

Resultados das análises das aplicações

EXECUÇÃO DOS TESTES

Nesta seção serão abordados as 10 vulnerabilidades listadas no OWASP *Mobile Top 10*, mostrando as ferramentas estáticas e/ou dinâmicas utilizadas para atestar a vulnerabilidade em um dado aplicativo vulnerável (*sandbox*) e fazendo menção ao requisito do MASVS, o qual a vulnerabilidade está afetando negativamente. Por fim, faz menção ao procedimento de teste apropriado descrito no MASTG para identificação da vulnerabilidade, além de sugestões de como corrigir o caso de uso e outras recomendações.

Ao todo, foram utilizadas 5 aplicações, no formato APK, para a realização dos experimentos, sendo 3 referenciadas no próprio MASTG para realização e práticas dos testes (Diva, *AndroGoat* e *InsecureBankv2*) e 2 advindas de pesquisa externa (*InsecureShop* e *VulnApp*).

O Diva¹ (*Damn insecure and vulnerable App*) é uma aplicação que foi projetada para ser insegura com o objetivo de ensinar falhas oriundas de codificação ruim ou insegura. Recebeu atualizações em 2016 e contém 13 desafios diferentes.

O *AndroGoat*² é um aplicativo inseguro que foi desenvolvido em *Kotlin* e contém mais de 20 vulnerabilidades para serem testadas.

*InsecureBankv2*³ é uma atualização de um projeto mais antigo, o *InsecureBank*, o qual foi projetado para ser explorado por apreciadores de segurança de aplicações, podendo encontrar diversas vulnerabilidades nele. O *backend* foi projetado em *Python*.

*InsecureShop*⁴ é um aplicativo projetado em *Kotlin* especificamente para ser vulnerável, contendo cerca de 20 vulnerabilidades, enquanto o *VulnApp* é um aplicativo desenvolvido especificamente para explorar a adulteração de código.

Uso inadequado da plataforma

Utilizando a ferramenta MobSF para analisar o aplicativo Diva, Figura 1, e observando o arquivo Manifesto, Figura 2, foi detectada a presença de um *intent-filter*, que reúne as *intents* implícitas a serem utilizadas, na componente *Activity* “*jakhar.aseem.diva.APICredsActivity*”, à qual chama uma das telas da aplicação, tornando-a explicitamente exportável e fazendo com que outra aplicação instalada no mesmo dispositivo

¹ Disponível em: <https://github.com/payatu/diva-android>

² Disponível em: <https://github.com/satishpatnayak/AndroGoat>

³ Disponível em: <https://github.com/dineshshetty/Android-InsecureBankv2>

⁴ Disponível em: <https://github.com/hax0rgb/InsecureShop>

possa acessar, além de também poder capturar, as possíveis informações contidas na componente sem ter que executar de fato a aplicação.

A detecção da ferramenta e o trecho do código onde ocorre o problema podem ser vistos nas Figuras 1 e 2, respectivamente, a seguir:

Figura 1 - Detecção da ferramenta MobSF da presença de uma *Activity* exportada publicamente na aplicação Diva

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.
---	---	---------	---

Fonte: elaborado pelo autor.

Figura 2 - Trecho do código onde mostra a presença perigosa 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>
```

Fonte: elaborado pelo autor.

Isso afeta o requisito de segurança do MASVS denominado “MSTG-PLATFORM-4: O aplicativo não exporta funcionalidades sensíveis através do IPC, a menos que esses mecanismos estejam devidamente protegidos” e o teste do MASTG para averiguar esse requisito é denominado “Teste para exposição de funcionalidade sensível por meio de IPC (MSTG-PLATFORM-4)”⁵. Se a lógica da aplicação indicar que esse componente só deva ser executado pelo próprio aplicativo, não se usa *intent-filter*, e sim define o atributo “*exported*” como “*false*” para a componente e utiliza-se *intents* explícitas.

Armazenamento de dados inseguro

Primeiramente, utilizando a ferramenta MobSF para analisar o aplicativo Diva, foi detectado que os dados movidos na aplicação eram registrados no *log* do dispositivo, mostrado na Figura 3, e o trecho de código onde acontece o problema é exibido na Figura 4. O problema acontece se dados confidenciais estiverem sendo registrados em texto plano, de

⁵ Teste do MASTG disponível em: <https://mas.owasp.org/MASTG/Android/0x05h-Testing-Platform-Interaction/#testing-for-sensitive-functionality-exposure-through-ipc-mstg-platform-4>

modo que alguém que esteja monitorando o *log* possa identificar esses dados. Na Figura 3, também, é visto que a análise da ferramenta citou uma referência a *Common Weakness Enumeration* (CWE), que é uma lista de tipos de vulnerabilidades em software e hardware criada e gerida pela MITRE, organização norte-americana que gerencia centros de pesquisa e desenvolvimento financiados pelo governo dos EUA. CWE-532⁶ remete a “Inserção de informações confidenciais no arquivo de *log*”. As figuras 4 e 5 são exibidas a seguir:

Figura 3 - Detecção da ferramenta MobSF que as informações são registradas nas mensagens de *log* na aplicação Diva

NO ↑↓	ISSUE ↑↓	SEVERITY ↑↓	STANDARDS ↑↓
1	The App logs information. Sensitive information should never be logged.	info	CWE: CWE-532: Insertion of Sensitive Information into Log File OWASP MASVS: MSTG-STORAGE-3

Fonte: elaborado pelo autor.

Figura 4 - Trecho de código que mostra que a informação está sendo registrada em mensagens de *log* na aplicação Diva

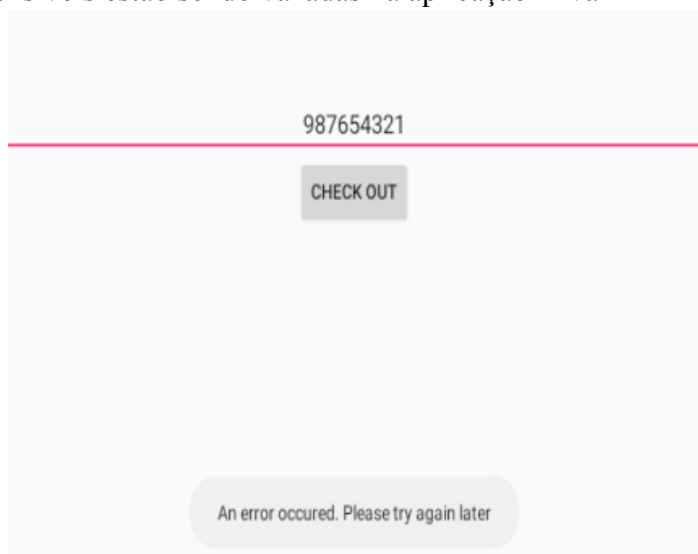
```
public void checkout(View view) {
    EditText cotxt = (EditText) findViewById(R.id.ccText);
    try {
        processCC(cotxt.getText().toString());
    } catch (RuntimeException e) {
        Log.e("diva-log", "Error while processing transaction with credit card: " + cotxt.getText().toString());
        Toast.makeText(this, "An error occurred. Please try again later", 0).show();
    }
}
```

Fonte: elaborado pelo autor.

Fazendo uso da aplicação na funcionalidade de cadastro de um cartão de crédito, Figura 5, e com a utilização da ferramenta adb e o seu utilitário *logcat* - monitoramento das mensagens de *log* do dispositivo -, é possível verificar que dados confidenciais estavam aparecendo em texto plano entre as mensagens de *log*, Figura 6. Nela, é possível ver que o dado inserido está contido na mensagem de erro gerada, o que é inapropriado. As Figuras 5 e 6 são mostradas a seguir:

⁶ CWE-532: <https://cwe.mitre.org/data/definitions/532.html>

Figura 5 - Visualização da utilização da aplicação em que é detectado que informações sensíveis estão sendo vazadas na aplicação Diva



Fonte: elaborado pelo autor.

Figura 6 - Visualização da mensagem de *log* que representa um erro na aplicação em que é exibida a informação digitada na figura 5

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

Fonte: elaborado pelo autor.

Esse problema afeta o requisito de segurança 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)”⁷. Considere bem o que deve ser passado nas mensagens de *log* e não registre informações confidenciais 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.

Comunicação insegura

Utilizando a ferramenta MobSF para analisar o aplicativo *InsecureShop* (ver Figura 7), foi detectado que uma *WebView*, que faz parte da *Activity* “*WebViewActivity*”, foi

⁷ Teste do MASTG disponível em: <https://mas.owasp.org/MASTG/Android/0x05d-Testing-Data-Storage/#testing-logs-for-sensitive-data-mstg-storage-3>

implementada com a falta de verificação do certificado SSL, sem verificar erros e a validade do mesmo. Na Figura 8, é possível ver o trecho do código onde a variável que representa o possível erro, “*SslError error*”, e prossegue com a solicitação para o caso de o certificado ser válido, indicada pela chamada de função “*handler.proceed()*”. Isso pode acarretar em aceitar comunicações com certificados defasados e o usuário ser 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. Pela Figura 7, é possível verificar que a ferramenta indica com o maior grau de risco (*high*) na métrica da ferramenta para a vulnerabilidade encontrada. Além disso, mais uma vez é citada uma referência a CWE, remetendo à vulnerabilidade “CWE-295⁸: Validação de certificado imprópria”, ou seja, que o software não valida ou valida incorretamente um certificado.

Figura 7 - Detecção da ferramenta MobSF da falta de verificação de erros de certificados SSL na aplicação *InsecureShop*

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
---	--	------	--

Fonte: elaborado pelo autor.

Figura 8 - Trecho do código onde está a 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();
        }
    }
}
```

Fonte: elaborado pelo autor.

Pode-se ainda verificar esse problema na aplicação *InsecureShop* utilizando uma análise dinâmica com as ferramentas *Burp Suite* e *adb*.

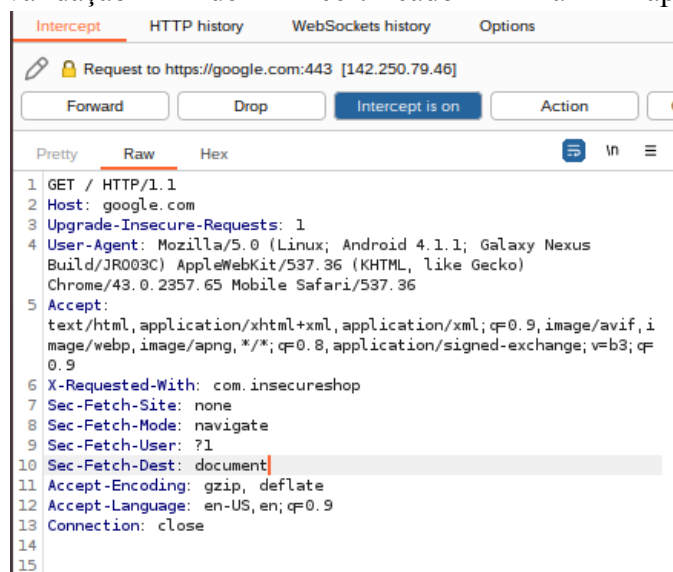
⁸ CWE-295: <https://cwe.mitre.org/data/definitions/295.html>

Primeiramente, o *Burp Suite* foi utilizado para interceptar a comunicação que utiliza o protocolo HTTP. Nesse caso em específico, diferente da recomendação de instalar no dispositivo um certificado confiável do *Burp Suite* para interceptar comunicações com HTTPS, será utilizado um dispositivo sem certificado instalado para averiguar a falta de validação do certificado na aplicação. Normalmente, se o certificado do *Burp Suite* não estiver instalado no dispositivo, a ferramenta não conseguirá interceptar tráfego HTTPS das aplicações, pois elas, corretamente, estão fazendo a verificação da utilização de um certificado. Se o certificado não estiver instalado e mesmo assim for possível observar comunicação HTTPS, indica que a aplicação em questão está com uma falha nessa verificação, e é o que será testado na *InsecureShop*.

Com o adb, foi utilizado o comando “*shell*” para se ter acesso ao dispositivo em que a aplicação está instalada e com o acesso, a *WebView* da aplicação que contém a vulnerabilidade é invocada por linha de comando, a partir da utilização do comando “*shell*” citado, para acessar a url “*google.com*”, que é um host que utiliza HTTPS para comunicação.

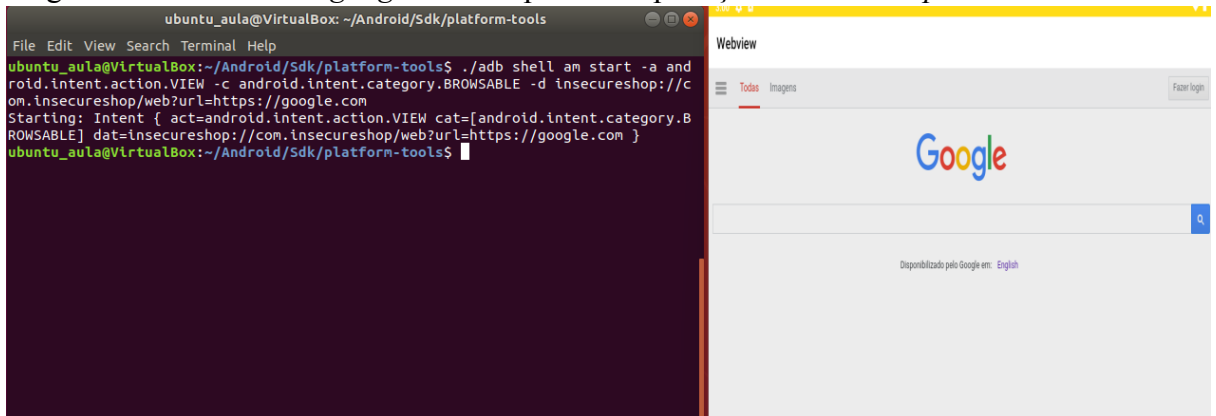
Esse procedimento é resumido nas imagens a seguir, em que a Figura 9 mostra uma comunicação interceptada pelo *Burp Suite* que utiliza HTTPS invocada a partir do aplicativo *InsecureShop*, indicado na linha 6, e que é acessada por meio da linha de comando, por meio do adb, que invoca a *WebView* e não pela aplicação diretamente acessada pelo usuário, como mostra a Figura 10. Por ela, é vista a invocação da *WebView* e a aplicação sendo executada acessando a url indicada.

Figura 9 - Visualização do *Burp Suite* que exemplifica o problema de validação de certificado na aplicação *InsecureShop*



Fonte: elaborado pelo autor.

Figura 10 - Acessando “google.com” a partir da aplicação *InsecureShop*



Fonte: elaborado pelo autor.

Esse problema afeta o requisito de segurança 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)”⁹. 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 assinado¹⁰. Uma maneira simples é utilizar a interface *TrustManager*, que contém os meios necessários para fazer as verificações de validade de um certificado.

Autenticação insegura

Utilizando a ferramenta MobSF para analisar o aplicativo *InsecureBankv2* e observando o arquivo Manifesto, foi observado que 4 *Activities* relacionadas com o procedimento de *login* estavam sendo exportadas explicitamente (“*PostLogin*”, “*DoTransfer*”, “*ViewStatement*” e “*ChangePassword*”), por meio do atributo “*exported*” com valor “*true*”. A Figura 11 a seguir mostra o ocorrido:

⁹ Teste do MASTG disponível em:

<https://mas.owasp.org/MASTG/Android/0x05g-Testing-Network-Communication/#testing-endpoint-identify-verification-mstg-network-3>

¹⁰ Segurança com HTTPS e SSL: <https://developer.android.com/training/articles/security-ssl>

Figura 11 - Trecho do código onde mostra as *Activities* exportadas na aplicação *InsecureBankv2*

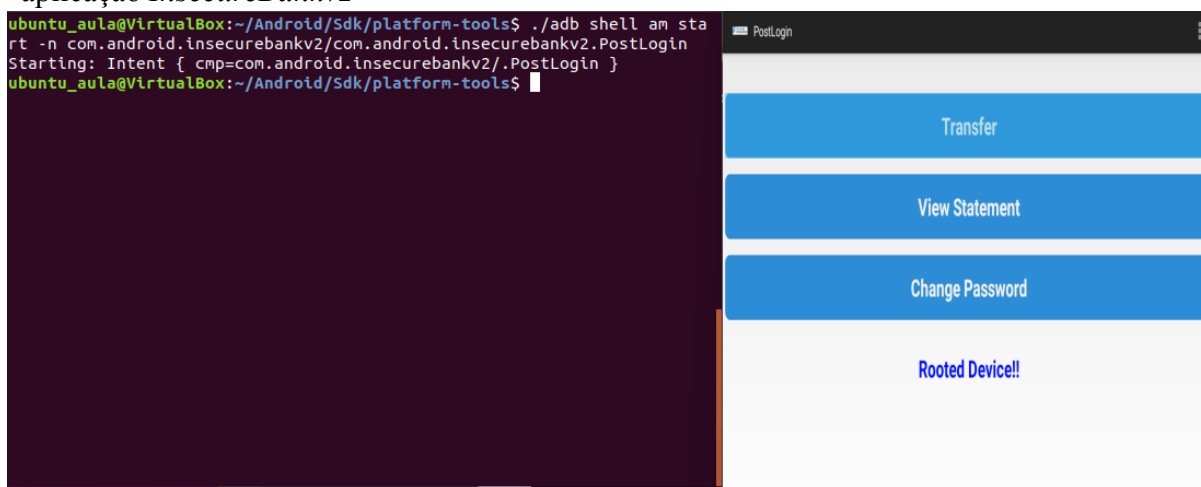
```
<activity android:exported="true" android:label="@string/title_activity_post_login"
android:name="com.android.insecurebankv2.PostLogin"/>
<activity android:label="@string/title_activity_wrong_login"
android:name="com.android.insecurebankv2.WrongLogin"/>
<activity android:exported="true" android:label="@string/title_activity_do_transfer"
android:name="com.android.insecurebankv2.DoTransfer"/>
<activity android:exported="true" android:label="@string/title_activity_view_statement"
android:name="com.android.insecurebankv2.ViewStatement"/>
<provider android:authorities="com.android.insecurebankv2.TrackUserContentProvider" android:exported="true"
android:name="com.android.insecurebankv2.TrackUserContentProvider"/>
<receiver android:exported="true" android:name="com.android.insecurebankv2.MyBroadCastReceiver">
    <intent-filter>
        <action android:name="theBroadcast"/>
    </intent-filter>
</receiver>
<activity android:exported="true" android:label="@string/title_activity_change_password"
android:name="com.android.insecurebankv2.ChangePassword"/>
```

Fonte: elaborado pelo autor.

Perceba que é mais um problema envolvendo *Activities*, mas em uma abordagem diferente. Assume-se, primariamente, que a *Activity* “*PostLogin*” está relacionada com a exibição de uma tela específica que só deveria ser acessada após o *login* feito com sucesso. Contudo, com a utilização da ferramenta *adb* e seu comando “*shell*”, a *Activity* é invocada com sucesso pela linha de comando, já que ela é exportável. Com isso, a tela de dados e funções do usuário autenticado, que só seria apresentada após o *login* efetuado com sucesso, é acessada, mostrando que a autenticação foi completamente ignorada.

Isso pode ter um impacto bem considerável se tratando de informações importantes, como os dados do usuário. O procedimento descrito é visto na Figura 12 a seguir:

Figura 12 - Resultado no aplicativo ignorando o procedimento de *login* na aplicação *InsecureBankv2*



Fonte: elaborado pelo autor.

O problema afeta o requisito de segurança 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)”¹¹. Como recomendações, atentar-se aos componentes exportados da aplicação, não deixando nenhum que seja referente a eventos privados na lógica da aplicação, como procedimento de autenticação, de forma pública. Além disso, as autenticações devem ser feitas pelo lado do servidor e deve-se evitar guardar e compartilhar dados de autenticação localmente no dispositivo.

Criptografia insuficiente

Utilizando a ferramenta MobSF para analisar o aplicativo *Androgoat*, foi detectado que está sendo utilizada uma função hash criptográfica MD5, que já é considerada problemática pelo risco de colisões entre os códigos *hash*.

A Figura 13 apresenta o *print* da detecção da ferramenta, enquanto a Figura 14 mostra o trecho do código onde ocorre o problema, mais especificamente na linha 112. Mais uma vez a CWE é referenciada, mais especificamente a “CWE-327¹²: Uso de um Algoritmo Criptográfico Quebrado ou Arriscado”, referente a algoritmos defasados e que não representam tanta segurança atualmente.

Figura 13 - Detecção da ferramenta MobSF da utilização de função criptográfica problemática do aplicativo *Androgoat*

4	MD5 is a weak hash known to have hash collisions.	warning	CWE: CWE-327: Use of a Broken or Risky Cryptographic Algorithm OWASP Top 10: M5: Insufficient Cryptography OWASP MASVS: MSTG-CRYPTO-4
---	---	---------	--

Fonte: elaborado pelo autor.

¹¹ Teste do MASTG disponível em: <https://mas.owasp.org/MASTG/Android/0x05d-Testing-Data-Storage/#testing-logs-for-sensitive-data-mstg-storage-3>

¹² CWE-327: <https://cwe.mitre.org/data/definitions/327.html>

Figura 14 - Trecho do código que mostra a utilização da função criptográfica problemática do aplicativo *Androgoat*

```
110.     public final String hashPIN(String pinValue) {
111.         Intrinsic.checkParameterNotNull(pinValue, "pinValue");
112.         MessageDigest messageDigest = MessageDigest.getInstance("MD5");
113.         byte[] bytes = pinValue.getBytes(Charsets.UTF_8);
114.         Intrinsic.checkExpressionValueNotNull(bytes, "(this as java.lang.String).getBytes(charset)");
115.         byte[] digest = messageDigest.digest(bytes);
116.         Intrinsic.checkExpressionValueNotNull(digest, "MessageDigest.getInstance(pinValue.toByteArray())");
117.         String md = ArraysKt.joinToString$default(digest, (CharSequence) "", (CharSequence) null, (CharSequence) nu
118.         return md;
119.     }
120. }
```

Fonte: elaborado pelo autor.

Esse problema está afetando o requisito de segurança denominado “MSTG-CRYPTO-4: O aplicativo não utiliza protocolos criptográficos ou algoritmos que são considerados amplamente obsoletos para uso em segurança”. O teste do MASTG para averiguar esse requisito é denominado “Testando a configuração de algoritmos padrão criptográficos (MSTG-CRYPTO-2, MSTG-CRYPTO-3 e MSTG-CRYPTO-4)”¹³. Esse problema pode ser solucionado modificando a função hash para SHA-2, que suporta tamanho de mensagens hash até 512 bits, enquanto a MD5 só suporta 128 bits. Ademais, se tratando de algoritmos criptográficos, é uma boa prática sempre buscar aqueles que sejam classificados ou publicados pela NIST¹⁴ mais recentemente e seguir os padrões da linguagem utilizada e da plataforma *Android*. Por exemplo, o módulo *Conscrypt*¹⁵ otimiza e melhora a segurança do dispositivo *Android*, utilizando código *Java* e fornecendo uma biblioteca nativa para desenvolvimento em plataforma *Android*, além de conter diversos algoritmos atualizados.

Autorização insegura

Utilizando a ferramenta MobSF para analisar o aplicativo *InsecureBankv2* e acessando o arquivo de código fonte referente a “*LoginActivity*”, observa-se o trecho de código que, aparentemente, tem uma funcionalidade de administrador oculta baseada no valor da string “*is_admin*”. Caso o valor seja “*no*”, o botão é ocultado da tela, como mostra a Figura 15 a seguir:

¹³ Teste do MASTG disponível em:

<https://mas.owasp.org/MASTG/Android/0x05e-Testing-Cryptography/#testing-the-configuration-of-cryptographic-standard-algorithms-mstg-crypto-2-mstg-crypto-3-and-mstg-crypto-4>

¹⁴ Algoritmos recomendados: <https://www.keylength.com/en/4/>

¹⁵ Mais sobre Conscrypt: <https://source.android.com/docs/core/architecture/modular-system/conscrypt>

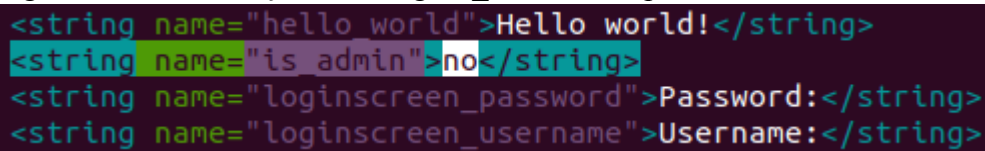
Figura 15 - Funcionalidade de administradores oculta do aplicativo *InsecureBankv2*

```
@Override // android.app.Activity
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_log_main);
    String mess = getResources().getString(R.string.is_admin);
    if (mess.equals("no")) {
        View button_CreateUser = findViewById(R.id.button_CreateUser);
        button_CreateUser.setVisibility(8);
    }
}
```

Fonte: elaborado pelo autor.

Utilizando a ferramenta apktool para decompilar apk, e acessando o arquivo “*strings.xml*”, localizado em “*/res/values/*”, onde ficam armazenados todos os recursos envolvendo *strings*, é possível localizar “*is_admin*” atribuída com o valor “*no*”, como mostra a Figura 16 a seguir:

Figura 16 - Visualização da string “*is_admin*” do aplicativo *InsecureBankv2*

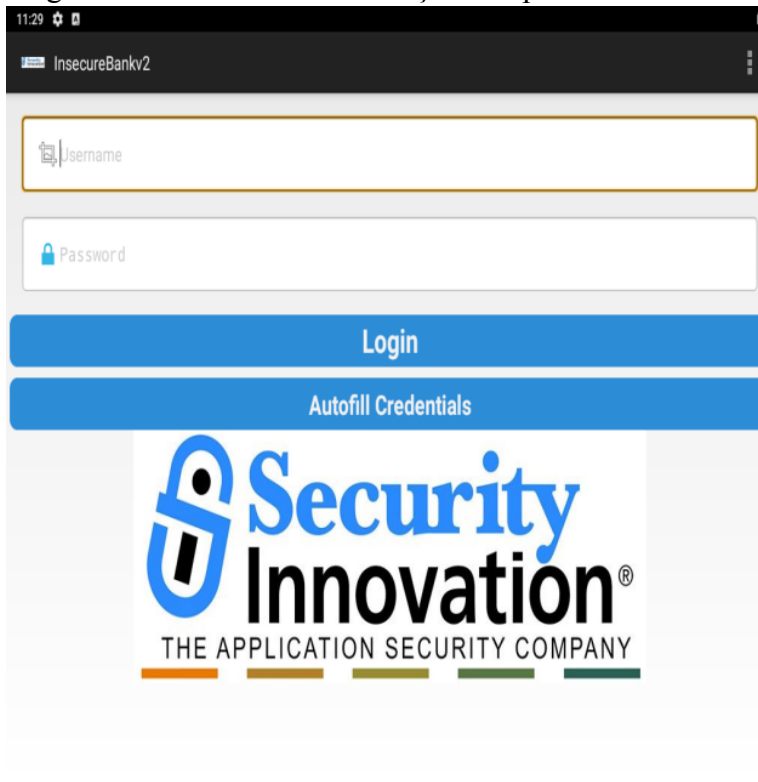


```
<string name="hello world">Hello world!</string>
<string name="is_admin">no</string>
<string name="loginscreen_password">Password:</string>
<string name="loginscreen_username">Username:</string>
```

Fonte: elaborado pelo autor.

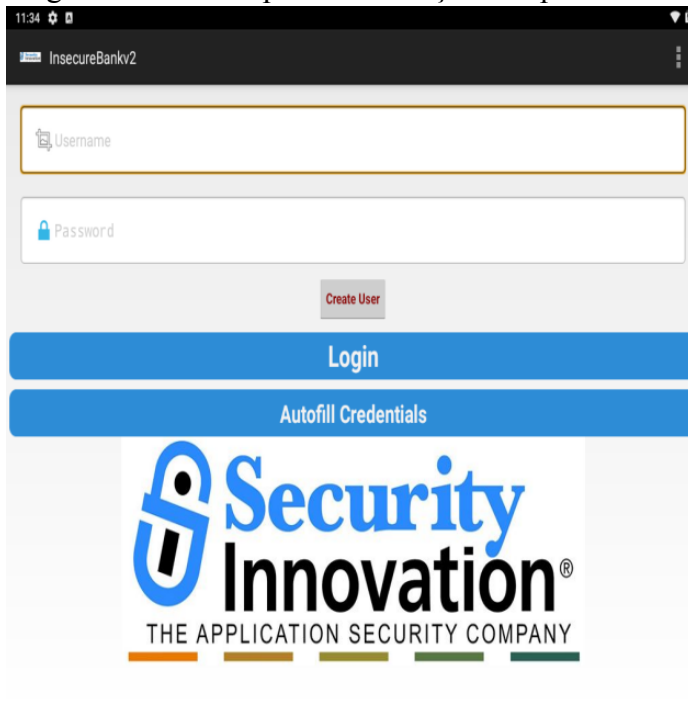
Trocando o valor para “*yes*”, compilando com o *apktool* o projeto da aplicação com essa alteração no arquivo e instalando novamente no dispositivo, tem-se o aparecimento do botão “*Create User*”, como mostram as figuras a seguir. A Figura 17 mostra a tela antes da alteração no código e a Figura 18 mostra após a alteração:

Figura 17 - Tela antes da alteração no aplicativo *InsecureBankv2*



Fonte: elaborado pelo autor.

Figura 18 - Tela depois da alteração no aplicativo *InsecureBankv2*



Fonte: elaborado pelo autor.

Esse problema afeta 2 requisitos de segurança do MASVS, respectivamente, denominados “MSTG-ARCH-1: Todos os componentes do aplicativo são identificados e reconhecidos como necessários” e “MSTG-RESILIENCE-3: O aplicativo detecta e responde para a manipulação de executáveis e dados críticos do próprio aplicativo”. O teste do MASTG para averiguar esse segundo requisito é denominado “Testando verificações de integridade de arquivo (MSTG-RESILIENCE-3)”¹⁶. Para o primeiro requisito, ainda não há um teste registrado no guia MASTG direcionado a ele.

É indicado que as permissões e funções que requerem um certo nível de autorização utilizem as informações para tal diretamente do sistema *backend*, como o servidor responsável pela autenticação dos usuários. É importante também verificar se todos os endpoints de API e funcionalidades necessárias estão disponíveis publicamente, além de que o aplicativo deve ser capaz de fazer verificação em tempo de execução para detectar que o código foi alterado a partir das informações que são conhecidas sobre a integridade da aplicação em tempo de compilação.

Qualidade do código do cliente

Utilizando a ferramenta MobSF para analisar o aplicativo Diva, foram detectadas que estão sendo feitas consultas SQL de forma bruta com os dados inseridos pelo usuário. Isso pode causar o problema de injeção SQL, que consiste em uma consulta SQL imprópria ao banco de dados a partir dos campos de entrada de dados da aplicação. A detecção da ferramenta é mostrada na Figura 19, enquanto o trecho do código onde ocorre o problema é indicado na Figura 20. A CWE é citada novamente, sendo a “CWE-89¹⁷: Neutralização imprópria de elementos especiais usados em um comando SQL ('SQL Injection')”, que remete ao tratamento incorreto ou inexistente dos dados de entrada, podendo deixar uma consulta SQL imprópria ser feita.

¹⁶ Teste do MASTG disponível em:

<https://mas.owasp.org/MASTG/Android/0x05j-Testing-Resiliency-Against-Reverse-Engineering/#testing-file-integrity-checks-mstg-resilience-3>

¹⁷ CWE-89: <https://cwe.mitre.org/data/definitions/89.html>

Figura 19 - Detecção da ferramenta MobSF do tratamento incorreto de dados de entrada do usuário do aplicativo Diva

4	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
---	--	---------	--

Fonte: elaborado pelo autor.

Figura 20 - Trecho de código onde mostra o tratamento incorreto de dados de entrada do usuário, mais especificamente na linha 35, do aplicativo Diva

```

32.     public void search(View view) {
33.         EditText srchtxt = (EditText) findViewById(R.id.ivilsearch);
34.         try {
35.             Cursor cr = this.mDB.rawQuery("SELECT * FROM sqliuser WHERE user = '" + srchtxt.getText().toString() + "'", null);
36.             StringBuilder strb = new StringBuilder("");
37.             if (cr == null || cr.getCount() <= 0) {
38.                 strb.append("User: (" + srchtxt.getText().toString() + ") not found");
39.             } else {
40.                 cr.moveToFirst();
41.                 do {
42.                     strb.append("User: (" + cr.getString(0) + ") pass: (" + cr.getString(1) + ") Credit card: (" + cr.getString(2) + ")\n");
43.                 } while (cr.moveToNext());
44.             }
45.             Toast.makeText(this, strb.toString(), 0).show();
46.         } catch (Exception e) {
47.             Log.d("Diva-sqli", "Error occurred while searching in database: " + e.getMessage());
48.         }
49.     }

```

Fonte: elaborado pelo autor.

Esse problema afeta o requisito de segurança denominado “MSTG-PLATFORM-2: Todas as entradas de fontes externas e do usuário são validadas e, se necessário, sanitizadas. Isso inclui dados recebidos através da UI, mecanismos de IPC como intenções, URLs personalizados e origens pela rede”. O teste do MASTG para averiguar esse requisito é denominado “Teste de falhas de injeção (MSTG-PLATFORM-2)”¹⁸. Para evitar que um usuário possa fazer uma consulta SQL inadequada e mal intencionada, deve-se, por exemplo, criar listas de entradas aceitáveis, fazer sempre a verificação do dado inserido e utilizar as bibliotecas mais indicadas pela documentação oficial da linguagem utilizada para seguir os padrões de segurança da linguagem.

¹⁸ Teste do MASTG disponível em: <https://mas.owasp.org/MASTG/Android/0x05h-Testing-Platform-Interaction/#testing-for-injection-flaws-mstg-platform-2>

Adulteração de código

Analisando a funcionalidade de acesso simplificada do aplicativo *VulnApp*, observa-se que é necessário digitar uma senha que, a princípio, o usuário não sabe. A interface referente a essa descrição com uma senha incorreta digitada pode ser vista na Figura 21 a seguir:

Figura 21 - Funcionalidade de acesso do aplicativo *VulnApp*



Fonte: elaborado pelo autor.

Utilizando a ferramenta apktool para decompilar a apk referente ao aplicativo *VulnApp*, e analisando o código *smali*, mais precisamente no arquivo localizado em “*smali/openssecurity/vulnapp/MainActivity\$1.smali*”, é possível perceber, com um certo esforço, que, no momento em que se clica no botão, uma comparação entre o dado inserido e a string “*vuln123*” é feita, dando indício de que essa é a senha correta. O trecho de código apresentado na Figura 22 mostra isso:

Figura 22 - Trecho de código onde mostra a comparação com a *string* “vuln123” no aplicativo *VulnApp*

```
move-result-object v1
invoke-virtual {v1}, Ljava/lang/Object;→toString()Ljava/lang/String;
move-result-object v1
const-string v2, "vuln123"
invoke-virtual {v1, v2}, Ljava/lang/String;→equals(Ljava/lang/Object;)Z
move-result v1
if-eqz v1, :cond_0
.line 28
new-instance v0, Landroid/content/Intent;
```

Fonte: elaborado pelo autor.

A função “*if-eqz*” está comparando 2 variáveis, a que contém o dado do usuário e a que contém a string da senha correta, e retorna verdadeiro se forem iguais e falso se forem diferentes. A função “*if-nez*” faz o oposto da função “*if-eqz*”: retorna falso se forem iguais e verdadeiro se forem diferentes.

Fazendo essa modificação no código *smali* (trocar a função “*if-eqz*” por “*if-nez*”), recompilando a estrutura do aplicativo com o *apktool*, e instalando o novo aplicativo, percebe-se que digitando qualquer coisa diferente de “vuln123”, o procedimento para acessar a aplicação é feito com sucesso, como mostra a Figura 23 a seguir:

Figura 23 - Funcionalidade de acesso após a manipulação no código do aplicativo *VulnApp*



Fonte: elaborado pelo autor.

Esse problema afeta o requisito de segurança do MASVS denominado “MSTG-RESILIENCE-3: O aplicativo detecta e responde para a manipulação de executáveis e dados críticos do próprio aplicativo”. O teste do MASTG para averiguar esse requisito é denominado “Testando verificações de integridade de arquivo (MSTG-RESILIENCE-3)”¹⁹. Para se proteger dessa vulnerabilidade é indicado que o aplicativo seja capaz de fazer verificação em tempo de execução para detectar que o código foi alterado. Essa verificação acontece com a aplicação “sabendo” de informações em tempo de compilação sobre a integridade da própria aplicação, por meio de verificação *Cyclic Redundancy Check* (CRC) no *bytecode* do aplicativo, bibliotecas nativas e arquivos de dados importantes. CRC é um código de detecção de erro que serve para analisar a integridade de dados.

¹⁹ Teste do MASTG disponível em: <https://mas.owasp.org/MASTG/Android/0x05j-Testing-Resiliency-Against-Reverse-Engineering/#testing-file-in-tegrity-checks-mstg-resilience-3>

Engenharia reversa

Analisando o aplicativo Diva, pode-se perceber uma funcionalidade de autenticação em que o usuário deve digitar uma senha de acesso à qual não se tem nenhuma informação à princípio. A interface é mostrada na Figura 24 a seguir:

Figura 24 - Funcionalidade de acesso do aplicativo Diva



Fonte: elaborado pelo autor.

Utilizando a ferramenta MobSF e observando o código fonte da aplicação após a decompilação, o arquivo referente a “*HardCodeActivity*” contém o procedimento referente à funcionalidade de acesso descrita. Nesse procedimento, pode-se verificar a comparação do dado inserido pelo usuário com a *string* “*vendorsecretkey*”, em texto simples, que é a senha secreta e que pode ser descoberta observando o código. A visualização do código, mais especificamente na linha indicada pela seta vermelha, pode ser vista na Figura 25 a seguir:

Figura 25 - Trecho de código referente ao procedimento de acesso do aplicativo Diva

```
public void access(View view) {  
    EditText hkey = (EditText) findViewById(R.id.hcKey);  
    → if (hkey.getText().toString().equals("vendorsecretkey")) {  
        Toast.makeText(this, "Access granted! See you on the other side :)", 0).show();  
    } else {  
        Toast.makeText(this, "Access denied! See you in hell :D", 0).show();  
    }  
}
```

Fonte: elaborado pelo autor.

Esse problema afeta o requisito de segurança do MASVS denominado “MSTG-AUTH-1: Se o aplicativo fornecer aos usuários acesso a um serviço remoto, alguma forma de autenticação, como autenticação de nome de usuário e senha, será executada no terminal remoto”. O teste do MASTG para averiguar esse requisito é denominado “Testando credenciais de confirmação (MSTG-AUTH-1 e MSTG-STORAGE-11)”²⁰. Toda chave de identificação e/ou dados confidenciais devem ser retirados do código fonte, além de utilizar a técnica de ofuscação em partes importantes do código para não revelar a lógica principal do aplicativo, equilibrando com a perda de desempenho que a técnica promove à aplicação.

Funcionalidade estranha

Utilizando a ferramenta MobSF para analisar o aplicativo *InsecureBankv2*, como também para decompilar a aplicação para obter seu código fonte *Java*, e acessando o arquivo de código fonte referente a “*LoginActivity*”, observa-se o trecho de código referente ao procedimento de *login*, onde se identifica a utilização da *Activity* “*DoLogin*” para tratar do dados de *login* inseridos pelo usuário. A Figura 26, mais especificamente na linha indicada pela seta vermelha, a seguir mostra a análise:

Figura 26 - Trecho de código onde mostra a utilização de “*DoLogin*” do aplicativo Diva

```
protected void performlogin() {  
    this.Username_Text = (EditText) findViewById(R.id.loginscreen_username);  
    this.Password_Text = (EditText) findViewById(R.id.loginscreen_password);  
    → Intent i = new Intent(this, DoLogin.class);  
    i.putExtra("passed_username", this.Username_Text.getText().toString());  
    i.putExtra("passed_password", this.Password_Text.getText().toString());  
    startActivity(i);  
}
```

Fonte: elaborado pelo autor.

Conferindo o arquivo de código fonte de “*DoLogin*”, é possível notar que há um usuário padrão denominado “*devadmin*”, que tem um tratamento especial e é mostrado na Figura 27: não informando uma senha como, também, informando uma senha qualquer, o *login* será realizado com sucesso para esse usuário, indicado das linhas 19 a 22. Além disso, é utilizado um endpoint diferente para realizar o *login* deste usuário, indicado na linha 15.

²⁰ Teste do MASTG disponível em:
<https://mas.owasp.org/MASTG/Android/0x05f-Testing-Local-Authentication/#testing-confirm-credentials-mstg-auth-1-and-mstg-storage-11>

Figura 27 - Trecho de código onde mostra o diferente tratamento para o usuário “devadmin” no aplicativo Diva

```
9 public void postData(String valueIWantToSend) throws ClientProtocolException, IOException, JSONException,
10 InvalidKeyException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidAlgorithmParameterException,
11 IllegalBlockSizeException, BadPaddingException {
12     HttpResponse responseBody;
13     DefaultHttpClient defaultHttpClient = new DefaultHttpClient();
14     HttpPost httpPost = new HttpPost(DoLogin.this.protocol + DoLogin.this.serverip + ":" + DoLogin.this.serverport + "/login");
15     HttpPost httpPost2 = new HttpPost(DoLogin.this.protocol + DoLogin.this.serverip + ":" + DoLogin.this.serverport + "/" + "devlogin");
16     List<NameValuePair> nameValuePairs = new ArrayList<>(2);
17     nameValuePairs.add(new BasicNameValuePair("username", DoLogin.this.username));
18     nameValuePairs.add(new BasicNameValuePair("password", DoLogin.this.password));
19     if (DoLogin.this.username.equals("devadmin")) {
20         httpPost2.setEntity(new UrlEncodedFormEntity(nameValuePairs));
21         responseBody = defaultHttpClient.execute(httpPost2);
22     } else {
23         httpPost.setEntity(new UrlEncodedFormEntity(nameValuePairs));
24         responseBody = defaultHttpClient.execute(httpPost);
25     }
26     InputStream in = responseBody.getEntity().getContent();
27     DoLogin.this.result = convertStreamToString(in);
28     DoLogin.this.result = DoLogin.this.result.replace("\n", "");
29     if (DoLogin.this.result == null) {
30         return;
31     }
32     if (DoLogin.this.result.indexOf("Correct Credentials") != -1) {
33         Log.d("Successful Login:", "account=" + DoLogin.this.username + ":" + DoLogin.this.password);
34         saveCreds(DoLogin.this.username, DoLogin.this.password);
35         trackUserLogins();
36     }
```

Fonte: elaborado pelo autor.

Esse problema afeta o requisito de segurança do MASVS denominado “MSTG-ARCH-1: Todos os componentes do aplicativo são identificados e reconhecidos como necessários” e ainda não há um teste registrado no MASTG direcionado a esse requisito. Recomenda-se verificar se todos os endpoints de API estão disponíveis publicamente, retirar códigos de teste antes da compilação da fase de produção e log de mensagens para garantir que nenhuma informação descritiva do backend fique registrada.