AppSweep
GUARDSQUARE

BUILD REPORT SUMMARY

appsweep.guardsquare.com

Leveraging AppSweep helps you identify and fix security issues in your code and dependencies with actionable recommendations and insights that will help you build more secure mobile apps.

11 PIVAA

com.htbridge.pivaa · Analysed Nov 15 2022, 14:01

Version

1.0

App size

3.90 MB

Commit Hash

b125d37396f822fa2ff196a00a6658b9b0bb0d18

App composition

Bytecode size:	4.35 MB
Number of Java classes:	3294

Analysis duration

1m 0s

Obfuscation mapping

✕ Not Provided

Tags

No tags provided.

Issue Summary

5 **High Severity Issues**
5 internal · 0 in dependencies

13 **Medium Severity Issues**
13 internal · 0 in dependencies

1 **Low Severity Issues**
1 internal · 1 in dependencies

39 **Dependencies**
38 transitive dependencies

High ● **Android manifest attribute `android:debuggable="true"` is set**

The attribute `android:debuggable` is set to `true` in the app's manifest. This means that your app can be debugged using Java Wired Debugging Protocol. Using JWDP, it is possible to gain full access to the Java process and execute arbitrary code in the context of a debuggable app.
Releasing an app with this flag set can lead to leakage of sensitive information and leaves the app vulnerable to debugging.
Note that setting `android:debuggable` to `false` is necessary to prevent debugging, but is not sufficient. An adversary can still connect a debugger and use it to reverse-engineer or tamper with the app's behaviour.

Recommendations

Ensure that the flag `android:debuggable` is set to `false` in your AndroidManifest.xml when building for release.

Fix with DexGuard

Setting the attribute `android:debuggable` to `"false"` is necessary to prevent debugging, but is not sufficient. An adversary can still connect a debugger and use it to reverse-engineer or tamper with the app's behaviour.
Enable debugging protection in DexGuard using this configuration line:

```
-raspchecks debug
```

or consult the RASP section in your Dexguard Manual to learn more.

External Links

OWASP recommendations regarding debuggable flags

1 Finding

- AndroidManifest.xml

High

User input from `Editable EditText.getText()` is used in an injection-vulnerable database method via `Cursor SQLiteDatabase.rawQuery(String,String[])`

The app contains vulnerable method calls to database APIs that allow for an SQL injection. This means that attackers can manipulate queries to the database, potentially leading to information leakage, login bypasses, or loss of data.

Recommendations

Refactor your application to use parametrized queries.

External Links

OWASP recommendations regarding Injection Flaws

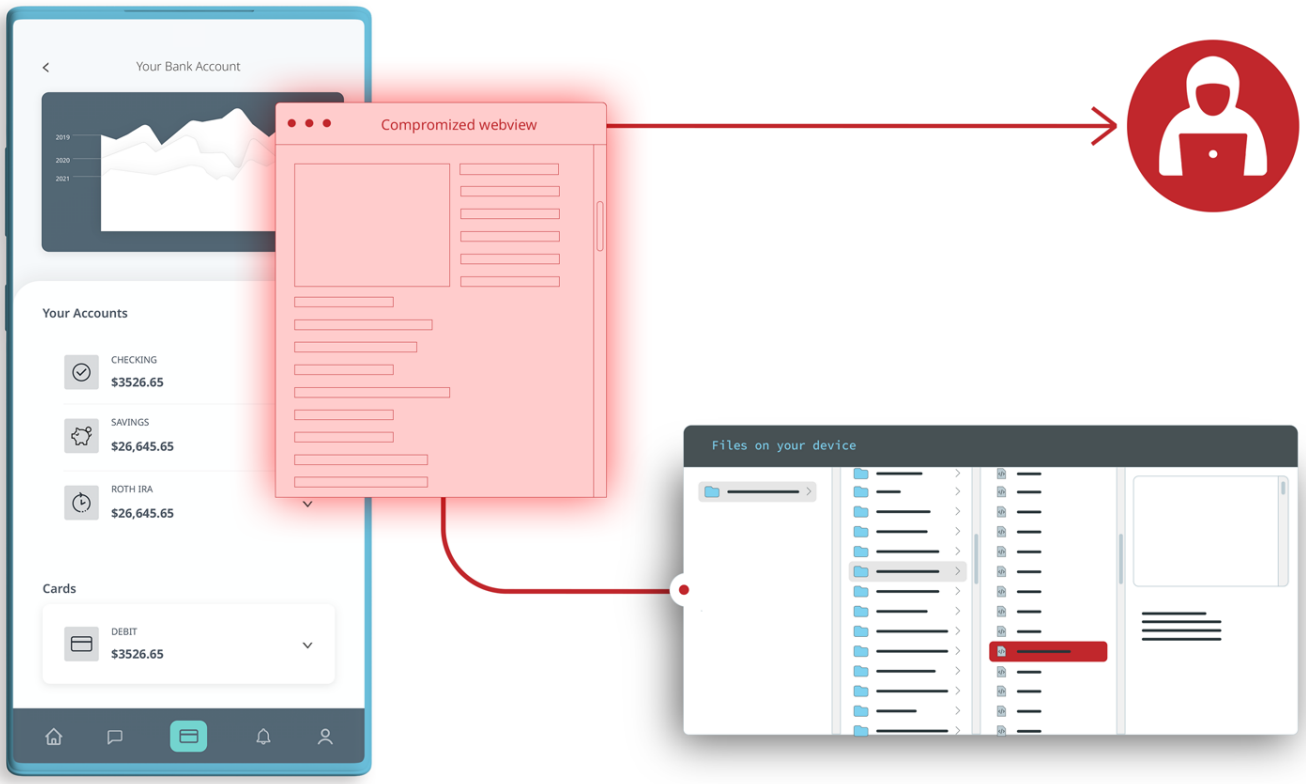
1 Finding

- com.htbridge.pivaa.DatabaseActivity$2:86
  com.htbridge.pivaa.handlers.database.DatabaseHelper:220

High

The app enables dangerous file access via `setAllowUniversalAccessFromFileURLs`

Your app uses the API setAllowUniversalAccessFromFileURLs, to enable dangerous file access. This method was deprecated and is not considered secure.
If local file access is enabled in a WebView, an attacker who gains access to that WebView (for example through a MitM attack), can gain access to user's files on the device through the vulnerable app.



Recommendations

Use `androidx#webkit#WebViewAssetLoader` to load file content securely.

External Links

WebSettings | Android Developers

2 Findings

- **setAllowUniversalAccessFromFileURLs**
  com.htbridge.pivaa.BroadcastReceiverActivity:53

- **setAllowUniversalAccessFromFileURLs**
  com.htbridge.pivaa.WebviewActivity:52

High

The class `com.htbridge.pivaa.handlers.API$1` implementing `HostnameVerifier` does not check TLS certificate validity correctly

A `HostnameVerifier` needs to check that an otherwise valid TLS certificate was issued specifically for use with the exact URL that is currently being accessed. If this step is skipped, a malicious actor can use their legitimately received certificate for “malicious.com” to pretend they are actually “google.com”.
Your class `com.htbridge.pivaa.handlers.API$1` always returns `true` for this check, making it easy for attackers to impersonate any server as part of a man-in-the-middle (MitM) attack.

Recommendations

The default implementation should be preferred over a custom `HostnameVerifier`.

External Links

CWE-295: Improper Certificate Validation

How attackers can exploit insecure TLS configurations in Android apps

OWASP recommendations regarding TLS certificates

1 Finding

- com.htbridge.pivaa.handlers.API$1

High

The class `com.htbridge.pivaa.handlers.API$2` implementing `X509TrustManager` does not check TLS certificate validity correctly

`X509TrustManager.checkServerTrusted(...)` needs to throw a `java.security.cert.CertificateException` if the certificate chain cannot be trusted. The class `com.htbridge.pivaa.handlers.API$2` never does this, so attackers can just provide an arbitrary certificate as part of a man-in-the-middle (MitM) attack and your app will always accept it.
Please note that for most use cases you won't need to create a custom `X509TrustManager`. If you really need to support situations where your server uses a non-standard certificate that is not accepted by the default trust manager, refer to our blog post to learn more about how to securely support this case. In this post you will also find suggestions on how to further improve TLS based security in your app, by applying certificate pinning or certificate transparency, another popular reason why developers choose to use custom trust managers.

Recommendations

Use the default `X509TrustManager` whenever possible. If you have to use a custom implementation, make sure to properly verify the certificate chain.

External Links

CWE-295: Improper Certificate Validation

Google suggestions on fixing unsafe X509TrustManager implementations

How to securely implement TLS certificate checking in Android apps

How attackers can exploit insecure TLS configurations in Android apps

OWASP recommendations regarding malfunctioning X509 trust managers

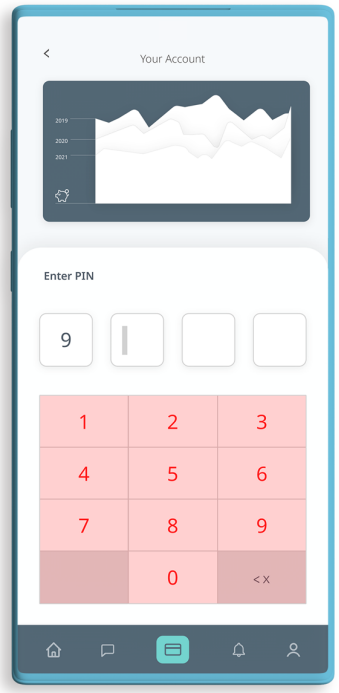
1 Finding

- com.htbridge.pivaa.handlers.API$2

Medium

Elements of `BroadcastReceiverActivity` are not protected against tapjacking

Tapjacking is a technique that allows an attacker to capture the taps in your app (for example, on a virtual pin-pad), or trick users into making taps without their consent (for example, switching off an important security setting). Tapjacking protection is especially important for security relevant parts of the app like pin or password entry.



The essence of the attack is that a malicious app places a window over your app.
If the attacker wants to capture user clicks, that window will be transparent. The overlay window gets an opportunity to learn about the taps made in your app without the device user being aware.
If the attacker wants to trick the user into clicking something in your app, the window will be opaque with fake controls lying exactly over the corresponding controls in your app.
For instance, as seen in the image to the left, placing transparent overlays over each button on a pinpad allows an attacker to capture users' pin codes.

Recommendations

Add `filterTouchesWhenObscured="true"` to the relevant view elements in the respective layout xml files, or set the protection programmatically.

External Links

Android Developer - View Security

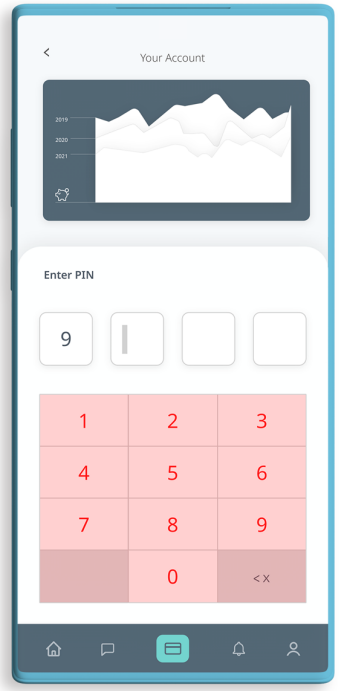
OWASP recommendations regarding tapjacking

1 Finding

- **Button**
  res/layout/content_broadcast_receiver.xml

Medium ● **Elements of `LoadCodeActivity` are not protected against tapjacking**

Tapjacking is a technique that allows an attacker to capture the taps in your app (for example, on a virtual pin-pad), or trick users into making taps without their consent (for example, switching off an important security setting). Tapjacking protection is especially important for security relevant parts of the app like pin or password entry.



The essence of the attack is that a malicious app places a window over your app.
If the attacker wants to capture user clicks, that window will be transparent. The overlay window gets an opportunity to learn about the taps made in your app without the device user being aware.
If the attacker wants to trick the user into clicking something in your app, the window will be opaque with fake controls lying exactly over the corresponding controls in your app.
For instance, as seen in the image to the left, placing transparent overlays over each button on a pinpad allows an attacker to capture users' pin codes.

Recommendations

Add `filterTouchesWhenObscured="true"` to the relevant view elements in the respective layout xml files, or set the protection programmatically.

External Links

Android Developer - View Security

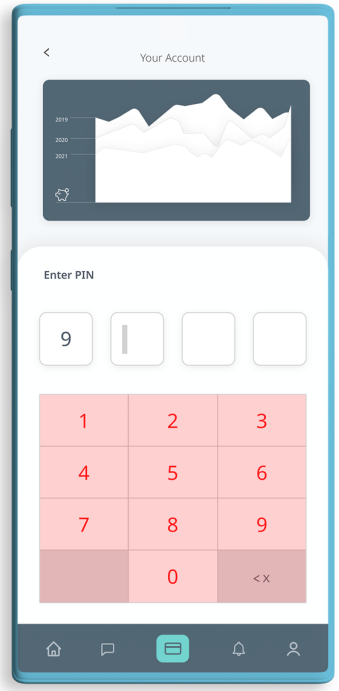
OWASP recommendations regarding tapjacking

1 Finding

- **Button**
  res/layout/content_load_code.xml

Medium ● **Elements of `EncryptionActivity` are not protected against tapjacking**

Tapjacking is a technique that allows an attacker to capture the taps in your app (for example, on a virtual pin-pad), or trick users into making taps without their consent (for example, switching off an important security setting). Tapjacking protection is especially important for security relevant parts of the app like pin or password entry.



The essence of the attack is that a malicious app places a window over your app.
If the attacker wants to capture user clicks, that window will be transparent. The overlay window gets an opportunity to learn about the taps made in your app without the device user being aware.
If the attacker wants to trick the user into clicking something in your app, the window will be opaque with fake controls lying exactly over the corresponding controls in your app.
For instance, as seen in the image to the left, placing transparent overlays over each button on a pinpad allows an attacker to capture users' pin codes.

Recommendations

Add `filterTouchesWhenObscured="true"` to the relevant view elements in the respective layout xml files, or set the protection programmatically.

External Links

Android Developer - View Security

OWASP recommendations regarding tapjacking

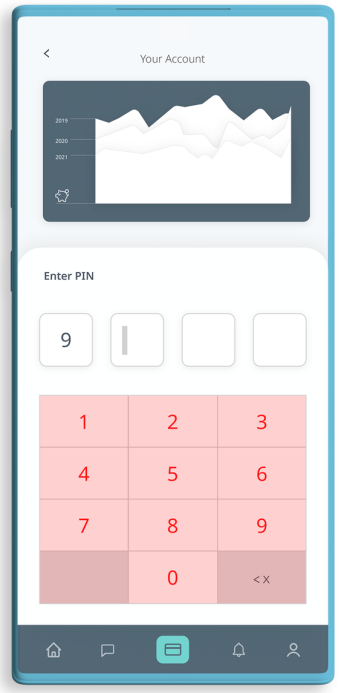
1 Finding

- **Button**
  res/layout/content_encryption.xml

Medium

Elements of `ContentProviderActivity` are not protected against tapjacking

Tapjacking is a technique that allows an attacker to capture the taps in your app (for example, on a virtual pin-pad), or trick users into making taps without their consent (for example, switching off an important security setting). Tapjacking protection is especially important for security relevant parts of the app like pin or password entry.



The essence of the attack is that a malicious app places a window over your app.
If the attacker wants to capture user clicks, that window will be transparent. The overlay window gets an opportunity to learn about the taps made in your app without the device user being aware.
If the attacker wants to trick the user into clicking something in your app, the window will be opaque with fake controls lying exactly over the corresponding controls in your app.
For instance, as seen in the image to the left, placing transparent overlays over each button on a pinpad allows an attacker to capture users' pin codes.

Recommendations

Add `filterTouchesWhenObscured="true"` to the relevant view elements in the respective layout xml files, or set the protection programmatically.

External Links

Android Developer - View Security

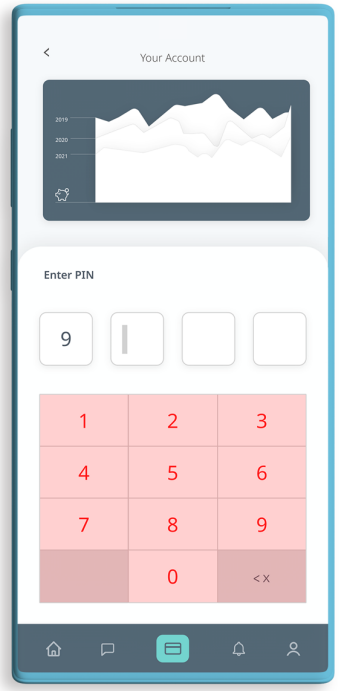
OWASP recommendations regarding tapjacking

1 Finding

- **Button**
  res/layout/content_provider.xml

Medium ● **Elements of `MainActivity` are not protected against tapjacking**

Tapjacking is a technique that allows an attacker to capture the taps in your app (for example, on a virtual pin-pad), or trick users into making taps without their consent (for example, switching off an important security setting). Tapjacking protection is especially important for security relevant parts of the app like pin or password entry.



The essence of the attack is that a malicious app places a window over your app.
If the attacker wants to capture user clicks, that window will be transparent. The overlay window gets an opportunity to learn about the taps made in your app without the device user being aware.
If the attacker wants to trick the user into clicking something in your app, the window will be opaque with fake controls lying exactly over the corresponding controls in your app.
For instance, as seen in the image to the left, placing transparent overlays over each button on a pinpad allows an attacker to capture users' pin codes.

Recommendations

Add `filterTouchesWhenObscured="true"` to the relevant view elements in the respective layout xml files, or set the protection programmatically.

External Links

Android Developer - View Security

OWASP recommendations regarding tapjacking

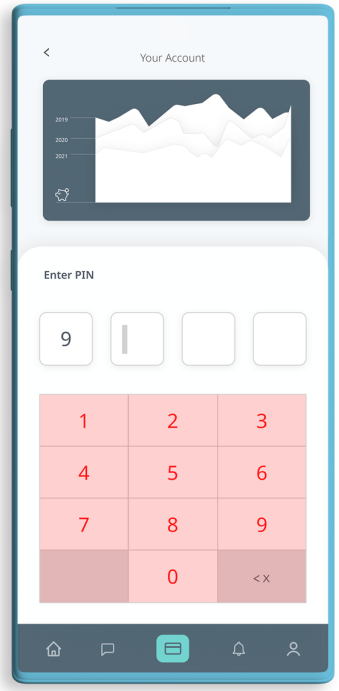
2 Findings

- **Button**
  res/layout/activity_main.xml

- **com.google.android.material.internal.CheckableImageButton**
  res/layout/design_text_input_start_icon.xml

Medium ● **Elements of `DatabaseActivity` are not protected against tapjacking**

Tapjacking is a technique that allows an attacker to capture the taps in your app (for example, on a virtual pin-pad), or trick users into making taps without their consent (for example, switching off an important security setting). Tapjacking protection is especially important for security relevant parts of the app like pin or password entry.



The essence of the attack is that a malicious app places a window over your app.
If the attacker wants to capture user clicks, that window will be transparent. The overlay window gets an opportunity to learn about the taps made in your app without the device user being aware.
If the attacker wants to trick the user into clicking something in your app, the window will be opaque with fake controls lying exactly over the corresponding controls in your app.
For instance, as seen in the image to the left, placing transparent overlays over each button on a pinpad allows an attacker to capture users' pin codes.

Recommendations

Add `filterTouchesWhenObscured="true"` to the relevant view elements in the respective layout xml files, or set the protection programmatically.

External Links

Android Developer - View Security

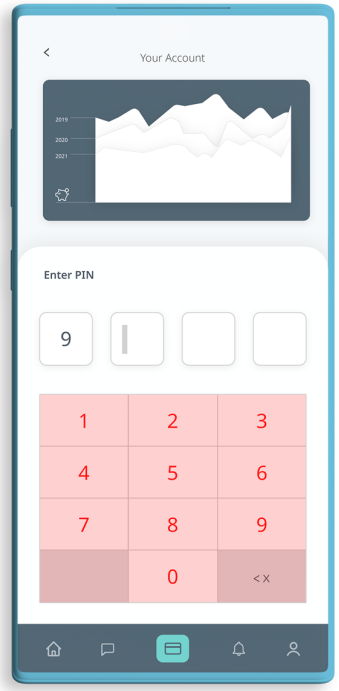
OWASP recommendations regarding tapjacking

1 Finding

- **Button**
  res/layout/content_database.xml

Medium ● **Elements of `ServiceActivity` are not protected against tapjacking**

Tapjacking is a technique that allows an attacker to capture the taps in your app (for example, on a virtual pin-pad), or trick users into making taps without their consent (for example, switching off an important security setting). Tapjacking protection is especially important for security relevant parts of the app like pin or password entry.



The essence of the attack is that a malicious app places a window over your app.
If the attacker wants to capture user clicks, that window will be transparent. The overlay window gets an opportunity to learn about the taps made in your app without the device user being aware.
If the attacker wants to trick the user into clicking something in your app, the window will be opaque with fake controls lying exactly over the corresponding controls in your app.
For instance, as seen in the image to the left, placing transparent overlays over each button on a pinpad allows an attacker to capture users' pin codes.

Recommendations

Add `filterTouchesWhenObscured="true"` to the relevant view elements in the respective layout xml files, or set the protection programmatically.

External Links

Android Developer - View Security

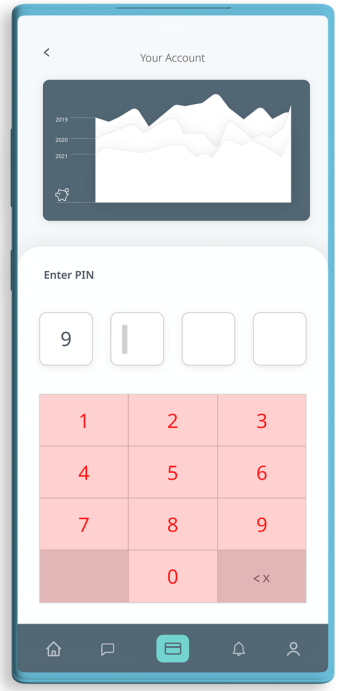
OWASP recommendations regarding tapjacking

1 Finding

- **Button**
  res/layout/content_service.xml

Medium ● **Elements of `SerializeActivity` are not protected against tapjacking**

Tapjacking is a technique that allows an attacker to capture the taps in your app (for example, on a virtual pin-pad), or trick users into making taps without their consent (for example, switching off an important security setting). Tapjacking protection is especially important for security relevant parts of the app like pin or password entry.



The essence of the attack is that a malicious app places a window over your app.
If the attacker wants to capture user clicks, that window will be transparent. The overlay window gets an opportunity to learn about the taps made in your app without the device user being aware.
If the attacker wants to trick the user into clicking something in your app, the window will be opaque with fake controls lying exactly over the corresponding controls in your app.
For instance, as seen in the image to the left, placing transparent overlays over each button on a pinpad allows an attacker to capture users' pin codes.

Recommendations

Add `filterTouchesWhenObscured="true"` to the relevant view elements in the respective layout xml files, or set the protection programmatically.

External Links

Android Developer - View Security

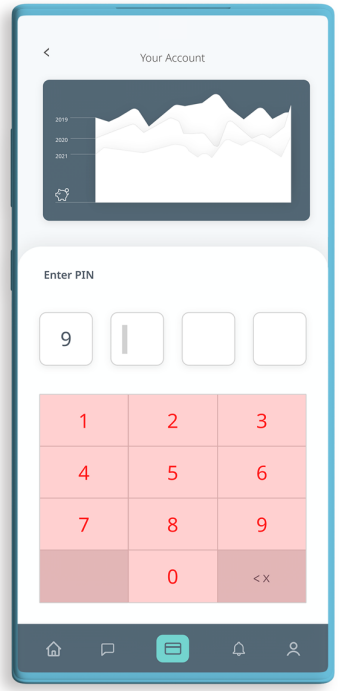
OWASP recommendations regarding tapjacking

1 Finding

- **Button**
  res/layout/content_serialize.xml

Medium ● **Elements of `WebviewActivity` are not protected against tapjacking**

Tapjacking is a technique that allows an attacker to capture the taps in your app (for example, on a virtual pin-pad), or trick users into making taps without their consent (for example, switching off an important security setting). Tapjacking protection is especially important for security relevant parts of the app like pin or password entry.



The essence of the attack is that a malicious app places a window over your app.
If the attacker wants to capture user clicks, that window will be transparent. The overlay window gets an opportunity to learn about the taps made in your app without the device user being aware.
If the attacker wants to trick the user into clicking something in your app, the window will be opaque with fake controls lying exactly over the corresponding controls in your app.
For instance, as seen in the image to the left, placing transparent overlays over each button on a pinpad allows an attacker to capture users' pin codes.

Recommendations

Add `filterTouchesWhenObscured="true"` to the relevant view elements in the respective layout xml files, or set the protection programmatically.

External Links

Android Developer - View Security

OWASP recommendations regarding tapjacking

1 Finding

- **Button**
  res/layout/content_webview.xml

Medium

Outdated protocol version TLS enabled

Initializing SSLContext with a generic or old TLS version may enable outdated and insecure communication protocols on devices with API level less than 26.

Recommendations

Explicitly initialize SSLContext with TLSv1.2 or TLSv1.3. Always check your server configuration to only use TLSv1.2 or newer versions.

External Links

OWASP recommendations regarding outdated TLS certificates

1 Finding

- com.htbridge.pivaa.handlers.API:53

Medium

The apk allows cleartext communication

Cleartext communication should be disabled as it allows attackers to spy on your traffic. Android allows developers to configure their cleartext communication through the manifest or by importing a network security configuration file. Beware, if you're uploading a network configuration file, the values in the network configuration file will always override the ones in the manifest (even if the flag is not set in the network security file the default value of that flag will override the values set in the manifest). Plus, below Android 28 cleartext is enabled by default and starting from Android 28 the flag is disabled by default. In order to disable cleartext communication in Android 27 and below we must the following:
If a security file is being imported, the tag `<base-config cleartextTrafficPermitted="false">` is set.
If no security file is imported, the flag `android:usesCleartextTraffic="false"` is set inside the `<application>` tag in the manifest.

Recommendations

Ensure that the flag `cleartextTrafficPermitted` is set to `false` in your `base-config` tag of the security network file. If the app doesn't contain a security file ensure `android:usesCleartextTraffic` is set to `false` in the `application` tag inside the manifest.

External Links

MSTG: Verifying Data Encryption on the Network

Android Documentation

1 Finding

- AndroidManifest.xml

Medium

Risk of AES misconfiguration for cipher `AES/ECB`

The application is using AES in mode `AES/ECB`, which is likely insecure or misconfigured. Using a misconfigured AES mode potentially allows an attacker to decrypt the encrypted data without a key. Possible misconfigurations could be a weak cipher mode or the usage of a predictable IV. To achieve confidentiality, the authenticated encryption mode `AES_256/GCM/NoPadding` is recommended.

Recommendations

Use `AES_256/GCM/NoPadding` as encryption mode for AES

External Links

MSTG-CRYPTO-3: Common Cryptographic Configuration Issues

OWASP recommendations regarding secure AES encryption modes

2 Findings

- com.htbridge.pivaa.handlers.Encryption:90
- com.htbridge.pivaa.handlers.Encryption:59

Medium

Risk of AES misconfiguration for cipher `AES/CBC`

The application is using AES in mode `AES/CBC`, which is likely insecure or misconfigured. Using a misconfigured AES mode potentially allows an attacker to decrypt the encrypted data without a key. Possible misconfigurations could be a weak cipher mode or the usage of a predictable IV. To achieve confidentiality, the authenticated encryption mode `AES_256/GCM/NoPadding` is recommended.

Recommendations

Use `AES_256/GCM/NoPadding` as encryption mode for AES

External Links

MSTG-CRYPTO-3: Common Cryptographic Configuration Issues

OWASP recommendations regarding secure AES encryption modes

1 Finding

- com.htbridge.pivaa.handlers.Encryption:124

Low

The app logs information

Logs may give important information to an attacker, in particular if sensitive data is logged. Logging may also slow the app down

Recommendations

Remove all logging statements before releasing the app. Using tools, this can be done in an automated way.

Fix with ProGuard

The ProGuard configuration can be modified to remove logging by adding

```
-assumenosideeffects class ↩️
                  android.util.Log {
    public static int v(...);
    public static int i(...);
    public static int w(...);
    public static int d(...);
    public static int e(...);
}
```

More detailed information can be found in the ProGuard Community.

External Links

OWASP recommendations regarding using loggers

56 Findings

- 3 occurences in androidx.appcompat.view
- 1 occurences in androidx.tracing
- 1 occurences in com.google.android.material.bottomsheet
- 101 occurences in androidx.fragment.app
- 1 occurences in com.google.android.material.internal
- 2 occurences in androidx.appcompat.view.menu

- 7 occurences in androidx.core.text
- 2 occurences in com.google.android.material.snackbar
- 27 occurences in androidx.core.graphics.drawable
- 68 occurences in androidx.constraintlayout.motion.widget
- 27 occurences in com.htbridge.pivaa.handlers
- 2 occurences in androidx.appcompat.graphics.drawable
- 2 occurences in androidx.activity.result
- 2 occurences in com.google.android.material.tabs
- 25 occurences in androidx.appcompat.app
- 1 occurences in androidx.customview.widget
- 6 occurences in androidx.localbroadcastmanager.content
- 1 occurences in com.google.android.material.card
- 2 occurences in com.google.android.material.resources
- 11 occurences in androidx.core.widget
- 8 occurences in androidx.core.os
- 1 occurences in androidx.appcompat.content.res
- 7 occurences in androidx.core.content.res
- 12 occurences in androidx.constraintlayout.motion.utils
- 30 occurences in androidx.core.app
- 34 occurences in androidx.constraintlayout.widget
- 9 occurences in androidx.vectordrawable.graphics.drawable
- 9 occurences in androidx.core.util
- 32 occurences in androidx.core.view
- 3 occurences in androidx.print
- 2 occurences in com.google.android.material.transition
- 12 occurences in com.htbridge.pivaa
- 10 occurences in androidx.transition
- 1 occurences in androidx.core.view.accessibility
- 6 occurences in androidx.documentfile.provider
- 12 occurences in androidx.loader.app
- 3 occurences in com.google.android.material.button
- 39 occurences in androidx.recyclerview.widget
- 1 occurences in androidx.core.content
- 1 occurences in androidx.loader.content
- 1 occurences in com.google.android.material.shape
- 1 occurences in com.htbridge.pivaa.handlers.about
- 3 occurences in com.google.android.material.slider
- 60 occurences in androidx.appcompat.widget
- 1 occurences in com.google.android.material.textfield
- 9 occurences in com.htbridge.pivaa.handlers.database
- 1 occurences in androidx.legacy.content
- 2 occurences in com.google.android.material.transition.platform

- 1 occurences in com.google.android.material.badge
- 2 occurences in androidx.viewpager.widget
- 1 occurences in com.google.android.material.animation
- 10 occurences in androidx.core.graphics
- 13 occurences in com.google.android.material.chip
- 3 occurences in com.google.android.material.floatingactionbutton
- 4 occurences in androidx.coordinatorlayout.widget
- 1 occurences in com.google.android.material.ripple