UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

JONAS CAETANO DA SILVA

ANÁLISE DE VULNERABILIDADES EM UM APLICATIVO MÓVEL

Florianópolis
2023

JONAS CAETANO DA SILVA

ANÁLISE DE VULNERABILIDADES EM UM APLICATIVO MÓVEL

Trabalho de conclusão de curso do Curso de Graduação em Ciência da Computação do Departamento de Informática e Estatística da Universidade Federal de Santa Catarina para a obtenção do título de Bacharel em Ciência da Computação.
Orientadora: Profª. Drª. Thaís Bardini Idalino
Coorientador: Gabriel Estevam de Oliveira, Me.

Florianópolis
2023

Silva, Jonas Caetano
ANÁLISE DE VULNERABILIDADES EM UM APLICATIVO MÓVEL /Jonas Caetano Silva ; orientadora, Thaís Bardini Idalino, coorientador, Gabriel Estevam Oliveira, 2023.
48 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Ciências da Computação, Florianópolis, 2023.

Inclui referências.

1. Ciências da Computação. 2. vulnerabilidades de segurança. 3. aplicativos móveis. 4. segurança de softwares. I. Idalino, Thaís Bardini. II. Oliveira, Gabriel Estevam. III. Universidade Federal de Santa Catarina. Graduação em Ciências da Computação. IV. Título.

JONAS CAETANO DA SILVA

ANÁLISE DE VULNERABILIDADES EM UM APLICATIVO MÓVEL

Este trabalho de conclusão de curso foi julgado adequado para obtenção do título de Bacharel em Ciência da Computação e aprovado em sua forma final pelo Curso de Graduação em Ciência da Computação.

Florianópolis, 23 de junho de 2023.

Banca Examinadora:

_____
Profª. Drª. Thaís Bardini Idalino (Orientadora)
Universidade Federal de Santa Catarina - UFSC

_____
Prof. Dr. Jean Everson Martina
Universidade Federal de Santa Catarina - UFSC

_____
Prof. Dr. Martin Augusto Gagliotti Vigil
Universidade Federal de Santa Catarina - UFSC

AGRADECIMENTOS

Chegar ao final de um ciclo como esse não seria viável sem a ajudar daqueles que não mediram esforços para que eu tivesse uma formação de qualidade: os meus pais, Rosa Caetano e Antônio Cirilo. Para vocês, agradeço a paciência e o empenho para que um dia eu chegasse aonde estou.

Estendo esse agradecimento a outros(as) membros de minha família, como meu irmão Rafael e minha irmã Beatriz, minhas tias e meus tios, por sempre estarem ao meu lado ensinando diariamente o verdadeiro sentido de fazer parte de uma família.

Agradeço a minha orientadora, a Profa. Dra. Thaís Bardini e ao meu coorientador Gabriel Estevam por todo apoio e paciência ao longo da difícil jornada de escrita do Trabalho de Conclusão de Curso (TCC). Para mim foi um verdadeiro exercício de superação, que só foi possível pelo suporte fornecido durante esses meses. Aproveito para me desculpar se, por vezes, estive aquém das expectativas geradas.

Ao Coordenador do Curso, Prof. Dr. Jean Martina, agradeço por não ter medido esforços para auxiliar discentes como eu a retornarem para o curso e concluírem sua formação de ensino superior. Obrigado pelo empenho que fez e faz toda diferença para as mudanças estruturais no nosso curso.

Também agradeço a todos os funcionários da Universidade Federal de Santa Catarina (UFSC) que diariamente estão lutando para que possamos ter uma universidade pública e de qualidade, mesmo diante de todos os retrocessos que vivenciamos em relação a pauta educacional do país.

Por fim, agradeço a minha namorada, Josiete Mendes, que não só me convenceu a voltar ao curso, como praticamente me carregou nas costas durante essa jornada. Sem seu apoio eu não estaria aqui, me arrisco a dizer que não conseguiria nem passar pelo processo de entrada, e com certeza teria ficado pelo caminho, mas ela nunca me abandonou. Por tudo isso, mais do que agradecer, dedico a ela esse trabalho.

LISTA DE ILUSTRAÇÕES

QUADROS

FIGURAS

LISTA DE SIGLAS

LGPD	Lei Geral de Proteção de Dados Pessoais
Inmetro	Instituto Nacional de Metrologia, Qualidade e Tecnologia
LabSEC	Laboratório de Segurança em Computação
NIST	National Institute of Standards and Technology
E-CIBER	Estratégia Nacional de Segurança Cibernética
OWASP	Open Worldwide Application Security Project
ENISA	European Union Agency for Network and Information Security
API	Application Programming Interface

SUMÁRIO

ABSTRACT

The aim of the present study was to identify vulnerabilities in a mobile application that seeks to connect the concerns of the government (Inmetro) and consumers who use fuel pumps. To achieve this, both static and dynamic tests were conducted, resulting in the identification of two main critical points regarding the vulnerability of the application in question. Finally, by providing a generic guide that provides direction for secure development, the intention is to disseminate this awareness among developers.

RESUMO

O presente trabalho teve como objetivo a identificação de vulnerabilidades em um aplicativo móvel que busca interligar as preocupações do Governo (Inmetro) e dos consumidores usuários das bombas de combustíveis. Para tanto, foram realizados os testes estáticos e dinâmicos e chegou-se em dois principais pontos críticos no que se refere a vulnerabilidade do aplicativo em questão. Por fim, ao apresentar um guia genérico que orienta sobre o desenvolvimento com foco na segurança, pretende-se difundir essa percepção entre os desenvolvedores.

Palavras-chave: vulnerabilidades de segurança; aplicativos móveis; segurança de softwares

1 INTRODUÇÃO

Depois do *boom* de desenvolvimento de *softwares* e aplicativos móveis, vivencia-se um período em que os critérios de segurança estão em destaque, tanto no que se refere as preocupações do setor empresarial quanto dos governos. Para Bisso *et al.* (2020), devido ao aumento de vazamento de dados sensíveis, que afetam os setores governamentais e empresariais de forma significativa, a segurança da informação virou uma das maiores prioridades de estado em muitos países.

Ainda de acordo com Bisso *et al.* (2020), os principais motivos que conduzem ao vazamento de dados são: (1) credenciais de usuários fracas; (2) falhas no projeto de implementação de aplicativos e sistemas; (3) falhas humanas, como por exemplo a configuração errada de determinados servidores; (4) falta de políticas de controle interno das instituições; e (5) alguns funcionários mal-intencionados (*malicious insiders*).

Quando se analisa a situação dos aplicativos móveis, é possível constatar maiores problemáticas em relação ao assunto, conforme alertam Nascimento e Cintra (2019, p. 1) “o crescimento exponencial do uso de dispositivos móveis para fins pessoais, profissionais e mesmo financeiros traz consigo a preocupação de se garantir que os dados particulares de cada usuário estejam devidamente seguros”. Deste modo, constata-se que tanto usuários consumidores, como governos e empresas podem encontrar-se vulneráveis aos inúmeros problemas ocasionados pela falta de segurança de dados dos sistemas.

Assim, devido a criticidade do cenário, os governos criaram leis com o intuito de definir os direitos de privacidade dos dados dos usuários, bem como penalizar severamente os responsáveis por ataques cibernéticos e vazamento de dados. Como exemplos desse aparato legal, pode-se citar a Lei nº 13.709, de 14 de agosto de 2018, conhecida como a Lei Geral de Proteção de Dados Pessoais (LGPD) (BRASIL, 2018a).

A referida lei preconiza que a disciplina da proteção de dados pessoais tem como fundamentos:

> I - o respeito à privacidade;
> II - a autodeterminação informativa;
> III - a liberdade de expressão, de informação, de comunicação e de opinião;
> IV - a inviolabilidade da intimidade, da honra e da imagem;
> V - o desenvolvimento econômico e tecnológico e a inovação;
> VI - a livre iniciativa, a livre concorrência e a defesa do consumidor; e

> VII - os direitos humanos, o livre desenvolvimento da personalidade, a dignidade e o exercício da cidadania pelas pessoas naturais (BRASIL, 2018a).

Tais princípios são fundamentais para a garantia de civilidade entre os indivíduos que convivem na sociedade. Por isso que o instrumento legal se torna relevante e aliado no combate as impunidades que cercam essa temática da segurança dos dados. Contudo, apenas a existência da legislação não é garantia de que as problemáticas em torno da temática serão resolvidas. Para tanto, o papel de estudos científicos é fundamental no intuito de entender a realidade vivenciada e apresentar soluções que contribuirão para a resolução dos problemas.

Diante desse cenário, o presente trabalho parte da problemática de identificação das possíveis vulnerabilidades de um aplicativo móvel que busca interligar as preocupações do Governo (Inmetro) e dos consumidores usuários das bombas de combustível.

Para contextualizar essa temática, importa destacar que o Instituto Nacional de Metrologia, Qualidade e Tecnologia (Inmetro) aprovou no corrente ano (2023), os primeiros modelos de uma nova geração de bombas de combustíveis. Estas bombas de combustíveis contam com certificação digital, o que as permite utilizar de técnicas criptográficas para realizar assinaturas digitais, tornando-as mais seguras, e assim, dificultando a ocorrência fraudes durante o abastecimento de veículos (Inmetro, 2023). Para Marcelo Morais (Diretor substituto de Metrologia Legal) “Com esse sistema de certificação, o resultado da medição é assinado digitalmente, de tal maneira que assegura que é verdadeira a informação que chega ao medidor, aos olhos do consumidor” (Inmetro, 2023).

Para extrair os dados e assinaturas das bombas de combustíveis, foi desenvolvido um aplicativo como prova de conceito. O aplicativo foi desenvolvido a partir dos esforços do Laboratório de Segurança em Computação (LabSEC) da Universidade Federal de Santa Catarina (UFSC), que é um laboratório que faz parte do Departamento de Informática e de Estatística (INE). O laboratório tem por objetivo estudar, pesquisar, avaliar e implementar soluções na área de segurança em computação, e em particular nos tópicos: Criptografia; Assinatura Digital; Segurança em Sistemas Computacionais; Infraestrutura de Chaves Públicas (ICPs); e Protocolos Criptográficos (LabSEC, 2023).

Posterior ao desenvolvimento do aplicativo, a missão empreendida agora pelo LabSEC é a de realizar testes de segurança para comprovar que realmente o que foi desenvolvido atende aos critérios exigidos pelo Inmetro, entendido aqui como a instituição contratante do projeto. Assim, o presente trabalho soma-se a esses esforços na medida em que analisou o aplicativo e apresentou um guia de desenvolvimento seguro de aplicativos móveis com bases nas vulnerabilidades e técnicas de segurança estudadas. Enquanto produto final, esse guia pode auxiliar futuros desenvolvedores a empreender melhores projetos de desenvolvimento cumprindo os critérios de segurança de dados.

Para tanto, na sequência apresenta-se os objetivos da referida pesquisa. Posteriormente a fundamentação teórica com as especificações dos principais conceitos que orientam o referido trabalho. E, por fim, o desenvolvimento, a análise dos dados e as considerações finais.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Identificar as possíveis vulnerabilidades de um aplicativo móvel que busca interligar as preocupações do Governo (Inmetro) e dos consumidores usuários das bombas de combustíveis.

1.1.2 Objetivos Específicos

- Estudar as vulnerabilidades de segurança mais comuns em aplicativos móveis;
- Estudar técnicas de garantia de segurança de armazenamento e comunicação de dados em dispositivos móveis;
- Avaliar e testar a segurança do aplicativo móvel em busca de vulnerabilidades;
- Propor melhorias de segurança com base no resultado da avaliação;
- Propor um guia de desenvolvimento seguro de aplicativos móveis com bases nas vulnerabilidades e técnicas de segurança estudadas.

2 FUNDAMENTAÇÃO TEÓRICA

A presente fundamentação teórica está estruturada em cinco tópicos. Inicialmente, discute-se a problemática em torno dos ataques, ou seja, os seus

malefícios. Posteriormente, trata-se sobre a importância de um desenvolvimento com foco na segurança. Em continuidade, aborda-se os tipos de análise que podem ser utilizadas para a compreensão da segurança, sendo essas de caráter estático e dinâmico. Por fim, são apresentadas as especificidades da segurança em dispositivos móveis.

2.1 MALEFÍCIOS DOS ATAQUES

Segundo Nascimento e Cintra (2019), o aumento exponencial do uso das tecnologias e dos sistemas informatizados por pessoas e organizações tornou o universo digital mais vulnerável as ameaças físicas e virtuais. Assim, dentre os pressupostos que se pode levantar para o aumento do número de ataques que acontecem em *sites, softwares, aplicativos, dispositivos etc.*, destaca-se que: (1) no geral, mais pessoas passaram a usar *smartphones* por mais tempo; e (2) o mercado de *smartphones* cresceu de maneira que os dispositivos são usados a todo os momentos em diversos contextos (NASCIMENTO; CINTRA, 2019).

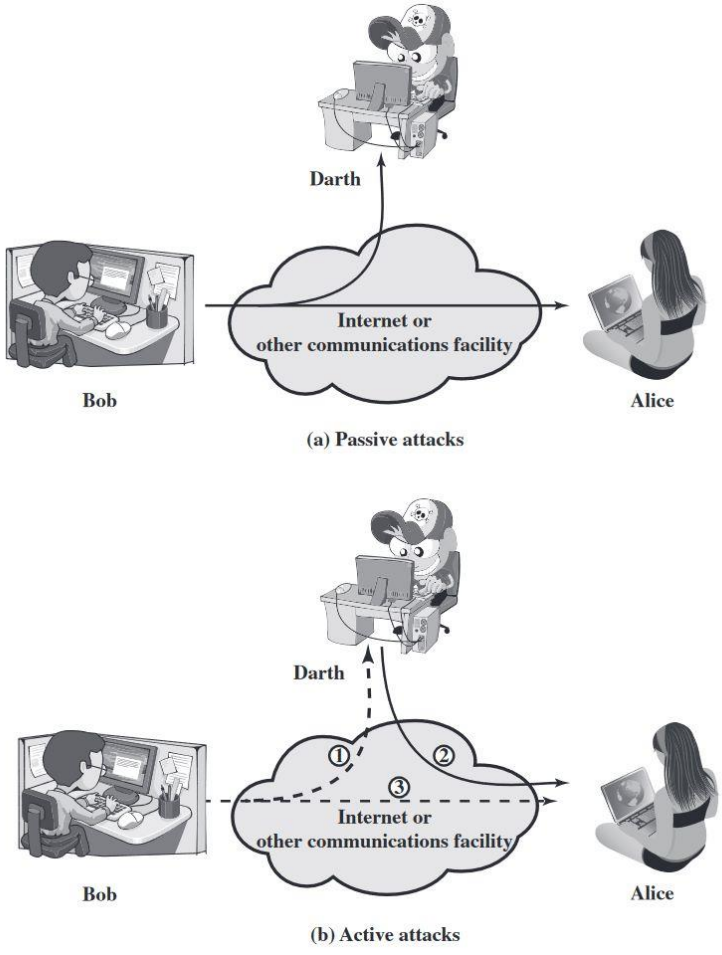
Avanços na área da tecnologia da informação e comunicação moveram serviços que antes oferecidos apenas por meios físicos para um ambiente digital, entre estes, serviços oferecidos por entidades governamentais. No entanto, ameaças também se adaptaram e com isso, o risco à sociedade se tornou foco dos governos. Para tanto, desenvolveu-se na área de segurança de dados, uma tríade que ajuda a conceituar os tipos de ataques, sendo descrita da seguinte forma: Confidencialidade, Integridade e Disponibilidade (NIST, 2004), que são os conceitos mais importantes, mas outros devem ser mencionados, como Autenticidade e Não-Repúdio (STALLINGS, 2014).

No geral, existem dois principais tipos de ataques: passivos e ativos. Os primeiros, ataques passivos, são aqueles que não envolvem a modificação de dados de usuários. Esse tipo de ataque é difícil de ser detectado, pois, para os envolvidos, todo o processo ocorreu normalmente. O principal conceito quebrado é o da confidencialidade (STALLINGS, 2014). Exemplos seriam monitorar ou espionar a transmissão de dados e mensagens. Como forma de proteção a criptografia de mensagens pode ser usada, mas nem sempre isso é suficiente, pois muitas vezes existe valor só de se ter conhecimento que uma mensagem foi transmitida, mesmo que não se descubra seu conteúdo.

Já os ataques ativos são aqueles que de alguma forma manipulam dados dos envolvidos (STALLINGS, 2014). Os danos causados por esse tipo de ataque geralmente são maiores do que nos ataques passivos. Embora sejam mais fáceis de serem detectados, devido a enorme quantidade de modos e meios que ataques ativos podem ser executados, não é possível criar uma defesa definitiva contra eles. Sendo assim, também é necessário estar preparado para se recuperar de eventuais invasões em sistemas e violações de dados.

Como forma de ilustrar e resumir os conceitos apresentados, Stallings (2014) apresenta a figura abaixo (Figura 1).

Figura 1 – Ataques passivo e ativo



Fonte: Stallings (2014, p. 15).

No que se refere as falhas de segurança, podem ser classificadas em três níveis: baixo, moderado e alto (NIST, 2004), conforme detalhado no Quadro 1.

Quadro 1 – Níveis das falhas de segurança

Nível	Impacto na operação	Dano a ativos	Dano financeiro	Dano Humano
Baixo	Afetado, mas continua sendo capaz de executar as funções primárias	Pouco ou nenhum	Pouco ou nenhum	Pouco ou nenhum
Moderado	Afetado de maneira significativa e com impactos na efetividade, mas continua sendo capaz de executar as funções primárias	Significante	Significante	Significante, mas não envolve mortes ou danos permanentes
Alto	Impactado severamente e pode não ser capaz de executar algumas de suas funções primárias	Enorme	Enorme	De grandes proporções tanto em quantidade de indivíduos afetados quanto na severidade, podendo causar morte ou danos permanentes

Fonte: Adaptado de NIST (2004).

A partir dos conceitos apresentados, entende-se que as violações de segurança afetam tanto os usuários como as empresas, seja de modo financeiro ou detrimento de sua reputação. Assim como as empresas, os governos podem sofrer das consequências de uma má gestão da segurança dos dados, desse modo leis foram criadas para mitigar o risco de perdas. No caso do Brasil, temos no artigo 50 da LGPD, onde preconiza-se que:

> Os controladores e operadores, no âmbito de suas competências, pelo tratamento de dados pessoais, individualmente ou por meio de associações, poderão formular regras de boas práticas e de governança que estabeleçam as condições de organização, o regime de funcionamento, os procedimentos, incluindo reclamações e petições de titulares, as normas de segurança, os padrões técnicos, as obrigações específicas para os diversos envolvidos no tratamento, as ações educativas, os mecanismos internos de supervisão e de mitigação de riscos e outros aspectos relacionados ao tratamento de dados pessoais (BRASIL, 2018a).

Muitos países tiveram que criar medidas próprias para conscientizar sobre a necessidade de segurança dentro de suas próprias instituições, no caso do Brasil foi criado o E-CIBER (Estratégia Nacional de Segurança Cibernética), que envolve um conjunto de ações estratégicas do governo federal relacionadas a área de segurança cibernética até 2023 (BRASIL, 2018b). Na Europa, pode-se citar o "*Smartphone Secure Development Guidelines*", que foi criado pela European Union Agency for

Network and Information Security (ENISA) com o intuito de fornecer um guia para desenvolvedores de aplicação móveis para que estes possam criar aplicativos seguros (ENISA, 2016).

2.2 DESENVOLVIMENTO COM FOCO EM SEGURANÇA

De acordo com o artigo 46 da LGPD, “Os agentes de tratamento devem adotar medidas de segurança, técnicas e administrativas aptas a proteger os dados pessoais de acessos não autorizados e de situações acidentais ou ilícitas de destruição, perda, alteração, comunicação ou qualquer forma de tratamento inadequado ou ilícito” (BRASIL, 2018a). Para tanto, a referida lei preconiza no segundo parágrafo do disposto no artigo que “as medidas de que trata o caput deste artigo deverão ser observadas desde a fase de concepção do produto ou do serviço até a sua execução” (BRASIL, 2018a).

Isso representa observar o ciclo de vida do desenvolvimento seguro de software, que tem como objetivo reduzir os riscos associados com o desenvolvimento e é um *framework* que lhe dá os meios para controlar o processo e atingir esse fim. Essa ideia segue o que alerta Oliveira (2019) sobre os procedimentos de desenvolvimento dos softwares com foco na segurança, que por estarem sob a hipótese de constantes ataques, precisam de medidas mais efetivas para resolver os possíveis problemas (firewalls, sistemas de deteção de intrusões e antivírus não são suficientes). Assim, o referido autor complementa:

> É crucial que em cada fase do desenvolvimento de software sejam incluídas análises de segurança, defesas e contramedidas que resultem no lançamento de software mais seguro. Desde o levantamento de requisitos, até ao desenho e implementação, a segurança deve ser incluída no **Ciclo de Desenvolvimento de Software**, de forma a garantir aos utilizadores o melhor e mais seguro software possível (OLIVEIRA, 2019, p. 7, grifo nosso).

Portanto, entender como se produz aplicativos móveis com segurança é fundamental para impedir que falhas aconteçam. Para tanto, faz-se necessário que os processos sejam bem definidos conforme alertou Oliveira (2019), o que implica em constante do código e novas análises a cada ciclo de desenvolvimento do aplicativo. Acontece que nem sempre os programadores estão preparados para pensar nos quesitos de segurança dos aplicativos, devido a diversos fatores, mas principalmente por questões de falta de sensibilidade técnica para o entendimento da relevância

sobre a temática. Talvez, por se tratar de um assunto novo, tendo em vista que a LGPD foi difundida apenas em 2018.

Por fim, sintetizando tais observações sobre o desenvolvimento seguro, Oliveira (2019) apresenta os *touchpoints*, por ordem de eficácia, que são os seguintes:

> • **Revisão de código** – utilizando ferramentas de análise estática;
> • **Análise de risco** – baseada em padrões de ataque e modelos de ameaça;
> • **Testes de intrusão** – utilizando uma abordagem *black-box* que deve ter em consideração a arquitetura do sistema;
> • **Testes de segurança baseados em risco** – com requisitos rastreáveis;
> • **Casos de abuso** – descrevendo o comportamento do sistema sob ataque;
> • **Requisitos de segurança** – os requisitos de segurança devem ser definidos;
> • **Segurança das operações** – monitorizar para falhas de segurança durante o funcionamento dos sistemas (OLIVEIRA, 2019, p. 13-14).

Ao longo do presente trabalho, seja em nível de conceituação ou metodologia, serão apresentados alguns dos conceitos dos *touchpoints* citados por Oliveira (2019).

2.3 GERENCIAMENTO DE VULNERABILIDADES

É possível classificar as vulnerabilidades nos seguintes eixos: potencial de dano, facilidade de repetição do ataque, facilidade de execução do ataque, alcance do ataque em termos de usuários e de facilidade em se descobrir a vulnerabilidade (OWASP, 2022). O risco da vulnerabilidade e o custo-benefício da solução podem ser avaliados por meio dessa classificação.

A orientação é que antes dos testes começarem, se defina quais são os dados sensíveis, pois, detectar vazamentos sem uma definição pode ser impossível (OWASP, 2022). Para tanto, é interessante conversar com a parte interessada nos resultados dos testes para avaliar as “necessidades” e os recursos disponíveis, como tempo e dinheiro. Assim, dispõe-se de determinado conjunto de testes que podem ser utilizados para descobrir possíveis vulnerabilidades, que podem ser classificados como: (1) testes black-box; (2) testes white-box; e (3) testes gray-box (STALLINGS, 2014).

Os testes black-box são feitos sem nenhum conhecimento prévio sobre o funcionamento interno da aplicação. Esse tipo de teste é o que mais se assemelha com o modo que um atacante real atuaria sobre a aplicação. Em segundo lugar, configuram os testes white-box, que são feitos com o auxílio de todos os dados referentes ao aplicativo, como código fonte, documentação, relatórios e diagramas.

Usando esse tipo de teste é possível descobrir vulnerabilidades de forma automática por meio da análise do código fonte. Também torna o processo mais rápido pois se pode criar testes mais específicos e diretos. Esse tipo de teste é comum na fase de desenvolvimento, do qual será adotado na análise apresentada nesta pesquisa. Por fim, os testes gray-box são o meio termo entro os dois mencionados acima, e se caracterizam por compartilhar com o testador apenas poucas informações, como por exemplo uma conta com um nível de acesso maior que a de um usuário comum.

2.4 ANÁLISE ESTÁTICA E DINÂMICA

No que se refere a conceituação, entende-se que a análise estática acontece quando se estuda um programa sem o mesmo estar rodando, ou seja, podem ser analisados seu código fonte, seus componentes, ambiente de execução, sistema operacional e qualquer outra variável que pode afetar o funcionamento do *software* ou sua construção. Tal análise pode ser feita tanto manualmente quanto de forma automática.

Na análise estática manual pode acontecer durante o desenvolvimento, em que se deve dedicar uma fase dentro do ciclo de vida do desenvolvimento de *software* para realização de estudos, revisão de código, verificações dentro do contexto de segurança, no caso de metodologias ágeis. A etapa descrita pode ser realizada a cada pequeno ciclo, e, embora não recomendado, também é possível que se deixe para analisar o *software* quando este já está finalizado e disponível ao público.

Na análise estática automática são utilizadas ferramentas que vasculham o código procurando vulnerabilidades comuns. Embora sejam uma ótima opção para iniciar uma análise, pois possuem a capacidade de analisar uma enorme quantidade de código e suas interações em pouco tempo, essas ferramentas possuem um grande defeito que é não possuir o contexto da aplicação, e por esse motivo, muitos dos resultados encontrados serão falsos positivos e que não são vulnerabilidades no contexto do software (OWASP, 2022). Uma vantagem da análise estática automática em relação à manual é a possibilidade de executar a verificação em executáveis, sejam estes em formato binário ou pacotes prontos para distribuição.

Já a análise dinâmica acontece com o *software* já em execução e tem como principal objetivo encontrar fraquezas nos pontos de contato do *software*, como por exemplo, interação com o usuário através dos meios de entrada, chamadas para o

sistema operacional, chamadas para uma API (Application Programming Interface) web externa e comunicação com outros dispositivos ou *softwares*.

Tanto a análise estática quanto a dinâmica são, de maneira geral, dependentes de ferramentas específicas para a linguagem em que o software foi escrito, o framework usado, o ambiente em que o software irá rodar ou uma união dos anteriores. Por esse motivo é possível que existam poucas opções dependendo das escolhas feitas na fase de projeto do software, desse modo, é essencial que as etapas de verificação de segurança já estejam planejadas deste o início do planejamento.

2.5 SEGURANÇA EM DISPOSITIVOS MÓVEIS

Diferente de outros meios, a segurança em dispositivos móveis é focada principalmente na proteção de dados (OWASP, 2023). Isso acontece pois os próprios sistemas operacionais dos dispositivos se encarregam de o proteger dos tipos de ataques mais comuns que acontecem em computadores, principalmente através de um sistema de isolamento entre apps e por restringir acesso ao hardware apenas por meios de APIs maduras e seguras do próprio sistema. Porém é justamente nos dispositivos móveis que os usuários armazenam seus dados mais importantes, como por exemplo: documentos de identificação digitais, acesso a aplicativos financeiros, acesso a redes sociais, mensagens, contatos, fotos pessoais etc.

Embora o escopo de vulnerabilidades acabe sendo menor, isso não torna o trabalho mais fácil, pois cada aplicativo deve ser implementado de modo que não acabe se tornando o vetor para roubo desses dados. Caso um aplicativo consiga do usuário a permissão para o acesso as suas fotos, por exemplo, nesse momento passa a ser responsabilidade do aplicativo que enquanto as estiver manipulando essas fotos não sejam compartilhadas sem a autorização do usuário.

Além de dados genéricos do usuário, também é necessário que o aplicativo proteja os dados no escopo do próprio aplicativo, como por exemplo seu banco de dados interno, o login do usuário, ou qualquer dado que o aplicativo seja responsável por armazenar ou transmitir.

3 ANÁLISE DO APLICATIVO

Com o intuito de exemplificar o desenvolvimento do presente trabalho, escolheu-se o aplicativo móvel “*App Consumidor Medida Inteligente”*, que é um

software em processo de construção a partir de uma parceria entre LabSEC e o Inmetro.

Foram idealizados dois aplicativos, um destinado a usuários consumidores finais e o outro a usuários fiscais, porém para os fins desse trabalho, serão avaliados os aspectos apenas do aplicativo destinado aos consumidores finais. O objetivo deste aplicativo é fornecer ao consumidor uma ferramenta de fiscalização quanto ao uso de bombas de combustíveis após finalizar o abastecimento do seu veículo (IDALINO *et al.*, 2022).

Na sequência, detalham-se as especificações técnicas do aplicativo.

3.1 REQUISITOS DO SISTEMA

Para atingir o grande público foi decidido oferecer o aplicativo nos sistemas operacionais móveis Android e iOS. Com esse alvo em vista foi escolhido usar o Flutter como framework principal para o desenvolvimento, pois este é capaz de gerar aplicativos para Android e iOS usando a mesma base de código, facilitando o desenvolvimento. No entanto, até o presente momento, apenas a versão Android foi desenvolvida.

Com a finalidade de atingir o objetivo, o aplicativo utiliza a interface Bluetooth para se comunicar com a bomba de combustível e a internet para se comunicar com uma API web para verificar a assinatura dos dados compartilhados pela bomba de combustíveis. Ao verificar um abastecimento, este ficará salvo no aplicativo para futuras referências e caso seja opção do usuário, ele poderá enviar uma reclamação ou denúncia à plataforma *Fala.br*, para isso o mesmo precisa de uma conta *Gov.br*.

O sistema desenvolvido propõe-se a atender os requisitos funcionais levantados pela equipe a partir do problema apresentado pelo Inmetro, a saber: (1) O sistema deve permitir ao consumidor consultar dados referentes a abastecimentos realizados em bombas de combustíveis; (2) O sistema deve permitir ao consumidor reportar reclamações e/ou denúncias referentes a abastecimentos realizados; (3) O sistema deve permitir consultar certificados, estabelecimentos, objetos metrológicos, registros metrológicos e reclamações/denúncias; (4) O aplicativo móvel deve ser capaz de obter dados das bombas de combustíveis através de comunicação bluetooth; e (5) O aplicativo móvel deve ser capaz de verificar as assinaturas digitais feitas pelas bombas de combustíveis (IDALINO *et al.*, 2022).

Com base nessas informações pode-se traçar pontos de partida para começar o desenvolvimento seguro:

- **Linguagem de Programação e Framework**

O framework Flutter usa a linguagem Dart, sabendo disso, é preciso entender quais aspectos relacionados à segurança são inerentes à linguagem e ao framework. Para isso é necessário ter alguém na equipe com conhecimento sólido de ambos, bem como pesquisar quais ataques e fraquezas existem explorando essa “stack”.

- **Ambiente de execução**

Com a decisão de oferecer o aplicativo em Android e iOS, o próximo passo é entender quais são as fraquezas desses sistemas e quais delas são relevantes ao software em análise.

- **Pontos de contato**

De acordo com a breve descrição do funcionamento esperado do aplicativo pode-se destacar os seguintes pontos de contato externo: comunicação bluetooth com bomba de combustíveis; comunicação através de uma API Web com um servidor externo para consulta de dados complementares, verificação da assinatura digital, login do usuário e envio de dados e manifestações; interação do usuário por meio da tela, seja visualmente ou com toques; uso da API do sistema operacional para salvar dados localmente.

Após entender os requisitos do sistema e os aspectos mais relevantes para o desenvolvimento seguro, dos quais foram considerados no momento de elaboração do aplicativo por parte do LabSEC, a seguir apresentam-se as etapas do guia proposto como produto desse trabalho. Destaca-se que os dados apresentados no referido guia são generalizáveis para outros contextos de análise e utilização.

3.2 PESQUISA DE RISCOS E FRAQUEZAS

Para a pesquisa se optou por focar nas fraquezas e ataques tanto do Framework e na linguagem quanto dos ambientes de execução (Android e iOS), apenas no que é relacionado aos pontos de contato descritos acima. Também é importante frisar que todas as medidas que serão citadas adiante envolvem apenas a camada de aplicação, pois esta é a que os desenvolvedores possuem controle e mesmo que a falha esteja em uma camada inferior será na camada de aplicação que a prevenção será executada.

- **Comunicação bluetooth**

Embora o bluetooth não seja um protocolo completamente seguro, existem poucas maneiras que um desenvolvedor pode se proteger de ataques, pois a maiorias das falhas encontradas ao longo do tempo aconteceram em níveis inferiores onde o controle é feito pelo sistema operacional ou mesmo no firmware do chip bluetooth. Para a aplicação estudada existe apenas a necessidade de receber dados, sendo assim a possibilidade de se transmitir dados sensíveis é baixa, tornando o escopo de riscos menor. Porém, ainda assim existem alguns pontos onde necessitam a atenção do desenvolvedor, sendo eles: receber e ler apenas o conteúdo necessário para a tarefa a ser executada; conectar apenas pelo período necessário; usar corretamente os modos de segurança disponíveis e escolher uma biblioteca bluetooth que seja testada e aprovada pela comunidade.

- **Comunicação por meio de uma API Web com um servidor externo para verificação da bomba, para login do usuário e para envio de denúncia**

Uma das maneiras de estabelecer uma conexão segura com um servidor externo é utilizando protocolos que usam criptografia para proteger os dados transmitidos e que validam a identidade do servidor. Para validar uma identidade é preciso haver um aval de algum indivíduo ou grupo e que esses sejam confiáveis. Felizmente todo sistema operacional já possui uma lista de entidades confiáveis (Entidades Certificadoras) que possuem a habilidade de emitir certificados validando a identidade. Porém, a nível de aplicação, é possível ir um passo adiante e escolher uma entidade específica e permitir comunicação externa apenas quando a mesma foi a emissora do certificado de identificação. Esse processo é chamado Identity Pinning, ou, Identidade Fixada em tradução livre. Porém, essa prática só é recomendada quando se possui total controle sobre o servidor externo, pois caso se opte por mudar detalhes do servidor, como por exemplo, seu endereço ou mudar até mesmo a própria entidade certificadora, a aplicação precisará ter sua Identity Pinning atualizada para os novos valores simultaneamente. Caso contrário, corre-se o risco de haver interrupção na disponibilidade de serviços. Para o aplicativo Medida Inteligente, apenas um servidor externo estará sobre o controle dos desenvolvedores, desse modo apenas esse poderá se beneficiar dessa técnica.

- **Interação do usuário por meio da tela, seja visualmente ou com toques**

Existe na literatura demonstrações de ataques que podem, em determinadas situações, simular toques e gestos na tela de um smartphone, porém não foi encontrado nenhum relato de que essa técnica tenha sido usada fora de laboratórios. Além disto, qualquer defesa a ataque similar ficaria fora do escopo do desenvolvedor de uma aplicação pois o mesmo não possui controle sobre como o smartphone detecta entradas de toque.

- **Uso da API do sistema operacional para salvar dados localmente**

"Resumidamente, podemos dizer que dados públicos devem estar disponíveis para todos e dados privados ou sensíveis devem ser protegidos, ou melhor, deixados de fora do armazenamento do dispositivo" (OWASP, 2023). No contexto de desenvolvedores de aplicação, a primeira ação a ser tomada é decidir se existem dados sensíveis que precisam ser salvos no dispositivo, pois é possível armazenar esses dados em um servidor externo e os acessando apenas quando necessário, porém se tornaria necessário estar online para usar corretamente a aplicação. Caso o armazenamento local seja a única opção, deve-se então usar as APIs corretas para dados sensíveis, sempre usando opções que permitam a cifragem dos dados.

Com o material encontrado, finaliza-se a etapa de levantamento teórico e pode-se então partir para o início do desenvolvimento.

3.3 FERRAMENTAS UTILIZADAS

O presente estudo utilizará ferramentas de análise estática e dinâmica para avaliar a segurança do aplicativo. A análise estática será realizada por meio das ferramentas MobSF e Ostorlab, que permitem examinar o aplicativo diretamente pelo pacote de distribuição, APK (Android) e IPA (iOS), em busca de possíveis vulnerabilidades e problemas de segurança conhecidos. Essas ferramentas fornecerão insights valiosos sobre as falhas de autenticação, vazamento de dados e permissões excessivas presentes no aplicativo.

Além disso, será empregada a ferramenta Objection para a análise dinâmica do aplicativo. Essa ferramenta permitirá a execução do aplicativo em um ambiente controlado, possibilitando a identificação de comportamentos maliciosos e exploração de vulnerabilidades em tempo de execução. A análise dinâmica fornecerá informações cruciais sobre o acesso não autorizado a recursos do dispositivo e possíveis explorações de falhas de segurança.

Por meio da combinação dessas ferramentas, será possível realizar uma avaliação abrangente da segurança do aplicativo, identificando vulnerabilidades tanto no código estático quanto no comportamento dinâmico. Os resultados obtidos serão fundamentais para aprimorar a segurança do aplicativo, proteger os dados dos usuários e garantir uma experiência segura e confiável.

3.3.1 MobSF

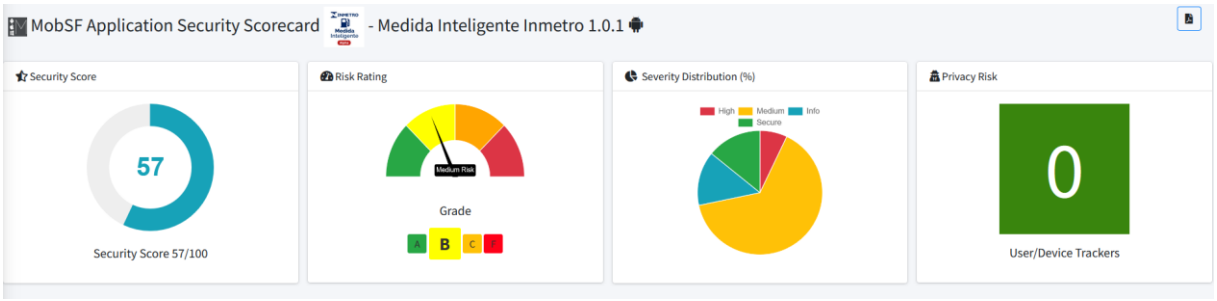
Figura 2 – MobSF Logo

O MobSF (Mobile Security Framework) é uma ferramenta poderosa e de código aberto que se destina a testar e avaliar a segurança de aplicativos móveis. É uma ferramenta que oferece análise estática e dinâmica de forma automática. Para a análise dinâmica é usado um simulador. Possui código aberto e possibilita execução localmente.

Uma das vantagens do MobSF é sua capacidade de gerar relatórios detalhados sobre as vulnerabilidades encontradas, fornecendo informações valiosas para os desenvolvedores corrigirem os problemas de segurança detectados. Esses relatórios incluem uma lista das vulnerabilidades encontradas, suas descrições e possíveis soluções.

No entanto, é importante observar que, no caso do aplicativo exemplo utilizado neste estudo, não é possível realizar a análise dinâmica com o MobSF devido à falta de suporte do simulador para a comunicação bluetooth com outros dispositivos. Apesar dessa limitação específica, o MobSF continua sendo uma ferramenta poderosa para a análise estática de aplicativos móveis, especialmente para aplicativos desenvolvidos usando o Flutter, pois permite a análise direta do arquivo apk, diferentemente de grande parte das outras ferramentas que suportam apenas analisar códigos Java ou Kotlin.

Figura 3 – MobSF Application Scorecard



3.3.2 Ostorlab

Figura 4 – Ostorlab Logo

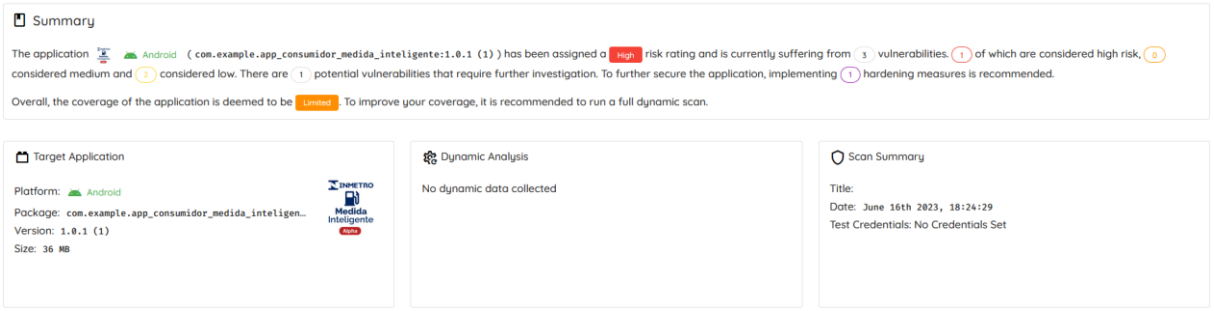
Ostorlab

Muito similar ao MobSF porém possui resultados melhores e com mais detalhes sobre as vulnerabilidades encontradas, outra vantagem é sua interface intuitiva e amigável, que permite que desenvolvedores e especialistas em segurança utilizem a ferramenta de maneira eficiente.

Um recurso valioso do Ostorlab é a sua capacidade de integração com pipelines de desenvolvimento e sistemas de controle de versão, facilitando a incorporação das análises de segurança em todo o ciclo de vida do desenvolvimento do aplicativo. Isso permite que os desenvolvedores identifiquem e corrijam as vulnerabilidades de segurança desde as fases iniciais do desenvolvimento, garantindo a criação de aplicativos móveis mais seguros.

Também aceita APKs como entrada. Diferentemente do MobSF, o Ostorlab é uma ferramenta proprietária.

Figura 5 – Ostorlab sumário das vulnerabilidades



3.3.1 Objection (Frida)

Figura 6 – Objection Logo

Objection é um toolkit de exploração móvel em tempo de execução que tem como motor a ferramenta Frida e será utilizado para a análise dinâmica do aplicativo.

Frida, por sua vez, é outra ferramenta de instrumentação dinâmica que permite injetar código em processos em execução, alterar seu comportamento e inspecionar seus dados. Frida é útil para manipulação e análise dinâmica de aplicações móveis, web e desktop.

Objection usa Frida para se conectar a processos em execução e executar scripts Python ou JavaScript que interagem com os objetos na memória. Objection também tem uma interface de linha de comando que permite executar comandos para explorar e manipular o sistema de arquivos, o keychain, as preferências, os bancos de dados, os certificados, as redes e muito mais.

Outro recurso do Objection é ser capaz de criar uma camada ao redor do APK e permitir que se acesse a API de scripts ou linha de comando sem a necessidade de que se faça jailbreak ou root do dispositivo móvel.

3.4 VULNERABILIDADES ENCONTRADAS

Com o uso das ferramentas mencionadas anteriormente foram encontradas as seguintes vulnerabilidades:

3.4.1 Task Hijacking

Em aplicativos Android *Taks* são um conjunto de *Activities* que por sua vez são telas onde um usuário pode realizar uma ação específica. Diferentes aplicativos Android podem possuir *Activities* residindo em uma mesma *Task*, e isto pode acarretar em uma *Activity* maliciosa sendo usada no lugar de uma *Activity* legítima dentro de uma *Task* de um aplicativo alvo.

Esse tipo de ataque é possível de ser executado ao manipular os seguintes parâmetros da aplicação:

- **Task Affinity** controlado pelo atributo *taskAffinity*;
- **Task Reparenting** controlado pelo atributo *allowTaskReparenting.*

Task Affinity é um atributo definido na tag <activity> no arquivo *AndroidManifest.xml*. Este atributo indica de qual *Task* esta *Activity* fará parte e por padrão todas as *Activities* em um aplicativo possuem a mesma afinidade, sendo esta definida pelo nome do pacote do aplicativo.

Task Hijacking pode ser usada para executar ataques de phishing e de negação de serviço e existem variações do ataque que o tornam muito difícil de ser detectado.

As defesas que podem ser aplicadas para mitigação desse ataque dependem da variação que está sendo aplicada, mas em geral as principais recomendações são:

- Definir o *Task Affinity* como uma string vazia, desse modo forçando a aplicação a gerar e usar um valor aleatório;
- Definir o modo de lançamento como *singleInstance*.

3.3.2 Uso de componente desatualizado e vulnerável

A aplicação usa um componente desatualizado e com vulnerabilidades publicamente conhecidas. Este tipo de vulnerabilidade é particularmente grave pois devido ao conhecimento do público geral podem existir ferramentas prontas construídas com o objetivo de explorar as falhas conhecidas do componente, deste modo tornando muito mais simples a realização de um ataque.

O componente afetado no aplicativo é a biblioteca *openssl* que pode ser descrita como uma biblioteca de criptografia de proósito geral e também como um kit de ferramentas para auxiliar no uso dos protocolos TSL e SSL. A versão da biblioteca

atualmente em uso é a **1.1.1g** e as vulnerabilidades conhecidas estão catalogadas no NIST sob as seguintes entradas: CVE-2020-1971, CVE-2018-16395, CVE-2021-23840, CVE-2021-3449 e CVE-2016-7798.

Dentre os danos que o uso dessa versão pode acarretar o mais significativo é que dentro de certas circunstâncias é possível que certos certificados inválidos sejam aceitos como válidos.

Para solucionar essa vulnerabilidade bastar que se atualize a biblioteca para a versão mais recente disponível.

3.4.3 Atributo hasFragileUserData não foi definido

O atributo *hasFragileUserData* em uma aplicação Android define se um aplicativo contém dados sensíveis que precisam ser protegidos. Dados sensíveis podem ser definidos como quaisquer dados onde caso sejam roubados ou perdidos causariam dano para o usuário. Exemplos seriam número de documentos, de cartões de crédito, fotos pessoais etc.

Caso este atributo não esteja definido como *true* na definição da aplicação, quando um usuário desinstalar a aplicação corre-se o risco de dados sensíveis não serem excluídos no processo.

Para solucionar essa vulnerabilidade bastar introduzir o atributo e então quando ocorrer a desinstalação do aplicativo o usuário será questionado se desejo excluir os dados da aplicação.

3.4.4 Modo de backup habilitado

A função de backup permite a usuários Android a opção de salvar dados e configurações de aplicações externamente para que possam ser restaurados no futuro. Porém caso a aplicação possua dados sensíveis do usuário não é aconselhável que essas informações sejam salvas em múltiplos locais.

Para solucionar essa vulnerabilidade é necessário definir o atributo *allowBackup* como *true*.

4 GUIA DE DESENVOLVIMENTO SEGURO

A seguir define-se o guia prático proposto com foco no desenvolvimento de aplicação seguras. Destaca-se que este trabalho poderá ser utilizado para entender outras aplicações e conduzir um processo de desenvolvimento seguro.

4.1 SETUP DO AMBIENTE DE DESENVOLVIMENTO

O primeiro passo é escolher um sistema para controle de versão do software. Dentre as suas funcionalidades, um sistema de controle de versão salva todas as mudanças feitas por qualquer um dos colaboradores e desse modo permite que as versões geradas no decorrer do projeto sejam recuperadas a qualquer momento. Tal controle é de suma importância para que, durante uma investigação sobre existência de uma falha de segurança, por exemplo, seja possível descobrir quando a mesma foi introduzida e os motivos que levaram à modificação.

Entre as opções de sistemas de controle de versão, a recomendada por esse guia será o Git, pois esta é o atual padrão da indústria e possui um amplo suporte de outras ferramentas que hospedam os repositórios e proveem mais funcionalidades que não fazem parte do protocolo Git. Entre essas funcionalidades adicionais, estão ferramentas de integração, entrega e implantação contínua e interfaces simplificadas para revisão de código. Muitas das etapas só são possíveis caso a ferramenta que hospeda o repositório Git, permita que se crie listas de comandos que serão executados usando o código integrado como entrada. Esses comandos podem ser definidos manualmente ou através da ajuda de serviços externos. Como recomendação de hospedeiro remotos de repositórios Git indica-se GitHub e GitLab.

A metodologia contínua permite que durante o desenvolvimento de software se coloque testes, analisadores e barreiras que vão dar maior confiança que cada melhoria e recurso adicionada ao produto final esteja funcionando corretamente e sem falhas de segurança. Dito isto, essa garantia dada pelos testes só é tão boa quanto os próprios testes, ou seja, é preciso criá-los com cuidado e conhecimento para que estes não deixem falhas passar.

Terminada esta etapa de escolha, pode-se começar então a definir um fluxo de trabalho que irá delinear os principais branches que o repositório terá, bem como criar processos para definir como novos branches secundários serão criados e como esses irão interagir com os principais.

O fluxo precisa permitir com que múltiplos desenvolvedores possam trabalhar independentemente e que quando a tarefa for finalizada seja possível conciliar o código de modo que qualquer conflito possa ser analisado e resolvido.

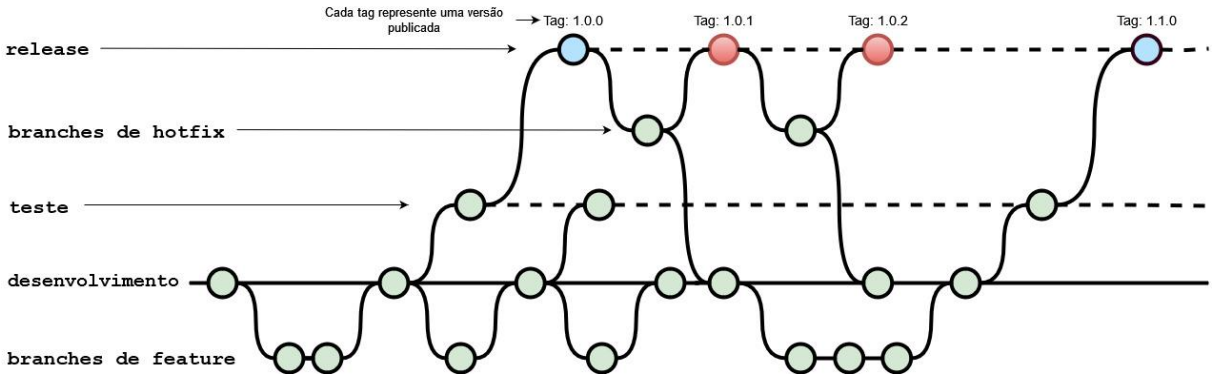
Também é preciso que haja uma separação, para fins de organização e gerência, de qual versão do código está atualmente sendo oferecida para os usuários, qual versão está apta a ser promovida para a próxima versão a ser oferecida e qual versão possui as últimas mudanças adicionadas pelos desenvolvedores. Essa separação é necessária para que se possa agir de maneira mais rápida no caso de alguma falha, pois existe clara distinção de qual versão de código a falha ocorreu.

Com base nesses critérios foi decidido adotar uma versão mais simplificada do git-flow (CWIKLINSKI, 2020), onde iremos usar um branch de desenvolvimento, branches de feature, branches de hotfix, um branch de teste e um branch de release.

Resumidamente, o fluxo funciona da seguinte maneira:

- **Release:** Branch onde o código se encontra na versão onde foi publicado para o público pela última vez. Recebe integração do branch de teste e de branches de hotfixes.
- **Teste:** Branch com as últimas features e bugfixes adicionados e que já passaram por todos os testes e revisões automáticas e manuais. Recebe integração apenas do branch de desenvolvimento.
- **Desenvolvimento:** Branch com as últimas features e bugfixes adicionados e que já passaram por todos os testes e revisões automáticas. Recebe integração dos branches de features, bugfixes e hotfixes.
- **Feature:** Branches criados a partir do branch de desenvolvimento e são onde acontecem as principais modificações do software. São integrados apenas ao branch de desenvolvimento.
- **Hotfix:** Branches criados a partir do branch de release e são criados apenas quando uma falha grave é descoberta e é preciso resolução imediata. Após o término da tarefa, os testes automáticos e manuais são realizados e quando aprovado ele é integrado ao branch de release e uma nova versão do software é publicada. A seguir o branch também é integrado ao branch de desenvolvimento.

Figura 7 – Git workflow baseado em git-flow



Fonte: elaboração própria (2023)

Definido o fluxo, segue-se para a definição de regras para a integração de branches.

Muitas das etapas enunciadas a seguir não são estritamente relacionadas à segurança e têm o propósito de garantir um código com menos chance de conter bugs. Porém como mencionam Graff e Van Wyk (2003, p. 18), “um código com bugs inofensivos não é realmente seguro. É mais como uma pessoa que possui uma doença fatal sobre controle.”. Desse modo, ao tomar precauções para evitar o surgimento de bugs, se está indiretamente deixando o aplicativo mais seguro.

A seguir mostra-se os check-lists a serem cumpridos, posteriormente será detalhado o que cada item consiste e como foram implementadas no aplicativo alvo desse estudo.

Etapas presente em todas as integrações: Verificação automática da formatação do código, análise estática automática, build automático, testes automáticos, testes dinâmicos automáticos e análise dinâmica automática.

- **Branches de feature para o branch de desenvolvimento**: Revisão manual do código e análise dinâmica manual.
- **Branch de desenvolvimento para o branch de teste**: Testes manuais de funcionalidade.
- **Branch de teste para o branch de release**: Aprovação de desenvolvedor sênior.
- **Branches de hotfix para o branch de release**: Revisão manual do código, análise dinâmica manual, testes manuais de funcionalidade e aprovação de desenvolvedor sênior.

- **Branches de hotfix para o branch de desenvolvimento**: Revisão manual do código, análise dinâmica manual e testes manuais de funcionalidade.

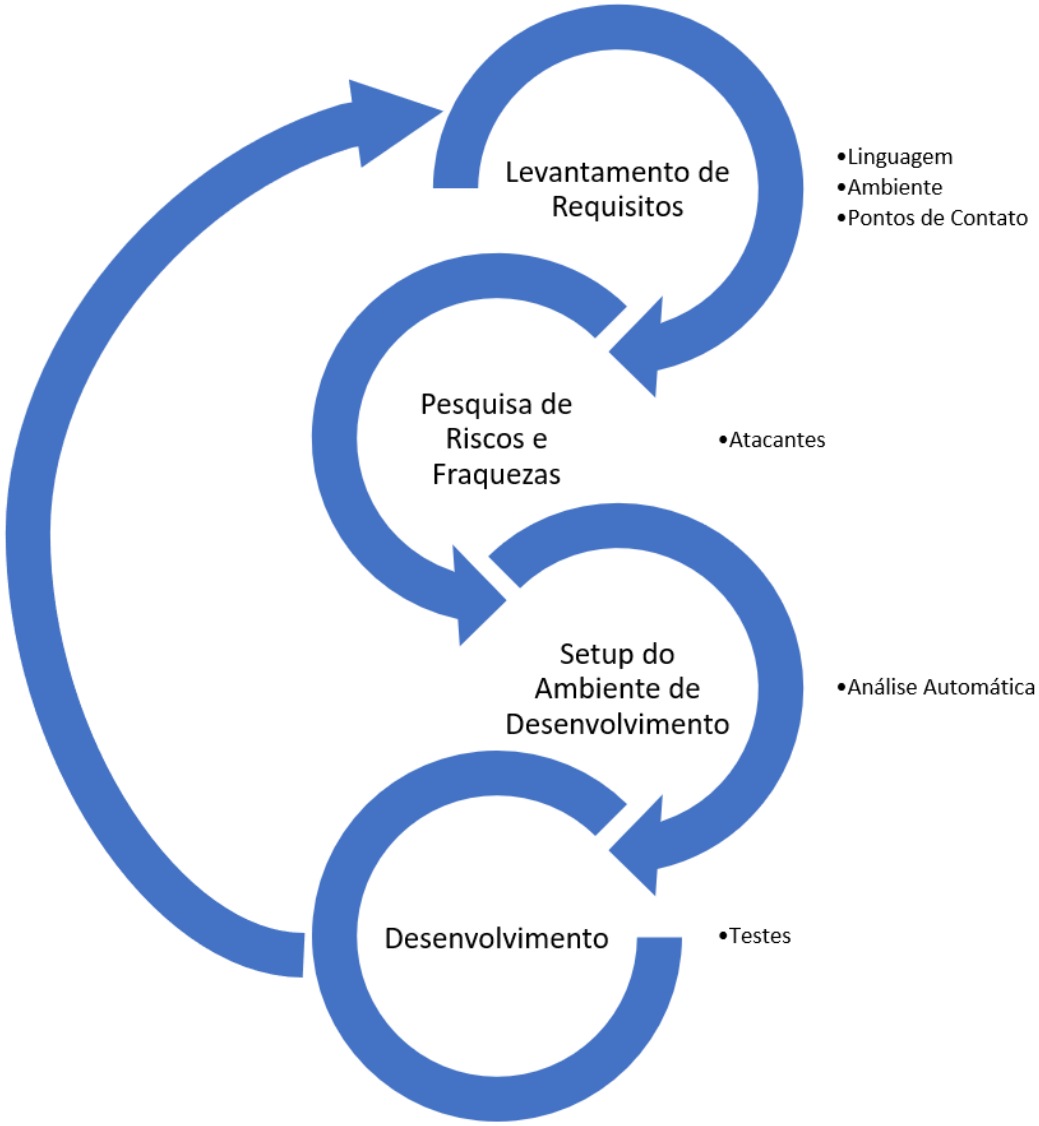
Detalhamento das etapas dos check-lists executadas na integração dos branches:

- **Verificação automática da formatação do código**: Com o intuito de obter um código mais homogêneo um formatador automático é aplicado, isso permite que os desenvolvedores entendam o código escrito pelos colegas com mais facilidade e assim possam contribuir com mais velocidade.
- **Análise estática automática**: Esta etapa pode ser dividida em várias outras sub-etapas que são executadas independentemente. Aqui serão usadas ferramentas que analisam o código reportando, por exemplo, erros específicos da linguagem e framework usados e que só seriam notados na execução da aplicação; más-práticas que podem resultar em bugs ou falhas de segurança; verificação de vazamento de segredos; necessidade de atualização de dependências; entre outros.
- **Build automático**: Gera uma versão do aplicativo.
- **Testes automáticos**: Roda testes usando partes do código da aplicação. Estes testes podem se aplicar, por exemplo, desde pequenas funções independentes até a interação entre módulos. Cada mudança introduzida deve ser acompanhada de testes, deste modo, a cobertura total de testes deve se manter estável a cada versão.
- **Testes dinâmicos automáticos**: São testes que precisam do aplicativo em execução para serem rodados, geralmente funcionam simulando interações que são passiveis de serem executadas por usuários e analisam se o efeito gerado foi o esperado.
- **Análise dinâmica automática**: Assim como os testes dinâmicos também é executada com a aplicação em funcionamento, porém a análise dinâmica irá focar na busca por falhas no que se condiz ao funcionamento interno, como por exemplo, verificar se segredos não estão sendo guardados em partes da memória acessível a outros programas; como a aplicação reage a entradas e comandos inesperados; se existe vazamento de memória; entre outros.

- **Revisão manual do código**: Um desenvolvedor, diferente do que criou as mudanças propostas, analisa a nova versão do código e verifica:
    - Se o objetivo das mudanças foi alcançado;
    - Se as mudanças introduzidas não possuem efeitos colaterais;
    - Se não existe uma maneira melhor de atingir o objetivo;
    - Se o novo código continua sendo seguro;

  Todas as verificações devem levar em conta o código da aplicação como um todo, e não ser analisadas em isolamento.
- **Análise dinâmica manual**: Assim como na revisão manual, um desenvolvedor irá usar seu conhecimento da aplicação para procurar falhas que possam ocorrer enquanto o aplicativo estiver em execução, no contexto das mudanças introduzidas. Nessa etapa podem ser usadas ferramentas que permitem a quem está analisando observar a interação do aplicativo com o ambiente de execução e os outros pontos de contato.
- **Testes manuais de funcionalidade**: São testes executados por uma pessoa em áreas do aplicativo onde os testes dinâmicos automáticos não conseguiram abranger por algum motivo.
- **Aprovação de desenvolvedor sênior**: Análise final de um desenvolvedor sênior garantindo que a versão foi gerada seguindo corretamente os passos propostos.

Figura 8 – Fluxograma do Guia proposto na pesquisa



Fonte: elaboração própria (2023)

A intenção do fluxograma (Figura 8) é de resumir tanto os aspectos teóricos quanto metodológicos que foram apresentados ao longo do trabalho. Apesar de simples, percebeu-se ao durante a pesquisa que a falta de um instrumento mais palpável e objetivo para os desenvolvedores, os leva a cometer falhas no processo de desenvolvimento dos aplicativos móveis.

5 APLICAÇÃO DO GUIA NO CONTEXTO DO APLICATIVO

A partir de agora entraremos em mais detalhes de como essas etapas poderiam ser implantadas no aplicativo em discussão.

Primeiramente, a ferramenta escolhida para hospedar o repositório Git foi a GitLab e esta escolha se deu por conta de oferecer as funcionalidades citadas anteriormente no guia e por ser possível rodar o sistema em um servidor próprio.

- **Verificação automática da formatação do código**

A formatação automática de código pode ser integrada diretamente no ambiente de desenvolvimento e para a linguagem Dart também é possível realizar o mesmo através do comando *dart analyze*. Para que a formatação ocorra é necessário que se defina um conjunto de regras. Para o projeto foi decidido usar o conjunto de regras oficial recomendado para projetos Flutter.

Definida as regras segue-se então para a verificação. Esta pode ser feita em diferentes pontos e para o projeto ficou definido que a verificação ocorreria na integração de branches, mas cada desenvolvedor poderia adicionar também ao seu ambiente de desenvolvimento.

- **Análise estática automática**

Durante esta etapa uma das metas era encontrar uma ferramenta que fosse capaz de analisar o código e encontrar, durante o desenvolvimento, padrões e certos usos de bibliotecas que são conhecidamente inseguros. Infelizmente para a linguagem Dart parece não haver ainda uma ferramenta para essa finalidade.

Optou-se então pelo uso de ferramentas que analisam o objeto gerado a partir do build da versão atual do código. Esse tipo de ferramenta consegue verificar o uso de bibliotecas, se estas estão atualizadas, se são vulneráveis ou possuem dependências comprometidas; verifica o uso de APIs do sistema operacional e notifica em caso da existência de mal uso das mesmas; expõe as permissões requeridas pelo aplicativo e analisa se são necessárias; verifica se existem segredos em texto claro; também procura por configurações que são conhecidamente portas para possíveis ataques.

Dentre as ferramentas analisadas decidiu-se usar o MobSF, por esta oferecer melhor suporte à integração com o GitLab, porém também irão ser usadas outras ferramentas, como: quark-engine, AndroShield, AndroBugs e Ostorlab, porém essas

ferramentas serão executadas de forma manual e somente quando o resultado obtido pelo MobSF for insuficiente.

- **Build automático**

O build de uma aplicação Flutter é simples e feito através do comando *flutter build apk*.

- **Testes automáticos**

Para criar os testes de unidade foi usada uma biblioteca oficial do Flutter desenvolvida para este fim. Esta biblioteca também é capaz de analisar quanto do código está coberto por testes.

- **Testes dinâmicos automáticos**

Este tipo de teste também usou uma biblioteca Flutter oficial. Com ela pode-se acessar uma tela específica do aplicativo e simular interações. Através do uso de respostas e entradas prontas é possível testar partes especificas do aplicativo sem que seja necessário passar por todo o processo que um usuário comum passaria.

- **Análise dinâmica automática**

Embora existam muitas ferramentas disponíveis para realizar análise dinâmica, não foi possível fazer uso delas com o aplicativo pois este necessita realizar uma conexão bluetooth com uma bomba de combustíveis para um funcionamento complete.

- **Revisão manual do código**

A revisão manual pode ser feita tanto pela interface do GitLab quanto do ambiente de desenvolvimento, ficando a critério do revisor. A vantagem que o GitLab oferece é de possuir um ambiente onde o revisor pode sugerir mudanças e adicionar questionamentos.

- **Análise dinâmica manual**

Para a análise dinâmica manual será usado a ferramenta Objection que é construído usando outra ferramenta, Frida, como base. Essa ferramenta adiciona uma camada extra no pacote do app que permite que possamos acessar diretamente a memória do aplicativo e rodar comandos através de uma API. O uso dessa ferramenta se tornou necessário pois os simuladores atuais não suportam a comunicação bluetooth entre dois dispositivos distintos.

- **Testes manuais de funcionalidade**

Estes testes são executados após o build automático gerar o pacote de instalação do aplicativo. Em seguida o aplicativo é instalado em um dispositivo real e se confirma se o objetivo das mudanças no código foi atingido. Também é essencial que todas as partes relacionadas continuem funcionando como antes.

Para testar a interação com bombas de combustíveis usou-se um simulador em hardware para criar uma conexão bluetooth com o aplicativo.

- **Aprovação de desenvolvedor sênior**

A integração do branch de teste ao branch de release significa o lançamento de uma nova versão. Para garantir a qualidade dessa versão, foi criada uma trava que só é liberada com a autorização do desenvolvedor sênior, que, após confirmar que todas as etapas manuais foram seguidas corretamente, pode permitir a integração do branch.

6 CONSIDERAÇÕES FINAIS

O referido trabalho teve como objetivo geral a identificação das possíveis vulnerabilidades de um aplicativo móvel que busca interligar as preocupações do Governo (Inmetro) e dos consumidores usuários das bombas de combustível. Importa destacar que dentre as possíveis vulnerabilidades do aplicativo em questão, encontrou-se o fato de que os desenvolvedores utilizaram (e estão utilizando) uma biblioteca com dependência desatualizada e vulnerabilidade conhecida. Além disso, foi possível constatar uma forte vulnerabilidade em relação a Task Hijacking. Entende-se que tais falhas não devem ser analisadas com preocupações, pois são totalmente passíveis de resolução.

No que se refere aos objetivos específicos, cumpriu-se o proposto de estudar as vulnerabilidades de segurança mais comuns em aplicativos móveis, bem como estudar técnicas para garantir de segurança de armazenamento e comunicação dos dados, e por fim, apresentar a proposta de um guia de desenvolvimento seguro de aplicativos móveis com bases nas vulnerabilidades e técnicas de segurança estudadas.

As propostas de melhorias na segurança foram apresentadas com base no resultado da avaliação e dos testes feitos, constando-se que, no geral, o aplicativo é seguro e segue a tradição das construções de produtos realizadas pelo LabSEC. Inclusive, tais delineamentos foram fundamentais para que como produto desse

trabalho pudesse propor um guia generalizável para outras situações semelhantes, sobre como desenvolver com segurança.

Por fim, compreende-se que a temática de segurança de dados precisa constar em todas as etapas de desenvolvimento dos softwares, e, portanto, deve ser introduzida com maior atenção por parte dos programadores, mas principalmente dos líderes dos projetos de desenvolvimento. Espera-se que a LGPD também sirva como instrumento capaz de introduzir mudanças significativas nos operadores de ação das equipes.

REFERÊNCIAS

BISSO, R.; KREUTZ, D.; RODRIGUES, G.; GIULIANO, P. Vazamentos de Dados: Histórico, Impacto Socioeconômico e as Novas Leis de Proteção de Dados. **Revista Eletrônica Argentina-Brasil de Tecnologias da Informação e da Comunicação**, v. 3, n. 1, 2020.

BRASIL. **Lei Geral de Proteção de Dados Pessoais (LGPD)** | Lei nº 13.709, de 14 de agosto de 2018, [2018a]. Disponível em: https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm. Acesso em: 03 jun. 2023.

BRASIL. **Política Nacional de Segurança da Informação** | Decreto nº 9.637, de 26 de dezembro de 2018, [2018b]. Disponível em: http://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/decreto/D9637.htm. Acesso em: 03 jun. 2023.

CWIKLINSKI, J. **git-flow Documentation**. 2020. Disponível em: https://buildmedia.readthedocs.org/media/pdf/git-flow/latest/git-flow.pdf. Acesso em: 03 jun. 2023.

ENISA - European Union Agency for Network and Information Security. **Smartphone Secure Development Guidelines**. 2016. Disponível em: https://www.enisa.europa.eu/publications/smartphone-secure-development-guidelines-2016. Acesso em: 03 jun. 2023.

GRAFF, M. G.; VAN WYK, K. R. **Secure Coding**: Principles & Practices. Sebastopol; O'Relly & Associates, Inc., 2003.

IDALINO, T. B.; VIGIL, M.; OLIVEIRA, G. E. de.; ABEL, P. de O.; NA, E. G.; GOMES, C. E. V.; HARDT, V. H.; TOMASZEWSK, M.; LOLI, G. B. **Estudo de Usabilidade de Assinaturas Digitais em Objetos Metrológicos**. Relatório Técnico, LabSEC, 2022. Disponível em: https://repositorio.ufsc.br/bitstream/handle/123456789/241804/rt-04-2022.pdf?sequence=1. Acesso em: 06 jun. 2023.

Inmetro - Instituto Nacional de Metrologia, Qualidade e Tecnologia**. Inmetro aprova primeira bomba de combustível com assinatura eletrônica**. 2023. Disponível em: https://www.gov.br/inmetro/pt-br/centrais-de-conteudo/noticias/inmetro-aprova-primeira-bomba-de-combustivel-com-assinatura-eletronica. Acesso em 03 jun. 2023.

LabSEC - Laboratório de Segurança em Computação. **Sobre o LabSEC.** Disponível em: https://labsec.ufsc.br/. Acesso em 03 jun. 2023.

NASCIMENTO, F. R. do; CINTRA, F. G. Vulnerabilidades de segurança em dispositivos Android: análises e estatísticas (2009-2019). **Revista EduFatec: educação, tecnologia e gestão**, v. 2, n. 1, p. 1-25, 2019.

NIST - National Institute of Standards and Technology. **Standards for Security Categorization of Federal Information and Information Systems**, Fips 199, 2004.

Disponível em: https://csrc.nist.gov/publications/detail/fips/199/final. Acesso em 03 jun. 2023.

OLIVEIRA, M. M. M. P. da C. e. **Privacidade no Ciclo de Vida do Desenvolvimento Seguro**. Orientador: José Manuel da Silva Cecílio. 2019. 104p. Dissertação (Engenharia Informática). Universidade de Lisboa, Lisboa, 2019.

OWASP - Open Worldwide Application Security Project. **Mobile Application Security Testing Guide Version v1.5.0**. 2022. Disponível em: https://mas.owasp.org/MASTG/. Acesso em 03 jun. 2023.

OWASP - Open Worldwide Application Security Project. **Mobile Application Security Verification Standard. V2.0.0**. 2023. Disponível em: https://mas.owasp.org/MASVS/. Acesso em 03 jun. 2023.

STALLINGS, W. **Cryptography and Network Security:** Principles and Practice (6th Edition). Londres: Pearson, 2014.

ANEXO(S) E APÊNDICE(S):

Análise de vulnerabilidades em um aplicativo móvel

Jonas C. da Silva[1]

[1]Instituto de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Campus Universitário Reitor João David Ferreira Lima Trindade – Florianópolis – SC

jonas.caetano@inf.ufsc.br

Abstract. *The aim of the present study was to identify vulnerabilities in a mobile application that seeks to connect the concerns of the government (Inmetro) and consumers who use fuel pumps. To achieve this, both static and dynamic tests were conducted, resulting in the identification of two main critical points regarding the vulnerability of the application in question. Finally, by providing a generic guide that provides direction for secure development, the intention is to disseminate this awareness among developers.*

Resumo. *O presente trabalho teve como objetivo a identificação de vulnerabilidades em um aplicativo móvel que busca interligar as preocupações do Governo (Inmetro) e dos consumidores usuários das bombas de combustíveis. Para tanto, foram realizados os testes estáticos e dinâmicos e chegou-se em dois principais pontos críticos no que se refere a vulnerabilidade do aplicativo em questão. Por fim, ao apresentar um guia genérico que orienta sobre o desenvolvimento com foco na segurança, pretende-se difundir essa percepção entre os desenvolvedores.*

1. Introdução

O boom do desenvolvimento de softwares e aplicativos móveis destaca a importância da segurança da informação para empresas e governos. O aumento de vazamento de dados sensíveis tornou a segurança uma prioridade em muitos países [Bisso et al., 2020]. As principais causas de vazamento de dados são: credenciais de usuários fracas, falhas de implementação em aplicativos e sistemas, falhas humanas, falta de políticas de controle interno e funcionários mal-intencionados [Bisso et al., 2020].

Os aplicativos móveis enfrentam desafios adicionais em relação à segurança dos dados dos usuários, o que requer medidas de proteção adequadas [Nascimento e Cintra, 2019]. Governos têm implementado leis para definir os direitos de privacidade dos usuários e penalizar os responsáveis por ataques cibernéticos e vazamento de dados, como a Lei Geral de Proteção de Dados Pessoais (LGPD) no Brasil [Brasil, 2018]. No entanto, a existência de legislação não garante a resolução dos problemas. Estudos científicos são fundamentais para entender a realidade e apresentar soluções

Neste contexto, este trabalho identifica as vulnerabilidades de um aplicativo móvel que conecta as preocupações do Governo (Inmetro) e dos consumidores de bombas de combustível. As novas bombas de combustível aprovadas pelo Inmetro utilizam certificação digital e assinaturas digitais para aumentar a segurança e prevenir fraudes [Inmetro, 2023]. Um aplicativo de prova de conceito foi desenvolvido para extrair dados e assinaturas das bombas de combustível. O aplicativo foi desenvolvido pelo Laboratório

de Segurança em Computação (LabSEC) da Universidade Federal de Santa Catarina (UFSC), que se dedica ao estudo e implementação de soluções de segurança em computação [LabSEC, 2023].

2. Análise do aplicativo

Foram idealizados dois aplicativos, um destinado a usuários consumidores finais e o outro a usuários fiscais, porém para os fins desse trabalho, serão avaliados os aspectos apenas do aplicativo destinado aos consumidores finais. O objetivo deste aplicativo é fornecer ao consumidor uma ferramenta de fiscalização quanto ao uso de bombas de combustíveis após finalizar o abastecimento do seu veículo [IDALINO et al., 2022].

2.1. Requisitos do sistema

Para alcançar um amplo público, optou-se por disponibilizar o aplicativo nos sistemas operacionais móveis Android e iOS. O framework Flutter foi escolhido para o desenvolvimento, pois permite gerar aplicativos para Android e iOS usando a mesma base de código, simplificando o processo de desenvolvimento. No entanto, até o momento, apenas a versão Android foi desenvolvida.

O aplicativo desenvolvido atende aos seguintes requisitos funcionais: (1) permitir ao consumidor consultar dados de abastecimentos; (2) permitir ao consumidor reportar reclamações e/ou denúncias; (3) permitir consultar certificados, estabelecimentos, objetos metrológicos, registros metrológicos e reclamações/denúncias; (4) obter dados das bombas de combustíveis por meio de comunicação Bluetooth; e (5) verificar assinaturas digitais feitas pelas bombas [Idalino et al., 2022].

2.2. Pesquisa de riscos e fraquezas

Na pesquisa, focamos nas fraquezas e ataques relacionados ao framework, linguagem e ambientes de execução (Android e iOS), apenas nos pontos de contato mencionados. É importante ressaltar que as medidas a seguir se aplicam à camada de aplicação, onde os desenvolvedores têm controle, mesmo que a falha ocorra em camadas inferiores.

- **Comunicação bluetooth**

Existem poucas maneiras que um desenvolvedor pode se proteger de ataques, pois a maiorias das falhas encontradas ao longo do tempo aconteceram em níveis inferiores onde o controle é feito pelo sistema operacional ou mesmo no firmware do chip bluetooth. Para a aplicação em estudo, que apenas recebe dados, a possibilidade de vazamento de dados sensíveis é baixa. Pontos de atenção para o desenvolvedor incluem receber e ler apenas o conteúdo necessário, conectar pelo período necessário, usar modos de segurança adequados e escolher uma biblioteca Bluetooth testada e aprovada e aprovada pela comunidade.

- **Comunicação por meio de uma API Web com um servidor externo para verificação da bomba, para login do usuário e para envio de denúncia**

É possível estabelecer uma conexão segura utilizando protocolos criptografados e validação da identidade do servidor. Os sistemas operacionais possuem entidades certificadoras confiáveis para emissão de certificados. A aplicação pode optar por "Identity Pinning", permitindo apenas comunicação com uma entidade certificadora

específica. Essa prática requer controle total sobre o servidor externo para evitar interrupções de serviço.

- **Interação do usuário por meio da tela, seja visualmente ou com toques**

Existe na literatura demonstrações de ataques que podem, em determinadas situações, simular toques e gestos na tela de um smartphone, porém não foi encontrado nenhum relato de que essa técnica tenha sido usada fora de laboratórios. A defesa contra esse tipo de ataque está fora do escopo do desenvolvedor, pois não há controle sobre a detecção de toques pelo smartphone.

- **Uso da API do sistema operacional para salvar dados localmente**

"Resumidamente, podemos dizer que dados públicos devem estar disponíveis para todos e dados privados ou sensíveis devem ser protegidos, ou melhor, deixados de fora do armazenamento do dispositivo" [OWASP, 2023]. No contexto de desenvolvedores de aplicação, a primeira ação a ser tomada é decidir se existem dados sensíveis que precisam ser salvos no dispositivo, pois é possível armazenar esses dados em um servidor externo e os acessando apenas quando necessário, porém se tornaria necessário estar online para usar corretamente a aplicação. Caso o armazenamento local seja a única opção, deve-se então usar as APIs corretas para dados sensíveis, sempre usando opções que permitam a cifragem dos dados.

3. Ferramentas utilizadas

Este estudo utilizará ferramentas de análise estática e dinâmica para avaliar a segurança do aplicativo. A análise estática será feita com as ferramentas MobSF e Ostorlab, examinando o APK (Android) em busca de vulnerabilidades e problemas de segurança conhecidos. Essas ferramentas fornecerão insights sobre falhas de autenticação, vazamento de dados e permissões excessivas.

Para a análise dinâmica, será usada a ferramenta Objection, permitindo a execução do aplicativo em um ambiente controlado. Isso possibilitará a identificação de comportamentos maliciosos e exploração de vulnerabilidades em tempo real. Essa análise revelará informações cruciais sobre o acesso não autorizado a recursos do dispositivo e possíveis explorações de falhas de segurança.

Com a combinação dessas ferramentas, será possível realizar uma avaliação abrangente da segurança do aplicativo, identificando vulnerabilidades tanto no código estático quanto no comportamento dinâmico. Os resultados obtidos serão fundamentais para aprimorar a segurança do aplicativo, proteger os dados dos usuários e garantir uma experiência segura.

3.1. MobSF (Mobile Security Framework)

O MobSF é uma ferramenta de código aberto que testa e avalia a segurança de aplicativos móveis. Ele oferece análise estática e dinâmica automatizada. Para a análise dinâmica, utiliza-se um simulador. O MobSF gera relatórios detalhados das vulnerabilidades encontradas, fornecendo informações valiosas para correções de segurança. Esses relatórios incluem listagem, descrição e soluções das vulnerabilidades. No entanto, a análise dinâmica não é suportada no aplicativo exemplo deste estudo devido à falta de suporte do simulador para a comunicação bluetooth. Apesar dessa limitação, o MobSF é uma poderosa ferramenta de análise estática para aplicativos móveis, especialmente para

aqueles desenvolvidos com Flutter, permitindo a análise direta do arquivo apk, ao contrário de muitas outras ferramentas que suportam apenas Java ou Kotlin.

3.1. Ostorlab

O Ostorlab é uma ferramenta semelhante ao MobSF, porém com resultados mais detalhados sobre as vulnerabilidades encontradas. Sua interface intuitiva e amigável permite que desenvolvedores e especialistas em segurança utilizem a ferramenta de forma eficiente. Além disso, o Ostorlab oferece integração com pipelines de desenvolvimento e sistemas de controle de versão, facilitando a incorporação das análises de segurança ao longo do ciclo de vida do aplicativo. Isso possibilita aos desenvolvedores identificar e corrigir as vulnerabilidades desde as fases iniciais do desenvolvimento, garantindo aplicativos móveis mais seguros. Assim como o MobSF, o Ostorlab também aceita APKs como entrada, mas é importante mencionar que o Ostorlab é uma ferramenta proprietária.

3.1. Objection

Objection é um toolkit de exploração móvel em tempo de execução que tem como motor a ferramenta Frida e será utilizado para a análise dinâmica do aplicativo.

Frida, por sua vez, é outra ferramenta de instrumentação dinâmica que permite injetar código em processos em execução, alterar seu comportamento e inspecionar seus dados. Frida é útil para manipulação e análise dinâmica de aplicações móveis, web e desktop.

Objection usa Frida para se conectar a processos em execução e executar scripts Python ou JavaScript que interagem com os objetos na memória. Objection também tem uma interface de linha de comando que permite executar comandos para explorar e manipular o sistema de arquivos, o keychain, as preferências, os bancos de dados, os certificados, as redes e muito mais.

Outro recurso do Objection é ser capaz de criar uma camada ao redor do APK e permitir que se acesse a API de scripts ou linha de comando sem a necessidade de que se faça *jailbreak* ou *root* do dispositivo móvel.

5. Vulnerabilidades encontradas

Com o uso das ferramentas mencionadas anteriormente foram encontradas as seguintes vulnerabilidades:

5.1. Task Hijacking

Em aplicativos Android *Taks* são um conjunto de *Activities* que por sua vez são telas onde um usuário pode realizar uma ação específica. É possível que diferentes aplicativos possuam *Activities* em uma mesma Task, o que pode permitir que uma *Activity* maliciosa substitua uma *Activity* legítima em um aplicativo alvo.

Esse tipo de ataque manipula parâmetros da aplicação, como o taskAffinity, que define a afinidade da *Task* na tag <activity> do AndroidManifest.xml. Por padrão, todas as *Activities* de um aplicativo possuem a mesma afinidade, baseada no nome do pacote.

O Task Hijacking pode ser usado para ataques de *phishing* e negação de serviço, sendo difícil de ser detectado devido a variações do ataque.

As defesas para mitigar esse ataque dependem da variação utilizada, mas recomendações comuns incluem definir o Task Affinity como uma string vazia para gerar um valor aleatório e definir o modo de lançamento como *singleInstance*.

5.2. Uso de componente desatualizado e vulnerável

A aplicação usa um componente desatualizado e com vulnerabilidades publicamente conhecidas. Esse tipo de vulnerabilidade é grave, pois podem existir ferramentas prontas para explorar as falhas conhecidas, tornando os ataques mais simples.

O componente afetado é a biblioteca *openssl*, uma biblioteca de criptografia e kit de ferramentas para protocolos TSL e SSL. A versão em uso é a 1.1.1g e as vulnerabilidades conhecidas são catalogadas como CVE-2020-1971, CVE-2018-16395, CVE-2021-23840, CVE-2021-3449 e CVE-2016-7798.

O dano mais significativo dessa versão é a possibilidade de aceitar certificados inválidos em certas circunstâncias.

Para corrigir essa vulnerabilidade, é necessário atualizar a biblioteca para a versão mais recente.

5.3. Atributo hasFragileUserData não foi definido

O atributo hasFragileUserData em uma aplicação Android define se um aplicativo contém dados sensíveis que precisam ser protegidos. Dados sensíveis podem ser definidos como quaisquer dados onde caso sejam roubados ou perdidos causariam dano para o usuário. Exemplos seriam número de documentos, de cartões de crédito, fotos pessoais etc.

Caso este atributo não esteja definido como true na definição da aplicação, quando um usuário desinstalar a aplicação corre-se o risco de dados sensíveis não serem excluídos no processo.

Para solucionar essa vulnerabilidade bastar introduzir o atributo e então quando ocorrer a desinstalação do aplicativo o usuário será questionado se desejo excluir os dados da aplicação.

5.4. Modo de backup habilitado

A função de backup permite a usuários Android a opção de salvar dados e configurações de aplicações externamente para que possam ser restaurados no futuro. Porém caso a aplicação possua dados sensíveis do usuário não é aconselhável que essas informações sejam salvas em múltiplos locais.

Para solucionar essa vulnerabilidade é necessário definir o atributo allowBackup como true.

6. Considerações finais

O referido trabalho teve como objetivo geral a identificação das possíveis vulnerabilidades de um aplicativo móvel que busca interligar as preocupações do Governo (Inmetro) e dos consumidores usuários das bombas de combustível. Importa

destacar que dentre as possíveis vulnerabilidades do aplicativo em questão, encontrou-se o fato de que os desenvolvedores utilizaram (e estão utilizando) uma biblioteca com dependência desatualizada e vulnerabilidade conhecida. Além disso, foi possível constatar uma forte vulnerabilidade em relação a Task Hijacking. Entende-se que tais falhas não devem ser analisadas com preocupações, pois são totalmente passíveis de resolução.

As propostas de melhorias na segurança foram apresentadas com base no resultado da avaliação e dos testes feitos, constando-se que, no geral, o aplicativo é seguro e segue a tradição das construções de produtos realizadas pelo LabSEC. Inclusive, tais delineamentos foram fundamentais para que como produto desse trabalho pudesse propor um guia generalizável para outras situações semelhantes, sobre como desenvolver com segurança.

Por fim, compreende-se que a temática de segurança de dados precisa constar em todas as etapas de desenvolvimento dos softwares, e, portanto, deve ser introduzida com maior atenção por parte dos programadores, mas principalmente dos líderes dos projetos de desenvolvimento. Espera-se que a LGPD também sirva como instrumento capaz de introduzir mudanças significativas nos operadores de ação das equipes.

References

Bisso, R., Kreutz, D., Rodrigues, G., and Paz, G. (2020). Vazamentos de Dados: Histórico, Impacto Socioeconômico e as Novas Leis de Proteção de Dados. *Revista Eletrônica Argentina-Brasil de Tecnologias da Informação e da Comunicação*, *3*(1).

Brasil. (2018). Lei Geral de Proteção de Dados Pessoais (LGPD) | Lei nº 13.709, de 14 de agosto de 2018**.** Recuperado de: https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm**.**

Idalino, T. B., Vigil, M., Oliveira, G. E., Abel, P. O., Na, E. G., Gomes, C. E. V., Hardt, V. H., Tomaszewsk, M., and Loli, G. B. (2022). Estudo de Usabilidade de Assinaturas Digitais em Objetos Metrológicos. *Relatório Técnico, LabSEC*, Recuperado de: https://repositorio.ufsc.br/bitstream/handle/123456789/241804/rt-04-2022.pdf?sequence=1.

Inmetro - Instituto Nacional de Metrologia, Qualidade e Tecnologia. (2023) Inmetro aprova primeira bomba de combustível com assinatura eletrônica. Recuperado de: https://www.gov.br/inmetro/pt-br/centrais-de-conteudo/noticias/inmetro-aprova-primeira-bomba-de-combustivel-com-assinatura-eletronica**.**

LabSEC **-** Laboratório de Segurança em Computação**.** Sobre o LabSEC. (2023). Recuperado de: https://labsec.ufsc.br/.

Nascimento, F. R., and Cintra, F. G. (2019). Vulnerabilidades de segurança em dispositivos Android: análises e estatísticas (2009-2019). *Revista EduFatec: educação, tecnologia e gestão*, *2*(1), p. 1-25.

OWASP - Open Worldwide Application Security Project. (2023). Mobile Application Security Verification Standard. V2.0.0. Recuperado de: https://mas.owasp.org/MASVS/.