

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

**Aplicação dos Requisitos de Segurança OWASP
no Desenvolvimento de Aplicações Móveis sob
o Ponto de Vista de um Desenvolvedor Júnior:
um Estudo de Caso**

Autor: Estevão de Jesus Reis
Orientadora: Prof.^a Dr.^a Elaine Venson

Brasília, DF
2023



Estevão de Jesus Reis

Aplicação dos Requisitos de Segurança OWASP no Desenvolvimento de Aplicações Móveis sob o Ponto de Vista de um Desenvolvedor Júnior: um Estudo de Caso

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.^a Dr.^a Elaine Venson

Brasília, DF

2023

Estevão de Jesus Reis

Aplicação dos Requisitos de Segurança OWASP no Desenvolvimento de Aplicações Móveis sob o Ponto de Vista de um Desenvolvedor Júnior: um Estudo de Caso/ Estevão de Jesus Reis. – Brasília, DF, 2023-

100 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof.^a Dr.^a Elaine Venson

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade UnB Gama – FGA , 2023.

1. Owasp Top Ten. 2. Segurança de Software. I. Prof.^a Dr.^a Elaine Venson. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Aplicação dos Requisitos de Segurança OWASP no Desenvolvimento de Aplicações Móveis sob o Ponto de Vista de um Desenvolvedor Júnior: um Estudo de Caso

CDU

Estevão de Jesus Reis

Aplicação dos Requisitos de Segurança OWASP no Desenvolvimento de Aplicações Móveis sob o Ponto de Vista de um Desenvolvedor Júnior: um Estudo de Caso

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.

Trabalho aprovado. Brasília, DF, :

Prof.^a Dr.^a Elaine Venson
Orientador

Prof. Dr. John Lenon Gardenghi
Convidado 1

Prof.^a Dr.^a Milene Serrano
Convidado 2

Brasília, DF
2023

Agradecimentos

Agradeço a Deus por ser minha força e sustento de cada dia. Sei que em cada fase da minha vida Ele está presente, me ajudando e me encorajando a vencer cada obstáculo que surge.

À minha família e, em especial, aos meus pais, por todo o apoio, suporte, carinho e compreensão em todo esse período da graduação. Sei que sem eles eu não poderia ter chegado tão longe.

Agradeço à minha noiva Vanessa por sempre me lembrar que já venci desafios que pensava não ser capaz. Em suas palavras encontrei, por diversas vezes, a força e coragem necessária para vencer mais uma vez.

Agradeço à minha orientadora, a Prof.^a Dr.^a Elaine Venson, pela dedicação e compromisso em me orientar durante todo o processo de desenvolvimento deste trabalho. Sua gentileza e cuidado ao esclarecer minhas dúvidas não apenas facilitaram o desenvolvimento, mas também tornaram o processo mais leve.

Aos amigos e colegas, dentro e fora da graduação, agradeço por tornarem esse período mais leve. Eles me ajudaram a superar momentos de estresse, ansiedade e medo, proporcionando descontração, paz e felicidade, lembrando-me que às vezes menos é mais.

Resumo

Os usuários de dispositivos móveis vem crescendo consideravelmente, visto que atualmente é possível executar diversas tarefas em um *smartphone*. Consequentemente, o mercado de desenvolvimento de aplicativos móveis tem se expandido nas últimas décadas. Com essa expansão, surge a necessidade de se pensar em segurança de software, para que sejam desenvolvidos aplicativos seguros e resistente contra ataques cibernéticos, diminuindo os riscos de roubo de dados, invasão de privacidade, entre outros. Porém, uma grande quantidade de desenvolvedores de software não tem conhecimento a respeito das implicações da segurança de software ou dos métodos de codificação segura. Um dos principais motivos para a existência de vulnerabilidades de software é a falta de conhecimento e experiência a respeito dos métodos de codificação segura na fase de desenvolvimento do software. A fundação OWASP definiu, em 2016, uma lista chamada *Top 10 Mobile Risks*, com as 10 principais vulnerabilidades encontradas em um dispositivo móvel. Buscando entender as dificuldades enfrentadas pelos desenvolvedores de aplicações móveis na aplicação de práticas de segurança para desenvolvimento em dispositivos móveis, este trabalho tem por objetivo explorar a aplicação dos requisitos de segurança de software definidos pela OWASP do ponto de vista de um desenvolvedor júnior, buscando entender qual é esforço e quais são os conhecimentos necessários para aplicá-las. Para tanto, será utilizada uma metodologia descritiva com o emprego da técnica Estudo de Caso.

Palavras-chave: OWASP Top Ten; segurança de software; desenvolvimento mobile; vulnerabilidades de software; práticas de segurança de software; requisitos de segurança.

Abstract

Mobile device users have been growing considerably, as it is currently possible to perform several tasks on a *smartphone*. According to [Turner \(2022\)](#), the number of people using *smartphones* exceeds 6.5 billion, which is more than 83% of the current world population. Most of these tasks are performed by applications developed for this type of device. Consequently, the mobile application development market has expanded in recent decades. With this expansion, there is a need to think about software security, so that applications are developed that are increasingly safe and resistant against cyber attacks, reducing the risks of data theft, invasion of privacy, among others. However, a large amount of software developers are unaware of the implications of software security or secure coding methods. One of the main reasons for the existence of software vulnerabilities is the lack of knowledge and experience regarding secure coding methods in the software development phase. The OWASP([OWASP, 2022](#)) foundation defined, in 2016, a list called *Top 10 Mobile Risks*, with the top 10 vulnerabilities found in a mobile device. Seeking to understand the difficulties faced by mobile application developers in applying security practices for development on mobile devices, this work aims to explore the application of software security requirements by OWASP from the point of view of a junior developer, seeking to understand which is effort and what knowledge is needed to apply them. To this end, a descriptive methodology will be used using the Case Study technique.

Key-words: Owasp Top Ten, software security, mobile development, software vulnerabilities, software security practices, software security requirements.

Lista de ilustrações

| | |
|---|----|
| Figura 1 – Níveis de requisito de segurança. Fonte: Mobile Application Security Verification Standard - MASVS. | 50 |
| Figura 2 – Implementação da classe responsável por armazenar dados sensíveis da aplicação de forma segura.Fonte: Criado pelo Autor | 60 |
| Figura 3 – Utilização da classe StoragaService no armazenamento do token de acesso. Fonte: Elaborado pelo Autor | 61 |
| Figura 4 – Implementação do requisito de segurança STORAGE 5 e 7. | 62 |
| Figura 5 – Cadastro do usuário fazendo uso do método de criptografia hashing para armazenar a senha do usuário. Fonte: Elaborado pelo Autor | 63 |
| Figura 6 – Validação do usuário, comparando a senha informada e a armazenada no banco de dados. Fonte: Elaborado pelo Autor | 64 |
| Figura 7 – Implementação da classe JWT responsável por criar o token de acesso. Fonte: Elaborado pelo Autor | 65 |
| Figura 8 – Implementação da interface utilizada na classe JWT responsável por criar o token de acesso. Fonte: Elaborado pelo Autor | 66 |
| Figura 9 – Implementação de login da aplicação gerando o token de acesso da sessão do usuário. Fonte: Elaborado pelo Autor | 67 |
| Figura 10 – Implementação do encerramento de sessão dentro do aplicativo móvel. Fonte: Elaborado pelo Autor | 68 |
| Figura 11 – Implementação da validação da senha no cadastro do usuário. Fonte: Elaborado pelo Autor | 69 |
| Figura 12 – Implementação do mecanismo para proteger contra o envio de credenciais em um número excessivo no terminal remoto. Fonte: Elaborado pelo Autor | 75 |
| Figura 13 – Implementação da configuração que invalida os tokens de acesso gerados depois de 24 horas. Fonte: Elaborado pelo Autor | 76 |
| Figura 14 – Exemplo de utilização de validação em um input de texto dentro da aplicação. Fonte: Elaborado pelo Autor | 76 |
| Figura 15 – Implementação da função de validação de email. Fonte: Elaborado pelo Autor | 77 |
| Figura 16 – Tela da aplicação mostrando o funcionamento das validações das entradas de texto. Fonte: Elaborado pelo Autor | 78 |
| Figura 17 – Correlação entre as principais vulnerabilidades e os grupos de requisitos de segurança. Fonte: Elaborado pelo Auto. | 79 |
| Figura 18 – Comparação de tempo de implementação entre os requisitos. Fonte: Elaborado pelo Autor. | 80 |

Figura 19 – Comparação de tempo de implementação entre os requisitos. Fonte:
Elaborado pelo Autor. 81

Figura 20 – Comparação de tempo de implementação entre os requisitos. Fonte:
Elaborado pelo Autor. 81

Lista de tabelas

| | |
|--|-----|
| Tabela 1 – Etapas do Protocolo de Estudo de Caso (BRERETON et al., 2008) . . . | 43 |
| Tabela 2 – V1 - Requisitos de Arquitetura, <i>Design</i> e Modelagem de Ameaças (OWASP et al., 2022) | 93 |
| Tabela 3 – V2 - Requisitos de Armazenamento de Dados e Privacidade (OWASP et al., 2022) | 94 |
| Tabela 4 – V3 - Requisitos de Criptografia (OWASP et al., 2022) | 95 |
| Tabela 5 – V4 - Requisitos de Autenticação e Gerenciamento de Sessão (OWASP et al., 2022) | 96 |
| Tabela 6 – V5 - Requisitos de Comunicação de Rede (OWASP et al., 2022) | 97 |
| Tabela 7 – V6 - Requisitos de Interação com a Plataforma (OWASP et al., 2022) . | 98 |
| Tabela 8 – V7 - Requisitos de Qualidade de Código e Configuração do Compilador (OWASP et al., 2022) | 99 |
| Tabela 9 – V8 - Requisitos de Resiliência (OWASP et al., 2022) | 100 |

Lista de abreviaturas e siglas

| | |
|-------|--|
| 2FA | <i>Two Factor Authentication</i> |
| API | <i>Application Programming Interface</i> |
| CA | <i>Certificate Authority</i> |
| DDoS | <i>Distributed Denial of Service</i> |
| HTTP | <i>Hypertext Transfer Protocol</i> |
| IPC | <i>Inter-Process Communication</i> |
| JWT | <i>JSON Web Token</i> |
| LDAP | <i>Lightweight Directory Access Protocol</i> |
| MASVS | <i>Mobile Application Security Verification Standard</i> |
| MD | <i>Message-Digest</i> |
| NFC | <i>Near Field Communication</i> |
| RC2 | <i>Rivest Cipher 2</i> |
| SHA1 | <i>Secure Hash Algorithm</i> |
| SMS | <i>Short Message Service</i> |
| SO | <i>Sistema Operacional</i> |
| SQL | <i>Standard Query Language</i> |
| TLS | <i>Transport Layer Security</i> |
| UI | <i>User interface</i> |
| XML | <i>Extensible Markup Language</i> |
| WiFi | <i>Wireless Fidelity</i> |

Sumário

| | | |
|------------|---|-----------|
| 1 | INTRODUÇÃO | 21 |
| 1.1 | Contexto | 21 |
| 1.2 | Problema | 22 |
| 1.3 | Objetivos | 23 |
| 1.4 | Metodologia | 23 |
| 1.5 | Organização | 24 |
| 2 | REFERENCIAL TEÓRICO | 27 |
| 2.1 | Segurança de Software | 27 |
| 2.2 | Principais Vulnerabilidades em Aplicativos Móveis | 28 |
| 2.2.1 | M1 - Uso Indevido da Plataforma | 29 |
| 2.2.2 | M2 - Armazenamento de Dados Inseguro | 30 |
| 2.2.3 | M3 - Comunicação Insegura | 31 |
| 2.2.4 | M4 - Autenticação Insegura | 32 |
| 2.2.5 | M5 - Criptografia Insuficiente | 33 |
| 2.2.6 | M6 - Autorização Insegura | 34 |
| 2.2.7 | M7 - Qualidade Pobre de Código | 35 |
| 2.2.8 | M8 - Adulteração de Código | 36 |
| 2.2.9 | M9 - Engenharia Reversa | 37 |
| 2.2.10 | M10 - Funcionalidade Estranha | 37 |
| 2.3 | Práticas de Segurança no Desenvolvimento de Aplicativos Móveis | 38 |
| 3 | METODOLOGIA | 41 |
| 3.1 | Considerações Iniciais | 41 |
| 3.2 | Planejamento de Pesquisa | 41 |
| 3.3 | Plano Metodológico | 42 |
| 3.4 | Coleta de Dados | 42 |
| 3.5 | Análise dos Resultados | 44 |
| 3.6 | Atividades Realizadas | 45 |
| 3.6.1 | Pesquisa Bibliográfica | 45 |
| 3.6.2 | Pesquisa Documental | 45 |
| 3.6.3 | Etapas do Protocolo de Estudo de Caso | 45 |
| 4 | ESTUDO DE CASO | 49 |
| 4.1 | Considerações Iniciais | 49 |
| 4.2 | Escopo do Projeto | 49 |

| | | |
|------------|--|-----------|
| 4.3 | Requisitos de Segurança de Software | 49 |
| 4.4 | Tecnologias Utilizadas | 53 |
| 4.4.1 | Dart | 54 |
| 4.4.2 | Flutter | 54 |
| 4.4.3 | JavaScript | 54 |
| 4.4.4 | NodeJs | 54 |
| 4.4.5 | NestJs | 55 |
| 4.5 | Sobre o Desenvolvedor | 55 |
| 4.6 | Metodologia de Desenvolvimento | 55 |
| 4.7 | Considerações Finais | 55 |
| 5 | ANÁLISE DA IMPLEMENTAÇÃO DOS REQUISITOS | 57 |
| 5.1 | Análise da implementação de cada requisito de segurança | 57 |
| 5.1.1 | ARCH-2 | 57 |
| 5.1.2 | ARCH-3 | 58 |
| 5.1.3 | ARCH-4 | 58 |
| 5.1.4 | STORAGE-1 | 59 |
| 5.1.5 | STORAGE-3 | 59 |
| 5.1.6 | STORAGE-5 | 61 |
| 5.1.7 | STORAGE-7 | 62 |
| 5.1.8 | CRYPTO-1 | 62 |
| 5.1.9 | CRYPTO-2 | 64 |
| 5.1.10 | CRYPTO-3 | 64 |
| 5.1.11 | AUTH-3 | 65 |
| 5.1.12 | AUTH-4 | 65 |
| 5.1.13 | AUTH-5 | 67 |
| 5.1.14 | AUTH-6 | 68 |
| 5.1.15 | AUTH-7 | 70 |
| 5.1.16 | AUTH-12 | 70 |
| 5.1.17 | PLATFORM-2 | 70 |
| 5.1.18 | CODE-5 | 71 |
| 5.2 | Correlação entre as vulnerabilidades e os grupos de requisitos de segurança | 72 |
| 5.3 | Análise dos conhecimentos | 72 |
| 5.4 | Análise do tempo gasto na aplicação dos requisitos por implementação | 73 |
| 5.5 | Tempo de Implementação dos Requisitos de Segurança e da Aplicação | 73 |
| 5.6 | Considerações Finais | 73 |
| 6 | CONCLUSÃO | 83 |

| | | |
|------------|--|-----------|
| 6.1 | Proposta para trabalhos futuros | 84 |
| | REFERÊNCIAS | 87 |
| | APÊNDICES | 91 |
| | APÊNDICE A – LISTA DE REQUISITOS DE SEGURANÇA PARA APLICATIVOS MÓVEIS | 93 |

1 Introdução

1.1 Contexto

A cada dia que passa, novas tecnologias são desenvolvidas, evoluídas ou melhoradas. O mundo digital vem tomando seu espaço na sociedade, auxiliando setores como o financeiro, de telecomunicação, transporte, gestão, entre outros. Exemplo disso, é a criação de variadas redes sociais que revolucionou a interação social ao longo dos anos. Além disso, serviços do sistema bancário tem se modernizado, tornando possível executar inúmeras tarefas, tais como fazer pagamentos, consultar saldo, fazer transferências, entre tantas outras, direto de algum dispositivo eletrônico. Com isso, o meio digital tem assumido múltiplas responsabilidades dentro da sociedade, sendo usado para armazenar informações, inclusive pessoais e confidenciais, além de poder executar diversas tarefas para se atingir um objetivo (MOHAMMED et al., 2017). Dessa forma, é de suma importância que haja formas de tornar esse meio digital, um meio seguro e confiável para que não haja prejuízos para os que o utilizam.

Conforme Rashid et al. (2021), o conceito de risco, definido de forma ampla, é a possibilidade de que uma ação ou evento exerça impacto sobre coisas que as pessoas valorizam. Trazendo para o contexto de segurança de software, risco à segurança de software é a possibilidade de uma aplicação ser atacada, por meio de brechas ou vulnerabilidades não tratadas em sua construção. Baseado em Ransome e Misra (2013), 70% das vulnerabilidades de segurança intrinsecamente ligadas ao negócio da aplicação são descobertas no código, não nas tecnologias de redes de internet. Com base em Bione (2015), riscos à segurança de software são circunstâncias em que usuários podem encontrar maior probabilidade de serem atacados por agentes maliciosos devidos a vulnerabilidades existentes na aplicação. Essas tentativas de ataque podem acontecer a partir de configurações arriscadas, definidas por parte do usuário, que levam a riscos que poderiam ser evitados.

Segundo Goertzel et al. (2007), os atacantes de software possuem três principais objetivos: (1) sabotar o software para que ele deixe de funcionar ou para que ele não esteja disponível para o uso dos usuários; (2) corromper a aplicação, de modo que ela passe a funcionar de forma diferente, modificando sua lógica ou inserindo lógicas com intenções maliciosas; e (3), invadir o software para que possam estudá-lo, entendendo a lógica inserida dentro dele e o ambiente de software em que está inserido, visando aprimorar seu funcionamento. Os dois primeiros objetivos sempre resultam na quebra da segurança do software, além de afetar diversas propriedades do software de extrema importância como a confiabilidade, desempenho, usabilidade, entre outros.

Segundo [Goertzel et al. \(2007\)](#), um software seguro é um software capaz de se defender da maioria dos ataques a que é submetido. Quando isso não é possível, é capaz de tolerá-los; e para os poucos ataques que não conseguiu resistir, é capaz de se regenerar de forma rápida dos poucos danos que recebeu com essa menor quantidade de ataques. Somando-se a isso, ele complementa dizendo que um software seguro é um software que, em sua construção, foi desenvolvido de forma robusta contra ataques. Dessa forma, o software se manterá confiável mesmo quando esta característica estiver sendo ameaçada. Ele conclui dizendo que, um software seguro não possui pontos de vulnerabilidades que podem dar brechas para que invasores ou *malwares*¹ danifiquem o sistema.

Conforme [Halkidis et al. \(2008\)](#), a necessidade de se levar em consideração a temática de segurança de software vem sendo evidenciada desde que se descobriu que os maiores riscos de ataque em uma aplicação se dão pelo desenvolvimento descuidado dos mesmos, não levando em consideração inúmeras questões de segurança em sua arquitetura. Além disso, ele revela que quanto antes forem implementados requisitos de segurança em uma aplicação, menor será o esforço e, portanto, menor o custo para aplicá-los.

Os usuários de dispositivos móveis vem crescendo consideravelmente, visto que a capacidade de executar tarefas em *smartphones* cresce a cada lançamento de nova versão. De acordo com [Turner \(2022\)](#), o número de pessoas que utilizam *smartphones* supera 6,5 bilhões, sendo que esse valor é mais de 83% da população mundial. Além disso, a quantidade de pessoas que utilizam um celular minimamente inteligente e com funcionalidades já ultrapassa os 7 bilhões, representando 90% da população mundial atual, o que torna esse meio um dos principais alvos de ataques. Esses dados enfatizam a importância da aplicação de práticas de desenvolvimento seguro em aplicações móveis.

1.2 Problema

Como tratado por [Gasiba et al. \(2020\)](#), uma grande quantidade de desenvolvedores de software não tem conhecimento a respeito das implicações da segurança de software ou dos métodos de codificação segura, alegando que parte desse problema se dá pelo fato de não haver investimento em treinamentos de segurança cibernética. [Gasiba et al. \(2020\)](#) ainda afirma que, apesar de existirem ferramentas automatizadas de análise de vulnerabilidades em código, se o desenvolvedor não tem conhecimento a respeito dessas vulnerabilidades e não sabe como corrigi-las ou mitigá-las, esse tipo de ferramenta não será efetivo. Ele também cita que um dos principais motivos para a existência de uma grande quantidade de vulnerabilidades de software é a falta de conhecimento e experiência a respeito dos métodos de codificação segura na fase de desenvolvimento do software.

¹ Malware, de acordo com [Jajodia \(2006\)](#) é um software desenvolvido com a intenção de fazer mal ou que seu efeito é malicioso

A fundação OWASP, entidade sem fins lucrativos que reúne esforços para para aprimorar os aspectos de segurança de software, definiu, em 2016, uma lista chamada *Top 10 Mobile Risks*, com as dez principais vulnerabilidades encontradas em um sistema móvel, validada pela comunidade (OWASP, 2016). Nesta lista, é feito o detalhamento das vulnerabilidades, suas consequência e formas de preveni-las ou mitigá-las.

Além disso, a fundação OWASP et al. (2022), definiu no documento Mobile Application Security Verification Standard (MASVS), requisitos de segurança de software necessários para garantir a segurança de aplicações móveis.

Visando entender as dificuldades enfrentadas pelos desenvolvedores de aplicações móveis na aplicação de práticas de segurança para desenvolvimento em dispositivos móveis, considerando as vulnerabilidades mais frequentes para esta plataforma de acordo com a OWASP, a questão de pesquisa abordada neste trabalho é:

Qual é o esforço e quais são os conhecimentos necessários para aplicar os requisitos de segurança de forma a prevenir as vulnerabilidades definidas no Top 10 Mobile Risks definidos pela Owasp, do ponto de vista de um desenvolvedor júnior?

1.3 Objetivos

O objetivo geral do presente trabalho consiste em explorar a implementação dos requisitos de segurança definidos pelo *Mobile Application Security Verification Standard (MASVS)*, do ponto de vista de um desenvolvedor júnior. Como objetivos específicos, temos:

- Analisar as dez principais vulnerabilidades de aplicações em dispositivos móveis;
- Identificar os requisitos de segurança mapeadas para essas vulnerabilidades;
- Descrever a forma de implementação dessas práticas;
- Desenvolver o componente de autenticação de uma aplicação móvel utilizando as práticas sugeridas, e
- Avaliar o nível de dificuldade, conhecimentos necessários e esforço para implementação dos requisitos.

1.4 Metodologia

A metodologia deste trabalho foi definida e classificada quanto à natureza, à abordagem e aos procedimentos técnicos. A natureza pode ser identificada como aplicada, por ter como objetivo produzir conhecimento para aplicações práticas a soluções de problemas

específicos. A abordagem caracteriza-se por uma união de métodos qualitativos e quantitativos (MORESI, 2003). Para realizar este trabalho, os seguintes procedimentos técnicos foram adotados: Pesquisa Bibliográfica, Pesquisa Documental e Estudo de Caso.

Foi decidido que o caso a ser estudado será um componente de autenticação para um aplicativo móvel desenvolvido pelo próprio autor deste trabalho. Para isto, foram escolhidos a linguagem de programação Dart (GOOGLE, 2023a) em conjunto com o *framework*² Flutter (GOOGLE, 2023b), tornando possível o desenvolvimento de aplicativos móveis que podem de ser executados tanto em sistemas Android quanto iOS.

No intuito de orientar o pesquisador na execução do Estudo de Caso, foi utilizado o protocolo definido por Brereton et al. (2008), que, por sua vez, guia o pesquisador no planejamento de 11 etapas que o compõem o Estudo de Caso.

1.5 Organização

Este trabalho de conclusão de curso está estruturado em capítulos da seguinte forma:

- **Capítulo 1 - Introdução:** Neste capítulo, são apresentados o contexto, no qual está inserido este trabalho; o problema de pesquisa; os objetivos a serem alcançados, e uma síntese da metodologia utilizada.
- **Capítulo 2 - Referencial Teórico:** São abordados os principais conceitos com o objetivo de fundamentar este trabalho. O capítulo é subdividido nas sessões *Segurança de Software*, *Principais Vulnerabilidades em Aplicativos Móveis* e, por último, as *Práticas de Segurança no Desenvolvimento de Aplicativos Móveis*.
- **Capítulo 3 - Metodologia:** São descritos a abordagem metodológica, o plano de pesquisa, o plano metodológico e as atividades definidas para concretizar este trabalho.
- **Capítulo 4 - Estudo de Caso -** São descritos os elementos e informações primordiais para o entendimento da aplicação do estudo de caso.
- **Capítulo 5 - Análise da Implementação dos Requisitos:** Nesse capítulo, são apresentados os resultados obtidos a partir da aplicação do estudo de caso, bem como uma análise dos dados coletados.
- **Capítulo 6 - Conclusão:** Neste capítulo, são retomados todos os objetivos específicos deste relatório com seus respectivos resultados, e por fim é apresentada uma

² Cardoso (2021), define Framework como conjuntos de códigos de uma linguagem de programação específica que auxiliam o desenvolvimento de projetos.

conclusão sobre esta monografia, as impressões do autor e sugestões de possíveis de trabalhos futuros.

2 Referencial Teórico

2.1 Segurança de Software

A área de segurança de software busca estudar meios para que um software seja considerado seguro, capaz de resistir a ataques externos, lidando com as vulnerabilidades existentes em um software. Na visão de [Rashid et al. \(2021\)](#), um software para ser considerado seguro deve satisfazer os três principais objetivos de segurança de software: confidencialidade, integridade e disponibilidade. [Ransome e Misra \(2013\)](#) trazem a definição de cada uma dessas características, integrando ao final todas elas na definição de segurança da informação:

- **Confidencialidade:** preservação das informações restritas ao acesso, incluindo formas de proteger informações proprietárias e garantir a privacidade pessoal.
- **Integridade:** proteção da informação, de forma a prevenir a modificação ou destruição indevida da mesma, garantindo sua integridade.
- **Disponibilidade:** garantia que a informação estará disponível e acessível de forma oportuna e confiável na maior parte do tempo.
- **Segurança da informação:** proteção das informações contidas em um sistema, de modo a impedir acessos não autorizados de fazer uso, divulgar, modificar, ou destruir qualquer informação, cumprindo assim com os objetivos de confidencialidade, integridade e disponibilidade de um sistema.

De acordo com [Ransome e Misra \(2013\)](#), a *confidencialidade* é assegurada quando o software garante que quem está de posse da informação é alguém que tem a permissão para acessá-la, impedindo o acesso de usuários, seja pessoa ou outro software, a essas informações. Dessa forma, o sistema será considerado confiável. Para suporte à confiabilidade, duas propriedades devem existir dentro de um sistema, sendo a autenticação e a autorização, definidas da seguinte forma:

- **Autenticação:** é o processo de verificar que uma entidade ou indivíduo é, de fato, quem diz ser.
- **Autorização:** é o processo de verificar se tal entidade possui papel e privilégios adequados para fazer uma requisição ou visualizar determinados dados.

Já a *integridade*, segundo [Ransome e Misra \(2013\)](#), é definida pela forma como uma aplicação recebe, envia e armazena os dados. Eles não podem ser alterados por usuários sem permissão para isso, além de garantir que as informações contidas nos dados são confiáveis em todo o processo de transação dos dados. Uma das técnicas que confere apoio a essa objetivo é a utilização de mecanismos de encriptação de dados. Baseado em [Stallings \(2006\)](#), criptografia é o estudo de técnicas para assegurar o sigilo e a veracidade das informações.

Por fim, a *disponibilidade*, com base em [Ransome e Misra \(2013\)](#), trata-se da porcentagem de tempo que uma aplicação se mantém disponível, excluindo os momentos em que foi programada uma pausa na disponibilidade do sistema, inativando-o.

2.2 Principais Vulnerabilidades em Aplicativos Móveis

Vulnerabilidade de software representa um risco encontrado dentro do software que pode afetar de alguma forma a segurança do mesmo. Todos os dias são reportadas novas categorias de vulnerabilidades de segurança em tecnologias e produtos de software. O programa *Common Vulnerabilities and Exposures (CVE, 2022)* fornece um catálogo com as principais vulnerabilidades descobertas até hoje. Caso uma nova vulnerabilidade tenha sido descoberta e tenha sido documentada, é possível adicioná-la à lista de vulnerabilidades, onde receberá um identificador único ([ZAMFIROIU, 2020](#)).

De acordo com a fundação *Open Web Application Security Project (OWASP, 2016)*, as 10 principais vulnerabilidades de segurança encontradas em aplicações móveis, listadas em 2016, são:

- M1 - Uso Indevido da Plataforma;
- M2 - Armazenamento de Dados Inseguro;
- M3 - Comunicação Insegura;
- M4 - Autenticação Insegura;
- M5 - Criptografia Insuficiente;
- M6 - Autorização Insegura;
- M7 - Qualidade Pobre de Código;
- M8 - Adulteração de Código;
- M9 - Engenharia Reversa, e
- M10 - Funcionalidade Estranha.

As subseções a seguir apresentam o detalhamento de cada uma das vulnerabilidades citadas acima.

2.2.1 M1 - Uso Indevido da Plataforma

À luz de (OWASP, 2016), esta categoria engloba a utilização de algum recurso da plataforma (Android IOS, Windows Phone) de forma imprópria, ou alguma falha no uso dos controles de segurança da plataforma. Como exemplo, podem ser inclusos intenções do Android, permissões concedidas, uso inadequado do *TouchID*¹, do *Keychain*², ou alguma outra forma de ferramenta de segurança que faça parte do sistema operacional móvel. Essa vulnerabilidade pode vir à tona através de API's que foram desenvolvidas de forma a não tratar suas vulnerabilidades. Dessa forma, um indivíduo com más intenções pode inserir entradas maliciosas no aplicativo que são mandados para a API, podendo trazer sérios riscos à segurança da aplicação como um todo, como por exemplo, o acesso à toda a base de dados do sistema. Essa vulnerabilidade possui os mesmos impactos que a vulnerabilidade do *Top Ten Web (OWASP, 2017) A1:2017-Injection*, que trata das falhas de injeção. Em conformidade a (OWASP, 2017), falhas de injeção, como injeções de SQL, SO e LDAP acontecem quando dados não confiáveis são enviados para um interpretador como parte de uma consulta autêntica ou algum comando. Esses dados podem confundir o interpretador fazendo com que ele envie dados sem a autorização necessária, ou execute comandos que não pretendia executar. Os aplicativos podem enfrentar esse risco por vários motivos como:

- **Violação das diretrizes publicadas:** Todas as plataformas compreendem diretrizes de desenvolvimento voltados para a segurança. Se um aplicativo não segue as práticas recomendadas pelo fabricante, ele estará vulnerável a esse risco.
- **Violação de convenção ou prática comum:** Nem todas as melhores práticas de desenvolvimento seguem as orientações do fabricante.
- **Uso indevido involuntário:** Alguns aplicativos cometem erros no desenvolvimento da aplicação gerando falhas, ou instalando dependências e bibliotecas não verificadas ou que estão sendo usadas para um propósito diferente do que se espera.

¹ De acordo com Cherapau et al. (2015), o Touch ID é um sensor de autenticação biométrica baseado em um scanner de impressão digital de alta definição embutido no botão de início em iPhones e iPads. Esse sensor permite que os usuários desbloqueiem seus dispositivos simplesmente tocando no botão de início

² Segundo Xing et al. (2015), o Keychain um mecanismo de gerenciamento de senhas encontrados em sistemas Mac OS, com o objetivo de proteger as credenciais do usuário. É o arquivo que mantém o espaço para armazenar contas de usuário criptografadas, pares de chaves pública/privada, certificados, senhas criptografadas de volumes e notas de segurança.

2.2.2 M2 - Armazenamento de Dados Inseguro

Conforme [OWASP \(2016\)](#), esse tipo de vulnerabilidade dentro de um aparelho móvel está relacionado com o risco de alguma outra pessoa, ou malware, controlado por algum agente mal intencionado, ter acesso às informações contidas dentro do diretório do aparelho e, portanto, como esses dados são armazenados dentro do sistema de arquivos do aparelho. Essa vulnerabilidade pode ser explorada em situações em que o dono do aparelho é roubado e quem o roubou pode então ter acesso às diversas informações do dono do aparelho simplesmente navegando entre as pastas do seu gerenciador de arquivos. Pode acontecer também em casos onde um malware é inserido dentro do aparelho, através de uma conexão via cabo com um computador que já possui o malware dentro dele, por exemplo. Quando esse malware é executado dentro do aparelho, pode ter acesso à todos os dados contidos nos diretórios, incluindo aplicativos de terceiros que possuem informações importantes, pessoais e muitas vezes sigilosas, permitindo que o malware execute seu propósito, como roubar dados ou modificar um aplicativo real para roubar todo tipo de dados ([OWASP, 2016](#)).

Essas vulnerabilidades podem aparecer quando o time de desenvolvimento do aplicativo acredita que uma pessoa usando seu aplicativo ou um malware não terá acesso ao sistema de arquivos do dispositivo e aos dados sigilosos contidos neles. Isso é, na verdade, um erro grave, pois o sistema de arquivos de um aparelho móvel é bastante simples de acessar e, portanto, a equipe de desenvolvimento deve estar atenta a isso e esperar que tanto pessoas com más intenções quanto malwares podem buscar acessar o sistema de arquivos de um aparelho ([OWASP, 2016](#)).

Algumas formas de armazenamentos que podem ser feitas de forma insegura ou que podem ter vazamento de dados não intencional são ([OWASP, 2016](#)):

- Base de dados criado em SQL;
- Arquivos de *log*;
- Armazenamento de dados XML;
- Armazenamento de dados binários;
- *Cookies*;
- Cartão SD, e
- Armazenamento em *Cloud* sincronizado.

A materialização desse risco gerado por essa vulnerabilidade pode causar diversos impactos técnicos como a perda de dados de um ou muitos usuários. Além disso ou-

tros impactos são a extração de informações secretas do aplicativo através de malwares, aplicativos que foram alterados ou ferramentas forenses [OWASP \(2016\)](#).

Além dos impactos técnicos, ainda podem existir impactos ao negócio da aplicação como roubo de identidade, transgressão da identidade, falsificação, prejuízos à reputação ou violação das políticas externas como a do *Payment Card Industry Data Security Standard* (PIC), que, segundo [Bhutta \(2022\)](#), é uma política que aperfeiçoa a segurança dos dados do titular de um cartão, fornecendo medidas de segurança para todas as entidades envolvidas no processamento de pagamentos.

2.2.3 M3 - Comunicação Insegura

Baseado em [OWASP \(2016\)](#), a vulnerabilidade relacionada à comunicação insegura trata-se do risco de algum dado que está em tráfego de um ponto A para o ponto B ser, de alguma forma, interceptado por um agente C, podendo esse, realizar alguma ação sobre o dado. Ao desenvolver um aplicativo móvel, geralmente existem diversas trocas de dados usando a arquitetura cliente-servidor e, portanto, muitos dados são transmitidos pela rede de internet. Dessa forma, é possível explorar vulnerabilidades para capturar dados secretos no momento em que estão trafegando por ela. Dificilmente os aplicativos são desenvolvidos pensando na proteção dos dados quando trafegados pela rede. Isso faz com que dados como IDs de sessão, por exemplo, sejam facilmente interceptados ([OWASP, 2016](#)).

Essa falha pode gerar diversos problemas, dentre eles, a exposição dos dados de um usuário, podendo o interceptador, ter acesso à sua conta. Isso pode se agravar mais ainda se esses dados forem de um administrador do sistema, pois, dependendo de suas permissões dentro de uma aplicação, pode expô-la por inteiro. Além dos prejuízos técnicos, os negócios podem sofrer vários impactos negativos como fraude, roubo de identidade ou danos à reputação ([OWASP, 2016](#)).

Essa vulnerabilidade cobre todo tipo de risco que pode existir em uma obtenção de informação de um ponto A para o ponto B, podendo essa comunicação ser de celular para celular, do celular para qualquer outro aparelho, comunicação do aplicativo com uma API externa. Essa vulnerabilidade pode se expressar em todo tipo de tecnologia de comunicação entre aparelhos, dentre eles ([OWASP, 2016](#)):

- Bluetooth;
- TCP/PI;
- WiFi, e

- NFC ³.

2.2.4 M4 - Autenticação Insegura

Na visão de OWASP (2016), a vulnerabilidade de autenticação insegura trata de fragilidades no processo de autenticação do aplicativo, que geralmente são explorados por meio de ataques automatizados. Quando o atacante percebe que a autenticação do sistema é frágil, ele pode facilmente falsificar uma autenticação ou desconsiderá-la, fazendo o envio direto de solicitações ao servidor do aplicativo sem que haja a necessidade de interagir com ele. Esse processo geralmente é feito através de um malware móvel ou botnets⁴ do invasor.

Uma autenticação mal desenvolvida ou inexistente dá a possibilidade do atacante executar uma funcionalidade de forma anônima dentro do aplicativo ou no servidor backend consumido pelo aplicativo. Por se tratar de uma aplicação móvel, frequentemente prevalece tipos de entradas mais fracas como PINs de 4 ou 6 dígitos. Além disso, autenticação em aplicações móveis pode se diferir bastante em relação à autenticações web, visto que em aplicações móveis não se espera que o usuário esteja o tempo inteiro conectado à internet. Por outro lado, as conexões web são mais confiáveis que as conexões de internet móvel. Por isso, é preciso definir requisitos de tempo de atividade para aplicativos móveis que demandam autenticação *offline* (OWASP, 2016).

Essa vulnerabilidade gera impactos no sentido de que a aplicação não consegue identificar o usuário que executa alguma tarefa. Dessa forma, o sistema não poderá registrar a atividade de um usuário e, portanto, não conseguirá investigar quem executou alguma ação, caso precise, pois não existe um usuário vinculado. Isso colabora com falta de competência da aplicação de detectar a origem de um ataque, ou como evitar novos ataques, por exemplo. Além disso, falhas na autenticação geram problemas de autorização, pois se existe alguém usando a aplicação sem uma sessão, a aplicação, por conseguinte, não terá conhecimento da identidade do usuário e suas permissões. Se um invasor conseguir ter acesso anonimamente à funcionalidades sigilosas, ele perceberá que o aplicativo não está verificando as permissões que está solicitando uma ação. Somando-se a isso, a autenticação fraca pode gerar problemas ao negócio como, por exemplo, acesso não autorizado aos dados, furto de informações e danos à reputação (OWASP, 2016).

³ A Comunicação por Campo de Proximidade (Near Field Communication - NFC) é uma forma de transmissão sem fio de curto alcance, utilizando padrões previamente estabelecidos, já empregados em tecnologias como Identificação por Radiofrequência e cartões inteligentes. Essa tecnologia possibilita a troca de informações entre dispositivos sem a necessidade de contato físico direto (ROCHA, 2022).

⁴ Uma botnet é uma rede de computadores comprometidos, coordenados por uma única fonte, geralmente remota. Essa rede é capaz de se auto-propagar e infectar máquinas vulneráveis, mantendo comunicação com um controlador para receber comandos. Sua flexibilidade e capacidade de recrutamento de novos sistemas a tornam uma ameaça potente, podendo comprometer diversas topologias de rede. (OGU et al., 2019).

Um aplicativo pode estar exposto a essa vulnerabilidade por diversos motivos como (OWASP, 2016):

- Se um usuário conseguir fazer requisições ao *back-end* sem que haja a necessidade de envio de *token*;
- Se o aplicativo guardar localmente quais quer senhas que seja, e
- Se usar abordagens de senhas fracas para simplificar a digitação de uma senha.

2.2.5 M5 - Criptografia Insuficiente

Sob o ponto de vista de OWASP (2016), Criptografia Insuficiente trata-se de uma vulnerabilidade que aparece na aplicação quando a equipe de desenvolvimento não implementa a criptografia de dados confidenciais, utilizando os métodos de criptografia confiáveis e amplamente aceitos pela comunidade. Em termos quantitativos, uma criptografia suficientemente forte será considerada uma criptografia que resista ao teste de descriptografia por pelo menos dez anos. Criptografias inferiores a isso serão consideradas criptografias insuficientes. Ataques à dados criptografados podem acontecer em chamadas de API, onde dados criptografados são passados no momento da interação entre a API e o dispositivo móvel. Essa vulnerabilidade é explorada, caso o atacante consiga descriptografar o dado criptografado e obter a informação confidencial original (OWASP, 2016).

Essa vulnerabilidade possui impactos técnicos, de modo que, dados não autorizados de informações sigilosas poderão ser recuperados. Isso impacta o negócio da aplicação em muitos aspectos como violação de privacidade, roubo de informações, roubo de código, roubo de propriedade intelectual, danos à reputação (OWASP, 2016).

Existem dois motivos fundamentais para que seja possível quebrar a criptografia de um dispositivo móvel. O primeiro é que o aplicativo móvel pode utilizar um procedimento por traz da criptografia e descriptografia que é fundamentalmente falho e pode ser explorado pelo atacante para descriptografar a informação. O segundo motivo é que pode ser usado um algoritmo de criptografia que seja fraco por natureza e que pode ser diretamente descriptografado pelo invasor (OWASP, 2016).

A maioria dos aplicativos não se preocupa em como o usuário utiliza e armazena suas chaves de criptografia. Geralmente, utilizam algoritmos de criptografia de forma adequada, mas possuem sua própria maneira de utilizá-lo. Alguns exemplos são (OWASP, 2016):

- Inserir as chaves de criptografia e descriptografia no mesmo lugar onde o conteúdo está criptografado;
- Disponibilizar as chaves de alguma forma que facilite para o invasor encontrar, e

- Inclusão da chave dentro do binário do aplicativo móvel.

Além disso, existem aplicações que utilizam algoritmos de criptografia personalizadas com seus próprios protocolos de criptografia. Em vez de utilizar formas de criptografias que modernas, populares e aceitas por toda a comunidade, escolhem por criar suas próprias formas de criptografar seus dados, aumentando os riscos de atacantes obterem os dados da aplicação (OWASP, 2016).

Muitos algoritmos de criptografia estão obsoletos e não devem serem utilizados por mostrarem-se fracos ou insuficientes para os requisitos de segurança modernos. Alguns deles são (OWASP, 2016):

- RC2;
- MD4;
- MD5;
- SHA1.

2.2.6 M6 - Autorização Insegura

Em consonância com OWASP (2016), a vulnerabilidade de autorização insegura se concretiza quando o invasor percebe que o processo de autorização é falho, possuindo vulnerabilidades em sua lógica. O atacante passa pelo processo de autorização como usuário autêntico do sistema. Assim que consegue passar pela autenticação, o invasor força o acesso a páginas do sistema para acessar as funcionalidades administrativas do sistema.

Para verificar a existência dessa vulnerabilidade, o atacante pode executar ataques binários contra o aplicativo em questão e tentar efetuar tarefas que são destinadas a usuários de privilégios superiores, no momento em que o aplicativo estiver em modo *offline*. Além disso, podem testar essa vulnerabilidade no servidor desse aplicativo móvel, fazendo requisições HTTP, enviando um *token* de autorização que não condiz com as permissões necessárias para se ter uma resposta delas (OWASP, 2016).

Esse tipo de vulnerabilidade possui os mesmo tipos de impacto técnico que a vulnerabilidade de autenticação insegura. Os impactos técnicos dependem da natureza das funcionalidades privilegiadas da aplicação. Ela pode, por exemplo, permitir que usuários não autorizados tenham acesso a informações confidenciais, ou permitir que os mesmos destruam todo o sistema, modificando, roubando ou apagando informações necessárias para que a aplicação permaneça utilizável. Além disso, caso um usuário tenha acesso a privilégios que não foram dados a ele, pode acarretar muitos problemas à empresa como fraude, roubo de informações ou danos à reputação (OWASP, 2016).

As vulnerabilidade de autenticação e autorização, apesar de possuírem suas diferenças, estão conectadas. Não é possível que uma aplicação verifique as permissões de um usuário para executar determinada tarefa, sem que antes saiba quem esse usuário é e que tipo de usuário é dentro do contexto da aplicação. Se a aplicação não consegue fazer essas verificações, por consequência, não consegue também verificar seus privilégios antes de executar cada tarefa.

Existem algumas implementações dentro do código do sistema que podem facilitar a invasão da aplicação, explorando a vulnerabilidade de autenticação insegura (OWASP, 2016):

- Rotas do servidor escondidas: isso acontece quando os desenvolvedores assumem que determinada requisição não será exibida para qualquer usuário, exceto para aqueles que possuem permissão para acessá-la, dessa forma, não são verificados os privilégios dos usuários nessas rotas, e
- Envio de funções ou privilégios: isso acontece quando informações de privilégios, funções, ou informações do papel desse usuário são enviadas em requisições para o *back-end*. Dessa maneira, a aplicação móvel possuirá vulnerabilidades de autorização, visto que essas informações estão sendo verificadas a partir de informações enviadas pelo aplicativo móvel, em vez de pelo *token* enviado junto na requisição, podendo essa informação ser enviada intencionalmente de modo a burlar o sistema de autenticação.

2.2.7 M7 - Qualidade Pobre de Código

Em conformidade a OWASP (2016), qualidade pobre de código é uma vulnerabilidade que geralmente se expressa quando as entradas de usuários do aplicativo não são tratadas de forma correta, podendo acarretar em estouros de *buffer* ou alteração do uso da memória, por exemplo. Apesar de não se tratar de um problema direto de segurança, pode levar à exploração de outras vulnerabilidades, como à ataques de *Jailbreak*⁵. Essa vulnerabilidade é explorada quando um invasor cuidadosamente fornece entradas que possibilitem a exploração de problemas dentro do aplicativo, como vazamento de memória ou estouro de *buffer*.

Essa vulnerabilidade geralmente ocorre em sistemas aonde existem problemas de qualidade de código, resultantes de más práticas de desenvolvimento. Geralmente, os invasores identificam se a aplicação possui problemas na qualidade de código através de

⁵ Jailbreak consiste na modificação do software do telefone para permitir que os usuários ultrapassem as restrições estabelecidas pela Apple para o uso do dispositivo. (MAGAUDDA, 2010)

ferramentas que realizam análises estáticas ou *fuzzing*⁶. Essas ferramentas podem identificar vazamentos de memória ou estouros de *buffer* entre outros problemas que podem ser explorados pelos invasores (OWASP, 2016).

Geralmente, os impactos técnicos dessa vulnerabilidade em dispositivos móveis não são sérios, mas caso haja estouro de memória dentro do aplicativo, pode dar vazão a outras vulnerabilidades e deve ser corrigido o quanto antes. Os impactos comerciais dependem da natureza da exploração. Problemas que envolvem qualidade pobre de código que resultem na execução de código de forma remota podem gerar problemas como roubo de informações, danos à reputação e roubo de propriedade intelectual (OWASP, 2016).

2.2.8 M8 - Adulteração de Código

Com base em (OWASP, 2016), a adulteração de código é uma vulnerabilidade que comumente é explorada em aplicativos hospedados em lojas de aplicativos de terceiros. O invasor modifica o código de algum aplicativo existente e disponibiliza em lojas de terceiros, em que usuários podem baixar e assim o invasor pode atuar para cumprir seus objetivos. Outra forma de induzir o usuário a instalar seu aplicativo modificado é por meio de ataques de *phishing*⁷. Um atacante geralmente utiliza algumas formas de explorar essa vulnerabilidade: a alteração direta do binário principal do pacote do aplicativo, modificação dos recursos e funcionalidades dentro do aplicativo e redirecionamento ou troca da API utilizada no sistema para barrar a comunicação e executar códigos maliciosos.

Ao fazer o *download* do aplicativo, suas informações como código e recursos de dados passam a habitar dentro do dispositivo móvel. Dessa forma, o invasor pode modificar diretamente o código do aplicativo, alterar ou substituir as APIs existentes no sistema ou modificar os recursos do aplicativo. Com isso, o invasor pode corromper o uso original pretendido da aplicação para ganhos pessoais (OWASP, 2016).

Os impactos técnicos que essa vulnerabilidade pode trazer incluem a criação de novas funcionalidades não autorizadas no aplicativo móvel, roubo de identidade e fraude. Além disso, os impactos nos negócios podem incluir a perda de receita da empresa por conta da pirataria sobre seu produto e prejuízos à reputação (OWASP, 2016).

Toda aplicação móvel está sujeita à deturpação de seu código, pois essas modifica-

⁶ Fuzzing é um método automatizado de teste de software, que fornece dados aleatórios como entrada a um sistema de software na esperança de expor uma vulnerabilidade. Para ser eficaz, a entrada fuzzed deve ser comum o suficiente para passar por verificações de consistência elementares. Por outro lado, a entrada fuzzed deve ser incomum o suficiente para desencadear comportamentos excepcionais, como uma falha no interpretador. (HOLLER; HERZIG; ZELLER, 2012).

⁷ Ataques de phishing são ataques fraudulentos com o objetivo de roubar informações sensíveis, incluindo credenciais de login e números de cartão de crédito das vítimas. Tipos comuns de ataques de phishing utilizam e-mails, sites ou dispositivos móveis para enganar as vítimas. Os atacantes se disfarçam como partes confiáveis e enviam e-mails falsos, mostram sites falsos ou enviam SMS ou mensagens instantâneas falsas. (RASHID et al., 2021)

ções são realizadas em um ambiente em que a organização que é dona do aplicativo não possui controle (OWASP, 2016).

2.2.9 M9 - Engenharia Reversa

Consoante a OWASP (2016), engenharia reversa é uma vulnerabilidade que ocorre quando um invasor faz o download de algum aplicativo e consegue analisar o código em seu próprio dispositivo com o auxílio de ferramentas como as de inspeção binária. Geralmente, um invasor realiza uma análise do binário principal da aplicação com o propósito de determinar sua tabela de *strings* original, código fonte, bibliotecas utilizadas, algoritmos e recursos existentes no aplicativo. De modo geral, todo código de uma aplicação móvel está sujeito à engenharia reversa. Aplicativos que possuem códigos escritos em linguagens ou *frameworks* como Java, .NET, Objective C e Swift, que permitem introspecção dinâmica em tempo de execução, são mais suscetíveis, e estão particularmente em risco de engenharia reversa.

Os impactos técnicos envolvidos nessa vulnerabilidade incluem expor informações sobre servidores *back-end*, expor constantes criptográficas e cifras, roubo de propriedade intelectual, execução de ataques contra sistemas *back-end*, ou adquirir o conhecimento necessário para aplicar modificações de código posteriormente. Ademais, essa vulnerabilidade pode gerar diversos impactos nos negócios como roubo de propriedade intelectual, danos à reputação, roubo de identidade ou o comprometimento do sistema *back-end* (OWASP, 2016).

Hoje em dia, a maioria das linguagens de programação usadas para desenvolver um aplicativo móvel possui bastante metadados que ajudam o programador na depuração do aplicativo. Esse mesmo artifício também auxilia o invasor a entender como o aplicativo funciona. Um aplicativo móvel pode ser visto como um aplicativo vulnerável à engenharia reversa se um invasor for capaz de realizar alguma das seguintes tarefas (OWASP, 2016):

- Compreender de forma clara o conteúdo existente na tabela de *strings* de um binário.
- Realizar análises multifuncionais com precisão, e
- Conseguir recriar o código-fonte de forma razoavelmente precisa a partir do binário.

2.2.10 M10 - Funcionalidade Estranha

De acordo com OWASP (2016), funcionalidade estranha trata-se de uma vulnerabilidade que é causada pelo descuido dos programadores no desenvolvimento do aplicativo ao deixar passar na compilação para ambiente de produção resquícios de desenvolvimento como arquivos de *logs*, arquivos de configuração ou o próprio binário da aplicação. O invasor geralmente busca compreender funcionalidades estranhas em um aplicativo no intuito

de descobrir funcionalidades escondidas nos sistemas de *back-end*. Para isso, o invasor faz o *download* e o analisa em seu próprio ambiente local para descobrir possibilidades escondidas ou códigos de teste que foram esquecidos pelos desenvolvedores no momento do empacotamento do aplicativo móvel.

Existem grande chances de que um aplicativo móvel possua funcionalidades estranhas que apareçam para o usuário através da interface do aplicativo. Grande parte delas não dará ao invasor uma visão adicional sobre os recursos do *back-end*. Porém, algumas delas podem ser úteis para o atacante. Funcionalidades que revelam informações relacionadas a ambientes de teste ou *endpoints* da API utilizadas como recursos administrativos, por exemplo, não devem ser adicionadas em compilação de produção (OWASP, 2016).

Alguns dos impactos técnicos existentes relacionados a essa vulnerabilidade são a exposição do funcionamento do sistema *back-end* e a execução de ações de alto privilégio não autorizadas. Ademais, alguns dos impactos no negócios da aplicação são o acesso não autorizado de funcionalidades sigilosas, dano à reputação da empresa e roubo de propriedade intelectual (OWASP, 2016).

Desenvolvedores de aplicativos móveis constantemente adicionam funcionalidades *backdoor* escondidas ou outros controles internos de segurança de desenvolvimento que não devem ser compilados juntamente para produção. Um exemplo disso é quando um desenvolvedor, de forma descuidada, inclui no meio do código do aplicativo alguma senha como comentário. Outro exemplo seria a desativação de autenticação de dois fatores durante o teste (OWASP, 2016).

2.3 Práticas de Segurança no Desenvolvimento de Aplicativos Móveis

A organização OWASP, em seu documento “*Mobile Application Security Verification Standard*” (MASVS), define alguns níveis de verificação de requisitos de segurança que devem ser implementados dependendo da natureza da aplicação, objetivos e nível de segurança que se deseja ter essa aplicação. Conforme OWASP et al. (2022) o MASVS propõe dois níveis de requisitos de segurança, além de um grupo de requisitos que torne a aplicação resiliente à engenharia reversa do código, fazendo com que o invasor tenha dificuldade para analisar o código e a arquitetura do aplicativo. Elas são definidos da seguinte forma:

- MASVS-L1: Conjunto de requisitos básicos que é indicado para todo tipo de aplicação móvel para ser considerada minimamente segura. Caso esse conjunto de requisitos seja cumprido, resultará em uma aplicação móvel segura que aplica as melhores

práticas de segurança indicadas, além de ser resistente contra as vulnerabilidades mais comuns.

- **MASVS-L2:** Conjunto de requisitos que abrange controles de segurança adicionais que podem ser usados em aplicações onde existe a manipulação de dados altamente sigilosos. A aplicação desse conjunto de requisitos resulta em uma aplicação mais preparada para resistir a ataques mais complexos.
- **MASVS-R:** Conjunto de requisitos que adiciona controle de segurança adicional, caso um dos objetivos do projeto seja prevenir ameaças do lado do usuário. A aplicação desses requisitos ajuda a atenuar ameaças no caso de o usuário final ter intenções maliciosas no uso do aplicativo, ou quando o sistema operacional móvel esteja danificado.

Além disso, a [OWASP et al. \(2022\)](#) definem oito grupos de requisitos que orientam os desenvolvedores na implementação de um aplicativo móvel seguro. Em cada um dos requisitos, é possível visualizar os níveis de verificação de segurança em que estão inclusos. Segue a lista de conjuntos de requisitos definidos pelo MASVS:

- **V1: Requisitos de Arquitetura, Design e Modelagem de Ameaças:** lista os requisitos necessários para garantir que segurança tenha sido abordada no momento do planejamento da arquitetura e do *design* do aplicativo ([OWASP et al., 2022](#)).
- **V2: Requisitos de Armazenamento de Dados e Privacidade:** lista de requisitos necessários para garantir que dados sensíveis não sejam expostos de forma não intencional pela aplicação. Podem ser considerados dados sensíveis, credenciais do usuário ou quaisquer outros dados considerados sensíveis em contextos particulares, como números de cartão de crédito, dados de conta bancária, informações de saúde, informações de contrato, entre outros ([OWASP et al., 2022](#)).
- **V3: Requisitos de Criptografia:** lista de requisitos necessários para que dados armazenados ou enviados pela aplicação sejam devidamente protegidos, utilizando a criptografia como forma de proteger o conteúdo desses dados. Estes requisitos de criptografias buscam garantir que o aplicativo verificado utilize a criptografia de acordo com as melhores práticas do setor, incluindo o uso de bibliotecas criptográficas comprovadas, escolha e configuração adequada das primitivas criptográficas, e o uso de um gerador de números aleatórios adequado sempre que a aleatoriedade for necessária ([OWASP et al., 2022](#)).
- **V4: Requisitos de Autenticação e Gerenciamento de Sessão:** lista de requisitos que ditam como as contas e sessões dos usuários devem ser gerenciadas, afim

de garantir uma autenticação segura na aplicação, além de garantir a verificação adequada de autorização de cada usuário.(OWASP et al., 2022).

- **V5: Requisitos de Comunicação de Rede:** lista de requisitos necessários para garantir a confidencialidade e a integridade das informações trocadas entre o aplicativo móvel e os terminais de serviço remoto (OWASP et al., 2022).
- **V6: Requisitos de Interação com a Plataforma:** lista de requisitos necessários para garantir que o aplicativo utilize APIs da plataforma de forma segura. Além disso, os requisitos cobrem a comunicação entre aplicativos (IPC) (OWASP et al., 2022).
- **V7: Requisitos de Qualidade de Código e Configuração do Compilador:** lista de requisitos necessários para garantir que práticas básicas de codificação segura são seguidas no desenvolvimento do aplicativo móvel, e que as funcionalidades de segurança do compilador estão habilitadas (OWASP et al., 2022).
- **V8: Requisitos de Resiliência:** lista de requisitos que cobrem as medidas de segurança recomendadas para aplicativos móveis que processam, ou dão acesso para, dados ou funcionalidades sensíveis. A falta de qualquer um desses controles não causa uma vulnerabilidade. Pelo contrário, seu objetivo é aumentar a resiliência do aplicativo móvel contra engenharia reversa e ataques específicos do lado do cliente (OWASP et al., 2022).

A lista com todos os requisitos e o detalhamento sobre cada um deles podem ser encontrada no [Apêndice A](#).

3 Metodologia

3.1 Considerações Iniciais

Neste capítulo, é retomado e descrito em maior detalhe o plano metodológico apresentado anteriormente de forma breve no [Capítulo 1](#). Esta pesquisa é do tipo Descritiva, com a aplicação da Técnica Estudo de Caso. Nas próximas seções, são apresentados o detalhamento da proposta do trabalho em questão, evidenciando as atividades realizadas para cumprir com os objetivos deste trabalho.

3.2 Planejamento de Pesquisa

Nesta etapa, foram definidos o tema de pesquisa, a questão de pesquisa, e o objetivo a ser atingido. Esta etapa é parcialmente contemplada no [Capítulo 1](#). Além disso, a classificação metodológica e o plano Plano de Pesquisa, componentes desta etapa, foram brevemente explanados no [Capítulo 1](#) e serão detalhadas a seguir.

A metodologia deste trabalho foi definida e classificada quanto à natureza, à abordagem e aos procedimentos técnicos. A natureza pode ser identificada como aplicada, por ter como objetivo produzir conhecimento para aplicações práticas a soluções de problemas específicos. A abordagem caracteriza-se por uma união de métodos qualitativos e quantitativos. O uso de medições, característica de uma abordagem quantitativa e a análise subjetiva de resultados, característica de uma abordagem qualitativa ([MORESI, 2003](#)), estão presentes nesta pesquisa. Para realizar este trabalho, os seguintes procedimentos técnicos foram adotados:

- **Pesquisa Bibliográfica:** Trata-se de um estudo estruturado realizado com base em material publicado principalmente em livros, revistas e jornais ([MORESI, 2003](#)). Foi realizado através de consultas em bases bibliográficas sobre os temas principais referentes ao trabalho como segurança de Software, principais vulnerabilidades e práticas de segurança de software com o foco no desenvolvimento de aplicativos móveis.
- **Pesquisa Documental:** Trata-se de estudos realizados em documentos provenientes de órgãos públicos e privados de qualquer tipo ([MORESI, 2003](#)). Foi realizado através de documentos majoritariamente produzidos pela organização OWASP, com o objetivo de conhecer mais a respeito das principais vulnerabilidades encontradas em um dispositivo móvel, além de buscar compreender quais práticas de desen-

volvimento são adotadas para prevenir, detectar ou mitigar essas vulnerabilidades, e

- **Estudo de Caso:** Método utilizado com o objetivo de investigar de forma detalhada um fenômeno, de forma a analisar o contexto e os processos envolvidos no fenômeno em estudo (MORESI, 2003). Foi escolhido para o trabalho, pois foi desenvolvido um aplicativo móvel com o objetivo de analisar aspectos inerentes ao problema em questão.

3.3 Plano Metodológico

Como descrito por Brereton et al. (2008), as tarefas envolvidas no desenvolvimento de um estudo de caso consistem em quatro principais etapas: Planejamento; Coleta dos Dados; Análise dos Dados; e Relatórios, que compreendem o plano metodológico escolhido neste trabalho:

- Planejamento: nesta etapa, foram estabelecidos a questão de pesquisa, os objetivos geral e específicos, a classificação e o plano da metodologia de pesquisa.
- Coleta dos Dados: nessa etapa, são designados e utilizados os procedimentos de revisão bibliográfica e documental, além da técnica estudo de caso.
- Análise dos Dados: nessa etapa, os dados coletados e resultados alcançados são analisados quantitativamente e qualitativamente.
- Relatórios: e por fim, nessa etapa, o relatório estabelece a escrita deste documento de Trabalho de Conclusão de Curso.

Foi escolhido para este trabalho o protocolo definido por Brereton et al. (2008), com o objetivo de orientar o pesquisador em questão no planejamento das onze etapas que o compõem o Estudo de Caso, apresentadas na Tabela 1.

3.4 Coleta de Dados

Nesta fase, para a coleta de dados, foram aplicadas as técnicas Pesquisa Bibliográfica e Pesquisa Documental, além das etapas do Protocolo da Técnica de Estudo de Caso, com a descrição do objeto de estudo.

Para a realização do Estudo de Caso, foi necessário selecionar um subconjunto dos requisitos do nível L1 conforme descritos no documento Mobile Application Security and Privacy Standard (MASVS) (OWASP et al., 2022). Os requisitos selecionados, foram escolhidos com o objetivo de direcionar o foco do Estudo de Caso. A seleção baseou-se

Tabela 1 – Etapas do Protocolo de Estudo de Caso (BRERETON et al., 2008)

| Nº | Etapas | Descrição |
|----|--|--|
| 1 | <i>Background</i> | Procurar por pesquisas anteriores sobre o tema; definir principal questão de pesquisa abordada neste estudo; identificar qualquer outras questões adicionais a serem abordadas; |
| 2 | <i>Design</i> | Identificar se serão usados caso único ou casos múltiplos; descrever o objeto de estudo; identificar quaisquer proposições ou subquestões derivadas de cada questão de pesquisa; |
| 3 | Seleção de caso | Definir os critérios para seleção de casos; |
| 4 | Procedimentos e papéis do estudo de caso | Definir regras para conduzir dos procedimentos de campo; definir os papéis de cada membro do time de pesquisa; |
| 5 | Coleta dos dados | Identificar os dados que serão coletados; Definir um plano de coleta de dados; Definir como os dados serão armazenados; |
| 6 | Análise | identificar os critérios para interpretar os resultados do estudo de caso; Identificar quais dados respondem determinadas questões de pesquisa e como os dados serão combinados para responder a questão da pesquisa; Considerar a gama de possíveis resultados; |
| 7 | Validade do plano | Verificar o planejamento em relação aos aspectos gerais, a validade do constructo, a validade interna e a validade externa; |
| 8 | Limitações do estudo | Especificar as questões de validade residual incluindo potenciais conflitos de interesse; |
| 9 | Relatórios | Identificar o público-alvo e o relacionamento com outros estudos; descrever os resultados; |
| 10 | Cronograma | Dar uma estimativa sobre a duração das quatro etapas principais: Planejamento, Coleta de dados, Análise dos dados e Geração de Relatórios; |
| 11 | Apêndices | Atualizar durante a condução do estudo, observando quaisquer divergências das etapas acima. |

em critérios específicos, visando tanto a relevância para a implementação de uma robusta camada de autenticação, quanto à consideração da possível facilidade de implementação.

A decisão de priorizar requisitos vinculados à autenticação justifica-se pela natureza do aplicativo, onde seu objetivo central é proporcionar um processo seguro e eficiente de autenticação para os usuários. Dessa forma, os requisitos foram selecionados para alinhar-se de maneira mais precisa às necessidades específicas desta aplicação.

A inclusão do critério da possível facilidade de implementação reflete a consideração sobre os desafios potenciais associados à execução desses requisitos no contexto do

desenvolvimento do aplicativo. Embora a certeza sobre a simplicidade da implementação não seja absoluta, a inclusão desse critério permite uma abordagem mais realista para otimizar o desenvolvimento e conclusão do trabalho dentro do prazo estipulado.

Assim, a delimitação do escopo do Estudo de Caso para esses requisitos específicos proporciona uma base sólida para a avaliação da segurança da autenticação no contexto de uma aplicação móvel, contribuindo para a obtenção de resultados significativos e relevantes no âmbito deste trabalho. Os requisitos de segurança aplicados na fase do Estudo de Caso encontram-se descritos em detalhes no [Capítulo 4](#).

Para avaliar cada requisito, foram utilizados mecanismos de medição a fim de quantificar os esforços necessários para sua implementação. As equações que regem a mensuração dos esforços envolvidos no desenvolvimento do aplicativo e na aplicação das práticas de codificação segura estão detalhadamente definidas nas equações [Equação 3.1](#), [Equação 3.2](#) e [Equação 3.3](#).

$$ED = Xh; \quad (3.1)$$

$$ES = Xh; \quad (3.2)$$

$$ET = ED + ES \quad (3.3)$$

onde:

Xt = Quantidade de tempo medido em horas;

ED = Esforço no desenvolvimento do aplicativo;

ES = Esforço na aplicação dos requisitos de segurança;

ET = Esforço total no desenvolvimento do aplicativo;

3.5 Análise dos Resultados

Nesta etapa, após a execução da técnica de Estudo de Caso, os dados e resultados obtidos foram apresentados analisados.

A partir dos resultados, foi realizada uma análise quantitativa para compreender a dificuldade enfrentada para aplicar o requisito. Essa análise foi baseada no tempo dispendido na implementação do referido requisito.

Além disso, foi realizada uma análise qualitativa para compreender a dificuldade percebida pelo desenvolvedor júnior na implementação do requisito. Essa análise foi re-

alizada, baseada na impressão de desenvolver a respeito das dificuldades enfrentadas no momento da implementação do requisito.

Adicionalmente, foram realizadas comparações entre o tempo de implementação entre os requisitos, o tempo de implementação total dos requisitos de segurança e das funcionalidades e apresentado uma correção entre as principais vulnerabilidades e os requisitos de segurança.

A descrição da Análise de Dados é apresentada no [Capítulo 5](#)

3.6 Atividades Realizadas

3.6.1 Pesquisa Bibliográfica

A pesquisa bibliográfica foi conduzida mediante consulta às principais bases científicas, concentrando-se especificamente na temática de Segurança de Software e segurança no desenvolvimento de aplicativos móveis.

Nos capítulos de introdução ([Capítulo 1](#)) e referencial teórico ([Capítulo 2](#)), discutiu-se a necessidade da existência de aplicativos móveis seguros. Além disso, enfatizou-se a importância de abordar o desenvolvimento de aplicações com a consideração primordial da capacidade do aplicativo em resistir a ataques de segurança. Este enfoque é sustentado por práticas de segurança eficazes, as quais foram exploradas na identificação e estabelecimento de requisitos de segurança. Estes requisitos têm como objetivo assegurar a resiliência do aplicativo contra potenciais ameaças e vulnerabilidades.

3.6.2 Pesquisa Documental

Como parte da revisão documental deste trabalho, foram conduzidos estudos sobre as principais vulnerabilidades identificadas em aplicativos móveis, juntamente com as práticas de segurança destinadas a mitigar cada uma dessas fragilidades. Todas essas informações foram predominantemente obtidas nos documentos disponibilizados pela organização OWASP, sendo detalhadas nos capítulos de referencial teórico ([Capítulo 2](#)) e na lista de requisitos de segurança para aplicativos móveis ([Apêndice A](#)).

3.6.3 Etapas do Protocolo de Estudo de Caso

Com base no protocolo proposto por [Brereton et al. \(2008\)](#), utilizado como guia para a execução do estudo de caso neste trabalho, apresenta-se a seguir uma descrição detalhada de cada etapa do processo.

1. **Background:** A revisão bibliográfica e a definição do tema, questão de pesquisa

principal, os objetivos. Contendo os [Capítulo 1](#) - Introdução e [Capítulo 2](#) - Referencial Teórico;

2. **Design:** Este é um estudo de caso único, cuja descrição do objeto de estudo e questões foram apresentados no [Capítulo 1](#) - Introdução;
3. **Seleção de caso:** A seleção do caso, assim como os detalhes do desenvolvimento do aplicativo são apresentados no [Capítulo 1](#) - Introdução. Foi desenvolvido um aplicativo com apenas a funcionalidade de autenticação, com o objetivo de aplicar os requisitos definidos para o escopo deste trabalho. A lista de requisitos definidos estão detalhados no capítulo de Estudo de Caso ([Capítulo 4](#));
4. **Procedimentos e papéis do estudo de caso:** O autor pesquisador deste trabalho é o desenvolvedor desse aplicativo - objeto de estudo de caso.

Para o processo de desenvolvimento, foi adotada alguns componentes das metodologia de desenvolvimento ágil de forma adaptada para o contexto, visto que apenas o pesquisador foi também o desenvolvedor desse aplicativo. No processo de desenvolvimento, foram realizadas quatro macro atividades:

- Realização da elicitación dos requisitos, com o objetivo de ter uma melhor compreensão da aplicação a ser desenvolvida, guiado pelos requisitos de segurança previamente selecionados;
 - Planejamento da arquitetura da aplicação, com enfoque em estruturas que conferem prioridade à segurança de software, e
 - Desenvolvimento do aplicativo, tendo o cuidado de fazer uso das práticas de codificação segura em todo o processo de desenvolvimento.
5. **Coleta de Dados:** Para coletar dados a respeito de esforço e nível de dificuldade, foi realizado a medição do tempo de desenvolvimento tanto da aplicação quanto da aplicação dos requisitos de segurança de software para aplicativos móveis, fazendo distinção entre elas. Para medir os conhecimentos necessários para aplicar essas práticas, foram analisados os fundamentos inclusos em cada prática.
 6. **Análise:** Após a coleta dos dados, foi analisado o esforço a aplicação de cada requisito, o esforço total de aplicação dos requisitos de segurança e o total de esforço no desenvolvimento da funcionalidade de autenticação, fazendo uma comparação entre eles. Os detalhes da análise dos dados coletados estão descritos no capítulo de análise ([Capítulo 5](#));
 7. **Validade do Plano:** O propósito é discernir o âmbito no qual as descobertas do estudo podem ser estendidas ou generalizadas;

-
8. **Limitações do Estudo:** O âmbito desta pesquisa encontra-se restrito a um único processo de um projeto, limitando, assim, a generalização dos resultados obtidos;
 9. **Relatório:** Esta monografia de TCC representa o relatório abrangente da pesquisa conduzida.
 10. **Cronograma:** O desenvolvimento do aplicativo teve seu início em Abril de 2023 e foi finalizado ao final de Setembro de 2023 totalizando cinco meses de desenvolvimento.
 11. **Apêndices:** O documentos produzido pelo autor deste trabalho constam com apêndices.
 - [Apêndice A](#) - Lista de Requisitos de Segurança para Aplicativos Móveis.

4 Estudo de Caso

4.1 Considerações Iniciais

Nesse capítulo é detalhado o contexto do estudo de caso, com a descrição da estratégia de desenvolvimento, tecnologias utilizadas, escopo do aplicativo móvel desenvolvido, e descrição do desenvolvedor.

4.2 Escopo do Projeto

O escopo do projeto foi definido orientado pela aplicação dos requisitos de segurança, de nível L1, que contemplam toda a parte de autenticação de uma aplicação móvel, visto que as funcionalidades de autenticação de uma aplicação são parte essencial em uma aplicação segura. Além desses requisitos, o escopo desse projeto se propõe a aplicar alguns requisitos de segurança que se correlacionam com a funcionalidade de autenticação.

Dessa forma, o desenvolvimento dessa aplicação se propôs a implementar de forma completa, a funcionalidade de autenticação em uma aplicação móvel, aplicando todos os requisitos necessários para garantir uma autenticação segura.

Os repositórios relacionados à aplicação desenvolvida para este trabalho estão disponíveis nos seguintes endereços:

- **Repositório da Aplicação Móvel:** <https://github.com/TCC-Estevao/app>;
- **Repositório do Back-end:** <https://github.com/TCC-Estevao/api>.

4.3 Requisitos de Segurança de Software

A OWASP et al. (2022), em seu documento *Mobile Application Security Verification Standard - MASVS*, define diversos requisitos de segurança que devem ser aplicados em um desenvolvimento de aplicativo móvel, dependendo da natureza da aplicação e do nível de segurança que se deseja obter. Os requisitos são classificados de duas maneiras: segundo o nível de segurança e segundo a categoria que o requisito está inserido dentro do desenvolvimento de software.

A classificação dos requisitos segundo o nível de segurança, ocorre em 3 níveis:

- **MASVS-L1:** Possui requisitos básicos de segurança recomendados para todo e qualquer tipo de aplicativos para dispositivos móveis,

- **MASVS-L2:** É recomendado para uso em aplicativos que lidam com informações de extrema confidencialidade, e
- **MASVS-R:** Inclui medidas de segurança suplementares que podem ser implementadas caso um dos objetivos do projeto seja a mitigação de ameaças provenientes do usuário. Esse conjunto de requisitos tem por objetivo garantir a resiliência a engenharia reversa.

De acordo com o MASVS, atender aos requisitos do MASVS-L1 resulta em uma aplicação segura que adere às melhores práticas de segurança recomendadas, não sendo suscetível às vulnerabilidades mais comuns. O MASVS-L2 acrescenta controles adicionais de defesa em profundidade, conferindo à aplicação resistência contra ataques mais sofisticados, contanto que os controles de segurança do sistema operacional móvel permaneçam intactos e que o usuário não seja considerado um potencial agressor. A conformidade com todos ou parte dos requisitos de proteção de software no MASVS-R auxilia na mitigação de ameaças específicas no lado do cliente quando o usuário final age de maneira maliciosa ou o sistema operacional móvel está comprometido.

Na [Figura 1](#), são representados os níveis de segurança e em que contexto cada um se aplica.



Figura 1 – Níveis de requisito de segurança. Fonte: Mobile Application Security Verification Standard - MASVS.

Segundo MASVS (OWASP et al., 2022), os requisitos são classificados e agrupados em 8 tipos de categorias. Segue abaixo a numeração, identificação e o detalhamento de cada uma das categorias:

- **V1 - ARCH:** Requisitos relativos à arquitetura e design do aplicativo. A lista dos requisitos dessa categoria se encontra na [Tabela 2](#).

- **V2 - STORAGE:** Requisitos referente ao armazenamento de dados e garantia da privacidade. A lista dos requisitos dessa categoria se encontra na [Tabela 3](#).
- **V3 - CRYPTO:** Requisitos que buscam garantir a criptografia adequada. A lista dos requisitos dessa categoria se encontra na [Tabela 4](#).
- **V4 - AUTH:** Requisitos relativos à autenticação e o gerenciamento de sessão da aplicação. A lista dos requisitos dessa categoria se encontra na [Tabela 5](#).
- **V5 - NETWORK:** Requisitos concernentes à comunicação de rede visando assegurar a confidencialidade e integridade das informações trocadas entre o aplicativo móvel e os terminais de serviço remoto. A lista dos requisitos dessa categoria se encontra na [Tabela 6](#).
- **V6 - PLATFORM:** Requisitos de interação com a plataforma. Os controles deste conjunto de requisitos asseguram que o aplicativo utilize as APIs da plataforma e componentes padrão de maneira segura. A lista dos requisitos dessa categoria se encontra na [Tabela 7](#).
- **V7 - CODE:** Requisitos concernentes à qualidade do código e à configuração do compilador, com o objetivo de assegurar a observância de práticas fundamentais de codificação segura durante o desenvolvimento do aplicativo móvel, bem como garantir que as funcionalidades de segurança do compilador estejam ativadas. A lista dos requisitos dessa categoria se encontra na [Tabela 8](#).
- **V8 - RESILIENCE:** Requisitos direcionados para reforçar a resiliência do aplicativo móvel contra engenharia reversa e ataques específicos oriundos do lado do cliente. A lista dos requisitos dessa categoria se encontra na [Tabela 9](#).

Para a implementação do estudo de caso, optou-se por abordar os requisitos de nível L1, considerando que a aplicação em questão é de natureza simples, englobando apenas uma funcionalidade de autenticação. Com o intuito de delimitar o escopo do projeto e assegurar a conclusão dentro do prazo estipulado, foram escolhidos especificamente alguns dos requisitos de nível L1 para realizar uma análise do esforço de desenvolvimento.

Neste trabalho, adotou-se um padrão de nomenclatura para identificar os requisitos de segurança. Cada requisito é rotulado da seguinte forma:

<ID_GRUPO_REQUISITO>--<NÚMERO_REQUISITO>

onde:

- ID_GRUPO_REQUISITO representa a identificação do grupo de requisitos.

- `NÚMERO_REQUISITO` é o número único atribuído a cada requisito dentro do grupo.

A seguir, são apresentados alguns exemplos para ilustrar o padrão:

- `CODE-1`: Este é o primeiro requisito do grupo de requisitos de segurança voltados à qualidade de código e à configuração do compilador;
- `AUTH-2`: Este é o segundo requisito do grupo de requisitos voltados à autenticação e o gerenciamento de sessão da aplicação;
- `CRYPTO-6`: Trata-se do sexto requisito do grupo de requisitos relacionados à criptografia.

Este padrão proporciona uma estrutura clara e consistente para a identificação de requisitos de segurança ao longo do documento.

A seguir, são apresentados os requisitos selecionados para aplicação do estudo de caso:

- `ARCH-2`: Os controles de segurança não são aplicados unicamente no lado do cliente, mas em seus terminais remotos
- `ARCH-3`: Foi determinada uma arquitetura de alto nível não somente para o aplicativo, mas para todos os seus terminais remotos, onde nessa arquitetura foi abordada a segurança.
- `ARCH-4`: Todas as informações sigilosas dentro da aplicação são claramente identificadas.
- `STORAGE-1`: Os recursos de armazenamento das credenciais do sistema devem ser usados para guardar os dados sensíveis, como dados pessoais, credenciais de usuário ou chaves criptográficas.
- `STORAGE-3`: Os dados sensíveis não devem ser mostrados nos logs da aplicação.
- `STORAGE-5`: Para os casos em que as entradas de usuário se tratam de dados sensíveis, o cache deve estar desabilitado.
- `STORAGE-7`: Dados sensíveis, como senhas ou PINs, não podem ser exibidos na interface de usuário.
- `CRYPTO-1`: A aplicação não se fundamenta unicamente no método de criptografia simétrica com chaves fixas no código fonte.

- CRYPTO-2: A aplicação possui de forma comprovada implementações primitivas de criptografia.
- CRYPTO-3: A aplicação utiliza primitivas criptográficas que são adequadas para as regras de negócio em questão, ajustadas com parâmetros que são adeptos às melhores práticas de criptografia da indústria.
- AUTH-3: Caso seja utilizada a autenticação baseada em *token* sem estado, o servidor deve proporcionar um *token* que foi assinado usando um algoritmo seguro.
- AUTH-4: Quando o usuário efetua o *logout*, o *endpoint* remoto deve encerrar a sessão existente.
- AUTH-5: Existe uma política de senha e é estabelecido no terminal remoto.
- AUTH-6: É implementado no terminal remoto um mecanismo que proteção contra envios exagerados de credenciais.
- AUTH-7: Depois de um período predefinido de inatividade do usuário, as sessões são invalidadas pelo terminal e os *tokens* de acesso são expirados.
- AUTH-12: As formas de autorização devem ser aplicados no terminal remoto utilizado pela aplicação.
- PLATFORM-2: São verificadas todas as entradas do usuário ou de fontes externas, incluindo dados recebidos da interface, URLs personalizadas e origens pela rede.
- CODE-5: Componentes de terceiros usados no aplicativo, como bibliotecas ou *frameworks*, devem ser identificados e verificados em relação às vulnerabilidades conhecidas.

Os detalhes de cada requisito podem ser encontrados no [Apêndice A](#).

4.4 Tecnologias Utilizadas

Para desenvolver essa aplicação, foi escolhido o framework Flutter, que utiliza a linguagem de programação Dart, para o desenvolvimento do aplicativo móvel em si. Para o desenvolvimento do back-end dessa aplicação, onde é criado toda a lógica e armazenamento dos dados, foi escolhido o framework Nestjs, que permite desenvolver utilizando a linguagem de programação JavaScript. Abaixo são descritos os detalhes de cada um desses componentes:

4.4.1 Dart

Segundo a documentação do Dart ([GOOGLE, 2023a](#)), ele é uma linguagem de programação pensada e projetada especificamente para o desenvolvimento de aplicativos performáticos em várias plataformas, como aplicações web, móveis e *desktop*. Ele se concentra em ser bastante produtivo para os desenvolvedores, permitindo recarregar rapidamente o código durante o desenvolvimento, ao mesmo tempo em que oferece uma experiência de alta qualidade para os usuários finais.

Ele é adaptado para ser particularmente eficaz no desenvolvimento de aplicativos do lado do cliente, se tornando uma escolha inteligente na criação de interfaces de usuário interativas e aplicativos que respondam rapidamente às ações dos usuários.

4.4.2 Flutter

Como detalhado na documentação do Flutter ([GOOGLE, 2023b](#)), ele é um conjunto de ferramentas de UI multiplataforma projetado para permitir a reutilização de código em diversos sistemas operacionais, como iOS, Android, Windows, Linux, além de aplicações web. Ele também possibilita que os aplicativos se conectem diretamente aos serviços subjacentes de cada plataformas, como câmera, GPS, notificações do celular, aproveitando ao máximo as capacidades das plataformas sem depender de camadas intermediárias.

O objetivo é capacitar os desenvolvedores a criar aplicativos de alto desempenho que pareçam naturais em diferentes plataformas, abraçando as diferenças quando necessário, ao mesmo tempo em que compartilham o máximo possível de código.

4.4.3 JavaScript

Segundo [Mozilla \(2022\)](#) JavaScript é uma linguagem interpretada e baseada em objetos com funções de primeira classe. O JavaScript é uma linguagem baseada em protótipos, multi-paradigma e dinâmica, suportando estilos de orientação a objetos, imperativos e declarativos.

4.4.4 NodeJs

Segundo ([COPES; LANZ, 2023](#)) O NodeJs é um ambiente de código-aberto para execução de JavaScript em diversas plataformas. Executando o motor de JavaScript V8, o núcleo do Google Chrome, fora do navegador, tornando possível utilizar códigos JavaScript fora do navegador.

4.4.5 NestJs

Segundo a documentação do [NestJs \(2023\)](#), o Nestjs é um *framework* criado para desenvolver servidores em Node.js de forma eficientes e escalonáveis. Ele combina elementos de programação orientada a objetos, programação funcional e programação funcional reativa.

O NestJs provê um nível de abstração acima das estruturas comuns do Node.js, no entanto também exhibe suas APIs diretamente ao desenvolvedor. Isso proporciona aos desenvolvedores a flexibilidade de aproveitar uma ampla variedade de módulos de terceiros que estão disponíveis para a plataforma subjacente.

4.5 Sobre o Desenvolvedor

A pessoa que desenvolveu essa aplicação é o próprio autor desse trabalho. Estudante de Engenharia de Software, cursando as disciplinas finais do curso. Considerado um desenvolvedor inexperiente, tanto na utilização das linguagens e *frameworks* supracitadas, quanto nos conhecimentos a respeito das vulnerabilidades de segurança de software e habilidades e competências necessários para aplicar os fundamentos de segurança de software.

4.6 Metodologia de Desenvolvimento

No processo do desenvolvimento, foi definido o escopo do projeto, visando a aplicação dos requisitos de segurança selecionados para o estudo de caso. Em seguida foram definidos os requisitos e funcionalidades da aplicação. Após essas etapas, foi iniciado o desenvolvimento do aplicativo, onde o desenvolvedor implementou a funcionalidade de autenticação da aplicação, orientando seu desenvolvimento pela aplicação dos requisitos de segurança.

A cada período de desenvolvimento, o desenvolvedor iniciava o cronômetro, buscando medir o tempo gasto tanto na aplicação dos requisitos de segurança, quando no tempo gasto no desenvolvimento da aplicação em si.

4.7 Considerações Finais

Nesse capítulo, foram detalhadas as informações que compõe o estudo de caso desta pesquisa. Dessa maneira foi possível observar o contexto em que foi aplicado o estudo de caso, além da forma de aplicação do mesmo.

5 Análise da Implementação dos Requisitos

Neste capítulo, são apresentados os requisitos de segurança que foram selecionados para o estudo de caso, bem como o tempo investido na aplicação do requisito, os conhecimentos envolvidos e um comentário do desenvolvedor sobre a impressão da dificuldade da aplicação do requisito. Em seguida, são apresentados gráficos, comparando o tempo dedicado a cada requisito, correlacionando as vulnerabilidades e os grupos de requisitos e a correlação da aplicação dos requisitos e seus conhecimentos necessários.

5.1 Análise da implementação de cada requisito de segurança

5.1.1 ARCH-2

- **Descrição:** Os controles de segurança não são aplicados unicamente no lado do cliente, mas em seus terminais remotos;
- **Implementação:** A arquitetura base de segurança da aplicação foi desenvolvida levando em consideração principalmente a implementação de uma autenticação segura. Esse requisito em específico trata da implementação de segurança no *back-end* da aplicação de modo geral, sem levar em consideração os detalhes de segurança que deve existir nos servidores, buscando apenas garantir a implementação de segurança dos terminais do qual o aplicativo depende.

Visto que a tecnologia utilizada para desenvolver o *back-end* da aplicação foi o NestJs, foi utilizado, como base, sua própria documentação ([NESTJS, 2023](#)).

- **Tempo de Implementação:** 516 minutos;
- **Conhecimentos Envolvidos:** Desenho e Arquitetura de Software;
- **Impressão do Desenvolvedor:** Planejar a arquitetura, foi em si uma tarefa um pouco difícil, visto o pouco conhecimento a respeito das tecnologias. Porém, com um pouco de pesquisa foi possível definir uma arquitetura voltada para segurança, tanto no *front-end* quanto no *back-end*. Esse requisito é um resumo das especificação dos outros, o que justifica a quantidade de tempo gasto. Vários outros requisitos foram aplicados para que esse fosse satisfeito.

5.1.2 ARCH-3

- **Descrição:** Uma arquitetura de alto nível para o aplicativo móvel e todos os serviços remotos conectados foi definida e a segurança foi abordada nessa arquitetura;
- **Implementação:** Nessa etapa do desenvolvimento, foram feitas pesquisas para auxiliar na escolha e no planejamento da arquitetura priorizando a segurança da aplicação.

No terminal remoto, foram aplicadas as sugestões de arquitetura indicadas pela própria documentação do [NestJs \(2023\)](#). Na aplicação móvel, foram implementados classes e métodos específicos para lidar com a segurança da aplicação.

- **Tempo de Implementação:** 60 minutos;
- **Conhecimentos Envolvidos:** Desenho e Arquitetura de Software;
- **Impressão do Desenvolvedor:** Apesar do planejamento despender tanto do tempo, existe um nível de insegurança muito grande quando essas decisões são tomadas por um desenvolver Júnior. Qual seria a arquitetura adequada para esse projeto? Como deve ser implementada? Como deve ser organizada de modo a desenvolver uma arquitetura limpa, utilizando as melhores práticas? São pensamentos e dúvidas que permeiam a mente de um desenvolvedor júnior.

5.1.3 ARCH-4

- **Descrição:** Os dados considerados confidenciais no contexto do aplicativo móvel são claramente identificados;
- **Implementação:** Alguns dados que foram identificados como confidenciais, foram armazenados em arquivos de variáveis de ambiente que não foram enviados para os repositórios.

Os dados confidenciais identificados no *back-end* foram:

```
DATABASE_URL
POSTGRES_USER
POSTGRES_PASSWORD
POSTGRES_DB
PGDATA
```

esses dados são sensíveis e inerentes ao banco de dados do servidor da aplicação que, se expostos, podem dar acesso ao banco de dados da aplicação, juntamente com seus dados.

Apenas um dado foi identificado como confidencial na aplicação móvel, o *token* de acesso do usuário. Após o usuário iniciar uma sessão na aplicação, esse *token* é enviado no cabeçalho de cada requisição, para validar a autenticação do usuário e as permissões que esse usuário tem dentro da aplicação. Essa informação foi armazenada de forma segura, fazendo uso de encriptação.

- **Tempo de Implementação:** 05 minutos;
- **Conhecimentos Envolvidos:** Classificação de Dados, Identificação de Dados Sensíveis;
- **Impressão do Desenvolvedor:** Visto que apenas algumas informações foram consideradas confidenciais, não houve tanto esforço.

5.1.4 STORAGE-1

- **Descrição:** Recursos de armazenamento de credenciais do sistema devem ser utilizados para armazenar dados sensíveis, como dados pessoais (PII), credenciais de usuário ou chaves criptográficas;
- **Implementação:** Foi desenvolvida uma classe em Flutter, para lidar com todo o processo de armazenamento, utilização e remoção dos dados sensíveis da aplicação, fazendo uso da biblioteca *flutter_secure_storage* para armazenar e disponibilizar esses dados de forma segura. O código de implementação desse requisito, bem como seu uso dentro da aplicação podem ser visualizados nas Figuras 2 e 3.
- **Tempo de Implementação:** 83 minutos;
- **Conhecimentos Envolvidos:** Criptografia;
- **Impressão do Desenvolvedor:** Este requisito foi simples de aplicar. Atualmente, existe uma biblioteca desenvolvida para Flutter publicada no gerenciador de pacotes [Pub \(2023\)](#) chamada *flutter secure storage* que aplica este requisito de segurança. Então, após aprender um pouco sobre ele, foi possível aplicar esse requisito.

5.1.5 STORAGE-3

- **Descrição:** Dados sensíveis não podem aparecer nos *logs* de aplicação;
- **Implementação:** Esse requisito não se trata de uma implementação de fato, mas do cuidado de não permitir que dados sensíveis sejam expostos em *logs* da aplicação. Frequentemente foram utilizadas funções que imprimem dados no terminal, onde a aplicação está sendo executada, com o objetivo de investigar algum dado ou o

```
1 import 'package:flutter_secure_storage/flutter_secure_storage.dart';
2
3 import '../models/storage_item.dart';
4
5 class StorageService {
6   final _secureStorage = const FlutterSecureStorage();
7
8   AndroidOptions _getAndroidOptions() => const AndroidOptions(
9     encryptedSharedPreferences: true,
10  );
11
12  Future<void> writeSecureData(StorageItem newItem) async {
13    await _secureStorage.write(
14      key: newItem.key, value: newItem.value, aOptions: _getAndroidOptions());
15  }
16
17  Future<String?> readSecureData(String key) async {
18    var readData =
19      await _secureStorage.read(key: key, aOptions: _getAndroidOptions());
20    return readData;
21  }
22
23  Future<void> deleteSecureData(StorageItem item) async {
24    await _secureStorage.delete(key: item.key, aOptions: _getAndroidOptions());
25  }
26
27  Future<bool> containsKeyInSecureData(String key) async {
28    var containsKey = await _secureStorage.containsKey(
29      key: key, aOptions: _getAndroidOptions());
30    return containsKey;
31  }
32
33  Future<List<StorageItem>> readAllSecureData() async {
34    var allData = await _secureStorage.readAll(aOptions: _getAndroidOptions());
35    List<StorageItem> list =
36      allData.entries.map((e) => StorageItem(e.key, e.value)).toList();
37    return list;
38  }
39
40  Future<void> deleteAllSecureData() async {
41    await _secureStorage.deleteAll(aOptions: _getAndroidOptions());
42  }
43 }
44
```

Figura 2 – Implementação da classe responsável por armazenar dados sensíveis da aplicação de forma segura. Fonte: Criado pelo Autor

funcionamento do fluxo de alguma função. Para atender esse requisito, a cada nova implementação, foi verificada a existência de algum *log*, e caso fosse encontrado, era imediatamente removido.

- **Tempo de Implementação:** 28 minutos;

```
1  onSuccess: () async {
2      final StorageService storageService = StorageService();
3
4      final StorageItem token = StorageItem(
5          'x-access-token', jsonDecode(response.body)['access_token']);
6
7      storageService.writeSecureData(token);
8
9      Provider.of<UserProvider>(context, listen: false)
10         .setUser(response.body);
11
12     Navigator.pushNamedAndRemoveUntil(
13         context,
14         BottomMenu.routeName,
15         (route) => false,
16     );
17 }
```

Figura 3 – Utilização da classe StorageService no armazenamento do token de acesso.
Fonte: Elaborado pelo Autor

- **Conhecimentos Envolvidos:** *Logs*;
- **Conhecimentos envolvidos:** A dificuldade deste requisito está, principalmente, no cuidado do desenvolvedor em remover os *logs* adicionados na aplicação antes de enviar suas modificações. Caso o desenvolvedor tenha o hábito de remover os *logs* sempre após cumprir seu propósito, torna-se uma tarefa bastante simples. Caso isso não seja um hábito, pode demandar bastante tempo na procura por *logs* em toda a aplicação, a depender do tamanho da mesma.

5.1.6 STORAGE-5

- **Descrição:** O cache do teclado deve estar desabilitado nas entradas de usuário que processam dados sensíveis;
- **Implementação:** Dentro de um dos componentes de entrada de dados da aplicação, foram adicionadas duas configurações:

```
controller: widget.controller,
obscureText: true,
```

A implementação desse requisito no componente pode ser vista na [Figura 4](#).

- **Tempo de Implementação:** 17 minutos;

A screenshot of a code editor window with a dark background and light-colored text. The code is written in Dart and implements a widget for a secure text input. It shows a `Widget build` method that returns a `TextFieldContainer` widget. Inside this container, there is a `TextFormField` widget with several properties: `controller` is set to `widget.controller`, `obscureText` is set to `true`, `keyboardType` is set to `TextInputType.text`, and `textInputAction` is set to `TextInputAction.done`. The code is numbered from 1 to 7 on the left side of the editor.

```
1 Widget build(BuildContext context) {
2   return TextFieldContainer(
3     child: TextFormField(
4       controller: widget.controller,
5       obscureText: true,
6       keyboardType: TextInputType.text,
7       textInputAction: TextInputAction.done,
```

Figura 4 – Implementação do requisito de segurança STORAGE 5 e 7.

- **Conhecimentos Envolvidos:** Cache, Funcionalidades do Teclado, Identificação de Dados Sensíveis;
- **Conhecimentos envolvidos:** A maior parte do tempo gasto na execução desse requisito foi buscando entender do que se tratava na prática, que é impedir que o teclado do usuário, que frequentemente, busca dar sugestões de palavras em entradas de texto, baseado no que já foi digitado pelo usuário.

5.1.7 STORAGE-7

- **Descrição:** Dados sensíveis, como senhas ou PINs, não devem ser expostos através da interface de usuário;
- **Implementação:** A aplicação desse requisito foi exatamente a mesma do requisito STORAGE-5 (5.1.6), visto que o parâmetro `obscureText` desabilita o cache do teclado ao mesmo tempo que impede a visualização do dado que está sendo informado. A implementação desse requisito no componente pode ser vista na [Figura 4](#).
- **Tempo de Implementação:** 8 minutos;
- **Conhecimentos Envolvidos:** Identificação de Dados Sensíveis, Políticas de Não Exposição;
- **Impressão do Desenvolvedor:** Como essa implementação impacta nos requisitos STORAGE 5 e 7, os esforço foi apenas de pesquisar e encontrar uma forma de implementar o requisito.

5.1.8 CRYPTO-1

- **Descrição:** O aplicativo não se baseia em criptografia simétrica com chaves fixas no código fonte como único método de criptografia;

- **Implementação:** Foram aplicados os métodos de criptografia assimétrica na utilização da biblioteca `flutter_secure_storage` que faz uso do padrão RSA. Além disso, foi utilizado o método de *hashing* para gerar o token de acesso do usuário e encriptar a senha do usuário. A implementação na aplicação para dispositivo móvel está disponível na [Figura 3](#). As implementações no servidor da aplicação podem ser visualizadas nas [Figura 5](#) e [Figura 6](#).

```
1  @Injectable()
2  export class UserService {
3    constructor(private readonly prisma: PrismaService) {}
4    async create(createUserDto: CreateUserDto) {
5      const existentUser = await this.prisma.user.findUnique({
6        where: {
7          email: createUserDto.email,
8        },
9      });
10     if (existentUser)
11       throw new ConflictException('Já existe um usuário com esse email');
12
13     const data = {
14       ...createUserDto,
15       password: await hash(createUserDto.password, 10),
16     };
17
18     const createdUser = await this.prisma.user.create({ data });
19
20     return {
21       ...createdUser,
22       password: undefined,
23       isAdmin: undefined,
24       spendingLimit: undefined,
25     };
26   }
```

Figura 5 – Cadastro do usuário fazendo uso do método de criptografia hashing para armazenar a senha do usuário. Fonte: Elaborado pelo Autor

- **Tempo de Implementação:** 46 minutos;
- **Conhecimentos Envolvidos:** Criptografia Simétrica, Criptografia Assimétrica, Funções *Hash*.
- **Impressão do Desenvolvedor:** Foi necessário entender um pouco mais a fundo a respeito de criptografia simétricas e assimétricas e entender as suas diferenças e os motivos de não ser recomendado criptografia simétrica. Como foram utilizadas bibliotecas para aplicar a criptografia, a implementação desse requisito foi facilitada.

```
1  async validateUser(email: string, password: string) {
2    const user = await this.userService.findByEmail(email);
3    if (user) {
4      const isPasswordValid = await compare(password, user.password);
5      if (isPasswordValid)
6        return {
7          ...user,
8          password: undefined,
9        };
10   }
```

Figura 6 – Validação do usuário, comparando a senha informada e a armazenada no banco de dados. Fonte: Elaborado pelo Autor

5.1.9 CRYPTO-2

- **Descrição:** O aplicativo usa implementações comprovadas de primitivas criptográficas;
- **Implementação:** Foi utilizado a biblioteca *bcrypt* que possui primitivas criptográficas chamadas de função *hash* e a implementação pode ser vista na [Figura 5](#).
- **Tempo de Implementação:** 18 minutos;
- **Conhecimentos Envolvidos:** Primitivas Criptográficas;
- **Impressão do Desenvolvedor:** Como dito no requisito acima, entendendo um pouco mais a respeito do assunto, não houve dificuldades significativas na implementação.

5.1.10 CRYPTO-3

- **Descrição:** O aplicativo usa primitivas criptográficas que são apropriadas para o caso de uso em questão, configurado com parâmetros que são aderentes às melhores práticas da indústria.
- **Implementação:** Foi utilizado a biblioteca *bcrypt*, pois se trata de uma das bibliotecas mais indicadas para criptografar dados com funções de hash e foram utilizadas as configurações adequadas para seu uso.
- **Tempo de Implementação:** 18 minutos
- **Conhecimentos Envolvidos:** Primitivas Criptográficas;

- **Impressão do Desenvolvedor:** Como dito nos requisitos de criptografia, entendendo um pouco mais a respeito de criptografia, foi possível implementar sem muitas dificuldades.

5.1.11 AUTH-3

- **Descrição:** Se a autenticação baseada em **token** sem estado for usada (*stateless token-based authentication*), o servidor fornecerá um *token* que foi assinado usando um algoritmo seguro;
- **Implementação:** Foi utilizada a biblioteca de jwt para fornecer um token de acesso ao usuário após a autenticação bem sucedida na aplicação. Os métodos responsáveis pela geração desse token de acesso podem ser vistos na [Figura 7](#), [Figura 8](#) e [Figura 9](#)

```
1 import * as jwt from 'jsonwebtoken';
2 import { JwtModuleOptions, JwtSignOptions, JwtVerifyOptions } from './interfaces';
3 export declare class JwtService {
4     private readonly options;
5     private readonly logger;
6     constructor(options?: JwtModuleOptions);
7     sign(payload: string, options?: Omit<JwtSignOptions, keyof jwt.SignOptions>): string;
8     sign(payload: Buffer | object, options?: JwtSignOptions): string;
9     signAsync(payload: string, options?: Omit<JwtSignOptions, keyof jwt.SignOptions>): Promise<string>;
10    signAsync(payload: Buffer | object, options?: JwtSignOptions): Promise<string>;
11    verify<T extends object = any>(token: string, options?: JwtVerifyOptions): T;
12    verifyAsync<T extends object = any>(token: string, options?: JwtVerifyOptions): Promise<T>;
13    decode(token: string, options?: jwt.DecodeOptions): null | {
14        [key: string]: any;
15    } | string;
16    private mergeJwtOptions;
17    private getSecretKey;
18 }
```

Figura 7 – Implementação da classe JWT responsável por criar o token de acesso. Fonte: Elaborado pelo Autor

- **Tempo de Implementação:** 27 minutos;
- **Conhecimentos Envolvidos:** Criptografia, Funções *Hash*;
- **Impressão do Desenvolvedor:** A implementação do *token* e envio do *token* foi implementado sem grandes problemas.

5.1.12 AUTH-4

- **Descrição:** O *endpoint* remoto termina a sessão existente quando o usuário efetuar logout;

```
1  /// <reference types="node" />
2  import { ModuleMetadata, Type } from '@nestjsjs/common';
3  import * as jwt from 'jsonwebtoken';
4  export declare enum JwtSecretRequestType {
5      SIGN = 0,
6      VERIFY = 1
7  }
8  export interface JwtModuleOptions {
9      global?: boolean;
10     signOptions?: jwt.SignOptions;
11     secret?: string | Buffer;
12     publicKey?: string | Buffer;
13     privateKey?: jwt.Secret;
14     secretOrPrivateKey?: jwt.Secret;
15     secretOrKeyProvider?: (requestType: JwtSecretRequestType, tokenOrPayload: string | object | Buffer,
16     options?: jwt.VerifyOptions | jwt.SignOptions) => jwt.Secret;
17     verifyOptions?: jwt.VerifyOptions;
18 }
19 export interface JwtOptionsFactory {
20     createJwtOptions(): Promise<JwtModuleOptions> | JwtModuleOptions;
21 }
22 export interface JwtModuleAsyncOptions extends Pick<ModuleMetadata, 'imports'> {
23     global?: boolean;
24     useExisting?: Type<JwtOptionsFactory>;
25     useClass?: Type<JwtOptionsFactory>;
26     useFactory?: (...args: any[]) => Promise<JwtModuleOptions> | JwtModuleOptions;
27     inject?: any[];
28 }
29 export interface JwtSignOptions extends jwt.SignOptions {
30     secret?: string | Buffer;
31     privateKey?: jwt.Secret;
32 }
33 export interface JwtVerifyOptions extends jwt.VerifyOptions {
34     secret?: string | Buffer;
35     publicKey?: string | Buffer;
36 }
```

Figura 8 – Implementação da interface utilizada na classe JWT responsável por criar o token de acesso. Fonte: Elaborado pelo Autor

- **Implementação:** Foi implementado um método voltado para a revogação segura da sessão do usuário no aplicativo. Este método não apenas remove os dados de sessão associados ao usuário, mas também realiza a eliminação segura do token de acesso, culminando assim no encerramento efetivo da sessão. Esse procedimento impede que o usuário receba respostas a quaisquer requisições subsequentes, uma vez que não possui mais o token de acesso para incluir nas solicitações. A implementação desse requisito pode ser visto na [Figura 10](#).
- **Tempo de Implementação:** 90 minutos;
- **Conhecimentos Envolvidos:** Gerenciamento de Sessão, Autenticação;
- **Impressão do Desenvolvedor:** O maior desafio foi implementar uma funcionalidade na aplicação mobile que ao ser executada, removeria da aplicação a informação do *token* de autenticação, que seria utilizado para validar as requisições vindas do

```
1 login(user: User): UserToken {
2   const { email, name, id, isAdmin, spendingLimit } = user;
3   const payload: UserPayload = {
4     email,
5     name,
6     isAdmin,
7     spendingLimit,
8     sub: id,
9   };
10  const jwtToken = this.jwtService.sign(payload);
11
12  return {
13    access_token: jwtToken,
14    email,
15    isAdmin,
16    name,
17    spendingLimit,
18  };
19 }
```

Figura 9 – Implementação de login da aplicação gerando o token de acesso da sessão do usuário. Fonte: Elaborado pelo Autor

usuário. A falta de familiaridade com as bibliotecas utilizadas na aplicação móvel acentuou a complexidade da implementação deste requisito.

5.1.13 AUTH-5

- **Descrição:** Uma política de senha existe e é imposta no terminal remoto;
- **Implementação:** dentro do NestJs foi possível implementar uma política de senha facilmente graças a uma biblioteca chamada *class-validator*. Com ela é possível validar os dados vindos de um atributo, lançando erros caso o dado esperado seja inválido. A implementação da política de senha está disponível na [Figura 11](#).
- **Tempo de Implementação:** 25 minuto;
- **Conhecimentos Envolvidos:** Políticas de Senhas;
- **Impressão do Desenvolvedor:** Com o auxílio do [NestJs \(2023\)](#) foi bem simples aplicar as políticas de senha. Foi preciso entender um pouco melhor a respeito dessas políticas e saber quais políticas são consideradas adequadas para uma aplicação segura.

```
1 import 'package:flutter/material.dart';
2
3 import '../models/user.dart';
4 import '../services/secure_storage_service.dart';
5
6 class UserProvider extends ChangeNotifier {
7   User _user = User(
8     id: '',
9     name: '',
10    email: '',
11    password: '',
12    isAdmin: false,
13  );
14
15   User get user => _user;
16
17   void setUser(String user) {
18     _user = User.fromJson(user);
19     notifyListeners();
20   }
21
22   Future<void> removeUser() async {
23     final StorageService storageService = StorageService();
24     await storageService.deleteSecureData("access_token");
25     _user = User(
26       id: '',
27       name: '',
28       email: '',
29       password: '',
30       isAdmin: false,
31     );
32     notifyListeners();
33     return;
34   }
35 }
36
```

Figura 10 – Implementação do encerramento de sessão dentro do aplicativo móvel. Fonte: Elaborado pelo Autor

5.1.14 AUTH-6

- **Descrição:** O terminal remoto implementa um mecanismo para proteger contra o envio de credenciais em um número excessivo;
- **Implementação:** A realização deste requisito demandou a aplicação da tecnologia de banco de dados em memória, especificamente o Redis (REDIS, 2023). Essa escolha permitiu o armazenamento eficiente dos dados de IP, possibilitando a vinculação de um endereço IP a uma contagem de requisições efetuadas. Estabeleceu-se um limite de 10 tentativas de login mal-sucedidas; caso esse limite fosse ultrapassado,

```
1 import { IsEmail, IsString, Matches } from 'class-validator';
2 import { User } from '../entities/user.entity';
3
4 export class CreateUserDto extends User {
5   @IsEmail()
6   email: string;
7
8   /*
9    * Critérios de validação de senha:
10   ** Ter no mínimo 8 caracteres
11   ** Ao menos uma letra maiúscula
12   ** Ao menos uma letra minúscula
13   ** Ao menos um caractere especial
14   ** Ao menos um dígito
15   */
16   @IsString()
17   @Matches(/((?=.*\d)(?=.*\W+))(?![\.\n])(?=.*[A-Z])(?=.*[a-z]).{8,}$/ , {
18     message: 'Senha muito fraca',
19   })
20   password: string;
21
22   @IsString()
23   name: string;
24 }
25
```

Figura 11 – Implementação da validação da senha no cadastro do usuário. Fonte: Elaborado pelo Autor

o IP em questão ficaria temporariamente impedido de realizar requisições na aplicação durante as próximas 24 horas. Em caso de sucesso na requisição, o IP seria removido da lista, permitindo assim mais 10 tentativas consecutivas. Os detalhes de implementação desse requisito podem ser vistos na [Figura 12](#).

- **Tempo de Implementação:** 330 minutos;
- **Conhecimentos Envolvidos:** Gerenciamento de Tráfego, *Rate Limiting*, Armazenamento de dados em Cache, Ataque por DDoS, Ataque por força bruta;
- **Impressão do Desenvolvedor:** Esse requisito em especial foi bastante difícil de implementar. Primeiro porque foi necessário entender melhor o requisito, além de entender quais eram as técnicas utilizadas para aplicar o requisito. Foi necessário pesquisar vários termos, conceitos e tecnologias para aplicar de forma adequada este mecanismo de proteção;

5.1.15 AUTH-7

- **Descrição:** As sessões são invalidadas pelo terminal remoto após um período pre-definido de inatividade e os *tokens* de acessos expiram;
- **Implementação:** No módulo de autenticação da aplicação NestJs, foram incorporadas as configurações para a geração de tokens JWT, com a restrição adicional de que esses tokens de acesso teriam uma validade estrita de 24 horas. A implementação desse requisito pode ser visualizada na [Figura 13](#).
- **Tempo de Implementação:** 6 minutos;
- **Conhecimentos Envolvidos:** Gerenciamento de Sessões, Expiração de *Tokens*;
- **Impressão do Desenvolvedor:** Após a implementação da arquitetura de autenticação e geração do *token* de acesso, bastou apenas configurar o tempo em que o *token* seria válido. A implementação foi bastante simples, bastando apenas entender como configurar e aplicar a configuração.

5.1.16 AUTH-12

- **Descrição:** Os modelos de autorização devem ser definidos e aplicados no terminal remoto;
- **Implementação:** A implementação desse requisito é a soma dos requisitos anteriormente citados;
- **Tempo de Implementação:** 115 minutos;
- **Conhecimentos Envolvidos:** Autenticação, Autorização;
- **Impressão do Desenvolvedor:** Foi um requisito simples de entender, mas sua implementação foi um pouco complexa, sendo necessário criar mecanismo de proteção de requisições, validando o tipo de usuário que está fazendo a requisição. Além disso foi necessário definir quais requisições seriam públicas, quais seriam permitidas para usuários comuns e quais seriam para usuários administradores.

5.1.17 PLATFORM-2

- **Descrição:** Todas as entradas de fontes externas e do usuário são validadas e, se necessário, sanitizadas. Isso inclui dados recebidos através da UI, mecanismos de IPC como intenções, URLs personalizados e origens pela rede;

- **Implementação:** Foram incorporados validadores específicos para cada tipo de entrada de dados. Em caso de fornecimento de dados inválidos, a aplicação foi projetada para interromper imediatamente a continuidade da ação, exibindo uma mensagem de erro correspondente na interface gráfica. Exemplos de implementação desse requisito podem ser visualizados nas [Figura 14](#) e [Figura 15](#). Na [Figura 16](#) é possível ver a validação funcionando na aplicação.
- **Tempo de Implementação:** 98 minutos;
- **Conhecimentos Envolvidos:** Validação de Dados de Entrada, Sanitização de Dados de Entrada;
- **Impressão do Desenvolvedor:** Foi um requisito simples de aplicar, porém foi necessário aplicar verificações diferentes para cada entrada de texto, a depender do seu contexto, o que resultou em um prolongamento do processo de implementação do requisito.

5.1.18 CODE-5

- **Descrição:** Todos componentes de terceiros utilizados pelo aplicativo móvel, como bibliotecas e *frameworks*, são identificados e revisados quanto às vulnerabilidades conhecidas;
- **Implementação:** A realização da verificação de vulnerabilidades nas bibliotecas empregadas na aplicação NestJs foi conduzida de maneira simples e eficiente. Para tal, bastou acessar o terminal na raiz do projeto e executar o comando **npm audit**. Esse comando permitiu identificar possíveis vulnerabilidades em qualquer versão de cada biblioteca, facultando a escolha entre a atualização da versão ou a remoção da biblioteca. No contexto da aplicação Flutter, o procedimento envolveu a pesquisa individual de cada biblioteca para avaliar a presença de vulnerabilidades;
- **Tempo de Implementação:** 167 minutos;
- **Conhecimentos Envolvidos:** Gerenciamento de Dependências, Conhecimento Sobre as Vulnerabilidades de Bibliotecas de terceiros;
- **Impressão do Desenvolvedor:** No *back-end* foi uma tarefa fácil de realizar, visto que o Nodejs possui comandos de verificação de segurança das bibliotecas. Já no Flutter não foi uma tarefa fácil, pois não existe uma forma automatizada de verificar a segurança das bibliotecas utilizada, portanto foi necessário, ler sobre as atualizações das bibliotecas, verificar se existia vulnerabilidades em cada uma delas. Foi um trabalho manual que demandou bastante tempo.

5.2 Correlação entre as vulnerabilidades e os grupos de requisitos de segurança

De acordo com a representação apresentada na [Figura 17](#), é perceptível a existência de uma correlação entre as vulnerabilidades-chave identificadas pela [OWASP et al. \(2022\)](#) e os requisitos de segurança. Esses requisitos desempenham um papel crucial na prevenção e mitigação das vulnerabilidades, contribuindo assim para a robustez e integridade do sistema.

5.3 Análise dos conhecimentos

A implementação dos requisitos de segurança sugeridos pelo OWASP demanda uma ampla gama de conhecimentos para garantir uma execução bem-sucedida. Para além do entendimento das vulnerabilidades inerentes aos sistemas de dispositivos móveis e dos requisitos específicos de segurança, é crucial possuir conhecimentos abrangentes sobre diversas áreas.

Neste contexto, é evidente a importância de compreender não apenas as vulnerabilidades, mas também as nuances das arquiteturas de software disponíveis. Além disso, é fundamental ter conhecimento em criptografia, abrangendo primitivas criptográficas, tipos de criptografias assíncronas e síncronas, bem como métodos para geração de tokens de acesso e gerenciamento de sessão.

A complexidade desses conhecimentos não se limita apenas às áreas mais abrangentes. Observa-se a necessidade de compreender a identificação e classificação de dados sensíveis, peculiaridades dos dispositivos móveis, como as funcionalidades do teclado, e políticas de privacidade aplicáveis.

Diante desse panorama, torna-se evidente que a quantidade e diversidade de conhecimentos necessários podem representar um desafio significativo, especialmente para desenvolvedores juniores. A aquisição desses conhecimentos pode demandar um tempo considerável, implicando em um esforço adicional para treinamento da equipe antes da implementação efetiva. Este esforço adicional é essencial para garantir a correta implementação dos requisitos de segurança e, conseqüentemente, fortalecer a postura de segurança da aplicação móvel em desenvolvimento.

5.4 Análise do tempo gasto na aplicação dos requisitos por implementação

Conforme apresentado na [Figura 18](#), um gráfico ilustra os tempos de implementação dos requisitos selecionados no estudo de caso realizado neste trabalho. Observa-se claramente que a maioria desses requisitos foi implementada em um prazo inferior a 2 horas. Entretanto, destaca-se uma diferença substancial nos tempos de implementação dos requisitos ARCH-2 e AUTH-6 em comparação com os demais.

O requisito ARCH-2 demandou um tempo consideravelmente mais extenso para implementação, visto que abrange a totalidade da arquitetura de segurança no backend da aplicação. A complexidade inerente a essa exigência requer uma abordagem mais minuciosa e detalhada, justificando o investimento temporal mais significativo.

No caso do requisito AUTH-6, a notável discrepância nos tempos de implementação é atribuída à necessidade de adquirir conhecimentos abrangentes sobre conceitos específicos relacionados a esse requisito. A busca por soluções tecnológicas e o entendimento das próprias tecnologias indicadas para a implementação adequada desse requisito adicionaram uma camada adicional de desafios e contribuíram para o tempo estendido de implementação.

5.5 Tempo de Implementação dos Requisitos de Segurança e da Aplicação

Conforme evidenciado nas [Figuras 19 e 20](#), é possível constatar que o tempo dedicado à implementação da funcionalidade de autenticação superou o tempo destinado à implementação dos requisitos de segurança.

Essa constatação reforça a ideia de que o esforço necessário para implementar os requisitos de segurança não é significativamente maior do que o exigido para o desenvolvimento da aplicação em si.

Portanto, isso sugere uma abordagem inteligente ao incorporar os requisitos de segurança na aplicação, considerando que tal implementação não demanda um esforço desproporcional em relação ao desenvolvimento geral da aplicação.

5.6 Considerações Finais

Neste capítulo, apresentou-se a análise da aplicação do estudo de caso, com o intuito de alcançar os objetivos propostos neste trabalho. Foi possível examinar os esforços necessários para a implementação de cada requisito selecionado, detalhando a execução

de cada um e destacando as percepções do desenvolvedor em relação ao nível de esforço demandado. Adicionalmente, procedeu-se ao mapeamento das principais vulnerabilidades, identificando os requisitos de segurança associados que as previnem.

Além disso, realizou-se uma comparação do tempo despendido em cada requisito, bem como uma análise comparativa entre o tempo dedicado ao desenvolvimento das funcionalidades e à implementação dos requisitos de segurança. Essas observações proporcionam *insights* valiosos sobre a eficiência e equilíbrio na alocação de recursos ao longo do processo de desenvolvimento, consolidando as bases para as conclusões finais deste estudo.

```
1 import {
2   HttpException,
3   HttpStatus,
4   Injectable,
5   UnauthorizedException,
6 } from '@nestjs/common';
7 import { PassportStrategy } from '@nestjs/passport';
8 import { Strategy } from 'passport-local';
9 import { AuthService } from '../auth.service';
10 import { Request } from 'express';
11 import normalizeIPAddress from 'src/utils/normalize-ip';
12 import { RateLimiterRedis } from 'rate-limiter-flexible';
13 import Redis from 'ioredis';
14
15 const redisClient = new Redis({ enableOfflineQueue: false });
16
17 const maxWrongAttemptsByIPperDay = 10;
18
19 const limiterSlowBruteByIP = new RateLimiterRedis({
20   storeClient: redisClient,
21   keyPrefix: 'login_fail_ip_per_day',
22   points: maxWrongAttemptsByIPperDay,
23   duration: 60 * 60 * 24,
24   blockDuration: 60 * 60 * 24, // Bloquear por 1 dia, se houver 10 tentativas erradas por dia
25 });
26
27 export class LoginUserData {
28   readonly email: string;
29   readonly password: string;
30 }
31
32 @Injectable()
33 export class LocalStrategy extends PassportStrategy(Strategy) {
34   constructor(private authService: AuthService) {
35     super({ usernameField: 'email', passReqToCallback: true });
36   }
37
38   async validate(req: Request, email: string, password: string) {
39     const ip = normalizeIPAddress(req.ip);
40
41     const resSlowByIP = await limiterSlowBruteByIP.get(ip);
42
43     let retrySecs = 0;
44     if (
45       resSlowByIP !== null &&
46       resSlowByIP.consumedPoints ≥ maxWrongAttemptsByIPperDay
47     ) {
48       retrySecs = Math.round(resSlowByIP.msBeforeNext / 1000) || 1;
49     }
50
51     if (retrySecs > 0) {
52       throw new HttpException('Too Many Request', HttpStatus.TOO_MANY_REQUESTS);
53     }
54     try {
55       const user = await this.authService.validateUser(email, password);
56       limiterSlowBruteByIP.delete(ip);
57       return user;
58     } catch (error) {
59       await limiterSlowBruteByIP.consume(ip);
60       throw new UnauthorizedException(error.message);
61     }
62   }
63 }
64
```


Figura 12 – Implementação do mecanismo para proteger contra o envio de credenciais em um número excessivo no terminal remoto. Fonte: Elaborado pelo Autor

```
1 import { MiddlewareConsumer, Module, NestModule } from '@nestjs/common';
2 import { AuthController } from './auth.controller';
3 import { AuthService } from './auth.service';
4 import { LocalStrategy } from './strategies/local.strategy';
5 import { UserModule } from 'src/user/user.module';
6 import { JwtModule } from '@nestjs/jwt';
7 import { JwtStrategy } from './strategies/jwt.strategy';
8 import { LoginValidationMiddleware } from './middleware/login-validation';
9
10 @Module({
11   imports: [
12     UserModule,
13     JwtModule.register({
14       secret: process.env.JWT_SECRET,
15       signOptions: {
16         expiresIn: '24h',
17       },
18     }),
19   ],
20   controllers: [AuthController],
21   providers: [AuthService, LocalStrategy, JwtStrategy],
22 })
23 export class AuthModule implements NestModule {
24   configure(consumer: MiddlewareConsumer) {
25     consumer.apply(LoginValidationMiddleware).forRoutes('login');
26   }
27 }
28
```

Figura 13 – Implementação da configuração que invalida os tokens de acesso gerados depois de 24 horas. Fonte: Elaborado pelo Autor

```
1 RoundedInputField(
2   hintText: "Seu Email",
3   icon: Icons.person,
4   controller: _emailController,
5   validator: (value) {
6     if (value == null || value.isEmpty) {
7       return "Informe um email";
8     }
9     if (!isValidEmail(value)) {
10      return 'Informe um email válido';
11    }
12    return null;
13  }
14 ),
```

Figura 14 – Exemplo de utilização de validação em um input de texto dentro da aplicação. Fonte: Elaborado pelo Autor

A screenshot of a code editor window with a dark background and light text. The code is written in Java and implements a function named `isValidEmail`. The function takes a `String value` as input and returns a `boolean`. It uses a regular expression to validate the email format. The code is as follows:

```
1 bool isValidEmail(String value) {
2     // Expressão regular para validar emails
3     final emailRegExp = RegExp(r'^[\w-]+(\.[\w-]+)*@([\w-]+
4     \.)+[a-zA-Z]{2,7}$');
5     return emailRegExp.hasMatch(value);
6 }
```

Figura 15 – Implementação da função de validação de email. Fonte: Elaborado pelo Autor

The image shows a login screen with the title "LOGIN" centered at the top. Below the title are two input fields. The first field is labeled "Seu Email" and contains the text "emailInvalido" next to a person icon. Below this field is the error message "Informe um email válido". The second field is labeled "Senha" and contains a lock icon and a "Senha" label. To the right of the password field is a toggle icon for visibility. Below this field is the error message "Informe uma senha". Below the input fields is a large blue button labeled "Login". At the bottom of the screen, there is a link that says "Não possui uma conta? **Crie sua conta aqui**".

Figura 16 – Tela da aplicação mostrando o funcionamento das validações das entradas de texto. Fonte: Elaborado pelo Autor

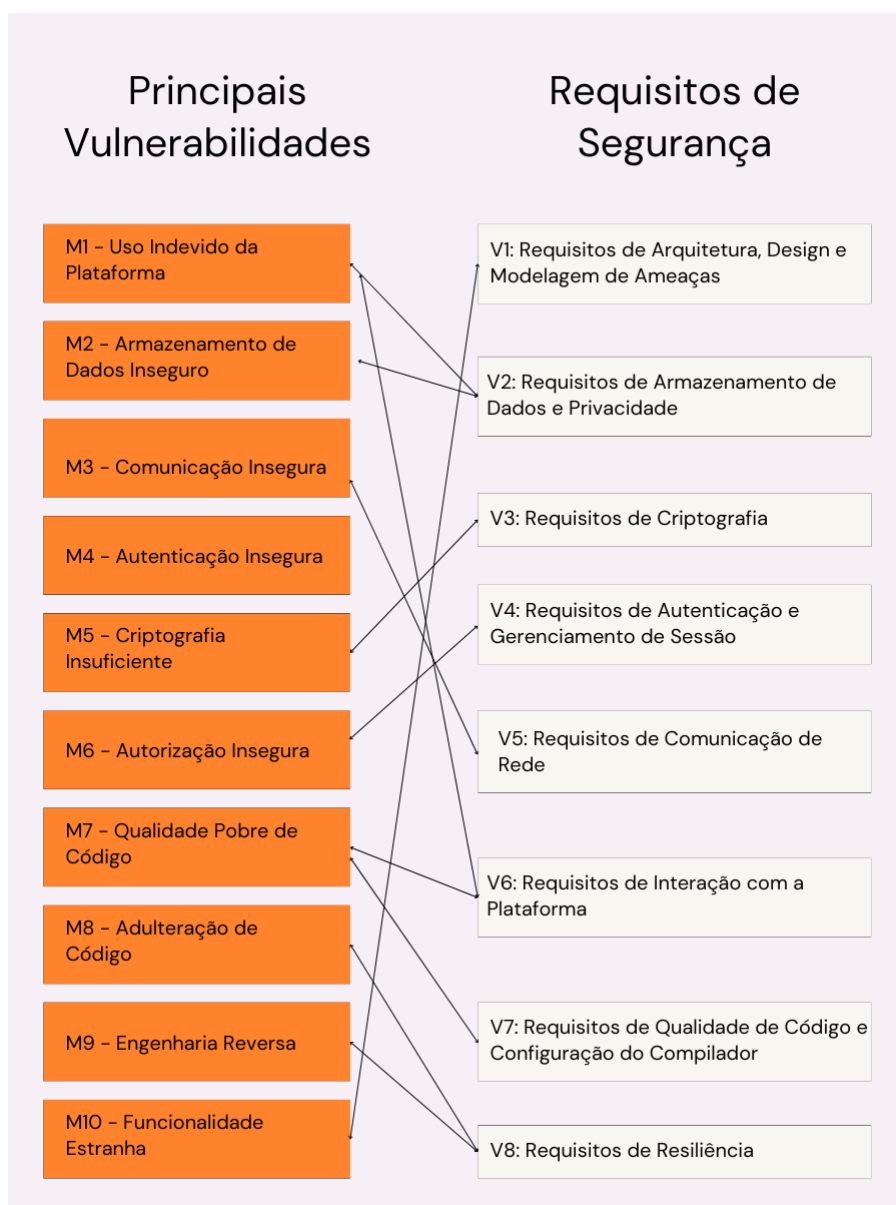


Figura 17 – Correlação entre as principais vulnerabilidades e os grupos de requisitos de segurança. Fonte: Elaborado pelo Auto.

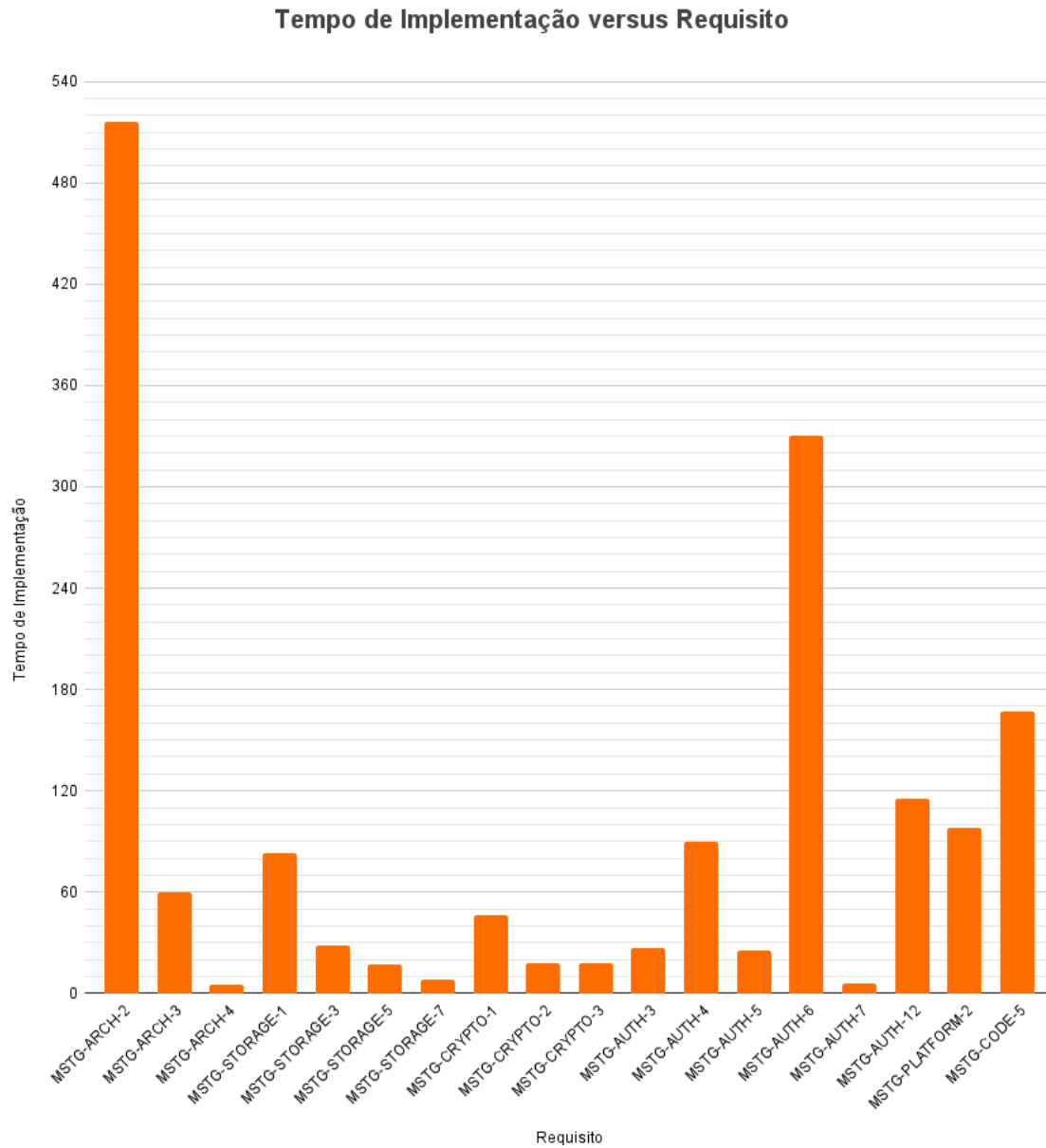


Figura 18 – Comparação de tempo de implementação entre os requisitos. Fonte: Elaborado pelo Autor.

Distribuição do Tempo de Desenvolvimento: Funcionalidades vs. Requisitos de Segurança

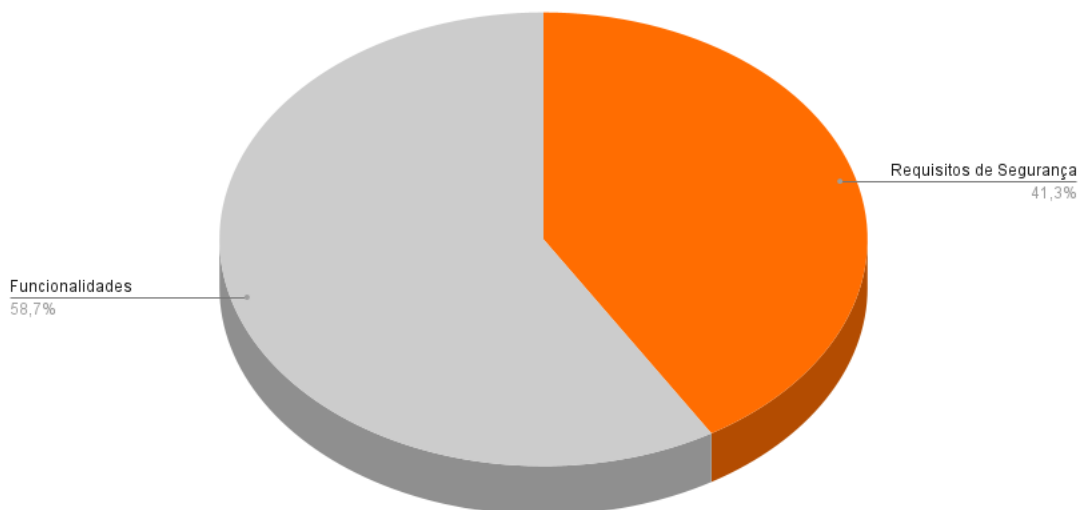


Figura 19 – Comparação de tempo de implementação entre os requisitos. Fonte: Elaborado pelo Autor.

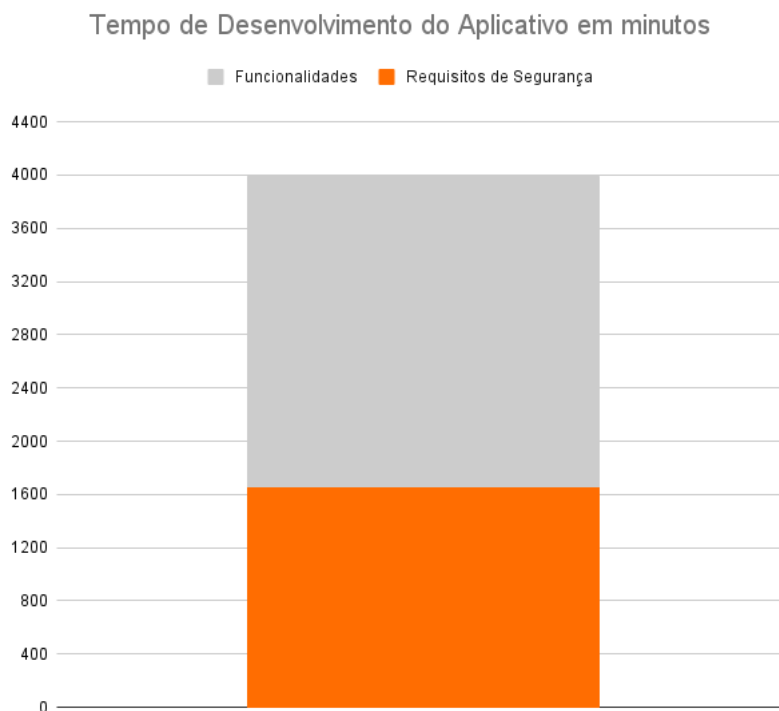


Figura 20 – Comparação de tempo de implementação entre os requisitos. Fonte: Elaborado pelo Autor.

6 Conclusão

Este trabalho teve como objetivo analisar a aplicação dos requisitos de segurança definidos pelo *Mobile Application Security Verification Standard - MASVS*, focando na perspectiva de um desenvolvedor júnior.

Dessa forma, foram apresentadas, no [Capítulo 2](#), conceitos de segurança de software, as principais vulnerabilidades encontradas em uma aplicação mobile, bem como os requisitos de segurança definidos pela [OWASP et al. \(2022\)](#) no documento MASVS. Com base nesse embasamento foram selecionados alguns requisitos de segurança, com o objetivo de analisar o nível de esforço empregado por um desenvolvedor júnior na aplicação desses requisitos.

A análise da implementação do componente de autenticação, detalhada no [Capítulo 5](#), revelou percepções significativas sobre a implementação dos requisitos de segurança.

A análise dos conhecimentos necessários para implementar os requisitos de segurança do OWASP revelou a complexidade envolvida, desde a compreensão das vulnerabilidades em dispositivos móveis até a expertise em criptografia. Essa diversidade de conhecimentos apresenta desafios, especialmente para desenvolvedores juniores, exigindo um esforço adicional no treinamento da equipe.

A aplicação prática desses requisitos demonstrou que, em geral, o esforço necessário para incorporar medidas de segurança não foi desproporcional em relação ao desenvolvimento das funcionalidades principais da aplicação.

Foi observado que a implementação da maioria dos requisitos de segurança demandou menos de duas horas, indicando uma aplicação eficiente dessas práticas, com exceções notáveis nos requisitos ARCH-2 e AUTH-6, que exigiram esforços substanciais de pesquisa e implementação. Além disso, o tempo total dedicado à segurança foi superado pelo dedicado ao desenvolvimento das funcionalidades principais, sugerindo que a integração de práticas de segurança não impõe um ônus significativo no cronograma de desenvolvimento.

Os requisitos de segurança implementados apresentaram uma correlação direta com as principais vulnerabilidades identificadas pela OWASP. Essa correlação reforça a eficácia dos requisitos na prevenção e mitigação de potenciais ameaças à segurança da aplicação.

Esses resultados ressaltam a importância de treinar os desenvolvedores para que conheçam as ameaças a que as aplicações móveis estão sujeitas e considerar a segurança

desde as fases iniciais do desenvolvimento, integrando-a de forma inteligente para garantir a proteção adequada contra ameaças potenciais.

O presente trabalho trouxe diversas contribuições para o campo de segurança de software, visto que através desse estudo foi possível observar que o esforço necessário para tornar uma aplicação segura não é tão grande quanto se imagina. Essa descoberta é um incentivo às equipes de desenvolvimento para passarem a integrar a implementação de práticas de segurança, visando tornar suas aplicações mais robustas e seguras.

Ao refletir sobre o desenvolvimento deste trabalho, destaco a significativa evolução em minha compreensão sobre a importância da segurança de software, adquirida por meio do estudo aprofundado de diversos tópicos relacionados. Além disso, a experiência proporcionou um amplo conhecimento sobre a elaboração de trabalhos acadêmicos, abrangendo nuances, metodologias e estrutura. Reconheço os obstáculos superados, enxergando o valor deste projeto não apenas como uma etapa acadêmica, mas como uma jornada integral que contribuiu de maneira essencial para meu desenvolvimento pessoal e profissional.

6.1 Proposta para trabalhos futuros

Sugere-se alguns direcionamentos para trabalhos futuros que possam estender e aprofundar a pesquisa realizada neste estudo de caso:

- Análise do nível de esforço aplicado nos requisitos de segurança de nível L2 definidos no documento MASVS: realizar um estudo de caso específico para mensurar o esforço necessário na aplicação dos requisitos de nível L2. Aprofundar a investigação sobre o nível de esforço aplicado a esses requisitos pode proporcionar uma compreensão mais precisa da dificuldade envolvida na integração da segurança em aplicações para dispositivos móveis.
- Avaliação dos impactos financeiros na implementação de práticas de segurança de software: realizar uma análise minuciosa dos benefícios financeiros decorrentes da incorporação de práticas de segurança de software em uma aplicação. Esta avaliação pode incluir a redução de custos relacionados ao retrabalho, o fortalecimento da confiança do cliente na aplicação, resultando em maior retenção de clientes, e os impactos positivos nas receitas devido a produtos mais seguros e alinhados às demandas do mercado.
- Análise da correlação entre segurança de software e outras características inerentes à qualidade de software: analisar como a segurança de software se correlaciona com outras características do software como usabilidade, manutenibilidade, portabilidade, entre outros. Essa análise pode se mostrar valiosa para entender a importância de

equilibrar efetivamente a segurança com outras exigências, contribuindo para o desenvolvimento de software mais completo e adaptável às necessidades do usuário e do mercado.

Referências

- BHUTTA, M. N. M. Towards Secure IoT-Based Payments by Extension of Payment Card Industry Data Security Standard (PCI DSS). 2022. Citado na página 31.
- BIONE, T. A. *Mitigação de Riscos de Segurança de Dispositivos Android Baseada na Melhoria das Decisões de Configuração do Usuário*. 2015. Disponível em: <<http://www.bcc.ufrpe.br/br/content/mitiga%C3%A7%C3%A3o-de-riscos-de-seguran%C3%A7a-de-dispositivos-android-baseada-na-melhoria-das-decis%C3%B5es-de>>. Citado na página 21.
- BRERETON, P. et al. Using a Protocol Template for Case Study Planning. 2008. Citado 5 vezes nas páginas 13, 24, 42, 43 e 45.
- CARDOSO, L. d. C. *Frameworks Back End*. [Digite o Local da Editora]: Editora Saraiva, 2021. ISBN 9786589965879. Disponível em: <<https://integrada.minhabiblioteca.com.br/#/books/9786589965879/>>. Citado na página 24.
- CHERAPAU, I. et al. On the Impact of Touch ID on iPhone Passcodes. In: *Eleventh Symposium On Usable Privacy and Security (SOUPS 2015)*. Ottawa: USENIX Association, 2015. p. 257–276. ISBN 978-1-931971-24-9. Disponível em: <<https://www.usenix.org/conference/soups2015/proceedings/presentation/cherapau>>. Citado na página 29.
- COPESES, F.; LANZ, R. *Introdução à Node.js*. 2023. Section: Node.js. Disponível em: <<https://nodejs.dev/pt/learn/>>. Citado na página 54.
- CVE, C. V. a. E. *Common Vulnerabilities and Exposures- CVE*. 2022. Disponível em: <<https://cve.mitre.org/index.html>>. Citado na página 28.
- GASIBA, T. et al. Ranking Secure Coding Guidelines for Software Developer Awareness Training in the Industry. 2020. Citado na página 22.
- GOERTZEL, K. et al. Software Security Assurance: A State-of-Art Report (SAR). jul. 2007. Citado 2 vezes nas páginas 21 e 22.
- GOOGLE, G. *Dart programming language*. 2023. Disponível em: <<https://dart.dev/>>. Citado 2 vezes nas páginas 24 e 54.
- GOOGLE, G. *Flutter - Build apps for any screen*. 2023. Disponível em: <<https://flutter.dev/>>. Citado 2 vezes nas páginas 24 e 54.
- HALKIDIS, S. T. et al. Architectural Risk Analysis of Software Systems Based on Security Patterns. *IEEE Transactions on Dependable and Secure Computing*, v. 5, n. 3, p. 129–142, jul. 2008. ISSN 1941-0018. Conference Name: IEEE Transactions on Dependable and Secure Computing. Citado na página 22.
- HOLLER, C.; HERZIG, K.; ZELLER, A. Fuzzing with code fragments. In: *21st USENIX Security Symposium (USENIX Security 12)*. [S.l.: s.n.], 2012. p. 445–458. Citado na página 36.

- JAJODIA, S. *Computer Viruses and Malware*. 2006. Citado na página 22.
- MAGAUDA, P. Hacking practices and their relevance for consumer studies: The example of the ‘jailbreaking’ of the iPhone. *Consumers, Commodities and Consumption*, v. 12, n. 1, p. 12–11, 2010. Citado na página 35.
- MOHAMMED, N. M. et al. Exploring software security approaches in software development lifecycle: A systematic mapping study. 2017. Citado na página 21.
- MORESI, E. *Metodologia da Pesquisa*. 2003. Citado 3 vezes nas páginas 24, 41 e 42.
- MOZILLA, M. D. N. *JavaScript / MDN*. 2022. Disponível em: <<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>>. Citado na página 54.
- NESTJS. *Documentation / NestJS - A progressive Node.js framework*. 2023. Disponível em: <<https://docs.nestjs.com>>. Citado 4 vezes nas páginas 55, 57, 58 e 67.
- OGU, E. C. et al. A botnets circumspection: The current threat landscape, and what we know so far. *Information*, v. 10, n. 11, p. 337, 2019. Publisher: MDPI. Citado na página 32.
- OWASP. *OWASP Mobile Top 10 / OWASP Foundation*. 2016. Disponível em: <<https://owasp.org/www-project-mobile-top-10/>>. Citado 12 vezes nas páginas 23, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37 e 38.
- OWASP. *OWASP Foundation, a Fundação de Código Aberto para Segurança de Aplicativos / Fundação OWASP*. 2022. Disponível em: <<https://owasp.org/>>. Citado na página 9.
- OWASP, O. et al. *Mobile Application Security Verification Standard*. 2022. Citado 18 vezes nas páginas 13, 23, 38, 39, 40, 42, 49, 50, 72, 83, 93, 94, 95, 96, 97, 98, 99 e 100.
- OWASP, W. *OWASP Top 10*. 2017. Disponível em: <https://owasp.org/www-pdf-archive/OWASP_Top_10-2017_%28en%29.pdf.pdf>. Citado na página 29.
- PUB. *flutter_secure_storage / Flutter Package*. 2023. Disponível em: <https://pub.dev/packages/flutter_secure_storage>. Citado na página 59.
- RANSOME, J.; MISRA, A. *Core Software Security: Security at the Source*. London, UNITED STATES: Auerbach Publishers, Incorporated, 2013. ISBN 978-1-4665-6096-3. Disponível em: <<http://ebookcentral.proquest.com/lib/univbrasilia-ebooks/detail.action?docID=1547083>>. Citado 3 vezes nas páginas 21, 27 e 28.
- RASHID, A. et al. The Cyber Security Body of Knowledge. p. 1067, 2021. Citado 3 vezes nas páginas 21, 27 e 36.
- REDIS. *Redis / A plataforma de dados em tempo real*. 2023. Disponível em: <<https://redis.com/pt/>>. Citado na página 68.
- ROCHA, T. V. d. C. Estudo da aplicação da tecnologia NFC em sistemas de pagamentos eletrônicos sem contato (contactless). 2022. Publisher: Pontifícia Universidade Católica de Goiás. Citado na página 32.

STALLINGS, W. *Cryptography and network security: principles and practice*. 4th ed. ed. Upper Saddle River, N.J: Pearson/Prentice Hall, 2006. OCLC: ocm63126393. ISBN 978-0-13-187316-2. Citado na página 28.

TURNER, A. *How Many People Have Smartphones Worldwide (Dec 2022)*. 2022. Section: Research. Disponível em: <<https://www.bankmycell.com/blog/how-many-phones-are-in-the-world>>. Citado 2 vezes nas páginas 9 e 22.

XING, L. et al. Cracking app isolation on apple: Unauthorized cross-app resource access on MAC os x and ios. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. [S.l.: s.n.], 2015. p. 31–43. Citado na página 29.

ZAMFIROIU, A. CrawVulns - A Software Solution for Vulnerabilities Analysis. *Informatica Economica*, v. 24, n. 1/2020, p. 38–47, mar. 2020. ISSN 14531305, 18428088. Disponível em: <<http://revistaie.ase.ro/content/93/04%20-%20zamfiroiu,%20pocatilu,%20capisizu.pdf>>. Citado na página 28.

Apêndices

APÊNDICE A – Lista de Requisitos de Segurança para Aplicativos Móveis

Tabela 2 – V1 - Requisitos de Arquitetura, *Design* e Modelagem de Ameaças (OWASP et al., 2022)

| ID | Descrição | L1 | L2 |
|---------|---|----|----|
| ARCH-1 | Todos os componentes da aplicação são identificados e considerados como necessários. | X | X |
| ARCH-2 | Os controles de segurança não são aplicados unicamente no lado do cliente, mas em seus terminais remotos. | X | X |
| ARCH-3 | Foi determinada uma arquitetura de alto nível não somente para o aplicativo, mas para todos os seus terminais remotos, onde nessa arquitetura foi abordada a segurança. | X | X |
| ARCH-4 | Todas as informações sigilosas dentro da aplicação são claramente identificadas. | X | X |
| ARCH-5 | Todos os componentes do aplicativo são definidos por suas funcionalidades de negócios e/ou funcionalidades de segurança que eles fornecem | | X |
| ARCH-6 | Um modelo de ameaças foi desenvolvido para o aplicativo móvel e os serviços remotos relacionados, que identifica possíveis ameaças e contramedidas. | | X |
| ARCH-7 | A implementação de todos os controles de segurança é centralizada. | | X |
| ARCH-8 | Existe uma política clara sobre como as chaves criptográficas, caso exista, são gerenciadas e como é imposto o ciclo de vida de cada chave criptográficas. | | X |
| ARCH-9 | Existe uma forma de atualizar o aplicativo móvel. | | X |
| ARCH-10 | A segurança é aplicada em cada parte do ciclo de desenvolvimento de software. | | X |
| ARCH-11 | Uma política responsável de propaganda está em vigor e é seguramente aplicada. | | X |
| ARCH-12 | O aplicativo deve estar de acordo com as leis e regulamentos de privacidade. | X | X |

Tabela 3 – V2 - Requisitos de Armazenamento de Dados e Privacidade (OWASP et al., 2022)

| ID | Descrição | L1 | L2 |
|------------|--|----|----|
| STORAGE-1 | Os recursos de armazenamento das credenciais do sistema devem ser usados para guardar os dados sensíveis, como dados pessoais, credenciais de usuário ou chaves criptográficas. | X | X |
| STORAGE-2 | Os dados sensíveis não devem ser guardados fora do aplicativo ou dos recursos de armazenamento das credenciais do sistema. | X | X |
| STORAGE-3 | Os dados sensíveis não devem ser mostrados nos <i>logs</i> da aplicação. | X | X |
| STORAGE-4 | Os dados sensíveis não podem ser compartilhados com terceiros, a menos que seja algo necessário definido pela arquitetura do sistema. | X | X |
| STORAGE-5 | Para os casos em que as entradas de usuário se tratam de dados sensíveis, o cache deve estar desabilitado. | X | X |
| STORAGE-6 | Dados sensíveis não devem ser mostrados através de mecanismos IPC. | X | X |
| STORAGE-7 | Dados sensíveis, como senhas ou PINs, não podem ser exibidos na interface de usuário. | X | X |
| STORAGE-8 | Dados sensíveis não podem ser incluídos nos <i>backups</i> feitos pelo sistema operacional móvel. | | X |
| STORAGE-9 | Quando o aplicativo ficar em segundo plano, os dados sensíveis precisam ser removidos da visualização. | | X |
| STORAGE-10 | O aplicativo deve manter dados sensíveis na memória mais que o necessário, além disso, depois do uso a memória deve ser completamente apagada. | | X |
| STORAGE-11 | O aplicativo deve enfatizar o uso de políticas mínimas de segurança no acesso ao aplicativo, como pedir que o usuário defina uma senha para acessar o aplicativo. | | X |
| STORAGE-12 | No aplicativo deve possuir instruções destinadas ao usuário sobre os tipos de informação de Identificação Pessoal que são processadas, além de ensinar ao usuário, no uso do aplicativo, seguir as melhores práticas de segurança. | X | X |
| STORAGE-13 | O aplicativo deve sempre buscar dados que são sensíveis de terminais remotos e mantê-los apenas em memória. Nunca deve ficar guardado no dispositivo móvel do usuário. | | X |
| STORAGE-14 | Caso seja necessário guardar dados pessoais de forma local, eles devem ser cifrados usando uma chave proveniente do armazenamento suportado pelo hardware que necessita de autenticação. | | X |
| STORAGE-15 | O armazenamento local do aplicativo deve ser completamente removido, depois de haver várias tentativas de autenticação sem êxito. | | X |

Tabela 4 – V3 - Requisitos de Criptografia (OWASP et al., 2022)

| ID | Descrição | L1 | L2 |
|----------|---|----|----|
| CRYPTO-1 | A aplicação não se fundamenta unicamente no método de criptografia simétrica com chaves fixas no código fonte. | X | X |
| CRYPTO-2 | A aplicação possui de forma comprovada implementações primitivas de criptografia. | X | X |
| CRYPTO-3 | A aplicação utiliza primitivas criptográficas que são adequadas para as regras de negócio em questão, ajustadas com parâmetros que são adeptos às melhores práticas de criptografia da indústria. | X | X |
| CRYPTO-4 | Não é utilizado dentro do aplicativo protocolos criptográficos ou algoritmos amplamente considerados obsoletos para utilização em segurança. | X | X |
| CRYPTO-5 | Não é reutilizado a mesma chave criptográficas para vários fins dentro do aplicativo. | X | X |
| CRYPTO-6 | Ao gerar valores aleatórios, um gerador de números aleatórios suficientemente seguro deve ser utilizado para todos os valores. | X | X |

Tabela 5 – V4 - Requisitos de Autenticação e Gerenciamento de Sessão (OWASP et al., 2022)

| ID | Descrição | L1 | L2 |
|---------|---|----|----|
| AUTH-1 | Caso a aplicação prover acesso a um serviço remoto aos usuários, deve existir alguma forma de autenticação de usuário e senha executada nesse terminal remoto. | X | X |
| AUTH-2 | Caso seja utilizado o gerenciamento de sessão com estado, o terminal remoto deve utilizar identificadores de sessão gerados de forma aleatória no intuito de autenticar as requisições de clientes, sem que as credenciais do usuário sejam enviadas. | X | X |
| AUTH-3 | Caso seja utilizada a autenticação baseada em <i>token</i> sem estado, o servidor deve proporcionar um <i>token</i> que foi assinado usando um algoritmo seguro. | X | X |
| AUTH-4 | Quando o usuário efetua o <i>logout</i> , o <i>endpoint</i> remoto deve encerrar a sessão existente. | X | X |
| AUTH-5 | Existe uma política de senha e é estabelecido no terminal remoto. | X | X |
| AUTH-6 | É implementado no terminal remoto um mecanismo que proteção contra envios exagerados de credenciais. | X | X |
| AUTH-7 | Depois de um período predefinido de inatividade do usuário, as sessões são invalidadas pelo terminal e os <i>tokens</i> de acesso são expirados. | X | X |
| AUTH-8 | Caso haja autenticação biométrica, não deve ser vinculada a eventos, usando APIs que apenas retornam verdadeiro ou falso. Pelo contrário, deve ser baseado no desbloqueio do <i>keychain</i> . | | X |
| AUTH-9 | No terminal remoto há um segundo fator de autenticação e o requisito de 2FA é aplicado de forma coerente. | | X |
| AUTH-10 | Transações sigilosas precisam de autenticação adicional. | | X |
| AUTH-11 | O usuário é informado pelo aplicativo sobre todas as atividades consideradas sigilosas em sua conta. Eles podem ver a lista de dispositivos no qual sua conta no aplicativo está sendo acessada, com informações de IP, endereço, entre outros, podendo bloquear qualquer um deles. | | X |
| AUTH-12 | As formas de autorização devem ser aplicados no terminal remoto utilizado pela aplicação. | X | X |

Tabela 6 – V5 - Requisitos de Comunicação de Rede (OWASP et al., 2022)

| ID | Descrição | L1 | L2 |
|-----------|--|----|----|
| NETWORK-1 | Devem ser criptografados os dados passados pela rede, fazendo uso do TLS. O canal seguro é usado pelo aplicativo de forma concisa. | X | X |
| NETWORK-2 | São utilizadas as melhores práticas atuais na configuração do TLS, ou ao menos estão tão próximas quanto possível, caso o sistema operacional do aparelho móvel não fornecer suporte aos padrões recomendados. | X | X |
| NETWORK-3 | O certificado X.509 do terminal remoto é verificado pelo aplicativo depois que o canal seguro é estabelecido. São considerados apenas certificados assinados por uma CA. | X | X |
| NETWORK-4 | O aplicativo usa seu próprio armazenamento de certificados, ou determina um certificado ou chave pública do terminal e, depois, não estabelece conexões com terminais que forneçam um certificado ou chave distinta, ainda que assinados por uma CA confiável. | | X |
| NETWORK-5 | Para operações críticas, como cadastros, registros, ou recuperação de conta, o aplicativo não confia em apenas uma forma de comunicação insegura como email ou SMS. | | X |
| NETWORK-6 | A aplicação precisa de conectividade além de bibliotecas de segurança atualizadas. | | X |

Tabela 7 – V6 - Requisitos de Interação com a Plataforma (OWASP et al., 2022)

| ID | Descrição | L1 | L2 |
|-------------|---|----|----|
| PLATFORM-1 | São solicitadas apenas um conjunto mínimo de permissões pelo aplicativo. | X | X |
| PLATFORM-2 | São verificadas todas as entradas do usuário ou de fontes externas, incluindo dados recebidos da interface, URLs personalizadas e origens pela rede. | X | X |
| PLATFORM-3 | Não é disponibilizado pelo aplicativo funcionalidades sensíveis através de URL personalizada, a não ser que estejam adequadamente protegidos. | X | X |
| PLATFORM-4 | O aplicativo não disponibiliza funcionalidades sensíveis pelo mecanismo IPC, a não ser que o IPC esteja adequadamente protegido. | X | X |
| PLATFORM-5 | A não ser que seja explicitamente necessário, código JavaScript é desativado em <i>WebViews</i> . | X | X |
| PLATFORM-6 | Só um conjunto mínimo de manipuladores de protocolo necessários estão configurados para <i>WebViews</i> , de preferência, apenas protocolos HTTP. Manipuladores como arquivos que são potencialmente perigosos estão desabilitados. | X | X |
| PLATFORM-7 | Examine se o <i>WebView</i> só renderiza o código JavaScript que está incluído no pacote do aplicativo, caso os métodos nativos da aplicação forem exposto a um <i>WebView</i> . | X | X |
| PLATFORM-8 | Caso haja a desserialização de objetos, devem ser implementadas usando APIs de serialização consideradas seguras. | | X |
| PLATFORM-9 | Em sistemas Android, o aplicativo é capaz de se proteger de ataques de sobreposição de tela. | | X |
| PLATFORM-10 | Antes que o <i>WebView</i> seja destruído, o armazenamento, o cache e os recursos carregados devem ser apagados. | | X |
| PLATFORM-11 | Em sistemas iOS deve ser verificado se a aplicação impossibilita o uso de teclados de terceiros toda vez que dados sigilosos são inseridos. | | X |

Tabela 8 – V7 - Requisitos de Qualidade de Código e Configuração do Compilador (OWASP et al., 2022)

| ID | Descrição | L1 | L2 |
|--------|---|----|----|
| CODE-1 | O aplicativo é assinado e disponível com um certificado apropriado e cuja chave privada está adequadamente protegida. | X | X |
| CODE-2 | O aplicativo foi criado em modo de <i>release</i> e com as configurações adequadas para ela. | X | X |
| CODE-3 | Devem ser apagados os símbolos de depuração dos binários nativos | X | X |
| CODE-4 | Toda parte do código de depuração que auxilia o desenvolvedor deve ser removido. Não deve ser exposto <i>logs</i> detalhado de erros e nenhuma mensagem de depuração. | X | X |
| CODE-5 | Componentes de terceiros usados no aplicativo, como bibliotecas ou <i>frameworks</i> , devem ser identificados e verificados em relação às vulnerabilidades conhecidas. | X | X |
| CODE-6 | São capturadas e tratadas adequadamente as possíveis exceções pelo aplicativo. | X | X |
| CODE-7 | Os controles de segurança não autorizam acesso no tratamento de erros, por padrão. | X | X |
| CODE-8 | A memória é alocada, utilizada e liberada com segurança na parte do código que não é gerenciado. | X | X |
| CODE-9 | As funcionalidades de segurança que são fornecidas gratuitamente pelas ferramentas, devem estar habilitadas. | X | X |

Tabela 9 – V8 - Requisitos de Resiliência (OWASP et al., 2022)

| ID | Descrição | R |
|---------------|--|---|
| RESILIENCE-1 | O aplicativo identifica e reage à presença de um dispositivo com <i>root</i> ou <i>jailbreak</i> alertando o usuário ou fechando o aplicativo. | X |
| RESILIENCE-2 | O aplicativo previne a depuração ou detecta e responde a ela. Devem ser cobertos todos os protocolos de depuração disponíveis. | X |
| RESILIENCE-3 | O aplicativo identifica e responde à modificação de executáveis e dados críticos do próprio aplicativo. | X |
| RESILIENCE-4 | O aplicativo identifica e responde à presença de ferramentas de engenharia reversa e <i>frameworks</i> bastante utilizados. | X |
| RESILIENCE-5 | O aplicativo identifica quando é utilizado em um emulador e responde a isso. | X |
| RESILIENCE-6 | O aplicativo identifica e responde à modificação de código e dados no seu espaço de memória. | X |
| RESILIENCE-7 | O aplicativo implementa vários mecanismos de defesa (8.1 a 8.6). Quanto mais mecanismos de defesa for implementado, maior será a sua capacidade de resistir a ataques de engenharia reversa. | X |
| RESILIENCE-8 | As técnicas de detecção produzem tipos distintos de respostas. | X |
| RESILIENCE-9 | É utilizada a ofuscação para as defesas do programa que, incapacita a desofuscação através de análise dinâmica. | X |
| RESILIENCE-10 | A vinculação ao dispositivo é impedida pelo aplicativo quando é usado um identificador único proveniente de várias propriedades únicas do dispositivo. | X |
| RESILIENCE-11 | Todos os arquivos executáveis e bibliotecas contidos no aplicativo precisam ser cifrados e/ou códigos significativos e segmentos de dados inseridos no executáveis precisam ser cifrados ou empacotados. Dessa forma, uma análise estática simples não irá revelar partes importantes do código e dos dados. | X |
| RESILIENCE-12 | Se o intuito da ofuscação é resguardar códigos relevantes, um mecanismo de ofuscação adequado precisa ser usado para essa atividade particular e resistentes contra métodos de desofuscação manual e automatizada, levando em conta às pesquisas atualmente publicadas. A eficácia do mecanismo precisa ser validada com testes manuais. | X |
| RESILIENCE-13 | Deve ser implementado a criptografia de dados no nível do aplicativo como uma providência adicional contra ataques <i>eavesdropping</i> . | X |