SPDM-WiD: Uma Ferramenta para Inspeção de Pacotes do Security Protocol Data Model (SPDM)

Thiago D. Ferreira[1], Ágatha de Freitas[2], Renan C. A. Alves[3], Bruno C. Albertini[2], Marcos A. Simplicio Jr.[2], Daniel M. Batista[1]

[1]Depto. Ciência da Computação, Universidade de São Paulo

[2]Depto. Engenharia de Computação e Sistemas Digitais, Universidade de São Paulo

[3]Escola de Artes, Ciências e Humanidades, Universidade de São Paulo

thiago.duvanel@usp.br, batista@ime.usp.br

{agatha.freitas, renanalves, balbertini, msimplicio}@usp.br

Abstract. *Network protocol implementations need to be tested in conjunction with good debugging tools to ensure that packets exchange follows specifications. Although packet sniffers like Wireshark can capture any packet transiting a network interface, the absence of protocol-specific dissectors makes debugging tedious and error-prone. This paper presents the SPDM-WiD (SPDM Wireshark Dissector) tool, a dissector for packet inspection of the Security Protocol Data Model (SPDM), an open communication standard for hardware and firmware authentication. Experiments carried out attest to the effectiveness of the tool and its usefulness in test scenarios for an SPDM implementation.*

Resumo. *Implementações de protocolos de rede precisam ser testadas em conjunto com boas ferramentas de depuração para garantir que a troca de pacotes segue as especificações. Embora sniffers de pacotes como o Wireshark consigam capturar qualquer pacote transitando por uma interface de rede, a ausência de dissecadores específicos para um protocolo torna a depuração tediosa e mais propensa a erros. Este artigo apresenta a ferramenta SPDM-WiD (SPDM Wireshark Dissector), um dissecador para inspeção de pacotes do Security Protocol Data Model (SPDM), um padrão aberto de comunicação para autenticação de hardware e firmware. Experimentos comprovam a eficácia da ferramenta e a sua utilidade em cenários de testes de uma implementação do SPDM.*

1. Introdução

Ao longo dos anos, testemunhou-se um crescimento significativo dos sistemas envolvidos em redes de computadores que, para preencher lacunas, tornaram-se cada vez mais robustos e complexos, justificando a criação de novos protocolos. Em termos de segurança, há poucos anos o TLS influenciou a especificação do SPDM (*Security Protocol and Data Model*), um padrão aberto proposto pela DMTF (*Distributed Management Task Force*) para autenticação de hardware e firmware [DMTF 2023]. Por ser um protocolo de camada de aplicação, o SPDM pode ser utilizado independente das camadas de mais baixo nível. Por exemplo, ele pode ser usado para troca de mensagens pelo barramento interno de um computador para aumentar a segurança nas leituras e escritas de um disco

rígido [Alves et al. 2022]. Ele também pode ser usado para troca de mensagens em cima do TCP, como um protocolo convencional da Internet. A popularização do SPDM depende de usuários da comunidade aderirem à sua especificação, utilizando o mesmo em cenários onde autenticação de hardware e firmware seja necessária. Tal popularização tende a motivar diferentes implementações do protocolo.

Implementações de protocolos de rede precisam ser testadas tanto de forma isolada, com os componentes de software de cada nó da comunicação sendo avaliados, quanto de forma integrada, com a troca de mensagens entre os vários nós sendo comparada com as especificações. Por exemplo, no caso do protocolo TLS, a implementação de um servidor pode ser testada com um cliente especial instrumentado para enviar requisições corretas e incorretas e para capturar as respostas a essas requisições. A interpretação das respostas no nível da camada de aplicação facilita a busca por vulnerabilidades na implementação do servidor antes do mesmo ser colocado em produção [Velozo et al. 2023]. Além das respostas no nível da aplicação, a inspeção de todas as informações contidas nos pacotes facilita a verificação da implementação do protocolo [Araujo Rodriguez and Batista 2021].

A inspeção de pacotes de rede exige que os mesmos sejam interceptados no sistema operacional, preferencialmente utilizando bibliotecas capazes de realizar a ação de *sniffing*, que consiste em capturar os pacotes de rede com base em algum filtro. Dentre as várias bibliotecas disponíveis para computadores de propósito geral, a biblioteca `libpcap` merece destaque [The Tcpdump Group 2024]. Ela é usada pelos principais *sniffers* existentes, como o `Wireshark` [Wireshark 2024b], um *sniffer* considerado por muitos como a principal ferramenta disponível para captura de pacotes. A sua execução pode ser feita tanto por meio de uma GUI quanto por meio do utilitário de linha de comando `tshark`. Independente da forma como seja invocado, o `Wireshark` depende de componentes de software específicos para exibir as informações de diferentes protocolos de uma forma que facilite o trabalho do(a) desenvolvedor(a) de software, destacando as partes mais importantes de um pacote e dando sentido a um emaranhado de *bytes*. Esses componentes são chamados de dissecadores. A instalação padrão do `Wireshark` tem vários dissecadores embutidos e novos podem ser instalados [Wireshark 2024a]. Não temos conhecimento de dissecadores para o SPDM, o que faz com que a captura de pacotes desse protocolo no `Wireshark` seja exibida sem detalhes, com a carga útil sendo classificada apenas como `Data` e o protocolo de transporte como TCP (vide Figura 1).

Este artigo apresenta o SPDM-WiD (SPDM Wireshark Dissector), o primeiro dissecador para inspeção de pacotes do SPDM no `Wireshark`. Ele é um software de código-aberto sob licença GPL-3.0, foi desenvolvido na linguagem Lua e é compatível integralmente com a versão 1.0.0 do SPDM. O objetivo desse dissecador é depurar uma ferramenta de testes do SPDM que encontra-se em desenvolvimento. Experimentos realizados em um cenário realista de utilização do SPDM atestam a eficácia do dissecador.

Este artigo está organizado como segue. A Seção 2 resume trabalhos que apresentam ferramentas para inspeção de pacotes e trabalhos relacionados. A Seção 3 descreve o método usado no desenvolvimento do SPDM-WiD, sua arquitetura e suas principais funcionalidades. A Seção 4 avalia a eficácia do SPDM-WiD por meio de experimentos de interceptação de pacotes. A Seção 5 lista as URLs do código-fonte, da documentação e de um vídeo-demonstração do SPDM-WiD. A Seção 6 conclui a discussão.

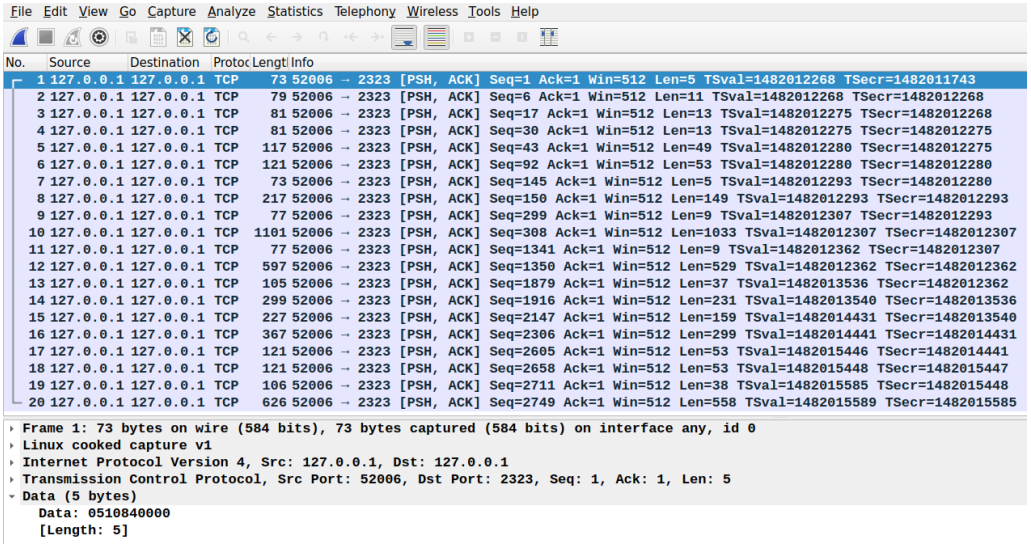

Figura 1. Captura de tela do `Wireshark` sem o SPDM-WiD.

2. Trabalhos Relacionados

Diversos dissecadores para o Wireshark existem. Por exemplo, em [Lucero et al. 2021], é apresentado o primeiro dissecador para o protocolo de roteamento RIFT (*Routing in Fat Trees*), um protocolo de roteamento proposto para *data centers* com tráfego intenso. O dissecador é escrito em Lua e em C e teve sua eficácia testada com duas implementações do RIFT, uma de código aberto e uma proprietária. Esse trabalho mostra a flexibilidade de testar implementações de protocolos de rede usando *sniffers*. Enquanto ferramentas utilizadas em desenvolvimento orientado a testes necessitariam do código-fonte da implementação, a captura dos pacotes por *sniffers* permite analisar a aderência da implementação à especificação sem necessidade de acesso ao código-fonte. Os autores disponibilizaram a ferramenta publicamente no gitlab sob licença Apache 2.0.

Em [Ryu et al. 2022] é apresentado um dissecador para algumas mensagens do protocolo de descoberta de nós do Ethereum, uma plataforma de *blockchain*. A diferença desse dissecador para outros que interpretam as mesmas mensagens é o fato dele ser capaz de interpretar pacotes criptografados do protocolo. O dissecador é escrito em Lua e em C++ e teve sua eficácia testada com um cliente real do Ethereum, embora os autores não descrevam qual dos vários possíveis clientes foi usado. Diversas estatísticas sobre os pacotes puderam ser extraídas com a utilização do dissecador. Os autores não disponibilizaram a ferramenta publicamente.

Em [Hu and Zhou 2022] é apresentado o primeiro dissecador para o protocolo DL/T 860, um protocolo de comunicação usado em subestações de uma *smart grid*. O dissecador é escrito em C. Os autores apresentam capturas de tela do `Wireshark` atestando a eficácia do dissecador mas não há informações detalhadas sobre o ambiente de experimentação e nem sobre acesso ao código-fonte da ferramenta.

O SPDM define mecanismos e formatos para autenticação de hardware e firmware. Em [Alves et al. 2022], ele foi integrado em um ambiente emulado baseado no software `QEMU` para aumentar a segurança nas leituras e escritas de um disco rígido. O SPDM considera a existência de dois nós na comunicação. Um chamado de *Requester*, que seria

o equivalente a um cliente, e um chamado de *Responder*, que seria o equivalente a um servidor. No cenário do disco rígido, o *Requester* foi o kernel do sistema operacional, mais precisamente o driver do disco, e o *Responder* foi o próprio disco.

O SPDM-WiD se inspira nos dissecadores anteriores ao ser o primeiro dissecador para seu respectivo protocolo, como os apresentados em [Lucero et al. 2021] e [Hu and Zhou 2022], e por ser uma ferramenta de código aberto, assim como em [Lucero et al. 2021]. Além disso, sua eficácia foi atestada capturando pacotes em uma implementação real do protocolo, não desenvolvida pelos mesmos desenvolvedores do dissecador, tal qual [Lucero et al. 2021] e [Ryu et al. 2022].

3. SPDM-WiD

O SPDM-WiD é um dissecador de pacotes SPDM para o *sniffer* `Wireshark` programado em Lua. O dissecador é baseado nas versões 1.1.0 [DMTF 2020] e 1.0.0 [DMTF 2019c] do SPDM, com suporte completo à 1.0.0. A implementação do SPDM utilizada para gerar os pacotes a serem interpretados pelo dissecador foi a presente na biblioteca `libSPDM` [DMTF 2024], que também está presente na implementação emulada do acesso seguro ao disco rígido descrita em [Alves et al. 2022].

3.1. Metodologia

A metodologia seguida no projeto dessa ferramenta foi de cascata. Por ser um software de poucos arquivos, com um objetivo claro e bem definido, além do fato de não ter sido produzido por uma equipe distribuída de desenvolvedores, o processo de produção pôde ser linear, seguindo o estudo das especificações do protocolo. A Figura 2 apresenta parte da troca inicial de mensagens entre os nós de comunicação do SPDM. O par de mensagens `GET_VERSION` e `VERSION` consulta e informa as versões suportadas do SPDM. O par de mensagens `GET_CAPABILITIES` e `CAPABILITIES` consulta e informa as funcionalidades suportadas. O par de mensagens `NEGOTIATE_ALGORITHMS` e `ALGORITHMS` negocia os algoritmos criptográficos a serem usados na comunicação segura. A título de comparação, esses três pares de mensagens são equivalentes às mensagens `ClientHello` e `ServerHello` do *handshake* do protocolo TLS.

O SPDM foi projetado para operar sobre um protocolo da camada de transporte. Neste caso, na implementação usada da `libSPDM` [DMTF 2024], é usado o MCTP (*Management Component Transport Protocol*) [DMTF 2019a]. Assim como o TCP está para o TLS, o MCTP está para o SPDM, auxiliando na troca de mensagens pelo *hardware*. Apesar do `Wireshark` oferecer suporte para o MCTP, na troca de pacotes do SPDM, emulado pelo QEMU, seus bytes estão sobre o TCP, o que impossibilita o *sniffer* de reconhecê-lo. Por isso, também foi adicionada a dissecação desses bytes no SPDM-WiD, caso existam (ao exibir os pacotes nós removemos esses bytes para focar apenas no SPDM). Mais detalhes sobre essas e as demais mensagens do protocolo podem ser obtidas em [DMTF 2020], [DMTF 2019c] e em [DMTF 2019b].

Também foi necessário estudar a arquitetura do `Wireshark`. Foram utilizadas ferramentas de depuração de scripts em Lua, embutidas no próprio `Wireshark`, permitindo observar os detalhes de cada um dos tipos de pacotes a partir do terminal em que o *sniffer* era executado.

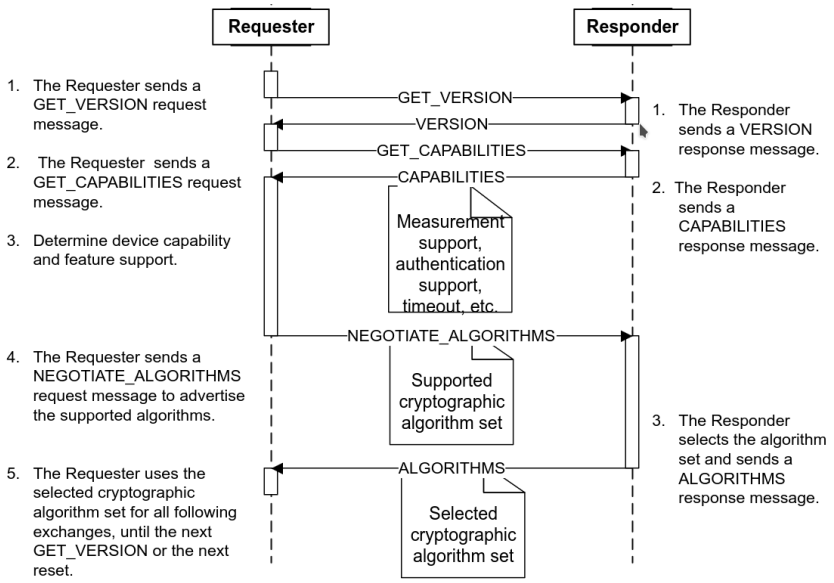

Figura 2. Mensagens iniciais do SPDM (Extraído de [DMTF 2020]).

Dois ambientes de experimentação precisaram ser criados para realização de testes do dissecador paralelamente à sua implementação, com a captura de pacotes ocorrendo na porta 2323 TCP do *Responder* pelo `Wireshark`. No primeiro ambiente, o *Requester* e o *Responder* originais presentes na `libSPDM` foram executados no localhost. No segundo ambiente, o cenário do disco rígido em [Alves et al. 2022] foi considerado, utilizando os códigos implementados no disco rígido emulado e no driver do disco rígido.

O segundo ambiente exigiu uma forma de exportar os pacotes trocados dentro do `QEMU` para que eles fossem capturados em uma instância do `Wireshark` na máquina hospedeira onde o `QEMU` estava sendo executado. Para isso, foi criado um socket dentro do `QEMU`, especificamente na área em que era emulado o disco rígido (*Responder*) autenticado, que exportava as mensagens do SPDM trocadas com o driver (*Requester*) também emulado, responsável pela autenticação. Esse socket criado foi lido na máquina hospedeira onde o `QEMU` era executado. Dessa forma o dissecador era capaz de interpretar as mensagens abstraindo o fato de que elas estavam sendo geradas internamente entre os componentes do computador emulado. A Figura 3 ilustra esse ambiente.

3.2. Arquitetura e Principais Funcionalidades

O `Wireshark` suporta duas linguagens principais para produzir dissecadores: C e Lua. Apesar da vantagem em velocidade do C, Lua foi escolhida pela simplicidade e grande portabilidade que possui para os objetivos do projeto. Além disso, o dissecador em C necessitaria de uma recompilação do *sniffer* inteiro cada vez que fosse alterado e/ou instalado, já que ele estaria embutido internamente no programa, e não como um *add-on*. Isso desestimularia seu uso e teste pela comunidade.

O SPDM-WiD suporta todas as mensagens da versão 1.0.0 do SPDM, o que inclui diversos padrões de segurança da informação, como assinaturas, certificados e *hashes*. Também são suportadas algumas mensagens da versão 1.1.0, que apesar de serem oficializadas nessa versão, já estavam incluídas na implementação do SPDM presente no

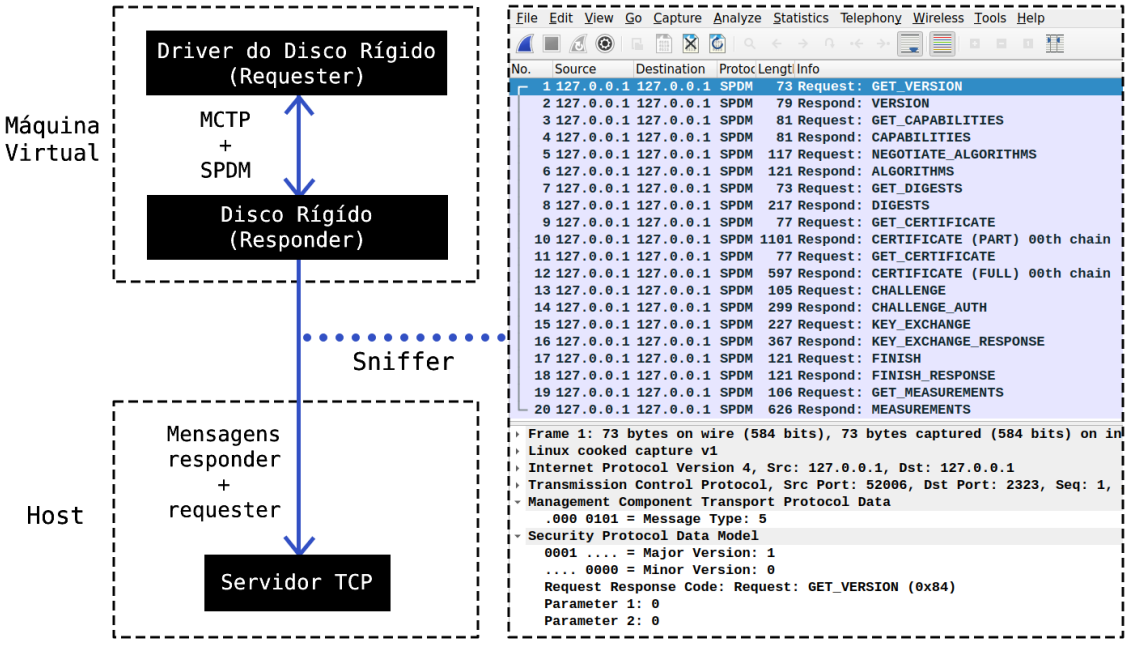

Figura 3. Segundo ambiente de experimentação, com o SPDM-WiD.

cenário emulado com o disco rígido no QEMU. As mensagens conseguem ser exibidas pelo dissecador tanto utilizando algum arquivo de *trace* quanto com a captura "ao vivo".

4. Resultados

A Figura 3 apresenta a interface do Wireshark com a utilização do SPDM-WiD durante a captura de pacotes no cenário emulado do disco rígido. Diferente do observado na Figura 1, as mensagens do protocolo são corretamente identificadas, como esperado.

Ainda no mesmo cenário emulado do disco rígido, a implementação do *Requester* foi modificada propositalmente para enviar uma mensagem CHALLENGE em desacordo com o protocolo (O par de mensagens CHALLENGE e CHALLENGE_AUTH autentica o *Responder* por meio de um protocolo *challenge-response*). A modificação em questão foi o envio de uma segunda mensagem CHALLENGE. Esse comportamento deveria ativar o envio de uma mensagem de erro do *Responder* e o Wireshark foi capaz de detectar essa mensagem conforme visto na Figura 4, mostrando que a implementação do *Responder* está agindo como esperado para esse caso.

5. Código, Documentação, Vídeo e Demonstração

O SPDM-WiD envolve dois repositórios no GitHub: um em que o dissecador propriamente dito está disponível: https://github.com/th-duvanel/spdm-wid?tab=readme-ov-file; e outro em que está disponibilizada a ferramenta que exporta os pacotes trocados no QEMU: https://github.com/th-duvanel/riscv-spdm. O dissecador também foi publicado e está disponível na Wiki oficial do Wireshark: https://wiki.wireshark.org/Contrib#display-filter-macros.

Ambos os repositórios possuem os arquivos e a documentação necessária para compilação, execução e modificação. Foram disponibilizados scripts automatizados que

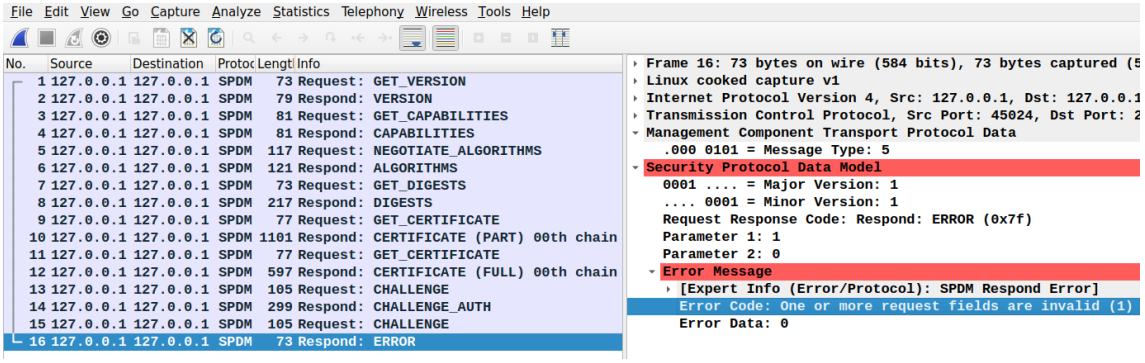

Figura 4. Exibição de um pacote com mensagem de erro.

facilitam o processo, além de observações importantes, como as dependências. Um vídeo, que detalha toda a compilação, execução e demonstração da ferramenta no cenário emulado com o disco rígido, está disponível em: `https://youtu.be/Wl6sOXs2tmg`. É importante esclarecer que a compilação apresentada no vídeo é necessária por conta do ambiente emulado no `QEMU`, e não por conta do SPDM-WiD.

6. Conclusão e Trabalhos Futuros

O estudo de protocolos é de extrema importância para compreender integralmente como um *software* em rede funciona, para buscar vulnerabilidades e para verificar se as implementações seguem as especificações. Uma forma de estudar os protocolos é com o uso de *sniffers*, que eventualmente precisam ser estendidos para suporte a novos protocolos. No caso do `Wireshark`, essa extensão é feita por meio de dissecadores.

Este artigo apresenta o software SPDM-WiD, o primeiro dissecador para inspeção de pacotes do SPDM no `Wireshark`. O SPDM-WiD, desenvolvido na linguagem Lua tem código-aberto sob licença GPL-3.0 e é compatível integralmente com a versão 1.0.0 do SPDM. Experimentos realizados em um cenário realista de utilização do SPDM atestaram a eficácia do dissecador. Como próximos passos, pretende-se desenvolver um *fuzzer* para o SPDM, visando testar implementações anteriores do protocolo em busca de vulnerabilidades, e a extensão do dissecador para versões mais recentes do protocolo.

Agradecimentos. Esta pesquisa é parte do Centro de Pesquisa em Engenharia SMARTNESS (FAPESP 21/00199-8). Também é suportada pelos projetos FAPESP 20/09850-0, 21/06995-0 e 23/16002-4, CNPq 307732/2023-1, e CAPES 001.

Referências

[Alves et al. 2022] Alves, R. C. A., Albertini, B. C., and Simplicio, M. A. (2022). Securing Hard Drives with the Security Protocol and Data Model (SPDM). In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 446–447.

[Araujo Rodriguez and Batista 2021] Araujo Rodriguez, L. G. and Batista, D. M. (2021). Towards Improving Fuzzer Efficiency for the MQTT Protocol. In *Proceedings of the IEEE Symposium on Computers and Communications (ISCC)*, pages 1–7.

[DMTF 2019a] DMTF (2019a). Management Component Transport Protocol 5 (MCTP) Base Specification. https://www.dmtf.org/sites/default/files/standards/documents/DSP0236_1.3.1.pdf. Acesso em 5/4/2024.

[DMTF 2019b] DMTF (2019b). Security Protocol and Data Model (SPDM) over MCTP binding specification. https://www.dmtf.org/sites/default/files/standards/documents/DSP0275_1.0.0.pdf. Acesso em 5/4/2024.

[DMTF 2019c] DMTF (2019c). Security Protocol and Data Model (SPDM) Specification. https://www.dmtf.org/sites/default/files/standards/documents/DSP0274_1.0.0.pdf. Acesso em 4/4/2024.

[DMTF 2020] DMTF (2020). Security Protocol and Data Model (SPDM) Specification. https://www.dmtf.org/sites/default/files/standards/documents/DSP0274_1.1.0.pdf. Acesso em 4/4/2024.

[DMTF 2023] DMTF (2023). Security Protocols and Data Models Working Group. https://www.dmtf.org/standards/spdm. Acesso em 4/4/2024.

[DMTF 2024] DMTF (2024). libspdm is a sample implementation that follows the DMTF SPDM specifications. https://github.com/DMTF/libspdm. Acesso em 4/4/2024.

[Hu and Zhou 2022] Hu, X. and Zhou, Y. (2022). Wireshark Packet Dissector for DL/T 860 Protocol. In *Proceedings of the 4th International Conference on Electrical Engineering and Control Technologies (CEECT)*, pages 30–34.

[Lucero et al. 2021] Lucero, M., Parnizari, A., Alberro, L., Castro, A., and Grampín, E. (2021). Routing in Fat Trees: a protocol analyzer for debugging and experimentation. In *Proc. of the IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 788–792.

[Ryu et al. 2022] Ryu, J., Kim, A., Essaid, M., and Ju, H. (2022). Development of Wireshark Dissector for Ethereum Node Discovery Protocol/v5. In *Proceedings of the 23rd Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 1–4.

[The Tcpdump Group 2024] The Tcpdump Group (2024). Home — TCPDUMP & LIBPCAP. https://www.tcpdump.org/. Acesso em 4/4/2024.

[Velozo et al. 2023] Velozo, F., Ferreira, T., Pacheco, E., Alves, R., Jr., M. S., Albertini, B., and Batista, D. (2023). Fuzzing para o Protocolo TLS: Estado da Arte e Comparação de Fuzzers Existentes. In *Anais Estendidos do XXIII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg)*, pages 303–308. SBC.

[Wireshark 2024a] Wireshark (2024a). Contrib - Wireshark Wiki. https://wiki.wireshark.org/Contrib#protocol-dissectors. Acesso em 4/4/2024.

[Wireshark 2024b] Wireshark (2024b). Wireshark · Go Deep. https://www.wireshark.org/. Acesso em 4/4/2024.