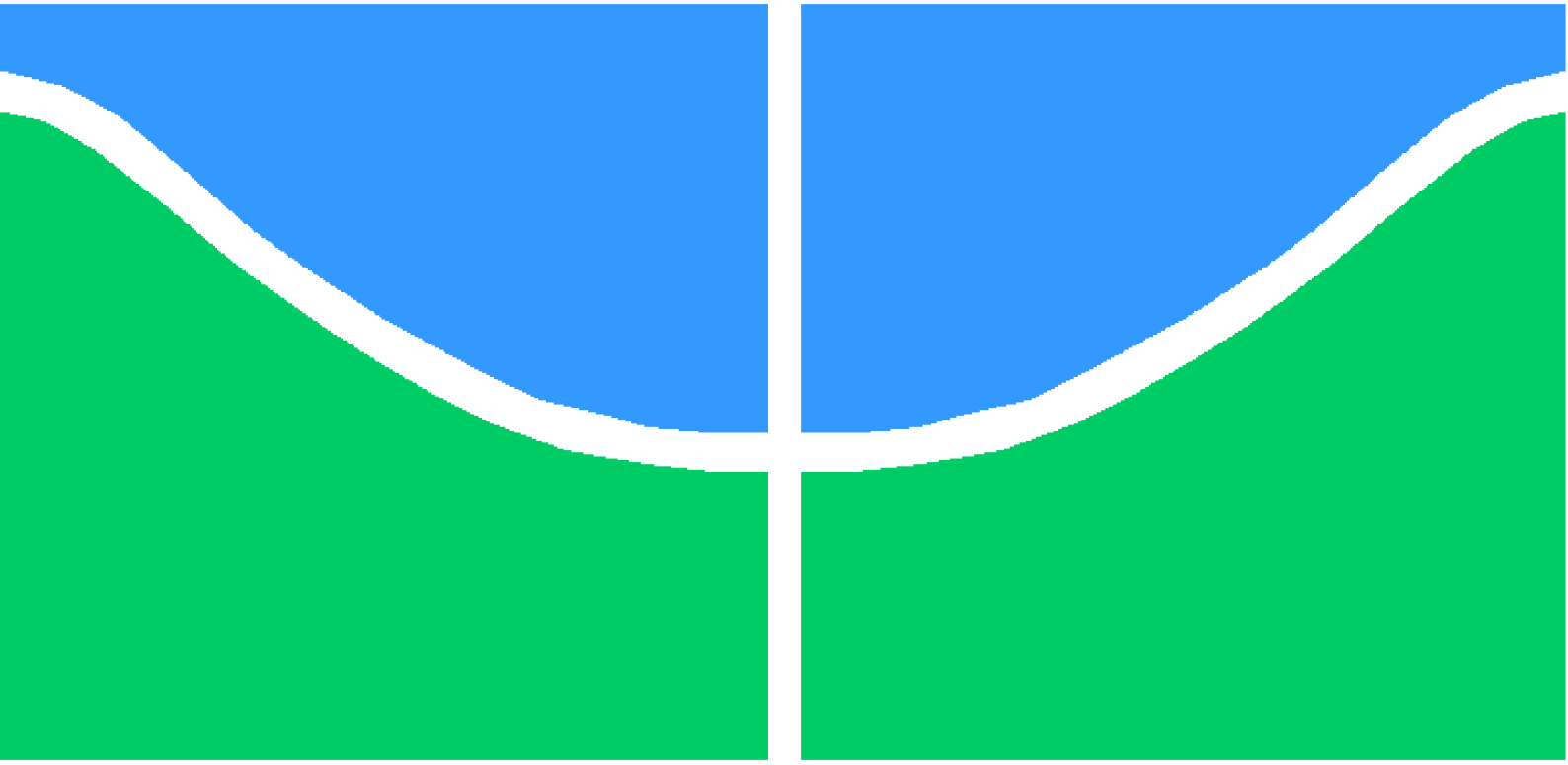


Universidade de Brasília – UnB
Faculdade UnB Gama – FGA
Engenharia de Software

Segurança na Era da IoT: Prevenção de Vulnerabilidades no Ciclo de Vida do Software

Autor: Guilherme Veríssimo Cerveira Braz
Orientadora: Prof. Dra. Elaine Venson

Brasília, DF
2023

Guilherme Veríssimo Cerveira Braz

Segurança na Era da IoT: Prevenção de Vulnerabilidades no Ciclo de Vida do Software

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Orientador: Prof. Dra. Elaine Venson

Brasília, DF

2023

Agradecimentos

Agradeço a minha orientadora Elaine por todo o apoio, paciência e direção fornecidos durante todo o desenvolvimento deste TCC. Sem o seu conhecimento e dedicação, este trabalho não teria sido possível. Gostaria também de agradecer à minha família, por sempre acreditar em mim e me apoiar incondicionalmente. Por fim, gostaria de agradecer aos meus amigos, que sempre estiveram ao meu lado, me animando e me ajudando a enfrentar todos os desafios.

Resumo

Em meio ao panorama tecnológico contemporâneo, observa-se um crescimento notável da Internet das Coisas (IoT), que interliga variados dispositivos em uma rede mundial, a Internet. Essa evolução, embora otimize a interação e o compartilhamento de dados, gera desafios emergentes relacionados à segurança e privacidade, que foram ainda mais exacerbados pelo contexto da pandemia COVID-19 e pela ascensão do modelo de trabalho remoto com dispositivos desprotegidos. No âmbito deste projeto, procura-se aprofundar na compreensão das vulnerabilidades de segurança na IoT, mediante a revisão bibliográfica, estudos de caso e análises de frameworks de segurança e relatórios como o MITRE, OWASP e ENISA - *Good Practices for Secure Development of IoT*, com o objetivo de identificar as principais ameaças em software de dispositivos IoT, analisar casos de ataque e por fim desenvolver um guia de recomendações para prevenir vulnerabilidades ao longo do ciclo de vida do software.

Palavras-chave: Internet das Coisas (IoT); Segurança; Privacidade; OWASP; MITRE; ENISA; Ciclo de Vida do Software; Guia de Recomendações; Análise de Casos.

Abstract

In the midst of the contemporary technological landscape, there is a notable growth of the Internet of Things (IoT), which interconnects various devices in a worldwide network, the Internet. This evolution, while optimizing interaction and data sharing, generates emerging challenges related to security and privacy, which have been further exacerbated by the context of the COVID-19 pandemic and the rise of the remote work model with unprotected devices. Within the scope of this project, an attempt is made to deepen the understanding of security vulnerabilities in IoT, through a bibliographical review, case studies and analysis of security frameworks and reports such as MITER, OWASP and ENISA - Good Practices for Secure Development of IoT, with the objective of identifying the main threats in IoT device software, analyzing attack cases and finally developing a recommendation guide to prevent vulnerabilities throughout the software lifecycle.

Key-words: Internet of Things (IoT); Security; Privacy; OWASP; MITRE; ENISA; Software Lifecycle; Recommendations Guide; Case Analysis.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ANATEL	Agência Nacional De Telecomunicações
API	Application Programming Interface
CEU	Commission of European Union
CEN	European Committee for Standardization
CoAP	Constrained Application Protocol
CVEs	Common Vulnerabilities and Exposures
DoS	Denial of Service
ENISA	European Union Agency for Cybersecurity
ETSI	European Telecommunications Standards Institute
FTP	File Transfer Protocol
SFTP	Secure File Transfer Protocol
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electronic and Electric Engineers
IETF	Internet Engineering Task Force
IoT	Internet of Things
ISO	International Standards Organization
ITU	International Telecommunications Union
LAN	Local Area Network
LTE	Long-Term Evolution
LGPD	Lei Geral de Proteção de Dados
MQTT	Message Queue Telemetry Transport
NFC	Near Field Communication

OWASP	Open Web Application Security Project
SDLC	Software Development Life Cycle
SFTP	Secure File Transfer Protocol
SSH	Secure Shell
SSL	Secure Sockets Layer
TI	Tecnologia da Informação
TIC	Tecnologias de Informação e Comunicação
UnB	Universidade de Brasília
WAN	Wide Area Network
XMPP	Extensible Messaging and Presence Protocol

Sumário

1 Introdução

1.1 Contexto

A Internet é uma rede de computadores que interconecta centenas de milhões de dispositivos de computação em todo o mundo (Kurose, Ross, 2014). Desde o seu surgimento, a Internet tem sido uma tecnologia revolucionária, permitindo a interação e troca de informações entre dispositivos de maneira mais rápida e eficiente do que nunca antes.

No entanto, a evolução rápida da Internet também trouxe muitos desafios e questões relacionadas à segurança dos dados, especialmente no que diz respeito à proteção de informações pessoais e confidenciais dos usuários. A crescente complexidade e amplitude da ameaça cibernética tem exigido esforços constantes por parte de instituições globais, empresas e governos para prevenir ataques cibernéticos e garantir a segurança dos dados.

Paralelamente, ocorreu o surgimento da Internet das Coisas (IoT, na sigla em inglês), que foi criado por Kevin Ashton em 1999, durante seu trabalho na Procter & Gamble, como uma forma de expressar a tendência dominante da época - a Internet. De acordo com a definição (MARWEDEL, 2021), o termo IoT descreve a presença generalizada de uma variedade de dispositivos, como sensores, atuadores e telefones móveis, que são capazes de interagir e cooperar entre si através de esquemas de endereçamento únicos, a fim de atingir objetivos comuns.

De acordo com uma previsão do setor (Ericson mobile, 2022), o número total de dispositivos conectados à Internet das Coisas (IoT) em todo o mundo deve mais do que dobrar de 14,6 bilhões em 2022 para 30,2 bilhões em 2030. Outro relatório (Cisco, 2020) estima que o número de dispositivos conectados a redes IP será mais de três vezes a população global até 2023, estimando que em 2025 serão transferidos cercas de 80 zettabytes de dados no ano (1 zettabyte $= 2^{70}$ bytes). Essa expansão traz muitos benefícios, como acesso à informação e conectividade, mas também cria novos problemas de segurança e privacidade.

A pandemia de COVID-19 também trouxe novas preocupações em relação à segurança da tecnologia, com muitas pessoas trabalhando de forma remota e usando dispositivos pessoais para acessar redes corporativas. Isso aumenta o risco de ataques cibernéticos, pois esses dispositivos não estão protegidos pelas medidas tradicionais de segurança, como firewalls de perímetro e outras tecnologias. De acordo com uma pesquisa realizada pela Gartner Inc. com 229 líderes de Recursos Humanos em 2 de abril (Gartner, 2020), quase 50% das organizações relataram que 81% ou mais de seus funcionários trabalharam de forma remota durante a pandemia de coronavírus. Outros 15% dos entrevistados disse-

ram que 61-80% dos funcionários estão trabalhando de forma remota até o momento da pesquisa. A pesquisa concluiu que muitos trabalhadores planejam trabalhar de forma remota com mais frequência no futuro, o que impactará nessa dinâmica já complexa da cibersegurança.

A maior utilização da tecnologia de nuvem também trouxe novos desafios de segurança, junto com outras tecnologias como a microsegmentação e o *edge computing* tornando mais difícil controlar e proteger os dados. Além disso, a complexidade das interações entre várias tecnologias e a falta de visibilidade dentro desse fluxos cada vez mais emaranhados torna mais difícil para os desenvolvedores e também para os próprios usuários entenderem como suas informações são usadas e compartilhadas. Isso levanta diversas questões éticas e legais sobre como garantir a privacidade dos usuários e proteger seus dados(ENISA, 2019).

Esse contexto em que o mundo da tecnologia da informação está em constante evolução e crescimento, com uma crescente globalização e número de pessoas conectadas à internet e usando dispositivos tecnológicos ou produtos de TIC (Tecnologia da Informação e Comunicação), alimenta a preocupação sobre como prevenir os ataques em dispositivos IoT de acordo com práticas de segurança no ciclo de vida de software.

1.2 Problema

O crescimento explosivo e a adoção massiva de dispositivos de Internet das Coisas (IoT) em todos os aspectos de nossas vidas trouxe à tona preocupações significativas de segurança. Na medida em que mais dispositivos se tornam conectados, o risco de ataques cibernéticos e exposição de dados privados aumenta.

No âmbito de software, esses desafios tornam-se ainda mais proeminentes, uma vez que a maioria dos dispositivos IoT opera sobre a base de softwares que são, em muitos casos, falhos em termos de segurança. De acordo com o relatório de ameaças IoT 2020 da Palo Alto (Unit 42 Palo Alto, 2020), 98% de todo tráfego IoT é desprotegido e não criptografado, expondo dados pessoais e confidenciais. Além disso, muitos desses dispositivos rodam em softwares ultrapassados, apresentando assim vulnerabilidades fáceis de serem exploradas.

A falta de uma cultura de segurança em software e de práticas de segurança durante o ciclo de vida do software (SDLC, na sigla em inglês) é um dos principais fatores que contribuem para essas falhas. Muitos fabricantes de dispositivos IoT não consideram as práticas de segurança de software desde o início do desenvolvimento do software, resultando em dispositivos vulneráveis e facilmente comprometíveis.

Além disso, a ausência de compreensão clara por parte dos usuários sobre como seus dados são usados e compartilhados por esses dispositivos, assim como a complexidade crescente de interações entre diferentes tecnologias, aumentam ainda mais o risco de ataques cibernéticos e violações de privacidade.

A OWASP (*Open Web Application Security Project*) (OWASP, 2023), uma fundação sem fins lucrativos que promove projetos focados em segurança de software, identificou as dez vulnerabilidades mais comuns em produtos IoT, entre as quais incluem senhas fracas, serviços de rede inseguros e a falta de mecanismos seguros para atualizações. Essa lista destaca a necessidade de medidas de segurança robustas para proteger contra essas ameaças comuns.

Neste contexto, torna-se importante discutir soluções para desenvolver e implementar práticas eficazes de segurança de software que possam ajudar a prevenir ataques em dispositivos IoT e proteger a privacidade e os dados dos usuários.

Portanto, este trabalho visa abordar a seguinte questão de pesquisa:

- Considerando as principais vulnerabilidades de segurança em dispositivos IoT introduzidas dentro do ciclo de vida do software, quais são as melhores práticas recomendadas para preveni-las ou detectá-las?

1.3 Objetivos

1.3.1 Objetivo Geral

Este projeto tem como objetivo principal desenvolver orientações aos desenvolvedores para prevenir vulnerabilidades de IoT no ciclo de desenvolvimento de software, com base na análise de casos reais ocorridos no mundo envolvendo vulnerabilidades pré-selecionadas do OWASP Top 10 IoT.

1.3.2 Objetivos Específicos

Para atingir este objetivo, serão seguidos os seguintes objetivos específicos:

1. Estudo do cenário de segurança em IoT com ênfase no ciclo de vida do software.
2. Seleção e análise de algumas vulnerabilidades dentre as top 10 vulnerabilidades de IoT identificadas pela OWASP que se enquadram no ciclo de desenvolvimento de software.
3. Investigação de casos ocorridos no mundo envolvendo as vulnerabilidades selecionadas, analisando como ocorreram e quais foram as consequências.

4. Produção de recomendações de contramedidas para prevenir ou detectar essas vulnerabilidades no ciclo de desenvolvimento de software, com base nas lições aprendidas com os casos analisados.

1.3.3 Metodologia

A abordagem adotada para conduzir o estudo visa produzir um conjunto de orientações de boas práticas que possam ser aplicadas para prevenir os principais ataques em dispositivos IoT. A metodologia utiliza uma combinação de pesquisa documental e análise bibliográfica. A pesquisa documental busca informações relevantes sobre ataques reais a dispositivos IoT e medidas de segurança aplicáveis, enquanto a análise bibliográfica consiste na revisão da literatura existente sobre o assunto. Através dessas abordagens, o estudo visa estabelecer um conjunto de boas práticas recomendadas para o ciclo de vida do desenvolvimento de software (SDLC) de dispositivos IoT. As etapas do estudo incluem a coleta de dados, análise das vulnerabilidades mais frequentes em software e a proposta de recomendações para prevenir essas vulnerabilidades no SDLC.

1.3.4 Organização do Trabalho

Este trabalho está organizado em capítulos:

- **Capítulo 2 - Referencial Teórico**: explora as áreas de redes, IoT, cibersegurança e desenvolvimento de software, abordando aspectos relevantes para o estudo.
- **Capítulo 3 - Metodologia**: apresenta a abordagem adotada para conduzir o estudo, incluindo a pesquisa documental e a análise bibliográfica. Elucida o processo de coleta de dados, análise das vulnerabilidades mais frequentes e a proposta de recomendações para mitigá-las no ciclo de vida do desenvolvimento de software.
- **Capítulo 4 - Análise de casos**: explora três casos de ataque relevantes no cenário de IoT. Analisando para cada um deles o contexto e detalhes do ataque, impacto, soluções pós-ataque e mitigação e, por fim, detalhes a cerca da prevenção e boas práticas no ciclo de vida de software.
- **Capítulo 5 - Guia de Recomendações**: esse capítulo, consiste em trazer uma guia de recomendações de segurança de software para os desenvolvedores de IoT, trazendo detalhes e recomendações para cada fase do SDLC e outras dicas importantes.
- **Capítulo 6 - Conclusão**: nesse capítulo é feita a conclusão final do trabalho.

2 Referencial Teórico

2.1 Conceitos de Rede

2.1.1 Conceitos de Rede e o Modelo OSI

As redes de computadores são uma parte integral da arquitetura IoT, e para um entendimento mais aprofundado sobre redes, é fundamental começar pelos conceitos básicos e pela organização das redes em camadas, o que nos leva ao modelo OSI (*Open Systems Interconnection*).

O modelo OSI consiste em um modelo conceitual usado para entender e descrever como diferentes protocolos de rede interagem e cooperam para fornecer serviços de rede. Este modelo divide as funções de uma rede em sete camadas: Física, Enlace de Dados, Rede, Transporte, Sessão, Apresentação e Aplicação, cada qual sendo responsável por um aspecto específico da comunicação de rede, fornecendo serviços para as camadas acima dela. A Figura 1 ilustra as camadas do modelo OSI (ZIMMERMANN, 1980).

Figura 1 – Modelo OSI (Open Systems Interconnection)

Fonte: Adaptação (ZIMMERMANN, 1980)

Cada camada do modelo OSI possui propósito e funcionamento específicos:

1. **Camada Física:** É a primeira camada do modelo OSI e lida com a transmissão de dados brutos ou bits entre dispositivos através de um meio de transmissão físico.

Essa camada define características elétricas, mecânicas e funcionais da interface e do meio físico, incluindo os aspectos relacionados a cabos, conectores e sinais elétricos (PINHEIRO, 2004).

2. **Camada de Enlace de Dados:** A segunda camada do modelo OSI, esta camada fornece um meio confiável para a transmissão de dados entre dispositivos que estão conectados diretamente entre si. Ela é responsável por detectar e, opcionalmente, corrigir erros que possam ocorrer na camada física. Também gerencia o controle de fluxo, garantindo que os dados sejam transmitidos na velocidade correta para evitar uma sobrecarga (PINHEIRO, 2004).

3. **Camada de Rede:** Esta é a terceira camada do modelo OSI e é responsável pela transferência de pacotes de dados de um ponto a outro na rede. Sua principal função é o roteamento, que inclui a determinação do caminho ideal e a comutação de pacotes de uma rede para outra. Esta camada também gerencia o endereçamento lógico e o controle de congestionamento (PINHEIRO, 2004).

4. **Camada de Transporte:** A Camada de Transporte, quarta camada do modelo OSI, estabelece conexões lógicas fim a fim, segmenta dados em unidades apropriadas e gerencia a multiplexação de conexões. Ela pode adaptar-se às necessidades do nível superior, aumentando a velocidade de transmissão ou reduzindo custos. Oferece várias classes de serviço, desde comunicação ponto a ponto livre de erros até o envio de mensagens para múltiplos destinos. As funções implementadas dependem da qualidade de serviço desejada, variando de protocolos simples até protocolos que detectam e recuperam erros (PINHEIRO, 2004).

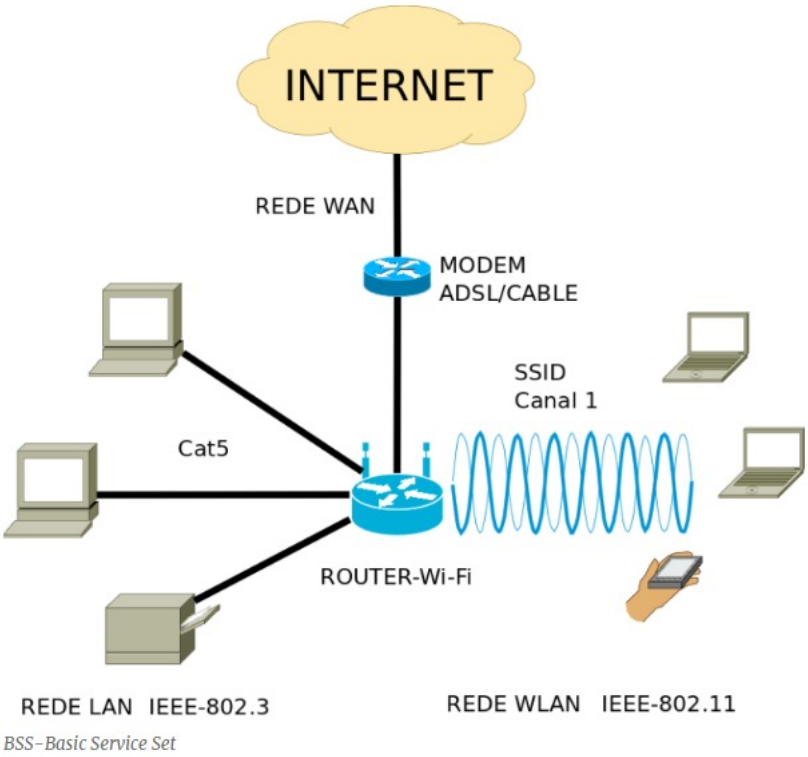
5. **Camada de Sessão:** A quinta camada do modelo OSI, a Camada de Sessão, estabelece, gerencia e termina conexões entre aplicações em dispositivos finais. Ela coordena a comunicação entre sistemas, regulando o estabelecimento, manutenção e terminação de sessões (PINHEIRO, 2004).

6. **Camada de Apresentação:** Esta é a sexta camada do modelo OSI, a camada de Apresentação, a qual se encarrega da transformação dos dados para garantir que sejam interpretáveis pela camada de aplicação. Ela se ocupa da codificação e conversão de caracteres, compressão de dados e, mais importante, da criptografia e descriptografia para garantir a privacidade dos dados (PINHEIRO, 2004).

7. **Camada de Aplicação:** A última e sétima camada do modelo OSI, a Camada de Aplicação, provê serviços de rede para aplicações, permitindo que elas comuniquem entre si. Protocolos comumente associados a essa camada incluem HTTP, FTP e SMTP. Essa camada é a interface entre as aplicações e os serviços de rede, permitindo que os usuários acessem a rede (PINHEIRO, 2004).

2.1.2 Tipos de Redes

Com base em diferentes critérios, como escala, propriedade, operação, arquitetura e topologia, as redes podem ser classificadas em vários tipos. Alguns dos tipos mais comuns de redes são redes de área local (LANs), redes de área ampla (WANs), e redes de área local sem fio (WLANs), conforme mostra a Figura 2.



Fonte: (REZ, 2018)

Figura 2 – Esquema das Redes WAN, LAN e WLAN

Uma LAN é uma rede que cobre uma área geográfica pequena, geralmente dentro de um único edifício, como uma casa ou escritório. Os dispositivos nesta rede podem compartilhar recursos, como impressoras e conexões com a Internet (KIZZA, 2015). Por outro lado, uma WAN cobre uma área geográfica maior, muitas vezes abrangendo cidades, estados e até mesmo países. As WANs são usadas para conectar várias LANs, permitindo que os usuários em diferentes locais compartilhem recursos e se comuniquem entre si. A Internet é a maior e mais conhecida WAN, conectando computadores e redes em todo o mundo (KIZZA, 2015). As WLANs, ou redes sem fio, são uma variação das LANs que usam ondas de rádio para transmitir e receber dados, permitindo que os dispositivos se conectem à rede sem a necessidade de cabos físicos (KIZZA, 2015).

2.1.3 Encaminhamento de Pacotes e Modelos de Conexão

Uma das principais funções de uma rede é o encaminhamento de pacotes, também conhecido como roteamento, o qual consiste da camada 3 do modelo OSI (ver Figura 1). O roteamento é o processo de mover pacotes de dados de uma rede para outra para chegar ao seu destino final. Os dispositivos conhecidos como roteadores são responsáveis por este processo, escolhendo o melhor caminho para cada pacote com base em várias métricas, como a distância e a quantidade de tráfego (Kurose, Ross, 2014).

Existem vários modelos de conexões de rede, cada um com suas características únicas e usos específicos. Alguns dos modelos de conexão mais comuns incluem redes peer-to-peer (P2P), cliente-servidor e device-to-device.

Em uma rede P2P, cada dispositivo na rede pode atuar como cliente ou servidor para outros dispositivos na rede. Isso é frequentemente usado para compartilhar arquivos e outros recursos diretamente entre os dispositivos na rede. Em contraste, uma rede cliente-servidor tem dispositivos dedicados, chamados servidores, que fornecem recursos e serviços a outros dispositivos na rede, chamados clientes. Finalmente, as redes device-to-device permitem que os dispositivos se conectem e se comuniquem diretamente uns com os outros, sem a necessidade de um servidor intermediário. Este tipo de rede é comum na Internet das Coisas (IoT), onde os dispositivos podem se comunicar diretamente para compartilhar informações e controlar outros dispositivos conectados (TANEMBAUM, 2011). Compreender esses conceitos básicos e modelos é fundamental para entender como as redes funcionam e como os dispositivos se comunicam e compartilham informações entre si, permitindo uma vasta gama de aplicações no mundo real.

2.1.4 Padrões de conexão

Existem diferentes padrões de conexão que podem ser usados em redes. Alguns dos mais comuns incluem:

- O Wi-Fi, definido pelo padrão IEEE 802.11, é uma tecnologia sem fio que permite a conexão de dispositivos a uma rede sem fio. Ele usa a faixa de frequência de 2,4 GHz e 5 GHz para se comunicar com dispositivos compatíveis (IEEE Computer Society, 2020a). No contexto de um sistema IoT, os dispositivos podem se conectar a uma rede Wi-Fi para se comunicar com outros dispositivos e com um servidor de backend. É importante lembrar que a conexão Wi-Fi pode ser mais lenta e menos confiável do que outras opções de conexão, como Ethernet ou LTE.
- Bluetooth é um protocolo de comunicação sem fio de curto alcance, conforme definido pelo padrão IEEE 802.15.1. Este protocolo é otimizado para conexões entre dispositivos em proximidade próxima, tornando-o menos abrangente que o Wi-Fi.

O Bluetooth é comumente empregado em situações como transferência de arquivos e conexão de dispositivos periféricos, como fones de ouvido e teclados, a dispositivos móveis. Sua eficiência energética e versatilidade o tornam adequado para uma ampla gama de aplicações, desde wearables até automóveis (IEEE Computer Society, 2005).

- O Zigbee é um protocolo de comunicação sem fio de curto alcance, estabelecido pelo padrão IEEE 802.15.4. Este protocolo é projetado para comunicações entre dispositivos a uma distância intermediária, maior que a do Bluetooth, mas menor que a do Wi-Fi. O Zigbee é amplamente utilizado em aplicações de Internet das Coisas (IoT), como automação residencial, monitoramento de sensores e sistemas de iluminação inteligente. Sua confiabilidade, segurança e baixo consumo de energia o tornam ideal para aplicações que requerem longa duração e mínima interação humana (IEEE Computer Society, 2020b).

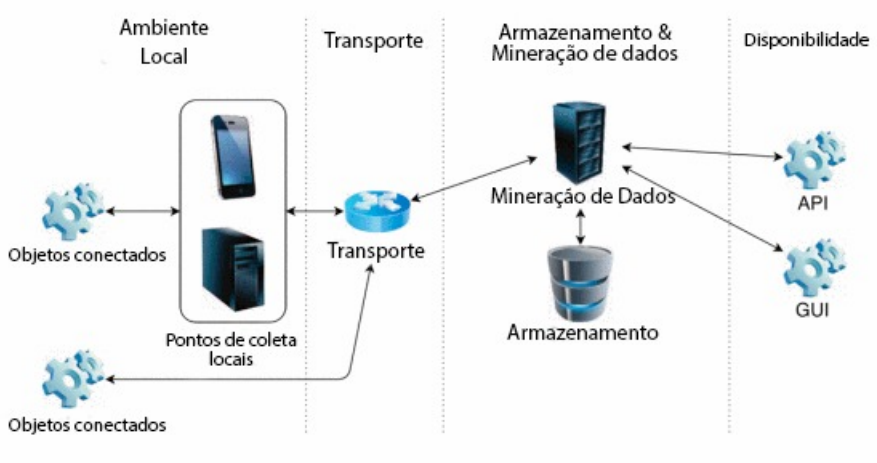
- O Ethernet é um padrão de conexão com fio comumente usado em redes LAN, conforme definido pelo padrão IEEE 802.3. Ele permite a conexão de dispositivos através de cabos de cobre ou fibra óptica. No contexto de um sistema IoT, o Ethernet pode ser usado para conectar dispositivos a um servidor de backend ou a outros dispositivos próximos. A vantagem do Ethernet é que ele oferece uma conexão mais segura e disponível (IEEE Computer Society, 2018).

2.2 Conceitos de IoT

De acordo com a ENISA (ENISA, 2023), Internet das Coisas (IoT) é uma área de tecnologia que abrange um amplo ecossistema de dispositivos e serviços interconectados que coletam, trocam e processam dados para se adaptarem dinamicamente a um contexto. Esses dispositivos podem ser de diversos tipos, como sensores de temperatura, luz e umidade, câmeras, dispositivos de segurança, dispositivos médicos e muito mais. Sendo assim, a área de IoT tem aplicações em vários setores, como saúde, transporte, indústria, agricultura e vem expandindo cada vez mais sua área de atuação.

De acordo com Dorsemaine et al. (2015), a arquitetura comumente presente em sistemas IoT é uma arquitetura em camadas, a qual pode ser mais facilmente compreendida se dividida em quatro camadas principais: ambiente local, transporte, armazenamento & mineração de dados e disponibilidade, conforme mostra a Figura 3. A camada de ambiente local inclui os dispositivos físicos da IoT, como sensores e atuadores, que coletam os dados do ambiente e podem realizar ações baseadas nesses dados, além também de outros dispositivos locais como servidores de armazenamento local. A camada de transporte é responsável por gerenciar a troca de dados entre os dispositivos e o local onde esses dados serão armazenados e processados em massa, sendo geralmente em nuvem. A camada

de armazenamento e mineração de dados é onde os dados coletados são armazenados e processados conforme a necessidade e contexto do produto. Finalmente, a camada de disponibilidade envolve disponibilizar esses dados para uso pelo usuário final, através de APIs, GUIs ou mesmo aplicações.



Fonte: (DORSEMAINE et al., 2015)

Figura 3 – Arquitetura IoT

2.2.1 Sistemas Embarcados

Sistemas Embarcados são sistemas de processamento de informações incorporados em produtos que os envolvem (MARWEDEL, 2021). Eles são projetados para ser compactos, confiáveis e eficientes em termos de energia. São utilizados em uma ampla gama de aplicações, incluindo aviônicos, sistemas de navegação, automóveis, aparelhos domésticos e sistemas de segurança.

No contexto da Internet das Coisas (IoT), os sistemas embarcados são usados para conectar objetos do mundo real ao mundo digital, permitindo que eles sejam monitorados e controlados remotamente. Isso é realizado através da conexão com a internet, o que possibilita o controle remoto de dispositivos como sensores de temperatura, por exemplo, através de aplicativos móveis.

Existem algumas características comuns de sistemas embarcados em IoT:

- Eles são geralmente compactos, o que é essencial para aplicações onde o espaço é limitado (MARWEDEL, 2021). Note que o tamanho de um sistema embarcado pode variar dependendo da sua aplicação específica. Alguns podem ser realmente muito pequenos, enquanto outros podem ser mais substanciais.
- Eles são projetados para ser confiáveis e robustos, pois frequentemente operam em ambientes adversos (MARWEDEL, 2021). Isso pode incluir ambientes com variações extremas de temperatura, pressão, umidade, vibração, etc.

- Eles são projetados para ser eficientes em termos de energia, uma vez que muitos dispositivos IoT são alimentados por baterias ou por fontes de energia limitadas (MARWEDEL, 2021).
- Eles são geralmente projetados para conectar-se à internet e transmitir e receber dados, o que é uma característica central da IoT (MARWEDEL, 2021).
- Eles geralmente permitem controle remoto, o que possibilita aos usuários acessar e gerenciar os dispositivos de praticamente qualquer lugar (MARWEDEL, 2021).

A principal característica dos sistemas embarcados é a interface do software com os sistemas de hardware. Devido à inflexibilidade do hardware, as decisões sobre a funcionalidade do software geralmente não podem ser adiadas até a implementação (SOMMERVILLE, 2016). Isso significa que o software embarcado precisa ser projetado e implementado de forma a se adequar ao hardware específico, tornando o desenvolvimento de software embarcado um desafio especial. Além disso, o software embarcado deve ser otimizado para consumir recursos de forma eficiente, como memória e processamento, já que os dispositivos que ele controla costumam ter recursos limitados.

Exemplos de embarcados comumente usados são:

- Rapberry pi;
- Arduino;
- ESP32.

2.2.2 Protocolos de Mensageria

Os protocolos de mensageria (camada 7 do modelo OSI) são fundamentais para o funcionamento da IoT, pois permitem a comunicação entre dispositivos e sistemas. De forma geral um protocolo é um conjunto de regras e procedimentos que regem a comunicação entre dispositivos. De acordo com Kurose, Ross (2014):

> "Um protocolo define o formato e a ordem das mensagens trocadas entre duas ou mais entidades comunicantes, bem como as ações realizadas na transmissão e/ou no recebimento de uma mensagem ou outro evento."

Sendo assim, existem vários protocolos diferentes que são utilizados na IoT, cada um com seus próprios benefícios e limitações. Um dos protocolos mais populares utilizados na IoT é o HTTP (*Hypertext Transfer Protocol*), que é o protocolo utilizado para transferir dados na *World Wide Web*. Ele é amplamente utilizado para comunicação entre dispositivos IoT e servidores, pois é simples, confiável e amplamente suportado.

Outro protocolo popular é o ICMP (*Internet Control Message Protocol*), que é utilizado para enviar mensagens de erro e diagnóstico entre dispositivos. Ele também é amplamente utilizado para garantir a disponibilidade e confiabilidade da rede. Além do HTTP e ICMP, existem outros protocolos importantes que são muito utilizados na IoT, como:

- O MQTT (*Message Queuing Telemetry Transport*) é um protocolo de mensagens leves que é projetado para conectar dispositivos em redes de baixa banda larga e de baixa potência. Ele é bastante utilizado em aplicativos de IoT, como monitoramento de sensores e automação residencial. Ele foi inventado pelo Dr. Andy Stanford-Clark da IBM e Arlen Nipper da Arcom (agora Eurotech), em 1999 (MQTT, 2023).
- O CoAP (*Constrained Application Protocol*), desenvolvido pela *Internet Engineering Task Force* (IETF), é um protocolo de aplicativo para dispositivos IoT restritos, projetado para ser leve e eficiente em termos de recursos. Ele permite que os dispositivos IoT se comuniquem facilmente, além de fornecer integração com a Web através da interface com o HTTP. O CoAP também atende a requisitos especializados, como suporte a multicast, baixo overhead e simplicidade para ambientes restritos (SHELBY; HARTKE; BORMANN, 2014).
- O XMPP (*Extensible Messaging and Presence Protocol*) é um protocolo de mensagens aberto desenvolvido pela XMPP Standards Foundation (XSF) que é amplamente utilizado em aplicativos de comunicação, como bate-papo e mensagens instantâneas (XSF, 2023). Ele é projetado para ser escalável e confiável, permitindo uma comunicação segura e eficiente entre dispositivos IoT.

2.2.3 Application programming Interface (API)

As APIs (*Application Programming Interfaces*) são um conjunto de protocolos e padrões que permitem a comunicação entre diferentes sistemas e serviços. De acordo com Kurose, Ross (2014), uma API pode ser explicada como:

> "Interface de Programação de Aplicação que especifica como o programa que é executado no sistema final solicita à infraestrutura da Internet que envie dados a um programa de destino específico, executado em outro sistema final".

No contexto da Internet das Coisas (IoT), as APIs são usadas para permitir que dispositivos IoT se comuniquem com sistemas de backend e com outros dispositivos. Algumas das principais vantagens de usar APIs em sistemas IoT incluem (Red Hat, 2020):

- Facilidade de integração: as APIs permitem que os dispositivos IoT se integrem a outros sistemas e serviços de maneira mais fácil e rápida, o que pode acelerar o desenvolvimento de soluções IoT.

- Flexibilidade: as APIs permitem que os dispositivos IoT sejam integrados a diferentes tipos de sistemas de backend, o que permite maior flexibilidade na escolha de plataformas e tecnologias.
- Segurança: as APIs podem ser projetadas com medidas de segurança, como autenticação e criptografia, para proteger os dados transmitidos entre dispositivos IoT e sistemas de backend.

Existem diferentes tipos de APIs que podem ser usadas em sistemas IoT, como APIs REST (*Representational State Transfer*) e APIs de mensagem. As APIs REST são baseadas em HTTP e usam verbos HTTP, como GET, POST e DELETE, para solicitar e receber dados. As APIs de mensagem são usadas para enviar e receber mensagens em tempo real entre dispositivos IoT e sistemas de backend (Red Hat, 2020).

Em resumo, as APIs são uma parte importante da arquitetura de sistemas IoT, pois permitem a comunicação e integração entre dispositivos IoT e sistemas de backend, proporcionando flexibilidade, facilidade de integração e segurança.

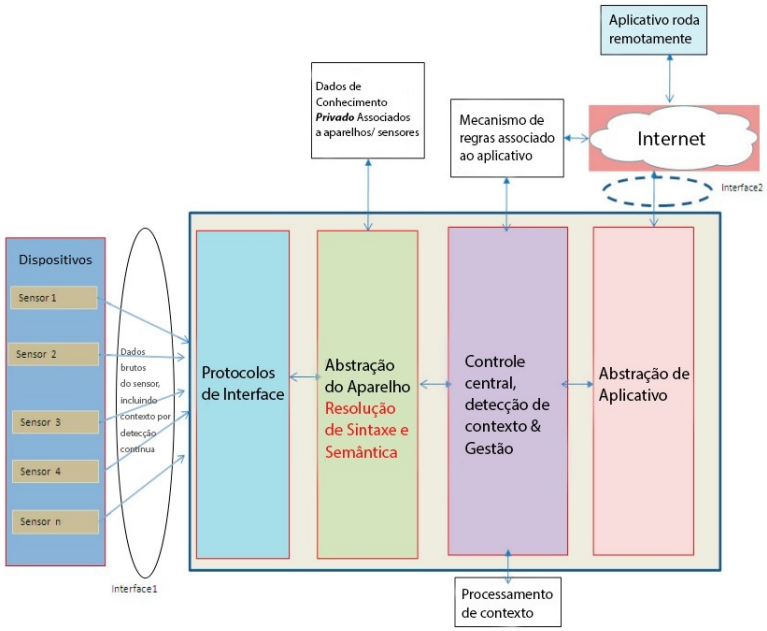
2.2.3.1 Middleware IoT

Middleware IoT é um pacote de software que atua como intermediário entre os componentes de um ecossistema de Internet das Coisas (IoT). Ele permite a comunicação entre esses elementos e integra sistemas heterogêneos, facilitando a interação entre aplicações e o sistema operacional (Soma Bandyopadhyay et al., 2011). Ele é importante porque permite a integração de sistemas heterogêneos, ou seja, a comunicação entre dispositivos e sistemas diferentes, como sensores, gateways, nuvem e aplicativos móveis. Além disso, ele também oferece segurança e escalabilidade para a solução IoT. De acordo com Soma Bandyopadhyay et al. (2011), ele pode ser divido nos seguintes componentes:

- Interoperação: permite a comunicação entre sistemas heterogêneos, permitindo que diferentes dispositivos e tecnologias sejam integrados.
- Detecção de contexto: permite a detecção do contexto, incluindo informações sobre o ambiente, a localização e o estado dos dispositivos, tornando possível o gerenciamento de dados e otimização das decisões em tempo real.
- Descoberta e gerenciamento de dispositivos: permite a descoberta e o gerenciamento de dispositivos, incluindo registro, configuração e atualização remota.
- Segurança e privacidade: oferece segurança e privacidade para a solução IoT, incluindo autenticação, criptografia e proteção de dados sensíveis.
- Gerenciando o volume de dados: permite a gestão de grandes volumes de dados, incluindo agregação, filtragem, armazenamento e análise. É importante para garantir

que os dados sejam usados de maneira eficaz e para evitar sobrecarga de rede e processamento.

A Figura 4, ilustra uma representação dessa arquitetura do Middleware IoT em camadas, as quais são: protocolos de interface, abstração de dispositivos para interoperabilidade, módulo central de gerenciamento para descoberta e gestão de dispositivos, e detecção de contexto. O módulo de abstração de aplicativos fornece a interface com aplicativos locais e remotos, sendo que os locais geralmente são *event-driven*. Outros componentes, como análise de contexto e banco de dados, podem estar em sistemas remotos, exigindo uma arquitetura distribuída.



Fonte: adaptado de Soma Bandyopadhyay et al. (2011)

Figura 4 – Arquitetura Middleware IoT

Existem diversos middlewares IoT disponíveis no mercado, alguns dos mais populares incluem o Azure IoT, o Google Cloud IoT, o AWS IoT, entre outros. Esses middlewares podem ser encontrados nos sites das respectivas plataformas e em fóruns e comunidades online de desenvolvedores IoT. É importante levar em consideração as necessidades específicas de cada projeto ao escolher o middleware certo, levando em consideração o tipo de dispositivos e tecnologias que serão usadas, as linguagens de programação e ferramentas de desenvolvimento preferidas pelo time, o nível de suporte e documentação fornecido e o custo de licenciamento e uso (Eclipse, 2022).

2.2.4 Desenvolvimento em IoT

O desenvolvimento no âmbito da Internet das Coisas (IoT) tem se tornado uma área de rápido crescimento, cuja evolução possui significativo impacto e potencial de

transformação em diversos setores. Segundo um estudo recente realizado pela Eclipse Foundation em 2022 (Eclipse, 2022), a agricultura desponta como líder na adoção dessas tecnologias, com destaque para a implementação de IoT e a computação em borda. Outros setores que seguem a mesma tendência incluem a automação industrial, o setor automotivo, a gestão de energia e o desenvolvimento de cidades inteligentes.

No que se refere à programação utilizada na IoT, uma diversidade de linguagens de programação pode ser empregada na construção de sistemas embarcados, entre elas, destacam-se C, C++, Python e Java. A seleção da linguagem de programação apropriada é uma decisão estratégica que deve considerar as características específicas da tarefa que o sistema precisa executar, bem como as restrições do hardware disponível (Eclipse, 2022).

Entretanto, o progresso nesta área de tecnologia também levanta questões e preocupações crescentes em relação à conectividade, à coleta e análise de dados e, em particular, à segurança. Como demonstrado na Figura 5, apesar da redução da porcentagem de preocupação em relação ao ano anterior, a segurança ainda é a preocupação predominante, embora ainda pouco se faça a respeito. Os desenvolvedores têm concentrado esforços cada vez mais intensos no sentido de implementar soluções que se mostrem bem-sucedidas para assegurar uma experiência do usuário cada vez melhor. Este enfoque tem contribuído para minimizar as preocupações relacionadas à complexidade da integração (Eclipse, 2022).

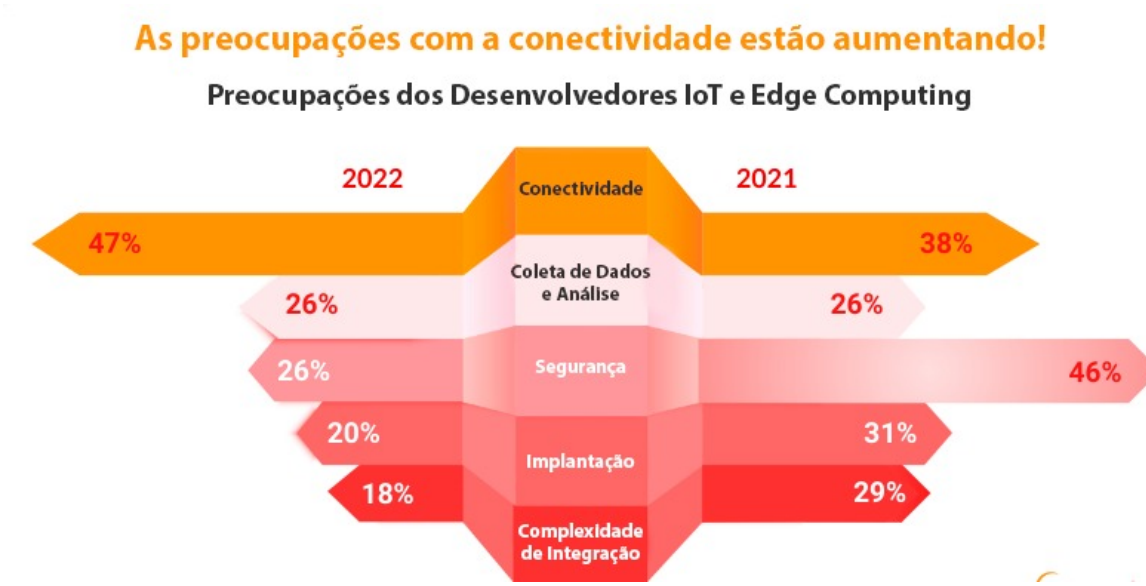


Figura 5 – Gráfico de preocupações em desenvolvimento IoT

Fonte: (Eclipse, 2022)

2.3 Conceitos de Cibersegurança

A cibersegurança, disciplina crítica na esfera da tecnologia da informação, é voltada para a proteção de sistemas, redes e dados contra ameaças digitais (STALLINGS;

BROWN, 2017). Essas ameaças, que incluem ataques de *hackers*, *malware* e *phishing*, ganharam destaque e se tornaram cada vez mais prevalentes com a popularização da Internet. Para engenheiros de software, a cibersegurança é tão fundamental quanto a fundação de uma casa: se não for sólida, pode levar a falhas estruturais (HOWARD; LEBLANC, 2002). A cultura de desenvolvimento ainda é deficiente nesse aspecto, embora práticas emergentes, como DevSecOps e TDD, estejam surgindo para auxiliar nesse cenário complexo que é o desenvolvimento de software.

Ao considerar o princípio da CIA - Confidencialidade, Integridade e Disponibilidade - que forma a base da segurança da informação, fica claro que a cibersegurança não é um complemento, mas um requisito essencial para assegurar a proteção adequada dos dados manipulados pelo software (WHITMAN; MATTORD, 2015).

No contexto da Internet das Coisas (IoT), a importância dos conceitos de cibersegurança se amplifica. Os dispositivos IoT, que abrangem desde geladeiras inteligentes até carros autônomos, estão constantemente conectados à Internet, coletando e compartilhando uma variedade de dados, sejam eles sensíveis ou não (ATZORI; IERA; MORABITO, 2010). São fatores como essa conectividade constante, aliada a outros como a falta de compromisso com a segurança do software e sistemas operacionais obsoletos, que os tornam alvos atraentes para ciberataques.

2.3.1 Conceitos Básicos

Alguns conceitos de cibersegurança que são importantes entender no contexto de um sistema IoT incluem:

- Criptografia: A criptografia é o processo de codificar dados para que eles possam ser transmitidos de forma segura. Isso é importante em um sistema IoT, pois permite que os dados sejam transmitidos e armazenados de forma segura entre os dispositivos e o servidor de backend (KONHEIM, 2007).

- Autenticação: A autenticação é o processo de verificar a identidade de um usuário ou dispositivo. Isso é importante em um sistema IoT, pois permite que as empresas verifiquem quem está acessando os dados e garantam que apenas pessoas autorizadas tenham acesso (KIZZA, 2015).

- Firewall: Um firewall, em sua essência, é um sistema de segurança que monitora e controla o tráfego de rede, permitindo ou bloqueando tráfego com base em um conjunto predefinido de regras de segurança. Ele atua como uma barreira entre redes confiáveis e não confiáveis, como a Internet. Em termos do Internet das Coisas (IoT), os firewalls são ainda mais vitais, pois protegem os dispositivos de IoT conectados de acessos não autorizados e ataques cibernéticos (TIPTON; KRAUSE, 2004).

2.3.2 Ataques de Cibersegurança

Ataques de cibersegurança são ações maliciosas realizadas por indivíduos ou grupos com o objetivo de invadir sistemas, obter informações confidenciais ou causar danos a redes e dispositivos conectados. Esses ataques são frequentemente executados por meio de vulnerabilidades em sistemas ou aplicações mal protegidos, ou por meio de técnicas de engenharia social para enganar as pessoas e obter informações sensíveis.

Alguns exemplos de ataques comuns são:

- Ataques por força bruta: Ataques por força bruta ocorrem quando um atacante tenta adivinhar a senha de um dispositivo ou sistema. Isso pode ser evitado usando senhas fortes e limitando o número de tentativas de login (KIZZA, 2015).
- Ataques de negação de serviço (DoS): Ataques DoS ocorrem quando um atacante sobrecarrega um dispositivo ou sistema para impedir o acesso a ele. Isso pode ser evitado usando medidas de proteção contra DoS (KIZZA, 2015).
- Ataques de man-in-the-middle: Ataques de espionagem ocorrem quando um atacante intercepta dados transmitidos entre dois dispositivos. Isso pode ser evitado usando criptografia forte (KIZZA, 2015).
- Ataques de engenharia social: Ataques de engenharia social ocorrem quando um atacante se aproveita da confiança ou da ignorância de um usuário para obter informações ou acesso a um dispositivo ou sistema. Isso pode ser evitado educando os usuários sobre segurança cibernética. Na definição de Hadnagy (2018):

  > "Engenharia social é qualquer ato que influencia uma pessoa a realizar uma ação que pode ou não ser de seu interesse."

2.3.3 Padrões de Segurança

Os padrões de segurança são uma forma de garantir que os sistemas de tecnologia da informação sejam seguros e protegidos contra ataques cibernéticos. A falta de padronização é um problema comum na segurança de redes de computadores. Protocolos, soluções e boas práticas de segurança podem vir em diferentes tipos e usar tecnologias diferentes, o que resulta em incompatibilidade de interfaces e uniformidade insuficiente entre muitos recursos de sistemas com tecnologias diferentes (KIZZA, 2015).

A formulação, desenvolvimento e manutenção desses padrões são responsabilidade de diferentes organizações internacionais e multinacionais, como a Internet Engineering Task Force (IETF), o Institute of Electronic and Electric Engineers (IEEE), a International Standards Organization (ISO) e a International Telecommunications Union (ITU).

Além disso, há também o European Committee for Standardization (CEN), a Commission of European Union (CEU) e o European Telecommunications Standards Institute (ETSI). A seguir, na Tabela 1, são mostrados alguns desses padrões e suas respectivas organizações reguladoras.

Tabela 1 – Tabela de Padrões

Organização	Padrões
IETF	IPSec, XML Signature XPath Filter2, X.509, Kerberos, S/MIME
ISO	ISO 7498–2:1989 Information processing systems – Open Systems Interconnection, ISO/IEC 979x, ISO/IEC 997, ISO/IEC 1011x, ISO/IEC 11xx, ISO/IEC DTR 13xxx, ISO/IEC DTR 14xxx
ITU	X.2xx, X.5xx, X.7xx, X.80x,
ECBS	TR-40x
ECMA	ECMA-13x, ECMA-20x
NIST	X3 Information Processing, X9.xx Financial, X12.xx Electronic Data Exchange
IEEE	P1363 Standard Specifications, For Public-Key Cryptography, IEEE 802.xx, IEEE P802.11 g, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications
RSA	PKCS #x – Public-Key Cryptographic Standard
W3C	XML Encryption, XML Signature, exXensible Key Management Specification (XKMS)

Fonte: (KIZZA, 2015)

2.4 Documentos e Modelos de Análise de Segurança

2.4.1 OWASP

A OWASP (*Open Web Application Security Project*) é uma comunidade global de especialistas em segurança de aplicativos que visa aumentar a conscientização sobre ameaças de segurança e fornecer recursos para ajudar a proteger aplicativos e usuários (OWASP, 2023). A OWASP Top 10 IoT 2018, Tabela 2, é uma lista dos 10 principais riscos de segurança enfrentados pelos dispositivos IoT.

Tabela 2 – OWASP Top 10 IoT 2018

Índice	Vulnerabilidade
1	Senhas Fracas, Adivinháveis ou Codificadas
2	Serviços de Rede Inseguros
3	Interfaces de Ecossistema Inseguras
4	Ausência de Mecanismo de Atualização Seguro
5	Uso de Componentes Inseguros ou Desatualizados
6	Proteção Insuficiente à Privacidade
7	Transferência e Armazenamento de Dados Inseguros
8	Falta de Gerenciamento de Dispositivos
9	Configurações Padrão Inseguras
10	Falta de Aperfeiçoamento Físico

Com base na (OWASP, 2018) e (KIZZA, 2015), na sequência são descritas as vulnerabilidades representadas na Tabela 2 com possíveis ataques e possíveis remediações:

1. Senhas Fracas, Adivinháveis ou Codificadas:

   - Possíveis ataques:
     - Uso de dicionários de senhas;
     - Ataques de força bruta;
     - Técnicas de engenharia social.
   - Possíveis remediações:
     - Fortalecer as senhas;
     - Implementar autenticação de múltiplos fatores;
     - Usar soluções de gerenciamento de senhas seguras.

   No contexto de IoT e ciclo de vida de software, as senhas fracas, adivinháveis ou codificadas são uma vulnerabilidade comum devido à necessidade de tornar os dispositivos acessíveis e fáceis de usar. No entanto, isso também torna mais fácil para os atacantes hackearem esses dispositivos. É importante fortalecer as senhas, implementar autenticação de múltiplos fatores e usar soluções de gerenciamento de senhas seguras para minimizar o risco de invasão de sistemas (OWASP, 2018). Conforme registrado em ENISA (2019):

   > "(..) em dispositivos IoT, as especificações para senhas de usuários devem ser incluídas como requisito. Por sua vez, durante a fase de projeto, isso implicaria a implementação de funções para gerenciar as senhas dos usuários conforme necessário (ciclos de mudança, tamanho mínimo da senha, símbolos especiais etc)."

2. Serviços de Rede Inseguros:

   - Possíveis ataques:
     - Uso de *exploits* conhecidos;
     - Aproveitamento de vulnerabilidades de segurança não corrigidas;
   - Remediação:
     - Desativar serviços de rede inseguros;
     - Usar soluções seguras, como SSH ou SFTP.

   No contexto de IoT e ciclo de vida de software, muitos dispositivos IoT usam serviços de rede inseguros, como FTP ou Telnet, para tornar mais fácil a conexão e administração. No entanto, isso também oferece aos atacantes uma porta de entrada para o sistema, permitindo que eles acessem dados confidenciais ou causem danos. É importante desativar serviços de rede inseguros e usar soluções seguras, como SSH ou SFTP, para proteger contra esses tipos de ataques (OWASP, 2018).

3. Interfaces de Ecossistema Inseguras:

   - Possíveis ataques:
     - Aproveitamento de vulnerabilidades de segurança;
     - Engenharia social;
     - Ataques a interfaces públicas.
   - Possíveis remediações:
     - Implementar autenticação robusta;
     - Criptografia de dados;
     - Monitorar e corrigir regularmente vulnerabilidades de segurança.

   No contexto de IoT e ciclo de vida de software, as interfaces de ecossistema, como aplicativos móveis ou gateways, são vulneráveis a ataques devido a sua exposição ao público. Essas interfaces podem ser alvos de ataques se não tiverem medidas de segurança adequadas, permitindo que os atacantes acessem dados confidenciais ou controlem dispositivos IoT. É importante implementar autenticação robusta, criptografia de dados e monitorar regularmente vulnerabilidades de segurança para garantir a segurança das interfaces de ecossistema (OWASP, 2018).

4. Interfaces de Atualização Inseguras:

   - Possíveis ataques:
     - Injeção de código malicioso durante o processo de atualização;
     - Ausência de validação de firmware;

  - Entrega não criptografada de atualizações;
  - Ausência de mecanismos de proteção contra retrocesso;
  - Ausência de notificações sobre mudanças de segurança.
- Possíveis remediações:
  - Implementar validação de firmware antes da instalação;
  - Criptografar as atualizações antes da entrega;
  - Adicionar mecanismos de proteção contra retrocesso;
  - Notificar os usuários sobre mudanças de segurança relevantes.

A ausência de um mecanismo de atualização seguro é uma vulnerabilidade comum em dispositivos IoT. O processo de atualização pode ser comprometido por código malicioso injetado durante a atualização, ausência de validação de firmware e entrega não criptografada de atualizações. Além disso, a ausência de mecanismos de proteção contra retrocesso e notificações sobre mudanças de segurança também são preocupações. Para proteger os dispositivos IoT, é importante implementar validações de firmware, criptografar as atualizações antes da entrega, adicionar mecanismos de proteção contra retrocesso e notificar os usuários sobre mudanças de segurança relevantes (OWASP, 2018).

5. Uso de Componentes Inseguros ou Desatualizados:

   - Possíveis ataques:
     - Exploração de vulnerabilidades conhecidas em componentes antigos ou desatualizados;
     - Comprometimento da cadeia de suprimentos de software ou hardware de terceiros;
     - Personalização insegura do sistema operacional.
   - Possíveis remediações:
     - Utilizar componentes atualizados e seguros;
     - Verificar a proveniência e integridade dos componentes de terceiros antes de utilizá-los;
     - Manter o sistema operacional atualizado com as últimas correções de segurança.

   Na indústria de IoT, a utilização de componentes inseguros ou desatualizados é uma vulnerabilidade comum devido à falta de atenção à segurança durante o desenvolvimento de software e hardware. Isso pode permitir a exploração de vulnerabilidades conhecidas e comprometer a integridade dos dispositivos. É importante utilizar

componentes atualizados e seguros, verificar a proveniência e integridade dos componentes de terceiros e manter o sistema operacional atualizado para minimizar esses riscos (OWASP, 2018).

6. Proteção Insuficiente à Privacidade:

   - Possíveis ataques:
     - Roubo de informações pessoais;
     - Vazamento de informações pessoais;
     - Uso inadequado ou sem permissão de informações pessoais.
   - Possíveis remediações:
     - Criptografar informações pessoais;
     - Implementar autenticação forte;
     - Restringir o acesso à informações pessoais somente às pessoas autorizadas;
     - Adotar políticas e práticas sólidas de privacidade.

   A proteção insuficiente à privacidade é uma preocupação importante no contexto de IoT e ciclo de vida de software. É fundamental proteger informações pessoais armazenadas em dispositivos ou no ecossistema, criptografando-as e implementando autenticação forte, restringindo o acesso somente às pessoas autorizadas e adotando políticas e práticas sólidas de privacidade para garantir a segurança de informações confidenciais (OWASP, 2018).

7. Transferência e Armazenamento de Dados Inseguros:

   - Possíveis ataques:
     - Roubo de dados sensíveis;
     - Vazamento de informações pessoais;
     - Modificação não autorizada de dados.
   - Possíveis remediações:
     - Implementar criptografia forte para proteger os dados em repouso, em trânsito e durante o processamento;
     - Controlar o acesso aos dados sensíveis com autorização e autenticação;
     - Adotar práticas de segurança sólidas para garantir a integridade e confidencialidade dos dados.

   A transferência e o armazenamento de dados inseguros são uma preocupação crítica na segurança cibernética. É fundamental implementar medidas de segurança para proteger os dados sensíveis, incluindo criptografia forte, controle de acesso baseado em autorização e autenticação e práticas de segurança sólidas. Sem essas medidas,

os dados podem ser roubados, vazados ou modificados sem autorização, comprometendo a privacidade e a integridade das informações (OWASP, 2018).

8. Falta de Gerenciamento de Dispositivos:

   - Possíveis ataques:
     - Invasão de sistemas;
     - Comprometimento da privacidade;
     - Roubo de dados confidenciais.
   - Possíveis remediações:
     - Implementar gerenciamento de ativos de segurança;
     - Atualizar com frequência os sistemas;
     - De-comissionar dispositivos de forma segura;
     - Monitorar os sistemas e responder rapidamente a incidentes.

   A falta de gerenciamento de dispositivos coloca em risco a segurança dos sistemas e dados confidenciais. Para minimizar o risco de invasão de sistemas, roubo de dados e comprometimento da privacidade, é importante implementar gerenciamento de ativos de segurança, atualizar com frequência os sistemas, de-comissionar dispositivos de forma segura e monitorar os sistemas para responder rapidamente a incidentes (OWASP, 2018).

9. Configurações Padrão Inseguras:

   - Possíveis ataques:
     - Exploração de configurações padrão inseguras;
     - Ausência de configurações de segurança personalizadas;
     - Dificuldade de mudar configurações de segurança.
   - Possíveis remediações:
     - Configurar dispositivos com configurações de segurança adequadas antes da implementação;
     - Permitir que os usuários configurem a segurança de acordo com suas necessidades;
     - Fornecer atualizações de segurança e configurações recomendadas para dispositivos em produção.

   Os dispositivos ou sistemas enviados com configurações padrão inseguras ou sem a capacidade de tornar o sistema mais seguro, restringindo os operadores de modificar as configurações, são uma vulnerabilidade comum na segurança da IoT. É importante configurar dispositivos com configurações de segurança adequadas antes

da implementação, permitir que os usuários configurem a segurança de acordo com suas necessidades e fornecer atualizações de segurança e configurações recomendadas para dispositivos em produção para minimizar o risco de invasão (OWASP, 2018).

10. Falta de Aperfeiçoamento Físico:

    - Possíveis ataques:
      - Ataques físicos diretos;
      - Roubo de dispositivos;
      - Espionagem e interceptação de informações sensíveis.
    - Possíveis remediações:
      - Implementar medidas de proteção física, como criptografia de dispositivos e aperfeiçoamento físico;
      - Controlar o acesso físico aos dispositivos;
      - Implementar políticas de segurança de dispositivos, incluindo criptografia de dados e autenticação de usuários.

    A falta de aperfeiçoamento físico é uma vulnerabilidade crítica, pois permite aos atacantes ter acesso físico aos dispositivos e a informações sensíveis armazenadas neles. É importante implementar medidas de proteção física, controlar o acesso físico aos dispositivos e implementar políticas de segurança de dispositivos para minimizar o risco de ataques (OWASP, 2018).

2.4.2 MITRE

A MITRE Corporation é uma organização sem fins lucrativos dedicada ao avanço da ciência e tecnologia. Suas contribuições para a segurança cibernética são significativas, gerenciando projetos como CVEs, CWEs e o modelo ATT&CK (MITRE, 2023).

Seus trabalhos fornecem diretrizes padrão para a identificação, categorização e mitigação de vulnerabilidades em softwares e sistemas, abrangendo desde softwares tradicionais até IoT.

2.4.2.1 CVEs

As *Common Vulnerabilities and Exposures* (CVEs) são identificadores únicos atribuídos a vulnerabilidades específicas em softwares e sistemas, incluindo dispositivos IoT. Esses identificadores padronizados facilitam a coordenação de esforços de defesa cibernética e permitem uma comunicação eficaz sobre ameaças potenciais (CVE, 2023).

Cada entrada CVE contém detalhes como a gravidade da vulnerabilidade, o tipo de ameaça e as ações corretivas sugeridas. Estas são amplamente divulgadas pelos for-

necedores de software e listadas em bancos de dados de segurança, como o NIST NVD (*National Vulnerability Database*) (NVD, 2023).

Por exemplo, suponha que um fabricante de dispositivos IoT descubra uma vulnerabilidade em um de seus produtos. Ele criará uma entrada CVE para essa vulnerabilidade e a disponibilizará publicamente para que os usuários saibam do risco e tomem as medidas adequadas. A página oficial[1] do CVE disponibiliza mais informações sobre as CVEs e consulta ao banco de dados.

2.4.2.2 CWE

A *Common Weakness Enumeration* (CWE) é uma iniciativa da MITRE que oferece uma linguagem comum para categorizar e descrever tipos de vulnerabilidades de segurança em softwares e sistemas, facilitando a discussão e a definição de estratégias de mitigação (CWE, 2023).

A diferenciação entre CVE e CWE reside no fato de que um CVE identifica uma vulnerabilidade específica em um produto ou sistema, enquanto uma CWE representa uma classe de vulnerabilidades de segurança, descrevendo o problema em um nível mais abstrato (SOUZA, 2022).

Por exemplo, se um tipo específico de falha de segurança for identificado em muitos produtos diferentes, isso seria classificado como uma CWE. Em contraste, uma instância específica dessa falha em um produto particular seria um CVE. A página oficial[2] do CWE apresenta informações sobre as CWEs e oferece consulta ao sistema de classificação.

2.4.2.3 ATT&CK

A estrutura ATT&CK (*Adversarial Tactics, Techniques, and Common Knowledge*) é um modelo criado pela MITRE para descrever comportamentos de adversários cibernéticos. A estrutura é um recurso inestimável para estratégias de defesa, proporcionando uma compreensão mais profunda dos métodos utilizados por atacantes (Blake E. Strom et al., 2020).

No contexto prático, por exemplo, a estrutura ATT&CK pode ser utilizada para investigar um incidente de segurança em um dispositivo IoT. Com a descrição detalhada das táticas e técnicas usadas pelo atacante, os engenheiros de segurança podem identificar como o dispositivo foi comprometido e, assim, trabalhar para mitigar a vulnerabilidade e prevenir incidentes semelhantes no futuro. A página oficial[3] da MITRE ATT&CK permite explorar detalhes e a estrutura ATT&CK em profundidade.

[1] https://cve.mitre.org/
[2] https://cwe.mitre.org/
[3] https://attack.mitre.org/

2.4.3 Modelo Diamond

O Modelo Diamond é uma metodologia estratégica para a investigação e análise de incidentes de segurança cibernética. Este modelo é estruturado em torno de quatro componentes principais: adversário, capacidade, infraestrutura e vítima (YUZUKA, 2023).

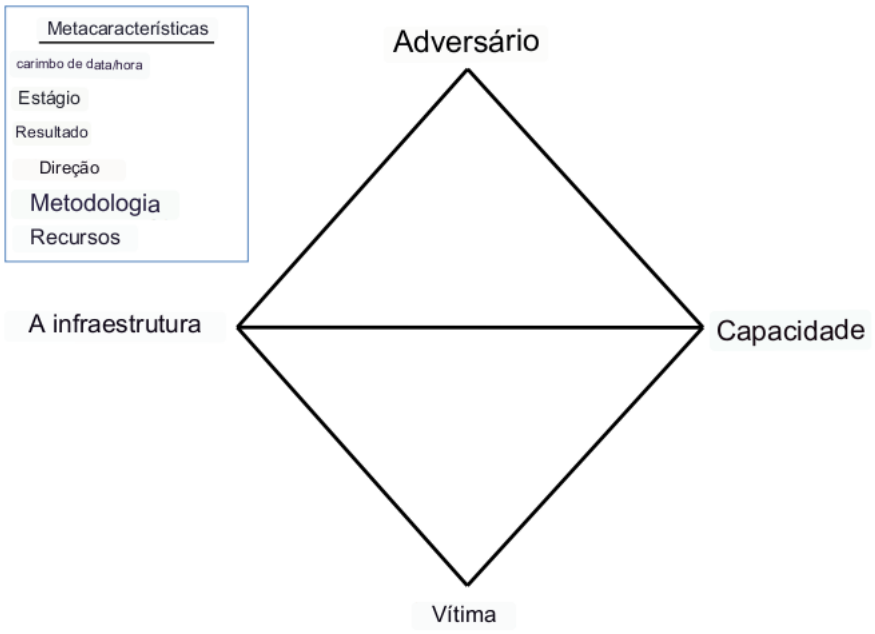


Figura 6 – Ilustração do Modelo Diamond

Fonte: Traduzido de (Sergio Caltagirone; Andrew Pendergast; Christopher Betz, 2013)

Como ilustrado na Figura 6, o Modelo Diamond permite uma avaliação sistemática de incidentes, identificando as interações entre esses quatro elementos. Por exemplo, em um cenário de Internet das Coisas (IoT), o "adversário" pode ser um ator de ameaças que explora vulnerabilidades em dispositivos IoT. A "capacidade" pode ser uma técnica de exploração de vulnerabilidade específica ou um malware. A "infraestrutura" pode ser a rede usada pelo adversário para lançar o ataque, e a "vítima" é o dispositivo IoT comprometido (YUZUKA, 2023).

A aplicação do Modelo Diamond facilita a compreensão do incidente de segurança, ajudando na elaboração de estratégias de prevenção e mitigação. Por exemplo, um software de segurança pode usar o Modelo Diamond para correlacionar dados de diferentes fontes e identificar padrões de ataque, o que pode levar a uma resposta mais rápida e eficaz a incidentes de segurança.

Frequentemente, o Modelo Diamond é usado em paralelo com outras ferramentas de análise de segurança, como os recursos da MITRE (CVEs, CWEs, ATT&CK), proporcionando uma visão abrangente do cenário de ameaças.

2.5 Conceitos de ciclo de vida de desenvolvimento de software - SDLC

No contexto da Internet das Coisas (IoT), o modelo de ciclo de vida de desenvolvimento de software (SDLC) é fundamental para garantir a segurança e a eficiência dos sistemas embarcados. Diferentes modelos SDLC foram criados ao longo dos anos para atender às necessidades específicas de diferentes projetos de software (SOMMERVILLE, 2016).

Alguns dos processos mais utilizados são:

- O modelo de cascata, por exemplo, divide o processo de desenvolvimento de software em fases distintas, como especificação de requisitos, projeto de software, implementação e teste. Este modelo é adequado para projetos grandes e bem definidos, com uma abordagem sistemática e detalhada (SOMMERVILLE, 2016).
- Desenvolvimento incremental, que intercala as atividades de especificação, desenvolvimento e validação. O sistema é desenvolvido como uma série de versões, com cada versão adicionando funcionalidades à versão anterior. Este modelo é mais adequado para projetos de software menores ou com mudanças frequentes (SOMMERVILLE, 2016).
- Modelo de integração e configuração se baseia na disponibilidade de componentes ou sistemas reutilizáveis. O processo de desenvolvimento do sistema se concentra na configuração desses componentes para uso em um novo ambiente e na integração deles em um sistema (SOMMERVILLE, 2016).

Sendo assim, não há um modelo SDLC universal que seja adequado para todos os tipos de desenvolvimento de software para IoT. O modelo certo depende dos requisitos do cliente e regulatórios, do ambiente onde o software será usado e do tipo de software sendo desenvolvido.

Mas de maneira geral todos os processos seguem uma estrutura similar a representada na Figura 7. Sendo elas (JACKSON, 2022):

1. Planejamento: Nesta etapa, é feito o planejamento do software, definindo objetivos, escopo, recursos e cronograma. É importante estabelecer claramente o que o software precisa fazer antes de começar o desenvolvimento.
2. Análise: Aqui é feita uma análise mais detalhada dos requisitos do software, garantindo que todas as necessidades do usuário estejam claramente definidas.

3. Design: Nesta etapa, é feito o design do software, incluindo a escolha da arquitetura, a definição das interfaces de usuário e a especificação de como os componentes do software se comunicarão entre si.

4. Implementação: Aqui o software é implementado, ou seja, programado. Este é o processo de transformar o design em código funcional.

5. Teste e Integração: Neste passo, o software é testado e integrado a outros sistemas, garantindo que ele esteja funcionando corretamente e de acordo com os requisitos do usuário.

6. Manutenção: Finalmente, o software é mantido, corrigindo bugs e adicionando novos recursos ao longo do tempo. É importante manter o software atualizado e funcionando corretamente para garantir a satisfação do usuário.

Figura 7 – SDLC

Fonte: (JACKSON, 2022)

2.5.1 Modelo de Desenvolvimento de Software Seguro

O desenvolvimento de software seguro envolve a segurança dos requisitos de software, segurança do design de software, segurança da construção de software e segurança de testes de software (BOUQUE; FAIRLEY, 2014). Em particular, é importante abordar a segurança em todas as fases do ciclo de vida do desenvolvimento de software (SDLC), incluindo:

- Análise de Requisitos de Segurança: neste estágio, os requisitos de segurança são identificados e avaliados para garantir que o software desenvolvido atenda aos padrões de segurança necessários.

- Design de Segurança de Software: durante o design de software, as preocupações de segurança são incorporadas ao projeto para garantir a proteção contra ameaças potenciais.
- Construção de Software Seguro: na construção do software, medidas de segurança são implementadas, tais como autenticação, criptografia e controle de acesso.
- Teste de Segurança de Software: por fim, o software é testado para verificar se ele atende aos requisitos de segurança e se é protegido contra possíveis ameaças.

Ao abordar a segurança em cada fase do SDLC, é possível garantir que o software seja desenvolvido de forma segura e protegido contra possíveis ameaças.

De acordo com Bouque e Fairley (2014):

> "Uma visão geralmente aceita sobre software segurança é que é muito melhor projetar segurança no software do que corrigi-lo depois que o software é desenvolvido."

2.5.1.1 Microsoft SDL

O Microsoft Security Development Lifecycle (SDL) é um modelo de ciclo de vida de software que introduz considerações de segurança e privacidade em todas as fases do processo de desenvolvimento. O objetivo é ajudar os desenvolvedores a construir software altamente seguro, atender aos requisitos de conformidade de segurança e reduzir os custos de desenvolvimento (Microsoft, 2023).

O Microsoft SDL é estruturado em torno de várias práticas essenciais, que incluem:

1. **Capacitação**: A segurança é um dever compartilhado. É imperativo que os desenvolvedores compreendam os princípios fundamentais da segurança e saibam como integrar a segurança no desenvolvimento de software e serviços (Microsoft, 2023).
2. **Estabelecimento de Requisitos de Segurança**: A segurança e a privacidade são elementos cruciais no desenvolvimento de aplicações e sistemas altamente seguros. Os requisitos de segurança precisam ser atualizados continuamente para acompanhar as mudanças na funcionalidade necessária e na evolução do cenário de ameaças (Microsoft, 2023).
3. **Definição de Métricas e Relatórios de Conformidade**: É vital definir os níveis mínimos aceitáveis de qualidade de segurança e responsabilizar as equipes de engenharia pelo cumprimento desses padrões (Microsoft, 2023).

4. **Modelagem de Ameaças**: A modelagem de ameaças deve ser empregada em ambientes com risco de segurança significativo. Ela permite que as equipes de desenvolvimento considerem, documentem e discutam as implicações de segurança dos projetos no contexto do ambiente operacional planejado (Microsoft, 2023).

5. **Definição de Requisitos de Design**: Para garantir a segurança, os engenheiros costumam depender de recursos de segurança, como criptografia, autenticação, registro, entre outros (Microsoft, 2023).

6. **Estabelecimento e Uso de Padrões de Criptografia**: Com a prevalência da computação móvel e em nuvem, é crucial garantir que todos os dados, incluindo informações sensíveis à segurança e dados de gerenciamento e controle, sejam protegidos contra divulgação ou alteração não intencionais durante a transmissão ou armazenamento (Microsoft, 2023).

7. **Gerenciamento do Risco de Segurança ao Usar Componentes de Terceiros**: Atualmente, a maioria dos projetos de software é construída usando componentes de terceiros (comerciais e de código aberto) (Microsoft, 2023).

8. **Uso de Ferramentas Aprovadas**: É necessário definir e publicar uma lista de ferramentas aprovadas e suas respectivas verificações de segurança associadas, como opções de compilador/linker e avisos (Microsoft, 2023).

9. **Realização de Testes de Segurança de Análise Estática (SAST)**: A análise do código-fonte antes da compilação oferece um método altamente escalonável de revisão de código de segurança e ajuda a garantir que as políticas de codificação segura estejam sendo seguidas (Microsoft, 2023).

10. **Realização de Testes de Segurança de Análise Dinâmica (DAST)**: A verificação em tempo de execução do software totalmente compilado ou empacotado verifica a funcionalidade que só é aparente quando todos os componentes estão integrados e em execução (Microsoft, 2023).

11. **Realização de Testes de Penetração**: Os testes de penetração são uma análise de segurança de um sistema de software realizada por profissionais de segurança qualificados que simulam as ações de um invasor (Microsoft, 2023).

12. **Estabelecimento de um Processo Padrão de Resposta a Incidentes**: A preparação de um Plano de Resposta a Incidentes é crucial para ajudar a lidar com novas ameaças que podem surgir ao longo do tempo (Microsoft, 2023).

2.5.1.2 Boas Práticas de Segurança para IoT da ENISA

O estudo da ENISA introduz boas práticas para a segurança do IoT, com um foco particular nas diretrizes de desenvolvimento de software para produtos e serviços IoT ao longo de sua vida útil. Ela visa, portanto, gerar recomendações e boas práticas de software ao longo de todo o SDLC conforme a figura 8, e promover uma cultura e de maneira geral um ecossistema IoT com mais reflexão para a segurança do mesmo (ENISA, 2019).

Figura 8 – SDLC - ENISA

Fonte: Traduzido (ENISA, 2019)

Conforme apresentado na Figura 8, o estudo da ENISA sugere uma abordagem de ciclo de vida de desenvolvimento de software seguro divido em:

1. **Definindo Conceitos e Requisitos:** Nesta etapa, são estabelecidas práticas, metodologias e conceitos essenciais para uma eficiente elucidação dos requisitos, sempre com foco em segurança e levando em consideração as particularidades do contexto IoT. A definição clara e segura dos requisitos é um passo fundamental para o desenvolvimento de um software seguro e funcional (ENISA, 2019).

2. **Design de Software:** Nesse estágio, são examinadas e aplicadas técnicas de design eficazes, com uma visão particular sobre a dinâmica do design dentro do contexto de IoT e segurança. Isso implica um projeto arquitetônico robusto e seguro, que

considera os desafios e demandas específicas das soluções IoT, garantindo que as preocupações de segurança estejam integradas desde o início do processo de desenvolvimento (ENISA, 2019).

3. **Desenvolvimento e Implementação:** Nesta etapa, os métodos de programação segura e a implementação dos requisitos de segurança definidos previamente são enfatizados. O desenvolvimento de software para IoT envolve a consideração de uma variedade de plataformas e tecnologias, e a segurança deve ser uma consideração primordial ao longo do processo (ENISA, 2019).

4. **Teste e Aceitação:** Aqui, a importância dos testes de segurança contínuos e abrangentes é sublinhada. Este passo envolve verificar e validar os requisitos de segurança implementados, utilizando-se de técnicas adequadas de testes de segurança para assegurar que o software funciona como esperado e que não apresenta falhas de segurança evidentes (ENISA, 2019).

5. **Manutenção:** Na fase de manutenção, é crucial implementar um processo de gestão de vulnerabilidades eficaz e garantir que o software seja atualizado regularmente para lidar com ameaças emergentes. Isso inclui a revisão e atualização periódica dos requisitos de segurança, bem como a disponibilização de atualizações de segurança para os usuários finais (ENISA, 2019).

É importante ressaltar que cada uma dessas fases deve ser considerada como parte de um processo contínuo e iterativo para manter a segurança do software IoT ao longo de seu ciclo de vida. E assim, garantir que durante todo o ciclo de vida se esta desenvolvendo um software robusto e bem planejado com o contexto de IoT em mente.

2.5.2 DevOps

DevOps é uma metodologia que promove a integração contínua entre as equipes de desenvolvimento (Dev) e operações (Ops) para melhorar a eficiência e qualidade do software. No contexto do Internet of Things (IoT), o DevOps se torna ainda mais crítico devido à complexidade e escala dos sistemas. Além disso, o DevOps é uma evolução das práticas ágeis de desenvolvimento de software, estendendo a iteração contínua e a automação para todo o ciclo de vida da entrega de software. A cultura DevOps enfatiza a comunicação contínua, colaboração e responsabilidade compartilhada entre todas as partes interessadas na entrega de software (IBM, 2023).

2.5.2.1 Segurança em DevOps

A segurança é uma consideração fundamental no DevOps, muitas vezes referida como DevSecOps, para enfatizar a necessidade de integrar práticas de segurança em todo

o ciclo de vida do desenvolvimento de software (IBM, 2023). Algumas práticas recomendadas incluem:

- **Integração contínua:** Testes de segurança são realizados automaticamente como parte do processo de integração contínua. Isso permite a detecção precoce e a correção de vulnerabilidades de segurança.
- **Entrega contínua:** As atualizações de software são lançadas de forma incremental e regular, permitindo que as correções de segurança sejam implementadas rapidamente. Isso minimiza o tempo de exposição a vulnerabilidades de segurança.
- **Infraestrutura como código:** A infraestrutura é gerenciada usando código, o que permite a auditoria e a replicação de configurações de segurança. Isso garante a consistência das configurações de segurança em todos os ambientes.
- **Shift Left Security:** A segurança é incorporada desde o início do ciclo de vida do desenvolvimento de software, quando os problemas de segurança são mais fáceis e menos caros de resolver. Isso é conhecido como "shifting left"na segurança.

2.5.2.2 Ciclo de Vida do DevOps

O ciclo de vida do DevOps consiste em seis fases: planejamento, desenvolvimento, integração, implantação, operações e aprendizado. Cada fase é iterativa e automatizada, permitindo a entrega rápida e de alta qualidade do software (IBM, 2023):

- **Planejamento:** As equipes definem novos recursos e funcionalidades para a próxima versão, com base no feedback dos usuários e em estudos de caso.
- **Desenvolvimento:** Os desenvolvedores codificam e testam novos recursos, com base nas histórias de usuários e itens de trabalho no backlog.
- **Integração:** O novo código é integrado à base de código existente, testado e preparado para implantação.
- **Implantação:** O código é implantado em um ambiente de produção, onde é monitorado e testado para garantir a qualidade, conformidade e segurança.
- **Operações:** As operações monitoram o desempenho do recurso, garantindo que ele esteja funcionando corretamente e que não haja interrupções no serviço.
- **Aprendizado:** O feedback é coletado dos usuários finais e dos clientes sobre os recursos, funcionalidades, desempenho e valor comercial para melhorar os lançamentos futuros.

2.5.2.3 DevOps no contexto do IoT

No contexto do IoT, o DevOps enfrenta desafios adicionais devido à diversidade dos ambientes de implantação e à escala dos sistemas, como evidenciado na Tabela 3. Por exemplo, um sistema de estacionamento inteligente em uma cidade inteligente pode envolver milhões de dispositivos e sensores (Vidhya V. Kumar, 2016).

Desafio	**Solução DevOps**
Diversidade de ambientes	Gestão de configuração automatizada
Escala do sistema	Implantação contínua e monitoramento
Segurança	Integração de práticas de segurança em todo o ciclo de vida

Tabela 3 – Desafios e Soluções do DevOps no contexto do IoT

Portanto, a adoção de uma cultura DevOps e o uso de ferramentas que facilitam a colaboração e a comunicação rápidas entre o desenvolvimento e as operações são essenciais para mitigar os riscos de entrega de software no IoT (Vidhya V. Kumar, 2016).

3 Metodologia

Este trabalho tem como principal objetivo apresentar um conjunto de orientações de boas práticas que possam ser aplicadas para prevenir os principais ataques em dispositivos IoT a partir da identificação e análise de ataques no cenário IoT de acordo com medidas de segurança de software no ciclo de vida do software. Para desenvolver esse estudo, o referencial metodológico adotado se insere na perspectiva da metodologia qualitativa de investigação.

A pesquisa documental é uma técnica de pesquisa qualitativa que coleta e seleciona informações a partir de documentos para entender o cenário em estudo. É semelhante à pesquisa bibliográfica, mas se diferencia na natureza das fontes, pois a pesquisa documental se concentra em fontes primárias, enquanto a pesquisa bibliográfica se concentra nas contribuições de autores sobre o tema, buscando fontes secundárias (Sá-SILVA; ALMEIDA; GUINDANI, 2009).

A Figura 9 sumariza os quatro objetivos específicos deste trabalho, associando-os aos métodos de pesquisa selecionados. As seções a seguir descrevem como os métodos serão aplicados para a realização de cada um dos objetivos.

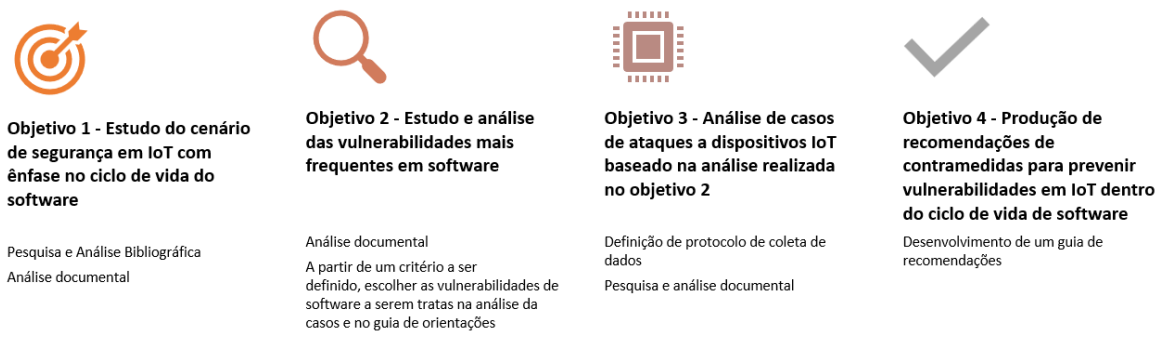


Figura 9 – Objetivos específicos

Fonte: Autor

3.1 Objetivo 1 - Estudo do cenário de segurança em IoT com ênfase no ciclo de vida do software

Compreende uma revisão de literatura para compilar informações sobre medidas de segurança aplicáveis a IoT. A análise foi orientada por uma revisão bibliográfica, identi-

ficando as principais áreas que merecem ser incluídas na fundamentação teórica, tais como dispositivos IoT, segurança da informação e esforços relevantes para proteger o ciclo de vida de software (SDLC) de IoT. A revisão de literatura foi ampla e extensiva, buscando reunir o máximo de informações possíveis sobre a implementação de SDLC seguro para IoT.

3.2 Objetivo 2 - Estudo e análise das vulnerabilidades mais frequentes em software

Neste objetivo, foi realizada uma análise do documento OWASP Top 10 IoT (OWASP, 2018) e outros como o ENISA *Good Practices for Security of IoT* (ENISA, 2019). A partir dessa revisão, foram selecionadas as vulnerabilidades que poderiam ser prevenidas ou atenuadas de maneira mais eficaz durante o Ciclo de Vida de Desenvolvimento de Software (SDLC). São elas:

- Senhas Fracas, Previsíveis ou Codificadas Diretamente;
- Serviços de Rede Inseguros;
- Interfaces de Ecossistema Inseguras;
- Falta de Mecanismo de Atualização Seguro;
- Proteção de Privacidade Insuficiente;
- Falta de Gerenciamento de Dispositivo;
- Configurações Padrão Inseguras.

3.3 Objetivo 3 - Análise de Incidentes de Segurança em Dispositivos IoT

Este objetivo consistiu na realização de uma análise aprofundada de incidentes de segurança reais envolvendo dispositivos IoT. Esta análise centrou-se especialmente nos casos que exploraram as vulnerabilidades identificadas durante o estudo no Objetivo 2, e para conduzir esta análise de maneira estruturada, foi estabelecido um protocolo de pesquisa documental. O protocolo, portanto, orienta a coleta e a interpretação dos dados, especificando o tipo de documentos a serem coletados e os critérios para sua relevância.

Os documentos selecionados para esta análise apresentam casos de ataques a dispositivos IoT que exploram uma ou mais das vulnerabilidades focadas no Objetivo 2.

Estes documentos foram selecionados devido ao seu valor informativo e à clareza com que ilustram a exploração prática das vulnerabilidades de segurança na IoT.

Cada incidente foi analisado detalhadamente, seguindo uma estrutura predefinida que contempla:

1. Introdução;
2. Contexto e Detalhes do Ataque;
3. Impacto do Ataque;
4. Soluções Pós-Ataque e Mitigação;
5. Prevenção e Boas Práticas no Ciclo de Desenvolvimento de Software;
6. Conclusão da Análise do Caso.

Processo de Coleta de Dados e Análise

Para a coleta de dados e a análise, seguimos um processo detalhado visando produzir um conjunto efetivo de recomendações para melhorar a segurança dos dispositivos IoT. O processo envolveu as seguintes etapas:

- **Fontes de Dados:** As fontes de dados foram diversas para garantir a coleta de uma gama variada de casos. Isso incluiu relatórios de incidentes de segurança publicados por organizações de segurança cibernética, estudos de caso publicados em periódicos acadêmicos, artigos jornalísticos, relatórios de empresas de segurança de TI e, se disponíveis, estudos de caso fornecidos por fabricantes de IoT.
- **Critérios de Seleção:** A seleção dos documentos foi feita com base em critérios predefinidos para garantir a relevância dos dados coletados, com os documentos devendo relatar ataques a dispositivos IoT relacionados às vulnerabilidades identificadas no Objetivo 2. Além disso, eles deviam fornecer informações suficientes sobre o ataque, como a natureza do ataque, as vulnerabilidades exploradas, as consequências do ataque e quaisquer medidas de segurança que poderiam ter prevenido o ataque.
- **Processo de Coleta:** Os documentos foram coletados através de pesquisa online, usando palavras-chave relevantes, como “ataque IoT”, “vulnerabilidade IoT”, “incidente de segurança IoT” e assim por diante, em combinação com as vulnerabilidades específicas identificadas no Objetivo 2. Além disso, os artigos acadêmicos foram coletados através de bases de dados acadêmicas relevantes, como IEEE Xplore, ACM Digital Library, Google Scholar, entre outras.

- **Análise dos Casos de Ataque:** Os casos selecionados foram analisados para extrair aprendizados valiosos. A análise foi realizada com o fim de adquirir as informações necessárias e destacar pontos relevantes, identificando também práticas de segurança de software que poderiam ter prevenido o incidente.

3.4 Objetivo 4 - Propor um conjunto de boas práticas a serem adotadas durante o ciclo de vida do software de dispositivos IoT

Com base nos resultados dos Objetivos 1, 2 e 3, foi proposto um conjunto de boas práticas a serem adotadas durante o ciclo de vida do software de dispositivos IoT. As boas práticas foram organizadas de acordo com as fases do SDLC (ENISA, 2019):

- Requisitos;
- Design;
- Desenvolvimento/Implementação;
- Testes e Aceitação;
- Implantação e Integração;
- Manutenção.

Cada prática recomendada foi devidamente justificada com base na literatura existente e nos estudos de caso analisados no Objetivo 3.

4 Análise de casos

4.1 Considerações Iniciais

Ao analisar vulnerabilidades no contexto de dispositivos de Internet das Coisas (IoT), é crucial considerar aquelas que se enquadram dentro do ciclo de vida do software, uma vez que muitas dessas falhas de segurança originam-se em alguma etapa desse ciclo. O ciclo de vida do software compreende as fases pelas quais um software passa, desde a sua concepção, passando pelo design, implementação, teste, até a manutenção e eventual desativação.

Nesse contexto, selecionamos as seguintes vulnerabilidades da lista OWASP Top 10 IoT de 2018 para uma análise mais detalhada: “Senhas Fracas, Adivinháveis ou Codificadas”, “Uso de Componentes Inseguros ou Desatualizados”, “Interfaces de Ecossistema Inseguras”, “Transferência e Armazenamento de Dados Inseguros”, “Configurações Padrão Inseguras” e “Proteção Insuficiente à Privacidade” (OWASP, 2018). Essas vulnerabilidades foram escolhidas por sua relação direta com falhas potenciais em diversas fases do ciclo de vida do software e por sua relevância particular no cenário IoT.

A vulnerabilidade “Senhas Fracas, Adivinháveis ou Codificadas” está intimamente relacionada às fases de requisitos e design do software. A presença de senhas fracas ou facilmente adivinháveis pode tornar os sistemas alvo de ataques de força bruta, enquanto senhas codificadas ou hard-coded podem ser extraídas por atacantes, comprometendo a segurança do sistema (OWASP, 2018).

O “Uso de Componentes Inseguros ou Desatualizados” representa um risco significativo na fase de implementação do software. A utilização de componentes desatualizados ou vulneráveis pode abrir brechas de segurança, expondo o sistema a ataques (OWASP, 2018).

“Interfaces de Ecossistema Inseguras” constituem uma vulnerabilidade crucial, principalmente nas fases de design e implementação do software. Interfaces inseguras podem permitir acesso não autorizado a funções críticas do sistema ou vazamento de informações sensíveis (OWASP, 2018).

A “Transferência e Armazenamento de Dados Inseguros” pode ser um ponto de falha em quase todas as fases do ciclo de vida do software. Informações sensíveis, se não protegidas adequadamente durante a transferência ou o armazenamento, podem ser interceptadas ou acessadas por atacantes (OWASP, 2018).

"Configurações Padrão Inseguras" são uma ameaça relevante, principalmente na fase de distribuição e instalação do software. Configurações inseguras podem permitir aos atacantes acessar interfaces, dados e funcionalidades restritas (OWASP, 2018).

Por fim, "Proteção Insuficiente à Privacidade". Este problema está intrinsecamente ligado a todas as fases do ciclo de vida do software, desde a concepção e design até a fase de manutenção. A privacidade dos usuários pode ser comprometida se os dados pessoais forem coletados, armazenados ou transmitidos de maneira insegura. Além disso, se os controles de acesso a esses dados não forem adequados, informações sensíveis podem cair nas mãos erradas. Portanto, uma análise rigorosa da proteção à privacidade é necessária para garantir que os dispositivos IoT estejam seguros e que os dados dos usuários estejam protegidos (OWASP, 2018).

A análise dessas vulnerabilidades fornece uma visão clara dos riscos presentes no ciclo de vida do software em dispositivos IoT, oferecendo *insights* valiosos para aprimorar as estratégias de segurança. Nas seções seguintes, exploraremos cada uma dessas vulnerabilidades em maior profundidade, apresentando 3 casos reais onde foram exploradas por atacantes.

4.2 Caso 1 - Mirai Botnet

4.2.1 Introdução

A Botnet Mirai representa um marco no mundo da segurança cibernética, demonstrando o potencial devastador dos ataques direcionados a dispositivos de Internet das Coisas (IoT). Em 2016, a Mirai conseguiu comprometer entre 200 mil a 300 mil dispositivos IoT em todo o mundo (ANTONAKAKIS et al., 2017), lançando ataques de negação de serviço distribuídos (DDoS) em larga escala que interromperam operações em vários alvos de alto perfil.

O sucesso da Mirai residiu/reside principalmente na exploração eficiente de uma vulnerabilidade comum em muitos dispositivos IoT: o uso de senhas padrão inseguras. Essa estratégia permitiu que a Mirai infectasse uma grande variedade de dispositivos, desde gravadores de disco digital (DVRs) e câmeras IP até roteadores e impressoras. Além disso, a simplicidade do comportamento do botnet permitiu que ele infectasse muitos dispositivos heterogêneos, evidenciando a necessidade de uma abordagem de segurança mais robusta no desenvolvimento de software para dispositivos IoT (ANTONAKAKIS et al., 2017).

4.2.2 Contexto e Detalhes do Ataque

Como dito anteriormente, a Mirai Botnet foi uma manifestação da falha de segurança crítica comum a muitos dispositivos IoT - a utilização de senhas padrão inseguras e configurações padrão inseguras, sendo a maior parte dos dispositivos comprometidos pela Mirai de natureza Linux-based e variavam de DVRs, câmeras IP, roteadores e até impressoras (ANTONAKAKIS et al., 2017).

O ataque começa primeiramente com um processo de varredura na rede, nesse caso seria principalmente a Internet, que busca dispositivos IoT com portas Telnet ou SSH abertas. Assim, a infecção se dá ao acessar esses dispositivos utilizando variantes de ataques de senha, com a utilização de uma lista de credenciais de login padrão ou fracas, ataques de *spraying*, *bruteforce* e outros. Quando a exploração é bem-sucedida, os atacantes rodam scripts maliciosos e adicionam esse dispositivo comprometido à botnet, tornando-o em um bot novo na rede. Posteriormente, os dispositivos infectados se conectam a um servidor de comando e controle (C2), que são usado pelos atacantes para controlar a botnet e lançar ataques DDoS, atualizar o malware ou remover a infecção do dispositivo e outros tipo de ataque (ENISA, 2019). O fluxo mostrado na Figura 10, demonstra cada passo desse ataque.

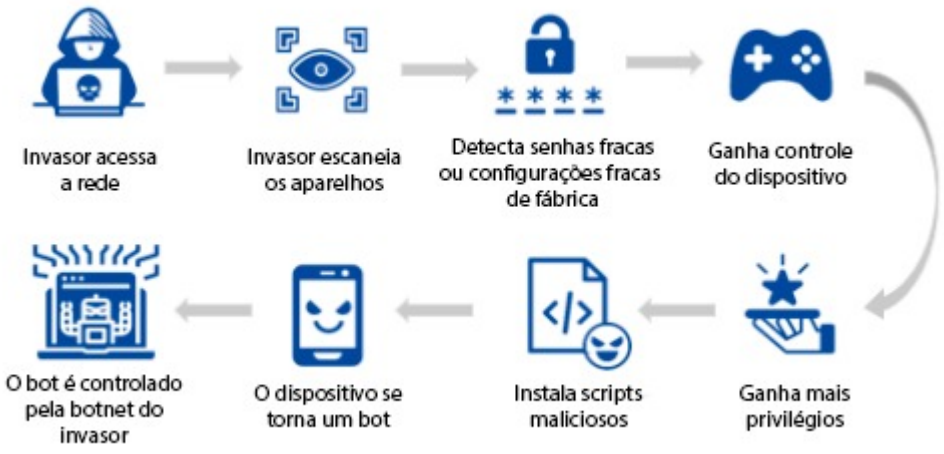


Figura 10 – Fluxo de Ataque - Credenciais Fracas/Padrões

Fonte: Adaptado de ENISA (2019)

Os ataques DDoS lançados pela Mirai foram alguns dos maiores já registrados, destacando-se o ataque ao provedor DNS Dyn em 2016, que causou interrupções na Internet em grande escala. Esses ataques consistiam no envio de uma grande quantidade de tráfego para o servidor ou serviço alvo, sobrecarregando sua capacidade e causando interrupções ou falhas.

Outro fator importante se apresenta na análise da geolocalização das infecções Mirai, que revelaram concentrações particularmente altas em regiões como Brasil, Colômbia e Vietnã, que juntas, representaram 41,5% das infecções (ANTONAKAKIS et al., 2017). Essa distribuição geográfica provavelmente se deve a tendências de mercado e decisões de design de fabricantes destas regiões.

4.2.3 Impacto do Ataque

O ataque da Mirai Botnet desencadeou uma série de impactos significativos, afetando várias esferas da sociedade e da economia.

Interrupções nos Serviços de Internet

A mais evidente consequência direta do ataque foi a interrupção de grande escala dos serviços de internet. Diversos ataques DDoS, coordenados pela Mirai, causaram falhas extensivas de rede, afetando uma série de sites e serviços online populares. Essas interrupções ocasionaram uma paralisação significativa nas operações de negócios e nas atividades cotidianas dos usuários da internet, gerando um impacto operacional e econômico considerável (ANTONAKAKIS et al., 2017).

Impacto Econômico

O impacto econômico do ataque da Mirai, apesar de difícil de ser quantificado com precisão, foi significativo. Como dito anteriormente, as empresas afetadas pelas interrupções de serviço enfrentaram perdas diretas de receita devido à inatividade e, em muitos casos, custos adicionais associados à resolução do problema e ao reforço de suas defesas contra futuros ataques. Além disso, a interrupção de serviços digitais essenciais geram efeitos em cascata em toda a economia, afetando a produtividade e a eficiência dos negócios em vários setores (CADZOW, 2019).

Desafios de Segurança de Software

O ataque da Mirai Botnet também expôs sérias questões sobre a segurança do software em dispositivos IoT. A facilidade com que um grande número de dispositivos foi comprometido, devido à utilização de senhas padrão inseguras, revelou falhas significativas nas práticas de desenvolvimento de software para IoT. Isso sublinhou a necessidade de melhores práticas de segurança no desenvolvimento de software, incluindo o uso de autenticação mais forte e a implementação de atualizações de segurança regulares.

4.2.4 Soluções Pós-Ataque e Mitigação

Os ataques da Mirai Botnet evidenciaram a necessidade de ações pós-incidente robustas e eficazes para mitigar o impacto e restaurar os sistemas comprometidos. Diversas estratégias podem e foram empregadas para lidar com as consequências desses ataques e reduzir os danos, entre elas:

- A capacidade de resposta rápida após um incidente é uma estratégia essencial de mitigação. Identificar e isolar os sistemas comprometidos e realizar análises profundas para entender o alcance e a natureza do ataque são etapas críticas no processo de recuperação. Esse processo pode requerer a substituição ou reparo de dispositivos comprometidos e a alteração de credenciais de acesso (KOLIAS et al., 2017; ANTONAKAKIS et al., 2017).

- Para neutralizar/mitigar uma botnet como a Mirai, o esforço conjunto de várias partes interessadas, como organizações de segurança, fabricantes de IoT e provedores de Internet, é essencial. Isso pode incluir rastrear e bloquear tráfego de rede suspeito e tomar medidas para limpar os dispositivos infectados (ANTONAKAKIS et al., 2017).

- Implementar um sistema eficiente de notificação é vital para informar os usuários sobre o comprometimento de seus dispositivos e orientá-los nas medidas necessárias para restaurar a segurança. Isso pode incluir questões como mudança de senhas, a realização de atualizações e a adoção de práticas seguras de uso de dispositivos (ANTONAKAKIS et al., 2017).

- A Mirai revelou a importância de uma padronização na identificação de dispositivos IoT. Assim, seria uma boa prática os fabricantes adotarem um padrão uniforme para indicar o modelo e a versão do software e hardware dos dispositivos(ANTONAKAKIS et al., 2017).

- A diversidade de sistemas operacionais nos dispositivos IoT pode complicar a segurança ainda mais. Assim, seria recomendado que os fabricantes adotassem um número limitado de sistemas operacionais para seus dispositivos IoT (ANTONAKAKIS et al., 2017).

- Os custos e o treinamento são considerações importantes na mitigação de danos pós-ataque e de futuros investimentos. Portanto, investir em tecnologia de segurança, treinamento de pessoal e manutenção de sistemas pode ser oneroso, mas é essencial para garantir uma resposta eficaz a incidentes e minimizar o impacto de ataques futuros.

4.2.5 Prevenção e Boas Práticas no Ciclo de Vida do Desenvolvimento de Software

Garantir a segurança adequada nas soluções de Internet das Coisas (IoT) é um processo contínuo que se estende por todo o ciclo de vida do desenvolvimento de software (SDLC). Neste contexto, o uso imprudente de credenciais padrão ou insuficientemente seguras representa uma ameaça significativa. Para combater essa vulnerabilidade, várias estratégias podem ser adotadas ao longo do SDLC.

1. Adoção de Políticas de Senhas Fortes e Autenticação Avançada

A experiência do ataque Mirai demonstra que a utilização de credenciais padrão ou fracas representa um risco de segurança significativo, posicionado como o principal na lista da OWASP para IoT (OWASP, 2018). Um dos mecanismos que poderiam ter minimizado o impacto do Mirai ou até mesmo prevenido totalmente é a aplicação de autenticação robusta e política de senhas fortes durante o desenvolvimento de dispositivos IoT (ENISA, 2019).

Por exemplo, o código do firmware de um dispositivo IoT (como o ESP32) poderia implementar autenticação baseada em chave pública SSH, ao invés de depender de credenciais de login padrão. Além disso, a inclusão de uma verificação de senha rigorosa no código, com critérios de senha fortes, como a exigência de um comprimento mínimo, a utilização de uma combinação de caracteres maiúsculos, minúsculos, numéricos e especiais, bem como a proibição de senhas comumente usadas ou sequências óbvias, poderia acrescentar uma camada adicional de segurança (ENISA, 2019).

No entanto, a implementação desses mecanismos de segurança no IoT tem sido lenta, tendo em vista que a utilização de novos métodos de autenticação podem gerar sobrecarga e complexidade a arquitetura, resultando em uma experiência menos amigável para o usuário e maior complexidade para os desenvolvedores. Além disso, muitos dispositivos IoT, por consistirem de sistemas embarcados, são limitados em termos de capacidade de processamento e armazenamento, o que pode tornar a implementação de autenticação avançada e verificação de senha forte um desafio. Adicionalmente, questões como interoperabilidade, usabilidade e custo também podem impedir a adoção dessas medidas.

Portanto, uma solução viável seria disponibilizar opções para uma autenticação mais segura, se desejado pelo usuário, mantendo sempre um requisito mínimo de segurança. Além disso, instruir os usuários a criar senhas seguras e memoráveis, evitando regras de complexidade desnecessárias, poderia ter diminuído a vulnerabilidade dos dispositivos IoT aos ataque de senhas explorados no caso Mirai (ENISA, 2019).

2. Exclusão de Senhas Padrão ou Hardcoded

No panorama de segurança em IoT, o uso de senhas padrão ou pré-definidas oferece canais de acesso facilmente exploráveis por invasores. O ataque Mirai é um exemplo ilustrativo desta vulnerabilidade, onde os criminosos utilizaram uma técnica de força bruta, apoiada por um dicionário de senhas comuns, para infiltrar-se nos dispositivos e incorporá-los à sua botnet (ANTONAKAKIS et al., 2017).

Senhas padrão ou pré-definidas constituem uma vulnerabilidade reconhecida que demanda estratégias eficientes para sua mitigação (OWASP, 2018), sendo uma medida de proteção solicitar aos usuários a criação de uma nova senha durante o processo inicial de configuração do dispositivo (ENISA, 2019). No entanto, mesmo com a aparente simplicidade dessa solução, ela nem sempre é implementada pelos desenvolvedores e fabricantes de IoT. Uma das razões é a preocupação com a experiência do usuário (UX). A imposição da configuração de uma senha personalizada durante a fase de instalação inicial pode adicionar um grau de complexidade que pode desencorajar os usuários. Ademais, existe o risco de esquecimento de senhas personalizadas, o que poderia resultar em solicitações adicionais de suporte ao cliente para redefinição de senha.

3. Uso de Listas de Senhas Comprometidas

A estratégia de utilizar bancos de dados online de senhas expostas ou padrão pode ser um mecanismo eficaz para prevenir a reutilização de senhas já comprometidas. Serviços como o "*Have I Been Pwned?*" permitem a verificação se uma senha proposta já foi exposta em alguma violação de dados (ENISA, 2019).

Este método proativo auxilia no fortalecimento da segurança de dispositivos IoT, uma vez que limita a utilização de senhas vulneráveis e já exploradas por invasores. No contexto do ataque Mirai, o emprego de uma estratégia como esta poderia ter diminuído o sucesso da infiltração, restringindo o uso de senhas que já haviam sido expostas e, portanto, constavam em listas de credenciais vazadas usadas para realizar ataques de senha.

No entanto, a implementação da verificação de senhas contra um banco de dados externo, apresenta alguns desafios, principalmente ao considerar certas limitações do cenário de IoT. Além das preocupações com a privacidade e segurança dos dados, deve-se preocupar com as particularidades da sua arquitetura, como a limitação de conectividade e o possível *overhead* de performance ao utilizar serviços externos.

Mas apesar dessas adversidades, existem algumas soluções como a técnica do k-anonymity para contorná-las ou ao menos diminuírem os riscos das mesmas. O k-anonimity em específico, consiste em fazer o *hash* parcial da senha antes do envio para o serviço, cuidando assim da questão da segurança e privacidade na comunicação (ALI, 2018).

4. Implementação de Autenticação Multifatorial (MFA)

A Autenticação Multifatorial (MFA) é um processo que proporciona uma camada adicional de proteção, demandando que os usuários confirmem sua identidade através de dois ou mais mecanismos de autenticação distintos. Tal processo pode combinar algo que o usuário conhece (como uma senha), algo que o usuário possui (como um smartphone), e algo que é intrínseco ao usuário (como uma impressão digital) (ENISA, 2019).

Na situação do ataque Mirai, a aplicação de MFA poderia ter dificultado consideravelmente a infecção dos dispositivos IoT, pois mesmo se um atacante obtivesse acesso às credenciais de login, sem os demais fatores de autenticação, a invasão seria bloqueada ou pelo menos dificultada, o que muitas vezes já seria o suficiente para não prosseguirem com esse dispositivo, já que muitas vezes utilizam de processos automatizados para o ataque, como no caso da Mirai (ANTONAKAKIS et al., 2017).

No entanto, a implementação de Autenticação Multifatorial (MFA) em dispositivos IoT não é algo trivial. Os desenvolvedores de IoT podem não querer implementar essa medida devido à possibilidade de hardware adicional, à maior complexidade da experiência do usuário e à necessidade de maior capacidade de processamento.

Assim, uma abordagem intermediária pode incluir o uso de métodos de autenticação multi-fator mais simples, como autenticação em duas etapas por meio de um código enviado por e-mail ou SMS, ou o uso de tokens baseados em software. A questão da maior complexidade da experiência do usuário pode ser tratada ao colocar a funcionalidade como algo extra configurável no dispositivo, os quais usuários mais preocupados e conscientes possam habilitar, não sendo algo obrigatório mas que seja incentivado de toda forma.

5. Minimização e Proteção de Serviços Expostos

A estratégia de minimizar a superfície de ataque é um princípio fundamental em segurança de software, e se mostra particularmente efetiva no contexto de dispositivos IoT. Esse método envolve a redução dos serviços disponíveis ao público e o fortalecimento dos que permanecem ativos, por exemplo, desativando protocolos de rede desnecessários ou limitando as permissões de serviço ao estritamente necessário para a funcionalidade (ENISA, 2019).

A importância desta estratégia é ilustrada no caso do ataque Mirai, onde a exposição desnecessária de serviços e portas em muitos dispositivos IoT permitiu a infecção pelo *malware*. Se uma política de minimização de serviços tivesse sido adotada, a capacidade do Mirai de comprometer esses dispositivos teria sido significativamente reduzida (KOLIAS et al., 2017).

No entanto, a implementação da minimização de serviços em dispositivos IoT não é um feito trivial. Muitas vezes, a funcionalidade e a conveniência são priorizadas em detrimento da segurança. Além disso, a falta de conhecimento sobre ameaças de segurança

pode levar desenvolvedores a deixarem serviços expostos desnecessariamente (SADEGHI; WACHSMANN; WAIDNER, 2015).

Para alcançar um equilíbrio entre funcionalidade e segurança, pode ser útil adotar políticas rigorosas de desenvolvimento seguro (ENISA, 2019). Essas podem incluir revisões regulares e minimização de serviços expostos, bem como práticas de endurecimento de segurança, como restrições de permissões de serviço (GEBREMICHAEL et al., 2020; ENISA, 2019). Com essas estratégias, é possível limitar a exposição a possíveis ataques, mantendo as funcionalidades essenciais e fortalecendo a segurança dos dispositivos IoT.

4.2.6 Conclusão da Análise Caso 1

As medidas de prevenção e as boas práticas no ciclo de vida do desenvolvimento de software são fundamentais para a segurança da IoT. A adoção de políticas de senhas fortes, a exclusão de senhas padrão *hardcoded*, o uso de listas de senhas comprometidas e a implementação de autenticação multifatorial são exemplos de estratégias eficazes que podem ser integradas ao longo do ciclo de desenvolvimento.

Os envolvidos na indústria de IoT, incluindo fabricantes de dispositivos, provedores de serviços de internet, organizações de segurança cibernética e usuários, têm um papel crucial na manutenção de um ecossistema IoT seguro. A colaboração entre essas partes interessadas é essencial para garantir a segurança dos dispositivos IoT. Por exemplo, os fabricantes podem garantir que os dispositivos sejam lançados com recursos de segurança robustos, enquanto os usuários podem garantir que seus dispositivos estejam sempre atualizados e que as senhas padrão sejam alteradas.

Ao comparar as estratégias de mitigação e prevenção, fica claro que nesse caso a prevenção é mais eficaz. Embora as ações pós-ataque tenham sido importantes para mitigar o impacto da Mirai, a implementação de práticas seguras desde o início do desenvolvimento de software pode prevenir tais ataques. Por exemplo, a implementação de autenticação robusta durante o desenvolvimento do software pode prevenir a infecção inicial, enquanto a mitigação, como a alteração de senhas após um ataque, pode não ser suficiente para evitar a reinfeção.

Importante frisar que, no contexto do Mirai, essas vulnerabilidades foram exploradas para criar uma BotNet - uma rede de dispositivos controlados por cibercriminosos. No entanto, é essencial notar que muitos desses dispositivos vulneráveis não são apenas usados para tal finalidade. Frequentemente, eles servem como vetor inicial ou elo mais fraco da rede para comprometer serviços mais vitais, como servidores. Portanto, a mitigação de tais vulnerabilidades e a promoção de uma cultura de segurança cibernética robusta são imperativos para garantir a integridade das nossas redes e sistemas.

4.3 Caso 2 - Boneca Cayla

4.3.1 Introdução

No cenário da segurança cibernética, o caso da boneca "My Friend Cayla" representa um alerta importante, evidenciando os riscos de segurança relacionados aos dispositivos IoT voltados para crianças. Em 2015, foi revelado que a boneca Cayla, que se conectava a um aplicativo de smartphone por meio de uma conexão Bluetooth, possuía falhas graves de segurança que permitiam conexões Bluetooth não autorizadas sem autenticação (BBC, 2017).

O perigo da Cayla residia principalmente na falta de segurança nos serviços de rede (OWASP, 2018), tornando o brinquedo um alvo fácil para hackers. Essa vulnerabilidade permitia que qualquer dispositivo dentro do alcance do Bluetooth se conectasse à boneca e tivesse acesso aos seus recursos de áudio. Isso não apenas possibilitava a vigilância não autorizada das interações entre crianças e a boneca, mas também permitia a comunicação direta do atacante com a criança através do sistema de alto-falantes da boneca (Tim Medin, 2015).

4.3.2 Contexto e Detalhes do Ataque

A boneca "My Friend Cayla" representa um claro exemplo da vulnerabilidade de Serviços de Rede Inseguros, que está classificada como a segunda maior vulnerabilidade crítica no IoT de acordo com a lista OWASP 2018 (OWASP, 2018). Este tipo de vulnerabilidade ocorre quando serviços de rede desnecessários ou inseguros estão em execução no dispositivo, especialmente aqueles expostos à internet, comprometendo a confidencialidade, integridade ou disponibilidade de informações ou permitindo controle remoto não autorizado (OWASP, 2018).

Mais especificamente, a vulnerabilidade da Cayla residia especificamente na implementação inadequada de serviços de rede - a conexão Bluetooth, a qual estava sempre disponível quando a boneca estava ligada e a mesma não exigia qualquer autenticação ou confirmação para o pareamento (Tim Medin, 2015). Em outras palavras, qualquer dispositivo dentro do alcance do Bluetooth poderia se conectar à boneca, o que levantou preocupações significativas, especialmente quando consideramos cenários mais densos, como complexos de apartamentos, onde vários dispositivos poderiam estar dentro do alcance da boneca, aumentando ainda mais a exposição à potenciais ameaças (Tim Medin, 2015).

Assim, a exploração dessa vulnerabilidade ocorria através de um processo bastante simples de detecção e conexão. Os atacantes podiam identificar a boneca Cayla por meio do seu sinal Bluetooth e se conectar a ela utilizando o aplicativo, conforme demonstrado em um fluxo geral na Figura 11. E uma vez estabelecida a conexão, eles tinham acesso

aos recursos de áudio da boneca, possibilitando a vigilância não autorizada e até mesmo a comunicação direta com a criança através do sistema de alto-falantes da boneca (BBC, 2017).

Figura 11 – Fluxo de Ataque - Serviços de Rede Inseguros

Fonte: Adaptado de ENISA (2019)

Essa exposição constante e a falta de autenticação ilustram o perigo dos serviços de rede inseguros, proporcionando uma superfície de ataque ampla para os invasores explorarem.

4.3.3 Impacto do Ataque

O caso da boneca "My Friend Cayla" causou uma série de impactos significativos, levantando questões importantes sobre a segurança e a privacidade na era da Internet das Coisas.

Violação da Privacidade e Segurança

O impacto mais direto e perturbador deste caso foi a violação da privacidade e a ameaça à segurança das crianças. As conversas captadas pela boneca Cayla poderiam conter informações sensíveis das crianças com as quais ela interagia (OAKLEY, 2015). E ainda mais alarmante, qualquer sistema com Bluetooth dentro de um alcance de cerca de dez metros poderia se conectar ao dispositivo e usá-lo como um alto-falante ou como um microfone remoto, permitindo aos atacantes ouvir e falar diretamente com as crianças através da boneca, abrindo a possibilidade de conduta imprópria ou manipulação (BBC, 2017).

Impacto Legal e na Indústria de Brinquedos

A indústria de brinquedos e as leis que a governam foram fortemente impactadas por este caso. A boneca Cayla, após testes e análises legais realizadas por Stefan Hessel, da Universidade de Saarland, foi classificada como um "transmissor proibido" de acordo com o § 90 da Lei de Telecomunicações Alemã (TKG) (Markus Reuter, 2017), uma classificação posteriormente confirmada pela Agência Federal de Rede da Alemanha (Bundesnetzagentur, 2023). O que resultou na retirada da boneca do mercado alemão e a exigência de que os pais destruíssem quaisquer bonecas Cayla que possuíssem, sob pena de até dois anos de prisão (Markus Reuter, 2017).

Implicações para o Desenvolvimento de Software Seguro

Este caso também ressaltou a necessidade de melhores práticas de segurança no desenvolvimento de software. A facilidade com que a boneca Cayla foi comprometida e a fragilidade da arquitetura do sistema reforçou a importância de implementar práticas de "segurança por design", incluindo a implementação de medidas de segurança desde o início do processo de desenvolvimento do software (OAKLEY, 2015).

4.3.4 Soluções Pós-Ataque e Mitigação

O incidente com a boneca "My Friend Cayla" sublinhou a necessidade urgente de práticas de mitigação eficazes e soluções robustas pós-incidente. As ações tomadas em resposta ao incidente visaram prevenir possíveis danos futuros e restaurar a confiança no ecossistema de brinquedos conectados à IoT.

- **Remoção do Mercado e Ações Regulatórias:** Em resposta à exposição da vulnerabilidade, a boneca "My Friend Cayla" foi retirada de muitos mercados. Na Alemanha, o órgão regulador de telecomunicações instruiu os pais a destruir quaisquer bonecas Cayla que possuíam, classificando-a como um "dispositivo de espionagem oculto" (Markus Reuter, 2017). Com isso, esse exemplo serviu para intensificar a supervisão regulatória sobre a segurança e privacidade dos brinquedos conectados à IoT (BBC, 2017).
- **Aprimoramento da Segurança de Software:** Fabricantes foram incentivados a implementar autenticação mais forte, aplicar regularmente atualizações de segurança e considerar a segurança desde o início do processo de desenvolvimento. Especificamente, uma solução sugerida foi a adição de um número ou senha que os proprietários precisariam inserir para se conectar via Bluetooth, prevenindo assim emparelhamentos indesejados (OAKLEY, 2015).
- **Conscientização e Educação:** A importância da conscientização e da educação dos usuários foi destacada, com os consumidores sendo notificados sobre o incidente

e aconselhados a alterar suas senhas e também a manter seus dispositivos atualizados, conforme melhores práticas de segurança. Isso incluiu o conselho para os pais desligarem a boneca Cayla quando não estivesse em uso e a protegerem seus telefones ou outros dispositivos com um PIN (OAKLEY, 2015). Além disso, também houve um esforço para esclarecer os consumidores sobre os riscos associados aos brinquedos conectados à IoT e como identificar produtos que respeitam a privacidade e a segurança dos dados (BBC, 2017).

4.3.5 Prevenção e Boas Práticas no Ciclo de Vida do Desenvolvimento de Software

A situação da boneca "My Friend Cayla" mostrou como o mau uso de práticas de privacidade e segurança pode resultar em brechas sérias. Para combater isso, várias estratégias podem ser implementadas ao longo do SDLC.

1. Desenvolvimento Seguro e Avaliação de Risco

As práticas de desenvolvimento seguro envolvem a consideração da segurança em todas as fases do SDLC, desde o projeto e implementação até a manutenção. Essas práticas podem incluir a revisão regular de código, o uso de ferramentas de análise de segurança, treinamento de segurança para desenvolvedores e a incorporação de avaliações de risco durante o processo de desenvolvimento (ENISA, 2019). No caso de "My Friend Cayla", uma avaliação de risco eficaz poderia ter identificado a vulnerabilidade da privacidade no design do brinquedo.

2. Autenticação Robusta

A implementação de autenticação robusta é essencial para proteger contra o acesso não autorizado. Isso pode incluir a exigência de senhas fortes, a autenticação de dois fatores, ou a autenticação baseada em biometria. No caso de "My Friend Cayla", a autenticação era insuficiente, permitindo a qualquer pessoa com um dispositivo Bluetooth nas proximidades se conectar à boneca (Tim Medin, 2015).

3. Atualizações de Segurança

A capacidade de atualizar o software do dispositivo é essencial para garantir a segurança a longo prazo. As atualizações permitem que os fabricantes corrijam essas vulnerabilidades à medida que são descobertas e evitando mais complicações (OWASP, 2018).

4. Minimização de Dados

A minimização de dados envolve a coleta e o armazenamento apenas dos dados necessários para a funcionalidade do dispositivo. Além disso, os dados devem ser armazenados apenas pelo tempo necessário. No caso de "My Friend Cayla", a boneca estava

coletando e transmitindo conversas que não eram necessárias para a sua funcionalidade (Forbrukerradet, 2016).

Implementar estas práticas em todo o SDLC pode ajudar a garantir a segurança e a privacidade dos usuários de brinquedos conectados à IoT. No entanto, também é importante que os reguladores estabeleçam e apliquem normas para proteger os usuários mais vulneráveis, como as crianças.

4.3.6 Conclusão da Análise Caso 2

A análise do caso da boneca "My Friend Cayla" realça a necessidade de priorizar a segurança durante todas as etapas do desenvolvimento de produtos de Internet das Coisas (IoT). As falhas de segurança observadas neste exemplo, como a falta de autenticação e criptografia adequadas, demonstram as consequências significativas de ignorar esses aspectos cruciais.

Como mencionado, existem diversas estratégias preventivas eficazes que poderiam ter sido implementadas, como a autenticação robusta e o uso de criptografia para proteger os dados transmitidos. Além disso, a capacidade de atualizar o software do dispositivo é crucial para lidar com novas ameaças à segurança à medida que elas surgem, considerando que é impossível fazer um software 100% seguro.

Sendo assim, os vários atores envolvidos na indústria de IoT têm papéis vitais na manutenção de um ecossistema seguro. Fabricantes devem garantir que os dispositivos sejam lançados com recursos de segurança robustos, da mesma maneira que os usuários, por sua vez, têm a responsabilidade de manter seus dispositivos atualizados e de substituir senhas padrão.

4.4 Caso 3 - Hello Barbie

4.4.1 Introdução

No caso 3, investigaremos as vulnerabilidades de segurança descobertas na boneca Hello Barbie, um outro brinquedo interativo com conexão à Internet direcionado a crianças. Dessa vez colocando o foco em outras vulnerabilidades da OWASP como Interfaces de Ecossistemas inseguros e proteção insuficiente à privacidade.

A Hello Barbie, fruto de uma parceria entre a Mattel e a ToyTalk, utiliza tecnologia de reconhecimento de voz para interagir com os usuários, respondendo suas falas com diálogos pré-gravados (Forbrukerradet, 2016). Essa interatividade é viabilizada por meio de servidores remotos encarregados de processar e responder às solicitações do brinquedo (Somerset Recon, 2015). No entanto, pesquisadores identificaram 14 vulnerabilidades distintas no produto, a maioria delas associada aos servidores web utilizados para

a comunicação do brinquedo, evidenciando sérias preocupações de segurança (BRAGA, 2016).

4.4.2 Contexto e Detalhes do Ataque

A Hello Barbie, uma boneca interativa produzida pela Mattel em parceria com a ToyTalk, apresenta um exemplo ilustrativo da vulnerabilidade referente às Interfaces de Ecossistema Inseguras, posicionada como uma preocupação crítica no IoT pela lista OWASP 2018 (OWASP, 2018). Essa vulnerabilidade é intensificada por dependências de software inseguras em serviços de nuvem, um desafio de segurança também identificado pela ENISA (ENISA, 2019).

Na arquitetura de software da Hello Barbie, as interfaces inseguras do ecossistema e as dependências de software inseguras residiam principalmente nos servidores remotos da ToyTalk, que processavam e respondiam às solicitações do brinquedo (Forbrukerradet, 2016; Somerset Recon, 2016). O descuido com a segurança dessas dependências permitiu que atacantes estudassem as interfaces expostas para essa solução de IoT, identificassem dependências de software e versões e explorassem vulnerabilidades conhecidas (Somerset Recon, 2016).

Diferentemente de outros brinquedos, como a boneca Cayla, a Hello Barbie se conecta diretamente à Internet através de sua própria conexão Wi-Fi, necessitando do aplicativo complementar apenas para o primeiro pareamento e registro da conta parental (Forbrukerradet, 2016). Tal abordagem, embora conveniente para o usuário, amplia a superfície de ataque, tornando a boneca suscetível a uma série de ameaças de segurança.

Durante as interações, a Hello Barbie transmitia dados, incluindo gravações de voz, para servidores da ToyTalk localizados em São Francisco (Forbrukerradet, 2016). Apesar dessa conexão ser criptografada com SSL, a vulnerabilidade reside no fato de que, uma vez que um invasor obtém acesso a essa comunicação, ele pode extrair informações sensíveis ou até mesmo alterar as respostas da boneca.

Portanto, este caso destaca a necessidade de uma análise de segurança abrangente para todos os aspectos do ecossistema de um dispositivo IoT, especialmente ao lidar com dependências de software externas. A Figura 12 ilustra como um atacante poderia explorar essas vulnerabilidades para obter acesso não autorizado à conta de um usuário, destacando o perigo potencial dessas dependências de software inseguras.

Figura 12 – Fluxo de Ataque - Dependências de Software Inseguras

Fonte: Adaptado de ENISA (2019)

4.4.3 Impacto do Ataque

O ataque à Hello Barbie representou um marco importante para os brinquedos conectados à Internet das Coisas (IoT), ressaltando questões cruciais de privacidade e segurança.

Violação da Privacidade e Segurança

As vulnerabilidades descobertas no brinquedo ofereceram a possibilidade de invasores interceptarem e até mesmo manipularem as comunicações entre o brinquedo e a nuvem. Isso apresentou um risco direto à privacidade e à segurança dos usuários, principalmente das crianças, já que suas conversas e interações com o brinquedo poderiam ser facilmente acessadas e exploradas (BRAGA, 2016).

Impacto na Confiança dos Consumidores e na Indústria

As falhas de segurança do brinquedo Hello Barbie afetaram a confiança dos consumidores nos brinquedos conectados à IoT. O incidente também ressaltou a necessidade de regulamentações mais rigorosas e práticas de segurança mais robustas na indústria de brinquedos (Laura Hautala, 2015).

Impacto nas Práticas de Desenvolvimento de Software

As falhas descobertas no brinquedo Hello Barbie enfatizaram a necessidade de melhores práticas de desenvolvimento de software. As empresas de brinquedos, assim como as de outros dispositivos IoT, foram instadas a melhorar suas práticas de segurança, aplicar autenticação mais forte e considerar a segurança desde o início do processo de desenvolvimento (Somerset Recon, 2016).

4.4.4 Soluções Pós-Ataque e Mitigação

Após a descoberta das vulnerabilidades, diversas medidas foram tomadas para corrigir as falhas e evitar incidentes semelhantes no futuro.

- **Correção de Vulnerabilidades e Aprimoramento da Segurança de Software:** A empresa ToyTalk, responsável pelo software do Hello Barbie, corrigiu algumas das vulnerabilidades descobertas e afirmou estar trabalhando para solucionar as demais. Além disso, a empresa foi incentivada a fortalecer suas práticas de segurança e a adotar um processo de desenvolvimento que priorizasse a segurança desde o início (Laura Hautala, 2015).
- **Programa de Bug Bounty:** A ToyTalk lançou um programa de Bug Bounty através da plataforma HackerOne, oferecendo recompensas monetárias para pesquisadores que reportassem vulnerabilidades adicionais. No entanto, pesquisadores apontaram que o programa pode ter sido uma alternativa a uma auditoria de segurança adequada antes do lançamento do Hello Barbie, o que não consiste em uma boa prática (BRAGA, 2016).
- **Conscientização e Educação dos Usuários:** O incidente também ressaltou a importância da conscientização e educação dos usuários em relação à segurança. Os pais foram aconselhados a usar uma senha forte para proteger a conta da Hello Barbie, a utilizar apenas redes sem fio confiáveis e protegidas com uma senha forte, e a compreender os riscos de armazenar informações pessoais em um servidor remoto (BRAGA, 2016).

Apesar dessas medidas, ainda persistem desafios. A natureza sempre em evolução da cibersegurança significa que não existe uma solução única para todos os problemas de segurança. Continua a ser essencial que as empresas desenvolvam um forte compromisso com a segurança em suas práticas de desenvolvimento, além de manterem-se atualizadas sobre as ameaças mais recentes.

4.4.5 Prevenção e Boas Práticas no Ciclo de Vida do Desenvolvimento de Software

A segurança dos brinquedos conectados à Internet das Coisas (IoT) precisa ser parte integrante do processo de desenvolvimento de software, desde a concepção do projeto até a sua manutenção pós-produção. No caso do ataque à Hello Barbie, diversas falhas de segurança foram identificadas, evidenciando a necessidade de adoção de práticas robustas de segurança no ciclo de vida do desenvolvimento de software (SDLC).

1. Segurança por Design

Desde a concepção do projeto, é essencial garantir que a segurança seja uma consideração fundamental. No caso da Hello Barbie, a empresa já implementou algumas medidas de segurança, como a criptografia de dados (Forbrukerradet, 2016), no entanto, falhou em não proteger adequadamente seus serviços web, que acabaram sendo a principal fonte de vulnerabilidade (BRAGA, 2016). Assim, é essencial a implementação de uma revisão de segurança abrangente durante a fase de projeto o que poderia ter identificado e corrigido essas vulnerabilidades antes que o produto chegasse ao mercado.

2. Autenticação Robusta e Gestão de Credenciais

A implementação de um sistema robusto de autenticação e a gestão adequada das credenciais dos usuários são aspectos cruciais da segurança. Um dos ataques à Hello Barbie explorou a possibilidade de realizar tentativas ilimitadas de adivinhação de senhas. Assim, o estabelecimento de limites para tentativas de login e a implementação de requisitos de complexidade para senhas poderiam ter prevenido essa vulnerabilidade (BRAGA, 2016).

3. Manutenção e Atualizações de Segurança

O software e o firmware dos dispositivos precisam ser regularmente atualizados e mantidos para corrigir quaisquer vulnerabilidades conhecidas e melhorar a segurança. No caso da Hello Barbie, embora algumas das vulnerabilidades tenham sido corrigidas após a detecção, a falta de uma estratégia de atualização contínua deixou a boneca exposta a possíveis futuros ataques (BRAGA, 2016).

4. Práticas de Segurança em Serviços Web

As práticas de segurança também precisam ser aplicadas aos serviços web associados ao dispositivo. No caso da Hello Barbie, a maior parte das vulnerabilidades estava associada aos serviços web da ToyTalk, e não ao próprio hardware do brinquedo (Somerset Recon, 2016). Com isso as empresas devem garantir que seus serviços web e outras possíveis dependências de terceiros como APIs e bibliotecas sejam adequadamente protegidos e verificados, possivelmente através da contratação de equipes de segurança profissionais para realizar auditorias de segurança regulares ou até mesmo ferramentas disponíveis.

A implementação dessas práticas no SDLC pode melhorar significativamente a segurança dos brinquedos conectados à IoT. Contudo, também é crucial que as empresas de brinquedos e tecnologia estejam cientes das regulamentações em vigor e trabalhem proativamente para garantir a segurança e a privacidade de seus usuários (Forbrukerradet, 2016).

4.4.6 Conclusão da Análise Caso 3

A análise do caso da "Hello Barbie" ilumina a crucialidade de medidas de segurança efetivas em todo o espectro da Internet das Coisas (IoT), não apenas no hardware do dispositivo, mas também nos serviços da web associados a ele. Este caso ressalta a necessidade de uma abordagem de segurança em camadas, onde a segurança do dispositivo IoT em si e a dos serviços da web são igualmente vitais.

O exemplo da "Hello Barbie" demonstra que embora os dispositivos de IoT possam ter suas próprias medidas de segurança robustas, como a criptografia de senhas, ainda podem ser vulneráveis se os serviços web associados não forem adequadamente protegidos. No caso em questão, as vulnerabilidades identificadas no servidor web e na aplicação que conectam a boneca à internet permitiram ataques bem-sucedidos, apesar das medidas de segurança implementadas no dispositivo (Somerset Recon, 2016).

Portanto, fica evidente a importância de não apenas desenvolver dispositivos IoT com segurança robusta, mas também garantir que todos os componentos associados sejam igualmente seguros. Além disso, a disponibilidade de um programa de recompensas por bugs, como o implementado pela ToyTalk, não deve substituir uma análise de segurança pré-produção abrangente.

5 Guia de recomendações

5.1 Introdução

5.1.1 Contexto da segurança em IoT

A Internet das Coisas (IoT) permeia muitos aspectos da nossa vida diária e do nosso ambiente de trabalho. Desde dispositivos médicos a sensores ambientais, carros inteligentes a eletrodomésticos, a IoT está integrada em quase todos os setores da economia como representado na Figura 13. No entanto, junto com os inúmeros benefícios e conveniências que esses dispositivos conectados trazem, também surgem sérias preocupações com a segurança, dado que muitos desses dispositivos são acessíveis via internet e podem coletar, processar e armazenar dados sensíveis, tornando-os em alvos atrativos para atacantes (ENISA, 2019; YOUSEFNEZHAD et al., 2023).

Figura 13 – IoT esquema

Fonte: Adaptado de Yousefnezhad et al. (2023)

5.1.2 Descrição do propósito e como usar o guia

Esse guia foi montado com o objetivo principal de proporcionar uma compreensão clara e prática das melhores práticas de segurança no contexto do desenvolvimento de software para a Internet das Coisas, para que o mesmo possa vir a ser uma referência útil para desenvolvedores, gerentes de projetos, engenheiros de segurança e qualquer profissional envolvido no desenvolvimento, operação e manutenção de dispositivos IoT.

Para usar este guia, é aconselhável seguir os tópicos em ordem, visto que cada seção apresenta uma fase do ciclo de vida de desenvolvimento de software (SDLC) e um depende do outro. As fases do ciclo de vida de software foram determinadas conforme o documento da *Enisa de 2019: Good Practices for Security of IoT* (ENISA, 2019) e o documento da *Software Assurance Forum for Excellence in Code (SAFECode)*: SAFECode Fundamental Practices for Secure Software Development, colocando ênfase na importância da segurança em cada etapa, assim como práticas recomendadas e considerações específicas, como a lista da OWASP Top 10 IoT.

5.2 Compreendendo as vulnerabilidades de segurança

5.2.1 Breve revisão do OWASP Top 10 IoT

A OWASP, uma respeitada organização internacional dedicada à melhoria da segurança de software (OWASP, 2023), publica regularmente a lista“OWASP Top 10 IoT", um recurso indispensável que destaca as dez vulnerabilidades de segurança mais críticas em dispositivos IoT (OWASP, 2018). Esta lista oferece aos profissionais uma visão valiosa dos riscos de segurança mais comuns que os dispositivos IoT enfrentam e oferece uma base para o desenvolvimento de estratégias robustas de mitigação de riscos (ver lista completa).

5.2.2 Discussão de casos de ataques famosos: Mirai, Cayla e Hello Barbie

A complexidade da segurança em IoT é melhor entendida através do estudo de alguns exemplos de ataques. Um bastante notável é o ataque Mirai, que utilizou uma botnet de dispositivos IoT comprometidos para executar um dos maiores ataques DDoS da história (ANTONAKAKIS et al., 2017). Nesse caso, a principal vulnerabilidade explorada foi surpreendentemente simples: muitos dispositivos IoT estavam usando nomes de usuários e senhas padrão, tornando-os alvos fáceis para os invasores, o que destaca a falta de considerações de segurança básicas, como a utilização de senhas padrões ou fracas, no desenvolvimento de software para dispositivos IoT.

Outros casos relevantes são os das “Bonecas Inteligentes”, envolvendo a boneca “My Friend Cayla” e a “Hello Barbie”, que destacam outras várias vulnerabilidades básicas no ecossistema IoT, com a mais impactante provavelmente sendo as implicações de

privacidade associadas a esses brinquedos que agora estão conectados à Internet (BBC, 2017; Somerset Recon, 2016). No caso da boneca Cayla, a vulnerabilidade explorada foi que a conexão Bluetooth não exigia autenticação, permitindo que qualquer dispositivo pareado pudesse se conectar e interagir com a boneca. No caso da Hello Barbie, a boneca utilizava serviços web para processar a fala, e várias vulnerabilidades foram encontradas nesses serviços, incluindo a falta de validações de entrada e a possibilidade de falsificar tokens de sessão (BRAGA, 2016).

Mostramos a importância da segurança em cada fase do ciclo de vida do software nos dispositivos IoT analisando brevemente esses exemplos de ataques. A fase de concepção é fundamental porque pode estabelecer o caminho para práticas seguras ou permitir práticas inseguras. As decisões tomadas durante a implementação podem aumentar a resistência do dispositivo a ataques ou, inversamente, deixá-lo vulnerável. A maneira como o dispositivo é usado pode proteger os dados do usuário ou, em vez disso, revelá-los. Por fim, o processo de descarte do dispositivo pode ter consequências significativas para a segurança dos dados do usuário. Como resultado, a segurança deve ser priorizada desde o início.

5.3 Modelos SDLC e a Incorporação de Práticas de Segurança em IoT

Os modelos de Ciclo de Vida de Desenvolvimento de Software (SDLC) funcionam como esquemas conceituais que mapeiam todas as etapas relacionadas ao desenvolvimento de software e as conexões entre essas etapas (ENISA, 2019). Eles são uma ferramenta vital para garantir um processo de desenvolvimento de software organizado, coordenado e eficientemente comunicado.

Vários modelos de SDLC foram propostos ao longo do tempo, cada um com suas próprias maneiras de transitar e interagir entre fases. E uma das diferenças cruciais entre esses modelos é como eles incorporam questões de segurança em suas respectivas fases, o que é de extrema relevância para o nosso contexto.

5.3.1 Modelo Cascata

O modelo Cascata, ou Waterfall do inglês, representa as fases do SDLC de maneira sequencial e linear (ver Figura 14). Nesse modelo, os requisitos são definidos no início do processo e os testes de segurança são realizados apenas nas fases finais, como Desenvolvimento/Implementação ou Integração e Testes. Esse modelo é tipicamente usado para projetos que são bem definidos e não mudam muito ao longo do tempo, com isso, ele tem sido criticado por sua falta de flexibilidade e por não estar bem adaptado ao ritmo atual

de mudanças como cloud, microserviços, IoT e outras tecnologias e arquiteturas que requerem maior nível de elasticidade e adaptabilidade (ENISA, 2019). Para uma discussão mais aprofundada sobre os desafios de segurança no modelo Cascata pode ser encontrada no documento NIST SP 800-64.

5.3.2 Modelo Ágil

O modelo Ágil, por outro lado, é projetado para ser mais flexível e adaptável. Ele faz uso de ciclos de desenvolvimento curtos e limitados no tempo, o que facilitam a adaptabilidade do produto ou serviço de software final, sendo que a principal preocupação com a segurança neste modelo é garantir a consistência dos requisitos de segurança, do design e dos testes em cada iteração. Algumas ferramentas como o OWASP SAMM podem ajudar as organizações a integrar práticas de segurança no desenvolvimento Ágil (ENISA, 2019).

5.3.3 Modelo Espiral

O modelo Espiral é uma extensão iterativa do modelo Cascata, combinando aspectos do design e prototipagem em estágios, com o rigor do modelo Cascata (ENISA, 2019). Diferentemente do Cascata e assimilando mais ao ágil, o modelo espiral permite a inclusão de requisitos adicionais (como os de segurança) necessários em cada iteração ou "espiral", o que o torna mais adaptável ao longo do SDLC. Uma discussão mais aprofundada sobre a incorporação de considerações de segurança no modelo Espiral pode ser encontrada no documento NIST SP 800-64.

5.3.4 Modelo DevOps e DevSecOps

O modelo DevOps é de certa forma uma extensão do modelo Ágil, proporcionando ciclos de lançamento mais rápidos, uma vez que a implantação e integração também fazem parte do ciclo. Já o modelo DevSecOps leva isso ainda mais longe, integrando práticas de segurança na metodologia DevOps, considerando a segurança em todas as fases do desenvolvimento de software de forma ágil. Algumas ferramentas como o Chef InSpec e o Gauntlt podem ajudar as organizações a implementar o modelo DevSecOps (ENISA, 2019).

Portanto, escolher o modelo SDLC adequado para um projeto IoT é uma decisão multifacetada que depende de vários fatores, incluindo o tipo de projeto, a equipe de desenvolvimento e o ambiente de implementação (ENISA, 2019; SOMMERVILLE, 2016).

De maneira complementar, existem várias ferramentas e recursos disponíveis para ajudar a incorporar práticas de segurança em todas as fases do SDLC. O OWASP Internet of Things Project fornece uma rica fonte de orientações sobre práticas de segurança

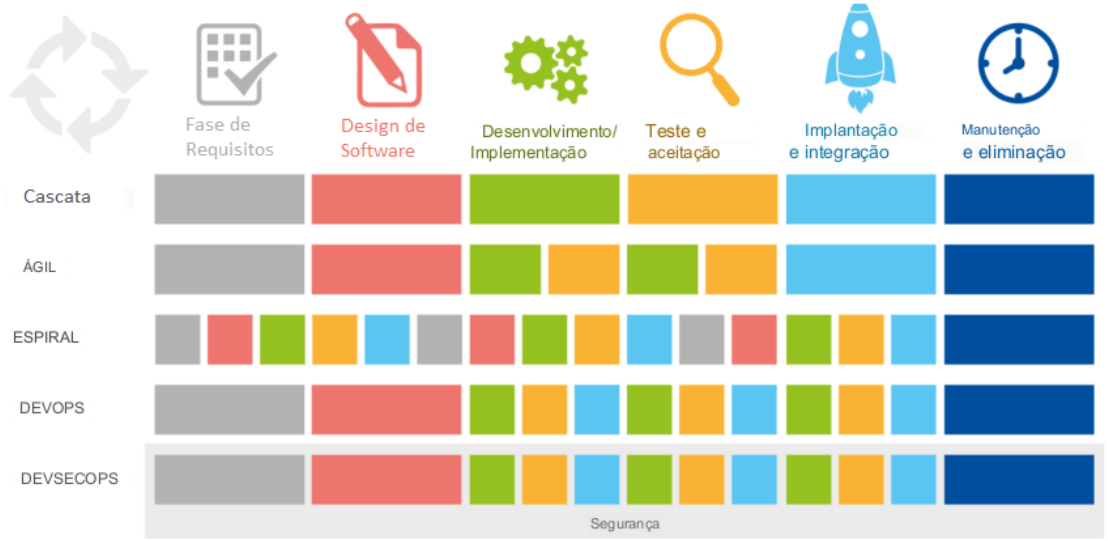


Figura 14 – Visão geral dos modelos de SDLC

Fonte: Adaptado de ENISA (2019)

específicas para o desenvolvimento de IoT. Além disso, o IoT Security Foundation tem diretrizes de melhores práticas que podem ser úteis para garantir a segurança ao longo do SDLC em projetos IoT.

Já para as equipes de desenvolvimento que escolheram adotar a metodologia DevOps/DevSecOps em seus projetos IoT, o Sonatype Nexus Platform e o Aqua Security são ferramentas poderosas que facilitam a integração de práticas de segurança em todo o ciclo de vida do desenvolvimento de software.

5.4 Fase de Requisitos

Na fase de requisitos, que é a etapa inicial de qualquer projeto de software, temos a oportunidade perfeita para construir uma fundação sólida para a segurança do dispositivo IoT (ENISA, 2019). Portanto, nesse momento, é fundamental obter uma compreensão abrangente dos requisitos de segurança e documentá-los de forma apropriada.

Essa fase, de maneira mais simples, pode ser vista como a elaboração de uma lista de compras para um projeto seguro: estamos definindo as necessidades do nosso sistema para garantir sua segurança.

Os requisitos de segurança abrangem tantos aspectos funcionais (o que o sistema deve fazer), quanto não funcionais (como o sistema deve operar) (SOMMERVILLE, 2016). Os requisitos funcionais são, portanto, as funcionalidades que nosso sistema precisa executar de forma segura (ENISA, 2019). Por exemplo, se nosso dispositivo IoT precisa coletar e armazenar dados do usuário, como ele fará isso de maneira a proteger a confidencialidade e a integridade desses dados?

Por outro lado, os requisitos não funcionais dizem respeito a como nosso sistema deve operar para assegurar sua segurança. Assim, esses requisitos podem incluir, por exemplo, elementos como o tempo de resposta necessário do nosso sistema frente a uma ameaça de segurança, ou as especificações técnicas como o tipo de sistema operacional, hardware e conexão que devem ser utilizados para garantir a segurança do produto IoT.

Dessa forma, na fase de requisitos, estamos elaborando uma lista completa de "ingredientes" necessários para projetar e implementar um dispositivo IoT seguro.

5.4.1 Importância dos requisitos de segurança

Os requisitos de segurança, como o próprio nome sugere, especificam as necessidades de segurança do seu projeto. Eles são uma parte crucial do processo de desenvolvimento, uma vez que fornecem um padrão contra o qual você pode avaliar a segurança do seu software. Esses requisitos não devem ser genéricos, mas sim específicos para o contexto do seu projeto, levando em consideração os riscos e ameaças potenciais à segurança que podem ser relevantes (ENISA, 2019).

Por exemplo, se você está desenvolvendo um dispositivo IoT médico, os requisitos de segurança podem incluir a criptografia de todos os dados de saúde do paciente em trânsito e em repouso, garantindo a autenticação segura de todos os usuários e a aplicação de atualizações de segurança em tempo hábil.

Além disso, é essencial manter uma visão holística ao elicitar os requisitos, considerando toda a arquitetura de IoT. Isso inclui as interações com a nuvem e a Internet, bem como com outros dispositivos. Assim, ao desenvolver para IoT, deve-se estar atento para evitar ataques que podem surgir devido a dependências e questões da cadeia de suprimentos. Com isso, pode se integrar e utilizar ferramentas e guias, como o OWASP Dependency-Check, que permite analisar e detectar vulnerabilidades públicas nas dependências da sua aplicação ou o OWASP Top 10 IoT (OWASP, 2018), que traz uma lista das principais vulnerabilidades vigentes em IoT.

Por exemplo, no caso da "Hello Barbie", muitas das vulnerabilidades exploradas estavam no serviço web associado, não na própria boneca. Portanto, quando se define os requisitos de segurança, é crucial abordar não só o dispositivo IoT em si, mas também todos os componentes da sua arquitetura.

5.4.2 Recomendações de boas práticas para requisitos de segurança

1. **Requisitos claros e específicos:** Seus requisitos de segurança devem ser precisos e detalhados. Vagueza e ambiguidade podem levar a mal-entendidos e falhas de segurança (Tony Rice et al., 2018). Por exemplo, em vez de simplesmente dizer "Os dados

devem ser seguros", diga "Os dados devem ser criptografados usando o algoritmo AES-256 quando estiverem em trânsito ou em repouso".

2. **Construção colaborativa dos requisitos:** Requisitos de segurança são uma preocupação de todos os envolvidos no projeto, e todos devem participar na sua elaboração. Isso inclui desenvolvedores, engenheiros de segurança, *product owners* e até mesmo os usuários finais, se possível (ENISA, 2019).

3. **Atender às normas e regulamentos:** Assegure-se de que seus requisitos de segurança estejam em conformidade com quaisquer normas ou regulamentos relevantes para o seu escopo. Para dispositivos IoT médicos, por exemplo, isso pode incluir a HIPAA, a GDPR na Europa e a LGPD no Brasil (Tony Rice et al., 2018).

4. **Revisão e atualização regulares:** Os requisitos de segurança devem ser revistos e atualizados regularmente para refletir a evolução das ameaças de segurança e as mudanças no seu projeto (ENISA, 2019).

Na fase de requisitos, nosso objetivo é estabelecer uma base sólida para o software IoT. Isso implica em seguir algumas recomendações: desenvolver requisitos claros e específicos; promover a construção dos requisitos de maneira colaborativa; observar rigorosamente as normas e regulamentos pertinentes ao escopo do projeto; e realizar revisões e atualizações regulares desses requisitos. Para uma visualização resumida essas recomendações, veja a Figura 15.

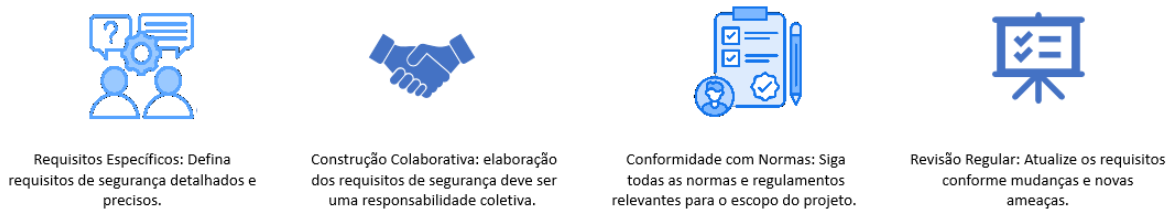


Figura 15 – Resumo Requisitos

Fonte: Autor

5.5 Fase de Design

Na fase de design de software, que segue a etapa de requisitos, é hora de esboçar como as funcionalidades do nosso dispositivo IoT serão concretizadas. Aqui, damos vida à nossa lista de requisitos, imaginando como as necessidades do usuário ou do negócio serão transformadas em recursos técnicos específicos do sistema (ENISA, 2019).

Mas não estamos apenas pensando no que nosso dispositivo IoT fará, também é crucial considerar como ele fará isso de forma segura. Isso significa que nesta fase começamos a desenhar as medidas de segurança. Por exemplo, como nosso dispositivo autenticará os usuários? Como garantirá a segurança dos dados que coleta e armazena?

Então, na fase de design, não estamos apenas projetando a aparência do nosso dispositivo IoT, mas também o "escudo" que o protegerá. Este é o momento de aliar criatividade e estratégia para desenvolver um projeto que seja ao mesmo tempo funcional e seguro (ENISA, 2019). Este trabalho minucioso no design será essencial para as fases de desenvolvimento e implementação que virão a seguir, garantindo que a segurança esteja entrelaçada em cada aspecto do nosso dispositivo IoT.

5.5.1 Importância do Design Seguro

O design seguro de software no desenvolvimento de IoT é uma etapa fundamental, onde as especificações de segurança são transformadas em um plano de implementação detalhado. Sem um design seguro, pode se tornar exorbitantemente caro ou praticamente inviável corrigir falhas de segurança em etapas posteriores do ciclo de desenvolvimento (ENISA, 2019).

Um exemplo poderoso da importância do design seguro é do ataque Mirai botnet, no qual inúmeros dispositivos IoT foram comprometidos e transformados em "bots" por conta do uso de senhas padrão ou fracas (ANTONAKAKIS et al., 2017).

Então, aqui entra uma questão interessante: se na fase de requisitos tivéssemos especificado a necessidade de garantir que os usuários alterem as senhas padrão, já estaríamos um passo à frente. Mas como fazer isso da melhor maneira e que ainda garanta uma boa experiência ao usuário? Isso é o que resolveríamos na fase de design.

Por exemplo, poderíamos ter projetado o sistema para exigir que o usuário alterasse a senha padrão na primeira configuração, ou ainda, poderíamos fazer com que o sistema bloqueasse o acesso se a senha padrão não fosse alterada após um certo período, o que talvez poderia impactar a experiência do seu produto, porém essa são decisões importantes que inevitavelmente deverão ser tomadas nessa fase do SDLC.

Assim, de maneira geral, com essas decisões de design seguro, o impacto do ataque Mirai poderia ter sido significativamente mitigado. O que mostra como o design não apenas

dá "forma" ao nosso dispositivo IoT, mas também projeta o "escudo" que o protegerá. Portanto, na fase de design, é crucial planejar e implementar de forma eficaz os requisitos de segurança estabelecidos anteriormente.

Além disso, é importante ressaltar que o design seguro vai além da proteção contra invasões e vazamento de dados. Em muitos casos, a integridade física do dispositivo e a segurança humana também podem estar em jogo. Em sistemas industriais, por exemplo, uma falha de segurança pode levar a danos materiais ou até mesmo perda de vidas (ENISA, 2019). Portanto, a segurança física do dispositivo deve ser igualmente considerada durante a fase de design.

5.5.2 Recomendações de Boas Práticas para Design Seguro

1. **Modelagem de Ameaças e Análise da Superfície de Ataque:** Utilizar técnicas de modelagem de ameaças, como o Microsoft Threat Modeling Process, pode ajudar a compreender o ambiente de segurança em que o sistema vai operar. Isto é vital para identificar potenciais pontos de entrada para invasores e as ameaças que seu sistema pode enfrentar (Tony Rice et al., 2018).

2. **Implementação de Controles de Segurança:** Baseado na análise de ameaças, é necessário projetar e implementar controles de segurança adequados (ENISA, 2019). Ferramentas como o OWASP Cheat Sheets podem fornecer orientação valiosa sobre a implementação de controles de segurança eficazes.

3. **Design de Segurança na Arquitetura:** A arquitetura do sistema deve ser projetada tendo em vista a segurança (Tony Rice et al., 2018). Isso pode envolver a implementação de design seguro e o uso de padrões de segurança, como os fornecidos pelo ISO/IEC 27001.

4. **Conformidade com Normas:** O design do software deve estar em conformidade com as normas e regulamentos relevantes para o seu escopo, como o GDPR na Europa e a LGPD no Brasil (Tony Rice et al., 2018).

5. **Revisão e Atualização Regular:** O design do software precisa ser revisado e atualizado regularmente. A adoção de práticas de DevSecOps pode ajudar a garantir que a segurança seja considerada durante todo o ciclo de vida do desenvolvimento de software (ENISA, 2019).

6. **Princípio do Privilégio Mínimo:** A aplicação deste princípio pode limitar o dano que um atacante pode causar se uma conta for comprometida (ENISA, 2019). Ferramentas como o BeyondTrust's Operational Technology (OT) Security Solutions podem ajudar a gerenciar o acesso privilegiado.

7. **Planejamento para Atualizações:** Sabemos que nenhum sistema estará para sempre livre de vulnerabilidades de segurança. Portanto, é essencial que, durante a fase de design, preparemos nosso dispositivo IoT para a instalação segura e confiável de futuras atualizações de segurança (Tony Rice et al., 2018). Por exemplo, poderíamos planejar uma função no nosso dispositivo que verifique periodicamente se existem atualizações disponíveis, e se sim, as instale automaticamente e de maneira segura. Assim, estaremos um passo à frente no jogo de gato e rato da segurança cibernética.

8. **Segurança Física:** Além da segurança lógica, o design deve levar em conta a segurança física do dispositivo (ENISA, 2019). A OWASP IoT Security Verification Standard fornece recomendações para a segurança física do dispositivo.

Na fase de design, temos a chance de traçar o caminho para um software IoT seguro. Esse mapa inclui a análise de ameaças, para entender onde podemos ser vulneráveis; definição de controles de segurança, para proteger nosso sistema; consideração de segurança no design da arquitetura; respeito às normas e regulamentos; planejamento para futuras atualizações de segurança; revisões regulares para manter nosso design atualizado; aplicação do princípio de mínimo privilégio, para limitar possíveis danos; e, por fim, pensando na segurança física do dispositivo. Como um resumo visual, veja a Figura 16.

Figura 16 – Resumo Design

Fonte: Autor

5.6 Fase de Desenvolvimento/Implementação

A fase de Desenvolvimento ou Implementação é o estágio onde nossos conceitos e planos da fase de Requisitos e Design ganham vida. Este é o ponto no Ciclo de Vida de Desenvolvimento de Software (SDLC) em que as funcionalidades técnicas começam a tomar forma e o software de nosso dispositivo IoT começa a ser codificado (ENISA, 2019).

Assim, as descrições e os esquemas dos estágios anteriores se transformam em funções e recursos reais, como um roteiro que se torna uma cidade vibrante. Agora, entramos na parte técnica de nosso projeto, tecendo a segurança diretamente em nossa lógica de programação (Tony Rice et al., 2018). É aqui então que começamos a aplicar nossas práticas de programação segura, garantindo que cada linha de código esteja de acordo com os princípios de segurança estabelecidos.

5.6.1 Importância da programação segura

Entender o valor da programação segura é um componente-chave no desenvolvimento de dispositivos IoT. Pense nos dispositivos de IoT como uma cidade movimentada: cheia de dados importantes, sempre conectada e aberta ao mundo. Esta exposição, porém, torna nossos "cidadãos digitais" alvos atraentes para os "foras da lei" cibernéticos. Programar de maneira segura é como construir fortes muralhas e sistemas de defesa para nossa cidade, minimizando as brechas por onde os invasores podem entrar.

Lembra quando falamos sobre design seguro? Agora é quando realmente colocamos isso em prática. As defesas que projetamos no papel agora são construídas em código, fortalecendo a segurança de nossos dispositivos.

Para ilustrar a importância disso, vamos recordar o ataque Mirai novamente: uma cidade de dispositivos IoT que se transformou em uma máquina de guerra digital (uma botnet). Este grande ataque DDoS só foi possível porque as "muralhas" desses dispositivos foram facilmente transpostas - os invasores simplesmente usaram as credenciais padrão que não foram alteradas na fase de implementação (ANTONAKAKIS et al., 2017). Assim, um forte design e uma sólida implementação podem fazer toda a diferença no campo de batalha da IoT (ANTONAKAKIS et al., 2017).

5.6.2 Recomendações de boas práticas para implementação segura

1. **Práticas de Codificação Segura:** Utilize diretrizes e padrões de codificação segura, como o NIST SSDF e o OWASP - Secure Coding Practices Quick Reference Guide para evitar erros comuns que podem levar a vulnerabilidades de segurança. Além disso, as 25 Fraquezas de Software Mais Perigosas da CWE (CWE Top 25 -

2022) oferecem uma visão detalhada das fraquezas mais críticas que podem levar a sérias vulnerabilidades em software (Tony Rice et al., 2018).

2. **Utilize Frameworks de IoT Seguros:** Os frameworks de IoT, tanto proprietários quanto de código aberto, podem ajudar os desenvolvedores a integrar rapidamente componentes de segurança, prevenir vulnerabilidades e fornecer segurança desde o início do desenvolvimento. Os frameworks proprietários, como o Cisco IoT Platforms e o Microsoft Azure IoT Suite, oferecem recursos exclusivos, suporte dedicado, opções de personalização, segurança aprimorada e maior estabilidade (Nupur Pal, 2022). Por outro lado, os frameworks de código aberto, como o Eclipse IoT e o DeviceHive, são conhecidos por seus custos mais baixos, desenvolvimento orientado pela comunidade, maior facilidade de integração com sistemas existentes e maior transparência (Nupur Pal, 2022). Para ver mais detalhes sobre frameworks cheque esse post, contendo uma análise aprofundada de 30 frameworks IoT.

3. **Análise de Código Estática:** Utilize ferramentas de análise de código estática, como SonarCloud, Fortify e Checkmarx, para ajudar a identificar possíveis problemas de segurança durante a fase de desenvolvimento de software (Tony Rice et al., 2018).

4. **Revisão de Código:** Encoraje revisões de código entre pares para permitir que outros desenvolvedores identifiquem possíveis problemas de segurança. As práticas de programação extrema (XP), por exemplo, enfatizam a importância das revisões de código para a qualidade geral do software (WELLS, 2013).

5. **Cuidado com Componentes de Terceiros:** Antes de integrar APIs, bibliotecas e outras ferramentas de terceiros em seu sistema, verifique-as cuidadosamente para possíveis vulnerabilidades de segurança. O OWASP Dependency-Check é uma ferramenta útil que detecta vulnerabilidades publicamente divulgadas em dependências de aplicativos (ENISA, 2019).

6. **Implemente a Gestão de Configuração:** Utilize sistemas de controle de versão como Git, e outras ferramentas de gestão de configuração como Ansible e Chef, para ajudar a manter a segurança e a flexibilidade do seu software (Tony Rice et al., 2018).

Na fase inicial da implementação, a meta é garantir a implementação segura e conforme planejado do código do software IoT. Isso envolve seguir práticas de codificação segura para prevenir erros comuns; fazer uso de frameworks de IoT seguros, sejam eles proprietários ou de código aberto, para uma integração efetiva dos componentes de segurança; aplicar ferramentas de análise de código estática para identificar possíveis problemas de segurança durante o desenvolvimento; realizar revisões de código entre pares

para detectar e corrigir eventuais problemas de segurança; examinar cuidadosamente todos os componentes de terceiros, tais como APIs, bibliotecas e ferramentas, à procura de vulnerabilidades potenciais; e, por fim, empregar sistemas de controle de versão e outras ferramentas para gerir a configuração, mantendo a segurança e a flexibilidade do software. Para uma visualização resumida destas etapas, consulte a Figura 17.

Figura 17 – Resumo Implementação

Fonte: Autor

5.7 Fase de Testes e Aceitação

Chegamos à fase de Testes e Aceitação, onde todo o trabalho duro de planejamento, design e implementação é meticulosamente examinado. Nesse estágio do ciclo de vida do desenvolvimento de software, avaliamos se o software desenvolvido cumpriu os critérios de sucesso definidos em fases anteriores. Nos tornamos, em certo sentido, os próprios "atacantes", buscando encontrar e consertar quaisquer brechas de segurança antes que o software seja liberado para produção (ENISA, 2019).

Assim, se a implementação segura é como construir nosso muro, então o teste de segurança é verificar cada tijolo para garantir que está sólido e bem colocado. Portanto, durante esta fase, examinamos se todas as medidas de segurança estão funcionando como esperado e buscamos ativamente qualquer fraqueza que possa ter escapado das fases anteriores.

5.7.1 Importância dos Testes de Segurança

A fase de Testes de Segurança e Aceitação é como uma linha de defesa estratégica que posicionamos antes de enviar nosso dispositivo IoT para o mundo do ciberespaço. Nesse estágio, assumimos a persona de "hackers éticos", explorando nosso próprio sistema com o objetivo de identificar quaisquer falhas de segurança que possam ter passado despercebidas nas etapas anteriores (ENISA, 2019).

Mesmo com a segurança sendo um pilar essencial em todas as etapas do desenvolvimento de software, ela se torna ainda mais crítica durante os testes e a aceitação. Sendo que essa fase é a nossa oportunidade de verificar que tanto as medidas de segurança implementadas quanto o software estão funcionando conforme planejado e isso adquire uma importância extra no universo da IoT, onde a complexidade dos sistemas e a interdependência dos componentes não apenas ampliam o potencial de ataques, mas também aumentam a complexidade dos testes (YOUSEFNEZHAD et al., 2023).

Para ilustrar a importância dos Testes de Segurança e Aceitação, vamos considerar o exemplo da Hello Barbie. Esta boneca inteligente usava serviços web para processar comandos de voz. No entanto, logo após seu lançamento, pesquisadores encontraram várias vulnerabilidades nesses serviços, principalmente relacionadas à segurança na web, e não ao sistema embarcado da boneca (BRAGA, 2016). Entre as vulnerabilidades detectadas estavam a falta de validações de entrada adequadas e a possibilidade de falsificação de tokens de sessão (BRAGA, 2016), falhas que poderiam ter sido facilmente identificadas e corrigidas por testes de segurança eficazes, mesmo aqueles automatizados, durante a fase de testes e aceitação. Assim, este exemplo serve como um lembrete poderoso de como o descuido com os detalhes durante a fase de testes pode resultar em vulnerabilidades reais e potencialmente perigosas para o cenário IoT.

5.7.2 Recomendações de Boas Práticas para Testes de Segurança

1. **Planejamento do Ambiente de Teste:** Configure um ambiente de teste que reflita o mais próximo possível o ambiente de produção. Ferramentas de virtualização e orquestração como Docker e Kubernetes podem ser úteis para replicar ambientes complexos (ENISA, 2019).

2. **Definição da Estratégia de Teste:** Defina uma estratégia de teste que inclua tanto testes automatizados quanto manuais, bem como testes de penetração (pentesting) e fuzzing conforme necessário. A estratégia de teste deve ser baseada na modelagem de ameaças realizada durante a fase de design (ENISA, 2019).

3. **Análise Estática e Dinâmica:** Utilize técnicas de análise estática e dinâmica para verificar o código e o comportamento do software em execução. Ferramentas como SonarQube para análise estática e OWASP ZAP para análise dinâmica podem ser úteis (Tony Rice et al., 2018).

4. **Análise de Componentes de Terceiros:** Sempre avalie cuidadosamente todos os componentes de terceiros, como APIs, bibliotecas e frameworks, para possíveis vulnerabilidades. O OWASP Dependency-Check pode ajudar a identificar vulnerabilidades conhecidas em componentes de terceiros (ENISA, 2019).

5. **Gestão de Configuração e Versão:** Use sistemas de controle de versão como Git para manter o rastreamento das alterações. Ferramentas de gestão de configuração como Ansible podem ajudar a manter a consistência e a segurança das configurações do software (ENISA, 2019).

6. **Coleta de Métricas:** Colete métricas de desempenho e segurança para obter insights e tomar decisões informadas. Ferramentas como Prometheus e Grafana podem ser úteis para coletar e visualizar métricas (ENISA, 2019).

7. **Verificação da Complexidade do Ambiente:** A análise da complexidade do ambiente de teste, quando comparado ao ambiente de produção, é uma peça-chave no quebra-cabeça da segurança do software. Esse processo envolve a verificação do número de integrações e uma compreensão profunda de como essas interconexões podem influenciar a segurança do software (ENISA, 2019).

   Na verdade, essa recomendação é intrinsecamente ligada a várias outras nesta lista. Por exemplo, no momento do planejamento do ambiente de teste, deve-se considerar a complexidade inerente das integrações, garantindo que o ambiente de teste reflita com precisão o ambiente de produção. Similarmente, durante a análise de componentes de terceiros, precisamos estar conscientes de como cada integração adiciona uma camada extra de complexidade e, potencialmente, de risco ao nosso software.

Para auxiliar nesse processo, podemos recorrer a ferramentas open source, já mencionadas como o OWASP Dependency-Check e o OWASP ZAP, que são úteis para identificar e analisar possíveis vulnerabilidades nas integrações. Além disso, o Grafana tambem pode ser usado para monitorar a atividade dessas integrações, fornecendo *insights* valiosos sobre como a complexidade e a interdependência podem afetar a segurança do software.

Na fase de testes, nossa foco principal é validar a segurança e o funcionamento do nosso software IoT. De maneira resumida, isso envolve a configuração de um ambiente de teste similar ao de produção; a definição de uma estratégia que abarque testes automatizados, manuais, pentesting e fuzzing conforme a necessidade; a realização de análises estáticas e dinâmicas para checar o código e o comportamento do software; a avaliação dos componentes de terceiros para identificar possíveis vulnerabilidades; a utilização de sistemas de controle de versão e ferramentas de gestão de configuração para manter a segurança; a coleta de métricas para auxiliar na tomada de decisões; e a avaliação da complexidade do ambiente de teste para compreender seu impacto na segurança. Confira essas etapas na Figura 18.

Figura 18 – Resumo Testes e Aceitação

Fonte: Autor

5.8 Fase de Implantação e Integração

No ciclo de vida do desenvolvimento de software IoT, após as fases cuidadosas de requisitos, design, desenvolvimento e os rigorosos testes de aceitação, encontramo-nos na fase de implantação e integração. Esse momento é uma junção minuciosa e estratégica de diversos componentes que têm o objetivo de trazer à vida real, o ambiente de produção, as ideias que até agora só existiam em nossos esquemas, protótipos e ambientes de teste (ENISA, 2019).

No entanto, uma implantação segura não significa simplesmente apertar um botão e esperar que o software funcione em perfeita harmonia com o mundo real. Assim como na fase de design, onde imaginamos não apenas as funcionalidades, mas também como elas seriam protegidas, na fase de implantação precisamos considerar fatores além do funcionamento simples do software.

De forma análoga, um ambiente de implantação ideal pode ser visualizado como um castelo bem fortificado, pronto para enfrentar qualquer adversidade. As muralhas robustas e imponentes representam as medidas de segurança meticulosamente projetadas nas fases anteriores. Cada pedra colocada, cada amarração feita, refletem os esforços despendidos para garantir a integridade do nosso software IoT.

Agora, no ambiente de produção, o campo de batalha real, essas muralhas serão testadas contra ameaças que, em sua maioria, acabam sendo de uma complexidade muito maior do que aquelas encontradas nos ambientes controlados de teste. Então, assim como as fortificações de um castelo são postas à prova pelos ataques de um exército inimigo, as defesas do nosso software IoT agora enfrentarão os desafios e riscos do mundo real. E nesse cenário, a pergunta essencial que fazemos não é apenas “O software está funcionando?”, mais do que isso, devemos perguntar “O software está funcionando de maneira segura?”. E isso seria como questionar não apenas se as muralhas do castelo continuam de pé, mas se estão efetivamente protegendo seus habitantes e resistindo às investidas inimigas.

Portanto, nesta fase, nos preocupamos não apenas com a funcionalidade, mas com a segurança e a resiliência do nosso “castelo”, no caso o ambiente de produção onde o software IoT será operacionalizado. Com isso, existem diversas considerações a serem analisadas pelo time de desenvolvedores e a equipe de operação do software como um todo. Entre elas:

- A capacidade de reverter uma implantação, o chamado “rollback”, é como ter uma saída secreta no nosso castelo (ENISA, 2019). Se um inimigo consegue entrar, ou seja, se um problema inesperado ocorre, podemos usar essa saída para retornar rapidamente à segurança da versão anterior do software. Isso minimiza o tempo de inatividade e o impacto nos usuários finais.

- A disponibilidade é outro componente crucial do nosso castelo. Como a guarda que deve estar de plantão 24/7, nossos sistemas IoT precisam estar sempre disponíveis. Para isso, eles devem ser capazes de lidar com falhas de maneira graciosa, implementando estratégias de redundância e *failover* para garantir a disponibilidade contínua do serviço (ENISA, 2019).
- Por último, mas não menos importante, a fase de implantação e integração deve preparar o terreno para a próxima fase, a manutenção. Para isso, devemos garantir que as alterações sejam rastreadas e que os registros e métricas de desempenho estejam bem configurados. Esses registros são como o diário do castelo, fornecendo insights valiosos para a manutenção e solução de problemas futuros (ENISA, 2019).

Assim, a fase de implantação e integração não é apenas sobre tornar o software operacional, mas sobre fazer isso de forma segura, eficiente e resiliente. É aqui que colocamos à prova todas as fases de desenvolvimento do software de nosso dispositivo IoT, para que todas as funcionalidades, assim como seu "escudo", entrem em ação no mundo real (ENISA, 2019).

5.8.1 Importância da Implantação e Integração Segura

A fase de implantação e integração é a fronteira entre o planejamento cuidadoso e a realidade indomada do mundo real, é a transição do software IoT do ambiente protegido de desenvolvimento e teste para o ambiente de produção, onde será exposto a uma variedade de condições imprevistas e potencialmente hostis.

Um exemplo que ilustra a importância dessa fase é o caso da Boneca Cayla. Essa boneca IoT foi projetada para interagir com crianças, respondendo suas perguntas e contando histórias, o que não traz nenhum alerta para o olhar normal, mas brevemente depois do seu lançamento foi explicitado por especialistas em cibersegurança como Tim Medim, que a boneca não tinha medidas minímas de segurança adequadas. Ela permitia que estranhos se conectassem à boneca via Bluetooth de maneira bem simples, pois a mesma não possuía qualquer tipo de autenticação ou verificação (Tim Medin, 2015). Esse fator mostra, para nós desenvolvedores, como más decisões nas etapas anteriores do SDLC, podem gerar resultados catastróficos ao serem implantados e jogados para o teste da "seleção natural" do mundo real, que acabou transformando um brinquedo aparentemente inofensivo em um potencial risco de segurança para as crianças.

Além disso, é importante lembrar que a complexidade aumenta no contexto IoT, devido à heterogeneidade dos ambientes de implantação e a natureza aberta dos dispositivos IoT e suas muitas conexões e dependências, as quais os administradores podem não ter controle total. Assim, por exemplo, um dispositivo IoT pode funcionar perfeitamente em um ambiente de teste, mas falhar quando exposto a condições de rede variáveis no

mundo real, ou pode ser seguro quando usado isoladamente, mas tornar-se vulnerável quando integrado a uma rede com outros dispositivos ou a casos de usuários maliciosos não refletidos nas decisões de design e requisitos do software (Nupur Pal, 2022).

Por isso, ao prepararmos o "castelo" para a batalha real, devemos considerar todos os cenários possíveis, não apenas os que esperamos encontrar, sendo fundamental ter medidas de segurança robustas e planos de contingência em vigor, incluindo mecanismos de redundância, *failover* e *rollback* (ENISA, 2019).

5.8.2 Recomendações de Boas Práticas para Implantação Segura

1. **Planejamento Cuidadoso da Implantação:** Uma implantação bem-sucedida começa com um planejamento cuidadoso. Isso envolve a comunicação clara com todas as partes envolvidas (por exemplo, usuários finais, equipes de produção, equipes de desenvolvimento, integradores) e a consideração cuidadosa dos desafios específicos do espaço IoT, como a heterogeneidade dos ambientes de implantação e as muitas interdependências envolvidas (ENISA, 2019). Ferramentas como JFrog Artifactory e Docker podem ajudar na automação do processo de implantação.

2. **Escolha da Estratégia de Implantação Correta:** A estratégia de implantação adequada é crucial para o sucesso do lançamento de uma nova versão de software. Cada estratégia tem suas vantagens e desvantagens, e a escolha da melhor estratégia depende de vários fatores, incluindo o impacto da mudança no sistema e/ou nos usuários finais, timings de implantação/reversão, requisitos de tempo de inatividade, entre outros (ENISA, 2019).

   Aqui estão alguns exemplos de estratégias de implantação:

   - **Azul-Verde:** Duas versões idênticas do ambiente de produção (Blue e Green) são configuradas. Em qualquer momento, apenas um ambiente está ativo. Após o teste, o roteador é alternado para o ambiente Green, tornando-o ativo (Martin Fowler, 2010). Exemplos de ferramentas que suportam essa estratégia incluem AWS CodeDeploy, Azure Pipelines e Cloud Foundry.
   - **Canario:** A nova versão do software é lentamente implantada para um pequeno subconjunto de usuários antes de ser lançada para toda a infraestrutura (Martin Fowler, 2014). Exemplos de ferramentas que suportam essa estratégia incluem AWS CodeDeploy e Azure Pipelines.
   - **Teste A/B:** Duas ou mais versões de um aplicativo são lançadas para os usuários em circunstâncias controladas para avaliar qual delas tem melhor desempenho (Google, 2023). Exemplos de ferramentas que suportam essa estratégia incluem Google Optimize e Optimizely.

Veja esse documento feito pela Google[1] caso queira ver mais detalhes sobre as estrategias de implantação.

3. **Gestão de Autorização de Ativos e Usuários:** A gestão cuidadosa da autorização de ativos e usuários é crucial para prevenir o acesso e a utilização não autorizados do software. Ferramentas como Keycloak e Okta podem auxiliar na gestão de autorizações (ENISA, 2019).

4. **Monitoramento de Métricas:** O monitoramento de métricas apropriadas é importante para garantir a "saúde" do software em execução. Tais métricas podem incluir o número de bugs relatados, número de vulnerabilidades e fraquezas identificadas, número de tentativas de exploração, etc. Ferramentas como Prometheus e Grafana podem ser úteis para coletar e visualizar métricas (ENISA, 2019).

5. **Gestão de Mudanças e Atualizações de Software:** Todas as atualizações de software devem seguir uma abordagem estruturada de gestão de mudanças e devem garantir que qualquer atualização no sistema retenha pelo menos o mesmo nível de segurança fornecido pela solução anterior. Ferramentas como Ansible e Puppet podem ajudar a automatizar o processo de gestão de mudanças (ENISA, 2019).

Na fase de implantação e integração, o objetivo é assegurar a melhor forma de colocar o software IoT no ambiente de produção, garantindo sua segurança e qualidade planejadas. Para isso, seguimos as seguintes práticas: planejamento cuidadoso da implantação, com uma comunicação eficaz de todas as etapas; escolha da estratégia de implantação adequada, que pode ser Canário, Teste A/B, Azul-Verde, entre outras; gestão efetiva de autorização, prevenindo acessos e usos não autorizados do software; monitoramento contínuo de métricas para garantir a saúde do software; e gestão de mudanças, assegurando que as atualizações mantenham ou melhorem a segurança do software. Para uma visão resumida e visual dessas práticas, consulte a Figura 19.

[1] https://cloud.google.com/architecture/application-deployment-and-testing-strategies?hl=pt-br

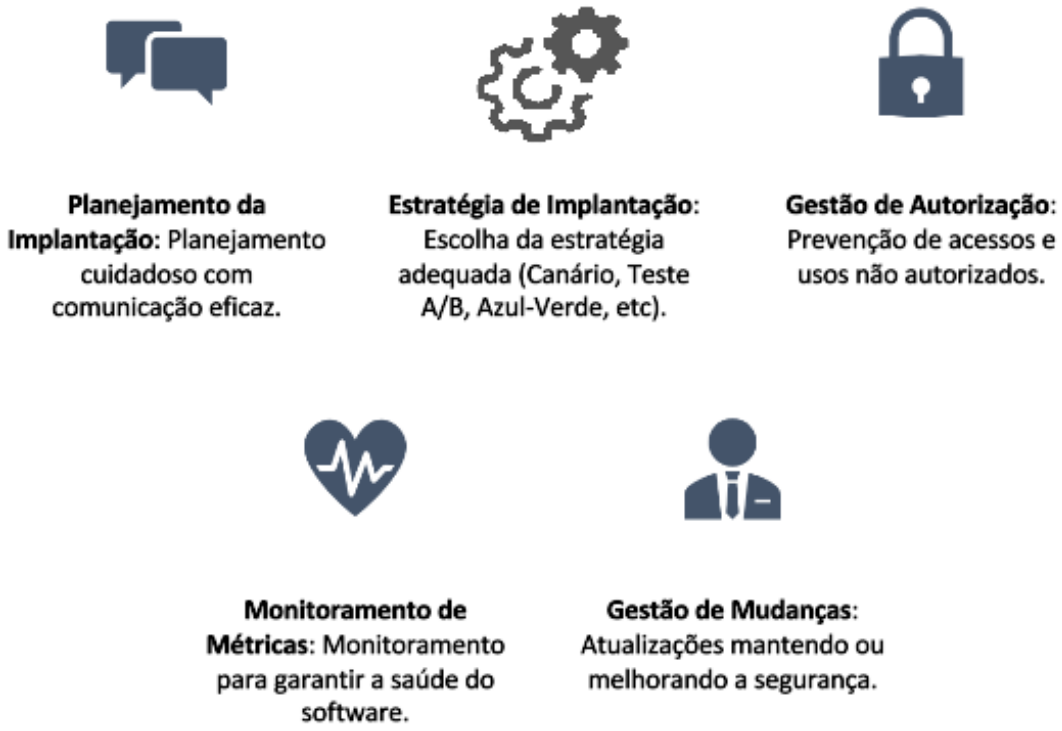


Figura 19 – Resumo Implantação e Integração

Fonte: Autor

5.9 Fase de Manutenção e Descarte

Na jornada de desenvolvimento de software para IoT, a fase de manutenção é como o período após a construção do nosso castelo, não basta apenas construir um castelo robusto e quase impenetrável e pensar que o trabalho está concluído. É essencial que o castelo seja continuamente mantido e aprimorado para enfrentar desafios futuros. Assim, a fase de manutenção se torna uma atividade essencial e contínua, onde o nosso "castelo" ou sistema IoT, está em constante evolução e adaptação (ENISA, 2019).

A manutenção de um sistema IoT envolve uma variedade de tarefas, como atualizações de software, correções de bugs, resposta a incidentes e, finalmente, a desativação segura ou "descarte". Dentre esses, um aspecto crítico é a atualização de software remota, também conhecida como atualização "over-the-air" (OTA) (ENISA, 2019). Tal qual um rei ou general que precisa enviar novas ordens ou instruções para sua guarnição a distância, as atualizações OTA permitem que os desenvolvedores corrijam problemas, atualizem funcionalidades ou reforcem a segurança, sem precisar de acesso físico ao dispositivo, o que é crucial no contexto de IoT, dada a natureza dispersa e a escala dos dispositivos.

Contudo, é importante observar que essas atualizações OTA devem ser realizadas de maneira segura, para evitar a criação de brechas de segurança. Neste cenário, ferra-

mentas como Mender e Eclipse hawkBit podem ser extremamente úteis para gerenciar essas atualizações de uma maneira segura e eficiente (ENISA, 2019).

Além disso durante a fase de manutenção, a monitoração contínua é outro aspecto fundamental. Como um rei que precisa estar ciente de todas as atividades em seu castelo, os administradores precisam monitorar continuamente todos os aspectos do sistema IoT. Ferramentas como ELK Stack e Zabbix podem fornecer insights importantes para detectar e responder a ameaças, assim como para garantir a saúde contínua do sistema IoT e suas dependências.

5.9.1 A Importância da Manutenção e Descarte Seguro

A fase de manutenção e descarte no ciclo de vida de um software IoT não é apenas uma extensão do trabalho anterior, mas representa um constante esforço de proteção e melhoria em um cenário cada vez mais complexo e imprevisível, que após sair de um ambiente controlado de desenvolvimento e teste, o sistema será continuamente exposto a diversas condições imprevistas e ameaças potenciais, o que exige um esforço com a manutenção contínua do software e também com descarte seguro do mesmo em caso do fim de vida do produto ou pedido do usuário (ENISA, 2019).

Tomemos como exemplo o caso da Hello Barbie e suponhamos que, após a implantação, os desenvolvedores descobrissem uma vulnerabilidade que poderia permitir a extração não autorizada de dados pessoais armazenados na boneca. Nesse caso, portanto, uma robusta estratégia de manutenção, que inclui atualizações de software seguras e eficazes "over-the-air" (OTA), poderia permitir a rápida implementação dessas correções de segurança em grande escala, evitando potencialmente a exposição dos dados pessoais de milhares de crianças e o impacto na marca da empresa.

Além disso, à medida que o produto chega ao fim de seu tempo de vida e entra na fase de descarte, emerge um novo desafio, que seria o descarte seguro desses dispositivos, sendo que muitas vezes podem conter muitos dados sensíveis. Como exemplo, consideremos a Hello Barbie novamente, e nesse caso uma boneca que passou anos interagindo com uma criança, coletando informações pessoais e aprendendo com as conversas, e que ao longo desse tempo esses dados coletados foram armazenados em servidores e/ou na própria boneca. Agora, imagine que a Mattel, a fabricante da boneca, declare o fim de vida do software e serviços da Hello Barbie, e não implemente esforços nenhum para com a prática de um descarte seguro. Esse caso de descarte sem um procedimento seguro e eficiente para o software e os dados, resulta nas informações sensíveis poderem ficar vulneráveis ao acesso não autorizado, causando graves violações de privacidade. Assim, para garantir que isso não ocorra, a empresa poderia implementar uma série de ações para assegurar um descarte seguro, que poderiam incluir a remoção completa de todos os dados armazenados nos servidores da empresa relacionados à boneca, ou a implementação de

atualizações de software que garantam que nenhum dado pessoal permanece na boneca após o fim de sua vida útil.

Portanto, a fase de manutenção e descarte é um componente crítico no ciclo de vida do desenvolvimento de software para IoT, não apenas permitindo que o sistema sobreviva, mas também prospere no dinâmico e desafiador cenário do mundo real. Além disso ele é essencial para garantir a segurança dos dados dos usuários até mesmo após o término da vida útil do produto (ENISA, 2019).

5.9.2 Recomendações para Práticas de Manutenção Segura

1. **Avaliações Regulares de Vulnerabilidade e Testes de Penetração:** As avaliações regulares de vulnerabilidade e os testes de penetração ajudam a prevenir ataques à nuvem, rede e dispositivos IoT de extremidade, bem como aos aplicativos desenvolvidos para esses dispositivos (ENISA, 2019). Ferramentas open source como OWASP ZAP, Burp Suite Community Edition e Kali Linux Tools são úteis para essa finalidade.

2. **Prestação de Contas e Planejamento de Continuidade:** A prestação de contas é crucial no ecossistema IoT, considerando especialmente que dispositivos com restrições de energia podem não ser capazes de criar ou armazenar arquivos de log (ENISA, 2019). Ferramentas como Veeam Backup e Elastic Beats podem ser usadas para gerenciar logs e backups.

3. **Descarte Seguro de Software e Dispositivos:** O descarte seguro de software obsoleto é necessário para preservar a privacidade e fornecer mecanismos de apagamento de dados (ENISA, 2019). Para isso, pode-se utilizar ferramentas como o Darik's Boot and Nuke (DBAN).

4. **Atualizações de Software e Conformidade:** É vital manter a conformidade regulatória e emitir patches de segurança de forma oportuna e confiável no ecossistema IoT (Google, 2023; GARCIA-MORCHON; KUMAR; SETHI, 2019). Ferramentas como Mender e Eclipse hawkBit ajudam a gerenciar atualizações de software.

5. **Manutenção de Terceiros:** Quando a manutenção de IoT é delegada a terceiros, ela deve ocorrer de forma segura, controlando o acesso, gerenciando permissões, auditando e garantindo a responsabilidade (ENISA, 2019; Tony Rice et al., 2018).

Na fase de manutenção e descarte, a intenção é garantir a segurança contínua do software IoT mesmo após sua implantação, enfrentando os desafios únicos do ambiente de IoT, com uma vasta gama de usuários e cenários de implementação. Assim, seguimos as seguintes páticas: atualizações de software e patches de segurança de forma oportuna,

mantendo a conformidade regulatória; realizamos avaliações regulares de vulnerabilidade e testes de penetração para prevenir ataques; efetuamos o descarte seguro de software e dispositivos obsoletos para prevenir a exploração indevida de dados; temos em vista a prestação de contas e o planejamento de continuidade, preparando-nos para eventuais contratempos através de mecanismos como backups automáticos e redundância; e cuidamos da manutenção de terceiros sob condições seguras, incluindo o controle de acesso e a realização de auditorias. Para um resumo visual dessas práticas, veja a Figura 20.

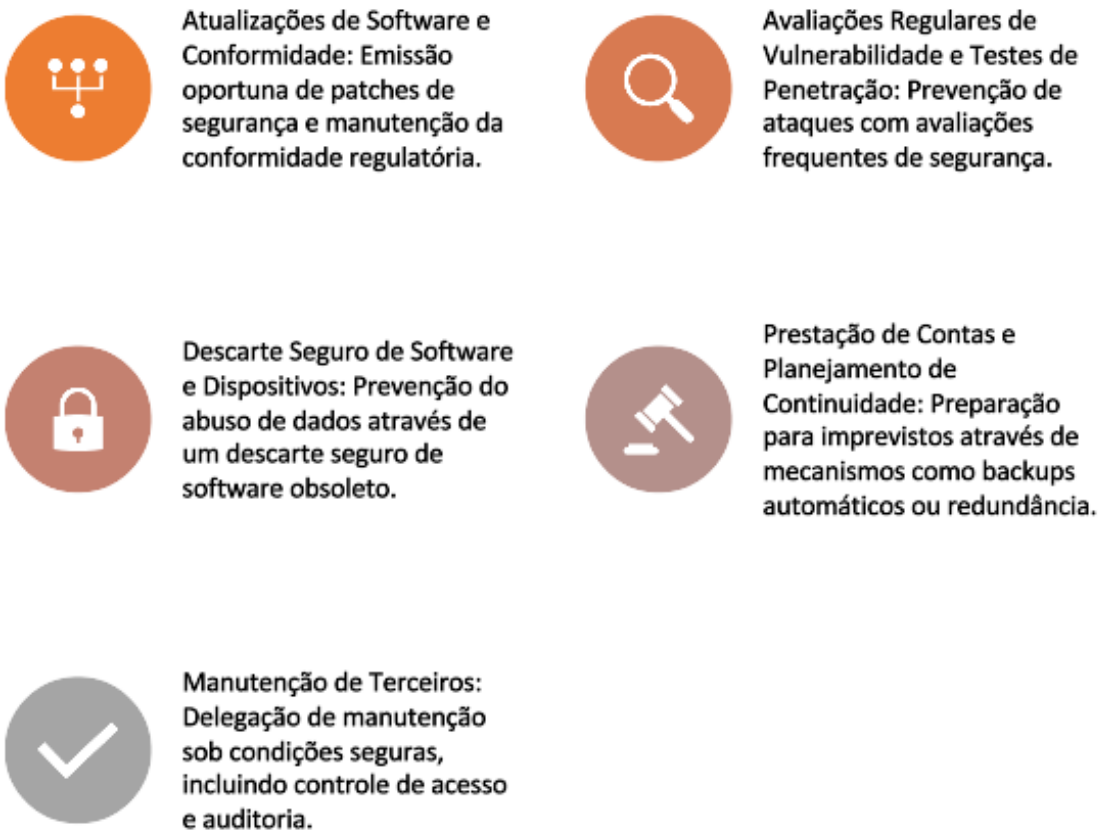


Figura 20 – Resumo Manutenção e Descarte

Fonte: Autor

5.10 Plantando a Semente de uma Cultura de Segurança

Cultivar uma cultura de segurança é como cuidar de uma planta em crescimento, a qual precisa ser alimentada e regada regularmente, e não apenas quando começa a murchar. Assim, a segurança de um software deve ser considerada como um processo contínuo o qual deve ser intrínseco a todas as atividades de uma organização (ENISA, 2019). Envolvendo a colaboração de todos, desde os jardineiros (desenvolvedores e operadores) até os paisagistas (profissionais de segurança) e os proprietários do jardim (a alta administração).

5.10.1 DevSecOps: O sistema de irrigação automático

Seguindo a nossa analogia, a prática de DevSecOps é como um sistema de irrigação automática incorporado ao seu jardim de software. Ela é uma maneira de garantir que a segurança seja tratada não como uma adição de última hora, mas como uma parte essencial e integrada do desenvolvimento (IBM, 2023). Assim, tudo, desde a germinação da ideia (fase de requisitos) até a plena floração (fase de implantação) é regado com práticas de segurança. Para obter mais informações sobre o processo de DevSecOps, confira a Comunidade DevSecOps.

5.10.2 Construindo um Solo Fértil para a Segurança

É importante ressaltar também uma cultura de segurança sólida, que é como um solo fértil que suporta o crescimento saudável de sua planta (software). Em que para se ter um desenvolvimento seguro apenas ferramentas e boas práticas não são suficientes, é necessário também uma equipe treinada e dedicada para como o processo, assumindo as responsabilidades e seguindo as diretrizes conforme estabelecidas (Tony Rice et al., 2018). Isso que, infelizmente, demanda muitas vezes um tempo considerável, as quais muitas empresas acabam ignorando, sendo por necessidade ou às vezes por escolha.

5.10.3 Diretrizes para Cultivar uma Cultura de Segurança

Assim, para manter o seu jardim de software seguro e florescente, aqui estão algumas recomendações:

- Promova treinamentos de segurança entre as equipes. Isso pode ser feito através de plataformas gratuitas como o Cybrary ou certificações e treinamentos pagos como CompTia Security+.
- Trate a segurança como um nutriente essencial, incorporando-a em todas as fases do ciclo de vida do desenvolvimento de software (ENISA, 2019).

- Incentive a colaboração e comunicação entre as equipes de desenvolvimento, operações e segurança. Evite times em silos, o que dificulta a identificação e solução de ameaças.
- Reflita a cerca da cultura da sua empresa/equipe e como ela se encaixaria melhor com essas boas praticas e processos (Tony Rice et al., 2018).

Com isso, ao seguir essas orientações, você estará um passo a frente para garantir que a segurança seja mais do que apenas um repelente de insetos aplicado após o surgimento de pragas (ameaças) e ao em vez disso, torna-se uma parte integrante do ecossistema, contribuindo para o crescimento saudável e a resiliência do seu software IoT.

5.11 Usando listas e modelos de segurança

Os frameworks e padrões de segurança fornecem um conjunto de melhores práticas para garantir a segurança do software.

5.11.1 Discussão sobre o uso do CWE em um cenário de IoT

Ao longo do SDLC, podemos utilizar de diversas práticas e documentos complementares para auxiliar na nossa busca por um software mais seguro. Nessa seção iremos explorar como usar o CWE, a partir de um exemplo. Como estamos no contexto de IoT, iremos abordar como exemplo um dispositivo que coleta dados de temperatura e umidade e os envia para um servidor na nuvem para processamento e análise. Assim, nesse caso podemos usar o CWE (Common Weakness Enumeration), uma lista de fraquezas comuns de segurança no desenvolvimento de software, da seguinte maneira:

- **Compreendendo o CWE**: O CWE é uma lista de fraquezas comuns que podem comprometer a segurança de um software durante o seu desenvolvimento. Cada fraqueza é identificada por um ID único e é acompanhada de uma descrição detalhada, possíveis consequências se a fraqueza for explorada e potenciais mitigações (CWE, 2023). Mais detalhes podem ser encontrados no site oficial da CWE.
- **Identifique as fraquezas relevantes**: No nosso caso, algumas fraquezas relevantes podem ser CWE-20 (Validação de entrada inadequada), CWE-319 (Dados sensíveis transmitidos em texto simples) e CWE-326 (Criptografia insuficientemente forte). Essas fraquezas são relevantes porque nosso sistema envolve a coleta de dados (entrada), transmissão de dados para a nuvem (dados em trânsito) e possivelmente o armazenamento de dados sensíveis, como informações de login do usuário (dados sensíveis) (CWE, 2023).

- **Entenda as potenciais mitigações**: Para cada fraqueza, a CWE fornece potenciais mitigações para cada fase do Ciclo de Vida do Desenvolvimento de Software (SDLC). Por exemplo, para a CWE-20, uma mitigação na fase de design pode ser "Use uma lista de permissões de entradas aceitáveis". Na fase de implementação, uma mitigação pode ser "Verifique se a entrada atende aos critérios de formato antes de usá-la" (CWE, 2023).

- **Aplique as mitigações**: Aplique as mitigações relevantes na fase apropriada do SDLC. Por exemplo, durante a fase de design do nosso sistema, podemos decidir que só aceitaremos leituras de temperatura entre -50 e 50 graus Celsius, e leituras de umidade entre 0 e 100%. Durante a fase de implementação, podemos adicionar verificações de código para garantir que as leituras estejam dentro desses limites antes de processá-las (ENISA, 2019).

- **Revise e repita**: A segurança do software é um processo contínuo, à medida que novas fraquezas são descobertas e adicionadas à CWE, ou à medida que nosso sistema evolui, devemos revisar regularmente as fraquezas relevantes e garantir que as mitigações apropriadas estejam em vigor.

5.11.2 Como utilizar o Modelo Diamond no cenário IoT

O Modelo *Diamond* é uma metodologia estratégica que auxilia na análise e na investigação de incidentes de segurança cibernética (Ver figura 21). Esse modelo se estrutura em quatro pilares fundamentais: Adversário, Capacidade, Infraestrutura e Vítima, em que cada componente representa um aspecto da intrusão, enquanto as ligações entre eles refletem as interações ocorridas durante um incidente de segurança (YUZUKA, 2023).

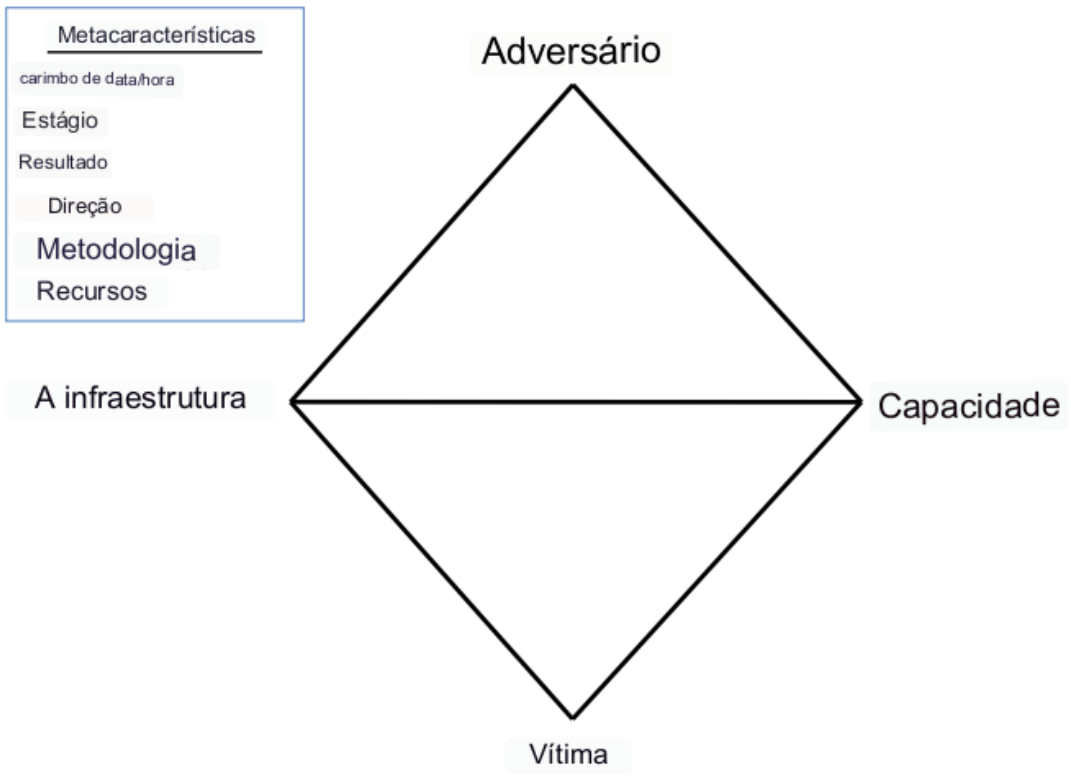


Figura 21 – Ilustração do Modelo Diamond

Fonte: Adaptado de Sergio Caltagirone, Andrew Pendergast e Christopher Betz (2013)

Agora vamos examinar como o Modelo *Diamond* pode contribuir para melhorar a segurança do nosso sistema IoT, utilizando o exemplo anterior do dispositivo de leitor de temperatura e humidade:

- **Adversário:** Inicia-se a análise entendendo possíveis interessados em comprometer o nosso sistema, que pode ser um hacker individual, um grupo de hackers ou até uma organização (ABOMHARA; KøIEN, 2015). No nosso cenário IoT, os adversários podem estar interessados em acessar os dados de temperatura e umidade para fins mal-intencionados, como a espionagem industrial.

- **Capacidade:** Este componente representa as habilidades e ferramentas que o adversário possui. Isso pode incluir conhecimento técnico, software de hacking, acesso a redes botnet, entre outros. No nosso caso, um adversário poderia ter a capacidade de lançar um ataque de negação de serviço para sobrecarregar nosso servidor ou usar técnicas de phishing para obter informações de login (ABOMHARA; KøIEN, 2015).

- **Infraestrutura:** Este componente engloba os recursos físicos e virtuais utilizados pelo adversário para conduzir o ataque. Isso pode incluir servidores, redes, domínios

e até pessoas. No contexto do nosso sistema IoT, a infraestrutura do adversário pode abranger um servidor de comando e controle para orquestrar um ataque de botnet (ABOMHARA; KøIEN, 2015).

- **Vítima:** Aqui representamos o alvo do ataque. No nosso caso, a vítima seria o nosso sistema IoT, englobando o dispositivo de coleta de dados e o servidor na nuvem. Porém, é importante ressaltar que muitas vezes os dispositivos IoT não são a vítima final e apenas um elo fraco de entrada para a rede alvo do ataque.

Agora, com uma compreensão básica do Modelo *Diamond*, podemos aplicá-lo na melhoria da segurança do nosso sistema IoT:

1. **Identifique o adversário e a capacidade:** Entender quem pode atacar nosso sistema e quais habilidades possuem permite que possamos projetar nosso sistema para resistir a esses ataques (ABOMHARA; KøIEN, 2015). Por exemplo, se sabemos que nosso adversário tem capacidade para lançar um ataque de negação de serviço, podemos projetar nosso sistema para lidar com grandes volumes de tráfego.
2. **Proteja a infraestrutura:** Entendendo a infraestrutura que o adversário pode usar para atacar nosso sistema, podemos adotar medidas para proteger nosso sistema. Por exemplo, monitorar o tráfego de rede para detectar atividades suspeitas, como tentativas de conexão a partir de endereços IP conhecidos por hospedar servidores de comando e controle.
3. **Proteja a vítima:** Identificando a vítima (nosso sistema IoT), podemos tomar medidas para protegê-la. Isso pode incluir a implementação de autenticação robusta, criptografia de dados em trânsito e em repouso, e atualizações regulares de software para corrigir vulnerabilidades de segurança.

É essencial ressaltar que o Modelo *Diamond* é apenas uma ferramenta para entender e documentar intrusões cibernéticas. Ele deve ser usado em conjunto com outras práticas de segurança, como a implementação de um programa de resposta a incidentes, treinamento de segurança para funcionários e auditorias de segurança regulares. Uma opção seria utilizar o Modelo *Diamond* de forma integrada a outros recursos de segurança, como a CWE (*Common Weakness Enumeration*) abordado anteriormente e outros instrumentos disponíveis da MITRE (como CVEs e ATT&CK), que permitem uma visão ampla e detalhada do cenário de ameaças, fortalecendo a estratégia de prevenção e mitigação.

5.12 Resumo e Lista de Verificação

Este guia apresentou uma variedade de recomendações e práticas recomendadas para cada estágio do Ciclo de Vida de Desenvolvimento de Software (SDLC). A Figura 22 abaixo proporciona uma lista de verificação sucinta dessas recomendações, ajudando a assimilar os pontos chaves e a aplicá-los prontamente:

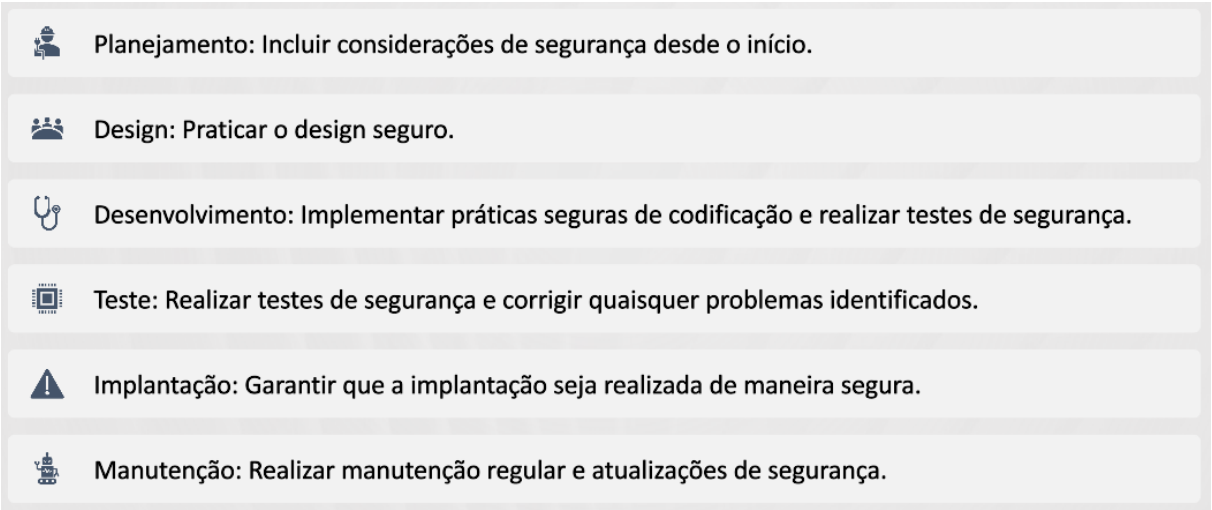


Figura 22 – Lista de Verificação Resumida

Fonte: Autor

5.13 Conclusões e Reflexões Finais

Garantir a segurança como uma prioridade não é somente uma opção, mas uma necessidade para proteger seus sistemas e dados de ameaças cibernéticas nesse ambiente caótico de IoT. E esse objetivo exige uma postura proativa e uma abordagem integrada à segurança, incorporando considerações e ferramentas de segurança em cada etapa do ciclo de vida do desenvolvimento de software.

Assim como os atacantes estão sempre evoluindo, os defensores e desenvolvedores como nós não podemos ficar parados. Por isso, é crucial que todos os participantes no processo de criação e manutenção de software estejam sempre atualizados, buscando incessantemente informações e aprendendo sobre as mais recentes tendências e avanços em segurança. Aqui estão alguns recursos, como o Instituto SANS que pode ser um excelente ponto de partida para realizar treinamentos de segurança, bem como as seguintes referências e documentos que podem servir como excelentes guias ao longo da jornada do desenvolvimento de software seguro para IoT:

- Boas Práticas para a Segurança de IoT da ENISA;
- Framework de Cibersegurança do NIST;

- Práticas Fundamentais para o Desenvolvimento Seguro de Software do SAFECode.

6 Conclusão

Este trabalho de conclusão de curso teve como tema ”Segurança na Era do IoT: Prevenção de Vulnerabilidades no Ciclo de Vida do Software”. Nesse trabalho, buscamos desenvolver diretrizes para os desenvolvedores de software prevenirem vulnerabilidades em IoT e apresentar a eles sugestões de práticas recomendadas de contramedidas para prevenir essas vulnerabilidades ao longo do ciclo de desenvolvimento de software, utilizando casos reais baseados no OWASP Top 10 IoT. Após um estudo do cenário de segurança em IoT, com ênfase no ciclo de vida do software, foram selecionadas e analisadas algumas vulnerabilidades que se enquadram neste ciclo a partir do Top 10 da OWASP.

Foi realizada uma investigação de casos reais de vulnerabilidades, tendo como preocupação observar como ocorreram e determinar quais foram as consequências para que alcançássemos o objetivo do nosso estudo: produzir recomendações de contramedidas para prevenir essas vulnerabilidades. Após essa investigação, foi possível constatar que a prevenção eficaz de vulnerabilidades pode ser obtida por meio de uma combinação de práticas recomendadas, com foco na segurança desde o início do ciclo de desenvolvimento. Ao analisar casos reais de vulnerabilidades e suas consequências, identificamos temas comuns e pontos de falha que levaram à sua ocorrência. Notou-se uma tendência de subestimar ou ignorar a segurança no início do ciclo de desenvolvimento, resultando em vulnerabilidades que foram exploradas em estágios posteriores. Com a utilização de pesquisa documental e análise bibliográfica, apresentamos uma série de recomendações, a partir de boas práticas usadas pelos desenvolvedores de software (SDLC) de dispositivos IoT, para evitar essas vulnerabilidades. Essas recomendações se tornam valiosas para os desenvolvedores, ajudando-os a criar um ambiente de IoT mais seguro. Porém, este estudo tem suas limitações, pois se baseou principalmente nos casos documentados selecionados, o que pode não abranger todas as possíveis vulnerabilidades ou cenários de IoT no Ciclo de Vida do Software.

O estudo feito, contribui para a área de segurança de software e IoT, fornecendo uma visão útil de como as vulnerabilidades podem ser prevenidas no ciclo de desenvolvimento de software. No entanto, reconhecemos que as tecnologias e as ameaças de segurança estão em constante evolução, portanto, é necessário um esforço contínuo para manter as orientações atualizadas. Para trabalhos futuros, sugerimos que se expanda este estudo para incluir vulnerabilidades emergentes de IoT e as melhores práticas para preveni-las. Além disso, seria interessante explorar a aplicação prática das nossas recomendações em cenários reais de desenvolvimento e testar sua eficácia na prevenção de vulnerabilidades.

Referências

ABOMHARA, M.; KøIEN, G. Cyber Security and the Internet of Things: Vulnerabilities, Threats, Intruders and Attacks. *Journal of Cyber Security*, v. 4, p. 65–88, maio 2015. Disponível em: <https://www.researchgate.net/publication/277718176_Cyber_Security_and_the_Internet_of_Things_Vulnerabilities_Threats_Intruders_and_Attacks>. Citado 2 vezes nas páginas 97 e 98.

ALI, J. *Validating Leaked Passwords with k-Anonymity*. 2018. Disponível em: <http://blog.cloudflare.com/validating-leaked-passwords-with-k-anonymity/>. Citado na página 57.

ANTONAKAKIS, M. et al. Understanding the Mirai Botnet. 2017. Citado 9 vezes nas páginas 52, 53, 54, 55, 57, 58, 71, 77 e 80.

ATZORI, L.; IERA, A.; MORABITO, G. The Internet of Things: A survey. *Computer Networks*, v. 54, n. 15, p. 2787–2805, 2010. Publisher: Elsevier. Citado na página 28.

BBC. German parents told to destroy Cayla dolls over hacking fears. *BBC News*, fev. 2017. Disponível em: <https://www.bbc.com/news/world-europe-39002142>. Citado 5 vezes nas páginas 60, 61, 62, 63 e 72.

Blake E. Strom et al. *MITRE ATT&CK: Design and Philosophy*. The MITRE Corporation, 2020. Disponível em: <https://attack.mitre.org/docs/ATTACK_Design_and_Philosophy_March_2020.pdf>. Citado na página 37.

BOUQUE, P.; FAIRLEY, R. E. D. (Ed.). *IEEE Computer Society Software Engineering Body of Knowledge (SWEBOK) Guide V3*. [S.l.]: IEEE computer society, 2014. v. 3. Citado 2 vezes nas páginas 40 e 41.

BRAGA, M. *More Security Vulnerabilities Found in Hello Barbie Toy's Servers*. 2016. Disponível em: <https://www.vice.com/en/article/4xav93/more-security-vulnerabilities-found-in-hello-barbie-toys-servers>. Citado 6 vezes nas páginas 65, 66, 67, 68, 72 e 83.

Bundesnetzagentur. *Bundesnetzagentur - Homepage - Milestones*. 2023. Disponível em: <https://www.bundesnetzagentur.de/EN/General/Bundesnetzagentur/AboutUs/faq_Cronik-table.html#FAQ48482>. Citado na página 62.

CADZOW, E. *Financial Impact of Mirai DDoS Attack on Dyn Revealed in New Data*. 2019. Disponível em: <https://www.corero.com/financial-impact-of-mirai-ddos-attack-on-dyn-revealed-in-new-data/>. Citado na página 54.

Cisco. *Cisco Annual Internet Report - Cisco Annual Internet Report (2018–2023) White Paper - Cisco*. [S.l.], 2020. Disponível em: <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html>. Citado na página 13.

CVE. *Overview | CVE*. 2023. Disponível em: <https://www.cve.org/About/Overview>. Citado na página 36.

CWE. *CWE - CWE-20: Improper Input Validation (4.12)*. 2023. Disponível em: <https://cwe.mitre.org/data/definitions/20.html>. Citado 3 vezes nas páginas 37, 95 e 96.

DORSEMAINE, B. et al. Internet of Things: A Definition & Taxonomy. In: *2015 9th International Conference on Next Generation Mobile Applications, Services and Technologies*. [S.l.: s.n.], 2015. p. 72–77. Citado 2 vezes nas páginas 21 e 22.

Eclipse. 2022 IoT & Edge Developer Survey Report. 2022. Citado 2 vezes nas páginas 26 e 27.

ENISA. *Good Practices for Security of IoT - Secure Software Development Lifecycle*. 2019. Disponível em: <https://www.enisa.europa.eu/publications/good-practices-for-security-of-iot-1>. Citado 38 vezes nas páginas 14, 31, 43, 44, 48, 50, 53, 56, 57, 58, 59, 61, 63, 65, 66, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 83, 84, 86, 87, 88, 89, 90, 91, 92, 94 e 96.

ENISA. Topic, *Internet of Things (IoT)*. 2023. Disponível em: <https://www.enisa.europa.eu/topics/iot-and-smart-infrastructures/iot>. Citado na página 21.

Ericson mobile. *Ericson report IoT*. [S.l.], 2022. Disponível em: <https://www.ericsson.com/49d3a0/assets/local/reports-papers/mobility-report/documents/2022/ericsson-mobility-report-june-2022.pdf>. Citado na página 13.

Forbrukerradet. *#Toyfail report*. [S.l.], 2016. 38 p. Disponível em: <https://fil.forbrukerradet.no/wp-content/uploads/2016/12/toyfail-report-desember2016.pdf>. Citado 4 vezes nas páginas 64, 65, 68 e 69.

GARCIA-MORCHON, O.; KUMAR, S.; SETHI, M. *Internet of Things (IoT) Security: State of the Art and Challenges*. RFC Editor, 2019. Issue: 8576 Num Pages: 50 Series: Request for Comments Published: RFC 8576. Disponível em: <https://www.rfc-editor.org/info/rfc8576>. Citado na página 92.

Gartner. *Employees Likely to Work Remotely Post COVID-19*. 2020. Disponível em: <https://www.gartner.com/en/newsroom/press-releases/2020-04-14-gartner-hr-survey-reveals-41--of-employees-likely-to->. Citado na página 13.

GEBREMICHAEL, T. et al. Security and Privacy in the Industrial Internet of Things: Current Standards and Future Challenges. *IEEE Access*, v. 8, p. 152351–152366, 2020. ISSN 2169-3536. Citado na página 59.

Google. *Estratégias de implantação e teste de aplicativos | Centro de arquitetura do Cloud*. 2023. Disponível em: <https://cloud.google.com/architecture/application-deployment-and-testing-strategies?hl=pt-br>. Citado 2 vezes nas páginas 88 e 92.

HADNAGY, C. *Social Engineering The Science of Human Hacking*. 2. ed. [S.l.]: Wiley, 2018. Citado na página 29.

HOWARD, M.; LEBLANC, D. *Writing Secure Code*. [S.l.]: Microsoft Press, 2002. Citado na página 28.

IBM. *What is DevOps? | IBM*. 2023. Disponível em: <https://www.ibm.com/topics/devops>. Citado 3 vezes nas páginas 44, 45 e 94.

IEEE Computer Society. *IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 15.1: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personal Area Networks (WPANs)*. 2005. Disponível em: <https://standards.ieee.org/standard/802_15_1-2005.html>. Citado na página 21.

IEEE Computer Society. *IEEE Standard for Ethernet*. 2018. Disponível em: <https://standards.ieee.org/standard/802_3-2018.html>. Citado na página 21.

IEEE Computer Society. *IEEE Standard for Information Technology–Telecommunications and Information Exchange Between Systems Local and Metropolitan Area Networks–Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. 2020. Disponível em: <https://standards.ieee.org/standard/802_11-2020.html>. Citado na página 20.

IEEE Computer Society. *IEEE Standard for Low-Rate Wireless Networks*. 2020. Disponível em: <https://standards.ieee.org/standard/802_15_4-2020.html>. Citado na página 21.

JACKSON, M. *Securing your SDLC (Software Development Life Cycle)*. 2022. Disponível em: <https://blog.gitguardian.com/securing-your-sdlc-software-development-life-cycle/>. Citado 2 vezes nas páginas 39 e 40.

KIZZA, J. M. *Guide to Computer Network Security*. London: Springer London, 2015. (Computer Communications and Networks). ISBN 978-1-4471-6653-5 978-1-4471-6654-2. Disponível em: <https://link.springer.com/10.1007/978-1-4471-6654-2>. Citado 5 vezes nas páginas 19, 28, 29, 30 e 31.

KOLIAS, C. et al. DDoS in the IoT: Mirai and other botnets. *Computer*, v. 50, p. 80–84, jan. 2017. Disponível em: <https://www.researchgate.net/publication/318288727_DDoS_in_the_IoT_Mirai_and_other_botnets>. Citado 2 vezes nas páginas 55 e 58.

KONHEIM, A. G. *COMPUTER SECURITY AND CRYPTOGRAPHY*. [S.l.]: Wiley-interscience, 2007. Citado na página 28.

Kurose, Ross. *Redes de computadores e a internet uma abordagem top-down*. 6a edição. ed. [S.l.]: Pearson, 2014. Citado 4 vezes nas páginas 13, 20, 23 e 24.

Laura Hautala. *Hello Barbie: She's just insecure*. 2015. Disponível em: <https://www.cnet.com/news/privacy/hello-headaches-barbie-of-the-internet-age-has-even-more-security-flaws/>. Citado 2 vezes nas páginas 66 e 67.

Markus Reuter. *Schnüffelpuppe „My Friend Cayla" in Deutschland verboten*. 2017. Disponível em: <https://netzpolitik.org/2017/schnueffelpuppe-my-friend-cayla-in-deutschland-verboten/>. Citado na página 62.

Martin Fowler. *bliki: BlueGreenDeployment*. 2010. Disponível em: <https://martinfowler.com/bliki/BlueGreenDeployment.html>. Citado na página 88.

Martin Fowler. *bliki: CanaryRelease*. 2014. Disponível em: <https://martinfowler.com/bliki/CanaryRelease.html>. Citado na página 88.

MARWEDEL, P. *Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems, and the Internet of Things*. Cham: Springer International Publishing, 2021. (Embedded Systems). ISBN 978-3-030-60909-2 978-3-030-60910-8. Disponível em: <http://link.springer.com/10.1007/978-3-030-60910-8>. Citado 3 vezes nas páginas 13, 22 e 23.

Microsoft. *Microsoft Security Development Lifecycle Practices*. 2023. Disponível em: <https://www.microsoft.com/en-us/securityengineering/sdl/practices>. Citado 2 vezes nas páginas 41 e 42.

MITRE. *Who We Are*. 2023. Disponível em: <https://www.mitre.org/who-we-are>. Citado na página 36.

MQTT. 2023. Disponível em: <https://mqtt.org/faq/>. Citado na página 24.

Nupur Pal. *30 IoT Frameworks that you must Look Forward to in 2023*. 2022. Disponível em: <https://www.cronj.com/blog/top-iot-frameworks/>. Citado 2 vezes nas páginas 81 e 88.

NVD. *NVD - Home*. 2023. Disponível em: <https://nvd.nist.gov/>. Citado na página 37.

OAKLEY, N. *Watch children's doll quote 50 Shades and Silence of the Lambs - it's creepy*. 2015. Section: Technology. Disponível em: <http://www.mirror.co.uk/news/technology-science/technology/friend-cayla-doll-can-hacked-5110112>. Citado 3 vezes nas páginas 61, 62 e 63.

OWASP. *OWASP IoT Top10 2018*. [S.l.], 2018. Disponível em: <https://owasp.org/www-pdf-archive/OWASP-IoT-Top-10-2018-final.pdf>. Citado 16 vezes nas páginas 31, 32, 33, 34, 35, 36, 48, 51, 52, 56, 57, 60, 63, 65, 71 e 75.

OWASP. *OWASP Foundation, the Open Source Foundation for Application Security | OWASP Foundation*. 2023. Disponível em: <https://owasp.org/>. Citado 3 vezes nas páginas 15, 30 e 71.

PINHEIRO, J. M. S. *Artigo - O Modelo OSI*. 2004. Disponível em: <https://www.projetoderedes.com.br/artigos/artigo_modelo_osi.php>. Citado na página 18.

Red Hat. *What is a REST API?* 2020. Disponível em: <https://www.redhat.com/en/topics/api/what-is-a-rest-api>. Citado 2 vezes nas páginas 24 e 25.

REZ, S. *Redes Locais Sem Fio "WLAN"*. 2018. Disponível em: <https://secbitrez.wordpress.com/2018/09/05/redes-locais-sem-fio-wlan/>. Citado na página 19.

SADEGHI, A.-R.; WACHSMANN, C.; WAIDNER, M. Security and privacy challenges in industrial Internet of Things. In: *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. [S.l.: s.n.], 2015. p. 1–6. ISSN: 0738-100X. Citado na página 59.

Sergio Caltagirone; Andrew Pendergast; Christopher Betz. *The Diamond Model of Intrusion Analysis*. 2013. Disponível em: <https://apps.dtic.mil/sti/pdfs/ADA586960.pdf>. Citado 2 vezes nas páginas 38 e 97.

SHELBY, Z.; HARTKE, K.; BORMANN, C. *The Constrained Application Protocol (CoAP)*. [S.l.], 2014. Num Pages: 112. Disponível em: <https://datatracker.ietf.org/doc/rfc7252>. Citado na página 24.

Soma Bandyopadhyay et al. Role Of Middleware For Internet Of Things: A Study. *International Journal of Computer Science & Engineering Survey*, v. 2, n. 3, p. 94–105, ago. 2011. ISSN 09763252. Disponível em: <http://www.airccse.org/journal/ijcses/papers/0811cses07.pdf>. Citado 2 vezes nas páginas 25 e 26.

Somerset Recon. *Hello Barbie Security: Part 1 - Teardown*. 2015. Disponível em: <https://www.somersetrecon.com/blog/2015/11/20/hello-barbie-security-part-1-teardown>. Citado na página 64.

Somerset Recon. *Hello Barbie Security: Part 2 - Analysis*. 2016. Disponível em: <https://www.somersetrecon.com/blog/2016/1/21/hello-barbie-security-part-2-analysis>. Citado 5 vezes nas páginas 65, 66, 68, 69 e 72.

SOMMERVILLE, I. *Software engineering*. 10. ed., global ed. ed. Boston Munich: Pearson, 2016. (Always learning). ISBN 978-1-292-09613-1. Citado 4 vezes nas páginas 23, 39, 73 e 74.

SOUZA, T. *CWE (Common Weakness Enumeration), saiba o que é*. 2022. Disponível em: <https://ostec.blog/geral/cwe-common-weakness-enumeration/>. Citado na página 37.

STALLINGS, W.; BROWN, L. *Computer Security: Principles and Practice*. [S.l.]: Pearson, 2017. Citado na página 28.

Sá-SILVA, J. R.; ALMEIDA, C. D. d.; GUINDANI, J. F. Pesquisa documental: pistas teóricas e metodológicas. *Revista Brasileira de História & Ciências Sociais*, v. 1, n. 1, jul. 2009. ISSN 2175-3423. Number: 1. Disponível em: <https://periodicos.furg.br/rbhcs/article/view/10351>. Citado na página 47.

TANEMBAUM, A. *Computer Networks - A Tanenbaum - 5th edition.pdf*. 5th. ed. [S.l.: s.n.], 2011. Citado na página 20.

Tim Medin. *Doll Hacking: The Good, The Bad(words) and the Ugly (features)*. 2015. Disponível em: <https://redsiege.com/tools-techniques/2015/11/doll-hacking-good-badwords-and-ugly/>. Citado 3 vezes nas páginas 60, 63 e 87.

TIPTON, H. F.; KRAUSE, M. Information Security Management Handbook, Fifth Edition. 2004. Citado na página 28.

Tony Rice et al. *Fundamental Practices for Secure Software Development*. 2018. Disponível em: <https://safecode.org/wp-content/uploads/2018/03/SAFECode_Fundamental_Practices_for_Secure_Software_Development_March_2018.pdf>. Citado 10 vezes nas páginas 75, 76, 78, 79, 80, 81, 84, 92, 94 e 95.

Unit 42 Palo Alto. *2020 Unit 42 IoT Threat Report*. [S.l.], 2020. Disponível em: <https://unit42.paloaltonetworks.com/iot-threat-report-2020/>. Citado na página 14.

Vidhya V. Kumar. *Embracing DevOps in the IoT Era - DevOps.com*. 2016. Disponível em: <https://devops.com/embracing-devops-iot-era/>. Citado na página 46.

WELLS, D. *Extreme Programming: A Gentle Introduction*. 2013. Disponível em: <http://www.extremeprogramming.org/index.html>. Citado na página 81.

WHITMAN, M.; MATTORD, H. *Principles of Information Security, 5th Edition*. [S.l.: s.n.], 2015. Citado na página 28.

XSF. *XMPP | XMPP Standards Foundation (XSF)*. 2023. Disponível em: <https://xmpp.org/about/xmpp-standards-foundation/>. Citado na página 24.

YOUSEFNEZHAD, N. et al. A Comprehensive Security Architecture for Information Management throughout the Lifecycle of IoT Products. *Sensors*, v. 23, p. 3236, mar. 2023. Citado 2 vezes nas páginas 70 e 83.

YUZUKA. *Diamond Model of Intrusion Analysis: A Quick Guide*. 2023. Disponível em: <https://securityboulevard.com/2023/03/diamond-model-of-intrusion-analysis-a-quick-guide/>. Citado 2 vezes nas páginas 38 e 96.

ZIMMERMANN, H. OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection. *IEEE Transactions on Communications*, v. 28, p. 425–432, 1980. Citado na página 17.