XIII Seminário de Extensão e Inovação
XXVIII Seminário de Iniciação Científica e Tecnológica da UTFPR
Ciência e Tecnologia na era da Inteligência Artificial: Desdobramentos no Ensino Pesquisa e Extensão
20 a 23 de novembro de 2023 - *Campus Ponta Grossa, PR*
SEI-SICITE 2023

Integração do protocolo de segurança TLS para a comunicação de veículos aéreos não tripulados

Integration of TLS security protocol for unmanned aerial vehicle communication

Beatriz Cristina de Faria[1], Natássya Barlate Floro da Silva[2]

RESUMO

Este trabalho aborda a implementação do protocolo TLS (*Transport Layer Security*) na comunicação entre o ArduCopter e o MAVProxy, para permitir uma troca de mensagens segura entre o piloto automático e a estação de controle na operação de Veículos Aéreos Não Tripulados (VANTs). Dessa forma, serão garantidas as propriedades de autenticidade, confidencialidade e integridade dos dados transmitidos, além de oferecer proteção contra ataques de *Man-in-the-Middle* com o uso de certificados digitais. Durante o desenvolvimento, foram realizadas alterações nos códigos-fonte de ambos os sistemas para estabelecer uma conexão segura e criptografada por meio da inclusão da biblioteca WolfSSL. Os resultados do projeto demonstram uma melhoria significativa na segurança para a comunicação entre o ArduCopter e o MAVProxy.
PALAVRAS-CHAVE: ArduCopter; MAVProxy; Segurança; TLS; WolfSSL.

ABSTRACT

This work addresses the implementation of the TLS (Transport Layer Security) protocol in the communication between ArduCopter and MAVProxy to allow a secure message exchange among the autopilot and the control station in unmanned aerial vehicle (UAV) operation. In this way, the properties of authenticity, confidentiality, and integrity of transmitted data will be guaranteed, in addition to offering protection against Man-in-the-Middle attacks with digital certificates. During development, changes were made to the source codes of both systems to establish a secure and encrypted connection by including the WolfSSL library. The results demonstrate a significant improvement in security for the communication between ArduCopter and MAVProxy.
KEYWORDS: ArduCopter; MAVProxy; Security; TLS; WolfSSL.

INTRODUÇÃO

Nas aplicações de Veículos Aéreos Não Tripulados (VANTs), a troca segura e confiável de dados entre componentes críticos é fundamental (VALAVANIS; VACHTSEVANOS, 2015). O ArduCopter e o MAVProxy, um piloto automático para multirrotores e um software de estação de controle, são importantes ferramentas de código aberto pertencente ao projeto ArduPilot e cuja comunicação constante é necessária para garantir precisão e controle na operação da aeronave (ARDUPILOT DEV TEAM, 2023). No entanto, em uma era de crescentes ameaças de cibersegurança, proteger essas comunicações torna-se essencial. Este artigo explora a implementação do protocolo TLS, um protocolo de segurança robusto que utiliza códigos de autenticação e criptografia, para aprimorar a

[1] Voluntária. Universidade Tecnológica Federal do Paraná, Cornélio Procópio, Paraná, Brasil. E-mail: beatrizcristina@alunos.utfpr.edu.br. ID Lattes: http://lattes.cnpq.br/1424160241564761.
[2] Docente do Departamento Acadêmico de Computação. Universidade Tecnológica Federal do Paraná, Cornélio Procópio, Paraná, Brasil. E-mail: natassyasilva@utfpr.edu.br. ID Lattes: http://lattes.cnpq.br/3393376801047734.

conexão entre esses sistemas, com o objetivo de garantir a autenticidade, confidencialidade e integridade dos dados. Diante disso, foi incorporada no código-fonte de ambos os sistemas a biblioteca WolfSSL, com a implementação do protocolo TLS para reforçar a segurança da comunicação.

MATERIAIS E MÉTODOS

A ferramenta utilizada durante todo o desenvolvimento foi o ambiente *Windows Subsystem for Linux* 2 (WSL 2) na versão 22.04.2 com o auxílio do software *Xming X Server for Windows* para possibilitar a visualização das interfaces gráficas do ArduCopter. No Quadro 1 estão detalhadas as demais ferramentas e versões instaladas na WSL 2 para a implementação do TLS 1.2.

Quadro 1 – Ferramentas utilizadas na implementação e nos testes do trabalho.

Ferramenta	Descrição	Versão
Python	Linguagem de programação utilizada no desenvolvimento e implementação	3.10.12
Mission Planner	Software para planejamento e controle de missões	1.3.79
Wireshark	Software *sniffer* para a análise de redes	3.6.2
ArduCopter	Software com código aberto do piloto automático de multirrotores	4.5.0
MAVProxy	Software da estação de controle terrestre para VANTs	1.8.66
Pymavlink	Biblioteca em Python para comunicação com o protocolo MAVLink	2.4.40
WolfSSL	Biblioteca com implementações do SSL e TLS para sistemas embarcados	5.6.0

Fonte: Elaborado pelos autores (2023).

Primeiramente, foi feito um estudo detalhado na comunicação do ArduCopter com o MAVProxy, analisando as mensagens trocadas no Wireshark. A partir disso, foi possível descobrir o ponto inicial da comunicação, os arquivos principais para a implementação e os papéis de servidor e cliente entre os mesmos.

No processo de integração, o ArduCopter atuava como servidor, enquanto o MAVProxy desempenhava o papel de cliente. Essa implementação envolveu alterações nos códigos-fonte de ambos os sistemas para estabelecer uma conexão segura e criptografada. A implementação incluiu a biblioteca WolfSSL (libwolfssl) no *script* de compilação *wscript* do diretório raiz do ArduPilot e no diretório ArduCopter. Nessa etapa foram enfrentados desafios significativos ao tentar incluir a biblioteca, pois, infelizmente, havia falta de documentação detalhada e orientações específicas para essa inclusão. Porém, foram superados por meio de diversas tentativas e análise do *script*.

O primeiro código-fonte do ArduCopter a ser modificado foi o *UARTDriver* encontrado na biblioteca *AP_HAL_SITL*. De início, foi necessário incluir o cabeçalho da API nativa WolfSSL e iniciar o objeto SSL, como uma variável global para a conexão SSL, no arquivo de cabeçalho *UARTDriver.h*, sendo um passo importante tanto para o cliente quanto para o servidor. Posteriormente, dentro da função *_tcp_start_connection*, foi iniciada a biblioteca WolfSSL com a função *wolfSSL_Init()*. Depois, criou-se a variável que representa o contexto da biblioteca *ctx*, que contém valores globais para as conexões e informações de certificado. O próximo passo foi criar o contexto da biblioteca usando a função *wolfSSL_CTX_new()*, sendo configurado o uso da versão 1.2 do protocolo TLS (WOLFSSL INC., 2023). Esses passos podem ser observados no Algoritmo 1.

Foi também necessário carregar o Certificado da Autoridade Certificadora (CA), o certificado do servidor e a chave privada do servidor no contexto *ctx*. Essa etapa permitiu que o servidor enviasse

Algoritmo 1 – Inicialização do contexto do wolfSSL e definição da versão do TLS.

```
/* Implementacao UARTDriver.cpp */
wolfSSL_Init();
WOLFSSL_CTX* ctx;

if ( (ctx = wolfSSL_CTX_new(wolfTLSv1_2_server_method())) == NULL) {
    fprintf(stderr, "wolfSSL_CTX_new error.\n");
    exit(EXIT_FAILURE);
}
```

seu próprio certificado para autenticação durante o processo de conexão com o cliente. Primeiro, foram carregados os certificados da CA com a função *wolfSSL_CTX_load_verify_locations()*, que permite que o servidor confie nos certificados emitidos por essa CA. Em seguida, foi carregado o certificado do servidor com *wolfSSL_CTX_use_certificate_file()* para que ele possa ser apresentado ao cliente. Por fim, a chave privada foi carregada com *wolfSSL_CTX_use_PrivateKey_file()*, garantindo que o servidor possua a chave correspondente ao seu certificado.

Em cada conexão TCP, um objeto *ssl* foi criado usando *wolfSSL_new()* e associado ao descritor de *socket* (*_fd*) para a sessão TLS. Se a criação do objeto falha, *NULL* é retornado; em caso de sucesso, o objeto *ssl* é vinculado a *_fd*, como visto no Algoritmo 2.

Algoritmo 2 – Associação do descritor de *socket* à sessão TLS.

```
if((ssl = wolfSSL_new(ctx)) == NULL) {
    fprintf(stderr, "wolfSSL_new error.\n");
    exit(EXIT_FAILURE);
}
wolfSSL_set_fd(ssl, _fd);
}
```

O Algoritmo 3 apresenta a chamada da primitiva *wolfSSL_accept()*, responsável por iniciar e completar o *handshake* TLS entre o servidor e o cliente. O *handshake* é a etapa em que é estabelecida uma conexão segura e autenticada, na qual as partes concordam com os métodos de criptografia e de geração do código de autenticação e compartilham as chaves usadas posteriormente (STALLINGS, 2015). Se não for bem-sucedido, uma mensagem de erro é registrada e a conexão é encerrada para garantir que a segurança da comunicação seja mantida.

Algoritmo 3 – Conexão do servidor que realiza o *handshake* do protocolo TLS.

```
if (wolfSSL_accept(ssl) != SSL_SUCCESS) {
    fprintf(stderr, "SSL handshake error.\n");
    exit(EXIT_FAILURE);
}
```

Por fim, para garantir a segurança dos dados transmitidos, foram substituídas as chamadas de sistema *recv* por *wolfSSL_read()* e de *send* por *wolfSSL_write()*. Essas mudanças permitem que os dados sejam lidos através da sessão TLS, garantindo que a transmissão seja segura. Após finalizadas as modificações, foram realizados testes iniciais dessa integração. Esses testes de integração do

servidor foram simplificados com o tutorial da documentação da WolfSSL, úteis para configurar e testar com êxito a implementação. Ao seguir as orientações do tutorial, foi possível estabelecer uma conexão segura do servidor modificado com o cliente construído com o auxílio do tutorial (WOLFSSL INC., 2023).

Para a integração do cliente, foram feitas modificações no código-fonte *mavutil.py*, encontrado na biblioteca do Pymavlink. O processo envolveu um aprofundamento na linguagem Python para a implementação, e essa tarefa foi facilitada pelo suporte oferecido pela WolfSSL. A integração do protocolo foi realizada diretamente na classe *mavtcp*, mais especificamente dentro da função *do_connect*, que é parte do bloco responsável pelo início da conexão e criação dos *sockets*.

Para a verificação do certificado do servidor foi criada a variável *CA_DATA* para armazenar o caminho do certificado. Também foi necessário criar o contexto da WolfSSL com o objeto *context* e realizar a associação com o descritor de *socket secure_socket* para estabelecer uma conexão segura com o servidor (WOLFSSL INC., 2018), de forma análoga ao que foi feito no *UART-Driver*. No Algoritmo 4 pode ser visto o código resultante, sendo que com a chamada da função *context.wrap_socket(self.port)* é criado o objeto *context*. Já o seu argumento, *self.port*, representa o socket de comunicação inicial. Isso significa que todas as comunicações por meio deste *socket* serão protegidas pelo protocolo TLS. Após a criação do *socket* seguro, a função *connect()* é usada para estabelecer a conexão com o servidor de destino com o endereço salvo no atributo *self.destination_addr*.

Algoritmo 4 – Modificações na classe *mavtcp* para integração do cliente com a WolfSSL com uso do protocolo TLS.

```
1       CA_DATA = '../ca-cert.pem'
2       context = wolfssl.SSLContext(wolfssl.PROTOCOL_TLSv1_2)
3       context.verify_mode = wolfssl.CERT_REQUIRED
4       context.load_verify_locations(CA_DATA)
5       secure_socket = context.wrap_socket(self.port)
6       secure_socket.connect(self.destination_addr)
```

Na mesma classe, porém nas funções *recv* e *write*, são tratadas as operações de receber e enviar dados, respectivamente. Para a função *recv*, a implementação foi direta: utilizando o *socket secure_socket*, os dados são recebidos e armazenados na variável *data* por *secure_socket.recv(n)*, onde *n* representa a quantidade de bytes a serem recebidos. Já na função *write*, os dados a serem enviados são contidos em um *buffer buf*, que é transmitido de forma segura para o servidor usando *secure_socket.send(buf)*.

Após essas modificações, foi possível realizar os experimentos finais com a estação de controle MavProxy sendo o cliente que se conecta com o servidor configurado no piloto automático ArduCopter, com a troca de mensagens segura e protegida pelo protocolo TLS, garantindo a segurança da comunicação, a autenticidade do servidor e a confidencialidade dos dados.

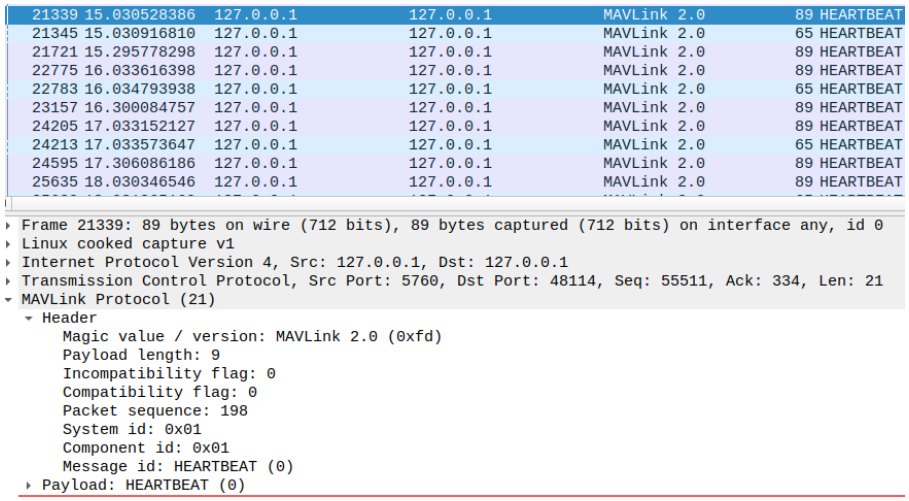
RESULTADOS E DISCUSSÕES

Após a implementação do protocolo TLS no sistema de comunicação entre o ArduCopter e o MAVProxy, foram executados os experimentos que demonstram o correto funcionamento do protocolo.

Os experimentos foram realizados pela observação das mensagens trocada com o uso do protocolo Mavlink por meio do *sniffer* Wireshark, tanto antes das modificações, quanto depois. Também foram observadas as mensagens trocadas durante uma simulação da movimentação da aeronave, sendo possível validar a integridade das mensagens.

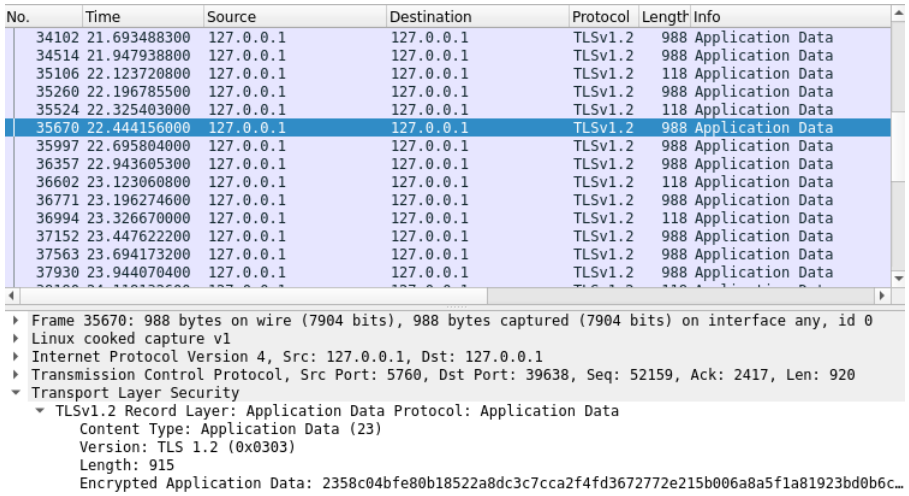
Uma das mudanças mais evidentes foi a criptografia das mensagens transmitidas pelo ArduCopter. Antes da implementação do protocolo de segurança, as mensagens trocadas com o protocolo MAVLink entre os sistemas eram facilmente visíveis no Wireshark, o que representava uma vulnerabilidade significativa à segurança da comunicação, como pode ser observado na Figura 1. No entanto, com a inclusão do protocolo TLS, todas as mensagens do ArduCopter e do MavProxy são criptografadas, tornando-as ilegíveis para observadores externos não autorizados, como pode ser visto na Figura 2. Isso representa uma melhoria substancial na privacidade e na segurança das informações transmitidas.

Figura 1 – Mensagem do ArduCopter ao MavProxy capturada no Wireshark antes das modificações.



Fonte: Elaborado pelos autores (2023).

Figura 2 – Mensagem do ArduCopter ao MavProxy capturada no Wireshark após a integração do protocolo TLS.



Fonte: Elaborado pelos autores (2023).

CONCLUSÃO

Em resumo, esta pesquisa contribui para a segurança da comunicação entre o ArduCopter e o MAVProxy, demonstrando ser possível incorporar requisitos de segurança mesmo em projetos já existentes e funcionais. A implementação do protocolo TLS fortaleceu a confidencialidade, autenticidade e integridade dos dados transmitidos, abordando preocupações de cibersegurança. As alterações nos códigos-fonte, a inclusão da biblioteca WolfSSL no servidor e cliente, os testes e a integração cuidadosa de cada componente permitiram o sucesso na inclusão do protocolo TLS para o sistema de operação de VANTs. Futuramente, espera-se realizar os testes também no hardware do piloto automático Pixhawk para permitir avaliar o desempenho do protocolo em sistemas embarcados.

AGRADECIMENTOS

Agradeço à Universidade Tecnológica Federal do Paraná pelo apoio concedido para a realização desta pesquisa.

DISPONIBILIDADE DE CÓDIGO

Todo o código desenvolvido neste projeto está disponível no repositório do GitHub em https://github.com/BeaComp/implementation-TLS.

CONFLITO DE INTERESSE

Não há conflito de interesse.

REFERÊNCIAS

ARDUPILOT DEV TEAM. **ArduPilot Copter**. Tokyo, 2023. Disponível em: https://ardupilot.org/copter/index.html. Acesso em: 20 ago. 2023.

STALLINGS, William. **Criptografia e Segurança de redes: princípios e práticas**. 6. ed. São Paulo: Pearson Education do Brasil, 2015. ISBN 978-85-430-1450-0.

VALAVANIS, Kimon P.; VACHTSEVANOS, George J. **Handbook of Unmanned Aerial Vehicles**. Dordrecht: Springer Publishing Company, 2015. ISBN 978-90-481-9706-4.

WOLFSSL INC. **SSL Tutorial**. Edmonds, 2023. Disponível em: https://www.wolfssl.com/documentation/manuals/wolfssl/chapter11.html. Acesso em: 15 jun. 2023.

WOLFSSL INC. **SSL/TLS Client Example Python**. Edmonds, 2018. Disponível em: https://wolfssl.github.io/wolfssl-py/index.html#ssl-tls-client-example. Acesso em: 20 ago. 2023.