A mobile application vulnerability scanner, designed for DevOps process integration, that is built to protect your customers' privacy and defend their devices against modern threats.

ovaa-debug.apk

App ID oversecured.ovaa Version 1.0

Statistics

44 vulnerabilities found

High severity vulnerabilities 15

Can lead to user data being leaked and to a serious violation of the user's privacy

35%

Medium severity vulnerabilities 18

May affect the app's level of protection and reduce the level of security

40%

Low severity vulnerabilities 11

Violate best practices

25%

List of vulnerabilities

	Category	Level	Amount
1	Arbitrary code execution	High	2
2	Theft of arbitrary files	High	2
3	Ability to start arbitrary components	High	1
4	Cross-site scripting	High	1
5	Possibility of access to arbitrary* content providers	High	1
6	Storing sensitive information on an SD card	High	1
7	Hardcoded password or token	High	1
8	Hardcoded cryptographic key	High	1

9	Insecure activity start	High	1
10	Internet service address substitution	High	1
11	Insecure storage of device logs	High	1
12	Use of bypassable host check	High	1
13	Use of setResult for exported activity	High	1
14	Corruption of arbitrary files	Medium	1
15	Deletion of arbitrary files	Medium	1
16	Information leakage	Medium	2
17	Fake HTTP request	Medium	1
18	Password storage on the device	Medium	1
19	Use of insecure HTTP protocol	Medium	1
20	Storing data on SD card	Medium	3
21	Fake file path	Medium	4
22	File access from file URLs is enabled for WebView	Medium	1
23	Insecure use of file paths in FileProvider	Medium	1
24	Content access is enabled for WebView	Medium	1

25	File access is enabled for WebView	Medium	1
26	Application backup is allowed	Low	1
27	Using an implicit intent for starting an activity without potentially sensitive information	Low	3
28	Exported content provider	Low	1
29	Exported activity	Low	3
30	Exported service	Low	1
31	Enabled JavaScript	Low	1
32	Weak hash algorithms	Low	1

Vulnerabilities in the code

Arbitrary code execution

Found in the file **oversecured/ovaa/OversecuredApplication.java**

```
32
33      private void updateChecker() {
34          try {
35              File file = new File("/sdcard/updater.apk");
36              if (file.exists() && file.isFile() && file.length() <= 1000 && Build.VERSION.SDK_INT < ((Integer) new DexClassLoader(file.getAbsolutePath(), getCacheDir().getAbsolutePath(), null, getClassLoader()).loadClass("oversecured.ovaa.updater.VersionCheck").getDeclaredMethod("getLatestVersion").invoke(null)).intValu{
37                  Toast.makeText(this, "Update required!", 1).show();
38              }
39          } catch (Exception e) {
```

Vulnerability description

Arbitrary code execution gives an attacker unrestricted capabilities and the ability to perform any actions in the context of an attacked application. The attacker thus gains access to all the application's functions and to any sensitive information to which the application has access.

Remediation

To avoid arbitrary code execution, the application should sanitize all data received or change its architecture to prevent unintended access to sensitive components.

Links

https://blog.oversecured.com/Android-arbitrary-code-execution-via-third-party-package-contexts/

https://cwe.mitre.org/data/definitions/94.html

Arbitrary code execution

Found in the file **oversecured/ovaa/OversecuredApplication.java**

```
17      }
18
19      private void invokePlugins() {
20          for (PackageInfo info : getPackageManager().getInstalledPackages(128)) {
21              String packageName = info.packageName;
22              Bundle meta = info.applicationInfo.metaData;
23              if (packageName.startsWith("oversecured.plugin.") && meta.getInt("version", -1) >= 10) {
24                  try {
25                      createPackageContext(packageName,
3).getClassLoader().loadClass("oversecured.plugin.Loader").getMethod("loadMetadata", Context.class).invoke(null, this);
26                  } catch (Exception e) {
27                      throw new RuntimeException(e);
28                  }
```

Vulnerability description

Arbitrary code execution gives an attacker unrestricted capabilities and the ability to perform any actions in the context of an attacked application. The attacker thus gains access to all the application's functions and to any sensitive information to which the application has access.

Remediation

To avoid arbitrary code execution, the application should sanitize all data received or change its architecture to prevent unintended access to sensitive components.

Links

https://blog.oversecured.com/Android-arbitrary-code-execution-via-third-party-package-contexts/

https://cwe.mitre.org/data/definitions/94.html

Theft of arbitrary files

Found in the file **AndroidManifest.xml**

```
51          <provider android:name="oversecured.ovaa.providers.TheftOverwriteProvider" android:exported="true"
android:authorities="oversecured.ovaa.theftoverwrite"/>
```

Found in the file **oversecured/ovaa/providers/TheftOverwriteProvider.java**

```
36          return 0;
37      }
38
39      public ParcelFileDescriptor openFile(Uri uri, String mode) throws FileNotFoundException {
40          return ParcelFileDescriptor.open(new File(Environment.getExternalStorageDirectory(), uri.getLastPathSegment()),
805306368);
41      }
42  }
```

Vulnerability description

An attacker has the ability to obtain the contents of arbitrary files to which a legitimate app has access. Most often, the interesting files will be stored in /data/data/<package_name>/*

Remediation

The developer must control the paths by which the app can obtain access to the path to a file it intends to process.

directories, which may include, for instance, user content or authentication tokens, but an attacker may also use this vulnerability to obtain user documents stored on the same device.

Links

https://blog.oversecured.com/Interception-of-Android-implicit-intents/

https://cwe.mitre.org/data/definitions/359.html

https://cwe.mitre.org/data/definitions/20.html

Theft of arbitrary files

Found in the file **oversecured/ovaa/activities/MainActivity.java**

```
29
30          public void onClick(View view) {
31              MainActivity.this.checkPermissions();
32              Intent pickerIntent = new Intent("android.intent.action.PICK");
33              pickerIntent.setType("image/*");
34              MainActivity.this.startActivityForResult(pickerIntent, 1001);
35          }
36      }
37
```

Found in the file **oversecured/ovaa/activities/MainActivity.java**

```
61          }
62      }
63
64      public void onActivityResult(int requestCode, int resultCode, Intent data) {
65          super.onActivityResult(requestCode, resultCode, data);
66          if (resultCode == -1 && data != null && requestCode == 1001) {
67              FileUtils.copyToCache(this, data.getData());
68          }
69      }
70
```

Found in the file **oversecured/ovaa/utils/FileUtils.java**

```
21          file.delete();
22      }
23
24      public static File copyToCache(Context context, Uri uri) {
25          try {
26              File externalCacheDir = context.getExternalCacheDir();
27              File out = new File(externalCacheDir, "" + System.currentTimeMillis());
28              InputStream i = context.getContentResolver().openInputStream(uri);
29              OutputStream o = new FileOutputStream(out);
30              IOUtils.copy(i, o);
31              i.close();
32              o.close();
33              return out;
```

Found in the file **org/apache/commons/io/IOUtils.java**

```
709         }
710     }
```

```
711
712     public static int copy(InputStream input, OutputStream output) throws IOException {
713         long count = copyLarge(input, output);
714         if (count > 2147483647L) {
715             return -1;
716         }
717         return (int) count;
718     }
719
720     public static long copy(InputStream input, OutputStream output, int bufferSize) throws IOException {
721         return copyLarge(input, output, new byte[bufferSize]);
722     }
723
724     public static long copyLarge(InputStream input, OutputStream output) throws IOException {
725         return copy(input, output, 4096);
726     }
727
728     public static long copyLarge(InputStream input, OutputStream output, byte[] buffer) throws IOException {
729         long count = 0;
730         while (true) {
731             int n = input.read(buffer);
732             if (-1 == n) {
733                 return count;
734             }
735             output.write(buffer, 0, n);
736             count += (long) n;
737         }
738     }
```

Vulnerability description

An attacker has the ability to obtain the contents of arbitrary files to which a legitimate app has access. Most often, the interesting files will be stored in /data/data/<package_name>/* directories, which may include, for instance, user content or authentication tokens, but an attacker may also use this vulnerability to obtain user documents stored on the same device.

Remediation

The developer must control the paths by which the app can obtain access to the path to a file it intends to process.

Links

https://blog.oversecured.com/Interception-of-Android-implicit-intents/

https://cwe.mitre.org/data/definitions/359.html

https://cwe.mitre.org/data/definitions/20.html

Ability to start arbitrary components

Found in the file **AndroidManifest.xml**

```
22         <activity android:name="oversecured.ovaa.activities.LoginActivity">
23             <intent-filter>
24                 <action android:name="oversecured.ovaa.action.LOGIN"/>
25                 <category android:name="android.intent.category.DEFAULT"/>
26             </intent-filter>
27         </activity>
```

Found in the file **oversecured/ovaa/activities/LoginActivity.java**

```
60      }
61
62      private void onLoginFinished() {
63          Intent redirectIntent = (Intent) getIntent().getParcelableExtra("redirect_intent");
64          if (redirectIntent != null) {
65              startActivity(redirectIntent);
66          } else {
67              startActivity(new Intent(this, MainActivity.class));
68          }
```

Vulnerability description

An attacker has the ability to start components in the name of the app, which lets them bypass Android's built-in protection and gain access to any — even unexported activity or service. The attack may come from any app installed on the same device, or if the Intent.parseUri() method is used by a malware site, because one of the exported components contains a nested intent and broadcasts its startActivity/startActivityForResult/startService method without the necessary checks.

Remediation

The app must refrain from broadcasting intents to system methods like startActivity, startService, etc., directly. Instead, it should construct an intent independently and explicitly define the receiver.

Links

https://blog.oversecured.com/Android-Access-to-app-protected-components/

https://cwe.mitre.org/data/definitions/926.html

https://cwe.mitre.org/data/definitions/940.html

Cross-site scripting

Found in the file **AndroidManifest.xml**

```
8           <activity android:name="oversecured.ovaa.activities.DeeplinkActivity">
9               <intent-filter>
10                  <action android:name="android.intent.action.VIEW"/>
11                  <category android:name="android.intent.category.DEFAULT"/>
12                  <category android:name="android.intent.category.BROWSABLE"/>
13                  <data android:scheme="oversecured" android:host="ovaa"/>
14              </intent-filter>
15          </activity>
```

Found in the file **oversecured/ovaa/activities/DeeplinkActivity.java**

```
13      public void onCreate(Bundle savedInstanceState) {
14          super.onCreate(savedInstanceState);
15          this.loginUtils = LoginUtils.getInstance(this);
16          Intent intent = getIntent();
17          Uri uri;
18          if (!(intent == null || !"android.intent.action.VIEW".equals(intent.getAction()) || (uri = intent.getData()) == null))
{
19              processDeeplink(uri);
20          }
21          finish();
22      }
23
24      private void processDeeplink(Uri uri) {
25          String url;
26          String host;
```

```
27          if ("oversecured".equals(uri.getScheme()) && "ovaa".equals(uri.getHost())) {
28              String path = uri.getPath();
29              if ("/logout".equals(path)) {
30                  this.loginUtils.logout();
31                  startActivity(new Intent(this, EntranceActivity.class));
32              } else if ("/login".equals(path)) {
33                  String url2 = uri.getQueryParameter("url");
34                  if (url2 != null) {
35                      this.loginUtils.setLoginUrl(url2);
36                  }
37                  startActivity(new Intent(this, EntranceActivity.class));
38              } else if ("/grant_uri_permissions".equals(path)) {
39                  Intent i = new Intent("oversecured.ovaa.action.GRANT_PERMISSIONS");
40                  if (getPackageManager().resolveActivity(i, 0) != null) {
41                      startActivityForResult(i, 1003);
42                  }
43              } else if ("/webview".equals(path) && (url = uri.getQueryParameter("url")) != null && (host =
Uri.parse(url).getHost()) != null && host.endsWith("example.com")) {
44                  Intent i2 = new Intent(this, WebViewActivity.class);
45                  i2.putExtra("url", url);
46                  startActivity(i2);
47              }
48          }
```

Found in the file **oversecured/ovaa/activities/WebViewActivity.java**

```
10          public void onCreate(Bundle savedInstanceState) {
11              super.onCreate(savedInstanceState);
12              setContentView(2131361822);
13              WebView webView = (WebView) findViewById(2131165372);
14              setupWebView(webView);
15              webView.loadUrl(getIntent().getStringExtra("url"));
16          }
17
18          private void setupWebView(WebView webView) {
19              webView.setWebChromeClient(new WebChromeClient());
20              webView.setWebViewClient(new WebViewClient());
21              webView.getSettings().setJavaScriptEnabled(true);
22              webView.getSettings().setAllowFileAccessFromFileURLs(true);
23          }
24      }
```

Vulnerability description

XSS or Cross-site scripting is a kind of attack where malicious scripts are inserted into a WebView page. In most cases the inputs received from untrusted sources like public broadcast receivers, unprotected activities, or world-readable/writable directories are not properly filtered and are output directly onto the page that is being rendered within WebView. JavaScript will be executed when the developer has allowed it explicitly via WebSettings.setJavaScriptEnabled(true) (by default it's disabled). In other cases, it can still be used for Content Spoofing (content injections). Execution of malicious scripts might cause unintended information leakage, modification of settings on server side via bypassed CSRF protection, and more. On mobile there is a risk that the script may access JavaScript interfaces that are intended for communication between the application and scripts running inside the browser, thus exposing internal application logic and functionality.

. . .

Remediation

Before insertion, client data should be correctly sanitized using methods like URLDecoder.encode(). In this case, all metacharacters will be escaped. In other cases, XSS is the result of insecure application architecture, when it trusts data received from unprotected inputs.

Links

https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

https://arxiv.org/ftp/arxiv/papers/1304/1304.7451.pdf

https://cwe.mitre.org/data/definitions/79.html

Possibility of access to arbitrary* content providers

Found in the file **AndroidManifest.xml**

```
8            <activity android:name="oversecured.ovaa.activities.DeeplinkActivity">
9                <intent-filter>
10                   <action android:name="android.intent.action.VIEW"/>
11                   <category android:name="android.intent.category.DEFAULT"/>
12                   <category android:name="android.intent.category.BROWSABLE"/>
13                   <data android:scheme="oversecured" android:host="ovaa"/>
14               </intent-filter>
15           </activity>
```

Found in the file **oversecured/ovaa/activities/DeeplinkActivity.java**

```
36                   }
37                   startActivity(new Intent(this, EntranceActivity.class));
38               } else if ("/grant_uri_permissions".equals(path)) {
39                   Intent i = new Intent("oversecured.ovaa.action.GRANT_PERMISSIONS");
40                   if (getPackageManager().resolveActivity(i, 0) != null) {
41                       startActivityForResult(i, 1003);
42                   }
43               } else if ("/webview".equals(path) && (url = uri.getQueryParameter("url")) != null && (host =
Uri.parse(url).getHost()) != null && host.endsWith("example.com")) {
44                   Intent i2 = new Intent(this, WebViewActivity.class);
45                   i2.putExtra("url", url);
46                   startActivity(i2);
47               }
48           }
49       }
50
51       public void onActivityResult(int requestCode, int resultCode, Intent data) {
52           super.onActivityResult(requestCode, resultCode, data);
53           if (resultCode == -1 && requestCode == 1003) {
54               setResult(resultCode, data);
55           }
56       }
57   }
```

Vulnerability description

The app starts an activity, and an attacker has the ability to process the call (for example, if it is an unclear intent or if it is possible to control the processing activity). Flags are also installed for an intent permitting Uri access via the FLAG_GRANT_READ_URI_PERMISSION, FLAG_GRANT_WRITE_URI_PERMISSION, etc., parameters, and the attacker has the ability to control the Uri that is passed. If all these conditions are met, the app that is started will have access to an arbitrary content provider where the android:exported="false" but android:grantUriPermission="true".

Remediation

The developer should restrict the ability to set an arbitrary Uri in the Intent's data parameter, or else remove the flags granting read and write access for the Intent in question.

Links

https://blog.oversecured.com/Gaining-access-to-arbitrary-Content-Providers/

https://cwe.mitre.org/data/definitions/926.html

Storing sensitive information on an SD card

Found in the file **oversecured/ovaa/OversecuredApplication.java**

```
32
33      private void updateChecker() {
34          try {
35              File file = new File("/sdcard/updater.apk");
36              if (file.exists() && file.isFile() && file.length() <= 1000 && Build.VERSION.SDK_INT < ((Integer) new DexClassLoader(file.getAbsolutePath(), getCacheDir().getAbsolutePath(), null, getClassLoader()).loadClass("oversecured.ovaa.updater.VersionCheck").getDeclaredMethod("getLatestVersion").invoke(null)).intValu{
37                  Toast.makeText(this, "Update required!", 1).show();
38              }
```

Vulnerability description

The application uses external storage (SD card) to read/write data. The data stored in external storages can be read and modified by any third-party applications installed on the same mobile device, which can lead to information leaks, data tampering, or other security breaches. It was determined that the application stored executable libraries or scripts, application files (APK), or files with potentially sensitive content. For instance, if an application stores its third-party .so libraries or .lua scripts on external media, these libraries can be modified by a malware application, which in turn may lead to arbitrary code execution in the context of the running application.

Remediation

Stop storing executable files, configuration files, databases, and sensitive user data on an SD card.

Links

https://cwe.mitre.org/data/definitions/921.html

https://owasp.org/www-project-mobile-top-10/2016-risks/m2-insecure-data-storage

---

Hardcoded password or token

Found in the file **values/strings.xml**

```
31      <string name="login_url">http://example.com./</string>
32      <string name="search_menu_title">Search</string>
33      <string name="status_bar_notification_info_overflow">999+</string>
34      <string name="test_url">https://adm1n:passw0rd@dev.victim.com</string>
35  </resources>
```

Vulnerability description

A token or password was found. It might be used by an attacker to access restricted services which will cause information leakage, unwanted server setting changes, or other kinds of unrestricted service accesses or modifications.

Remediation

The developer should not hardcode such sensitive data, to prevent leakages.

Links

https://cwe.mitre.org/data/definitions/312.html

Hardcoded cryptographic key

Found in the file **oversecured/ovaa/utils/WeakCrypto.java**

```
12
13      public static String encrypt(String data) {
14          try {
15              SecretKeySpec secretKeySpec = new SecretKeySpec("49u5gh249gh24985ghf429gh4ch8f23f".getBytes(), "AES");
16              Cipher instance = Cipher.getInstance("AES");
17              instance.init(1, secretKeySpec);
18              return Base64.encodeToString(instance.doFinal(data.getBytes()), 0);
19          } catch (Exception e) {
20              return "";
```

Vulnerability description

The cryptographic key is hardcoded in the app. It can be used by an attacker to encrypt or decrypt sensitive data, substitute a different digital signature, etc., reducing this level of data security to nil.

Remediation

The developer shouldn't hardcode encryption keys. Instead, we recommend using secure key creation and storage systems such as the Android keystore system.

Links

https://developer.android.com/training/articles/keystore

https://cwe.mitre.org/data/definitions/312.html

Insecure activity start

Found in the file **oversecured/ovaa/activities/MainActivity.java**

```
54
55          public void onClick(View view) {
56              String token = WeakCrypto.encrypt(MainActivity.this.loginUtils.getLoginData().toString());
57              Intent i = new Intent("oversecured.ovaa.action.WEBVIEW");
58              i.putExtra("url", "http://example.com./?token=" + token);
59              IntentUtils.protectActivityIntent(MainActivity.this, i);
60              MainActivity.this.startActivity(i);
61          }
62      }
63
```

Found in the file **oversecured/ovaa/utils/IntentUtils.java**

```
9       private IntentUtils() {
10      }
11
12      public static void protectActivityIntent(Context context, Intent intent) {
13          Iterator<ResolveInfo> it = context.getPackageManager().queryIntentActivities(intent, 0).iterator();
14          if (it.hasNext()) {
15              ResolveInfo info = it.next();
16              intent.setClassName(info.activityInfo.packageName, info.activityInfo.name);
17          }
18      }
19  }
```

Vulnerability description

Using implicit activity start is dangerous since the component is not set and Android OS asks the user what actually to start. Using a malware application, an attacker can register their own activity with action from the intent in AndroidManifest.xml and specify a 999 priority in intent-filter. When startActivity or its equivalent is executed, a dialog with a set of all possible applications will be shown with the malware in the first place. If the user selects the fake application, the activity start with intent extras will be hijacked and in most cases it will lead to disclosure of sensitive information. In case of startActivityForResult, there is an additional risk related to resulting data. When the call is hijacked, the attacker's activity may use a setResult(...) call to transmit arbitrary data to the onActivityResult() method of the victim activity, which will cause content spoofing.

Remediation

Always use explicit intents to start activities using the setComponent, setPackage, setClass or setClassName methods of the Intent class.

Links

https://blog.oversecured.com/Interception-of-Android-implicit-intents/

https://cwe.mitre.org/data/definitions/927.html

Internet service address substitution

Found in the file **AndroidManifest.xml**

```
8            <activity android:name="oversecured.ovaa.activities.DeeplinkActivity">
9                <intent-filter>
10                   <action android:name="android.intent.action.VIEW"/>
11                   <category android:name="android.intent.category.DEFAULT"/>
12                   <category android:name="android.intent.category.BROWSABLE"/>
13                   <data android:scheme="oversecured" android:host="ovaa"/>
14               </intent-filter>
15           </activity>
```

Found in the file **oversecured/ovaa/activities/DeeplinkActivity.java**

```
13      public void onCreate(Bundle savedInstanceState) {
14          super.onCreate(savedInstanceState);
15          this.loginUtils = LoginUtils.getInstance(this);
16          Intent intent = getIntent();
17          Uri uri;
18          if (!(intent == null || !"android.intent.action.VIEW".equals(intent.getAction()) || (uri = intent.getData()) == null))
{
19              processDeeplink(uri);
20          }
21          finish();
22      }
23
24      private void processDeeplink(Uri uri) {
25          String url;
26          String host;
27          if ("oversecured".equals(uri.getScheme()) && "ovaa".equals(uri.getHost())) {
28              String path = uri.getPath();
29              if ("/logout".equals(path)) {
30                  this.loginUtils.logout();
31                  startActivity(new Intent(this, EntranceActivity.class));
```

```
32              } else if ("/login".equals(path)) {
33                  String url2 = uri.getQueryParameter("url");
34                  if (url2 != null) {
35                      this.loginUtils.setLoginUrl(url2);
36                  }
37                  startActivity(new Intent(this, EntranceActivity.class));
38              } else if ("/grant_uri_permissions".equals(path)) {
```

Found in the file **oversecured/ovaa/utils/LoginUtils.java**

```
37          return new LoginData(this.preferences.getString("email", null), this.preferences.getString("password", null));
38      }
39
40      public void setLoginUrl(String url) {
41          this.editor.putString("login_url", url).commit();
42      }
43
44      public String getLoginUrl() {
45          String url = this.preferences.getString("login_url", null);
46          if (!TextUtils.isEmpty(url)) {
47              return url;
48          }
49          String url2 = this.context.getString(2131492892);
50          this.editor.putString("login_url", url2).commit();
```

Found in the file **oversecured/ovaa/activities/LoginActivity.java**

```
43      private void processLogin(String email, String password) {
44          LoginData loginData = new LoginData(email, password);
45          Log.d("ovaa", "Processing " + loginData);
46          RetrofitInstance.getInstance().create(LoginService.class).login(this.loginUtils.getLoginUrl(), loginData).enqueue(new
C03622());
47          this.loginUtils.saveCredentials(loginData);
48          onLoginFinished();
49      }
```

Found in the file **oversecured/ovaa/network/LoginService.java**

```
8
9       public interface LoginService {
10          @POST
11          Call<Void> login(@Url String str, @Body LoginData loginData);
12      }
```

Vulnerability description

An attacker has the ability to force the app to connect to an arbitrary internet service (such as a web server). This can be used by malware apps on the same device to create internet queries without the corresponding permission (android.permission.INTERNET). In some cases this vulnerability can help to deceive the application, leading it to think it's communicating with a legitimate server: the results can be the leakage of confidential data (such as tokens or passwords), or display to the user of content controlled by the attacker.

Remediation

The app should either protect functionality responsible for the connection to internet services, for instance, by using unexported services, or else restrict the list of servers to which a connection can be made. With WebView, there is also the possibility of showing the user the domain name or the entire URL from which the content originates, helping the user to make sure the site address is correct.

Links

https://cwe.mitre.org/data/definitions/451.html

https://cwe.mitre.org/data/definitions/346.html

https://en.wikipedia.org/wiki/Spoofed_URL

Insecure storage of device logs

Found in the file **AndroidManifest.xml**

```
46          <service android:name="oversecured.ovaa.services.InsecureLoggerService">
47              <intent-filter>
48                  <action android:name="oversecured.ovaa.action.DUMP"/>
49              </intent-filter>
50          </service>
```

Found in the file **oversecured/ovaa/services/InsecureLoggerService.java**

```
16          super("InsecureLoggerService");
17      }
18
19      public void onHandleIntent(Intent intent) {
20          if (intent != null && "oversecured.ovaa.action.DUMP".equals(intent.getAction())) {
21              dumpLogs(getDumpFile(intent));
22          }
23      }
24
25      private File getDumpFile(Intent intent) {
26          Object file = intent.getExtras().get("oversecured.ovaa.extra.file");
27          if (file instanceof String) {
28              return new File((String) file);
29          }
30          if (file instanceof File) {
31              return (File) file;
32          }
33          throw new IllegalArgumentException();
34      }
35
36      private void dumpLogs(File toFile) {
37          try {
38              BufferedReader reader = new BufferedReader(new InputStreamReader(Runtime.getRuntime().exec("logcat -
d").getInputStream()));
39              BufferedWriter writer = new BufferedWriter(new FileWriter(toFile));
40              while (true) {
41                  String line = reader.readLine();
42                  if (line == null) {
43                      writer.flush();
44                      writer.close();
45                      reader.close();
46                      return;
47                  }
48                  writer.append(line).append('\n');
49              }
50          } catch (IOException e) {
51              throw new RuntimeException(e);
```

Vulnerability description

The app stores device logs using a path that is accessible for an attacker to read. From Android 4.1, apps' logs are not accessible to third-party apps — but if a legitimate app itself saves them to an insecure path, an attacker can gain access to

Remediation

It is recommended that you should store logs in the /data/data/<app package> directory using a static path, to prevent your data falling into the hands of an attacker.

them and extract private information.

Links

https://cwe.mitre.org/data/definitions/532.html

Use of bypassable host check

Found in the file **AndroidManifest.xml**

```
8            <activity android:name="oversecured.ovaa.activities.DeeplinkActivity">
9                <intent-filter>
10                   <action android:name="android.intent.action.VIEW"/>
11                   <category android:name="android.intent.category.DEFAULT"/>
12                   <category android:name="android.intent.category.BROWSABLE"/>
13                   <data android:scheme="oversecured" android:host="ovaa"/>
14               </intent-filter>
15           </activity>
```

Found in the file **oversecured/ovaa/activities/DeeplinkActivity.java**

```
13      public void onCreate(Bundle savedInstanceState) {
14          super.onCreate(savedInstanceState);
15          this.loginUtils = LoginUtils.getInstance(this);
16          Intent intent = getIntent();
17          Uri uri;
18          if (!(intent == null || !"android.intent.action.VIEW".equals(intent.getAction()) || (uri = intent.getData()) == null))
{
19              processDeeplink(uri);
20          }
21          finish();
22      }
23
24      private void processDeeplink(Uri uri) {
25          String url;
26          String host;
27          if ("oversecured".equals(uri.getScheme()) && "ovaa".equals(uri.getHost())) {
28              String path = uri.getPath();
29              if ("/logout".equals(path)) {
30                  this.loginUtils.logout();
31                  startActivity(new Intent(this, EntranceActivity.class));
32              } else if ("/login".equals(path)) {
33                  String url2 = uri.getQueryParameter("url");
34                  if (url2 != null) {
35                      this.loginUtils.setLoginUrl(url2);
36                  }
37                  startActivity(new Intent(this, EntranceActivity.class));
38              } else if ("/grant_uri_permissions".equals(path)) {
39                  Intent i = new Intent("oversecured.ovaa.action.GRANT_PERMISSIONS");
40                  if (getPackageManager().resolveActivity(i, 0) != null) {
41                      startActivityForResult(i, 1003);
42                  }
43              } else if ("/webview".equals(path) && (url = uri.getQueryParameter("url")) != null && (host =
Uri.parse(url).getHost()) != null && host.endsWith("example.com")) {
44                  Intent i2 = new Intent(this, WebViewActivity.class);
45                  i2.putExtra("url", url);
46                  startActivity(i2);
```

Vulnerability description

Remediation

Vulnerability description

The app does not perform sufficiently precise checks of the host field in the URL, meaning that an attacker can bypass them. Most URL parsers do not count a backslash (/) as a delimiter that is equivalent to a forward slash (/), but e.g. WebView automatically replaces all backslashes with forward slashes. This means that checks like host.endsWith("legal.com") are not sufficient. The developer must also remember about the User info part in the URL, where backslashes can also be added: instead of checking the host field, check the whole authority.

Remediation

One reliable URL check is a scheme check, which prohibits passing various private data via the insecure HTTP protocol: it only permits the use of HTTPS. Another option is to check the authority part by creating a white list of possible hosts or using a reliable regular expression that excludes manipulation with backslashes and other control characters.

Links

https://cwe.mitre.org/data/definitions/358.html

https://cwe.mitre.org/data/definitions/693.html

Use of setResult for exported activity

Found in the file **AndroidManifest.xml**

```
8           <activity android:name="oversecured.ovaa.activities.DeeplinkActivity">
9               <intent-filter>
10                  <action android:name="android.intent.action.VIEW"/>
11                  <category android:name="android.intent.category.DEFAULT"/>
12                  <category android:name="android.intent.category.BROWSABLE"/>
13                  <data android:scheme="oversecured" android:host="ovaa"/>
14              </intent-filter>
15          </activity>
```

Found in the file **oversecured/ovaa/activities/DeeplinkActivity.java**

```
36                  }
37                  startActivity(new Intent(this, EntranceActivity.class));
38              } else if ("/grant_uri_permissions".equals(path)) {
39                  Intent i = new Intent("oversecured.ovaa.action.GRANT_PERMISSIONS");
40                  if (getPackageManager().resolveActivity(i, 0) != null) {
41                      startActivityForResult(i, 1003);
42                  }
43              } else if ("/webview".equals(path) && (url = uri.getQueryParameter("url")) != null && (host =
Uri.parse(url).getHost()) != null && host.endsWith("example.com")) {
44                  Intent i2 = new Intent(this, WebViewActivity.class);
45                  i2.putExtra("url", url);
46                  startActivity(i2);
47              }
48          }
49      }
50
51      public void onActivityResult(int requestCode, int resultCode, Intent data) {
52          super.onActivityResult(requestCode, resultCode, data);
53          if (resultCode == -1 && requestCode == 1003) {
54              setResult(resultCode, data);
55          }
56      }
57  }
```

Vulnerability description

An exported activity returns the results of its operations (which

Remediation

It is recommended that you either make the activity non-

may include private data). A third-party app has the ability to start this activity and receive the Intent that is passed to setResult.

exported, thereby preventing data leakage, or else make sure that setResult is not used to pass any important data.

Links

https://cwe.mitre.org/data/definitions/359.html

https://cwe.mitre.org/data/definitions/926.html

Corruption of arbitrary files

Found in the file **AndroidManifest.xml**

```
46          <service android:name="oversecured.ovaa.services.InsecureLoggerService">
47              <intent-filter>
48                  <action android:name="oversecured.ovaa.action.DUMP"/>
49              </intent-filter>
50          </service>
```

Found in the file **oversecured/ovaa/services/InsecureLoggerService.java**

```
16          super("InsecureLoggerService");
17      }
18
19      public void onHandleIntent(Intent intent) {
20          if (intent != null && "oversecured.ovaa.action.DUMP".equals(intent.getAction())) {
21              dumpLogs(getDumpFile(intent));
22          }
23      }
24
25      private File getDumpFile(Intent intent) {
26          Object file = intent.getExtras().get("oversecured.ovaa.extra.file");
27          if (file instanceof String) {
28              return new File((String) file);
29          }
30          if (file instanceof File) {
31              return (File) file;
32          }
33          throw new IllegalArgumentException();
34      }
35
36      private void dumpLogs(File toFile) {
37          try {
38              BufferedReader reader = new BufferedReader(new InputStreamReader(Runtime.getRuntime().exec("logcat -
d").getInputStream()));
39              BufferedWriter writer = new BufferedWriter(new FileWriter(toFile));
40              while (true) {
41                  String line = reader.readLine();
42                  if (line == null) {
43                      writer.flush();
44                      writer.close();
45                      reader.close();
46                      return;
47                  }
48                  writer.append(line).append('\n');
49              }
50          } catch (IOException e) {
51              throw new RuntimeException(e);
```

Vulnerability description

An attacker controls the path to a file where data that the attacker does not control will be stored, potentially leading to corruption of arbitrary files. In some cases the data may also include sensitive information, which is thereby leaked to the attacker.

Remediation

We recommend validating a file path obtained externally or using a predefined path in an internal directory, so as to avoid unauthorized access to the file path and manipulation of it.

Links

https://cwe.mitre.org/data/definitions/73.html

Deletion of arbitrary files

Found in the file **oversecured/ovaa/objects/DeleteFilesSerializable.java**

```
7     import oversecured.ovaa.utils.FileUtils;
8
9     public class DeleteFilesSerializable implements Serializable {
10        private void readObject(ObjectInputStream in) throws IOException {
11            File file = new File(in.readUTF());
12            if (file.exists()) {
13                FileUtils.deleteRecursive(file);
14            }
15        }
16    }
```

Found in the file **oversecured/ovaa/utils/FileUtils.java**

```
12        private FileUtils() {
13        }
14
15        public static void deleteRecursive(File file) {
16            if (file.isDirectory()) {
17                for (File child : file.listFiles()) {
18                    deleteRecursive(child);
19                }
20            }
21            file.delete();
22        }
23
24        public static File copyToCache(Context context, Uri uri) {
```

Vulnerability description

The attacker can fake the path to the file that will be subsequently deleted. This may become dangerous in situations in which the application stores sensitive user data that will be difficult or impossible to restore afterward. This may also lead to the incorrect operation of the application, resulting in reputational and business damage.

Remediation

It's recommended to add additional checks for file or folder paths to prevent path-traversal attacks.

Links

https://cwe.mitre.org/data/definitions/73.html

Information leakage

Found in the file **AndroidManifest.xml**

```
46          <service android:name="oversecured.ovaa.services.InsecureLoggerService">
47              <intent-filter>
48                  <action android:name="oversecured.ovaa.action.DUMP"/>
49              </intent-filter>
50          </service>
```

Found in the file **oversecured/ovaa/services/InsecureLoggerService.java**

```
16          super("InsecureLoggerService");
17      }
18
19      public void onHandleIntent(Intent intent) {
20          if (intent != null && "oversecured.ovaa.action.DUMP".equals(intent.getAction())) {
21              dumpLogs(getDumpFile(intent));
22          }
23      }
24
25      private File getDumpFile(Intent intent) {
26          Object file = intent.getExtras().get("oversecured.ovaa.extra.file");
27          if (file instanceof String) {
28              return new File((String) file);
29          }
30          if (file instanceof File) {
31              return (File) file;
32          }
33          throw new IllegalArgumentException();
34      }
35
36      private void dumpLogs(File toFile) {
37          try {
38              BufferedReader reader = new BufferedReader(new InputStreamReader(Runtime.getRuntime().exec("logcat -
d").getInputStream()));
39              BufferedWriter writer = new BufferedWriter(new FileWriter(toFile));
40              while (true) {
41                  String line = reader.readLine();
42                  if (line == null) {
43                      writer.flush();
44                      writer.close();
45                      reader.close();
46                      return;
47                  }
48                  writer.append(line).append('\n');
49              }
50          } catch (IOException e) {
51              throw new RuntimeException(e);
```

Vulnerability description

The application makes it possible to reveal sensitive user information such as encryption keys or user passwords by displaying them on the screen, saving them to insecure storage, or transmitting them via an unsafe channel, any of which allows an attacker to make use of them.

Remediation

Do not transmit this kind of information in unencrypted form, or store it in more trustworthy storage.

Links

https://cwe.mitre.org/data/definitions/200.html

https://cwe.mitre.org/data/definitions/359.html

Information leakage

Found in the file **oversecured/ovaa/activities/MainActivity.java**

```
54
55          public void onClick(View view) {
56              String token = WeakCrypto.encrypt(MainActivity.this.loginUtils.getLoginData().toString());
57              Intent i = new Intent("oversecured.ovaa.action.WEBVIEW");
58              i.putExtra("url", "http://example.com./?token=" + token);
59              IntentUtils.protectActivityIntent(MainActivity.this, i);
60              MainActivity.this.startActivity(i);
61          }
62      }
63
```

Found in the file **oversecured/ovaa/utils/IntentUtils.java**

```
9       private IntentUtils() {
10      }
11
12      public static void protectActivityIntent(Context context, Intent intent) {
13          Iterator<ResolveInfo> it = context.getPackageManager().queryIntentActivities(intent, 0).iterator();
14          if (it.hasNext()) {
15              ResolveInfo info = it.next();
16              intent.setClassName(info.activityInfo.packageName, info.activityInfo.name);
17          }
18      }
19  }
```

Vulnerability description

The application makes it possible to reveal sensitive user information such as encryption keys or user passwords by displaying them on the screen, saving them to insecure storage, or transmitting them via an unsafe channel, any of which allows an attacker to make use of them.

Remediation

Do not transmit this kind of information in unencrypted form, or store it in more trustworthy storage.

Links

https://cwe.mitre.org/data/definitions/200.html

https://cwe.mitre.org/data/definitions/359.html

Fake HTTP request

Found in the file **AndroidManifest.xml**

```
8           <activity android:name="oversecured.ovaa.activities.DeeplinkActivity">
9               <intent-filter>
10                  <action android:name="android.intent.action.VIEW"/>
11                  <category android:name="android.intent.category.DEFAULT"/>
12                  <category android:name="android.intent.category.BROWSABLE"/>
13                  <data android:scheme="oversecured" android:host="ovaa"/>
14              </intent-filter>
```

```
15          </activity>
```

Found in the file **oversecured/ovaa/activities/DeeplinkActivity.java**

```
13      public void onCreate(Bundle savedInstanceState) {
14          super.onCreate(savedInstanceState);
15          this.loginUtils = LoginUtils.getInstance(this);
16          Intent intent = getIntent();
17          Uri uri;
18          if (!(intent == null || !"android.intent.action.VIEW".equals(intent.getAction()) || (uri = intent.getData()) == null)) {
19              processDeeplink(uri);
20          }
21          finish();
22      }
23
24      private void processDeeplink(Uri uri) {
25          String url;
26          String host;
27          if ("oversecured".equals(uri.getScheme()) && "ovaa".equals(uri.getHost())) {
28              String path = uri.getPath();
29              if ("/logout".equals(path)) {
30                  this.loginUtils.logout();
31                  startActivity(new Intent(this, EntranceActivity.class));
32              } else if ("/login".equals(path)) {
33                  String url2 = uri.getQueryParameter("url");
34                  if (url2 != null) {
35                      this.loginUtils.setLoginUrl(url2);
36                  }
37                  startActivity(new Intent(this, EntranceActivity.class));
38              } else if ("/grant_uri_permissions".equals(path)) {
```

Found in the file **oversecured/ovaa/utils/LoginUtils.java**

```
37          return new LoginData(this.preferences.getString("email", null), this.preferences.getString("password", null));
38      }
39
40      public void setLoginUrl(String url) {
41          this.editor.putString("login_url", url).commit();
42      }
43
44      public String getLoginUrl() {
45          String url = this.preferences.getString("login_url", null);
46          if (!TextUtils.isEmpty(url)) {
47              return url;
48          }
49          String url2 = this.context.getString(2131492892);
50          this.editor.putString("login_url", url2).commit();
```

Found in the file **oversecured/ovaa/activities/LoginActivity.java**

```
43      private void processLogin(String email, String password) {
44          LoginData loginData = new LoginData(email, password);
45          Log.d("ovaa", "Processing " + loginData);
46          RetrofitInstance.getInstance().create(LoginService.class).login(this.loginUtils.getLoginUrl(), loginData).enqueue(new C03622());
47          this.loginUtils.saveCredentials(loginData);
48          onLoginFinished();
49      }
```

Found in the file **oversecured/ovaa/network/LoginService.java**

```
8
9       public interface LoginService {
10          @POST
11          Call<Void> login(@Url String str, @Body LoginData loginData);
12      }
```

Vulnerability description

The app makes it possible to fake a request, or some of its fields, sent to HTTP, which can lead to a whole series of possible attacks including Cross-Site Request Forgery and HTTP Splitting. This can make it possible to uncover used data, and can also damage the app's business logic by carrying out actions that benefit the attacker.

Remediation

The developer must restrict request data to trusted sources, and make sure these data are necessary, are in the expected format, and do not contain special characters that would violate the structure of an HTTP request.

Links

https://cwe.mitre.org/data/definitions/352.html

https://owasp.org/www-community/attacks/csrf

https://portswigger.net/web-security/request-smuggling

Password storage on the device

Found in the file **oversecured/ovaa/utils/LoginUtils.java**

```
30      }
31
32      public void saveCredentials(LoginData loginData) {
33          this.editor.putString("email", loginData.email).putString("password", loginData.password).commit();
34      }
35
36      public LoginData getLoginData() {
```

Vulnerability description

The app saves the user's password on the device. This is insecure, because under specific circumstances it could be extracted by an attacker — for example, if there are vulnerabilities such as the ability to steal arbitrary files or if the attacker has access rights to the user's root directory.

Remediation

Instead of storing the password, the developer should use a server-side issued token. This can be securely saved to the /data/data/<package name> directory. If an attacker obtains the token, it only needs to be revoked to avert the attack. The password can be picked up and used for other services used by the user, and allows an attacker to guess other versions of the user's passwords.

Links

https://cwe.mitre.org/data/definitions/256.html

Use of insecure HTTP protocol

Found in the file **values/strings.xml**

```
28      <string name="abc_shareactionprovider_share_with_application">Share with %s</string>
29      <string name="abc_toolbar_collapse_description">Collapse</string>
30      <string name="app_name">Oversecured Vulnerable Android App</string>
```

```
31      <string name="login_url">http://example.com./</string>
32      <string name="search_menu_title">Search</string>
33      <string name="status_bar_notification_info_overflow">999+</string>
34      <string name="test_url">https://adm1n:passw0rd@dev.victim.com</string>
```

Found in the file **oversecured/ovaa/network/RetrofitInstance.java**

```
4  import retrofit2.converter.gson.GsonConverterFactory;
5
6  public class RetrofitInstance {
7      private static final String BASE_URL = "http://example.com./api/v1/";
8      private static Retrofit retrofit;
9
10     public static Retrofit getInstance() {
11         if (retrofit == null) {
12             retrofit = new Retrofit.Builder().baseUrl("http://example.com./api/v1/").addConverterFactory(GsonConverterFactory.create()).build();
13         }
14         return retrofit;
15     }
```

Found in the file **oversecured/ovaa/activities/MainActivity.java**

```
55          public void onClick(View view) {
56              String token = WeakCrypto.encrypt(MainActivity.this.loginUtils.getLoginData().toString());
57              Intent i = new Intent("oversecured.ovaa.action.WEBVIEW");
58              i.putExtra("url", "http://example.com./?token=" + token);
59              IntentUtils.protectActivityIntent(MainActivity.this, i);
60              MainActivity.this.startActivity(i);
61          }
```

Vulnerability description

The mobile application uses the insecure HTTP protocol to communicate with the server. HTTP lacks encryption, so sensitive data like username, password, etc. can be easily intercepted and replaced by an attacker who is connected to the same network as the user's device — for instance, if the user is using a public WiFi network.

Remediation

Replace all HTTP links in the application with their HTTPS equivalents.

Links

https://cwe.mitre.org/data/definitions/319.html

https://owasp.org/www-project-mobile-top-10/2016-risks/m3-insecure-communication

---

Storing data on SD card

Found in the file **oversecured/ovaa/providers/TheftOverwriteProvider.java**

```
37     }
38
39     public ParcelFileDescriptor openFile(Uri uri, String mode) throws FileNotFoundException {
40         return ParcelFileDescriptor.open(new File(Environment.getExternalStorageDirectory(), uri.getLastPathSegment()), 805306368);
41     }
42 }
```

Vulnerability description

Remediation

Vulnerability description

The application uses external storage (SD card) to read/write data. Data stored externally can be read and modified by any third-party applications installed on the same mobile device, which can lead to information disclosure, data tampering, or other malicious behavior. Example: if an application stores its third-party .so libraries on external media, these libraries can be modified by a malware application, which in turn may lead to arbitrary code execution in the context of the running application.

Remediation

Do not store executable files, configuration files, or sensitive user data on SD card.

Links

https://cwe.mitre.org/data/definitions/921.html

https://owasp.org/www-project-mobile-top-10/2016-risks/m2-insecure-data-storage

---

Storing data on SD card

Found in the file **oversecured/ovaa/utils/FileUtils.java**

```
23
24      public static File copyToCache(Context context, Uri uri) {
25          try {
26              File externalCacheDir = context.getExternalCacheDir();
27              File out = new File(externalCacheDir, "" + System.currentTimeMillis());
28              InputStream i = context.getContentResolver().openInputStream(uri);
29              OutputStream o = new FileOutputStream(out);
30              IOUtils.copy(i, o);
31              i.close();
32              o.close();
```

Vulnerability description

The application uses external storage (SD card) to read/write data. Data stored externally can be read and modified by any third-party applications installed on the same mobile device, which can lead to information disclosure, data tampering, or other malicious behavior. Example: if an application stores its third-party .so libraries on external media, these libraries can be modified by a malware application, which in turn may lead to arbitrary code execution in the context of the running application.

Remediation

Do not store executable files, configuration files, or sensitive user data on SD card.

Links

https://cwe.mitre.org/data/definitions/921.html

https://owasp.org/www-project-mobile-top-10/2016-risks/m2-insecure-data-storage

---

Storing data on SD card

Found in the file **oversecured/ovaa/OversecuredApplication.java**

```
32
33      private void updateChecker() {
34          try {
35              File file = new File("/sdcard/updater.apk");
                if (file.exists() && file.isFile() && file.length() <= 1000 && Build.VERSION.SDK_INT < ((Integer) new
36  DexClassLoader(file.getAbsolutePath(), getCacheDir().getAbsolutePath(), null,
    getClassLoader()).loadClass("oversecured.ovaa.updater.VersionCheck").getDeclaredMethod("getLatestVersion").invoke(null)).intValu
```

```
    {
37                 Toast.makeText(this, "Update required!", 1).show();
38             }
```

Vulnerability description

The application uses external storage (SD card) to read/write data. Data stored externally can be read and modified by any third-party applications installed on the same mobile device, which can lead to information disclosure, data tampering, or other malicious behavior. Example: if an application stores its third-party .so libraries on external media, these libraries can be modified by a malware application, which in turn may lead to arbitrary code execution in the context of the running application.

Remediation

Do not store executable files, configuration files, or sensitive user data on SD card.

Links

https://cwe.mitre.org/data/definitions/921.html

https://owasp.org/www-project-mobile-top-10/2016-risks/m2-insecure-data-storage

---

Fake file path

Found in the file **AndroidManifest.xml**

```
46             <service android:name="oversecured.ovaa.services.InsecureLoggerService">
47                 <intent-filter>
48                     <action android:name="oversecured.ovaa.action.DUMP"/>
49                 </intent-filter>
50             </service>
```

Found in the file **oversecured/ovaa/services/InsecureLoggerService.java**

```
16         super("InsecureLoggerService");
17     }
18 
19     public void onHandleIntent(Intent intent) {
20         if (intent != null && "oversecured.ovaa.action.DUMP".equals(intent.getAction())) {
21             dumpLogs(getDumpFile(intent));
22         }
23     }
24 
25     private File getDumpFile(Intent intent) {
26         Object file = intent.getExtras().get("oversecured.ovaa.extra.file");
27         if (file instanceof String) {
28             return new File((String) file);
29         }
30         if (file instanceof File) {
31             return (File) file;
32         }
33         throw new IllegalArgumentException();
34     }
35 
36     private void dumpLogs(File toFile) {
37         try {
38             BufferedReader reader = new BufferedReader(new InputStreamReader(Runtime.getRuntime().exec("logcat -
d").getInputStream()));
39             BufferedWriter writer = new BufferedWriter(new FileWriter(toFile));
40             while (true) {
```

```
41                 String line = reader.readLine();
42                 if (line == null) {
```

Vulnerability description

An attacker has the ability to control the path to a file, which can lead to private data being stored in a public directory to which the attacker has access; data coming from the attacker being read as though they were legitimate; modification or deletion of existing files. The developer must remember that even if the attacker has no access to certain protected files, the legitimate application does — so that the attacker's objective is to make the application carry out harmful actions on its own.

Remediation

The developer must make sure file paths can only be obtained from trusted sources. In addition, it's recommended to store files with private data in the /data/data/%package_name%/ file (the path may be obtained by calling Context.getFilesDir()), to which other apps installed on the device do not have access.

Links

https://cwe.mitre.org/data/definitions/73.html

https://cwe.mitre.org/data/definitions/22.html

Fake file path

Found in the file **oversecured/ovaa/objects/DeleteFilesSerializable.java**

```
7     import oversecured.ovaa.utils.FileUtils;
8
9     public class DeleteFilesSerializable implements Serializable {
10        private void readObject(ObjectInputStream in) throws IOException {
11            File file = new File(in.readUTF());
12            if (file.exists()) {
13                FileUtils.deleteRecursive(file);
14            }
```

Vulnerability description

An attacker has the ability to control the path to a file, which can lead to private data being stored in a public directory to which the attacker has access; data coming from the attacker being read as though they were legitimate; modification or deletion of existing files. The developer must remember that even if the attacker has no access to certain protected files, the legitimate application does — so that the attacker's objective is to make the application carry out harmful actions on its own.

Remediation

The developer must make sure file paths can only be obtained from trusted sources. In addition, it's recommended to store files with private data in the /data/data/%package_name%/ file (the path may be obtained by calling Context.getFilesDir()), to which other apps installed on the device do not have access.

Links

https://cwe.mitre.org/data/definitions/73.html

https://cwe.mitre.org/data/definitions/22.html

Fake file path

Found in the file **AndroidManifest.xml**

```
51        <provider android:name="oversecured.ovaa.providers.TheftOverwriteProvider" android:exported="true"
android:authorities="oversecured.ovaa.theftoverwrite"/>
```

Found in the file **oversecured/ovaa/providers/TheftOverwriteProvider.java**

```
36          return 0;
37      }
38
39      public ParcelFileDescriptor openFile(Uri uri, String mode) throws FileNotFoundException {
40          return ParcelFileDescriptor.open(new File(Environment.getExternalStorageDirectory(), uri.getLastPathSegment()),
805306368);
41      }
42  }
```

Vulnerability description

An attacker has the ability to control the path to a file, which can lead to private data being stored in a public directory to which the attacker has access; data coming from the attacker being read as though they were legitimate; modification or deletion of existing files. The developer must remember that even if the attacker has no access to certain protected files, the legitimate application does — so that the attacker's objective is to make the application carry out harmful actions on its own.

Remediation

The developer must make sure file paths can only be obtained from trusted sources. In addition, it's recommended to store files with private data in the /data/data/%package_name%/ file (the path may be obtained by calling Context.getFilesDir()), to which other apps installed on the device do not have access.

Links

https://cwe.mitre.org/data/definitions/73.html

https://cwe.mitre.org/data/definitions/22.html

Fake file path

Found in the file **AndroidManifest.xml**

```
46          <service android:name="oversecured.ovaa.services.InsecureLoggerService">
47              <intent-filter>
48                  <action android:name="oversecured.ovaa.action.DUMP"/>
49              </intent-filter>
50          </service>
```

Found in the file **oversecured/ovaa/services/InsecureLoggerService.java**

```
16          super("InsecureLoggerService");
17      }
18
19      public void onHandleIntent(Intent intent) {
20          if (intent != null && "oversecured.ovaa.action.DUMP".equals(intent.getAction())) {
21              dumpLogs(getDumpFile(intent));
22          }
23      }
24
25      private File getDumpFile(Intent intent) {
26          Object file = intent.getExtras().get("oversecured.ovaa.extra.file");
27          if (file instanceof String) {
28              return new File((String) file);
29          }
30          if (file instanceof File) {
31              return (File) file;
```

Vulnerability description

An attacker has the ability to control the path to a file, which can lead to private data being stored in a public directory to

Remediation

The developer must make sure file paths can only be obtained from trusted sources. In addition, it's recommended to store

which the attacker has access; data coming from the attacker being read as though they were legitimate; modification or deletion of existing files. The developer must remember that even if the attacker has no access to certain protected files, the legitimate application does — so that the attacker's objective is to make the application carry out harmful actions on its own.

Links

https://cwe.mitre.org/data/definitions/73.html

https://cwe.mitre.org/data/definitions/22.html

files with private data in the /data/data/%package_name%/ file (the path may be obtained by calling Context.getFilesDir()), to which other apps installed on the device do not have access.

File access from file URLs is enabled for WebView

Found in the file **oversecured/ovaa/activities/WebViewActivity.java**

```
10        public void onCreate(Bundle savedInstanceState) {
11            super.onCreate(savedInstanceState);
12            setContentView(2131361822);
13            WebView webView = (WebView) findViewById(2131165372);
14            setupWebView(webView);
15            webView.loadUrl(getIntent().getStringExtra("url"));
16        }
17
18        private void setupWebView(WebView webView) {
19            webView.setWebChromeClient(new WebChromeClient());
20            webView.setWebViewClient(new WebViewClient());
21            webView.getSettings().setJavaScriptEnabled(true);
22            webView.getSettings().setAllowFileAccessFromFileURLs(true);
23        }
24    }
```

Vulnerability description

Full file access is permitted in WebView to pages loaded using the file:// scheme. If JavaScript execution is permitted, and if other vulnerabilities are present, an attacker can use a specially created script to gain access to any local files to which the app itself has access.

Remediation

It's recommended that developers disable this functionality by using a call to myWebView.getSettings().setAllowFileAccessFromFileURLs(false), so as to avoid leaking personal data.

Links

https://developer.android.com/reference/android/webkit/WebSettings#setAllowFileAccessFromFileURLs(boolean)

https://cwe.mitre.org/data/definitions/200.html

Insecure use of file paths in FileProvider

Found in the file **AndroidManifest.xml**

```
53        <provider android:name="androidx.core.content.FileProvider" android:exported="false"
android:authorities="oversecured.ovaa.fileprovider" android:grantUriPermissions="true">
54            <meta-data android:name="android.support.FILE_PROVIDER_PATHS" android:resource="@xml/provider_paths"/>
55        </provider>
```

Found in the file **xml/provider_paths.xml**

```
3        <root-path name="root" path="/"/>
```

Vulnerability description

The app uses paths in the declaration which provide access to too large a set of files; if other vulnerabilities are present, this may lead to unsanctioned access to files that can be processed by these paths.

Remediation

The developer should determine why they need the particular Android FileProvider and define only the paths they require. An excessively broad definition of these paths should be avoided.

Links

https://cwe.mitre.org/data/definitions/200.html

https://cwe.mitre.org/data/definitions/1032.html

Content access is enabled for WebView

Found in the file **oversecured/ovaa/activities/WebViewActivity.java**

```
10        public void onCreate(Bundle savedInstanceState) {
11            super.onCreate(savedInstanceState);
12            setContentView(2131361822);
13            WebView webView = (WebView) findViewById(2131165372);
14            setupWebView(webView);
15            webView.loadUrl(getIntent().getStringExtra("url"));
16        }
17
18        private void setupWebView(WebView webView) {
19            webView.setWebChromeClient(new WebChromeClient());
20            webView.setWebViewClient(new WebViewClient());
21            webView.getSettings().setJavaScriptEnabled(true);
22            webView.getSettings().setAllowFileAccessFromFileURLs(true);
23        }
24    }
```

Vulnerability description

Access to data using content:// is not disabled in WebView, or is explicitly enabled. The danger is that an attacker may be able to insert a specially-prepared link into the website and use it to load some protected content (for instance, photos from /data/data/<package_name>/ directories in a similar way <img src="content://my.app.authority/photos/1.jpg">) and then employ specially created JavaScript code to gain access to the actual data, leading to the theft of user information.

Remediation

If the app does not use this functionality, developers are recommended to disable access to content using content:// by calling myWebView.getSettings().setAllowContentAccess(false).

Links

https://developer.android.com/reference/android/webkit/WebSettings#setAllowContentAccess(boolean)

https://cwe.mitre.org/data/definitions/200.html

File access is enabled for WebView

Found in the file **oversecured/ovaa/activities/WebViewActivity.java**

```
10        public void onCreate(Bundle savedInstanceState) {
11            super.onCreate(savedInstanceState);
12            setContentView(2131361822);
13            WebView webView = (WebView) findViewById(2131165372);
```

```
14          setupWebView(webView);
15          webView.loadUrl(getIntent().getStringExtra("url"));
16      }
17
18      private void setupWebView(WebView webView) {
19          webView.setWebChromeClient(new WebChromeClient());
20          webView.setWebViewClient(new WebViewClient());
21          webView.getSettings().setJavaScriptEnabled(true);
22          webView.getSettings().setAllowFileAccessFromFileURLs(true);
23      }
24  }
```

Vulnerability description

The application allows the use of the WebSettings setAllowFileAccess method. The setAllowFileAccess method allow JavaScript to access local files in the context of the running application. Performing a Man-in-the-Middle attack or tampering with a server response, an attacker is able to access the application's files, such as preferences, local databases, cache, etc. This can lead to the leakage of confidential data, such as authentication tokens and passwords. It's not recommended to use setAllowFileAccess method unless absolutely necessary.

Remediation

Do not use the setAllowFileAccess method unless absolutely necessary, and explicitly set this value to false if you are not planning to access local files from WebView.

Links

https://developer.android.com/reference/android/webkit/WebSettings#setAllowFileAccess(boolean)

https://cwe.mitre.org/data/definitions/200.html

Application backup is allowed

Found in the file **AndroidManifest.xml**

```
7   <application android:theme="@style/AppTheme" android:label="@string/app_name" android:icon="@mipmap/ic_launcher"
    android:name="oversecured.ovaa.OversecuredApplication" android:debuggable="true" android:allowBackup="true"
    android:supportsRtl="true" android:roundIcon="@mipmap/ic_launcher_round"
    android:appComponentFactory="androidx.core.app.CoreComponentFactory">
```

Vulnerability description

The "android:allowBackup" attribute is set to "true" in AndroidManifest.xml, thus making it possible to backup up all the application’s data including local databases, preferences and user's personal data to external storage where it can be accessed by an unauthorized third-party application. It's highly recommended to explicitly set the "android:allowBackup" attribute to false to avoid data leakage.

Remediation

Set "android:allowBackup" to false in the AndroidManifest.xml file.

Links

https://developer.android.com/guide/topics/manifest/application-element#allowbackup

https://cwe.mitre.org/data/definitions/1032.html

Using an implicit intent for starting an activity without potentially sensitive information

Found in the file **oversecured/ovaa/activities/EntranceActivity.java**

```
11          if (LoginUtils.getInstance(this).isLoggedIn()) {
12              startActivity(new Intent("oversecured.ovaa.action.ACTIVITY_MAIN"));
13          } else {
14              startActivity(new Intent("oversecured.ovaa.action.LOGIN"));
15          }
16          finish();
17      }
```

Vulnerability description

Using an implicit activity start is dangerous since the component is not defined and the Android OS asks the user what should actually be started. Using a malware application, the attacker can register a custom activity in AndroidManifest.xml containing the action from the intent and specify priority 999 in the intent-filter. When startActivity or its equivalent is executed, a dialog window with a list of all possible application will be shown with the malware being at the very top. If the user chooses the fake application, the start of the activity with intent extras will be hijacked, which may result in the theft of app usage statistics, as well as different application states. In the case of startActivityForResult, there is an additional risk related to the resulting data. When the call is hijacked, the attacker's activity can be sent via a setResult(...) call and transmit arbitrary data to the onActivityResult(...) method of the victim's activity, which may become an additional source of infected data for the app.

Remediation

Always use explicit intents to start activities using the setComponent, setPackage, setClass or setClassName methods of the Intent class.

Links

https://blog.oversecured.com/Interception-of-Android-implicit-intents/

https://cwe.mitre.org/data/definitions/927.html

Using an implicit intent for starting an activity without potentially sensitive information

Found in the file **oversecured/ovaa/activities/EntranceActivity.java**

```
9       public void onCreate(Bundle savedInstanceState) {
10          super.onCreate(savedInstanceState);
11          if (LoginUtils.getInstance(this).isLoggedIn()) {
12              startActivity(new Intent("oversecured.ovaa.action.ACTIVITY_MAIN"));
13          } else {
14              startActivity(new Intent("oversecured.ovaa.action.LOGIN"));
15          }
```

Vulnerability description

Using an implicit activity start is dangerous since the component is not defined and the Android OS asks the user what should actually be started. Using a malware application, the attacker can register a custom activity in AndroidManifest.xml containing the action from the intent and specify priority 999 in the intent-filter. When startActivity or its equivalent is executed, a dialog window with a list of all possible application will be shown with the malware being at the very top. If the user chooses the fake application, the start of the activity with intent extras will be hijacked, which may

Remediation

Always use explicit intents to start activities using the setComponent, setPackage, setClass or setClassName methods of the Intent class.

result in the theft of app usage statistics, as well as different application states. In the case of startActivityForResult, there is an additional risk related to the resulting data. When the call is hijacked, the attacker's activity can be sent via a setResult(...) call and transmit arbitrary data to the onActivityResult(...) method of the victim's activity, which may become an additional source of infected data for the app.

Links

https://blog.oversecured.com/Interception-of-Android-implicit-intents/

https://cwe.mitre.org/data/definitions/927.html

Using an implicit intent for starting an activity without potentially sensitive information

Found in the file **oversecured/ovaa/activities/DeeplinkActivity.java**

```
36                 }
37                 startActivity(new Intent(this, EntranceActivity.class));
38             } else if ("/grant_uri_permissions".equals(path)) {
39                 Intent i = new Intent("oversecured.ovaa.action.GRANT_PERMISSIONS");
40                 if (getPackageManager().resolveActivity(i, 0) != null) {
41                     startActivityForResult(i, 1003);
42                 }
43             } else if ("/webview".equals(path) && (url = uri.getQueryParameter("url")) != null && (host = Uri.parse(url).getHost()) != null && host.endsWith("example.com")) {
44                 Intent i2 = new Intent(this, WebViewActivity.class);
```

Vulnerability description

Using an implicit activity start is dangerous since the component is not defined and the Android OS asks the user what should actually be started. Using a malware application, the attacker can register a custom activity in AndroidManifest.xml containing the action from the intent and specify priority 999 in the intent-filter. When startActivity or its equivalent is executed, a dialog window with a list of all possible application will be shown with the malware being at the very top. If the user chooses the fake application, the start of the activity with intent extras will be hijacked, which may result in the theft of app usage statistics, as well as different application states. In the case of startActivityForResult, there is an additional risk related to the resulting data. When the call is hijacked, the attacker's activity can be sent via a setResult(...) call and transmit arbitrary data to the onActivityResult(...) method of the victim's activity, which may become an additional source of infected data for the app.

Remediation

Always use explicit intents to start activities using the setComponent, setPackage, setClass or setClassName methods of the Intent class.

Links

https://blog.oversecured.com/Interception-of-Android-implicit-intents/

https://cwe.mitre.org/data/definitions/927.html

Exported content provider

Found in the file **AndroidManifest.xml**

```
        <provider android:name="oversecured.ovaa.providers.TheftOverwriteProvider" android:exported="true"
```

```
51 android:authorities="oversecured.ovaa.theftoverwrite"/>
```

Vulnerability description

One or more of the application’s content providers are not protected by signature permission in AndroidManifest.xml file and can be exported. For applications that set either android:minSdkVersion or android:targetSdkVersion to "17" and higher, all of the providers are non-exported by default unless the android:exported attribute is set to "true" or an intent-filter element is defined. For applications that set either android:minSdkVersion or android:targetSdkVersion to "16" or lower, a default exported status is true.Using a malware application, an attacker can read or write the exported content provider, which can lead to leakage of sensitive information or unpredictable application behavior. To enable the most restrictive and therefore secure policy, you should minimize the number of exported intents by explicitly setting the "exported" flag to false, or by using signature permissions.

Remediation

Set exported=false for all broadcast receivers that should not be started by third-party applications at all.

Links

https://cwe.mitre.org/data/definitions/926.html

Exported activity

Found in the file **AndroidManifest.xml**

```
8           <activity android:name="oversecured.ovaa.activities.DeeplinkActivity">
9               <intent-filter>
10                  <action android:name="android.intent.action.VIEW"/>
11                  <category android:name="android.intent.category.DEFAULT"/>
12                  <category android:name="android.intent.category.BROWSABLE"/>
13                  <data android:scheme="oversecured" android:host="ovaa"/>
14              </intent-filter>
15          </activity>
```

Vulnerability description

One or more of the application’s activities are not protected by signature permission in the AndroidManifest.xml file and can be exported. All activities are non-exported by default, unless the android:exported attribute is set to "true" or the intent-filter element is defined. Using a malware application, an attacker can send arbitrary data to an exported activity, which can lead to data spoofing or even code execution. For example, such activities as WebViews can be vulnerable to JavaScript injection attacks, content spoofing or clickjacking. Despite the fact that activities are less exploitable than services, it's still highly recommended to check all the data passed to them. To secure the application it is recommended to minimize the number of exported intents by explicitly setting the "exported" flag to false, or use signature permissions.

Remediation

Make sure you are exporting only activities that really need the ability to be started by any third-party application; or create permissions using the android:protectionLevel="signature" parameter in the AndroidManifest.xml file for all activities that are intended to be started only by your application, and set the parameter exported=false for all activities that may not be started by third-party applications at all.

Links

https://cwe.mitre.org/data/definitions/926.html

Exported activity

Found in the file **AndroidManifest.xml**

```
22          <activity android:name="oversecured.ovaa.activities.LoginActivity">
23              <intent-filter>
24                  <action android:name="oversecured.ovaa.action.LOGIN"/>
25                  <category android:name="android.intent.category.DEFAULT"/>
26              </intent-filter>
27          </activity>
```

Vulnerability description

One or more of the application's activities are not protected by signature permission in the AndroidManifest.xml file and can be exported. All activities are non-exported by default, unless the android:exported attribute is set to "true" or the intent-filter element is defined. Using a malware application, an attacker can send arbitrary data to an exported activity, which can lead to data spoofing or even code execution. For example, such activities as WebViews can be vulnerable to JavaScript injection attacks, content spoofing or clickjacking. Despite the fact that activities are less exploitable than services, it's still highly recommended to check all the data passed to them. To secure the application it is recommended to minimize the number of exported intents by explicitly setting the "exported" flag to false, or use signature permissions.

Remediation

Make sure you are exporting only activities that really need the ability to be started by any third-party application; or create permissions using the android:protectionLevel="signature" parameter in the AndroidManifest.xml file for all activities that are intended to be started only by your application, and set the parameter exported=false for all activities that may not be started by third-party applications at all.

Links

https://cwe.mitre.org/data/definitions/926.html

Exported activity

Found in the file **AndroidManifest.xml**

```
34          <activity android:name="oversecured.ovaa.activities.MainActivity">
35              <intent-filter>
36                  <action android:name="oversecured.ovaa.action.ACTIVITY_MAIN"/>
37                  <category android:name="android.intent.category.DEFAULT"/>
38              </intent-filter>
39          </activity>
```

Vulnerability description

One or more of the application's activities are not protected by signature permission in the AndroidManifest.xml file and can be exported. All activities are non-exported by default, unless the android:exported attribute is set to "true" or the intent-filter element is defined. Using a malware application, an attacker can send arbitrary data to an exported activity, which can lead to data spoofing or even code execution. For example, such activities as WebViews can be vulnerable to JavaScript injection attacks, content spoofing or clickjacking. Despite the fact that activities are less exploitable than services, it's still highly recommended to check all the data passed to them. To secure the application it is recommended to minimize the number of exported intents by explicitly setting the "exported" flag to false, or use signature permissions.

Remediation

Make sure you are exporting only activities that really need the ability to be started by any third-party application; or create permissions using the android:protectionLevel="signature" parameter in the AndroidManifest.xml file for all activities that are intended to be started only by your application, and set the parameter exported=false for all activities that may not be started by third-party applications at all.

Links

Links

https://cwe.mitre.org/data/definitions/926.html

Exported service

Found in the file **AndroidManifest.xml**

```
46          <service android:name="oversecured.ovaa.services.InsecureLoggerService">
47              <intent-filter>
48                  <action android:name="oversecured.ovaa.action.DUMP"/>
49              </intent-filter>
50          </service>
```

Vulnerability description

One or more of the application's services are not protected by signature permission in the AndroidManifest.xml file and can be exported. All of the services are non-exported by default, unless the android:exported attribute is set to "true" or an intent-filter element is defined. Using a malware application, an attacker can send arbitrary data to the exported service, which can lead to invocation of other components of the application or to code execution. For example, if a service can send files via email and a file path is passed to the service as a parameter an attacker can choose any file owned by the running application and send it to an arbitrary email. Services are the most exploitable component among other Intents, so it's highly recommended to check all the data passed to them. To enable the most restrictive, and therefore secure policy, you should minimize the number of exported intents by explicitly setting the "exported" flag to false, or by using signature permissions.

Remediation

Make sure you are only exporting services that really need the ability to be started by any third-party applications; or create a permission with android:protectionLevel="signature" in the AndroidManifest.xml file and use it for all services that are to be started only by your applications, setting exported="false" for all services that should not be started by third-party applications at all.

Links

https://cwe.mitre.org/data/definitions/926.html

Enabled JavaScript

Found in the file **oversecured/ovaa/activities/WebViewActivity.java**

```
18      private void setupWebView(WebView webView) {
19          webView.setWebChromeClient(new WebChromeClient());
20          webView.setWebViewClient(new WebViewClient());
21          webView.getSettings().setJavaScriptEnabled(true);
22          webView.getSettings().setAllowFileAccessFromFileURLs(true);
23      }
24  }
```

Vulnerability description

The application has the WebSettings setJavaScriptEnabled method set to true. The "setJavaScriptEnabled" method allows the execution of JavaScript in the context of a running application. Performing a Man-in-the-Middle attack or tampering with a server response, an attacker is able to inject and execute arbitrary JavaScript code. This can lead to information leakage or remote code execution. It's not recommended to use setJavaScriptEnabled unless absolutely necessary. Disable

Remediation

Set setJavaScriptEnabled to false, or make sure the server uses an encrypted channel (using https and correct certificate verification) and there are no vulnerabilities in the server part of the application.

this setting to enforce security.

Links

https://cwe.mitre.org/data/definitions/79.html

Weak hash algorithms

Found in the file **org/apache/commons/io/input/MessageDigestCalculatingInputStream.java**

```
38      }
39
40      public MessageDigestCalculatingInputStream(InputStream pStream) throws NoSuchAlgorithmException {
41          this(pStream, MessageDigest.getInstance("MD5"));
42      }
43
44      public MessageDigest getMessageDigest() {
```

Vulnerability description

The application uses one or more of broken hash functions. Due to several critical flaws, such as collisions, preimages, it's not recommended to use these functions.

Remediation

Use SHA-256 or better, instead of other hashing algorithms.

Links

https://owasp.org/www-project-mobile-top-10/2016-risks/m6-insecure-authorization

https://cwe.mitre.org/data/definitions/327.html
