Uma Abordagem para Testes de Segurança em Aplicações *Android*

Leonardo O. Silva[1], Emanuel B. Rodrigues[1], Ismayle de S. Santos[2]

[1]Departamento de Computação – Universidade Federal do Ceará (UFC)
Caixa Postal 60020-181 – Fortaleza, CE – Brasil

[2]Centro de Ciências e Tecnologia
Universidade Estadual do Ceará (UECE) – Fortaleza, CE – Brasil

leonardosilva_99@alu.ufc.br, emanuel@dc.ufc.br, ismayle.santos@uece.br

Abstract. *Like any other software, applications on the Android operating system can present compromising vulnerabilities, such as those listed by the OWASP Mobile Top 10. Even with the efforts of security professionals, it is still difficult to find standardized documentation of the techniques used so that software testers can reproduce them. them. For this reason, it is necessary to apply a security testing methodology that has a formalization, and ISO 829 was chosen for this purpose in this article, along with OWASP verification guides: MASVS and MASTG. The tests especially address the static and dynamic contexts to check for these vulnerabilities. With the analysis of each vulnerability, it was possible to notice that a good part of them were caused by bad programming practices, which can be corrected following recommendations of an OWASP guide specialized in good practices of code development.*

Resumo. *Como qualquer outro software, aplicações no sistema operacional Android podem apresentar vulnerabilidades comprometedoras, como as elencadas pelo OWASP Mobile Top 10. Mesmo com os esforços de profissionais de segurança, ainda é difícil encontrar documentações padronizadas das técnicas utilizadas para que testadores de software possam reproduzi-las. Por esse motivo, se faz necessária a aplicação de uma metodologia de testes de segurança que tenha uma formalização, e a ISO 829 foi escolhida para isso neste artigo, junto com guias de verificação OWASP: MASVS e MASTG. Os testes abordam especialmente os contextos estático e dinâmico, para averiguar a existência dessas vulnerabilidades. Com a análise de cada vulnerabilidade, foi possível notar que boa parte delas eram causadas por más práticas de programação, as quais podem ser corrigidas seguindo recomendações de um guia da OWASP especializado em boas práticas de desenvolvimento de código.*

1. Introdução

O *Android* é uma plataforma de código aberto baseado em *Linux* que serve como um sistema operacional móvel desenvolvido por um consórcio liderado pela *Google*. As aplicações para *Android* podem ser executadas em diferentes dispositivos tais como *smartphones*, *tablets* e *smart* TVs. Segundo o site oficial da plataforma *Android*, mais de 2 milhões de aplicativos estão disponíveis na *Google Play Store*, loja oficial de aplicativos *Android* da *Google*. Esses aplicativos podem ser voltados para diversas finalidades,

como entretenimento, comunicação, educação e finanças. O número de aplicações e a diversidade de domínios leva a crer que vulnerabilidades nas aplicações podem afetar pessoas no mundo todo.

Nesse contexto, os testes de segurança de aplicações móveis durante o ciclo de desenvolvimento estão ganhando força, principalmente nas fases de desenvolvimento e de testes, em que são utilizadas abordagens estáticas e dinâmicas para analisar o código ou a aplicação em execução, respectivamente, com o intuito de identificar vulnerabilidades.

As vulnerabilidades de *software*, como vazamento de dados confidenciais e utilização de algoritmos criptográficos obsoletos, acarretam em problemas para os desenvolvedores e usuários. Dentre os esforços da área de segurança, pode-se observar os projetos da *Open Source Foundation for Application Security* (OWASP[1]). OWASP é um projeto de código aberto que visa estudar e prevenir as diversas vulnerabilidades possíveis em *software*, disponibilizando metodologia, documentação e tecnologia apropriadas para cumprir o seu objetivo.

Um dos vários projetos que a OWASP gestiona é o OWASP *Mobile Top* 10[2], que lista e define 10 principais vulnerabilidades em aplicações móveis relatadas por empresas e profissionais de segurança. Essa lista é interessante para mostrar que as aplicações podem estar sujeitas a diversos problemas se não houver a devida atenção por parte dos responsáveis pelo desenvolvimento. Porém, ainda é necessária uma melhor especificação de como testes de segurança podem ser utilizados para averiguar se a aplicação é vulnerável a alguma vulnerabilidade da lista ou não. Além disso, as diretrizes existentes não seguem uma padronização de especificação de testes comum na área de testes de *software*, sendo elas descritas com uma linguagem mais voltada para profissionais de segurança e que não é tão comum para testadores, o que pode acabar dificultando o entendimento destes.

Por isso, o foco deste trabalho é apresentar a especificação de testes de segurança em um formato comum para profissionais da área de teste de *software* e de segurança. Seguindo 2 guias metodológicos, *Mobile Application Security Verification Standard* (MASVS[3]) e *Mobile Application Security Testing Guide* (MASTG[4]), ambos desenvolvidos pela OWASP, este trabalho apresenta uma metodologia didática para a condução de testes de segurança nas aplicações *Android*. Esses guias trazem requisitos de segurança que podem ser elencados em uma aplicação e os procedimentos de testes que verificam se os requisitos estão sendo seguidos. Além disso, os testes serão apresentados com a fomalização da *International Organization for Standardization* (ISO) 829, que é comumente utilizada por testadores. Vale ressaltar que este trabalho é uma extensão de [Silva 2022].

Na Seção 2 estão descritos alguns trabalhos realizados com foco semelhante ao deste artigo. Na Seção 3 é detalhada a metodologia proposta por este trabalho. Na Seção 4 são descritos os procedimentos de testes utilizados neste trabalho, os quais são relacionados com a ISO 829 e com o guia MASVS. Na Seção 5 são exibidos a execução e

[1] `https://owasp.org/`
[2] `https://owasp.org/www-project-mobile-top-10/`
[3] `https://mas.owasp.org/MASVS/`
[4] `https://mas.owasp.org/MASTG/`

os resultados dos procedimentos de testes para averiguar as vulnerabilidades do OWASP *Mobile Top* 10 nas aplicações escolhidas. Finalmente, a Seção 6 apresenta a conclusão do trabalho frente aos resultados dos procedimentos realizados e são indicados possíveis trabalho futuro.

2. Trabalhos relacionados

Na literatura, alguns artigos abordam a temática dos testes de segurança envolvendo as vulnerabilidades listadas no OWASP *Mobile Top* 10, cada um com suas peculiaridades e focos de trabalho. Como dito na introdução, este artigo parte de [Silva 2022][5], o qual abordou a metodologia OWASP para testes de seguranças e apresentou as análises de aplicações móveis para averiguar a existência das vulnerabilidades do OWASP *Mobile Top* 10. O foco dele parte mais para um tutorial de como utilizar os guias da OWASP para verificações de segurança no contexto móvel.

[Alanda et al. 2020] analisam algumas aplicações e tentam identificar vulnerabilidades do OWASP *Mobile Top* 10 nelas, mas sem entrar em muitos detalhes dos procedimentos realizados. Eles não fazem menção aos guias MASVS[6] e MASTG[7], da OWASP, o que se diferencia deste trabalho.

[Priambodo et al. 2022] focam em utilizar as ferramentas automatizadas MARA[8] e *Mobile Security Framework* (MobSF[9]) para analisar 2 aplicações do setor de saúde e verificar vulnerabilidades do OWASP *Mobile Top* 10 e Common Weakness Enumeration (CWE) encontradas. Contudo, eles não fazem menção aos guias MASVS e MASTG, não detalha a execução dos testes e nem os reports. Os autores apenas citam as vulnerabilidades e CWE's encontradas em tabelas. Isso também acontece em [Aljabri et al. 2022], que utilizam algumas ferramentas, testam 4 aplicações para identificar vulnerabilidades do OWASP *Mobile Top* 10 e CWE, além de identificar as estruturas da plataforma *Android* afetadas.

[Kohli and Mohaghegh 2020] focam na exploração aplicações Web e utilizam diversas ferramentas automatizadas para verificar a existência de vulnerabilidades do OWASP *Web Top* 10 2021. Porém, os autores não detalham a execução dos testes e nem os reports; além de não fazerem menção a guias da OWASP para verificação *Web*, como o *Web Security Testing Guide* (WSTG[10]).

Frente aos trabalhos citados anteriormente, o principal diferencial deste artigo é a inclusão de uma abordagem que integra as áreas de testes de software e segurança de aplicações, ou seja, tenta formalizar as verificações de segurança para um formato comum. Além disso, destaca as etapas realizadas durante os testes e não se concentra somente nos resultados.

[5] https://repositorio.ufc.br/handle/riufc/69782
[6] https://mas.owasp.org/MASVS/
[7] https://mas.owasp.org/MASTG/
[8] https://github.com/xtiankisutsa/MARA_Framework
[9] https://github.com/MobSF/Mobile-Security-Framework-MobSF
[10] https://owasp.org/www-project-web-security-testing-guide/

3. Metodologia

A Figura 1 apresenta o fluxograma da metodologia seguida nesta pesquisa e as seções seguintes detalham cada fase e os artefatos de entradas.

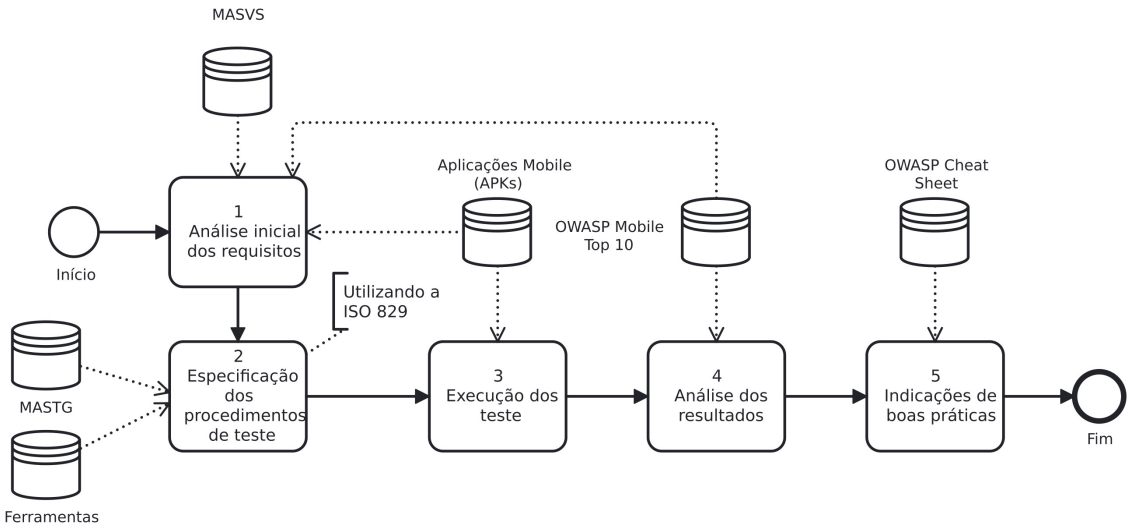


Figura 1. Fluxograma da metodologia de testes

3.1. Análise inicial dos requisitos

Nesta etapa inicial, é feita uma verificação da interface de cada aplicativo alvo, afim de compreender as características e funcionalidades do mesmo. Por meio dessa análise da interface da aplicação, é possível destacar quais requisitos de segurança estão relacionados com a(s) funcionalidade(s) analisada(s). Os requisitos verificados são mencionados no guia MASVS. Vale ressaltar que a não conformidade com os requisitos pode fazer com que a aplicação seja alvo de potenciais vulnerabilidades, como as elencadas na lista do OWASP *Mobile Top* 10. A versão utilizada da lista é de 2016, pois a de 2023 ainda está em fase de lançamento. A Tabela 1 apresenta essas vulnerabilidades.

Tabela 1. Lista de vulnerabilidades do OWASP *Mobile Top* 10

Índice	Vulnerabilidade
M1	Uso inadequado da plataforma
M2	Armazenamento de dados inseguro
M3	Comunicação insegura
M4	Autenticação insegura
M5	Criptografia insuficiente
M6	Autorização insegura
M7	Qualidade do código do cliente
M8	Adulteração de código
M9	Engenharia reversa
M10	Funcionalidade estranha

O guia MASVS da OWASP estabelece requisitos de segurança para aplicações móveis, podendo ser utilizado como orientação na fase de desenvolvimento e como uma lista de verificação de segurança. Além disso, o MASVS apresenta um modelo de

segurança separado por níveis de acordo com a criticidade de segurança, partindo do verificador a escolha do nível e dos requisitos associados. A versão 1.5.0 do guia foi adotada neste artigo [OWASP 2023].

3.2. Especificação dos procedimentos de teste

Após a análise inicial, em que são escolhidos os requisitos que serão utilizados como base na verificação de segurança, serão especificados os variados procedimentos de teste. Para tanto, será utilizado o guia MASTG e as ferramentas indicadas por ele.

O MASTG, um guia também da OWASP, trata da execução dos testes de segurança, apresentando de forma detalhada as técnicas, os processos e as ferramentas utilizadas durante a análise de segurança dos aplicativos. Cada teste tem a identificação de um ou mais requisitos apresentados no MASVS, e o resultado do teste servirá para garantir se o requisito está sendo cumprido ou não na aplicação móvel. Por exemplo, o teste "Verificando a criptografia de dados na rede (*MSTG-NETWORK-1*)" faz a ligação com o requisito 1 da categoria de comunicação de rede do MASVS, referenciada como *Network*. A versão 1.5.0 do guia foi adotada neste artigo [OWASP 2022].

Apesar do MASTG apresentar os testes a serem feitos, fica a cargo do profissional definir as etapas que ele irá seguir nos testes, ou seja, o passo a passo. Por causa disso, este artigo tenta apresentar as etapas a serem seguidas mais claramente, o que vai ajudar os testadores com informações mais precisas e formalizadas.

Podem ser utilizadas quaisquer ferramentas para analisar as aplicações, sejam elas disponíveis atualmente ou criadas futuramente. Para este trabalho, foram escolhidas ferramentas diretamente ligadas aos procedimentos de teste abordados na Seção 4.

A ISO[11] tem como objetivo principal aprovar normas internacionais em todos os campos técnicos, como normas técnicas, classificações de países, normas de procedimentos e processos, entre outros. Dentre as diversas normas, existe a ISO 829, que define um conjunto de documentos que devem ser produzidos durante o processo de teste de software. Esses documentos são elaborados para fornecer informações detalhadas sobre o planejamento, execução e relatório dos testes. A versão utilizada está documentada nos registros do *Institute of Electrical and Electronics Engineers* (IEEE) [IEEE 2008]. Para a especificação dos procedimentos de teste, foi utilizado esse padrão.

Um dos documentos é o de procedimentos de teste, o qual detalha os passos específicos a serem seguidos para executar cada caso de teste. Eles podem incluir instruções para configurar o ambiente de teste, inserir os dados de teste e verificar os resultados [IEEE 2008]. É com base nesse documento que a especificação dos procedimentos de teste apresentada neste artigo foi pensada.

3.3. Execução dos testes e análise dos resultados

Nessa fase, são feitos os testes propostos na fase anterior. Os tipos de testes podem ser divididos em testes estáticos, em que só é analisado o código fonte e binário da aplicação sem executá-la, e testes dinâmicos, em que é verificada a execução do aplicativo e o comportamento das funcionalidades [Grossman et al. 2009]. Também pode ser incluído o teste iterativo que abrange tanto testes automatizados como testes manuais, mas só foi

[11] `https://www.iso.org/home.html`

utilizado os 2 primeiros tipos neste trabalho. No final, é evidenciado o que a execução dos testes resultou e como comprovar a existência de uma vulnerabilidade na aplicação, por meio de imagens das saídas das ferramentas do ponto onde elas identificaram um problema. Além disso, são mencionadas maneiras de mitigar os problemas encontrados por meio de boas práticas de implementação de código.

3.4. Indicações de boas práticas

Visto a execução dos testes e a análise dos resultados gerados, falta ainda mencionar maneiras de como corrigir ou evitar as vulnerabilidades encontradas afim de conscientizar desenvolvedores e profissionais de testes sobre a importância de produzir software de maneira segura. Isso vai fazer com que problemas futuros oriundos de vulnerabilidades sejam evitados. Para esse fim, o artigo utiliza o OWASP *Cheat Sheet* [Manico and et al. 2023].

Esse guia da OWASP fornece informações e medidas, tanto para aplicações móveis como para aplicações *Web* indicadas por especialistas em segurança de aplicativos para combater as possíveis ameaças. Este artigo utiliza o guia para indicações de boas práticas a fim de mitigar as vulnerabilidades encontradas.

4. Procedimentos de teste

Durante a aplicação da metodologia apresentada na Seção 3, a fim de avaliar as vulnerabilidades do OWASP *Mobile Top* 10, foram definidos 6 procedimentos de testes.

Neste artigo, por limitação de espaço, são apresentados 6 procedimentos de testes, em que cada procedimento se relaciona com uma das categorias de requisitos do MASVS. Tal escolha levou em conta o quanto de vulnerabilidades do OWASP *Mobile Top* 10 são cobertas e analisadas pelos procedimentos, além de ser interessante mostrar um pouco da diferença entre as categorias do MASVS. Para cada procedimento, são destacados conforme indicação da ISO 829:

1. **Identificador**: é o identificador do procedimento de teste. Neste artigo foi utilizado o padrão PT_XX, onde XX se referencia a uma categoria de requisitos do MASVS, mostrando que o procedimento de teste cobre um ou mais requisitos dessa categoria. Por exemplo, PT_INT_PLAT indica que o procedimento se relaciona com a categoria “Requisitos de Interação com a Plataforma”.
2. **Objetivo**: define a finalidade da execução do procedimento.
3. **Requisitos Especiais**: refere-se a necessidades que precisam ser atendidas antes de realizar o procedimento, como a instalação de alguma ferramenta.
4. **Configuração**: são os ajustes que precisam ser feitos nas ferramentas, no ambiente em que o procedimento deve ser realizado ou nos objetos que serão alvos do procedimento, como as APKs e a máquina virtual que emula a imagem do *Android*.
5. **Início**: um passo ou gatilho específico para iniciar o procedimento, como clicar para executar uma ferramenta pela interface gráfica dela.
6. **Passos**: são as etapas realizadas enquanto o procedimento é executado.
7. **Parada**: descrição da(s) condição(ões) alcançadas que determinam o fim do procedimento.
8. **Saída**: descrição do que o testador deve utilizar para comprovar os resultados do procedimento, como um *print* dos resultados gerados por alguma ferramenta.

Os 6 procedimentos propostos neste trabalho são exibidos nas Figuras 2 e 3. As ferramentas utilizadas no procedimentos de teste que vão ser abordados na Seção 5 também estão descritas a seguir.

Figura 2. Procedimentos de teste PT_INT_PLAT, PT_ARM_DAD_PRIV e PT_CRIPTO

Identificador		PT_INT_PLAT
Objetivo		Identificar se dados confidenciais vazaram quando os mecanismos de comunicação inter-processos (IPC) são usados.
Requisitos especiais		Ferramenta MobSF instalada; Versão da aplicação no formato APK.
Etapas do procedimento	**Configuração**	MobSF configurada para análise estática;
	Início	Iniciar a ferramenta pela linha de comando
	Passos	Passo 1: acessar a interface gráfica da ferramenta Passo 2: fazer o upload da APK Passo 3: executar a ferramenta Passo 4: analisar as informações do relatório que a ferramenta gerou
	Parada	O testador encontrou uma ou mais tag(s) de exportação como true (android:exported="true") e constatou verificando a aba de código-fonte **ou** o testador encontrou uma ou mais <intent-filter> dentro de uma componente (<activity>, <service>, <receiver>) e constatou verificando a aba de código-fonte **ou** o testador não encontrou nada nesse contexto.
	Saída	Print da existência ou não dos elementos descritos em "Parada"

Identificador		PT_ARM_DAD_PRIV
Objetivo		Identificar quaisquer dados de aplicativo confidenciais nos logs do sistema e do aplicativo.
Requisitos especiais		Ferramentas adb e VirtualBox instaladas; Máquina virtual que execute uma imagem da plataforma Android (versão 9 ou superior) Versão da aplicação no formato APK instalada na máquina virtual; Firewall da máquina hospedeira desativado para não interferir na comunicação entre a ferramenta adb e a máquina virtual (opcional).
Etapas do procedimento	**Configuração**	Executar a APK na máquina virtual; Verificar o endereço IP da máquina virtual;
	Início	Executar a ferramenta adb pela linha de comando, utilizando o endereço IP anotado e indicando o uso do utilitário "logcat" (adb endereço_IP logcat).
	Passos	Passo 1: executar as funcionalidades da aplicação que envolvem dados confidenciais Passo 2: acompanhar os logs que a ferramenta adb mostra.
	Parada	O testador encontrou dados confidenciais expostos nos logs **ou** percorreu todo o arquivo de log gerado e não encontrou nenhum dado exposto
	Saída	Print do log contendo dados confidenciais expostos ou não.

Identificador		PT_CRIPTO
Objetivo		Identificar se os protocolos criptográficos e algoritmos usados não são obsoletos para fins de segurança.
Requisitos especiais		Ferramenta MobSF instalada; Versão da aplicação no formato APK.
Etapas do procedimento	**Configuração**	MobSF configurada para análise estática;
	Início	Iniciar a ferramenta pela linha de comando
	Passos	Passo 1: acessar a interface gráfica da ferramenta Passo 2: fazer o upload da APK Passo 3: executar a ferramenta Passo 4: analisar as informações do relatório que a ferramenta gerou
	Parada	O testador encontrou um ou mais algoritmos criptográficos obsoletos ou mal configurados indicados pela análise da ferramenta **ou** a ferramenta encontrou os algoritmos criptografados sem problemas.
	Saída	Print da existência ou não dos elementos descritos em "Parada"

MobSF é uma ferramenta automatizada utilizada para pentesting, análise de malware e fornece uma avaliação de testes estáticos e dinâmicos em aplicações móveis, ou seja, faz a busca por vulnerabilidades, suportando alguns binários como o APK.

O *Android Debug Bridge*[12] (adb) é uma ferramenta de linha de comando bastante prática e que permite a comunicação com um dispositivo *Android*. Por meio dela, são permitidas variadas ações no dispositivo, como instalação e depuração de aplicativos, visualização das mensagens de *log* e acesso a um terminal *Unix*.

[12] `https://developer.android.com/tools/adb`

Figura 3. Procedimentos de testes PT_AUT_GER_SES, PT_RES e PT_COM_REDE

Identificador		PT_AUT_GER_SES
Objetivo		Identificar se, caso haja alguma forma de autenticação, ela será executada no terminal remoto.
Requisitos especiais		Ferramenta MobSF instalada; Versão da aplicação no formato APK.
Etapas do procedimento	**Configuração**	MobSF configurada para análise estática;
	Início	Iniciar a ferramenta pela linha de comando
	Prosseguimento	Passo 1: acessar a interface gráfica da ferramenta Passo 2: fazer o upload da APK Passo 3: executar a ferramenta Passo 4: analisar na aba da ferramenta de arquivos de código-fonte se há formas de autenticação insegura, como credenciais armazenadas localmente e não protegidas corretamente com criptografia ou até mesmo no código-fonte
	Parada	O testador encontrou uma ou mais formas de autenticação insegura **ou** o testador encontrou autenticação utilizando terminal remoto corretamente configurada.
	Saída	Print da existência ou não dos elementos descritos em "Parada"

Identificador		PT_RES
Objetivo		Verificar se o aplicativo detecta que um ou mais arquivos do mesmo foram alterados (anti-adulteração de código).
Requisitos especiais		Ferramentas MobSF, apktool e VirtualBox instaladas; Versão da aplicação no formato APK;
Etapas do procedimento	**Configuração**	MobSF configurada para análise estática;
	Início	Iniciar a ferramenta MobSF pela linha de comando
	Prosseguimento	Passo 1: acessar a interface gráfica da ferramenta Passo 2: fazer o upload da APK Passo 3: executar a ferramenta Passo 4: identificar, na aba de "código-fonte decompilado" gerado pela MobSF, algum trecho de código que interfira em alguma funcionalidade crítica da aplicação Passo 5: modifique, manualmente, o trecho de código identificado anteriormente Passo 6: com o apktool, compile e assine novamente a aplicação com alteração Passo 7: instale a aplicação na máquina virtual Passo 8: manipule a funcionalidade da aplicação na máquina virtual
	Parada	Observe se a aplicação mostra algum comportamento estranho em relação à alteração de código **ou** se acontece naturalmente sem nenhuma mudança, ao menos, na interface.
	Saída	Print da existência ou não dos elementos descritos em "Parada"

Identificador		PT_COM_REDE
Objetivo		Verifique se um certificado vem de uma fonte confiável, ou seja, de uma CA (autoridade certificadora) confiável e se está sendo tratado corretamente.
Requisitos especiais		Ferramenta MobSF instalada; Versão da aplicação no formato APK.
Etapas do procedimento	**Configuração**	MobSF configurada para análise estática
	Início	Iniciar a ferramenta pela linha de comando
	Prosseguimento	Passo 1: acessar a interface gráfica da ferramenta Passo 2: fazer o upload da APK Passo 3: executar a ferramenta Passo 4: identificar os componentes "WebViews" da aplicação Passo 5: identificar trechos do código-fonte onde contém tratamento de certificados de rede Passo 6: verificar se os trechos estão sendo tratados corretamente
	Parada	Vericar se está ocorrendo tratamento e reconhecimento correto dos certificados **ou** verificar que há tratamento inadequado dos certificados.
	Saída	Print da existência ou não dos elementos descritos em "Parada"

Ao invés de se utilizar um dispositivo físico para instalar as aplicações e realizar a análise, é mais conveniente e seguro utilizar um emulador de dispositivo *Android*. Para este trabalho, será utilizado uma máquina virtual do *VirtualBox*[13] que irá executar a imagem oficial do *Android*[14].

5. Execução e resultados

Ao todo, foram utilizados 5 aplicações no formato APK para a realização dos experimentos: Diva, *InsecureShop*, *AndroGoat*, *InsecureBankv2* e *VulnApp*. Esses aplicativos,

[13] https://www.virtualbox.org/
[14] https://www.android-x86.org/

em conjunto, apresentam as 10 vulnerabilidades listadas no OWASP *Mobile Top* 10. Já em relação aos procedimentos de teste, eles são utilizados para averiguar a existência ou não de vulnerabilidades em aplicações. Por questões de espaço, a seguir são abordadas somente as 3 principais vulnerabilidades encontradas e os procedimentos de teste utilizados. Os resultados completos, com as outras 7 vulnerabilidades, estão disponíveis no repositório do Github[15].

5.1. Uso indevido de plataforma

Essa vulnerabilidade foi descoberta por meio do procedimento de teste com identificador "PT_INT_PLAT", o qual é descrito na Figura 2. A aplicação alvo foi a Diva, que foi projetada para ser insegura com o objetivo de ensinar falhas oriundas de codificação ruim ou insegura. A Figura 4 mostra a detecção de uma *intent-filter* pela ferramenta MobSF, enquanto a Figura 5 mostra a existência da *intent-filter* no código-fonte. Ela chama uma das telas da aplicação, tornando-a explicitamente exportável e fazendo com que outra aplicação instalada no mesmo dispositivo possa acessar as possíveis informações contidas na componente sem ter que executar de fato a aplicação.

3	**Activity** (jakhar.aseem.diva.APICredsActivity) is not Protected. An intent-filter exists.	warning	An Activity is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Activity is explicitly exported.

Figura 4. Detecção da ferramenta MobSF da presença de uma *Activity* exportada publicamente na aplicação Diva

```
<activity android:label="@string/apic_label" android:name="jakhar.aseem.diva.APICredsActivity">
    <intent-filter>
        <action android:name="jakhar.aseem.diva.action.VIEW_CREDS" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

Figura 5. Trecho do código onde mostra a presença perigosa de uma *Activity* exportada publicamente na aplicação Diva

Isso afeta o requisito de segurança do MASVS de nível L1 denominado "MSTG-PLATFORM-4: O aplicativo não exporta funcionalidades sensíveis através do IPC, a menos que esses mecanismos estejam devidamente protegidos". O teste do MASTG para averiguar esse requisito é denominado "Teste para exposição de funcionalidade sensível por meio de IPC (MSTG-PLATFORM-4)". Se a lógica da aplicação indicar que esse componente só deva ser executado pelo próprio aplicativo, não se usa *intent-filter*, e sim se define o atributo *"exported"* como *"false"* para a componente e se utiliza *intents* explícitas.

5.2. Armazenamento de dados inseguro

A vulnerabilidade foi encontrada utilizando o procedimento de teste com identificador "PT_ARM_DAD_PRIV" (ver Figura 2), e a aplicação alvo também foi a Diva. Executando o aplicativo na máquina virtual e verificando suas funcionalidades, observa-se

[15] https://github.com/leosilva99/verificacao_owasp_mobile_10

a funcionalidade de cadastro de um cartão de crédito, como poder ser visto Figura 6. Com o procedimento de teste, é possível verificar que dados confidenciais estavam aparecendo em texto plano entre as mensagens de *log*, como demonstrado na Figura 7. Nela, é possível ver que o dado inserido está contido na mensagem de erro gerada em texto plano, o que é inapropriado.

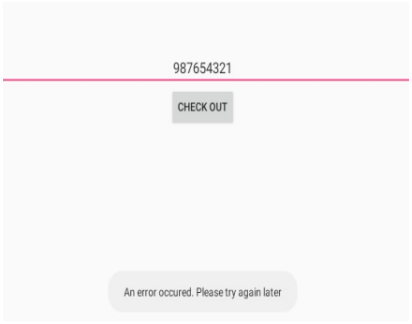


Figura 6. Visualização da utilização da aplicação em que é detectado que informações sensíveis estão sendo vazadas na aplicação Diva

```
10-12 19:39:36.612  3573  3573 E diva-log: Error while processing
transaction with credit card: 987654321
```

Figura 7. Visualização da mensagem de *log* que representa um erro na aplicação em que é exibida a informação digitada na Figura 6

Esse problema afeta o requisito de segurança do MASVS de nível L1 denominado "MSTG-STORAGE-3: Dados sensíveis não podem aparecer nos *'logs'* de aplicação". O teste do MASTG para averiguar esse requisito é denominado "Testando *logs* para dados confidenciais (MSTG-STORAGE-3)". Deve-se ter cuidado com o que será passado de informações nas mensagens de *log* e a forma como são exibidas. Não é indicado registrar informações confidenciais desprotegidas como parte de uma operação normal e por conta própria - que não seja mensagem gerada pelo próprio sistema *Android* e sem intervenção do programador - a não ser que seja estritamente importante para a aplicação [16].

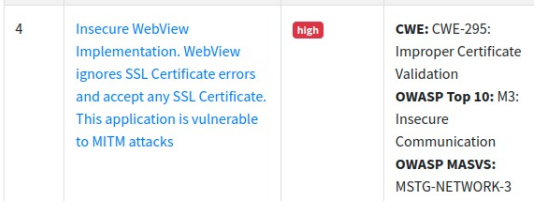
5.3. Comunicação insegura

A vulnerabilidade foi descoberta com o uso do procedimento de teste com identificador "PT_COM_REDE"(ver Figura 3). A aplicação alvo foi a *InsecureShop*, projetada em *Kotlin* especificamente para ser vulnerável e ser elemento de estudos de pesquisadores de segurança de aplicativos. Foi detectado que uma *WebView*, que faz parte da *Activity "WebViewActivity"*, foi implementada com a falta de verificação do certificado SSL, sem verificar erros e a validade do mesmo, como apresentado na Figura 8.

Na Figura 9, é possível ver o trecho do código onde está a variável que representa o possível erro de certificado SSL, *"SslError error"*. Não é feita nenhuma verificação nessa varável, somente é feita a chamada de função "handler.proceed()", que só deve ser chamada se o certificado é válido. Isso pode acarretar em aceitar comunicações com

[16] `https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html`

certificados defasados e o usuário ficar suscetível ao ataque MITM (ataque do homem no meio), em que um atacante pode monitorar a comunicação e roubar dados confidenciais que possam ser transferidos na comunicação.



4	Insecure WebView Implementation. WebView ignores SSL Certificate errors and accept any SSL Certificate. This application is vulnerable to MITM attacks	high	**CWE:** CWE-295: Improper Certificate Validation **OWASP Top 10:** M3: Insecure Communication **OWASP MASVS:** MSTG-NETWORK-3

Figura 8. Detecção da ferramenta MobSF da falta de verificação de erros de certificados SSL na aplicação *InsecureShop*

```
public final class CustomWebViewClient extends WebViewClient {
    @Override // android.webkit.WebViewClient
    public void onReceivedSslError(WebView view, SslErrorHandler handler, SslError error) {
        if (handler != null) {
            handler.proceed();
        }
    }
}
```

Figura 9. Trecho do código onde está a falta de verificação de erros de certificados SSL na aplicação *InsecureShop*

Esse problema afeta o requisito de segurança do MASVS de nível L1 denominado "MSTG-NETWORK-3: O aplicativo verifica o certificado X.509 do terminal remoto quando o canal seguro é estabelecido. Apenas certificados assinados por uma CA confiável são aceitos". O teste do MASTG para averiguar esse requisito é denominado "Testando para verificação de identificação do *endpoint* (MSTG-NETWORK-3)". A verificação de certificados SSL deve averiguar se a CA que assinou o certificado é confiável, se o certificado ainda não expirou e se ele é auto assinado. Uma maneira simples é utilizar a interface *TrustManager*, que contém os meios necessários para fazer as verificações de validade de um certificado [17].

6. Conclusão

Na Seção 5, onde foram apresentados os testes de segurança, os resultados indicaram que as vulnerabilidades identificadas podem ser bastante perigosas e que não devem ser ignoradas. Seguindo a metodologia definida na Seção 3 e os procedimentos de teste descritos na Seção 4, foi possível identificar as vulnerabilidades seguindo uma orientação padrão para testadores de software e indicar maneiras de como resolver esses problemas.

As más práticas de programação, pelo que foi mostrado nos testes, são as principais vilãs no contexto das vulnerabilidades enumeradas pelo OWASP *Mobile Top 10*. Tanto que as recomendações para se contornar esses problemas, em maioria, envolviam seguir os padrões atestados pela linguagem ou *framework* utilizada.

Além disso, uma padronização da maneira como atestar a existência ou não de vulnerabilidades é importante para se ter resultados convincentes. Foi mostrado, para

[17] `https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html`

cada vulnerabilidade, que um ou mais requisitos de segurança elencados pelo MASVS não estavam sendo seguidos. A utilização dele na fase de análise de requisitos e riscos do projeto, e a utilização das ferramentas indicadas pelo MASTG, bem como a maneira de realizar os testes, são instrumentos bastante completos e confiáveis, que trazem uma maior garantia de que o número de vulnerabilidades críticas no projeto será minimizada.

A utilização da ISO 829 também mostra-se como uma boa alternativa para formalizar os testes feitos. Utiliza uma abordagem que os testadores de *software* conseguem entender, além de fornecer uma boa descrição do que é feito em cada procedimento e os objetivos que ele deve alcançar.

Para trabalhos futuros, planeja-se fazer uma pesquisa entre testadores e profissionais de segurança para saber a opinião deles sobre a apresentação dos testes de segurança seguindo a ISO 829, já que isso deixaria mais claro o quanto a padronização dos testes é importante para agregar segurança a testes de *software*. Além disso, também pode-se abordar os testes de segurança em aplicações *Web* com o formato da ISO 829, utilizando o guia WSTG para verificações de segurança em aplicações *Web*.

Referências

Alanda, A., Satria, D., Mooduto, H., and Kurniawan, B. (2020). Mobile application security penetration testing based on owasp. In *IOP Conference Series: Materials Science and Engineering*, volume 846, page 012036. IOP Publishing.

Aljabri, M., Aldossary, M., Al-Homeed, N., Alhetelah, B., Althubiany, M., Alotaibi, O., and Alsaqer, S. (2022). Testing and exploiting tools to improve owasp top ten security vulnerabilities detection. In *2022 14th International Conference on Computational Intelligence and Communication Networks (CICN)*, pages 797–803. IEEE.

Grossman, J., Eng, C., Spitler, R., and Wood, M. (2009). Static dynamic analysis for web applications. `https://owasp.org/www-pdf-archive/Atlanta_March_2010_Presentation.pdf`.

IEEE (2008). IEEE standard for software test documentation. *IEEE 829:2008*.

Kohli, N. and Mohaghegh, M. (2020). Security testing of android based covid tracer applications. In *2020 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)*, pages 1–6. IEEE.

Manico, J. and et al., J. M. (2023). Owasp cheat sheet series. `https://cheatsheetseries.owasp.org/`. Acessado: 2023-06-21.

OWASP (2022). Owasp mobile application security testing guide (mastg). `https://mas.owasp.org/MASTG/`. Acessado: 2023-06-21.

OWASP (2023). Owasp mobile application security verification standard (masvs). `https://mas.owasp.org/MASVS/`. Acessado: 2023-06-21.

Priambodo, D. F., Ajie, G. S., Rahman, H. A., Nugraha, A. C. F., Rachmawati, A., and Avianti, M. R. (2022). Mobile health application security assesment based on owasp top 10 mobile vulnerabilities. In *2022 International Conference on Information Technology Systems and Innovation (ICITSI)*, pages 25–29. IEEE.

Silva, L. O. (2022). Testes de segurança em aplicações android baseados na metodologia owasp. *Repositório Institucional da Universidade Federal do Ceará - UFC*.