado exclusivamente durante o desenvolvimento e a manutenção da aplicação. A utilização de um usuário de banco de dados com privilégios além dos necessários em ambientes de produção ou homologação não é recomendada, pois pode representar uma vulnerabilidade. Portanto, é

essencial que tais práticas sejam evitadas, conforme as diretrizes estabelecidas em (48).

4.1.2.2 OPERADOR

O usuário **secureVehicleTrackerOperator** é utilizado somente pela aplicação Web no lado do servidor. Este usuário possui apenas privilégios mínimos para consulta ao banco de dados para apresentação das informações de posicionamento dos rastreadores. Exemplo da interface do operador no modo de depuração é apresentada na Figura 4.5. Ressalta-se que neste modo de depuração se está utilizando o protocolo HTTP sem segurança e esta situação é somente aceita durante o desenvolvimento em ambiente local controlado pelo desenvolvedor. Em hipótese alguma é recomendada a não utilização do protocolo *Transport Layer Security (TLS)* em ambientes de homologação ou produção. O código *DDL* é demonstrado no Pseudocódigo 4.

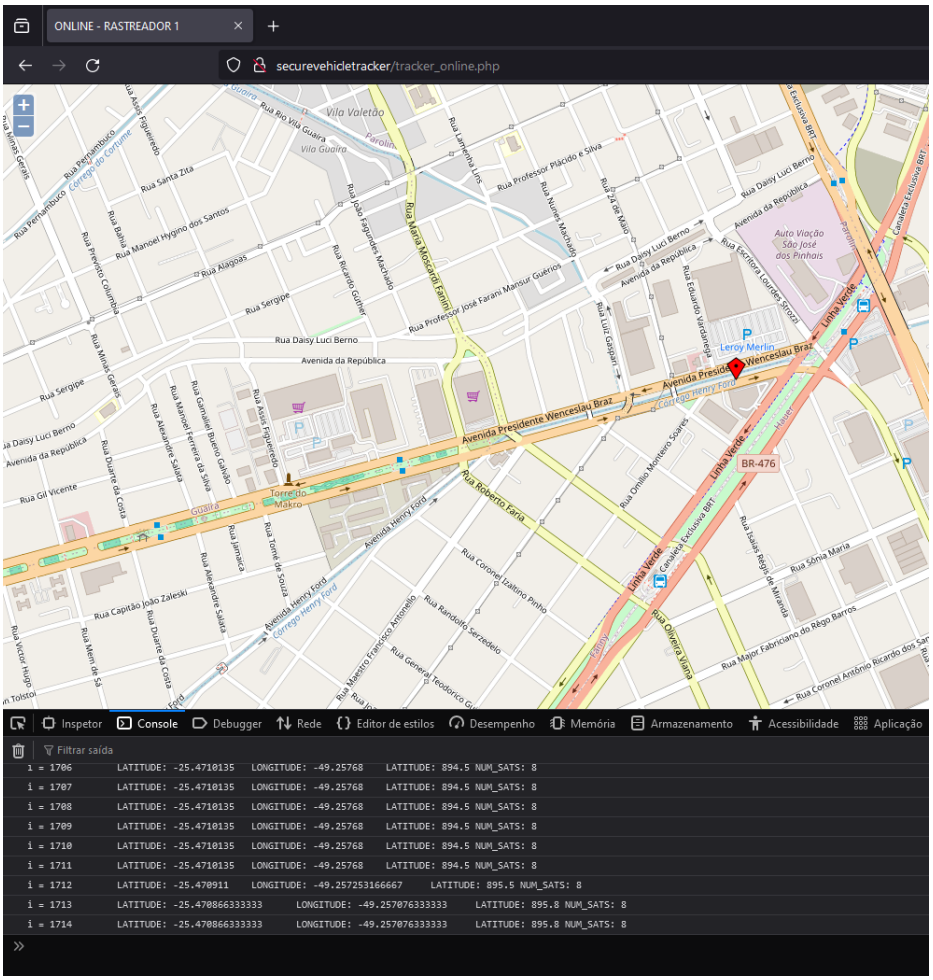

Figura 4.5: Interface do operador - modo de depuração

Algoritmo 4 Pseudocódigo DDL para Criação e Privilégios do Usuário SecureVehicleTrackerOperator

```
1: - Papel: "secureVehicleTrackerOperator"
2: - DROP ROLE IF EXISTS "secureVehicleTrackerOperator";
3: CREATE ROLE "secureVehicleTrackerOperator"WITH
4:   LOGIN
5:   NOSUPERUSER
6:   INHERIT
7:   NOCREATEDB
8:   NOCREATEROLE
9:   NOREPLICATION;
10: COMMENT ON ROLE "secureVehicleTrackerOperator"IS 'Usuário operador do Rastreador
    Veicular Seguro';
11: GRANT SELECT ON TABLE gps_tracker TO secureVehicleTrackerOperator;
12: GRANT SELECT ON TABLE tracking_data_simple TO secureVehicleTrackerOperator;
```

4.1.2.3 GRAVADOR

O usuário **secureVehicleTrackerWriterDBUser** foi projetado exclusivamente para ser utilizado no script **PHP-CLI** executado diretamente no servidor. Esse script desempenha a função crítica de receber mensagens no protocolo **MQTT** enviadas por múltiplos rastreadores. Posteriormente, os dados de posicionamento, incluindo latitude, longitude, e outras informações relevantes, são processados e armazenados no banco de dados com o auxílio desse usuário.

Esse design de uso específico segue o princípio dos privilégios mínimos, garantindo que o **secureVehicleTrackerWriterDBUser** tenha apenas as permissões necessárias para a inserção de dados no banco de dados. A criação desse usuário é realizada por meio de um código *DDL* (Data Definition Language), que define seus privilégios e restrições de acesso. Esse código está descrito no Pseudocódigo 5, servindo como referência para implementação em outros ambientes semelhantes.

Algoritmo 5 Pseudocódigo DDL para Criação e Privilégios do Usuário secureVehicleTrackerWriterDBUser

```
1: - Papel: "secureVehicleTrackerWriterDBUser"
2: - DROP ROLE IF EXISTS "secureVehicleTrackerWriterDBUser";
3: CREATE ROLE "secureVehicleTrackerWriterDBUser"WITH
4:   LOGIN
5:   NOSUPERUSER
6:   INHERIT
7:   NOCREATEDB
8:   NOCREATEROLE
9:   NOREPLICATION;
10: COMMENT ON ROLE "secureVehicleTrackerWriterDBUser"IS
11:   'Usuário da aplicação do lado do servidor que recebe os dados dos rastreadores
    via MQTT e faz a persistência destes no banco de dados.';
12: GRANT INSERT ON TABLE gps_tracker TO secureVehicleTrackerWriterDBUser;
13: GRANT INSERT ON TABLE tracking_data_simple TO secureVehicleTrackerWriterDBUser;
14: GRANT USAGE ON SEQUENCE tracking_data_simple_tds_id_seq;
15: GRANT USAGE ON SEQUENCE tracking_data_td_id_seq;
```

Este usuário não pode alterar nem excluir registros de rastreamento, apenas pode inseri-los no banco de dados, desta forma restringe-se o comportamento da aplicação, reforçando a segurança da arquitetura proposta.

4.1.3 CONFIGURAÇÕES DE SEGURANÇA DO SERVIDOR POSTGRESQL

O servidor de banco de dados PostgreSQL, versão 15.10, foi instalado em um servidor Linux, utilizando a distribuição Debian 12 (*bookworm*). A instalação foi seguida por uma configuração focada na segurança, garantindo o uso do protocolo TLS versão 1.3 para criptografia das conexões. As configurações de segurança, incluindo a habilitação do TLS, foram cuidadosamente ajustadas no arquivo de configuração **postgresql.conf**. Isso envolveu a especificação de parâmetros essenciais como o caminho para os certificados SSL, a definição de *ciphers* permitidos, e a configuração de protocolos mínimos de segurança, visando garantir a integridade e confidencialidade das comunicações entre o servidor de banco de dados e os clientes. Essas medidas ajudam a proteger o tráfego de dados e a evitar vulnerabilidades comuns associadas a protocolos de comunicação desatualizados. Conforme demonstrado no Pseudocódigo 6.

Algoritmo 6 Pseudocódigo da configuração do PostgreSQL para Conexões e TLS

```
1: – PostgreSQL configuration file
2: – CONNECTIONS AND AUTHENTICATION
3:
4: listen_addresses = '*'      ▷ what IP address(es) to listen on; comma-separated list of addresses;
   defaults to 'localhost'; use '*' for all (change requires restart)
5: port = 5432                                  ▷ Port for connections (change requires restart)
6: max_connections = 100              ▷ Maximum number of connections (change requires restart)
7: unix_socket_directories = '/var/run/postgresql'  ▷ Comma-separated list of directories for
   Unix sockets
8:
9: – Authentication
10:
11: – Configurações TLS
12: ssl = on                                                          ▷ Enable TLS
13: ssl_ca_file = '/etc/ssl/certs/ca.crt'                  ▷ SSL Certificate Authority file
14: ssl_cert_file = '/etc/ssl/certs/cert.pem'                  ▷ SSL Certificate file
15: ssl_key_file = '/etc/ssl/private/cert.key'                      ▷ SSL Key file
16: ssl_ciphers = 'HIGH:MEDIUM:+3DES:!aNULL'                ▷ Allowed SSL ciphers
17: ssl_prefer_server_ciphers = on                          ▷ Prefer server ciphers
18: ssl_ecdh_curve = 'prime256v1'                  ▷ ECDH curve used for key exchange
19: ssl_min_protocol_version = 'TLSv1.3'           ▷ Minimum protocol version for TLS
```

O arquivo **pg_hba.conf** que define a forma de autenticação dos clientes no servidor do PostgreSQL também foi modificado e ficou com a configuração conforme o Pseudocódigo 7.

Desta forma, o servidor obriga a autenticação utilizando o protocolo TLS versão 1.3 e somente os três usuários do banco de dados com perfis diferentes - **secureVehicleTrackerAdmin**, **secureVehicleTrackerOperator** e **secureVehicleTrackerWriterDBUser** - poderão acessar o banco de dados específico da aplicação **secureVehicleTrackerDB**, conforme demonstrado no Pseudocódigo 7.

Algoritmo 7 Pseudocódigo da configuração de Autenticação de Cliente PostgreSQL para Conexões SSL e Usuários Específicos

```
1: – PostgreSQL Client Authentication Configuration File
2: – ===================================================
3: – Refer to the "Client Authentication"section in the PostgreSQL documentation.
4:
5: – This file controls:
6: – 1. Which hosts are allowed to connect.
7: – 2. How clients are authenticated.
8: – 3. Which PostgreSQL user names they can use.
9: – 4. Which databases they can access.
10:
11: – Record format examples:
12: local DATABASE USER METHOD [OPTIONS]
13: host DATABASE USER ADDRESS METHOD [OPTIONS]
14: hostssl DATABASE USER ADDRESS METHOD [OPTIONS]
15: hostnossl DATABASE USER ADDRESS METHOD [OPTIONS]
16: hostgssenc DATABASE USER ADDRESS METHOD [OPTIONS]
17: hostnogssenc DATABASE USER ADDRESS METHOD [OPTIONS]
18:
19: – Connection types:
20: local: Unix-domain socket
21: host: TCP/IP socket (encrypted or not)
22: hostssl: SSL-encrypted TCP/IP socket
23: hostnossl: Non-SSL-encrypted TCP/IP socket
24: hostgssenc: GSSAPI-encrypted TCP/IP socket
25: hostnogssenc: Non-GSSAPI-encrypted TCP/IP socket
26:
27: – Example rules:
28: local all postgres peer                                   ▷ Administrative login via Unix domain socket
29: local all all peer                          ▷ Unix domain socket connections for all users and databases
30: hostssl secureVehicleTrackerDB secureVehicleTrackerAdmin 0.0.0.0/0 scram-sha-256 ▷
    Secure connection for admin user
31: hostssl     secureVehicleTrackerDB     secureVehicleTrackerOperator     0.0.0.0/0
    scram-sha-256                                                ▷ Secure connection for operator user
32: hostssl    secureVehicleTrackerDB    secureVehicleTrackerWriterDBUser    0.0.0.0/0
    scram-sha-256                                                ▷ Secure connection for writer user
```

4.1.4 CONFIGURAÇÕES DE SEGURANÇA DO MQTT SERVER/BROKER

Em um estudo comparativo entre protocolos utilizados em IoT (49), o protocolo MQTT obteve melhores resultados. Outro estudo mostrou que o MQTT é especialmente robusto para utilização na comunicação máquina-máquina em dispositivos IoT (50), pois é um protocolo leve que pode facilmente ser utilizado em dispositivos com limitação física de memória. Foi esse um dos motivos da sua escolha, além da sua capacidade de transferência de dados de forma segura. Foi escolhido o *Mosquitto MQTT Server* como servidor/broker MQTT por ser este de código aberto e amplamente utilizado pela comunidade mundial, além de ser aderente as especificações MQTT mais recentes. Além disso, foi feita uma instalação local

do servidor, de forma que diversas configurações pudessem ser testadas, além do comportamento e monitoramento de registros de log. Conforme (51), o maior problema relativo à segurança de aplicações IoT é a má configuração MQTT ou mesmo a utilização de configurações padrão. Para minimizar problemas relativos à segurança MQTT, a arquitetura propõe que todos os clientes (os rastreadores) obrigatoriamente façam autenticação junto ao servidor MQTT. Também é obrigatória a autorização, seja através de *Access Control Lists (ACLs)* ou através do plugin Dynamic Security. Além disso, a arquitetura obriga que seja adotada a segurança na camada de transporte utilizando o *Transport Layer Security (TLS)*. Desta forma, os rastreadores devem apresentar um certificado X.509 ao MQTT broker para estabelecerem uma conexão e conseguir enviar informações.

O servidor *Mosquitto MQTT Broker* versão 2.0.11 foi instalado e configurado em um servidor Linux, disbribuição Debian 12 (*bookworm*). O servidor foi configurado com um enfoque robusto em segurança, adotando o protocolo TLS como padrão para garantir a integridade e a confidencialidade das comunicações entre os dispositivos e o sistema central. Como parte dessa estratégia, foi implementado um mecanismo de autenticação mútua, exigindo que os rastreadores (clientes) apresentem certificados TLS válidos antes de estabelecerem conexão com o servidor. Essa abordagem não apenas reforça a proteção contra acessos não autorizados, mas também assegura que apenas dispositivos confiáveis possam interagir com o sistema.

Além disso, a configuração do servidor foi realizada de maneira criteriosa, seguindo as diretrizes e recomendações detalhadas na documentação oficial do *Mosquitto MQTT Broker* (52) e também em (53). Este procedimento garantiu que todas as opções de segurança fossem corretamente ajustadas, incluindo a seleção de algoritmos criptográficos robustos, o uso de certificados emitidos por autoridades confiáveis e a definição de políticas de acesso restritivas. Essas medidas visam proteger o ambiente contra ameaças externas e minimizar vulnerabilidades operacionais. Foram executados seis passos para a configuração, conforme demonstrado nos Pseudocódigos 8, 9, 10, 11, 12 e 13.

I. Criação da autoridade certificadora local

Algoritmo 8 Pseudocódigo da geração e Configuração de Certificados TLS para Autoridade Certificadora Local (Parte 1)

1: **– Passos para a criação de certificados:**
2: `madalozzo@vehicletrackereee: $ cd` ▷ Acessa o diretório inicial do usuário
3: `madalozzo@vehicletrackereee: $ rm -rf certs` ▷ Remove o diretório `certs`, se existente
4: `madalozzo@vehicletrackereee: $ mkdir certs` ▷ Cria o diretório `certs`
5: `madalozzo@vehicletrackereee: $ cd certs` ▷ Acessa o diretório `certs`
6: `madalozzo@vehicletrackereee: /certs$ openssl genrsa -out ca.key 4096` ▷ Gera a chave privada de 4096 bits para a Autoridade Certificadora (CA)
7: `madalozzo@vehicletrackereee: /certs$ openssl req -new -x509 -days 3650 -key ca.key -out ca.crt` ▷ Cria um certificado X.509 válido por 10 anos com base na chave gerada

Algoritmo 8 Pseudocódigo da geração e Configuração de Certificados TLS para Autoridade Certificadora Local (Continuação)

– Informações solicitadas durante a geração do certificado:

1: `Country Name (2 letter code) [AU]:BR` ▷ País: Brasil
2: `State or Province Name (full name) [Some-State]:PR` ▷ Estado: Paraná
3: `Locality Name (eg, city) []:Curitiba` ▷ Cidade: Curitiba
4: `Organization Name (eg, company) [Internet Widgits Pty Ltd]:LocalSecurity` ▷ Organização: LocalSecurity
5: `Organizational Unit Name (eg, section) []:Autoridade Certificadora para o Mosquitto MQTT Broker` ▷ Unidade Organizacional: CA para o Mosquitto MQTT Broker
6: `Common Name (e.g. server FQDN or YOUR name) []:` ▷ Nome comum: não especificado
7: `Email Address []:` ▷ Email: não especificado

II. Criação dos certificados do servidor

Algoritmo 9 Pseudocódigo e Assinatura de Certificado TLS para o Servidor Mosquitto MQTT Broker

1: **Generate the server private key:**
2: `openssl genrsa -out server.key 4096` ▷ Creates a 4096-bit RSA key and saves it in the `server.key` file.
3: **Generate the certificate signing request (CSR):**
4: `openssl req -new -key server.key -out server.csr` ▷ Generates a CSR using the private key. The user provides information like country, state, and server name.
5: **Provided information:**
   - **Country Name:** `BR`
   - **State or Province Name:** `PR`
   - **Locality Name:** `Curitiba`
   - **Organization Name:** `LocalSecurity`
   - **Organizational Unit Name:** `Mosquitto MQTT Server`
   - **Common Name:** `vehicletrackereee.ddns.net`
   - **Email Address:** `(Left blank)`
   - **Challenge Password:** `(Left blank)`
   - **Optional Company Name:** `(Left blank)`
6: **Sign the certificate with the certificate authority (CA):**
7: `openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out server.crt -days 3650` ▷ Signs the CSR using the CA's certificate and key, generating a certificate valid for 3650 days.

III. Criação dos certificados do cliente

Algoritmo 10 Pseudocódigo da criação e Assinatura de Certificado TLS para Cliente MQTT Broker

1: **Generate the client private key:**
2: `openssl genrsa -out client.key 4096` ▷ Creates a 4096-bit RSA private key and saves it in the `client.key` file.
3: **Generate the certificate signing request (CSR):**
4: `openssl req -new -key client.key -out client.csr` ▷ Generates a CSR using the private key. The user inputs information like country, state, and organization.
5: **Provided information:**
- **Country Name:** `BR`
- **State or Province Name:** `PR`
- **Locality Name:** `Curitiba`
- **Organization Name:** `LocalSecurity`
- **Organizational Unit Name:** `tracker`
- **Common Name:** `(Left blank)`
- **Email Address:** `(Left blank)`
- **Challenge Password:** `(Left blank)`
- **Optional Company Name:** `(Left blank)`

6: **Sign the certificate with the certificate authority (CA):**
7: `openssl x509 -req -in client.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out client.crt -days 3650` ▷ Signs the CSR using the CA's certificate and key, generating a client certificate (`client.crt`) valid for 3650 days.

IV. Configuração do servidor Mosquitto. Foram acrescentadas linhas ao arquivo de configuração do servidor **/etc/mosquitto/mosquitto.conf** conforme Algoritmo 11.

Algoritmo 11 Pseudocódigo da configuração de Autenticação TLS e Permissão Anônima no Mosquitto

1: **Enable anonymous access:**
2: `allow_anonymous true` ▷ Allows clients to connect without authentication. Useful for development and testing but less secure in production.
3: **Set listener for TLS connections:**
4: `listener 8883` ▷ Specifies the port (`8883`) to listen for secure MQTT connections over TLS.
5: **Define paths to TLS certificate files:**
6: `cafile /etc/mosquitto/certs/ca.crt` ▷ Path to the Certificate Authority (CA) certificate, used to verify client certificates.
7: `certfile /etc/mosquitto/certs/server.crt` ▷ Path to the server certificate for authenticating the broker.
8: `keyfile /etc/mosquitto/certs/server.key` ▷ Path to the private key associated with the server certificate.
9: **Enforce client certificate authentication:**
10: `require_certificate true` ▷ Requires clients to present valid certificates signed by the specified CA for authentication.

V. Cópia dos certificados e chaves gerados para o local apropriado e ajustes de permissões

Algoritmo 12 Pseudocódigo do gerenciamento e Configuração de Permissões para Certificados TLS no Mosquitto

1: **Execute the following command to subscribe to the topic:**

```
mosquitto_sub -h vehicletrackereee.ddns.net -p 8883 \
  --cafile /etc/mosquitto/certs/ca.crt \
  --cert /etc/mosquitto/certs/client.crt \
  --key /etc/mosquitto/certs/client.key \
  -t test -d
```

▷ The command subscribes to the topic `test` using TLS authentication.

2: **Connection Process:**

3: `Client (null) sending CONNECT` ▷ The client initiates a connection to the broker.

4: **Acknowledgment from the Broker:**

5: `Client (null) received CONNACK (0)` ▷ The broker acknowledges the connection request with a success code.

6: **Subscribe to the Topic:**

7: `Client (null) sending SUBSCRIBE (Mid: 1, Topic: test, QoS: 0, Options: 0x00)` ▷ The client subscribes to the topic `test` with Quality of Service (QoS) level 0.

8: **Subscription Acknowledgment:**

9: `Client (null) received SUBACK` ▷ The broker confirms the subscription request.

10: **Message Received:**

11: `Client (null) received PUBLISH (d0, q0, r0, m0, 'test', ... (33 bytes))` ▷ The client receives a published message from the topic.

12: **Example Message:**

13: `Teste básico de mensagem com TLS` ▷ This is the content of the published message.

14: **Terminate the Connection:**

15: `ĈClient (null) sending DISCONNECT` ▷ The client gracefully disconnects from the broker.

VI. Teste básico da configuração

Algoritmo 13 Pseudocódigo do teste de Subscrição MQTT com Certificados TLS

1: **Execute the following command to subscribe to an MQTT topic securely:**

```
mosquitto_sub -h vehicletrackereee.ddns.net -p 8883 \
  --cafile /etc/mosquitto/certs/ca.crt \
  --cert /etc/mosquitto/certs/client.crt \
  --key /etc/mosquitto/certs/client.key \
  -t test -d
```

▷ The command establishes a secure connection using TLS to the MQTT broker and subscribes to the topic `test`.

2: **Connection Initialization:**

3: `Client (null) sending CONNECT` ▷ The client sends a connection request to the broker.

4: **Connection Acknowledgment:**

5: `Client (null) received CONNACK (0)` ▷ The broker acknowledges the connection with a success code.

6: **Subscription Request:**

7: `Client (null) sending SUBSCRIBE (Mid: 1, Topic: test, QoS: 0, Options: 0x00)` ▷ The client subscribes to the specified topic using QoS level 0.

8: **Subscription Confirmation:**

9: `Client (null) received SUBACK` ▷ The broker confirms the subscription request.

10: **Message Reception:**

11: `Client (null) received PUBLISH (d0, q0, r0, m0, 'test', ... (33 bytes))` ▷ The client receives a published message from the subscribed topic.

12: **Example Published Message:**

13: `Teste básico de mensagem com TLS` ▷ This is the content of the message sent to the topic.

14: **Disconnection:**

15: `ĈClient (null) sending DISCONNECT` ▷ The client terminates the connection to the broker gracefully.

Desta forma o servidor foi configurado para escutar requisições na porta 8883, utilizando criptografia através do protocolo TLS para a troca de informações entre os rastreadores (clientes) e o servidor e também entre o servidor e o navegador Web no caso do acesso à interface de monitoramento pelo administrador ou operador, garantindo desta forma o sigilo das comunicações.

No passo 4 executado, o parâmetro **allow_anonymous** foi definido como **true**, caso contrário seria necessária uma dupla autenticação, utilizando usuário e senha, além da apresentação do certificado digital pelo rastreador ao servidor, o qual é obrigatório através da utilização do parâmetro **require_certificate** com valor **true**. Considera-se suficiente a emissão de um certificado digital exclusivo para cada rastreador colocado em funcionamento. A utilização do protocolo TLS garante a confidencialidade, integridade, disponibilidade e autenticidade na troca de informações entre os rastreadores e o servidor MQTT. A vantagem da utilização de um certificado para cada rastreador também permite maior controle dos rastreadores que eventualmente apresentarem problemas e tiverem que ser retirados do sistema, bastando a inclusão do certificado do rastreador com problema na Lista de Certificados Revogados (LCR). Essa inclusão inviabiliza automaticamente as comunicações nestes casos.

4.1.5 SEGURANÇA DO SERVIDOR DE APLICAÇÃO

O servidor de aplicação não deve permitir a execução de funções PHP potencialmente consideradas perigosas, como funções que permitem o acesso direto a execução de comandos do sistema operacional, as quais podem ser utilizadas para ataques do tipo Webshell conforme (54). As seguintes funções devem ser proibidas dentro do arquivo de configuração do PHP no servidor:

I. system - Executa um programa externo e mostra a saída

II. exec - Executa um programa externo

III. popen - Abre um processo como ponteiro de arquivo

IV. shell_exec - Executa um comando via shell e retorna a saída inteira como uma string

V. operador de execução (*backtick operator*) - Mesma coisa que o shell_exec

VI. pcntl_exec - Executa um programa no espaço de memória do processo atual

No arquivo de configuração do PHP no servidor (`/etc/php/7.4/apache2/php.ini`) as configurações foram aplicadas para desabilitar as funções vulneráveis, conforme demonstrado no Pseudocódigo 14.

Algoritmo 14 Pseudocódigo da configuração de Certificados e Permissões para o Mosquitto MQTT Broker

1: **Definir as funções desabilitadas no PHP:**

```
disable_functions = "system, exec, popen, shell_exec, pcntl_exec"
```

▷ Esta configuração desativa funções potencialmente perigosas no PHP.

O operador de execução (*backtick operator*) é desabilitado automaticamente quando a função `shell_exec` é desabilitada. É fundamental observar que as funções anteriormente listadas são apenas um guia de funções que devem ser desabilitadas, porém outras funções podem ser também desabilitadas se julgadas desnecessárias na aplicação. Para se ter uma segurança ainda mais robusta, considerando que a aplicação será instalada em ambiente de TI próprio, com total controle do hardware e software, a seguinte sequência de ações poderia ser executada para a linguagem PHP:

I. Criação de script ou aplicação independente que faça uma análise sintática (*parser*) de todos os arquivos PHP da aplicação e gere como resultado uma lista de funções utilizadas

II. Criação de um script ou aplicação independente que gere uma lista de todas as funções disponíveis na versão específica do PHP utilizada e faça uma diferença com a lista de funções que a aplicação efetivamente utiliza, gerando como resultado uma lista de funções que devem ser desabilitadas na saída

III. Fazer a inclusão da lista de funções da saída do item anterior na diretiva de configuração do PHP **`disable_functions`** manualmente ou através de script independente que altere esta configuração automaticamente

Desta forma o princípio dos privilégios mínimos também é respeitado, forçando o interpretador PHP a não ter a capacidade de interpretar funções que efetivamente não são utilizadas pela aplicação.

4.1.6 CONFIGURAÇÕES DE SEGURANÇA DO SERVIDOR APACHE

A proteção dos arquivos do servidor que estão fora da parte pública acessível através do protocolo HTTP deve ser obrigatoriamente realizada. Os Pseudocódigos 15 e 16 exibem estas configurações.

Algoritmo 15 Pseudocódigo da configuração de Acesso Negado para o Diretório Raiz

1: **Definir restrição de acesso para o diretório raiz:**

```
<Directory "/">
    Require all denied
</Directory>
```

▷ Esta configuração impede qualquer acesso ao diretório raiz do sistema.

Por padrão a configuração acima proíbe o acesso ao root ("/") do sistema.

Configurações adicionais são feitas para que somente a parte pública da aplicação seja acessível a partir do protocolo HTTP.

Algoritmo 16 Pseudocódigo da configuração de Permissões de Acesso para Diretórios Públicos e Privados

1: **Definir restrição de acesso para o diretório privado:**

```
<Directory "/var/www/html/vehicleTrackerEEE/private/">
    Require all denied
</Directory>
```

▷ Impede qualquer acesso ao diretório `private/`, protegendo arquivos sensíveis.

2: **Definir permissão de acesso para o diretório público:**

```
<Directory "/var/www/html/vehicleTrackerEEE/public/">
    Require all granted
</Directory>
```

▷ Permite acesso irrestrito ao diretório `public/`, tornando os arquivos acessíveis publicamente.

A estrutura de diretórios da aplicação proposta é organizada da seguinte maneira: o diretório **private** contém todas as bibliotecas de terceiros utilizadas no protótipo, além dos arquivos de configuração da aplicação. Esses arquivos não são expostos e, portanto, não podem ser acessados diretamente pelos clientes HTTP. O diretório **public**, por sua vez, armazena os arquivos de estilo (*Cascading Style Sheets - CSS*), imagens e scripts JavaScript, que são acessíveis diretamente pelo navegador do cliente quando solicitados pela aplicação, conforme demonstrado no Pseudocódigo 17.

Algoritmo 17 Pseudocódigo da estrutura de Diretórios do Projeto VehicleTrackerEEE

1: **Estrutura de Diretórios:**

```
vehicleTrackerEEE
+---private
|   +---cache
|   +---classes
|   +---configs
|   +---templates
|   +---templates_c
|   \---vendor
\---public
    +---css
    +---images
    \---js
```

▷ Organização dos diretórios para separar conteúdo privado e público.

2: **Diretório `private/`:**

- `cache`: Armazena arquivos temporários gerados pela aplicação.
- `classes`: Contém as definições de classes usadas no sistema.
- `configs`: Armazena arquivos de configuração da aplicação.
- `templates`: Contém os templates para renderização de páginas.
- `templates_c`: Armazena templates compilados para maior desempenho.
- `vendor`: Diretório gerenciado pelo Composer para dependências externas.

3: **Diretório `public/`:**

- `css`: Arquivos de estilos (Cascading Style Sheets).
- `images`: Recursos de imagens usados na aplicação.
- `js`: Arquivos de scripts JavaScript para interatividade.

Além disso, a estrutura de diretórios da aplicação foi configurada com privilégios mínimos, definidos por listas de controle de acesso (*Access Control Lists - ACLs*) no sistema de arquivos, conforme demonstrado no Pseudocódigo 18. Esses privilégios garantem que apenas usuários autorizados tenham acesso aos diretórios e arquivos necessários para o funcionamento da aplicação, assegurando a proteção e integridade dos dados.

Algoritmo 18 Pseudocódigo da estrutura de Diretórios de Aplicação VehicleTrackerEEE

1: **Comandos Executados:**

```
cd vehicleTrackerEEE/..
chown -R apache:apache vehicleTrackerEEE
```

▷ Comandos usados para navegar até o diretório superior e alterar a propriedade do diretório.

2: **Descrição do Código:**

- `cd vehicleTrackerEEE/..`: Move para o diretório pai de `vehicleTrackerEEE`. ▷ Útil para realizar operações no diretório completo.
- `chown -R apache:apache vehicleTrackerEEE`: Altera, de forma recursiva (`-R`), a propriedade do diretório `vehicleTrackerEEE` e de todos os arquivos e subdiretórios para o usuário e grupo `apache`. ▷ Garante que o servidor Apache tenha permissão para acessar e gerenciar os arquivos necessários.

3: **Contexto de Utilização:**

- Garantir que o servidor web (`apache`) tenha permissão adequada para executar a aplicação hospedada no diretório `vehicleTrackerEEE`.
- Prevenir erros de permissão ao acessar arquivos críticos da aplicação.

Desta forma, há maiores garantias com relação à segurança da aplicação, uma vez que mesmo com possíveis vulnerabilidades em códigos executáveis PHP, não há possibilidade de escrita nos diretórios da aplicação sendo executada em ambiente de produção. O único diretório com permissão de escrita é o **templates_c**, cuja permissão é necessária para que a biblioteca **Smarty** consiga compilar os templates HTML, que são lidos do diretório **templates**, interpretados e após isso são escritos nesse diretório. A permissão **de execução (x)** é necessária em todos os diretórios acessíveis pela aplicação para que a aplicação acesse o conteúdo destes subdiretórios.

Também é feita a alteração do usuário e do grupo proprietário da aplicação, para que somente o interpretador PHP dentro do servidor Apache tenha acesso aos diretórios e arquivos da aplicação.

Considerando que o usuário **apache** é o usuário básico, o comando é executado no diretório da aplicação para alterar o proprietário e o grupo da aplicação de forma recursiva. Conforme demonstrado no Pseudocódigo 19.

Algoritmo 19 Pseudocódigo da alteração de Propriedade do Diretório para o Usuário Apache

1: **Comandos:**

```
cd vehicleTrackerEEE/..
chown -R apache:apache vehicleTrackerEEE
```

2: **Descrição:**

- `cd vehicleTrackerEEE/..`: Navega para o diretório pai de `vehicleTrackerEEE`. ▷ Este passo posiciona o terminal no nível correto do sistema de arquivos para executar o próximo comando.
- `chown -R apache:apache vehicleTrackerEEE`: Altera, de forma recursiva (`-R`), a propriedade do diretório `vehicleTrackerEEE`, incluindo todos os arquivos e subdiretórios. ▷ Define o usuário `apache` e o grupo `apache` como proprietários.

3: **Objetivo:**

- Garantir que o servidor web (`Apache`) tenha controle total sobre os arquivos e subdiretórios do `vehicleTrackerEEE`.
- Evitar problemas relacionados a permissões, como falhas de leitura/escrita pelo servidor durante a execução da aplicação.

4: **Contexto de Utilização:**

- Após a instalação ou atualização da aplicação `vehicleTrackerEEE`, é necessário ajustar as permissões para que o Apache possa acessar e gerenciar os recursos corretamente.

Em relação ao servidor Apache, recomenda-se que a aplicação seja hospedada em um *Virtual Host* específico, garantindo a centralização e a organização das configurações. Além disso, é imprescindível a utilização obrigatória do protocolo HTTPS para toda a aplicação, com a versão mais recente do protocolo *TLS*, assegurando a segurança da comunicação.

A configuração do *Virtual Host* tem como objetivo isolar a aplicação das demais configurações de outras aplicações ou sistemas, permitindo uma administração mais eficiente e específica para cada ambiente. Dessa forma, a aplicação terá um espaço dedicado, com regras claras e personalizadas para seu funcionamento.

O *virtual host* foi configurado de forma a atender às exigências de segurança e desempenho, incluindo ajustes específicos no servidor para habilitar a criptografia TLS e otimizar o acesso a recursos essenciais.

Conforme demonstrado no Pseudocódigo 20, as configurações descritas acima são representadas visualmente, detalhando como a aplicação é isolada em seu próprio Virtual Host, com configurações específicas de segurança e desempenho, incluindo a implementação de TLS, HSTS, e controles de acesso para os diretórios. A Figura 20 ilustra a estrutura de configuração do Virtual Host no servidor Apache, demonstrando as boas práticas adotadas para garantir um ambiente seguro e eficiente.

Algoritmo 20 Pseudocódigo da configuração de VirtualHost com TLS e Controle de Acesso no Apache

1: **Configuração em Apache:**

```
<VirtualHost *:443>
    DocumentRoot "/var/www/html/vehicleTrackerEEE/public/"
    ServerName vehicletrackereee.ddns.net
    DirectoryIndex index.html

    # Configurações de TLS e certificado
    SSLEngine on
    SSLCertificateFile /etc/ssl/certs/vehicletrackereee.crt
    SSLCertificateKeyFile /etc/ssl/private/vehicletrackereee.private.key
    SSLProtocol all -SSLv3 -TLSv1 -TLSv1.1 -TLSv1.2
    SSLCipherSuite ECDHE-ECDSA-AES256-GCM-SHA384 ...

    # Configurações de cabeçalho
    Header always set Strict-Transport-Security max-age=31536000

    # Permissões de diretórios
    <Directory "/var/www/html/vehicleTrackerEEE/private/">
        Require all denied
    </Directory>
    <Directory "/var/www/html/vehicleTrackerEEE/public/">
        Require all granted
    </Directory>

    # Configuração de logs
    ErrorLog ${APACHE_LOG_DIR}/vehicletrackereee.error.log
    CustomLog ${APACHE_LOG_DIR}/vehicletrackereee.access.log common

    # Configurações PHP
    AddDefaultCharset utf-8
    php_flag log_errors on
    php_flag display_errors off
    php_value error_reporting 32767
    php_value error_log /var/log/apache2/vehicletrackereee.php.error.log
</VirtualHost>
```

2: **Descrição:**

- `<VirtualHost *:443>`: Define uma configuração para conexões HTTPS na porta 443.
- `DocumentRoot "/var/www/html/vehicleTrackerEEE/public/"`: Define o diretório público como o ponto de entrada principal da aplicação.
- `SSLEngine on`: Ativa o uso de SSL/TLS para conexões seguras.
- `SSLCertificateFile` e `SSLCertificateKeyFile`: Define os caminhos para os certificados e chave privada do servidor.
- `SSLProtocol` e `SSLCipherSuite`: Configura protocolos e cifras seguras para criptografia, desabilitando protocolos obsoletos como `SSLv3`.
- `Header always set Strict-Transport-Security max-age=31536000`: Implementa HSTS, forçando navegadores a usar conexões HTTPS.
- `<Directory>`: Configura permissões de acesso:
  - `/private/`: Negado a todos os usuários.
  - `/public/`: Acesso permitido para todos.
- `ErrorLog` e `CustomLog`: Define arquivos específicos para logs de erro e acesso.
- Configurações PHP: Ajusta relatórios de erro e logs para diagnósticos da aplicação.

3: **Objetivo:**

- Garantir conexões seguras utilizando TLS.
- Proteger diretórios privados e permitir acesso controlado ao diretório público.
- Registrar eventos importantes em logs para auditoria e depuração.

Este *virtual host* foi configurado de forma a aceitar apenas conexões na porta 443 utilizando o protocolo TLS para clientes que acessem o domínio `vehicletrackereee.ddns.net`.

Os possíveis erros gerados pelo Apache dentro do virtual host ficam armazenados em um arquivo de log específico para o domínio da aplicação. Da mesma forma os registros de acesso também ficam restritos mesmo domínio, facilitando desta forma a análise de qualquer tentativa de exploração de vulnerabilidades na aplicação através do protocolo HTTP. Conforme demonstrado na Tabela 4.4.

Já com relação aos erros gerados pelo PHP, em ambiente de homologação e produção é necessário que estes sejam enviados para um log de erros em arquivo. Como medida de segurança básica, nestes mesmos ambientes, a diretiva PHP que habilita a exibição de erros na tela (*display_errors*) é mantida desligada. O nível de registro de erros (*error_reporting*) é definido como o nível máximo, o que garante que todos os tipos de erro gerados pelo PHP sejam capturados. A Tabela 4.4 mostra todos os tipos de erros definidos dentro do núcleo do PHP conforme (55). É muito importante que qualquer aplicação tenha seus *logs* de erro monitorados constantemente em ambiente de produção. Na proposta específica desta arquitetura utilizando o PHP como linguagem principal, mesmo erros mais básicos do tipo E_NOTICE ou E_WARNING, que podem não alterar o comportamento da aplicação, devem ser eliminados. Idealmente, qualquer aplicação deveria passar por todos os casos de teste possíveis e não apresentar qualquer registro de erro. A melhor opção seria realizar a eliminação de todos os tipos de erros em ambiente de desenvolvimento, desta forma minimizando qualquer tentativa de exploração de vulnerabilidades no nível da aplicação.

Tabela 4.4: PHP: tipos de erros para registro em arquivo de *log*

VALOR	**CONSTANTE**	**DESCRIÇÃO**
1	E_ERROR (int)	Erros fatais de tempo de execução. Indicam erros que não podem ser recuperados, como um problema de alocação de memória. A execução do script é interrompida.
2	E_WARNING (int)	Avisos de tempo de execução (erros não fatais). A execução do script não é interrompida.
4	E_PARSE (int)	Erros de análise em tempo de compilação. Os erros de análise só devem ser gerados pelo analisador.
8	E_NOTICE (int)	Avisos de tempo de execução. Indicam que o script encontrou algo que pode indicar um erro, mas que também pode ocorrer no curso normal da execução de um script.
16	E_CORE_ERROR (int)	Erros fatais que ocorrem durante a inicialização do PHP. Isso é como um E_ERROR, exceto pelo fato de ser gerado pelo núcleo do PHP.
		Continua na próxima página...

VALOR	CONSTANTE	DESCRIÇÃO
32	E_CORE_WARNING (int)	Avisos (erros não fatais) que ocorrem durante a inicialização do PHP. Isso é como um E_WARNING, exceto pelo fato de ser gerado pelo núcleo do PHP.
64	E_COMPILE_ERROR (int)	Erros fatais de tempo de compilação. É como um E_ERROR, exceto pelo fato de ser gerado pelo *Zend Scripting Engine*.
128	E_COMPILE_WARNING (int)	Avisos de tempo de compilação (erros não fatais). Isso é como um E_WARNING, exceto pelo fato de ser gerado pelo *Zend Scripting Engine*.
256	E_USER_ERROR (int)	Mensagem de erro gerada pelo usuário. É como um E_ERROR, exceto pelo fato de ser gerada no código PHP usando a função PHP trigger_error().
512	E_USER_WARNING (int)	Mensagem de aviso gerada pelo usuário. É como uma E_WARNING, exceto pelo fato de ser gerada no código PHP usando a função PHP trigger_error().
1024	E_USER_NOTICE (int)	Mensagem de aviso gerada pelo usuário. É como um E_NOTICE, exceto pelo fato de ser gerado no código PHP usando a função PHP trigger_error().
2048	E_STRICT (int)	Permite que o PHP sugira alterações em seu código que garantirão a melhor interoperabilidade e compatibilidade futura de seu código.
4096	E_RECOVERABLE_ERROR (int)	Erro fatal capturável. Indica que ocorreu um erro provavelmente perigoso, mas que não deixou o mecanismo em um estado instável. Se o erro não for capturado por um identificador definido pelo usuário (consulte também set_error_handler()), o aplicativo será abortado como se fosse um E_ERROR.
8192	E_DEPRECATED (int)	Avisos em tempo de execução. Ative essa opção para receber avisos sobre códigos que não funcionarão em versões futuras.
16384	E_USER_DEPRECATED (int)	Mensagem de aviso gerada pelo usuário. É como um E_DEPRECATED, exceto pelo fato de ser gerado no código PHP usando a função PHP trigger_error().

Continua na próxima página...

VALOR	CONSTANTE	DESCRIÇÃO
32767	E_ALL (int)	Todos os erros, avisos e notificações.

Fonte: PHP <https://www.php.net/manual/en/errorfunc.constants.php>

5 RESULTADOS

Devido a falta de recursos de segurança nativos presentes no SMS e optando por utilizar protocolos mais modernos que atendam a requisitos mínimos de segurança, a arquitetura aqui proposta utilizou em seu protótipo o protocolo MQTT, que além de suprir as necessidades de segurança, é adequado para dispositivos IoT. Também optou-se por utilizar hardware baseado em microprocessador, como o Raspberry Pi que foi utilizado, por apresentar maior versatilidade na utilização de protocolos de mais alto nível.

A escolha de uma arquitetura baseada em microcontrolador ou microprocessador por si só não é fator determinante para implementação de requisitos de segurança. Normalmente arquiteturas baseadas em microcontroladores são mais limitadas na quantidade de memória principal e no poder de processamento de informações, no entanto possuem mais entradas e saídas para sensores e atuadores digitais ou analógicos. Arquiteturas baseadas em microprocessadores possuem maior poder de processamento de informações e maior quantidade de memória primária e secundária, portanto menores restrições na execução de códigos. Os princípios básicos de segurança em várias camadas utilizados na arquitetura apresentada neste trabalho também podem ser utilizados na implementação de um rastreador seguro baseado em microcontroladores.

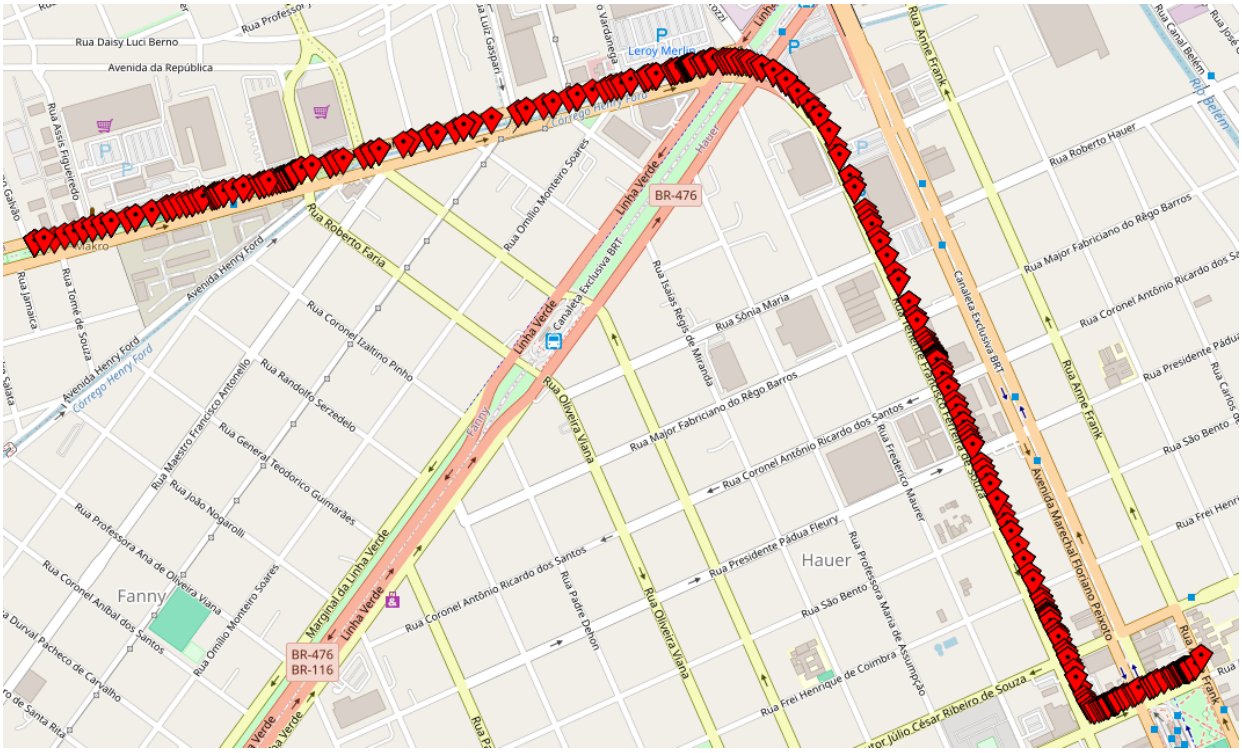

Figura 5.1: Exemplo de rota de rastreamento armazenada em banco de dados

A arquitetura do rastreador apresentada neste trabalho recebe as informações de posicionamento através do módulo GPS a cada segundo, armazenando as informações localmente e também enviando essas informações através do protocolo MQTT para o servidor, o qual se encarrega de fazer a persistência das informações de rastreamento no banco de dados. A Figura 5.1 mostra um exemplo de rota capturada e

armazenada em banco de dados.

As informações são armazenadas localmente na memória interna do Raspberry Pi como forma de contingência. Nas localidades onde o rastreador não conseguir sinal da rede de dados 3G/4G para envio online das informações de rastreamento, estas são enviadas quando o rastreador conseguir restabelecer o sinal da rede de dados, e o envio das informações ocorre atrasado, porém sem perda. Desta forma garante o funcionamento fundamental do rastreador, mesmo que a informação seja recebida pelo operador com atraso nestas situações.

O armazenamento em banco de dados das informações de rastreamento também traz vantagens em casos que se queira analisar rotas já armazenadas e a realização de auditoria nos casos de um possível desvio de rota já programada.

Para fazer os testes da arquitetura proposta foi criado um protótipo do rastreador com toda a infraestrutura necessária incluindo servidor Web, servidor de banco de dados e servidor MQTT. Para criação de um ambiente o mais real possível, foi criado um nome de domínio no provedor de serviços de DNS dinâmico gratuito No-IP (https://www.noip.com/pt-BR). O nome de domínio criado foi vehicletrackereee.ddns.net, significando *Vehicle Tracker with End-to-End Encryption*. O servidor Web Apache foi configurado com um Virtual Host de forma a aceitar requisições através da Internet acessadas a partir desde domínio. O sistema operacional utilizado no servidor foi o Debian GNU/Linux 12 (bookworm).

5.1 RESULTADOS DA CONFIGURAÇÃO SEGURA DO SERVIDOR WEB APACHE

Para proteção das comunicações entre o servidor Web Apache e os rastreadores, operadores e administradores foi utilizado o protocolo de segurança Transport Layer Security (TLS) versão 1.3. Apesar das tentativas de geração de um certificado válido para o servidor através de serviços de autoridades certificadoras que emitem certificados TLS X.509 gratuitamente, como *Let's Encrypt* (https://letsencrypt.org/pt-br/) ou *ZeroSSL* (https://zerossl.com), não foi possível a utilização destes serviços devido ao bloqueio da porta padrão HTTPS – porta 443 pela operadora de Internet utilizada para conexões de entrada.

Como forma alternativa, optou-se pela criação de um certificado digital autoassinado e configuração do servidor Apache utilizando este certificado (56). Para criação do certificado TLS foi utilizado o pacote OpenSSL com a versão 3.0.14 da biblioteca OpenSSL, última versão estável disponível para o sistema operacional utilizado durante a escrita deste trabalho.

Os seguintes passos foram utilizados para a criação de um certificado SSL autoassinado com a inclusão da extensão *SubjectAltName (SAN)*:

I. Geração de uma chave privada

II. Geração de uma Solicitação de Assinatura de Certificado (*Certificate Signing Request – CSR*)

III. Remoção da frase-senha da chave

IV. Criação de um arquivo de configuração para inclusão da informação SAN

V. Geração de um certificado autoassinado

Os comandos necessários para a criação do certificado digital são apresentados no Apêndice I.1. Esse apêndice contém uma descrição detalhada de cada etapa do processo, incluindo a execução dos comandos no terminal, a configuração dos parâmetros essenciais, como chave privada e pública, e as etapas subsequentes de verificação e validação. Além disso, são fornecidas explicações sobre as melhores práticas a serem seguidas, garantindo que o certificado gerado atenda aos requisitos de segurança e funcionalidade exigidos.

Localmente, o servidor Web Apache foi configurado na porta padrão HTTPS 443, porém para acesso remoto a partir de qualquer dispositivo através da Internet, incluindo os rastreadores, foi utilizada a porta 4443. O roteador da operadora foi configurado de forma a fazer o redirecionamento da porta 4443 para conexões TCP recebidas da Internet e direcioná-las para a porta 443 do servidor Apache local.

A simples utilização do protocolo HTTPS não garante segurança robusta o suficiente, pois algumas versões do protocolo já não são recomendadas e dependendo da forma de configuração do servidor Web, há possibilidade de várias vulnerabilidades estarem presentes.

Para certificação independente do servidor Web com relação à segurança e aos parâmetros de configuração foi utilizada a ferramenta ***testssl.sh*** (57), uma ferramenta de linha de comando gratuita que verifica o serviço de um servidor em qualquer porta quanto ao suporte de cifras e protocolos TLS/SSL, bem como falhas criptográficas recentes. A versão utilizada para os testes foi a 3.0.9. As Informações complementares sobre os resultados obtidos para análise estão no Apêndice I.2.

Com base nos resultados iniciais fornecidos pelo script **testssh.sh** foi possível identificar que alguns dos parâmetros analisados estavam fora dos padrões considerados mais seguros.

Quanto aos protocolos suportados, o servidor está atualmente configurado para aceitar apenas o protocolo TLS 1.2. Para garantir que o servidor passe a operar exclusivamente com a versão mais recente, TLS 1.3, é necessário realizar ajustes específicos em sua configuração. Essas configurações básicas, incluindo os parâmetros e etapas necessárias para a habilitação do TLS 1.3, estão descritas nos Pseudocódigos apresentados 21, 22, 23 e 24.

Algoritmo 21 Pseudocódigo Verificação de Suporte ao TLS 1.2

```
Initialize the process to verify TLS 1.2 support
Establish a secure connection to the server using HTTPS ▷ Opens a connection to the server with encryption enabled
Check if the server offers support for TLS 1.2
if TLS 1.2 is supported by the server then
    Display message: "TLS 1.2 offered (OK)" ▷ Indicates that the server supports TLS 1.2
else
    Display message: "TLS 1.2 not supported" ▷ Indicates that the server does not support TLS 1.2
end if
Terminate the secure connection ▷ Closes the connection to free up resources
```

Durante a execução dos testes de preferências do servidor, constatou-se que a configuração da ordem

de cifras (*cipher suites*) estabelecida no servidor não foi aprovada. Esse resultado revelou que a sequência de cifras configurada não atendia aos critérios de conformidade ou compatibilidade exigidos no ambiente de teste. Conforme demonstrado no Pseudocódigo 22.

Algoritmo 22 Pseudocódigo Ordenação de Cifradores pelo Servidor: Não Configurada (Problema Identificado)

```
Initialize the process to verify if the server enforces its cipher order
Establish a secure connection to the server using HTTPS ▷ Opens a connection to test server configurations
Check if the server enforces its cipher order
if The server enforces its cipher order then
    Display message: "Has server cipher order? yes (OK)" ▷ Indicates that the server enforces its preferred cipher order
else
    Display message: "Has server cipher order? no (NOT ok)" ▷ Indicates that the server does not enforce its cipher order, which could be insecure
end if
Terminate the secure connection ▷ Closes the connection to release resources
```

Durante os testes de conformidade com os padrões de servidor, constatou-se que o parâmetro de cadeia de confiança (*chain of trust*) não foi aprovado. Os detalhes estão ilustrados no Pseudocódigo 23.

Algoritmo 23 Pseudocódigo Cadeia de Confiança Inválida: Certificado Autoassinado Detectado

```
Initialize the process to verify the server's certificate chain of trust
Establish a secure connection to the server using HTTPS ▷ Opens a connection to validate the certificate chain
Check the server's certificate chain
if The certificate chain is valid and trusted then
    Display message: "Chain of trust OK" ▷ Indicates the server's certificate is signed by a trusted CA
else
    Display message: "Chain of trust NOT ok (self signed)" ▷ Indicates the server's certificate is self-signed or not trusted
end if
Terminate the secure connection ▷ Closes the connection to free up resources
```

Neste caso específico, a falha pode ser desconsiderada, uma vez que, após a configuração do servidor com um certificado válido e emitido por uma autoridade certificadora confiável, o problema relacionado à cadeia de confiança será resolvido e não ocorrerá novamente. Essa correção garantirá que a comunicação seja devidamente validada e que a confiança na conexão seja restabelecida, atendendo aos requisitos de segurança exigidos.

Adicionalmente, o servidor não passou no teste OCSP URI, o que significa que não foi possível validar o status do certificado em tempo real por meio do protocolo *Online Certificate Status Protocol* (OCSP) . Esse teste é fundamental para verificar se o certificado foi revogado ou se ainda é válido, e a falha pode indicar a ausência de uma configuração adequada para acessar a URI do servidor OCSP. Os detalhes dessa configuração, estão ilustrados no Pseudocódigo 24.

Este parâmetro, relacionado à validação do status do certificado por meio do protocolo OCSP URI, não tem relevância quando o servidor utiliza um certificado autoassinado, pois não é possível verificar seu status de revogação em tempo real de forma confiável sem o apoio de uma autoridade certificadora. Portanto, essa falha pode ser ignorada enquanto o servidor estiver configurado com um certificado autoassinado. No entanto, ao ser configurado com um certificado válido emitido por uma autoridade certificadora confiável, a verificação do status do certificado por OCSP se torna possível e relevante, garantindo a conformidade com os requisitos de segurança. A validação por OCSP URI passa a ser uma etapa importante para assegurar que o certificado não tenha sido revogado e que a conexão seja segura.

Algoritmo 24 Pseudocódigo Verificação de Revogação de Certificado Falha: Nenhuma URI CRL ou OCSP Fornecida

```
Initialize the process to check the availability of OCSP URI or CRL
Establish a secure connection to the server using HTTPS ▷ Opens a connection to check certificate revocation information
Check if the server's certificate includes an OCSP URI or CRL
if The server provides an OCSP URI or CRL then
    Display message: "OCSP URI: Available" ▷ Indicates that the server provides a valid OCSP URI or CRL for certificate status checking
else
    Display message: "OCSP URI: NOT ok - neither CRL nor OCSP URI provided" ▷ Indicates that the server does not provide an OCSP URI or CRL for certificate status checking
end if
Terminate the secure connection ▷ Closes the connection to release resources
```

Após a correção dos parâmetros a configuração do *Virtual Host* no Apache passou a ter a forma exibida no Apêndice I.4. Foram utilizados como referências para configuração o sítio oficial da Fundação Apache (58) e também (59). Todas as falhas apontadas pela primeira execução do *testssl.sh* foram corrigidas. A partir da restrição de aceitação somente da versão 1.3 do protocolo TLS, automaticamente o problema da aceitação das cifras tipo CBC obsoletas foi corrigido. Também foi incluído o parâmetro *SSLHonorCipherOrder*, habilitando este nas configurações do *Virtual Host*, desta forma foi aprovado no parâmetro *"Has server cipher order?"*, o qual garante a utilização prioritária de cifras mais fortes. Também foi habilitado o protocolo *HTTP Strict Transport Security*, o qual é recomendado pela RFC 6797 (60). Este protocolo ajuda a evitar ataques do tipo homem no meio *(man-in-the-middle)* que visem tentar fazer o *downgrade* do protocolo utilizando o HTTP e também evita o sequestro de *cookies*.

Uma das seções de resultados apresentados pelo *testssl.sh* é o **teste de vulnerabilidades**. Vulnerabilidades e exposições comuns (*Common Vulnerabilities and Exposures - CVE (61)*) são um conjunto de ameaças de segurança que estão incluídas em um sistema de referência que descreve riscos conhecidos publicamente. A lista de ameaças de CVE é mantida pela *MITRE Corporation*, uma organização sem fins lucrativos que administra centros de pesquisa e desenvolvimento patrocinados pelo governo federal dos EUA. O CVE é patrocinado pela *National Cyber Security Division (NCSD)* do Departamento de Segurança Interna dos EUA.

O servidor Apache foi reiniciado para carregar as novas configurações, e o script *testssl.sh* foi executado novamente para verificar a eficácia das mudanças. Informações adicionais sobre os resultados obtidos para análise podem ser consultadas no Apêndice I.3.

De acordo com os resultados apresentados na seção de **teste de vulnerabilidades**, a configuração testada obteve sucesso, superando todos os testes com sucesso. A Tabela 5.1 apresenta as vulnerabilidades evitadas, juntamente com os CVEs.

Tabela 5.1: Vulnerabilidades Evitadas e CVEs Associados

Vulnerabilidade	**CVE(s) Associado(s)**
Heartbleed	CVE-2014-0160
CCS	CVE-2014-0224
Ticketbleed	CVE-2016-9244 (experimental)
ROBOT	-
Secure Renegotiation	RFC 5746
Secure Client-Initiated Renegotiation	-
CRIME, TLS	CVE-2012-4929
BREACH	CVE-2013-3587
POODLE, SSL	CVE-2014-3566
TLS_FALLBACK_SCSV	RFC 7507
SWEET32	CVE-2016-2183, CVE-2016-6329
FREAK	CVE-2015-0204
DROWN	CVE-2016-0800, CVE-2016-0703
LOGJAM	CVE-2015-4000 (experimental)
BEAST	CVE-2011-3389
LUCKY13	CVE-2013-0169 (experimental)
RC4	CVE-2013-2566, CVE-2015-2808

5.2 RESULTADOS DA CONFIGURAÇÃO DE SEGURANÇA DO SERVIDOR POSTGRESQL

Dois testes foram realizados para certificar a segurança das comunicações no servidor de banco de dados PostgreSQL. No primeiro teste, a segurança TLS foi desabilitada, o que corresponde à configuração padrão em versões mais antigas do PostgreSQL. Durante o teste, foi executada uma consulta simples ao

banco de dados, e os resultados foram monitorados utilizando o analisador de tráfego de rede *Wireshark*. O código PHP utilizado para realizar a consulta está escrito no Pseudocódigo 25.

Algoritmo 25 Pseudocódigo Conexão e Consulta ao Banco de Dados PostgreSQL com PDO em PHP

```
Initialize database connection parameters
$user = 'secureVehicleTrackerOperator'
$password = '*************'
$host = 'localhost'
$db = 'vehicleTrackerEEE'                    ▷ Define the user credentials and connection details
Try to establish a connection to the PostgreSQL database using PDO
Set the Data Source Name (DSN) for the PostgreSQL connection
Create a PDO instance for database interaction with error handling enabled
if The connection is successful then
    Display message: "Conectado ao banco de dados $db com sucesso!"    ▷ Indicates a successful connection to the database
    Execute a query to retrieve a record from the tracking_data_simple table
    Fetch the result and store it in $tracking_data
    Display the first record of the fetched data    ▷ The first record from the table is printed
else
    Catch any exception and display the error message    ▷ Handles any errors during the connection or query execution
end if
Close the database connection    ▷ Ensures that the connection is properly closed after the operation
```

O resultado da consulta simples a uma linha dos dados de posicionamento do rastreador é apresentado no Pseudocódigo 26.

Algoritmo 26 Pseudocódigo Exemplo de Consulta ao Banco de Dados PostgreSQL - Dados de Rastreamento

```
Output message: "Conectado ao banco de dados $db com sucesso!"    ▷ Indicates that the database connection was established successfully
Display the contents of the retrieved database record
The record includes the following fields:
tds_id = 1
data_hora = '2024-04-28 17:01:30'
latitude = -25.476656333333
longitude = -49.2839595
altitude = 946.7
num_sats = 4
gps_tracker_id = 1    ▷ The retrieved data is displayed as an associative array, showing the tracking details
```

A Figura 5.2 mostra que os dados realmente são transmitidos em claro, sendo possível visualizar o comando **"SELECT * FROM tracking_data_simple LIMIT 1"** executado no banco de dados.

Os resultados da consulta ao banco de dados podem ser visualizados em uma marcação mais clara, destacando as respostas da consulta realizadas sem o uso de TLS. Isso evidencia a transmissão dos dados em texto simples, permitindo que as informações sejam acessadas de forma visível durante a comunicação.

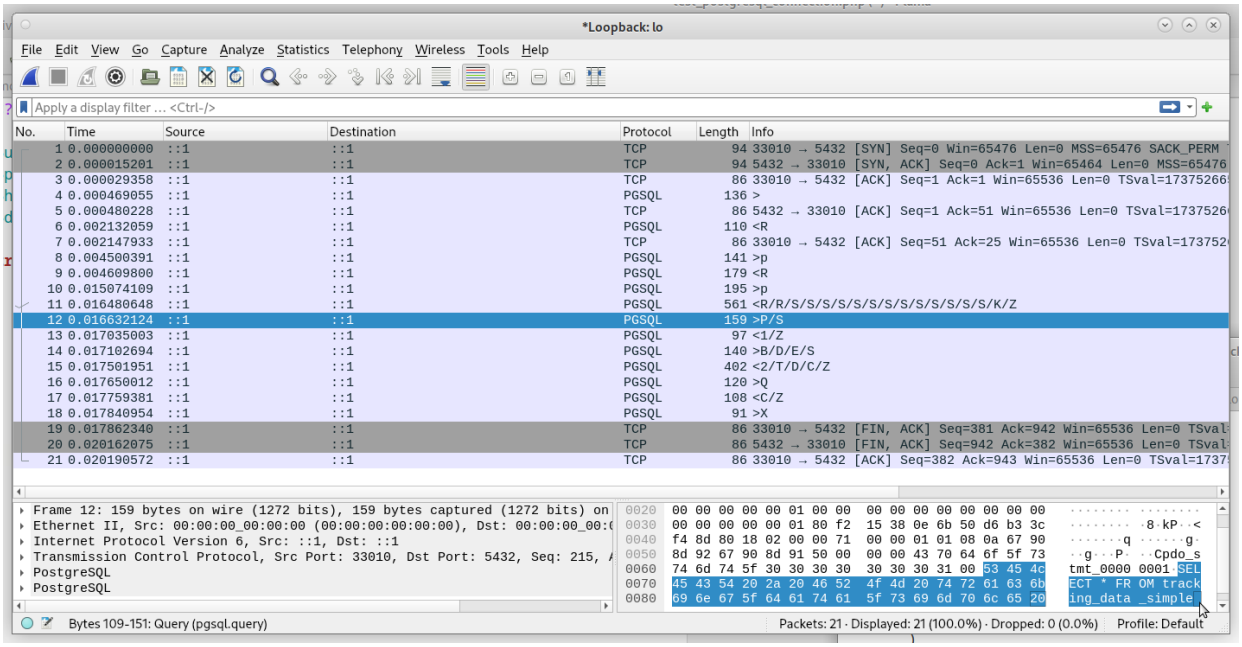

Figura 5.2: Resultado de uma consulta ao banco de dados PostgreSQL sem TLS - *query*

A Figura 5.3 demonstra este cenário, mostrando como os dados são apresentados, expondo as respostas obtidas diretamente da consulta ao banco de dados.

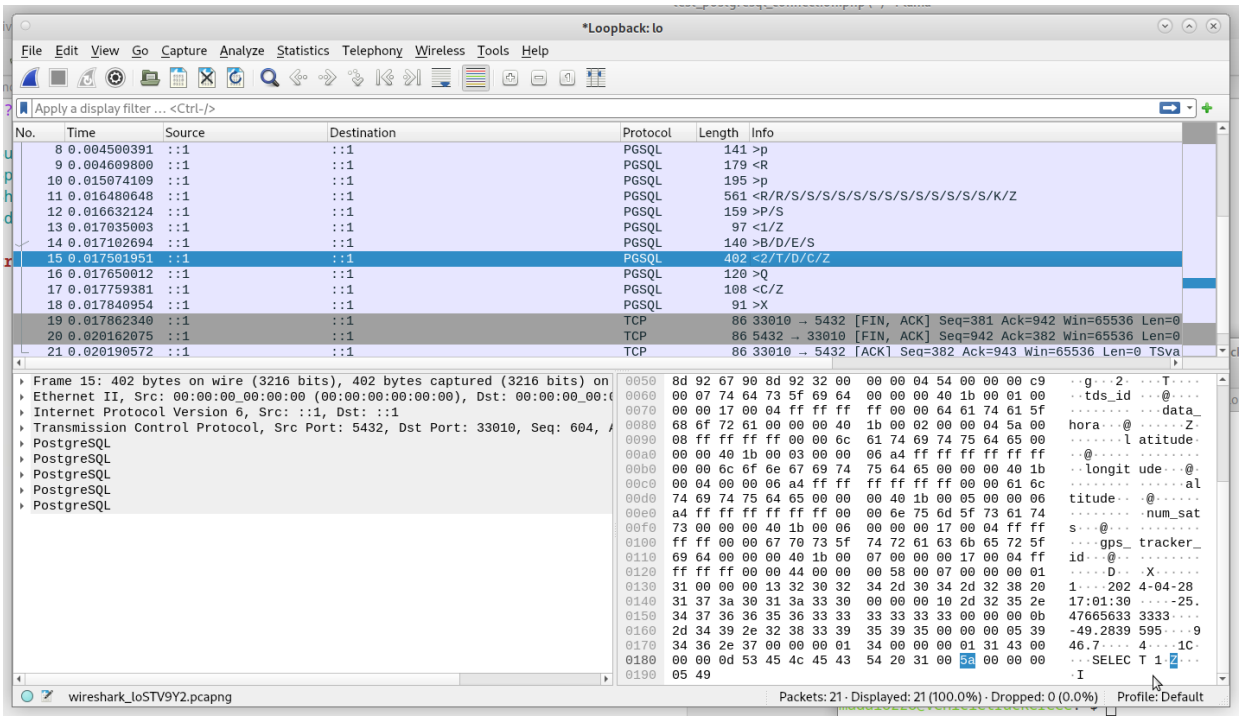

Figura 5.3: Resultado de uma consulta ao banco de dados PostgreSQL sem TLS - resposta a *query*

No segundo teste, as configurações de segurança TLS foram habilitadas no PostgreSQL conforme configurações sugeridas em 4.1.3. O resultado foi a efetiva utilização de criptografia utilizando o protocolo TLS 1.3 conforme Figura 5.4, não sendo mais possível decifrar de forma trivial nem a consulta ao banco

de dados nem os resultados.

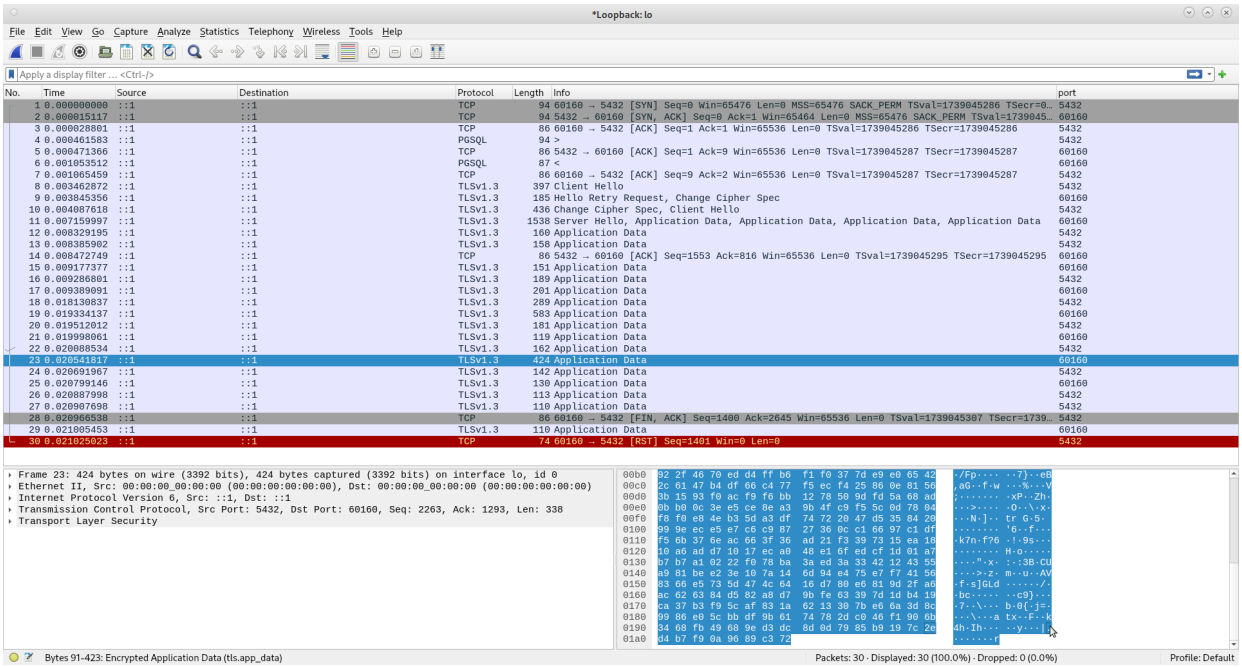

Figura 5.4: Resultado de uma consulta ao banco de dados PostgreSQL com TLS

5.3 RESULTADOS DA CONFIGURAÇÃO SEGURA DO MQTT SERVER

Para comprovação da efetividade da criptografia utilizando o protocolo TLS v1.3 nas configurações do Mosquitto MQTT Server, foram realizados dois testes. No primeiro deles, o servidor foi configurado para aceitar conexões utilizando o protocolo MQTT sem qualquer camada de segurança adicional. O arquivo de log do servidor foi excluído, o servidor reiniciado e, utilizando duas ferramentas básicas do próprio Mosquitto, um teste de envio de mensagem foi realizado.

Em um shell bash, foi disparado um comando para inscrição em um tópico chamado test, conforme ilustrado na Figura 5.5. Em outra aba, foi disparado um comando para o envio da mensagem "Teste de mensagem sem TLS", com tamanho de 25 bytes, como mostrado na Figura 5.6.

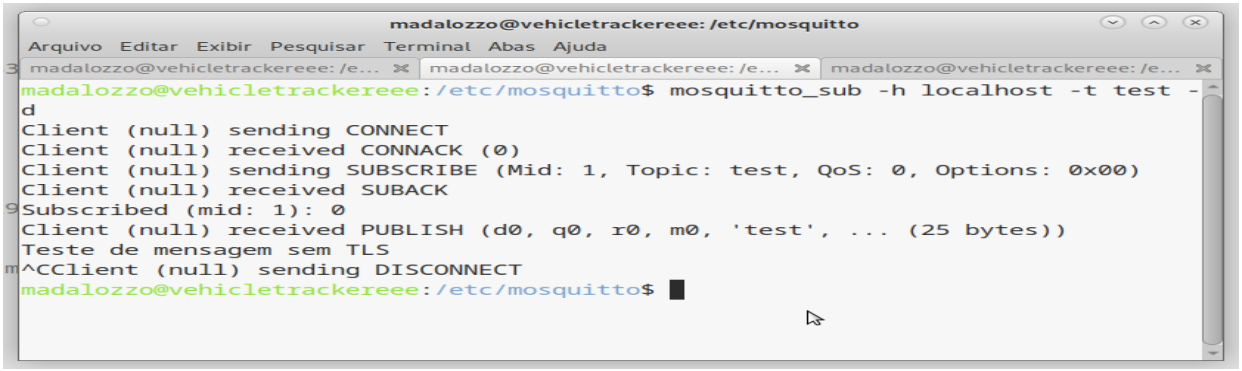

Figura 5.5: Inscrição no tópico **test** sem TLS

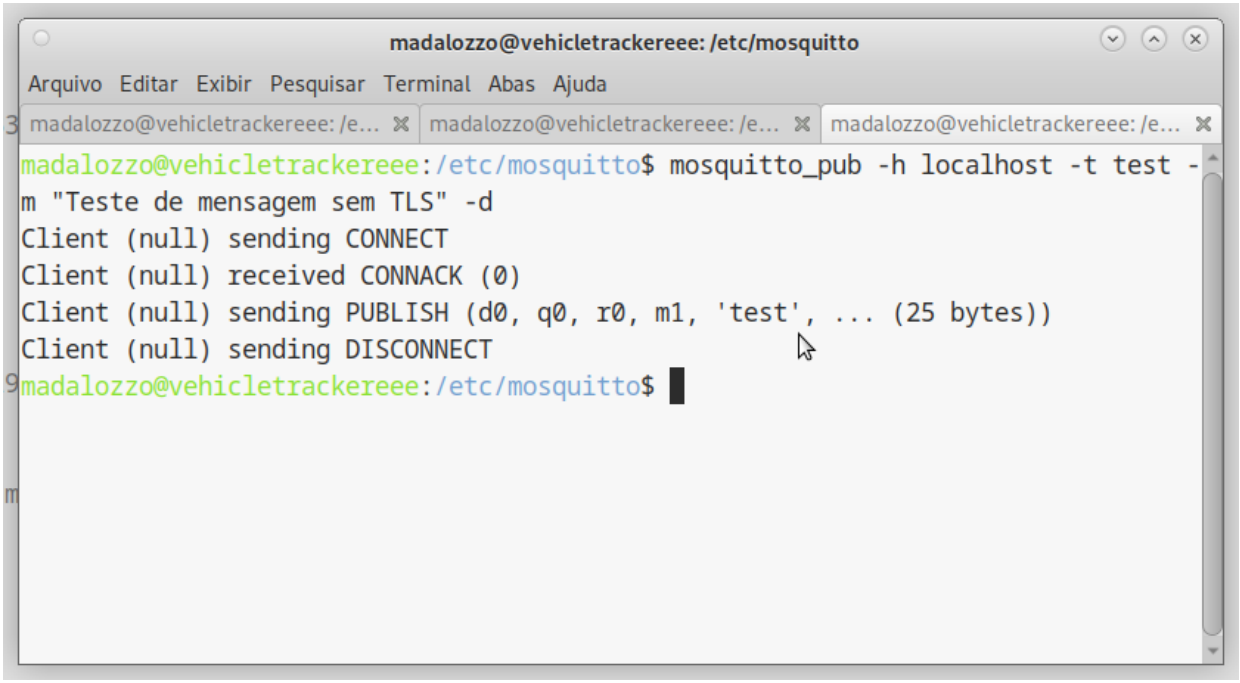

Figura 5.6: Envio de mensagem de teste sem TLS

Foi realizada a captura dos registros de log do servidor *Mosquitto* conforme Figura 5.7. Pode-se observar que na interação padrão houve efetivamente a troca de mensagem porém na porta padrão 1883 e sem a utilização de criptografia.

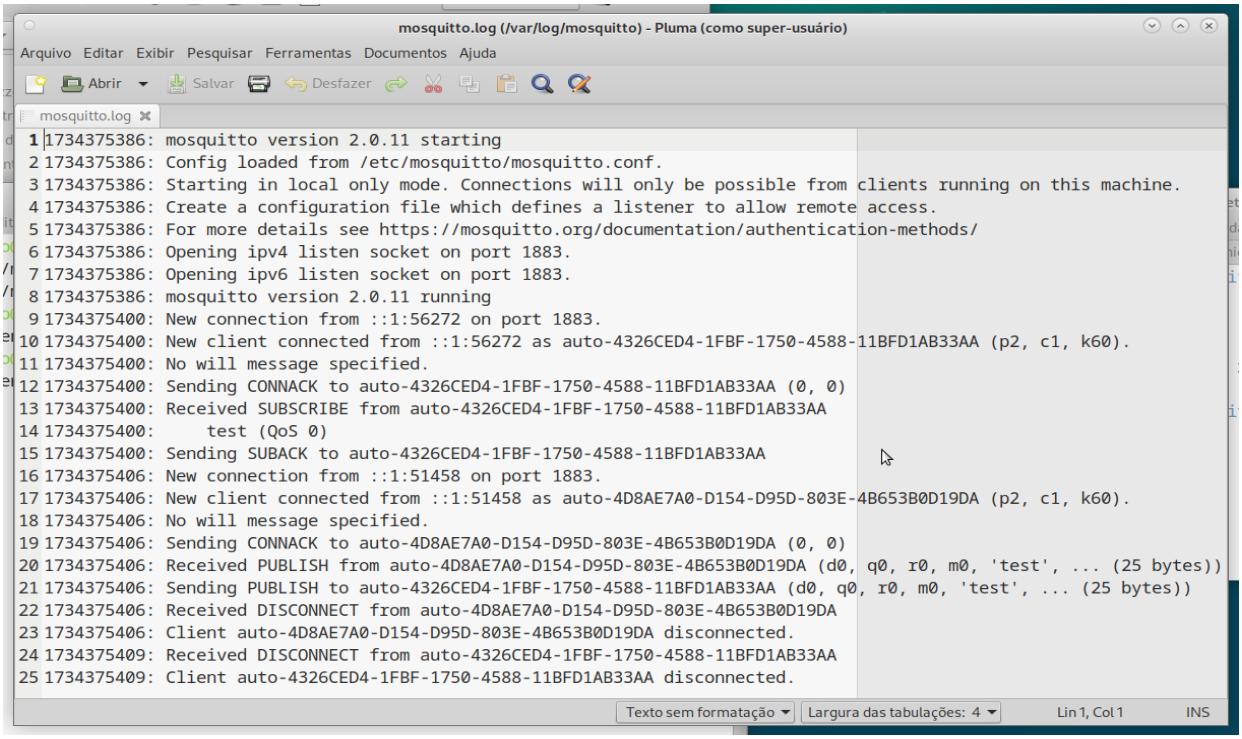

Figura 5.7: Log do servidor Mosquitto sem utilização do TLS

Foi utilizado o analisador de protocolos Wireshark versão 4.0.17 para verificação dos resultados con-

forme Figura 5.8. A mensagem é enviada em claro sem qualquer proteção adicional, ficando suscetível a ataques do tipo *man-in-the-middle*.

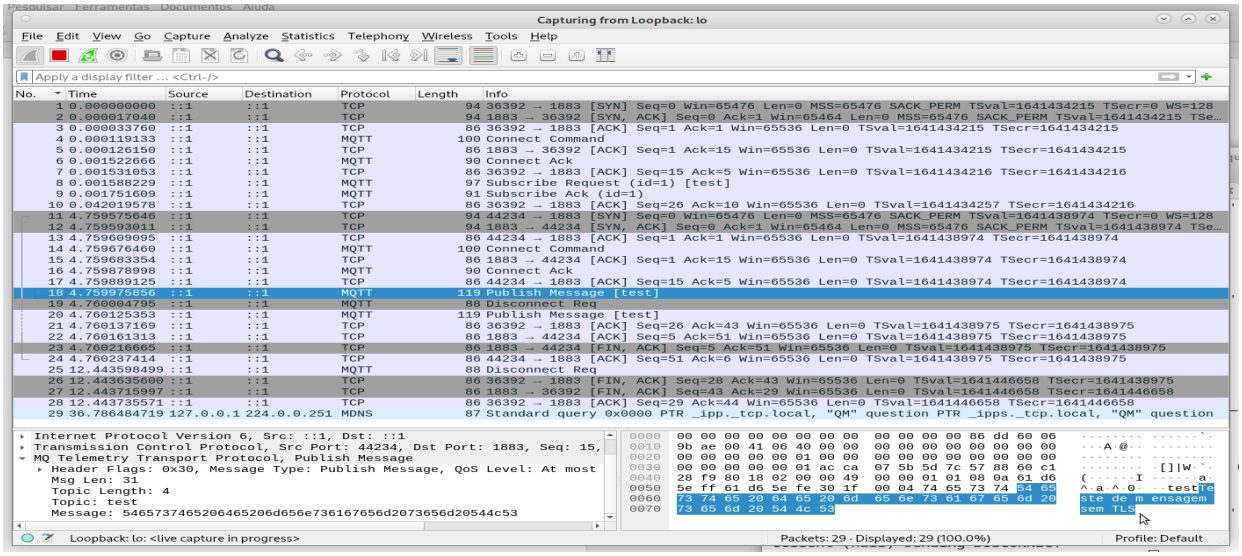

Figura 5.8: Wireshark - Mensagem MQTT sem TLS

No segundo teste, as configurações propostas na seção 4.1.4 foram utilizadas e de forma semelhante ao primeiro teste um *shell bash* foi aberto e executado o comando para inscrição em um tópico chamado **test** conforme Figura 5.9. Em outra aba foi disparado um comando para envio da mensagem "Teste de mensagem com TLS"com tamanho de 25 bytes conforme Figura 5.10.

Nos registros de log do servidor Mosquitto conforme Figura 5.11 observa-se que na interação padrão houve efetivamente a troca de mensagem na porta padrão 8883, padrão para comunicação TLS sobre o protocolo MQTT.

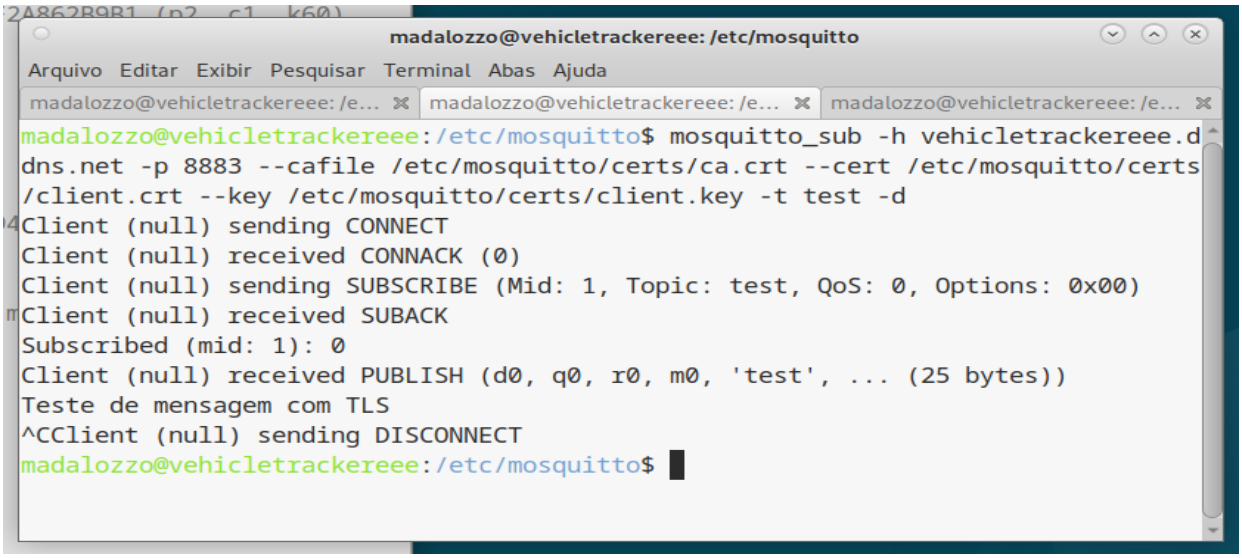

Figura 5.9: Inscrição no tópico **test** com TLS

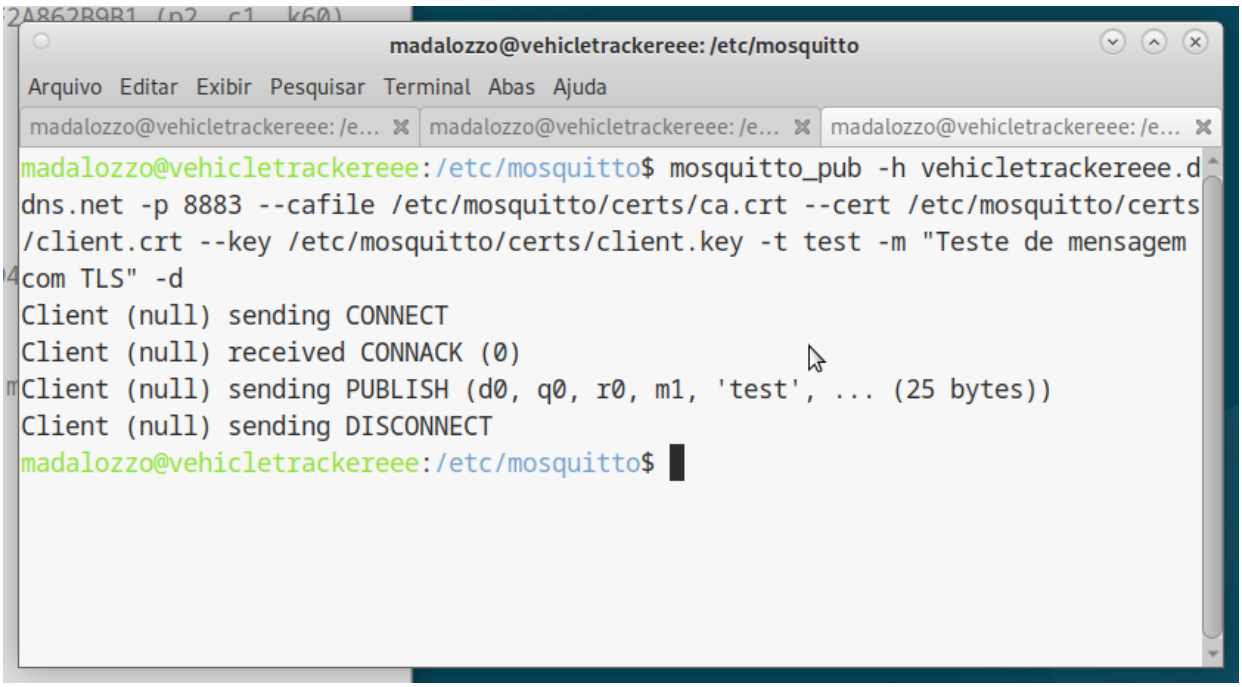

Figura 5.10: Envio de mensagem de teste com TLS

Os resultados para a configuração utilizando o TLS são apresentados na Figura 5.11. A mensagem é enviada com criptografia utilizando o protocolo TLS versão 1.3, garantindo o sigilo das comunicações de forma adequada.

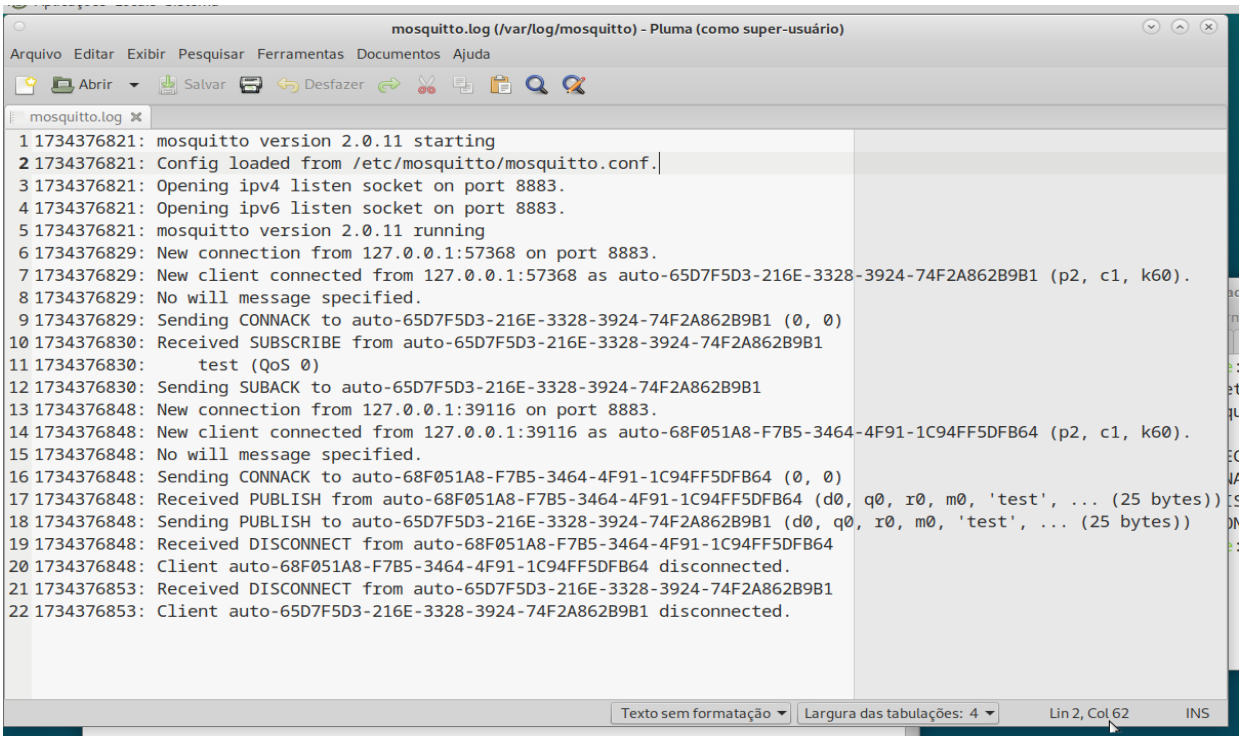

Figura 5.11: Log do servidor Mosquitto com utilização do TLS

A Figura 5.12 apresenta uma captura de tela do Wireshark, mostrando uma mensagem MQTT trans-

mitida utilizando o protocolo TLS. Nela, é possível observar os detalhes da comunicação segura entre os dispositivos, com os dados criptografados pelo TLS, assegurando a integridade e confidencialidade das informações. A visualização permite analisar os pacotes envolvidos na comunicação MQTT, destacando como o TLS garante a proteção dos dados durante a transmissão.

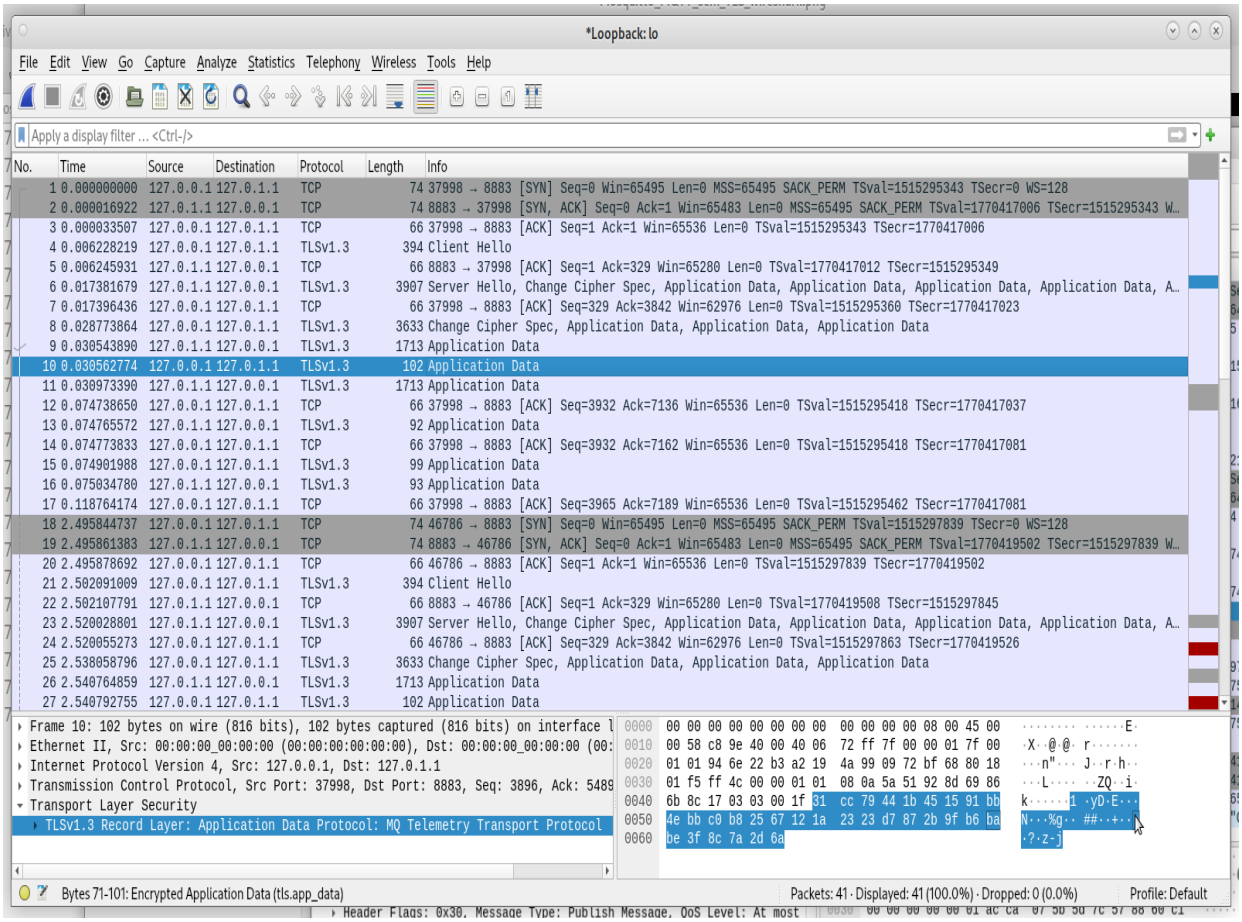

Figura 5.12: Wireshark - Mensagem MQTT com TLS

6 CONCLUSÃO E TRABALHOS FUTUROS

6.1 CONCLUSÃO

O crescente número de ameaças cibernéticas, especialmente direcionadas a dispositivos de IoT, juntamente com a falta de desenvolvimento de produtos e serviços com foco em segurança desde a sua concepção, tem causado prejuízos tanto a indivíduos quanto a empresas. Ataques cibernéticos bem-sucedidos contra alvos diversos podem gerar consequências como a paralisação parcial ou total de serviços, além de dificuldades operacionais em diferentes escalas. Além dos danos financeiros, é possível que ocorram prejuízos à imagem da empresa ou instituição afetada.

É fundamental ressaltar a importância de integrar a segurança de uma solução, seja ela implementada puramente em software, hardware ou em uma combinação de ambos, desde a sua concepção. Sempre que possível, deve-se considerar a adoção de um modelo de segurança multicamadas, conforme descrito por (62), especialmente em soluções de IoT. A simples utilização de criptografia de mercado não é suficiente, sendo imprescindível a análise criteriosa de protocolos, vulnerabilidades e as configurações adequadas. Mesmo tecnologias amplamente reconhecidas como essenciais para a segurança, como o protocolo TLS, podem apresentar falhas, seja por deficiências em sua concepção ou por implementações vulneráveis.

Este trabalho apresentou uma proposta de arquitetura para um rastreador veicular com criptografia de ponta a ponta, com foco na segurança em diversas camadas, como a infraestrutura do servidor, o banco de dados e os protocolos de comunicação entre o rastreador e a infraestrutura. A proposta visa mitigar problemas relacionados à segurança, garantindo uma abordagem robusta e segura para a solução.

6.2 TRABALHOS FUTUROS

Devido às limitações de hardware da versão do *Raspberry Pi 3 Model B*, que possui apenas uma porta serial, não foi possível integrar o módulo GSM/GPRS. Propõe-se, portanto, a utilização do *Raspberry Pi Zero 2W*, que conta com duas portas seriais, além de ser uma versão mais compacta, com consumo de energia significativamente menor. Para comunicação GSM/GPRS, seria incluído o módulo SIM 800L.

Além disso, sugere-se a construção de uma caixa modular para acomodar tanto o Raspberry Pi quanto os módulos GPS e GSM/GPRS, incluindo também compartimentos para a bateria e antenas. Para uma instalação permanente que não dependa de baterias externas e que seja amplamente compatível com veículos, poderia ser desenvolvida uma interface OBD2 para fornecer energia ao rastreador diretamente do veículo. A utilização correta da interface OBD2 oferece a vantagem de não alterar as características de fábrica dos veículos, preservando a garantia.

Adicionalmente, além de utilizar algoritmos de criptografia convencional, poderia ser desenvolvido um protótipo que aplicasse algoritmos criptográficos pós-quânticos, aumentando a segurança da comunicação.

REFERÊNCIAS BIBLIOGRÁFICAS

1 FORCE, U. S. S. *GPS: The Global Positioning System*. 2024. Disponível em: <https://www.gps.gov/>. Acesso em: 11/11/2024.

2 CORPORATION, T. A. *A Brief History of GPS*. 2024. Disponível em: <https://aerospace.org/article/brief-history-gps>. Acesso em: 11/11/2024.

3 FORCE, U. S. S. *Space Segment*. 2024. Disponível em: <https://www.gps.gov/systems/gps/space/>. Acesso em: 11/11/2024.

4 EARTH, P. C. of; GEOGRAPHY, M. S. D. of. *GEOG 862 - GPS AND GNSS FOR GEOSPATIAL PROFESSIONALS: Satellite Blocks*. 2024. Disponível em: <https://www.e-education.psu.edu/geog862/node/1775#:~:text=The%2011%20GPS%20satellites%20launched,last%20was%20decommissioned%20in%202007.> Acesso em: 11/11/2024.

5 LLC, S. *The History of GPS Vehicle Tracking*. 2021. Disponível em: <https://www.skypatrol.com/blog/the-history-of-gps-vehicle-tracking>. Acesso em: 11/11/2024.

6 HALL, R. *A History Of GPS Vehicle Tracking and Fleet Telematics*. 2016. Disponível em: <https://www.trackyourtruck.com/blog/history-gps-vehicle-tracking-and-fleet-telematics/>. Acesso em: 11/11/2024.

7 BASTOS, F. *Rastreadores: História e sua importância*. Disponível em: <https://gxtecnologia.com/rastreadores-historia-e-sua-importancia/>. Acesso em: 11/11/2024.

8 RIEBEEK, H. *Catalog of Earth Satellite Orbits*. 2009. Disponível em: <https://earthobservatory.nasa.gov/features/OrbitsCatalog/page1.php>. Acesso em: 11/11/2024.

9 MCDUFFIEPUBLISHED, J. *Why the Military Released GPS to the Public*. POPULAR MECHANICS, 2017. Disponível em: <https://www.popularmechanics.com/technology/gadgets/a26980/why-the-military-released-gps-to-the-public/>. Acesso em: 11/11/2024.

10 FORCE, U. S. S. *GPS Accuracy*. 2024. Disponível em: <https://www.gps.gov/systems/gps/performance/accuracy/>. Acesso em: 11/11/2024.

11 BASE, T. O. W. of L. A. A. F. *GPS ADVANCED CONTROL SEGMENT (OCX)*. SMC Public Affairs, 2011. Disponível em: <https://web.archive.org/web/20120503181621/http://www.losangeles.af.mil/library/factsheets/factsheet.asp?id=18676>. Acesso em: 11/11/2024.

12 MOORE, S. K. *Superaccurate GPS Chips Coming to Smartphones in 2018*. IEEE Spectrum, 2017. Disponível em: <https://spectrum.ieee.org/superaccurate-gps-chips-coming-to-smartphones-in-2018>. Acesso em: 11/11/2024.

13 HANACEK, N. *How Do You Measure Your Location Using GPS?* National Institute of Standards and Technology, 2021. Disponível em: <https://www.nist.gov/how-do-you-measure-it/how-do-you-measure-your-location-using-gps>. Acesso em: 11/11/2024.

14 FORCE, U. S. S. *The Global Positioning System*. 2021. Disponível em: <https://www.gps.gov/systems/gps/>. Acesso em: 11/11/2024.

15 FORCE, U. S. S. *GPS Applications*. 2014. Disponível em: <https://www.gps.gov/applications/>. Acesso em: 11/11/2024.

16 (NMEA), N. M. E. A. *Serial Data Networking - Latest Version of NMEA 0183 (Version 4.30)*. National Marine Electronics Association (NMEA), 2024. Disponível em: <https://www.nmea.org/nmea-0183.html>. Acesso em: 11/11/2024.

17 FRANçA, L. *Entendendo os dados NMEA*. GeoOne. Disponível em: <https://geoone.com.br/entendendo-nmea/>. Acesso em: 11/11/2024.

18 VAILSHERY, L. S. *Number of IoT connected devices worldwide 2019-2023, with forecasts to 2030*. 2023. Disponível em: <https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/>. Acesso em: 27/07/2023.

19 SHAH, Y.; SENGUPTA, S. A survey on classification of cyber-attacks on iot and iiot devices. In: IEEE. *2020 11th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*. [S.l.], 2020. p. 0406–0413.

20 MORGUNOV YAROSLAV SHMELEV, K. S. S. K. I. C. V. *Overview of IoT threats in 2023*. 2023. Disponível em: <https://securelist.com/iot-threat-report-2023/110644/>. Acesso em: 21/09/2023.

21 KASPERSKY. *Kaspersky unveils an overview of IoT-related threats in 2023*. Kaspersky, 2023. Disponível em: <https://www.kaspersky.com/about/press-releases/kaspersky-unveils-an-overview-of-iot-related-threats-in-2023>. Acesso em: 11/11/2024.

22 CELL, N. J. C. . C. I. *Mirai Botnet*. New Jersey Cybersecurity Communications Integration Cell, 2016. Disponível em: <https://web.archive.org/web/20161212084605/https://www.cyber.nj.gov/threat-profiles/botnet-variants/mirai-botnet>. Acesso em: 11/11/2024.

23 MALWAREMUSTDIE. *MMD-0056-2016 - Linux/Mirai, how an old ELF malcode is recycled.*. MalwareMustDie, 2016. Disponível em: <https://web.archive.org/web/20160905023500/http://blog.malwaremustdie.org/2016/08/mmd-0056-2016-linuxmirai-just.html>. Acesso em: 11/11/2024.

24 BIGGS, J. *Hackers release source code for a powerful DDoS app called Mirai*. TechCrunch, 2016. Disponível em: <https://techcrunch.com/2016/10/10/hackers-release-source-code-for-a-powerful-ddos-app-called-mirai/>. Acesso em: 11/11/2024.

25 MIESSLER AARON GUZMAN, V. R. C. S. D. *OWASP IoT Top 10*. 2018. Disponível em: <https://wiki.owasp.org/index.php/OWASP_Internet_of_Things_Project#tab=IoT_Top_10>. Acesso em: 21/05/2024.

26 JOHNSTON, N. *OWASP IoT Top 10. A gentle introduction and an exploration of root causes*. 2018. Disponível em: <https://owasp.org/www-chapter-toronto/assets/slides/2019-12-11-OWASP-IoT-Top-10---Introduction-and-Root-Causes.pdf>. Acesso em: 21/05/2024.

27 MQTT.ORG. *MQTT: The Standard for IoT Messaging*. MQTT.org, 2024. Disponível em: <https://mqtt.org/>. Acesso em: 11/11/2024.

28 OPEN, O. *MQTT Version 5.0*. OASIS Open. Disponível em: <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html>. Acesso em: 11/11/2024.

29 OASIS. *OASIS: About Us*. MQTT.org, 2024. Disponível em: <https://www.oasis-open.org/org/>. Acesso em: 11/11/2024.

30 JAHAN, N.; HOSSEN, K.; PATWARY, M. K. H. Implementation of a vehicle tracking system using smartphone and sms service. In: IEEE. *2017 4th International Conference on Advances in Electrical Engineering (ICAEE)*. [S.l.], 2017. p. 607–612.

31 HLAING, N. N. S.; NAING, M.; NAING, S. S. Gps and gsm based vehicle tracking system. *International Journal of Trend in Scientific Research and Development (IJTSRD)*, v. 3, n. 4, p. 271–275, 2019.

32 JETHWA, A. H.; SHEFFER, E.; THOMPSON, M. Vehicle tracking system using gps and gsm modem-a review. *Int. J. Recent Sci. Res*, v. 6, n. 6, p. 4805–4808, 2015.

33 CROFT, N. J.; OLIVIER, M. S. Using an approximated one-time pad to secure short messaging service (sms). In: *Proceedings of the Southern African Telecommunication Networks and Applications Conference. South Africa*. [S.l.: s.n.], 2005. p. 26–31.

34 TU, G.-H.; LI, C.-Y.; PENG, C.; LI, Y.; LU, S. New security threats caused by ims-based sms service in 4g lte networks. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. [S.l.: s.n.], 2016. p. 1118–1130.

35 TRAYNOR, P.; ENCK, W.; MCDANIEL, P.; PORTA, T. L. Exploiting open functionality in sms-capable cellular networks. *Journal of Computer Security*, IOS Press, v. 16, n. 6, p. 713–742, 2008.

36 PANNU, M.; BIRD, R.; GILL, B.; PATEL, K. Investigating vulnerabilities in gsm security. In: IEEE. *2015 International Conference and Workshop on Computing and Communication (IEMCON)*. [S.l.], 2015. p. 1–7.

37 TRAYNOR, P.; ENCK, W.; MCDANIEL, P.; PORTA, T. L. Mitigating attacks on open functionality in sms-capable cellular networks. In: *Proceedings of the 12th annual international conference on Mobile computing and networking*. [S.l.: s.n.], 2006. p. 182–193.

38 IBEKWE, I.; ALJAREH, S. Sms security: highlighting its vulnerabilities & techniques towards developing a solution. In: LIVERPOOL JOHN MOORES UNIVERSITY,. *13th Annual Post Graduate Network Symposium on the Convergence of Telecommunications, Networking and Broadcasting*. [S.l.], 2012.

39 LO, J. L.-C.; BISHOP, J.; ELOFF, J. H. Smssec: An end-to-end protocol for secure sms. *Computers & Security*, Elsevier, v. 27, n. 5-6, p. 154–167, 2008.

40 ALSHAMSI, H.; KËPUSKA, V.; ALSHAMSI, H. Real time vehicle tracking using arduino mega. *International Journal of Science and Technology*, v. 5, n. 12, p. 624–627, 2016.

41 KHIN, J. M. M.; OO, N. N. Real-time vehicle tracking system using arduino, gps, gsm and web-based technologies. *International Journal of Science and Engineering Applications*, v. 7, n. 11,433-436, 2018.

42 BARNES MARTIN THOMSON, A. P. A. L. R. *Deprecating Secure Sockets Layer Version 3.0*. Internet Engineering Task Force (IETF), 2015. Disponível em: <https://datatracker.ietf.org/doc/html/rfc7568>. Acesso em: 21/05/2024.

43 KOONTZ, N. *SSLClient*. 2024. Disponível em: <https://www.arduino.cc/reference/en/libraries/sslclient/>. Acesso em: 22/08/2024.

44 FOUNDATION, R. P. *Physical Computing with Python - GPIO pins*. Raspberry Pi Foundation, 2024. Disponível em: <https://projects.raspberrypi.org/en/projects/physical-computing/1>. Acesso em: 11/11/2024.

45 PINOUT, R. P. *The Raspberry Pi GPIO pinout guide.* Raspberry Pi Pinout, 2024. Disponível em: <https://pinout.xyz/>. Acesso em: 11/11/2024.

46 MIN, J. W. W.; JALI, N.; CHAI, W. Y. Vehicle tracking system. *International Journal of scientific & technology research*, v. 10, n. 09, 2021.

47 AL-SADOON, M. A. G.; ASIF, R.; AL-YASIR, Y. I. A.; ABD-ALHAMEED, R. A.; EXCELL, P. S. Aoa localization for vehicle-tracking systems using a dual-band sensor array. *IEEE Transactions on Antennas and Propagation*, IEEE, v. 68, n. 8, p. 6330–6345, 2020.

48 LU, J.; XIN, Y.; ZHANG, Z.; PENG, H.; HAN, J. Supporting user authorization queries in rbac systems by role–permission reassignment. *Future Generation Computer Systems*, Elsevier, v. 88, p. 707–717, 2018.

49 ÇORAK, B. H.; OKAY, F. Y.; GÜZEL, M.; MURT, Ş.; OZDEMIR, S. Comparative analysis of iot communication protocols. In: IEEE. *2018 International symposium on networks, computers and communications (ISNCC)*. [S.l.], 2018. p. 1–6.

50 MISHRA, B.; KERTESZ, A. The use of mqtt in m2m and iot systems: A survey. *Ieee Access*, IEEE, v. 8, p. 201071–201086, 2020.

51 ANTHRAPER, J. J.; KOTAK, J. Security, privacy and forensic concern of mqtt protocol. In: *Proceedings of International Conference on Sustainable Computing in Science, Technology and Management (SUSCOM), Amity University Rajasthan, Jaipur-India*. [S.l.: s.n.], 2019.

52 FOUNDATION, E. *mosquitto-tls — Configure SSL/TLS support for Mosquitto*. Eclipse Foundation. Disponível em: <https://mosquitto.org/man/mosquitto-tls-7.html>. Acesso em: 11/11/2024.

53 BLOG, G. *[Tutorial] How to Set Up a Mosquitto MQTT Broker Securely — Using Client Certificates*. Gravio Blog. Disponível em: <https://www.gravio.com/en-blog/tutorial-how-to-set-up-a-mosquitto-mqtt-broker-securely----using-client-certificates>. Acesso em: 11/11/2024.

54 YE, S.; DAI, K.; FAN, G.; ZHANG, L.; LIANG, Z. Exploring the intersection of network security and database communication: a postgresql socket connection case study. *Transactions on Computer Science and Intelligent Systems Research*, v. 3, p. 1–9, 2024.

55 GROUP, P. *Constantes pré-definidas de erros e registros de log*. 2024. Disponível em: <https://www.php.net/manual/pt_BR/errorfunc.constants.php>. Acesso em: 21/05/2024.

56 KEITHYEH. *How to create a self-signed SSL Certificate with SubjectAltName(SAN)*. GitHub, Inc., 2024. Disponível em: <https://gist.github.com/KeithYeh/bb07cadd23645a6a62509b1ec8986bbc>. Acesso em: 11/11/2024.

57 TESTSSL.SH. *Testing TLS/SSL encryption*. testssl.sh, 2024. Disponível em: <https://testssl.sh/>. Acesso em: 11/11/2024.

58 FOUNDATION, T. A. S. *SSL/TLS Strong Encryption: How-To*. The Apache Software Foundation, 2023. Disponível em: <https://httpd.apache.org/docs/trunk/ssl/ssl_howto.html>. Acesso em: 11/11/2024.

59 OVERFLOW, S. *How can I disable Cipher Suites for Apache2*. Stack Overflow, 2022. Disponível em: <https://stackoverflow.com/questions/71113480/how-can-i-disable-cipher-suites-for-apache2>. Acesso em: 11/11/2024.

60 HODGES PAYPAL, C. J. C. M. U. A. B. G. I. J. *RFC 6797*. Internet Engineering Task Force (IETF), 2012. Disponível em: <https://datatracker.ietf.org/doc/html/rfc6797>. Acesso em: 11/11/2024.

61 CORPORATION, T. M. *CVE*. The MITRE Corporation, 2023. Disponível em: <https://www.cve.org/>. Acesso em: 11/11/2024.

62 YANG, X.; LI, Z.; GENG, Z.; ZHANG, H. A multi-layer security model for internet of things. In: SPRINGER. *Internet of Things: International Workshop, IOT 2012, Changsha, China, August 17-19, 2012. Proceedings*. [S.l.], 2012. p. 388–393.

APÊNDICES

I.1 COMANDOS UTILIZADOS PARA CRIAÇÃO DO CERTIFICADO DIGITAL AUTOASSINADO

```
madalozzo@vehicletrackereee:~/certificate$ openssl version
OpenSSL 3.0.14 4 Jun 2024 (Library: OpenSSL 3.0.14 4 Jun 2024)
madalozzo@vehicletrackereee:~/certificate$ openssl genrsa -aes256 -out
↪  vehicletrackereee.private.key 4096
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
madalozzo@vehicletrackereee:~/certificate$ openssl req -key
↪  vehicletrackereee.private.key -new -out vehicletrackereee.csr
Enter pass phrase for vehicletrackereee.private.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:BR
State or Province Name (full name) [Some-State]:PR
Locality Name (eg, city) []:Curitiba
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Locl alSecu urity
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:vehicletrackereee.ddns.net
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
madalozzo@vehicletrackereee:~/certificate$ openssl x509 -signkey
↪  vehicletrackereee.private.key -in vehicletrackereee.csr -req -days 365 -out
↪  vehicletrackereee.crt -sha256 -extfile v3.ext
Enter pass phrase for vehicletrackereee.private.key:
Certificate request self-signature ok
subject=C = BR, ST = PR, L = Curitiba, O = LocalSecurity, CN =
↪  vehicletrackereee.ddns.net
madalozzo@vehicletrackereee:~/certificate$ ls -l
total 16
-rw-r--r-- 1 madalozzo madalozzo    0 out 30 10:57 typescript
```

```
-rw-r--r-- 1 madalozzo madalozzo  328 out 30 10:57 v3.ext
-rw-r--r-- 1 madalozzo madalozzo 2350 out 30 10:58 vehicletrackereee.crt
-rw-r--r-- 1 madalozzo madalozzo 1700 out 30 10:58 vehicletrackereee.csr
-rw------- 1 madalozzo madalozzo 3434 out 30 10:57 vehicletrackereee.private.key
madalozzo@vehicletrackereee:~/certificate$

exit
```

I.2 RESULTADOS OBTIDOS COM A EXECUÇÃO DO TESTSSH.SH

```
###################################################################
    testssl.sh version 3.0.9 from https://testssl.sh/

  This program is free software. Distribution and modification under
  GPLv2 permitted. USAGE w/o ANY WARRANTY. USE IT AT YOUR OWN RISK!

    Please file bugs @ https://testssl.sh/bugs/

###################################################################

 Using bash 5.2.15.  OpenSSL 1.0.2-bad (1.0.2k-dev) [~183 ciphers]
    on vehicletrackereee:./bin/openssl.Linux.x86_64
    (built: Sep  1 14:03:44 2022, platform: linux-x86_64)

 Start 2024-10-29 16:22:27        -->> 127.0.1.1:443 (vehicletrackereee.ddns.net) <<--

 A record via:           /etc/hosts
 rDNS (127.0.1.1):       --
 Service detected:       HTTP


 Testing protocols via sockets except NPN+ALPN

 SSLv2      not offered (OK)
 SSLv3      not offered (OK)
 TLS 1      not offered
 TLS 1.1    not offered
 TLS 1.2    offered (OK)
 TLS 1.3    offered (OK): final
 NPN/SPDY   not offered
 ALPN/HTTP2 http/1.1 (offered)

 Testing cipher categories

 NULL ciphers (no encryption)                not offered (OK)
 Anonymous NULL Ciphers (no authentication)  not offered (OK)
 Export ciphers (w/o ADH+NULL)               not offered (OK)
 LOW: 64 Bit + DES, RC[2,4] (w/o export)     not offered (OK)
 Triple DES Ciphers / IDEA                   not offered
 Obsolete CBC ciphers (AES, ARIA etc.)       offered
 Strong encryption (AEAD ciphers)            offered (OK)


 Testing robust (perfect) forward secrecy, (P)FS -- omitting Null Authentication/Encryption, 3DES, RC4

 PFS is offered (OK)          TLS_AES_256_GCM_SHA384 TLS_CHACHA20_POLY1305_SHA256 ECDHE-RSA-AES256-GCM-SHA384 ECDHE-RSA-AES256-SHA384 ECDHE-RSA-AES256-SHA DHE-RSA-AES256-GCM-SHA384
                              ECDHE-RSA-CHACHA20-POLY1305 DHE-RSA-CHACHA20-POLY1305 DHE-RSA-AES256-CCM8 DHE-RSA-AES256-CCM DHE-RSA-AES256-SHA256 DHE-RSA-AES256-SHA
                              ECDHE-RSA-CAMELLIA256-SHA384 DHE-RSA-CAMELLIA256-SHA256 DHE-RSA-CAMELLIA256-SHA DHE-RSA-ARIA256-GCM-SHA384 ECDHE-ARIA256-GCM-SHA384 TLS_AES_128_GCM_SHA256
                              ECDHE-RSA-AES128-GCM-SHA256 ECDHE-RSA-AES128-SHA256 ECDHE-RSA-AES128-SHA DHE-RSA-AES128-GCM-SHA256 DHE-RSA-AES128-CCM8 DHE-RSA-AES128-CCM DHE-RSA-AES128-SHA256
                              DHE-RSA-AES128-SHA ECDHE-RSA-CAMELLIA128-SHA256 DHE-RSA-CAMELLIA128-SHA256 DHE-RSA-CAMELLIA128-SHA DHE-RSA-ARIA128-GCM-SHA256 ECDHE-ARIA128-GCM-SHA256
 Elliptic curves offered:     prime256v1 secp384r1 secp521r1 X25519 X448
 Finite field group:          ffdhe2048 ffdhe3072 ffdhe4096 ffdhe6144 ffdhe8192

 Testing server preferences

 Has server cipher order?     no (NOT ok)
 Negotiated protocol          TLSv1.3
 Negotiated cipher            TLS_AES_256_GCM_SHA384, 253 bit ECDH (X25519) (limited sense as client will pick)
 Negotiated cipher per proto  (limited sense as client will pick)
     ECDHE-RSA-AES256-GCM-SHA384:   TLSv1.2
     TLS_AES_128_GCM_SHA256:        TLSv1.3
 No further cipher order check has been done as order is determined by the client


 Testing server defaults (Server Hello)

 TLS extensions (standard)    "renegotiation info/#65281" "server name/#0" "EC point formats/#11" "supported versions/#43" "key share/#51" "supported_groups/#10" "max fragment length/#1"
                              "application layer protocol negotiation/#16" "encrypt-then-mac/#22" "extended master secret/#23"
 Session Ticket RFC 5077 hint no -- no lifetime advertised
 SSL Session ID support       yes
 Session Resumption           Tickets no, ID: yes
 TLS clock skew               Random values, no fingerprinting possible
 Signature Algorithm          SHA256 with RSA
 Server key size              RSA 4096 bits
 Server key usage             Digital Signature, Non Repudiation, Key Encipherment, Data Encipherment, Key Agreement, Certificate Sign
 Server extended key usage    --
 Serial                       5EAA2A209C37FC08A1A2BE987DC5D443C27AA551 (OK: length 20)
 Fingerprints                 SHA1 2E26A0FAC97F0A79CD0FEE6CBBE74DF59F445CB6
                              SHA256 64156BDD2C1167898459B87DBB11E384F9325C90002D3951F0CFFD95FAB9E2FC
 Common Name (CN)             vehicletrackereee.ddns.net
 subjectAltName (SAN)         vehicletrackereee.ddns.net
 Issuer                       vehicletrackereee.ddns.net (LocalSecurity from BR)
 Trust (hostname)             Ok via SAN (same w/o SNI)
 Chain of trust               NOT ok (self signed)
 EV cert (experimental)       no
 ETS/"eTLS", visibility info  not present
 Certificate Validity (UTC)   361 >= 60 days (2024-10-25 19:54 --> 2025-10-25 19:54)
 # of certificates provided   1
 Certificate Revocation List  --
 OCSP URI                     --
                              NOT ok -- neither CRL nor OCSP URI provided
 OCSP stapling                not offered
 OCSP must staple extension   --
 DNS CAA RR (experimental)    not offered
 Certificate Transparency     --


 Testing HTTP header response @ "/"

 HTTP Status Code             200 OK
 HTTP clock skew              0 sec from localtime
 Strict Transport Security    not offered
 Public Key Pinning           --
 Server banner                Apache/2.4.62 (Debian)
 Application banner           --
 Cookie(s)                    (none issued at "/")
 Security headers             --
 Reverse Proxy banner         --


 Testing vulnerabilities

 Heartbleed (CVE-2014-0160)               not vulnerable (OK), no heartbeat extension
 CCS (CVE-2014-0224)                      not vulnerable (OK)
 Ticketbleed (CVE-2016-9244), experiment.  not vulnerable (OK), no session ticket extension
 ROBOT                                    not vulnerable (OK)
```

Secure Renegotiation (RFC 5746)	**supported (OK)**
Secure Client-Initiated Renegotiation	not vulnerable (OK)
CRIME, TLS (CVE-2012-4929)	not vulnerable (OK)
BREACH (CVE-2013-3587)	potentially NOT ok, "gzip" HTTP compression detected. - only supplied "/" tested Can be ignored for static pages or if no secrets in the page
POODLE, SSL (CVE-2014-3566)	**not vulnerable (OK)**, no SSLv3 support
TLS_FALLBACK_SCSV (RFC 7507)	No fallback possible (OK), no protocol below TLS 1.2 offered
SWEET32 (CVE-2016-2183, CVE-2016-6329)	**not vulnerable (OK)**
FREAK (CVE-2015-0204)	**not vulnerable (OK)**
DROWN (CVE-2016-0800, CVE-2016-0703)	**not vulnerable on this host and port (OK)** make sure you don't use this certificate elsewhere with SSLv2 enabled services https://search.censys.io/search?resource=hosts&virtual_hosts=INCLUDE&q=64156BDD2C1167898459B87DBB11E384F9325C90002D3951F0CFFD95FAB9E2FC
LOGJAM (CVE-2015-4000), experimental	common prime with 4096 bits detected: *RFC3526/Oakley Group 16* (4096 bits), but no DH EXPORT ciphers
BEAST (CVE-2011-3389)	not vulnerable (OK), no SSL3 or TLS1
LUCKY13 (CVE-2013-0169), experimental	potentially **VULNERABLE**, uses cipher block chaining (CBC) ciphers with TLS. Check patches
RC4 (CVE-2013-2566, CVE-2015-2808)	no RC4 ciphers detected (OK)

Testing 370 ciphers via OpenSSL plus sockets against the server, ordered by encryption strength

Hexcode	Cipher Suite Name (OpenSSL)	KeyExch.	Encryption	Bits	Cipher Suite Name (IANA/RFC)
x1302	TLS_AES_256_GCM_SHA384	ECDH 253	AESGCM	256	TLS_AES_256_GCM_SHA384
x1303	TLS_CHACHA20_POLY1305_SHA256	ECDH 253	ChaCha20	256	TLS_CHACHA20_POLY1305_SHA256
xc030	ECDHE-RSA-AES256-GCM-SHA384	ECDH 521	AESGCM	256	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
xc028	ECDHE-RSA-AES256-SHA384	ECDH 521	AES	256	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
xc014	ECDHE-RSA-AES256-SHA	ECDH 521	AES	256	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
x9f	DHE-RSA-AES256-GCM-SHA384	DH 4096	AESGCM	256	TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
xcca8	ECDHE-RSA-CHACHA20-POLY1305	ECDH 521	ChaCha20	256	TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256
xccaa	DHE-RSA-CHACHA20-POLY1305	DH 4096	ChaCha20	256	TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256
xc0a3	DHE-RSA-AES256-CCM8	DH 4096	AESCCM8	256	TLS_DHE_RSA_WITH_AES_256_CCM_8
xc09f	DHE-RSA-AES256-CCM	DH 4096	AESCCM	256	TLS_DHE_RSA_WITH_AES_256_CCM
x6b	DHE-RSA-AES256-SHA256	DH 4096	AES	256	TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
x39	DHE-RSA-AES256-SHA	DH 4096	AES	256	TLS_DHE_RSA_WITH_AES_256_CBC_SHA
xc077	ECDHE-RSA-CAMELLIA256-SHA384	ECDH 521	Camellia	256	TLS_ECDHE_RSA_WITH_CAMELLIA_256_CBC_SHA384
xc4	DHE-RSA-CAMELLIA256-SHA256	DH 4096	Camellia	256	TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA256
x88	DHE-RSA-CAMELLIA256-SHA	DH 4096	Camellia	256	TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA
x9d	AES256-GCM-SHA384	RSA	AESGCM	256	TLS_RSA_WITH_AES_256_GCM_SHA384
xc0a1	AES256-CCM8	RSA	AESCCM8	256	TLS_RSA_WITH_AES_256_CCM_8
xc09d	AES256-CCM	RSA	AESCCM	256	TLS_RSA_WITH_AES_256_CCM
x3d	AES256-SHA256	RSA	AES	256	TLS_RSA_WITH_AES_256_CBC_SHA256
x35	AES256-SHA	RSA	AES	256	TLS_RSA_WITH_AES_256_CBC_SHA
xc0	CAMELLIA256-SHA256	RSA	Camellia	256	TLS_RSA_WITH_CAMELLIA_256_CBC_SHA256
x84	CAMELLIA256-SHA	RSA	Camellia	256	TLS_RSA_WITH_CAMELLIA_256_CBC_SHA
xc051	ARIA256-GCM-SHA384	RSA	ARIAGCM	256	TLS_RSA_WITH_ARIA_256_GCM_SHA384
xc053	DHE-RSA-ARIA256-GCM-SHA384	DH 4096	ARIAGCM	256	TLS_DHE_RSA_WITH_ARIA_256_GCM_SHA384
xc061	ECDHE-ARIA256-GCM-SHA384	ECDH 521	ARIAGCM	256	TLS_ECDHE_RSA_WITH_ARIA_256_GCM_SHA384
x1301	TLS_AES_128_GCM_SHA256	ECDH 253	AESGCM	128	TLS_AES_128_GCM_SHA256
xc02f	ECDHE-RSA-AES128-GCM-SHA256	ECDH 521	AESGCM	128	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
xc027	ECDHE-RSA-AES128-SHA256	ECDH 521	AES	128	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
xc013	ECDHE-RSA-AES128-SHA	ECDH 521	AES	128	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
x9e	DHE-RSA-AES128-GCM-SHA256	DH 4096	AESGCM	128	TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
xc0a2	DHE-RSA-AES128-CCM8	DH 4096	AESCCM8	128	TLS_DHE_RSA_WITH_AES_128_CCM_8
xc09e	DHE-RSA-AES128-CCM	DH 4096	AESCCM	128	TLS_DHE_RSA_WITH_AES_128_CCM
xc0a0	AES128-CCM8	RSA	AESCCM8	128	TLS_RSA_WITH_AES_128_CCM_8
xc09c	AES128-CCM	RSA	AESCCM	128	TLS_RSA_WITH_AES_128_CCM
x67	DHE-RSA-AES128-SHA256	DH 4096	AES	128	TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
x33	DHE-RSA-AES128-SHA	DH 4096	AES	128	TLS_DHE_RSA_WITH_AES_128_CBC_SHA
xc076	ECDHE-RSA-CAMELLIA128-SHA256	ECDH 521	Camellia	128	TLS_ECDHE_RSA_WITH_CAMELLIA_128_CBC_SHA256
xbe	DHE-RSA-CAMELLIA128-SHA256	DH 4096	Camellia	128	TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA256
x45	DHE-RSA-CAMELLIA128-SHA	DH 4096	Camellia	128	TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA
x9c	AES128-GCM-SHA256	RSA	AESGCM	128	TLS_RSA_WITH_AES_128_GCM_SHA256
x3c	AES128-SHA256	RSA	AES	128	TLS_RSA_WITH_AES_128_CBC_SHA256
x2f	AES128-SHA	RSA	AES	128	TLS_RSA_WITH_AES_128_CBC_SHA
xba	CAMELLIA128-SHA256	RSA	Camellia	128	TLS_RSA_WITH_CAMELLIA_128_CBC_SHA256
x41	CAMELLIA128-SHA	RSA	Camellia	128	TLS_RSA_WITH_CAMELLIA_128_CBC_SHA
xc050	ARIA128-GCM-SHA256	RSA	ARIAGCM	128	TLS_RSA_WITH_ARIA_128_GCM_SHA256
xc052	DHE-RSA-ARIA128-GCM-SHA256	DH 4096	ARIAGCM	128	TLS_DHE_RSA_WITH_ARIA_128_GCM_SHA256
xc060	ECDHE-ARIA128-GCM-SHA256	ECDH 521	ARIAGCM	128	TLS_ECDHE_RSA_WITH_ARIA_128_GCM_SHA256

Running client simulations (HTTP) via sockets

Android 6.0	TLSv1.2 ECDHE-RSA-AES128-GCM-SHA256, 256 bit ECDH (P-256)
Android 7.0 (native)	TLSv1.2 ECDHE-RSA-AES128-GCM-SHA256, 256 bit ECDH (P-256)
Android 8.1 (native)	TLSv1.2 ECDHE-RSA-AES128-GCM-SHA256, 253 bit ECDH (X25519)
Android 9.0 (native)	TLSv1.3 TLS_AES_128_GCM_SHA256, 253 bit ECDH (X25519)
Android 10.0 (native)	TLSv1.3 TLS_AES_128_GCM_SHA256, 253 bit ECDH (X25519)
Android 11 (native)	TLSv1.3 TLS_AES_128_GCM_SHA256, 253 bit ECDH (X25519)
Android 12 (native)	TLSv1.3 TLS_AES_128_GCM_SHA256, 253 bit ECDH (X25519)
Chrome 79 (Win 10)	TLSv1.3 TLS_AES_128_GCM_SHA256, 253 bit ECDH (X25519)
Chrome 101 (Win 10)	TLSv1.3 TLS_AES_128_GCM_SHA256, 253 bit ECDH (X25519)
Firefox 66 (Win 8.1/10)	TLSv1.3 TLS_AES_128_GCM_SHA256, 253 bit ECDH (X25519)
Firefox 100 (Win 10)	TLSv1.3 TLS_AES_128_GCM_SHA256, 253 bit ECDH (X25519)
IE 6 XP	No connection
IE 8 Win 7	No connection
IE 8 XP	No connection
IE 11 Win 7	TLSv1.2 ECDHE-RSA-AES256-SHA384, 256 bit ECDH (P-256)
IE 11 Win 8.1	TLSv1.2 ECDHE-RSA-AES256-SHA384, 256 bit ECDH (P-256)
IE 11 Win Phone 8.1	TLSv1.2 AES128-SHA256, No FS
IE 11 Win 10	TLSv1.2 ECDHE-RSA-AES256-GCM-SHA384, 256 bit ECDH (P-256)
Edge 15 Win 10	TLSv1.2 ECDHE-RSA-AES256-GCM-SHA384, 253 bit ECDH (X25519)
Edge 101 Win 10 21H2	TLSv1.3 TLS_AES_128_GCM_SHA256, 253 bit ECDH (X25519)
Safari 12.1 (iOS 12.2)	TLSv1.3 TLS_CHACHA20_POLY1305_SHA256, 253 bit ECDH (X25519)
Safari 13.0 (macOS 10.14.6)	TLSv1.3 TLS_CHACHA20_POLY1305_SHA256, 253 bit ECDH (X25519)
Safari 15.4 (macOS 12.3.1)	TLSv1.3 TLS_AES_128_GCM_SHA256, 253 bit ECDH (X25519)
Java 7u25	No connection
Java 8u161	TLSv1.2 ECDHE-RSA-AES256-SHA384, 256 bit ECDH (P-256)
Java 11.0.2 (OpenJDK)	TLSv1.3 TLS_AES_128_GCM_SHA256, 256 bit ECDH (P-256)
Java 17.0.3 (OpenJDK)	TLSv1.3 TLS_AES_256_GCM_SHA384, 253 bit ECDH (X25519)
go 1.17.8	TLSv1.3 TLS_AES_128_GCM_SHA256, 253 bit ECDH (X25519)
LibreSSL 2.8.3 (Apple)	TLSv1.2 ECDHE-RSA-CHACHA20-POLY1305, 253 bit ECDH (X25519)
OpenSSL 1.0.2e	TLSv1.2 ECDHE-RSA-AES256-GCM-SHA384, 256 bit ECDH (P-256)
OpenSSL 1.1.0l (Debian)	TLSv1.2 ECDHE-RSA-AES256-GCM-SHA384, 253 bit ECDH (X25519)
OpenSSL 1.1.1d (Debian)	TLSv1.3 TLS_AES_256_GCM_SHA384, 253 bit ECDH (X25519)
OpenSSL 3.0.3 (git)	TLSv1.3 TLS_AES_256_GCM_SHA384, 253 bit ECDH (X25519)
Apple Mail (16.0)	TLSv1.2 ECDHE-RSA-AES256-GCM-SHA384, 256 bit ECDH (P-256)
Thunderbird (91.9)	TLSv1.3 TLS_AES_128_GCM_SHA256, 253 bit ECDH (X25519)

Done 2024-10-29 16:23:50 [84s] -->> 127.0.1.1:443 (vehicletrackereee.ddns.net) <<--

I.3 RESULTADOS OBTIDOS COM A EXECUÇÃO DO TESTSSH.SH APÓS CORREÇÕES

No engine or GOST support via engine with your /usr/bin/openssl

###
    testssl.sh version 3.0.9 from https://testssl.sh/

  This program is free software. Distribution and modification under
  GPLv2 permitted. USAGE w/o ANY WARRANTY. USE IT AT YOUR OWN RISK!

    Please file bugs @ https://testssl.sh/bugs/

###

 Using *bash 5.2.15. OpenSSL 3.0.15 3 Sep 2024 (Library: OpenSSL 3.0.15 3 Sep 2024)* [~76 ciphers]
    on vehicletrackereee:/usr/bin/openssl
    (built: *Oct 27 14:16:28 2024*, platform: *debian-amd64*)

 Start 2024-11-25 14:36:03   -->> 127.0.1.1:443 (vehicletrackereee.ddns.net) <<--

 A record via: /etc/hosts
 rDNS (127.0.1.1):   --
 Service detected: HTTP

 Testing protocols via sockets except NPN+ALPN

 SSLv2   **not offered (OK)**
 SSLv3   **not offered (OK)**
 TLS 1   not offered
 TLS 1.1   not offered
 TLS 1.2   not offered
 TLS 1.3   **offered (OK)**: final
 NPN/SPDY   not offered
 ALPN/HTTP2 http/1.1 (offered)

 Testing cipher categories

 NULL ciphers (no encryption)   **not offered (OK)**
 Anonymous NULL Ciphers (no authentication) **not offered (OK)**
 Export ciphers (w/o ADH+NULL)   **not offered (OK)**
 LOW: 64 Bit + DES, RC[2,4] (w/o export)   not offered (OK)
 Triple DES Ciphers / IDEA   not offered
 Obsolete CBC ciphers (AES, ARIA etc.)   not offered
 Strong encryption (AEAD ciphers)   **offered (OK)**

 Testing robust (perfect) forward secrecy, (P)FS -- omitting Null Authentication/Encryption, 3DES, RC4

 PFS is offered (OK)   TLS_AES_256_GCM_SHA384 TLS_CHACHA20_POLY1305_SHA256 TLS_AES_128_GCM_SHA256
 Elliptic curves offered: prime256v1 secp384r1 secp521r1 X25519 X448
 Finite field group: ffdhe2048 ffdhe3072 ffdhe4096 ffdhe6144 ffdhe8192

 Testing server preferences

 Has server cipher order?   yes (TLS 1.3 only)
 Negotiated protocol   **TLSv1.3**
 Negotiated cipher   **TLS_AES_256_GCM_SHA384**, 253 bit ECDH (X25519)
 Cipher order
    TLSv1.3:   TLS_AES_256_GCM_SHA384 TLS_CHACHA20_POLY1305_SHA256 TLS_AES_128_GCM_SHA256

 Testing server defaults (Server Hello)

 TLS extensions (standard)   "supported versions/#43" "key share/#51" "server name/#0"
 Session Ticket RFC 5077 hint no -- no lifetime advertised
 SSL Session ID support   no
 Session Resumption   Tickets no, ID: no
 TLS clock skew   Random values, no fingerprinting possible
 Signature Algorithm   SHA256 with RSA
 Server key size   RSA 4096 bits
 Server key usage   Digital Signature, Non Repudiation, Key Encipherment, Data Encipherment, Key Agreement, Certificate Sign
 Server extended key usage   --
 Serial   5EAA2A209C37FC08A1A2BE987DC5D443C27AA551 (OK: length 20)
 Fingerprints   SHA1 2E26A0FAC97F0A79CD0FEE6CBBE74DF59F445CB6
                                  SHA256 64156BDD2C1167898459B87DBB11E384F9325C90002D3951F0CFFD95FAB9E2FC
 Common Name (CN)   *vehicletrackereee.ddns.net*
 subjectAltName (SAN)   *vehicletrackereee.ddns.net*
 Issuer   *vehicletrackereee.ddns.net* (*LocalSecurity* from *BR*)
 Trust (hostname)   Ok via SAN (same w/o SNI)
 Chain of trust   **NOT ok** (self signed)
 EV cert (experimental)   no
 ETS/"eTLS", visibility info not present
 Certificate Validity (UTC)   334 >= 60 days (2024-10-25 19:54 --> 2025-10-25 19:54)
 # of certificates provided   1
 Certificate Revocation List --
 OCSP URI   --
                                  NOT ok -- neither CRL nor OCSP URI provided
 OCSP stapling   not offered
 OCSP must staple extension   --
 DNS CAA RR (experimental)   **not offered**
 Certificate Transparency   --

 Testing HTTP header response @ "/"

 HTTP Status Code   200 OK
 HTTP clock skew   0 sec from localtime
 Strict Transport Security   365 days=31536000 s, just this domain
 Public Key Pinning   --
 Server banner   Apache/2.4.62 (**Debian**)
 Application banner   --
 Cookie(s)   (none issued at "/")
 Security headers   --
 Reverse Proxy banner   --

 Testing vulnerabilities

 Heartbleed (CVE-2014-0160)   **not vulnerable (OK)**, no heartbeat extension
 CCS (CVE-2014-0224)   **not vulnerable (OK)**

Ticketbleed (CVE-2016-9244), experiment. **not vulnerable (OK)**, no session ticket extension
ROBOT **Server does not support any cipher suites that use RSA key transport**
Secure Renegotiation (RFC 5746) **not vulnerable (OK)**
Secure Client-Initiated Renegotiation **not vulnerable (OK)**
CRIME, TLS (CVE-2012-4929) **not vulnerable (OK)**
BREACH (CVE-2013-3587) no HTTP compression (OK) - only supplied "/" tested
POODLE, SSL (CVE-2014-3566) **not vulnerable (OK)**, no SSLv3 support
TLS_FALLBACK_SCSV (RFC 7507) No fallback possible (OK), TLS 1.3 is the only protocol
SWEET32 (CVE-2016-2183, CVE-2016-6329) **not vulnerable (OK)**
FREAK (CVE-2015-0204) **not vulnerable (OK)**
DROWN (CVE-2016-0800, CVE-2016-0703) **not vulnerable on this host and port (OK)**
make sure you don't use this certificate elsewhere with SSLv2 enabled services
https://search.censys.io/search?resource=hosts&virtual_hosts=INCLUDE&q=64156BDD2C1167898459B87DBB11E384F9325C90002D3951F0CFFD95FAB9E2FC
LOGJAM (CVE-2015-4000), experimental not vulnerable (OK): no DH EXPORT ciphers, no DH key detected with <= TLS 1.2
BEAST (CVE-2011-3389) not vulnerable (OK), no SSL3 or TLS1
LUCKY13 (CVE-2013-0169), experimental **not vulnerable (OK)**
RC4 (CVE-2013-2566, CVE-2015-2808) **not vulnerable (OK)**

Testing 370 ciphers via OpenSSL plus sockets against the server, ordered by encryption strength

Hexcode	Cipher Suite Name (OpenSSL)	KeyExch.	Encryption	Bits	Cipher Suite Name (IANA/RFC)
x1302	TLS_AES_256_GCM_SHA384	ECDH 253	AESGCM	256	TLS_AES_256_GCM_SHA384
x1303	TLS_CHACHA20_POLY1305_SHA256	ECDH 253	ChaCha20	256	TLS_CHACHA20_POLY1305_SHA256
x1301	TLS_AES_128_GCM_SHA256	ECDH 253	AESGCM	128	TLS_AES_128_GCM_SHA256

Running client simulations (HTTP) via sockets

Android 6.0	No connection
Android 7.0 (native)	No connection
Android 8.1 (native)	No connection
Android 9.0 (native)	TLSv1.3 TLS_AES_256_GCM_SHA384, 253 bit ECDH (X25519)
Android 10.0 (native)	TLSv1.3 TLS_AES_256_GCM_SHA384, 253 bit ECDH (X25519)
Android 11 (native)	TLSv1.3 TLS_AES_256_GCM_SHA384, 253 bit ECDH (X25519)
Android 12 (native)	TLSv1.3 TLS_AES_256_GCM_SHA384, 253 bit ECDH (X25519)
Chrome 79 (Win 10)	TLSv1.3 TLS_AES_256_GCM_SHA384, 253 bit ECDH (X25519)
Chrome 101 (Win 10)	TLSv1.3 TLS_AES_256_GCM_SHA384, 253 bit ECDH (X25519)
Firefox 66 (Win 8.1/10)	TLSv1.3 TLS_AES_256_GCM_SHA384, 253 bit ECDH (X25519)
Firefox 100 (Win 10)	TLSv1.3 TLS_AES_256_GCM_SHA384, 253 bit ECDH (X25519)
IE 6 XP	No connection
IE 8 Win 7	No connection
IE 8 XP	No connection
IE 11 Win 7	No connection
IE 11 Win 8.1	No connection
IE 11 Win Phone 8.1	No connection
IE 11 Win 10	No connection
Edge 15 Win 10	No connection
Edge 101 Win 10 21H2	TLSv1.3 TLS_AES_256_GCM_SHA384, 253 bit ECDH (X25519)
Safari 12.1 (iOS 12.2)	TLSv1.3 TLS_AES_256_GCM_SHA384, 253 bit ECDH (X25519)
Safari 13.0 (macOS 10.14.6)	TLSv1.3 TLS_AES_256_GCM_SHA384, 253 bit ECDH (X25519)
Safari 15.4 (macOS 12.3.1)	TLSv1.3 TLS_AES_256_GCM_SHA384, 253 bit ECDH (X25519)
Java 7u25	No connection
Java 8u161	No connection
Java 11.0.2 (OpenJDK)	TLSv1.3 TLS_AES_256_GCM_SHA384, 256 bit ECDH (P-256)
Java 17.0.3 (OpenJDK)	TLSv1.3 TLS_AES_256_GCM_SHA384, 253 bit ECDH (X25519)
go 1.17.8	TLSv1.3 TLS_AES_256_GCM_SHA384, 253 bit ECDH (X25519)
LibreSSL 2.8.3 (Apple)	No connection
OpenSSL 1.0.2e	No connection
OpenSSL 1.1.0l (Debian)	No connection
OpenSSL 1.1.1d (Debian)	TLSv1.3 TLS_AES_256_GCM_SHA384, 253 bit ECDH (X25519)
OpenSSL 3.0.3 (git)	TLSv1.3 TLS_AES_256_GCM_SHA384, 253 bit ECDH (X25519)
Apple Mail (16.0)	No connection
Thunderbird (91.9)	TLSv1.3 TLS_AES_256_GCM_SHA384, 253 bit ECDH (X25519)

Done 2024-11-25 14:36:45 [46s] -->> 127.0.1.1:443 (vehicletrackereee.ddns.net) <<--

I.4 CONFIGURAÇÕES FINAIS DO *VIRTUALHOST* NO APACHE

```
<VirtualHost *:443>
    DocumentRoot "/var/www/html/vehicleTrackerEEE/"
    ServerName vehicletrackereee.ddns.net
    DocumentRoot "/var/www/html/vehicleTrackerEEE/public/"
    DirectoryIndex index.html

    # Configurações de criptografia - TLS
    # Utilizando certificado auto assinado
    SSLEngine on
    SSLCertificateFile /etc/ssl/certs/vehicletrackereee.crt
    SSLCertificateKeyFile /etc/ssl/private/vehicletrackereee.private.key

    # Configurações para criptografia forte
    SSLProtocol         all -SSLv3 -TLSv1 -TLSv1.1 -TLSv1.2
    SSLCipherSuite      ECDHE-ECDSA-AES256-GCM-SHA384 ECDHE-RSA-AES256-GCM-SHA384
    ↪  ECDHE-ECDSA-CHACHA20-POLY1305 ECDHE-RSA-CHACHA20-POLY1305
    ↪  ECDHE-ECDSA-AES128-GCM-SHA256 ECDHE-RSA-AES128-GCM-SHA256
    ↪  ECDHE-ECDSA-AES256-SHA384 ECDHE-RSA-AES256-SHA384 ECDHE-ECDSA-AES128-SHA256
    ↪  ECDHE-RSA-AES128-SHA256
    SSLHonorCipherOrder on
    SSLCompression      off
    SSLSessionTickets   off

    # Desabilita a compressão gzip
    SetEnv no-gzip 1

    # Configurações de cabeçalho HSTS
    Header always set Strict-Transport-Security max-age=31536000

    # Configurações de diretórios privado e público
    <Directory "/var/www/html/vehicleTrackerEEE/private/">
        Require all denied
    </Directory>

    <Directory "/var/www/html/vehicleTrackerEEE/public/">
        Require all granted
    </Directory>

    # Configurações personalizadas de logs de erros
```

```
    ErrorLog ${APACHE_LOG_DIR}/vehicletrackereee.error.log
    CustomLog ${APACHE_LOG_DIR}/vehicletrackereee.access.log common

    # Configurações adicionais
    AddDefaultCharset utf-8

    php_flag log_errors on
    php_flag display_errors off
    php_value error_reporting 32767
    php_value error_log /var/log/apache2/vehicletrackereee.php.error.log
</VirtualHost>
```