dx.doi.org/10.7437/NT2236-7640/2022.02.003
Notas Técnicas, v. 12, n. 2, p. 25–30, 2022

Comunicações seguras entre dispositivos IoT utilizando o ESP32

Alan Franco Rosa,* David Vaz Teixeira,† and Nilton Alves Júnior‡

Laboratório de Instrumentação, Informação e IoT (Lab3I)

Centro Brasileiro de Pesquisas Físicas (CBPF),

Rua Dr. Xavier Sigaud 150,

CEP: 22290-180, Urca, RJ, Brasil

Submetido em 20 de maio de 2022 *Aceito em 25 de maio de 2022*

Resumo: Neste trabalho foi desenvolvida uma metodologia para aumentar a segurança na comunicação em *Internet of Things* (IoT) que utilizem o ESP32. No projeto desenvolvido foram implementadas criptografias na camada de comunicação, utilizando o protocolo TLSv1.2, além da criptografia do *firmware*, para mitigar ataques físicos com o intuito de inserir um *firmware* modificado.

Palavras chave: Segurança, Instrumentação, IoT, ESP32.

Abstract: In this work, a methodology was developed to increase communication security in *Internet of Things* (IoT) projects that uses ESP32. In this project we have implemented encryption in the layer communication, using the TLSv1.2 protocol, besides firmware cryptography, to mitigate physical attacks that have the objective to inserting a modified firmware.

Keywords: Security, Instrumentation, IoT, ESP32.

I. INTRODUÇÃO

Com a crescente quantidade de equipamentos conectados à Internet, também aumentam os números relativos a ataques que exploram as vulnerabilidades desses dispositivos. Com isso, o padrão IEEE 2413-2019 [1] destacou a importância da segurança, definindo-a como parte da natureza dos dispositivos e aplicações em *Internet of Things* (IoT).

No ambiente IoT, geralmente as questões relacionadas com a segurança são negligenciadas, seja pela limitação de *hardware*, pelas dificuldades técnicas envolvidas ou por se julgar desnecessário. A grande quantidade de tecnologias que podem ser utilizadas também dificultam a determinação de uma infraestrutura padronizada [2], sendo necessário que se avalie cada caso.

Em 2018 foi publicado pelo *Open Web Application Security Project* (OWASP) [3] o *Top Ten* das vulnerabilidades mais comuns em IoT, dentre as quais podem ser destacadas as que envolvem a transferência de dados sem criptografia e dispositivos fisicamente vulneráveis. Com a preocupação de melhorar a segurança em dispositivos IoT houve um trabalho desenvolvido pelo *National Institute of Standards and Technology* (NIST) dos Estados Unidos, que buscou definir quais orientações as empresas devem seguir para coibir ataques a esses tipos de dispositivos [4]. Em 12 de Maio de 2022 o presidente dos EUA assinou a Ordem Executiva 14028, resultante desse projeto, para apoiar a segurança cibernética, proteger a infraestrutura crítica e as redes do governo federal [5].

O presente trabalho implementa formas de mitigar ataques que envolvam a interceptação de dados e a alteração do *firmware* em dispositivos IoT que utilizam o ESP32. É esperado que essas implementações sirvam de orientação para interessados em desenvolver projetos semelhantes. Também é importante observar que as soluções desenvolvidas apenas acrescentam novas camadas de segurança, mas existem ataques mais sofisticados que podem ser feitos em laboratórios, como os ataques de *Fault Injection and eFuse protection* [6], *Fault Injection and Secure Boot* [7], *Zero PMK Installation* [8], *EAP DoS* [9] e *Secure Boot Bypass* [10].

* Endereço Eletrônico: alanfr@cbpf.br
† Endereço Eletrônico: davidvaz@cbpf.br
‡ Endereço Eletrônico: naj@cbpf.br

II. ASPECTOS DE SEGURANÇA

Em um ataque do tipo *Man-in-the-middle* (Figura 1), o invasor busca interceptar a comunicação do dispositivo com o servidor [11], para que as informações enviadas sejam adulteradas ou descobertas, de forma passiva (*Eavesdropping*), possibilitando que o atacante amplie o controle da rede através de técnicas de pivoteamento ou de movimento lateral. No ambiente de IoT essa é uma vulnerabilidade passível de ser explorada, porque em geral os dados enviados pelos dispositivos são considerados como de baixa criticidade, fazendo com que os usuários não se preocupem com a criptografia da informação transmitida, devido ao fato de geralmente os dispositivos utilizados possuírem limitações de *hardware* que dificultam a implementação de metodologias seguras, em razão de essas exigirem mais recursos de processamento e memória.

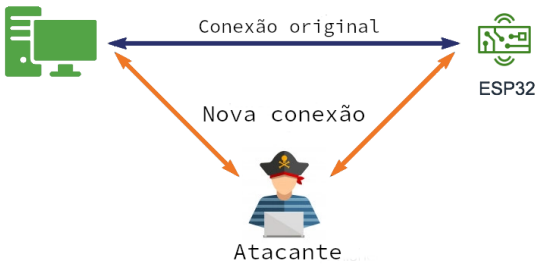

Figura 1. Ataque *Man-in-the-middle*.

Além da segurança associada ao dispositivo, também existe a necessidade de se manter seguro o servidor que faz a comunicação. A escolha do protocolo de comunicação é outra tarefa crítica no sistema, pois ela deve levar em consideração a complexidade para implementação, taxa de transferência necessária e quantidade de dados enviadas por cada dispositivo. A escolha mais comum para os projetos IoT acaba sendo o protocolo *Message Queuing Telemetry Transport* (MQTT) ao invés do tradicional HTTP, pois é um protocolo que possui a natureza de ser utilizado para a coleta de dados entre muitos dispositivos que se conectam a uma infraestrutura computacional para processamento e armazenamento [12].

A característica que torna o MQTT um protocolo amplamente utilizado é o fato de ele necessitar de uma quantidade menor de pacotes para realizar e manter a comunicação. Porém, foi verificado [13] que para sistemas onde a conexão entre os dispositivos e o servidor não é reaproveitada (os dados não são enviados na forma de *stream*), o custo no tráfego acaba sendo muito alto, tornando o protocolo menos eficiente que o HTTP. Sendo assim, devido a natureza das conexões entre os dispositivos de interesse serem de forma intermitente, a utilização de um protocolo com arquitetura *publish/subscribe* não é relevante para a grande maioria dos projetos desenvolvidos pelo laboratório. O protocolo HTTP foi o escolhido por também possuir suporte a comunicações seguras através de criptografia TLS (HTTPS), além de ser compatível com outros sistemas e aplicações.

III. METODOLOGIA

O ambiente utilizado está esquematizado na Figura 2. O servidor foi implementado em uma máquina virtual (VM Server), utilizando a distribuição Rocky Linux em um sistema VMWare. A *Application Programming Interface* (API) que recebe os dados obtidos pelo ESP32 foi elaborada em Python. A segurança é feita através de um firewall com políticas de filtragem *stateful*, baseada em iptables, sistema de detecção de intrusão (IDS) e a utilização do SELinux, para, em caso de invasão, conter o escalonamento de privilégios. A comunicação com a API é feita através de um proxy, que utiliza um *Web Application Firewall* (WAF). Um bom ponto de partida para implementar uma configuração minimamente segura do servidor web é seguir as recomendações do *Mozilla SSL Configuration Generator* [14].

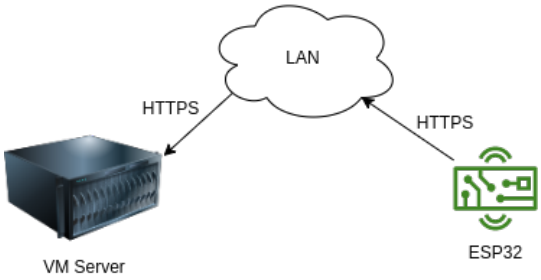

Figura 2. Topologia do ambiente.

A comunicação entre o ESP32 e o servidor é estabelecida através de conexão WiFi, com autenticação baseada na especificação WPA2-PSK. O protocolo de criptografia escolhido foi o TLS v1.2, por ser amplamente suportado pelos sistemas e por muitas bibliotecas disponíveis para o ESP32.

Para captura e análise dos dados de rede foram utilizadas os *softwares* tcpdump [15] e Wireshark [16]. As interfaces de desenvolvimento utilizadas (IDE) foram a do Arduino e o esp-idf.

IV. RESULTADOS

IV.1. Criptografia na comunicação

Inicialmente, para comparação, a comunicação entre os dispositivos foi feita utilizando o protocolo HTTP e todo o tráfego da comunicação foi interceptada com o *tcpdump*. A segurança da comunicação, nesse caso, fica dependente da rede e de suas políticas de acesso.

Na Figura 3 é possível observar o resultado da interceptação, com o fluxo da informação não crip-

tografada, sendo possível identificar quando os dados são enviados (POST) e a rota utilizada (/medidores/).

```
  74 45898 → 8000 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_
  58 8000 → 45898 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS
  60 45898 → 8000 [ACK] Seq=1 Ack=1 Win=29200 Len=0
 418 POST /medidores/ HTTP/1.1  (application/json)
  54 8000 → 45898 [ACK] Seq=1 Ack=365 Win=30016 Len=0
  76 8000 → 45898 [PSH, ACK] Seq=1 Ack=365 Win=30016 Len=22
  60 45898 → 8000 [ACK] Seq=365 Ack=23 Win=29200 Len=0
 500 HTTP/1.1 201 Created  (application/json)
  60 45898 → 8000 [ACK] Seq=365 Ack=469 Win=30016 Len=0
  60 45898 → 8000 [FIN, ACK] Seq=365 Ack=469 Win=30016 Len=0
  54 8000 → 45898 [FIN, ACK] Seq=469 Ack=366 Win=30016 Len=0
  60 45898 → 8000 [ACK] Seq=366 Ack=470 Win=30016 Len=0
  74 45900 → 8000 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_
```

Figura 3. Interceptação da comunicação não criptografada (HTTP) entre o ESP32 e o servidor, utilizando o Wireshark.

Ao analisar o fluxo, fica claro a facilidade de recuperar informações sensíveis, como o formato em que os dados são enviados (JSON), o token de validação e as informações enviadas (Figura 4).

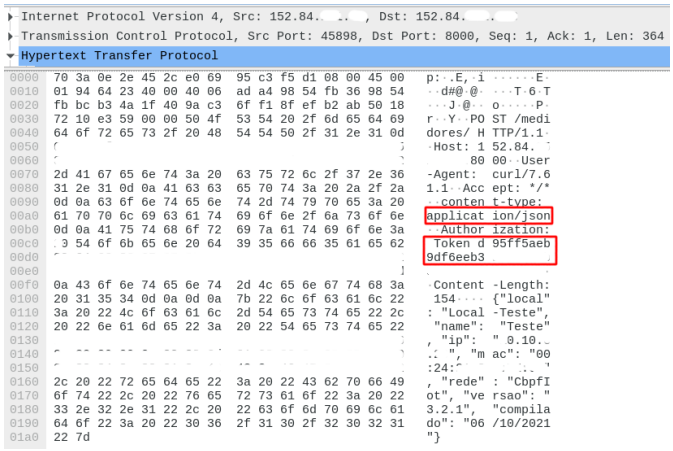

Figura 4. Dados não criptografados enviados pelo ESP32 para a API.

A programação para a comunicação através do protocolo HTTPS foi implementada de duas formas: a primeira foi com a utilização da IDE do ESP32 [17] (esp-idf), utilizando a biblioteca mbedtls [18]; a segunda foi através da biblioteca WiFiClientSecure [19], utilizando a interface do Arduino.

Com a comunicação segura, foi possível observar que o tráfego de dados é feito de forma criptografada, com a utilização do protocolo TLSv1.2. No fluxo de dados que está na Figura 5 fica claro a diferença em relação aos dados não criptografados (Figura 3), sendo possível notar a etapa do *handshake* da comunicação entre o ESP32 e o servidor.

```
  TCP      60 61412 → 4242 [SYN] Seq=0 Win=5744 Len=0 MSS=1436
  TCP      58 4242 → 61412 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS
  TCP      60 61412 → 4242 [ACK] Seq=1 Ack=1 Win=5744 Len=0
  TLSv1.2 357 Client Hello
  TCP      54 4242 → 61412 [ACK] Seq=1 Ack=304 Win=30016 Len=0
  TLSv1.2 1589 Server Hello, Certificate, Server Key Exchange, Server
  TCP      60 61412 → 4242 [ACK] Seq=304 Ack=1536 Win=4209 Len=0
  TCP      60 [TCP Window Update] 61412 → 4242 [ACK] Seq=304 Ack=1536
  TLSv1.2 161 Client Key Exchange
  TLSv1.2  60 Change Cipher Spec
  TCP      54 4242 → 61412 [ACK] Seq=1536 Ack=417 Win=30016 Len=0
  TLSv1.2  99 Encrypted Handshake Message
  TLSv1.2 105 Change Cipher Spec, Encrypted Handshake Message
  TLSv1.2 357 Application Data
  TLSv1.2 577 Application Data
  TCP      54 4242 → 61412 [ACK] Seq=1587 Ack=1288 Win=32160 Len=0
  TLSv1.2 1477 Application Data
  TCP      60 61412 → 4242 [ACK] Seq=1288 Ack=3010 Win=5744 Len=0
  TCP      60 61412 → 4242 [FIN, ACK] Seq=1288 Ack=3010 Win=5744 Len=
  TCP      54 4242 → 61412 [FIN, ACK] Seq=3010 Ack=1289 Win=32160 Len
  TCP      60 61412 → 4242 [ACK] Seq=1289 Ack=3011 Win=5743 Len=0
```

Figura 5. Interceptação da comunicação criptografada (HTTPS) entre o ESP32 e o servidor, utilizando o Wireshark.

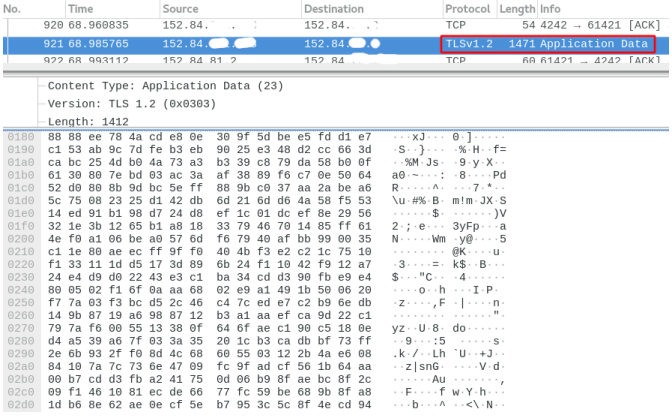

Figura 6. Dados criptografados através do protocolo TLSv1.2.

Analisando o fluxo *Application Data* apresentado na Figura 6 é possível observar a melhoria na segurança, já que não é mais possível identificar informações sensíveis na comunicação, em relação ao que está na Figura 4.

IV.2. Criptografia do *firmware*

A taxonomia de uma vulnerabilidade de *hardware* pode ser divida entre a sua natureza e o seu domínio [20], como pode ser visto na Figura 7.

Caso uma vulnerabilidade de *hardware* possua natureza intencional, significa que ela foi implantada pelo fabricante/desenvolvedor de forma a permitir o acesso ao equipamento. Caso essa natureza não seja intencional, ela pode ser devido a falha do equipamento ou devido a lógica não segura de implementação (*bug*).

Por outro lado, uma vulnerabilidade de domínio lógico é implementada na fase inicial do projeto, enquanto a do

domínio físico é implantada no final do projeto, na parte de design do equipamento.

Dentre as diversas categorias de ameaças, o objetivo com a criptografia do *firmware* é mitigar os ataques de Engenharia reversa e de modificação do *firmware* [21]. Estes tipos de vulnerabilidades estão relacionadas ao domínio físico, onde um atacante que possua acesso ao *hardware* seja capaz de modificar o *firmware* e colocar um outro dispositivo semelhante no lugar.

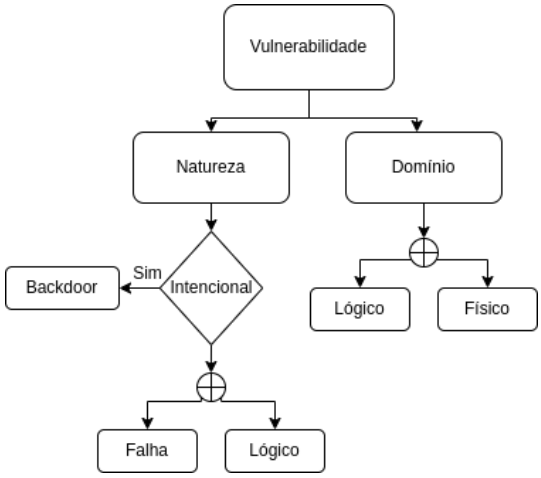

Figura 7. Taxonomia de uma vulnerabilidade de *hardware*.

Como muitas vezes os dispositivos ficam instalados em ambientes isolados e sem vigilância, um invasor de posse do ESP32 pode fazer o *dump* dos dados da memória *flash* e ter acesso a informações sensíveis, além de ser capaz de gerar e implantar um novo *firmware* capaz de se comportar como o original, porém alterando as informações. Por exemplo, um atacante pode mascarar o envio de dados de temperatura, fazendo com que o dispositivo envie informações de que o ambiente está dentro dos limites de temperatura adequados, quando, na verdade, está ocorrendo um incêndio.

Um exemplo de ataque foi o causado pelo *worm* Stuxnet [22]. A dinâmica envolvia infectar o sistema supervisório de uma usina de enriquecimento de urânio do Irã, que era executado em um sistema SCADA, mascarar os valores exibidos e acelerar as centrífugas utilizadas no projeto nuclear iraniano. Com isso foi possível danificar os equipamentos utilizados, prejudicando o programa nuclear daquele país.

Na Figura 8 é possível ver a informação contida no *firmware* a, partir do endereço 0001:0A40, não criptografado, obtida através da extração dos dados da memória. São visíveis os dados da chave utilizada para comunicação com a API, assim como o endereço do servidor e as rotas utilizadas.

Já na Figura 9 está o trecho com os dados para autenticação Wifi. Por questões de segurança algumas partes das informações foram omitidas, mas é possível identificar o SSID no qual o ESP32 deve se conectar (CBPF-IoT), além de trechos da senha de acesso utilizada (-i2-7-).

```
0001:0A40 4C363478 65506D41 3D3D0A2D 2D2D2D2D L64xePmA==.-----
0001:0A50 454E4420 43455254 49464943 41544520 END CERTIFICATE-
0001:0A60 2D2D2D2D 0A006874 74707373 3A2F2F31 -----.https://15
0001:0A70 32[illegible]                       3A343234 2.84.[illegible]:424
0001:0A80 322F6D65 64696461 732F0068 74747073 2/medidas/.https
0001:0A90 3A2F2F31[illegible]                          ://152.84.[illegible]
0001:0AA0 383A3432 3432 2F6D 65646964 6F726573 8:4242/medidores
0001:0AB0 2F00332E 302E3800 4C616233 69004C61 /.3.0.8.Lab3i.La
0001:0AC0 623369 2D                   00434250 b3i-1.[illegible].CBP
0001:0AD0 462D496F 54003932 33633536 62393236 F-IoT.923c56b926
0001:0AE0 66306636 65653161 63353830 62343963 f0f6ee1ac580b49c
0001:0AF0 ?[illegible]                                  004b0
0001:0B00 000080 3F CDCCCC3D 6F12833A 95BFD633 ...?ÍÌÌ=o..:.¿Ö3
0001:0B10 7D1D9026 B3CE010C CDCCCC3D 0AD7233C }..&³Î..ÍÌÌ=.×#<
```

Figura 8. Trecho do *firmware* não criptografado com os dados da apikey.

```
0000:B160 00000000 00000000 00000000 00000000 ................
0000:B170 00FFFFFF FFFFFFFF FFFFFFFF FFFFFFFF .ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0000:B180 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0000:B190 FFFFFFFF FFFFFFFF 08000000 43425046 ÿÿÿÿÿÿÿÿ....CBPF
0000:B1A0 2D496F54 00000000 00000000 00000000 -IoT............
0000:B1B0 00000000 00000000 00000000 69       ............i2
0000:B1C0 63       00000000 00000000 00000000   7............
0000:B1D0 00000000 00000000 00000000 00000000 ................
0000:B1E0 00000000 00000000 00000000 00000000 ................
0000:B1F0 00000000 00000000 00000000 00FFFFFF ............ÿÿÿ
0000:B200 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
```

Figura 9. Trecho do *firmware* não criptografado com os dados de acesso wifi.

O ESP32 possui mecanismos de segurança que permitem a criptografia do *firmware* através de seus eFuses [23].

O procedimento adotado nesse projeto foi o de gerar uma chave que deve ser gravada na memória do ESP32. Sendo assim, somente de posse dela que será possível gerar um novo *firmware* capaz de ser executado no ESP32 (Figura 10).

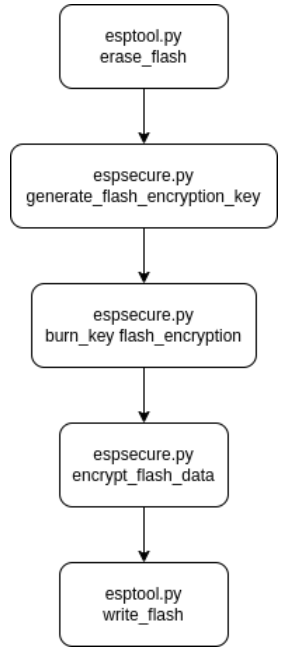

Figura 10. Fluxograma do procedimento de encriptação do *firmware*.

Com isso é possível desenvolver o *firmware* utilizando a IDE do ESP32 ou a IDE do Arduino, sendo necessário apenas possuir o código binário gerado pelo compilador.

Figura 11. Trecho do *firmware* criptografado com os dados da apikey.

Figura 12. Trecho do *firmware* criptografado com os dados de acesso wifi.

É importante observar que após o procedimento de gravação da chave no *Efuse Block 1* do ESP32, todo o *firmware* que for enviado deverá ser encriptado com a chave gerada. Caso a chave seja perdida, não será mais possível executar um novo código no dispositivo.

Após a encriptação e gravação do código no firware do ESP32, foi executado o procedimento de extração e abertura do binário em um editor hexadecimal. O conteúdo pode ser visto nas Figuras 11 e 12. Com isso é possível verificar a efetividade do procedimento, pois o código foi gravado no *firmware* de maneira criptografada.

V. CONCLUSÃO

O tema relacionado as boas práticas de segurança é amplo e neste projeto foi possível desenvolver uma abordagem que atende às necessidades de boa parte dos projetos que utilizam o ESP32. Como já foi mencionado no texto, cada projeto deverá ser avaliado de acordo com as suas especificidades. As técnicas abordadas nesse trabalho não visam impedir os ataques, mas mitigar os possíveis efeitos causados em uma invasão.

No caso deste trabalho houve o cuidado de se habilitar somente o protocolo TLS v1.2 ou acima (o protocolo TLS v1.3 ainda não é suportado pela maioria das bibliotecas disponíveis para o ESP32) e em não utilizar versões vulneráveis dos algoritmos de criptografia [24] [25].

Caso seja necessário a utilização do protocolo TLS v1.3 é possível fazer a implementação utilizando o módulo criptográfico ATECC608A através da conexão I2C. Este chip já vem embutido no ESP32-WROOM-32SE e com ele é possível fazer todas as operações de criptografia mais modernas.

Outro aspecto é referente a conexão WiFi utilizada pelo ESP32. O ideal é possuir uma rede independente, utilizando o protocolo WPA2-PSK com AES (CCMP) e utilizar uma senha longa para dificultar os ataques de força-bruta [26], porém, mesmo isso não impede ataques mais sofisticados que exploram a vulnerabilidade do protocolo WPA2 [27].

Para a prevenção de ataques é essencial que também sejam implementadas formas físicas de proteção, através da utilização de sistemas de monitoramento ou blindagens para restringir o acesso ao equipamento. O constante monitoramento dos fluxos de comunicação entre o dispositivo e o servidor também é importante para identificação de alguma vulnerabilidade que possa surgir futuramente.

Para os trabalhos futuros existe a ideia de implementar a parte de criptografia na comunicação utilizando o protocolo TLS v1.3, além da utilização de FPGA e *blockchain* [28], de forma a garantir a integridade dos dados.

[1] IEEE. Standard for an architectural framework for the internet of things (iot) ieee 2413. *IEEE-2413 Working Group. Technical Report*, 2019.

[2] RUSSELL, B.; DUREN, D. V. *Practical internet of things security*. [S.l.]: Packt Publishing Ltd, 2018.

[3] OWASP Internet of Things Project. 2018. https://wiki.owasp.org/index.php/OWASP_Internet_of_Things_Project#tab=IoT_Top_10. [Online; Maio-2022].

[4] IOT Device Criteria. 2021. https://www.nist.gov/itl/executive-order-improving-nations-cybersecurity/iot-device-criteria. [Online; Maio-2022].

[5] EXECUTIVE Order on Improving the Nations's Cybersecurity. 2022. https://www.cisa.gov/executive-order-improving-nations-cybersecurity. [Online; Maio-2022].

[6] CVE-2019-17391 - Fault Injection and eFuse protections. 2019. https://nvd.nist.gov/vuln/detail/CVE-2019-17391. [Online; Maio-2022].

[7] CVE-2019-15894 - Fault Injection and Secure Boot. 2019. https://nvd.nist.gov/vuln/detail/CVE-2019-15894. [Online; Maio-2022].

[8] CVE-2019-12587 - Zero PMK Installation. 2019. https://nvd.nist.gov/vuln/detail/CVE-2019-12587. [Online; Maio-2022].

[9] CVE-2019-12586 - EAP DoS. 2019. https://nvd.nist.gov/vuln/detail/CVE-2019-12586. [Online; Maio-2022].

[10] CVE-2018-18558 - Secure Boot Bypass. 2018. https://nvd.nist.gov/vuln/detail/CVE-2018-18558. [Online; Maio-2022].

[11] BUTUN, I.; ÖSTERBERG, P.; SONG, H. Security of the internet of things: Vulnerabilities, attacks, and countermeasures. *IEEE Communications Surveys & Tutorials*, IEEE, v. 22, n. 1, p. 616–644, 2019.

[12] GENG, H. *Internet of things and data analytics handbook*. [S.l.]: John Wiley & Sons, 2017.

[13] HTTP vs. MQTT: A tale of two IoT protocols. 2018. https://cloud.google.com/blog/products/iot-devices/http-vs-mqtt-a-tale-of-two-iot-protocols. [Online; Maio-2022].

[14] MOZILLA SSL Configuration Generator. 2021. https://ssl-config.mozilla.org/. [Online; Maio-2022].

[15] TCPDUMP. 2021. https://www.tcpdump.org/. [Online; Maio-2022].

[16] WIRESHARK. 2021. https://www.wireshark.org/. [Online; Maio-2022].

[17] ESPRESSIF - HTTPS server. 2021. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/protocols/esp_https_server.html. [Online; Maio-2022].

[18] MBED TLS. 2021. https://github.com/ARMmbed/mbedtls. [Online; Maio-2022].

[19] WIFICLIENTSECURE. 2021. https://github.com/espressif/arduino-esp32/tree/master/libraries/WiFiClientSecure. [Online; Maio-2022].

[20] PRINETTO, P.; ROASCIO, G. Hardware security, vulnerabilities, and attacks: A comprehensive taxonomy. In: *ITASEC*. [S.l.: s.n.], 2020. p. 177–189.

[21] TEHRANIPOOR, M.; WANG, C. *Introduction to hardware security and trust*. [S.l.]: Springer Science & Business Media, 2011.

[22] THE real story of STUXNET. 2013. https://spectrum.ieee.org/the-real-story-of-stuxnet. [Online; Maio-2022].

[23] ESPRESSIF Docs: Flash Encryption. 2018. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/security/flash-encryption.html. [Online; Maio-2022].

[24] MCKAY, K.; COOPER, D. *Guidelines for the selection, configuration, and use of Transport Layer Security (TLS) implementations*. [S.l.], 2017.

[25] SSL/TLS Vulnerabilities. 2021. https://www.hhs.gov/sites/default/files/securing-ssl-tls-in-healthcare-tlpwhite.pdf. [Online; Maio-2022].

[26] SALMON, A.; LEVESQUE, W.; MCLAFFERTY, M. *Applied Network Security*. [S.l.]: Packt Publishing Ltd, 2017.

[27] VANHOEF, M.; PIESSENS, F. Key reinstallation attacks: Forcing nonce reuse in wpa2. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. [S.l.: s.n.], 2017. p. 1313–1328.

[28] XU, L. et al. Fpga based blockchain system for industrial iot. In: IEEE. *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. [S.l.], 2020. p. 876–883.