Teste de Segurança em Aplicativos Mobile

Renan Dogenski

Instituto de Tecnologia – Universidade de Passo Fundo (UPF)
Caixa Postal 99.052-900 – Passo Fundo – RS – Brazil

Instituto de Tecnologia
Universidade de Passo Fundo (UPF) – Passo Fundo, RS –
Brazil

134397@upf.br

Abstract. *Due to technological evolution and the advent of the smartphone, vulnerability problems arise that can go unnoticed. Every day, the Play Store, the platform that distributes apps, receives applications to be added to its store. In view of this, it is imperative to take care that these applications do not have vulnerabilities that could compromise security and, thus, harm their users. In this sense, it is important to use testing strategies to check for possible adversities. This article, therefore, focuses on vulnerabilities in the Android system. As such, the results obtained were an exposition of what each of the vulnerabilities can cause and also a brief solution*

Resumo. *Devido à evolução tecnológica e ao advento do smartphone, surgem problemas de vulnerabilidade que podem passar despercebidos. Diariamente, a Play Store, plataforma que distribui aplicativos, recebe aplicações para serem inseridas em sua loja. Diante disso, é imperativo o cuidado que tais aplicações não possuam vulnerabilidades que possam vir a comprometer a segurança e, assim, prejudicar seus usuários. Nesse sentido, torna-se relevante o uso de estratégias de testes para averiguar possíveis adversidades quanto à presença delas. Por isso, este artigo enfoca em vulnerabilidades do sistema Android. Sendo assim, os resultados obtidos foram uma exposição do que cada uma das vulnerabilidades pode causar e também uma breve solução.*

1. Introdução

O Android consiste em um sistema operacional para dispositivos móveis, desenvolvido pela Google. Ele é usado em uma variedade de dispositivos, tais como: smartphones, tablets, TVs inteligentes, relógios inteligentes e outros aparatos eletrônicos. Ademais, também é conhecido por sua interface de usuário personalizável e pela ampla gama de aplicativos disponíveis na *Google Play Store*. Dessa forma, existem várias linguagens utilizadas para desenvolver aplicações; dentre elas, destacam-se: *Java*, *Kotlin* e *C++*, ainda que, atualmente, haja *frameworks* para o desenvolvimento de aplicações para o *Android* em outras linguagens, como, por exemplo, *Flutter* e *React Native* (MASTG, 2016). Segundo o site oficial da plataforma *Android*, mais de 3 milhões de aplicativos estão disponíveis na *Google Play Store*. Desse montante, tais dispositivos podem apresentar diversas funções, podendo ser fotografia e vídeos, educação, saúde e bem-estar, entre outras.

Além disso, segundo matéria do site *Convergência Digital* (2023), o mundo

possui 8,4 bilhões de celulares ativos, o que significa que a penetração sobre a população foi de 106% ao final de 2022.

Até um certo tempo, o programador se preocupava somente com o desenvolvimento da aplicação, deixando de lado a parte de segurança. No entanto, atualmente - onde praticamente tudo é possível ser feito através de aplicações - a preocupação com a segurança se faz muito necessária devido aos dados sensíveis que são utilizados. Um exemplo disso são aplicativos financeiros que demandam total sigilo e segurança das informações dos usuários, transações, saldo em conta, entre outros dados.

Nesse contexto, testes de segurança de aplicações móveis têm se mostrado cruciais, principalmente nas fases de desenvolvimento e de testes propriamente ditos, em que são utilizadas abordagens estáticas e dinâmicas para analisar o código ou a aplicação em execução e identificar possíveis vulnerabilidades.

Diante disso, este trabalho possui como objetivo avaliar os recursos disponíveis na internet para testar a segurança em aplicativos móveis.

2. Aporte teórico

Ameaças na área de desenvolvimento de software são potenciais problemas, riscos e desafios que podem afetar negativamente o processo de desenvolvimento de tal, a qualidade do produto final ou a segurança dos sistemas. Elas podem surgir em diferentes estágios do ciclo de vida do desenvolvimento de software e podem ter várias origens.

Vulnerabilidades em aplicativos *Android* referem-se a fraquezas ou falhas de segurança que podem ser exploradas por atacantes para comprometer a integridade, confidencialidade ou disponibilidade de um aplicativo ou do dispositivo *Android* em si. Essas brechas podem ocorrer devido a erros de programação, configurações inadequadas, falta de atualizações de segurança e outras questões relacionadas à estabilidade. Esses riscos são referentes à possibilidade de ameaças explorarem os pontos de fraqueza existentes (Peltier, 2010). Nesse sentido, convém identificar as fontes de possíveis problemas que podem fazer com que as fraquezas se tornem vulnerabilidades.

2.1 CWE

Common Weakness Enumeration (CWE™) é uma lista, desenvolvida pela comunidade de desenvolvimento de tipos comuns de fraquezas de software e hardware que têm ramificações de segurança. Uma *fraqueza* é uma condição num software, firmware, hardware ou componente de serviço que, sob certas circunstâncias, pode contribuir para a introdução de vulnerabilidades (CWE,2023).

Voltado para as comunidades de desenvolvimento e profissionais de segurança, o principal objetivo do CWE é interromper as vulnerabilidades na fonte, educando arquitetos de software e hardware, designers, programadores e adquirentes sobre como eliminar os erros mais comuns antes da entrega dos produtos. Em última análise, o uso desse sistema ajuda a prevenir os tipos de vulnerabilidades de segurança que têm atormentado usuários, indústrias de software e hardware e colocado empresas em risco

(CWE,2023).

2.2 OWASP

A *Open Worldwide Application Security Project* (OWASP) é uma fundação sem fins lucrativos que trabalha para melhorar a segurança dos softwares, sendo que tudo é feito através de seus projetos de código aberto que são liderados pela comunidade (OWASP, 2023).

A OWASP lidera vários projetos, dentre os quais um deles é o *OWASP Top Ten Mobile*, cuja função é listar as dez principais vulnerabilidades que são reportadas pela comunidade e as quais possuem como principal motivo alertar aos desenvolvedores os principais riscos de segurança (OWASP, 2023).

As vulnerabilidades da tabela abaixo são do ano de 2023, estando ranqueadas conforme o número de casos reportados para as vulnerabilidades em dispositivos móveis.

Tabela 1. Tabela de vulnerabilidades

Número	Vulnerabilidade
1	Uso impróprio de credenciais
2	Segurança inadequada da cadeia de suprimentos
3	Autenticação/autorização insegura
4	Validação de entrada/saída insuficiente
5	Comunicação insegura
6	Controles deprivacidade inadequados
7	Proteções binárias insuficientes
8	Configuração incorreta de segurança
9	Armazenamento de dados inseguros
10	Criptografia insuficiente

Fonte: https://owasp.org/www-project-mobile-top-10/

A entidade fornece também diretrizes com as principais metodologias, guias técnicos e boas práticas para o pentest, assim como aplicativos vulneráveis e ferramentas para pentest para quem se interessa pela área e tem pouco conhecimento, mas gostaria de aprender mais sobre isso.

2.3 Diva

Damn Insecure Vulnerable App, também conhecido como *Diva*, é um aplicativo vulnerável que foi desenvolvido pelo indiano Aseem Jakhar com o intuito de ensinar sobre vulnerabilidades em aplicativos *Android*. Sendo assim, apresenta como função ensinar aos desenvolvedores/profissionais de controle de qualidade/segurança sobre as falhas que geralmente estão presentes nos aplicativos devido a práticas de codificação inadequadas ou inseguras (GITHUB,2023).

De posse o arquivo apk, pode ser instalado num smartphone físico para efetuar

testes dinâmicos, mas o recomendado é utilizar emuladores Android para ter mais segurança e evitar que o dispositivo possa ter algum problema de segurança devido a algumas permissões que serão necessárias serem dadas a esse aplicativo e para o sistema do smartphone, que seria a habilitação do *root* para ter permissões de super usuário.

2.4 AndroGoat

AndroGoat é um aplicativo vulnerável que foi desenvolvido por dois colaboradores: os indianos Dheeraj Kakkar e Satish Patnayak. Seu intuito se baseia em compreender e defender as vulnerabilidades na plataforma *Android*. Esse aplicativo pode ser obtido através do *Github* (GITHUB,2023).

2.5 MobSF

MobSF trata-se de uma ferramenta de código aberto que foi desenvolvida por quatro colaboradores, a saber: o indiano *Ajin Abrão*, o chinês *Magaofei*, o israelense *Matan Dobrushin* e o francês *Vincent Nadal*. Tal ferramenta é utilizada para fazer análise estática e dinâmica de aplicativos móveis em busca de vulnerabilidades, servindo tanto para *Android* como *iOS*. Para tanto, pode ser feita uma instalação local em computador ou também pode ser executada de forma online, através de navegador (THOUGHTWORKS, 2023).

3. Metodologia

A fim de avaliar a eficiência da ferramenta MobSF e de conhecer mais detalhes acerca dos cuidados com a segurança no desenvolvimento de aplicativos para dispositivos móveis, neste capítulo, será apresentado de que forma a ferramenta MobSF pode ser utilizada para fins de verificação de fraquezas em aplicativos. Em sentido estrito, serão apresentados os passos para colocá-la em funcionamento na análise do código de uma aplicação.

Como mencionado anteriormente, a MobSF é uma ferramenta que pode ser instalada localmente em um computador ou pode ser utilizada em sua versão on-line por meio de um navegador sem necessidade de instalação de qualquer pacote.

3.1 Versão online

Para a utilização dela diretamente via navegador e sem a instalação de qualquer pacote, basta acessar *https://mobsf.live/*. Em seguida, fazer upload do arquivo que será analisado. Para que a análise estática possa ser feita, utiliza-se o arquivo binário **(**apk, xapk, ippa e appx) ou um pacote zipado contendo todos os arquivos do aplicativo. Já em relação à análise dinâmica, necessita-se não apenas instalar o aplicativo em um smartphone e executá-lo mantendo-o conectado via USB com as permissões de administrador, como também estar em modo de depuração, ou utilizar um emulador.

Após o upload do arquivo, a ferramenta inicia a análise automaticamente. Ao final dela, pode ser gerado um relatório em formato PDF ou serem verificados os resultados dentro da ferramenta. Ademais, são apresentadas informações, como: nome do aplicativo, ícone, tamanho do aplicativo e também o *hash*. Também, há informações a respeito do certificado, se possuir. As permissões do aplicativo ficam abaixo das informações do certificado; nela, contem qual o tipo da permissão, status e informações

sobre cada permissão contendo uma breve descrição. Além disso, existem informações sobre *API Android*; nela, pode-se visualizar a API usada no aplicativo. As atividades navegáveis possuem alguns menus interativos que ajudam na navegação, conforme mostra a figura abaixo.

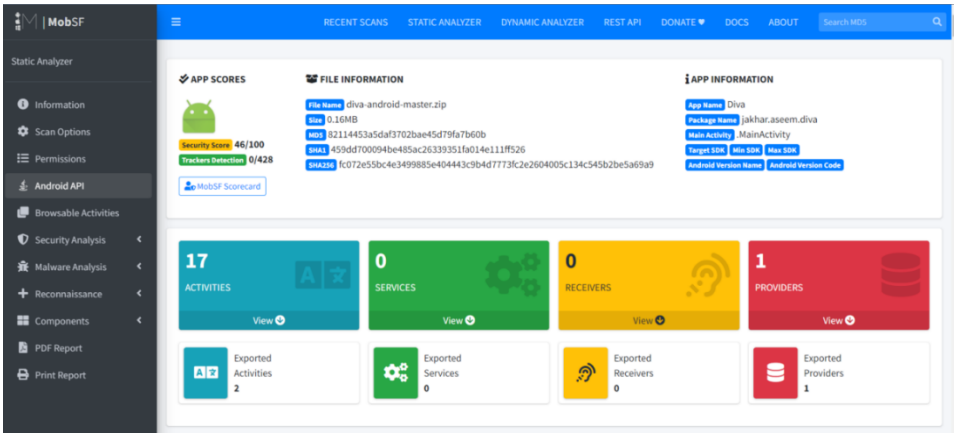


Figura 1 - Tela inicial do *MobSF* após varredura de aplicativo

Fonte: https://mobsf.live/

3.2 Versão local

Na versão com instalação local, há uma sequência de passos a ser seguida dentre os quais é preciso utilizar o sistema operacional *Linux*. Os seguintes são: instalar o *Oracle JDK 1.7* ou uma versão superior; fazer um git clone no repositório do *MobSF* com ***git clone*** *https://github.com/MobSF/Mobile-Security-Framework-MobSF.git* e ***cd Mobile-Security-Framework-MobSF***; *Python* na versão 2.7 ou superior; *MobSF* com ***./setup.sh***; ***e***xecutar a ferramenta com ***./run.sh***.

Outro modo, utilizando a configuração rápida, basta executar dois comandos na sequência: ***docker pull opensecurity/mobile-security-framework-mobsf:latest*** e, após, ***docker run -it --rm -p 8000:8000 opensecurity/mobile-securityframework-mobsf:latest***

Após a instalação do *MobSF*, há uma janela no navegador para executar a verificação do aplicativo. As funcionalidades são as mesmas das utilizadas na versão online.

3.3 Aplicativo a ser analisado

O aplicativo *Diva*, que será utilizado nos testes desta pesquisa, pode ser obtido através do github, no link: *https://github.com/payatu/diva-android*, fazendo download do arquivo *.zip* que depois será carregado no *MobSF* para a análise.

4. Resultados e Discussões

A fim de avaliar o trabalho proposto para que possa contribuir com o meio científico, na sequência, serão apresentados os resultados obtidos pela ferramenta *MobSF* ao analisar o aplicativo *Diva* e *AndroGoat*, que são notoriamente ferramentas com problemas de segurança.

Ao executar a ferramenta *MobSF*, ela informou ter encontrado seis possíveis vulnerabilidades com sua varredura no aplicativo *Diva* e nove possíveis vulnerabilidades no aplicativo *AndroGoat*, sendo que três das nove estão sendo detalhadas, e as demais não são iguais àquelas que já foram citadas no aplicativo *Diva*. Também está listado o que cada uma dessas fraquezas pode ocasionar caso se encontre em aplicativos. Nas seções a seguir serão detalhados os resultados encontrados pela ferramenta *MobSF* ao fazer varredura nos aplicativos.

4.1 Vulnerabilidades do aplicativo *Diva*

Nas próximas seções, serão apresentadas fraquezas detectadas pela ferramenta *MobSF* ao analisar o aplicativo *Diva*. Ao total, a ferramenta detectou quatro fraquezas que serão detalhadas a seguir.

4.1.1 Uso inadequado da plataforma

Após uma análise estática com o auxílio da ferramenta *MobSF*, foi detectada uma vulnerabilidade que, segundo a OWASP (2016), abrange o uso indevido de um recurso da plataforma ou a falha no uso dos controles de segurança dela. Pode incluir intenções do Android, permissões de plataforma, uso indevido do *TouchID*, do *Keychain* ou algum outro controle de segurança que faça parte do sistema operacional móvel, conforme mostra a seguir.

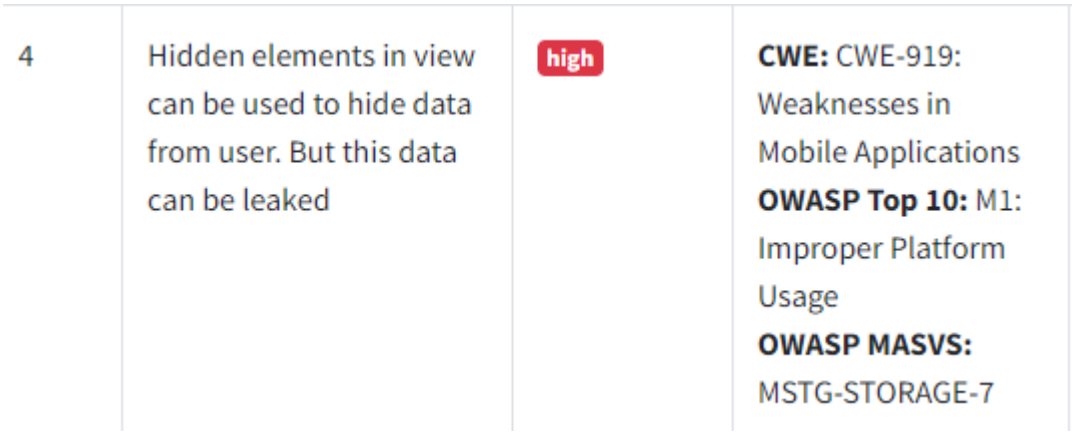


Figura 2 - Falha encontrada e seu grau de severidade

Fonte: captura de tela via https://mobsf.live/static_analyzer/82114453a5daf3702bae45d79fa7b60b/

Ainda segundo a OWASP (2016), para que essa vulnerabilidade seja explorada, a organização deve expor um serviço web ou uma chamada de API que seja consumida pelo aplicativo móvel. O serviço exposto ou chamada de API é implementado usando técnicas de codificação inseguras que produzem uma vulnerabilidade. Através da interface móvel, um atacante, assim, é capaz de inserir entradas maliciosas ou sequências inesperadas de eventos para o endpoint vulnerável.

```
if (userpin.equals(pin)) {
    // Display the private notes
    ListView  lview = (ListView) findViewById(R.id.aci3nlistView);
    Cursor cr = getContentResolver().query(NotesProvider.CONTENT_URI, new String[] {"_id", "title", "note"}, null, null, null);
    String[] columns = {NotesProvider.C_TITLE, NotesProvider.C_NOTE};
    int [] fields = {R.id.title_entry, R.id.note_entry};
    SimpleCursorAdapter adapter = new SimpleCursorAdapter(this, R.layout.notes_entry ,cr, columns, fields, 0);
    lview.setAdapter(adapter);
    pinTxt.setVisibility(View.INVISIBLE);
    abutton.setVisibility(View.INVISIBLE);
    //cr.close();

}
```

Figura 3 - Trecho onde está a falha

Fonte: Captura de tela via https://mobsf.live/view_file/?file=jakhar/aseem/diva/AccessControl3NotesActivity.java&md5=82114453a5daf3702bae45d79fa7b60b&type=studio&lines=72,73

Segundo a OWASP (2016), para evitar ser vítima desse tipo de vulnerabilidade, práticas seguras de codificação e configuração devem ser usadas no lado do servidor do aplicativo móvel. Abaixo uma possível correção.

pinTxt.setVisibility(View.Visible)

Abutton.setVisibility(View.Visible)

4.1.2 Armazenamento de Dados Inseguros

O mesmo aplicativo também registra dados que são movimentados dentro da aplicação para o arquivo de log, que, segundo o *MSTG-STORAGE-3* da OWASP (2016) e também a CWE-532 (2016), o registro de todas as informações é útil na etapa de desenvolvimento. Caso sejam deixados permanentemente, podem ocorrer vazamentos de dados confidenciais, como, por exemplo, usuário e senha.

Isso pode ocorrer quando os dados são armazenados de forma não criptografada, ou quando não são implementadas práticas de segurança adequadas para proteger os dados armazenados localmente no dispositivo móvel.

Na figura a seguir, podem-se notar mais detalhes sobre a vulnerabilidade que está presente na versão do aplicativo.

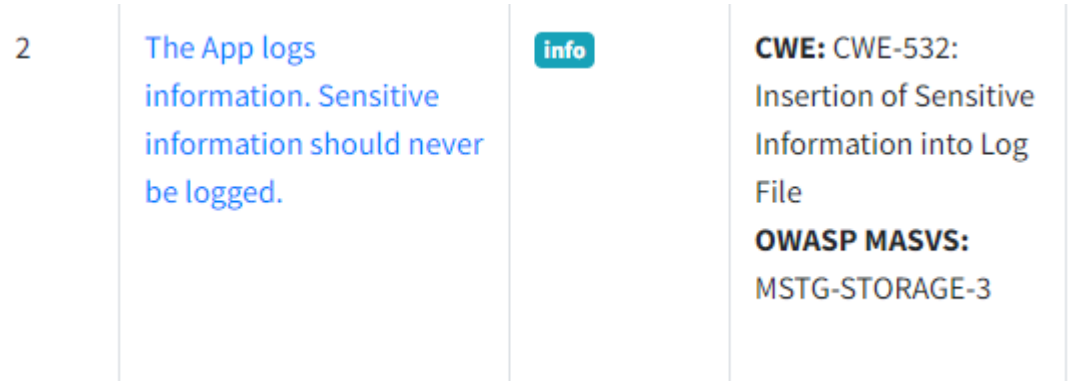


Figura 4 - Detecção do MobSF Fonte: captura de tela via https://mobsf.live/static_analyzer/82114453a5daf3702bae45d79fa7b60b/

Encontrando essa vulnerabilidade, o atacante pode obter uma série de vantagens, como acesso não autorizado a informações sensíveis; dados pessoais dos usuários;

informações de login; histórico de navegação; informações de transações financeiras e outras informações confidenciais. Há também a possibilidade de ocorrer roubo de identidade e informações pessoais, como, por exemplo, números de identificação, dados bancários e/ou de cartão de crédito.

Na figura a seguir, há um trecho do código que contém o problema.

```
public void checkout(View view) {
    EditText cctxt = (EditText) findViewById(R.id.ccText);
    try {
        // Assuming we do some HTTP requests credit card validation and processing
        //Everything seems fine and then we hit some unforseen error
        processCC(cctxt.getText().toString());
    } catch (RuntimeException re) {
        Log.e("diva-log", "Error while processing transaction with credit card: " + cctxt.getText().toString());
        Toast.makeText(this, "An error occured. Please try again later", Toast.LENGTH_SHORT).show();
    }
}
```

Figura 5 - Trecho do código Fonte: captura de tela via https://mobsf.live/view_file/?file=jakhar/aseem/diva/LogActivity.java&md5=82114453a5daf3702bae45d79fa7b60b&type=studio&lines=56

Para resolver esse problema em um aplicativo móvel, é crucial implementar práticas de segurança robustas e seguir as diretrizes recomendadas de segurança de aplicativos móveis. Nesse contexto, destacam-se algumas medidas que podem ser adotadas para garantir o armazenamento seguro de dados em um aplicativo móvel:

- **Criptografia de Dados**: Todos os dados sensíveis devem ser criptografados antes de serem armazenados localmente no dispositivo móvel. Isso ajudará a garantir que os dados sejam ilegíveis se forem comprometidos por um invasor.
- **Práticas de Desenvolvimento Seguro**: Seguir práticas de desenvolvimento seguro e conduzir testes de segurança regulares para identificar e corrigir quaisquer vulnerabilidades de segurança no aplicativo móvel.
- **Atualizações e Patches**: Manter o aplicativo atualizado e aplicar patches de segurança regularmente para garantir que quaisquer vulnerabilidades conhecidas sejam corrigidas.
- **Educação do Usuário**: Educar os usuários sobre a importância da segurança dos dados e incentivar práticas seguras, como o uso de senhas fortes e a ativação de autenticação em dois fatores, se aplicável.

4.1.3 Qualidade do código do cliente

Nessa detecção, cujos detalhes podem ser vistos na figura 6, conforme CWE-89 (2023), a injeção de SQL tornou-se um problema comum em sites baseados em banco de dados. A falha é facilmente detectada e explorada e, como tal, qualquer site ou pacote de produto com uma base mínima de usuários provavelmente estará sujeito a uma tentativa de ataque desse tipo. Essa falha depende do fato de que o SQL não faz distinção real entre os planos de controle e de dados.

Segundo o OWASP (2016), um invasor normalmente explorará vulnerabilidades nessa categoria, fornecendo informações cuidadosamente elaboradas à vítima. Essas entradas são passadas para o código que reside no dispositivo móvel no qual ocorre a exploração. Modalidades típicas de ataques exploram vazamentos de memória e buffer overflows.

3	App uses SQLite Database and execute raw SQL query. Untrusted user input in raw SQL queries can cause SQL Injection. Also sensitive information should be encrypted and written to the database.	warning	**CWE:** CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') **OWASP Top 10:** M7: Client Code Quality

Figura 6 - Falhas encontradas com tratamento de dados incorretos no aplicativo Diva

Fonte: captura de tela via https://mobsf.live/static_analyzer/82114453a5daf3702bae45d79fa7b60b/

A maioria das explorações que se enquadram nessa categoria resultam na execução de código estrangeiro ou na negação de serviço em terminais de servidores remotos (e não no próprio dispositivo móvel). No entanto, caso existam estouros/excessos de buffer no dispositivo móvel e a entrada possa ser derivada de uma parte externa, isso poderá ter um impacto técnico gravemente alto e deverá ser remediado.

Conforme mostra a figura a seguir, que contém o trecho do código, foi possível encontrar consultas de SQL que podem ser feitas de forma bruta através de dados inseridos pelo usuário permitindo que o invasor leia ou grave no banco de dados dependendo das permissões concedidas pelo servidor.

```
public void search(View view) {
    EditText srchtxt = (EditText) findViewById(R.id.ivilsearch);
    Cursor cr = null;
    try {
        cr = mDB.rawQuery("SELECT * FROM sqliuser WHERE user = '" + srchtxt.getText().toString() + "'", null);
        StringBuilder strb = new StringBuilder("");
        if ((cr != null) && (cr.getCount() > 0)) {
            cr.moveToFirst();

            do {
                strb.append("User: (" + cr.getString(0) + ") pass: (" + cr.getString(1) + ") Credit card: (" + cr.getString(2) + ")\n");
            } while (cr.moveToNext());
        }
        else {
            strb.append("User: (" + srchtxt.getText().toString() +") not found");
        }
        Toast.makeText(this, strb.toString(), Toast.LENGTH_SHORT).show();
    }
    catch(Exception e) {
        Log.d("Diva-sqli", "Error occurred while searching in database: " + e.getMessage());
    }
}
```

Figura 7 - Demonstração de trecho do código onde tem uma das vulnerabilidades

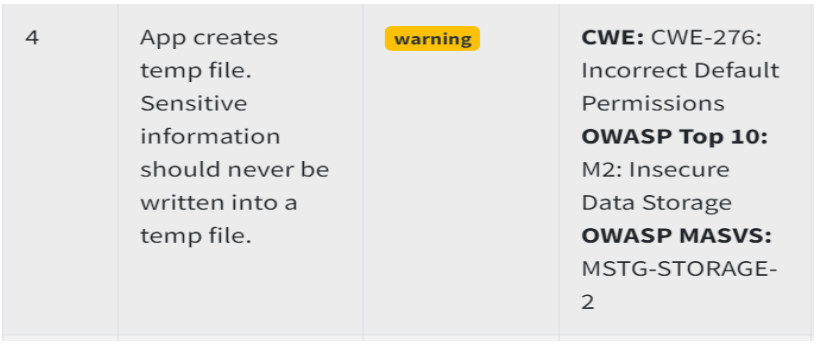
Fonte: captura de tela via https://mobsf.live/view_file/?file=jakhar/aseem/diva/SQLInjectionActivity.java & md5=82114453a5daf3702bae45d79fa7b60b&type=studio&lines=35,70

Como recomendação da própria OWASP (2016), há alguns procedimentos que podem ser adotados para prevenir esse tipo de vulnerabilidade:

- Manter padrões de codificação consistentes com os quais todos na organização concordem.
- Escrever um código fácil de ler e bem documentado.
- Ao usar buffers, sempre validar se os comprimentos de quaisquer dados de buffer recebidos não excederão o comprimento do buffer de destino.
- Por meio da automação, identificar buffer overflows e vazamentos de memória por meio do uso de ferramentas de análise estática de terceiros.
- Priorizar a solução de buffer overflows e vazamentos de memória em detrimento de outros problemas de "qualidade de código".

4.1.4 Armazenamento de dados inseguros

Segundo CWE-276 (2023), cujos detalhes podem ser vistos na figura a seguir, essa vulnerabilidade diz respeito a permissões incorretas em aplicativos que podem comprometer a segurança dos dados sensíveis dos usuários, tornando-os suscetíveis a violações de privacidade e, até mesmo, a ataques cibernéticos. Quando as permissões são definidas para que qualquer pessoa modifique esses arquivos, esses dados podem ser acessados por aplicativos mal-intencionados ou por usuários não autorizados que tenham acesso ao dispositivo.



4	App creates temp file. Sensitive information should never be written into a temp file.	warning	**CWE:** CWE-276: Incorrect Default Permissions **OWASP Top 10:** M2: Insecure Data Storage **OWASP MASVS:** MSTG-STORAGE-2

Figura 8 - Fraqueza identificada com armazenamento de dados Fonte: captura de tela via https://mobsf.live/static_analyzer/82114453a5daf3702bae45d79fa7b60b/

Tal vulnerabilidade pode trazer alguns problemas, tais como: risco de acesso não autorizado em que o invasor, caso consiga acesso ao dispositivo, pode explorar os arquivos temporários para extrair informações confidenciais, comprometendo a segurança do usuário, conforme mostra a figura a seguir.

```
try {
    File uinfo = File.createTempFile("uinfo", "tmp", ddir);
    uinfo.setReadable(true);
    uinfo.setWritable(true);
    FileWriter fw = new FileWriter(uinfo);
    fw.write(usr.getText().toString() + ":" + pwd.getText().toString() + "\n");
    fw.close();
    Toast.makeText(this, "3rd party credentials saved successfully!", Toast.LENGTH_SHORT).show();
    // Now you can read the temporary file where ever the credentials are required.
}
catch (Exception e) {
    Toast.makeText(this, "File error occurred", Toast.LENGTH_SHORT).show();
    Log.d("Diva", "File error: " + e.getMessage());
```

Figura 9 - Trecho do código em que é possível identificar a fraqueza

Fonte: captura de tela via

https://mobsf.live/view_file/?file=jakhar/aseem/diva/InsecureDataStorage3Activity.java&md5=82114453a5daf3702bae45d79fa7b60b&type=studio&lines=61

Uma possível solução para essa vulnerabilidade é evitar a gravação de informações sensíveis em arquivos temporários e optar por métodos mais seguros de manipulação de dados, como o armazenamento criptografado ou o uso de APIs seguras de armazenamento de dados.

4.2 Vulnerabilidades aplicativo *AndroGoat*

Nas próximas seções, serão apresentadas fraquezas detectadas pela ferramenta *MobSF* ao analisar o aplicativo *AndroGoat*. Ao total, a ferramenta detectou quatro fraquezas que serão detalhadas a seguir.

4.2.1 Privilégios desnecessários

Com o auxílio da ferramenta *MobSF* fazendo uma varredura no aplicativo *AndroGoat*, foi encontrada uma possível vulnerabilidade que, de acordo com a CWE-250 (2023), o produto executa uma operação com um nível de privilégio superior ao nível mínimo exigido, o que cria novos pontos fracos ou amplifica as consequências de outros pontos fracos, conforme mostra a figura a seguir que contém a possível vulnerabilidade.

2	This App may request root (Super User) privileges.	warning	**CWE:** CWE-250: Execution with Unnecessary Privileges **OWASP MASVS:** MSTG-RESILIENCE-1

Figura 10 - Fraqueza do aplicativo no qual há privilégios desnecessários Fonte: captura de tela via

https://mobsf.live/static_analyzer/82114453a5daf3702bae45d79fa7b60b/

Ainda consoante a CWE-250 (2023), pontos fracos podem ser expostos porque a execução com privilégios extras, como root ou Administrador, pode desativar as verificações normais de segurança realizadas pelo sistema operacional ou pelo ambiente circundante. Sendo assim, outras fraquezas pré-existentes podem transformar-se em vulnerabilidades de segurança se ocorrerem durante a operação com privilégios elevados, conforme pode ser visto no trecho do código da figura abaixo.

```
fun isRooted():Boolean{

 val file= arrayOf("/system/app/Superuser/Superuser.apk", "/system/app/Superuser.apk","/sbin/su", "/system/bin/su", "/system/xbin/su", "/data/local/xbin/su", "/data/local/bin/su", "/system/sd/xbin/su",
    "/system/bin/failsafe/su", "/data/local/su", "/su/bin/su", "re.robv.android.xposed.installer-1.apk","/data/app/eu.chainfire.supersu-1/base.apk");
  var result:Boolean=false
     for(files in file) {
         val f=File(files)
         result=f.exists()
         if (result) {
             break
         }
     }
     return result
 }
```

Figura 11 - Trecho do código expondo a fraqueza
Fonte: captura de tela via :
https://mobsf.live/view_file/?file=owasp/sat/agoat/RootDetectionActivity.kt&md5=808c2dd15465146f354da09fd682d6d9&type=studio&lines=35,35

Para evitar ser vítima dessa possível vulnerabilidade, é necessário ter um controle mais rigoroso quanto aos privilégios que cada usuário tem dentro do aplicativo, evitando dar permissões desnecessárias para os usuários.

```
"/system/bin/failsafe/", "/data/local/", "/su/bin/",
"re.robv.android.xposed.installer-1.apk","/data/app/eu.chainfire.supersu-1/base.apk");
```

4.2.2 Engenharia Reversa

Com o auxílio do *MobSF* analisando o aplicativo *AndroGoat*, foi constatado que ele pode estar suscetível a uma vulnerabilidade do tipo de engenharia reversa, conforme detalhes mostrado nas figuras abaixo.

9	Files may contain hardcoded sensitive information like usernames, passwords, keys etc.	warning	**CWE:** CWE-312: Cleartext Storage of Sensitive Information **OWASP Top 10:** M9: Reverse Engineering **OWASP MASVS:** MSTG-STORAGE-14

Figura 12 - Fraqueza encontrada em engenharia reversa Fonte: captura de tela via
https://mobsf.live/static_analyzer/82114453a5daf3702bae45d79fa7b60b/

Segundo a OWASP (2016), todo código móvel é suscetível à engenharia reversa. Alguns aplicativos são mais do que outros. Assim sendo, códigos escritos em linguagens/frameworks que permitem a introspecção dinâmica em tempo de execução

(*Java*, *.NET*, *Objective C*, *Swift*) estão particularmente em risco de sofrerem essa ação.

Essa vulnerabilidade pode causar vários impactos, como um de tipo técnico ou como de tipo de negócio. Conforme a OWASP (2016), no impacto primeiro, um invasor pode explorar a engenharia reversa para alcançar qualquer um dos seguintes:

- Revelar informações sobre servidores back-end.
- Revelar constantes criptográficas e cifras.
- Roubar propriedade intelectual.
- Realizar ataques contra sistemas back-end.
- Obter inteligência necessária para realizar modificações subsequentes no código.

Já um impacto que essa vulnerabilidade pode trazer com o aplicativo em base de produção de uma empresa OWASP (2016) varia bastante. Há a possibilidade de ser:

- Roubo de Propriedade Intelectual
- Danos Reputacionais
- Roubo de identidade
- Comprometimento de sistemas *backend*.

```
SQLibutton.setOnClickListener{
    var qry:String="SELECT * FROM users WHERE username='"+username.text.toString()+"'";
    try {
        this.mDB = openOrCreateDatabase("aGoat", 0, null)
        val QryResult = this.mDB!!.rawQuery(qry, null)
        val strb = StringBuilder("")
        if (QryResult == null || QryResult.count <= 0) {
            strb.append("User: ("+username.text.toString()+") not found")
        } else {
            QryResult.moveToFirst()
            do {
                strb.append("Username: (" + QryResult.getString(1) + ") password: (" + QryResult.getString(2) + ")\n")
            } while (QryResult.moveToNext())
        }
        Toast.makeText(this, strb.toString(), Toast.LENGTH_LONG).show();
        Log.e("Error:",strb.toString());
    } catch (e: Exception) {
        Log.d("Error", "Error occurred when inserting data into database: " + e.message)
        Toast.makeText(applicationContext, "Data not saved: Error occured - " + e.message, Toast.LENGTH_LONG).show();
    }
    Log.v("Query",qry);
```

Figura 13 -

Fonte: captura de tela via https://mobsf.live/view_file/?file=owasp/sat/agoat/SQLinjectionActivity.kt&md5=808c2dd15465146f354da09fd682d6d9&type=studio&lines=24

Para evitar uma engenharia reversa eficaz, você deve usar uma ferramenta de ofuscação. Existem muitos ofuscadores de nível comercial no mercado, como, por exemplo, o *SmartAssembly*. Entretanto, antiteticamente, também existem muitos desofuscadores no mercado. Então, para medir a eficácia de qualquer ferramenta de ofuscação escolhida, pode-se desofuscar o código usando ferramentas como *IDA Pro*.

4.2.3 Criptografia Insuficiente

Outra fraqueza encontrada no aplicativo *AndroGoat* foi: conter pouco ou nenhuma criptografia presente. Segundo a CWE-327 (2023), essa vulnerabilidade é ainda mais desafiadora de gerenciar com a implantação de algoritmos criptográficos em hardware, em oposição à implementação de software. Primeiro, se uma falha for descoberta na criptografia implementada por hardware, a falha não poderá ser corrigida na maioria dos casos sem um recall do produto, porque o hardware não é facilmente substituível como o software. Em segundo lugar, como se espera que o produto de hardware funcione durante anos, o poder computacional do adversário só aumentará com o tempo.

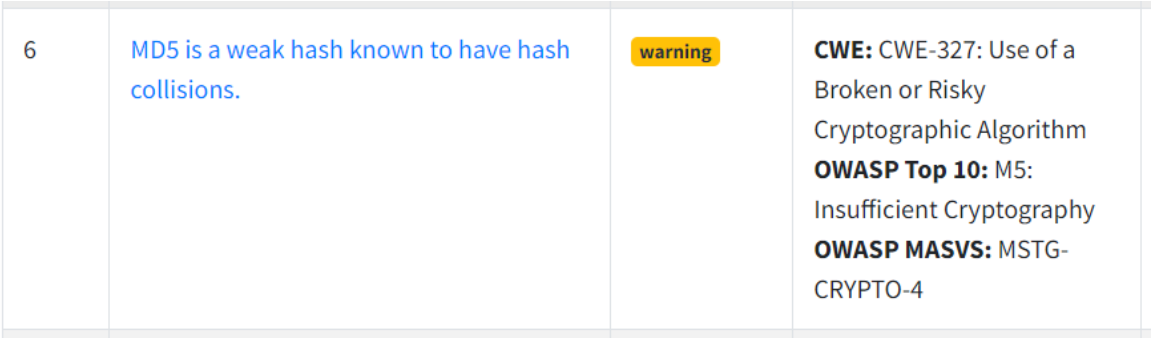


Figura 14 - Fraqueza encontrada com ausência ou pouca criptografia

Fonte: captura de tela via https://mobsf.live/static_analyzer/82114453a5daf3702bae45d79fa7b60b/

Segundo a CWE-327(2023), essa fraqueza pode causar uma série de impactos técnicos, em que o armazenamento inseguro de dados pode resultar em perda de dados para um usuário, na melhor das hipóteses. Na pior delas, para muitos usuários. Segue abaixo uma relação de dados valiosos que podem ser armazenados:

- Nomes de usuário
- Tokens de autenticação
- Senhas
- Cookies
- Dados de localização
- UDID/EMEI, nome do dispositivo, nome da conexão de rede
- Informações pessoais: data de nascimento, endereço, redes sociais, dados de cartão de crédito
- Dados de aplicativos: logs de aplicativos armazenados, por exemplo, para um logcat ADB de aplicativos Android; informações de depuração; mensagens de aplicativos em cache; históricos de transações

Na figura abaixo, encontra-se destacado o código relativo à fraqueza acima descrita:

```
fun hashPIN(pinValue: String):String{
    val md= MessageDigest.getInstance("MD5").digest(pinValue.toByteArray()).joinToString("") { "%02x".format(it) }
    return md

}
```

Figura 15 - Trecho do código que contém a fraqueza Fonte: captura de tela via https://mobsf.live/view_file/?file=owasp/sat/agoat/AccessControlIssue1Activity.kt&md5=808c2dd15465146f354da09fd682d6d9&type=studio&lines=79

Para não ser vítima desse tipo de vulnerabilidade, existem alguns requisitos a serem cumpridos, sendo eles:

- **Uso de Algoritmos Criptográficos Fortes**: certifique-se de que algoritmos de criptografia robustos e atualizados estejam sendo utilizados e, acima de tudo, evite o uso de algoritmos desatualizados, fracos ou vulneráveis.
- **Implementação de TLS/SSL Adequada:** para proteger a comunicação entre o aplicativo móvel e o servidor, é essencial implementar o *Transport Layer Security* (TLS) ou o *Secure Sockets Layer* (SSL) de maneira acordante. Certifique-se de que as configurações e certificados estejam corretamente configurados.
- **Gerenciamento Adequado de Chaves:** mantenha práticas seguras de gerenciamento de chaves, como o armazenamento seguro de chaves criptográficas e a rotação regular de chaves.
- **Condução de Testes de Segurança:** realize auditorias de segurança regulares, incluindo testes de penetração para identificar e corrigir possíveis vulnerabilidades de criptografia no aplicativo.

Uma possível correção do código, na figura abaixo:

```
hashPIN    divertido ( pinValue: String ) : String {
        val md=MessageDigest. getInstance ( "SHA512" ) . digerir ( pinValue.toByteArray (
)) . joinToString ( "" ) { "%02x" . formato ( isto ) }
       retornar md
   }
```

5. Conclusão

Atualmente, vulnerabilidades são um problema sério devido ao fato de que periodicamente são reportados problemas dessa natureza por empresas especializadas na área.

Paralelamente, há também uma preocupação da comunidade na área em trazer esses problemas, seja a OWASP com um ranking com as vulnerabilidades que estão em alta, bem como ferramentas que ajudam a detectar esse tipo de adversidade.

Além da comunidade OWASP, existe também o CWE, que possui catalogadas várias fraquezas desse tipo, contendo uma explicação de como funciona cada uma delas, assim como exemplos de ataques para cada uma delas e uma possível dica de como evitar pontos de vulnerabilidade.

Referências

ANDROGOAT. 2023. Disponível em: <https://github.com/satishpatnayak/AndroGoat>. Acesso em: 25 out. 2023.

CONVERGÊNCIA DIGITAL. **Mundo tem 8,4 bilhões de celulares ativos**. 2023. Disponível em: <https://www.convergenciadigital.com.br/Internet-Movel/Mundo-tem-8%2C4-bilhoes-de

-celulares-ativos-62713.html?UserActiveTemplate=mobile>. Acesso em: 15 de setembro de 2023.

CWE. 2023. About CWE. Disponível em: <https://cwe.mitre.org/about/index.html>. Acesso em: 01 set. 2023.

CWE-89. 2023. Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection'). Disponível em: <https://cwe.mitre.org/data/definitions/89>. Acesso em: 12 out. 2023.

CWE-250. 2023. Execution with Unnecessary Privileges. Disponível em: <https://cwe.mitre.org/data/definitions/250.html>. Acesso em: 30 out. 2023.

CWE-276. 2023. Incorrect Default Permissions. Disponível em: <https://cwe.mitre.org/data/definitions/276.html>. Acesso em: 20 out. 2023.

CWE-312. 2023. Cleartext Storage of Sensitive Information. Disponível em: <https://cwe.mitre.org/data/definitions/312.html>. Acesso em: 02 nov. 2023.

CWE-327. 2023. Use of a Broken or Risky Cryptographic Algorithm. Disponível em: < https://cwe.mitre.org/data/definitions/327.html>. Acesso em: 05 nov. 2023.

CWE-532. 2023. Insertion of Sensitive Information into Log File. Disponível em: < https://cwe.mitre.org/data/definitions/532.html>. Acesso em: 09 out. 2023.

CWE-919. 2023. Weaknesses in Mobile Applications. Disponível em: <https://cwe.mitre.org/data/definitions/919.html>. Acesso em: 09 out. 2023.

DIVA<https://github.com/payatu/diva-android>. Acesso em: 01 set. 2023.

IDA PRO. 2023. Página inicial. Disponível em: <https://hex-rays.com/ida-pro/>. Acesso em: 10 nov. 2023.

KOMAL. 2020. Hunting for Bugs in Damn Insecure and Vulnerable App. Disponível em: <https://infosecwriteups.com/hunting-for-bugs-in-android-app-fe9158a556d3>. Acesso em: 06 out. 2023.

MOBSF.thoughtworks, 2023. Ferramentas. Disponível em:<https://www.thoughtworks.com/pt-br/radar/tools/mobsf#:~:text=MobSF%20%C 3% A9%20uma%20ferramenta%20de,relat%C3%B3rios%20detalhados%20sobre%20as %20 vulnerabilidades.>. Acesso em: 01 set. 2023.

MOBSF. 2023. Analisador estático. Disponível em: <https://mobsf.live/static_analyzer/82114453a5daf3702bae45d79fa7b60b/https://mobsf.live/view_file/?file=jakhar/aseem/diva/ >. Acesso em: 28 out. 2023.

MOBSF. 2023. Controle de acesso. Disponível em: <https://mobsf.live/view_file/?file=jakhar/aseem/diva/AccessControl3NotesActivity.java&md5=82114453a5daf3702bae45d79fa7b60b&type=studio&lines=72,73>. Acesso em: 29 out.2023.

MOBSF. 2023. *Log activity*. Disponível em: <https://mobsf.live/view_file/?file=jakhar/aseem/diva/LogActivity.java&md5=82114453a5daf3702bae45d79fa7b60b&type=studio&lines=56>. Acesso em: 30 out.2023.

MOBSF. 2023. Atividade de injeção SQL. Disponível em: <https://mobsf.live/view_file/?file=jakhar/aseem/diva/SQLInjectionActivity.java&m

d5=82114453a5daf3702bae45d79fa7b60b&type=studio&lines=35,70>. Acesso em: 31 out. 2023.

MOBSF. 2023. Atividade de armazenamento inseguro de dados. Disponível em: <https://mobsf.live/view_file/?file=jakhar/aseem/diva/InsecureDataStorage3Activity.java&md5=82114453a5daf3702bae45d79fa7b60b&type=studio&lines=61>. Acesso em: 01 nov. 2023.

MOBSF. 2023. Atividade de detecção raiz. Disponível em: <https://mobsf.live/view_file/?file=owasp/sat/agoat/RootDetectionActivity.kt&md5=808c2dd15465146f354da09fd682d6d9&type=studio&lines=35,35>. Acesso em: 02 nov.2023.

MOBSF. 2023. Atividade de injeção SQL. Disponível em: <https://mobsf.live/view_file/?file=owasp/sat/agoat/SQLinjectionActivity.kt&md5=808c2dd15465146f354da09fd682d6d9&type=studio&lines=24>. Acesso em: 28 nov. 2023.

MOBSF. 2023. Problema de controle de acesso. Disponível em: <https://mobsf.live/view_file/?file=owasp/sat/agoat/AccessControlIssue1Activity.kt&md5=808c2dd15465146f354da09fd682d6d9&type=studio&lines=79>. Acesso em: 29 nov. 2023

OWASP. 2023. Página inicial. Disponível em: <https://owasp.org/> Acesso em: 01 set. 2023.

OWASP MASVS. 2023. Segurança. Disponível em: <https://mobile-security.gitbook.io/masvs/security-requirements/0x07-v2-data_storag e_a nd_privacy_requirements>. Acesso em: 09 out. 2023.

OWASP M1. 2023. Top 10 riscos. Disponível em: <https://owasp.org/www-project-mobile-top-10/2023-risks/m1-improper-credential-u sag e.html>. Acesso em: 09 out. 2023.

PELTIER, T. R. **Information security risk analysis**. 3.ed. Auerbach Publication, 2010. 331p. v. 1

SILVA, Leonardo Oliveira. **Testes de segurança em aplicações Android baseados na metodologia OWASP**. 2022. 52 f. Monografia (Graduação em Ciência da Computação) - Universidade Federal do Ceará, Fortaleza, 2022.
SmartAssembly. 2023. Produtos. Disponível em: <https://www.red-gate.com/products/smartassembly/>. Acesso em: 10 nov. 2023.