

[Open in app](#)

Search



My fav 7 methods for Bypassing Android Root detection



Kishor balan · Following

6 min read · Oct 16, 2022

[Listen](#)[Share](#)[More](#)

Hola H3ck3rs,

You peeps already know, when we are into an android pentest/BB, the root detection protections are basically a creepy thing for all of us.

I am not going to explain a new methodology or any of my own discovery rather than listing out a few of the existing traditional and modern methods.

Prerequisites:

Familiar with Basic Android Pentesting and tools such as adb, frida, Objection, Magisk application, Decompiling/Recompiling APK, APK Signing and Dex to Jar Conversion.

What we are gonna cover today:

1. MagiskHide module in the Magisk application (= <v23.0)
2. Zygisk DenyList in the Magisk Application (>v23.0)
3. Tampering the Smali code

4. Objection's common method

5. Bit manual with Objection

6. Using Frida scripts

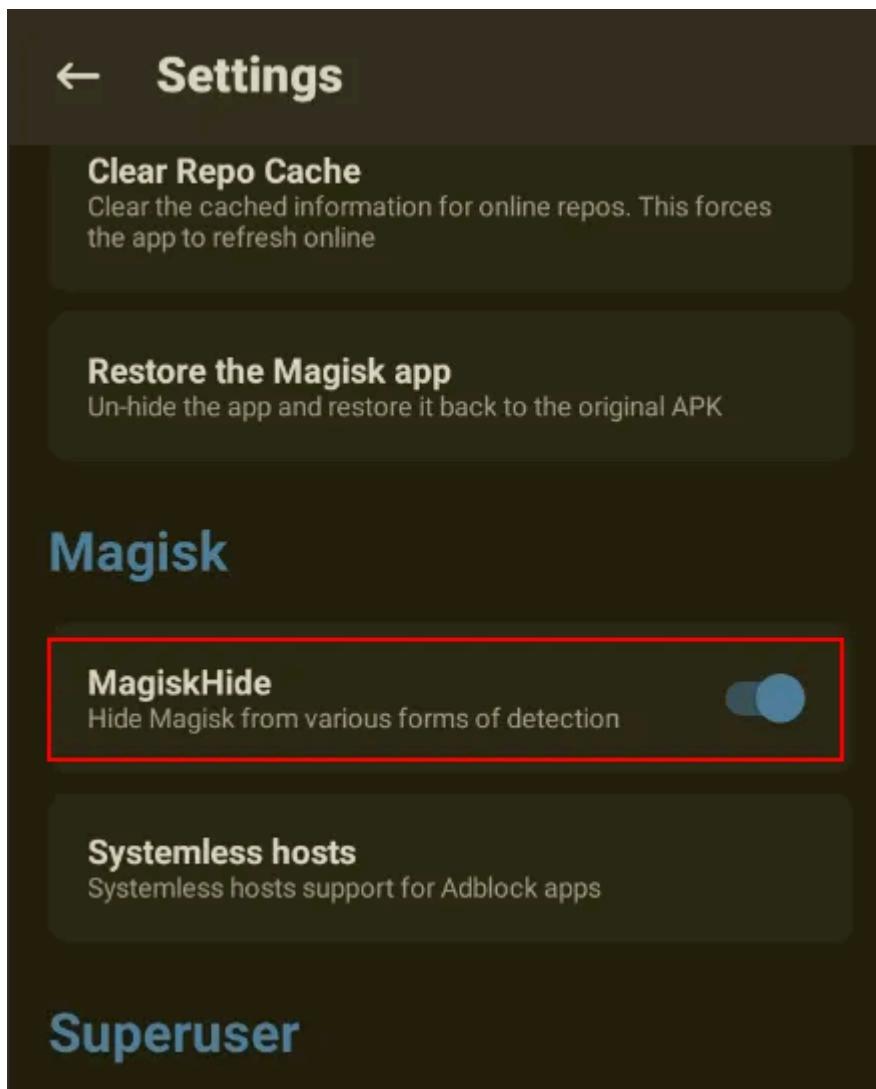
7. Medusa framework

!!! I haven't covered the basic stuff like setting up the frida server, decompiling/Re-compiling the android application, etc. Hope u guys are already familiar with them!!!

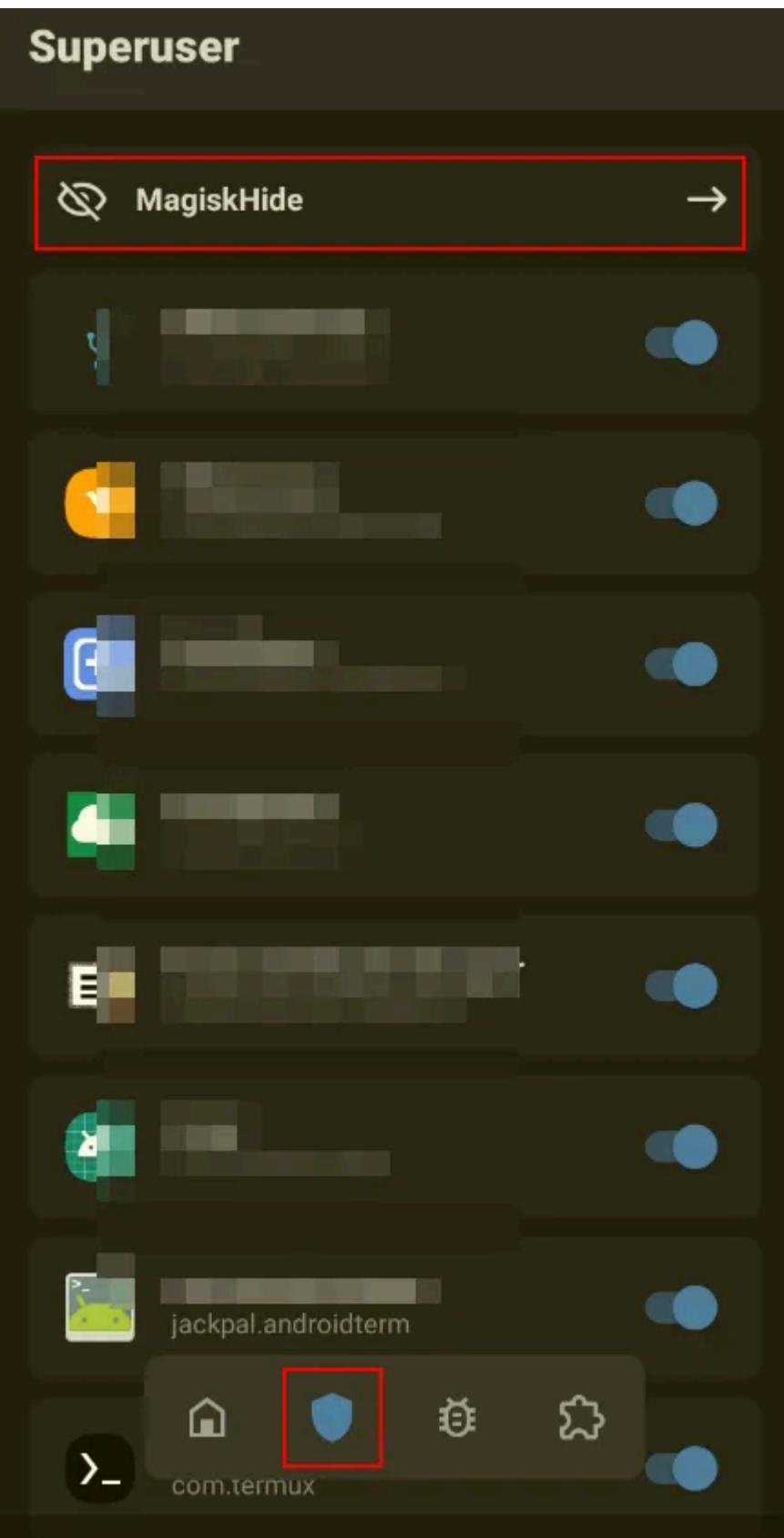
1) MagiskHide module in the Magisk application (v23.x<=)

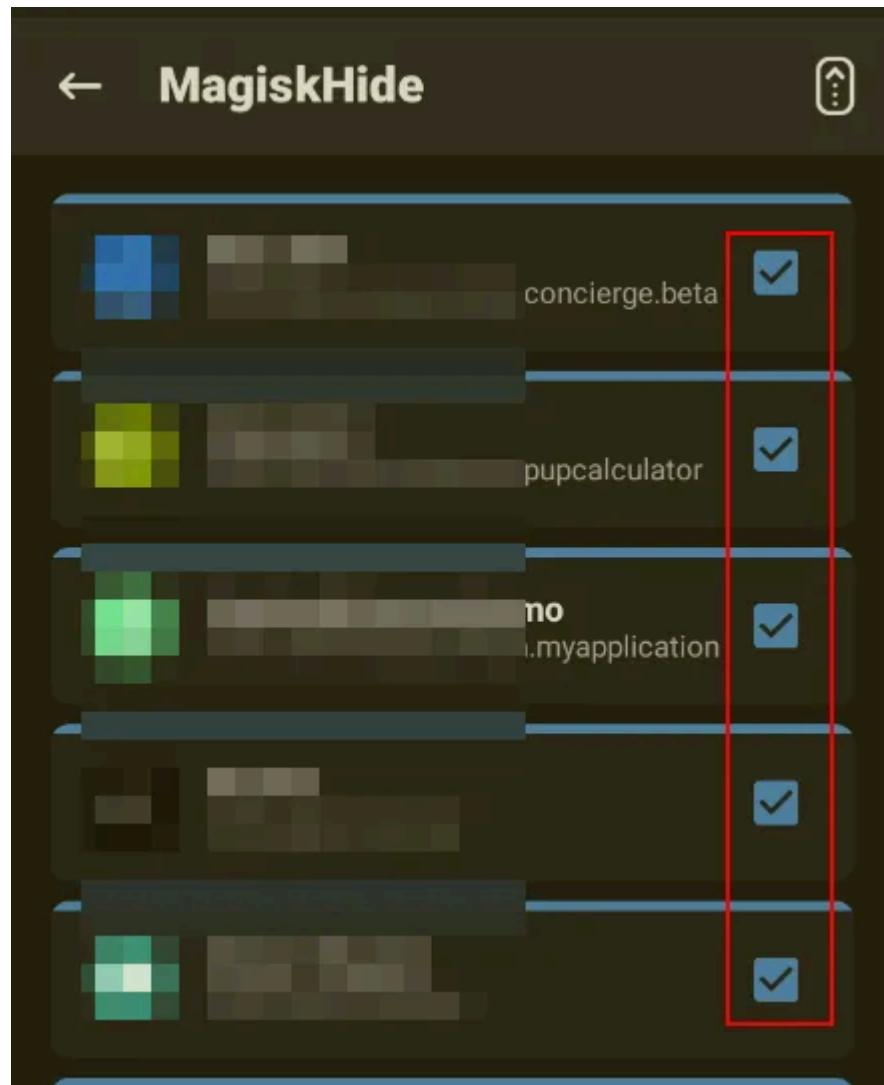
If your device is rooted with Magisk, then you can try the MagiskHide available in the Magisk Application. And **please note that this feature is only available up to the Magisk version 23.x**

Step:1 Enable the MagiskHide module from the app settings



Step 2: Choose the application in which we have to hide the root

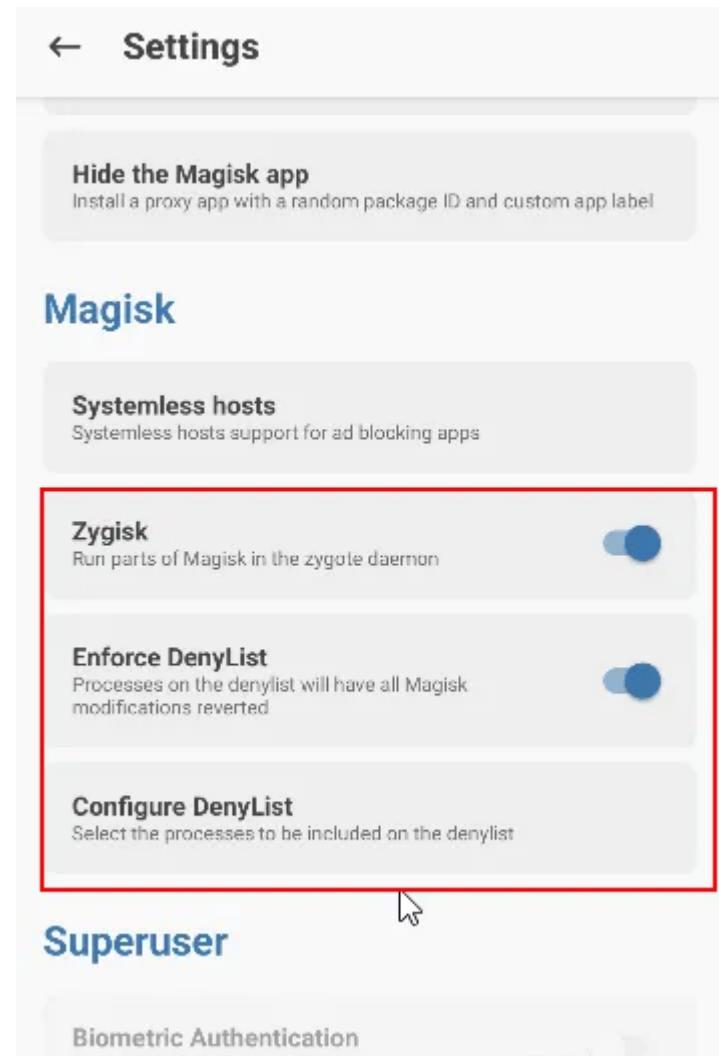




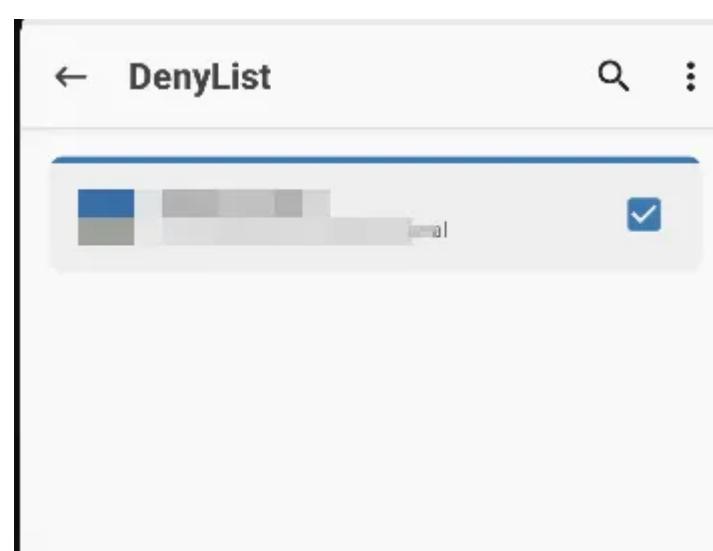
2. Zygisk Denylist (Magisk App > v23.x)

Magisk App > v23.0 provides an alternative feature -Zygisk Denylist instead of the MagiskHide. By enabling this feature, we can bypass the root detection of most applications.

Step 1: Enable Zygisk, DenyList from the app settings



Step 2: Choose the application that we have to hide the root detection

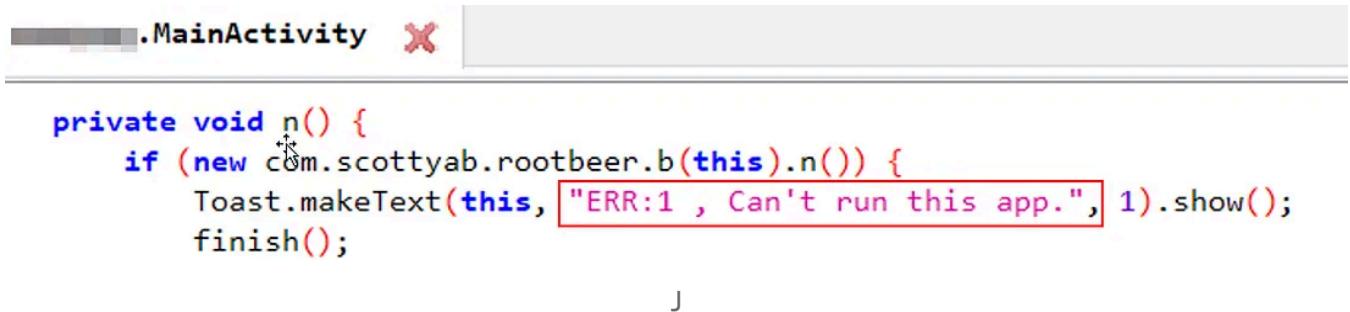


3. Tampering the Smali Code

Yes, sometimes we can do a spell with the application smali code which can be obtained after decompiling the application with Apktool.

Step 1: Decompile the apk file using JADX-GUI or any other alternative.

Step 2: Identify the code which is in charge of the root detection process:



```
private void n() {
    if (new com.scottyab.rootbeer.b(this).n()) {
        Toast.makeText(this, "ERR:1 , Can't run this app.", 1).show();
        finish();
    }
}
```

Step 3: In this case, the application is using the rootbeer library for the root detection. And we can see there is a “if” condition is the decision maker element that decides whether the application is rooted or a Non-rooted device.

Step 4: Now decompile the APK with Apktool and find smali code for the above “if” statement.

```
>MainActivity.smali (active) x

109     .locals 3
110
111     .line 1
112     new-instance v0, Lcom/scottyab/rootbeer/b;
113
114     invoke-direct {v0, p0}, Lcom/scottyab/rootbeer/b;
115
116     invoke-virtual {v0}, Lcom/scottyab/rootbeer/b; ->
117
118     move-result v0
119
120     const/4 v1, 0x1
121
122     if-eqz v0, :cond_0
123
124     const-string v0, "ERR:1 , Can't run this app."
125
126     .line 2
127     invoke-static {p0, v0, v1}, Landroid/widget/Toast;
128
129     move-result-object v0
130
131     invoke-virtual {v0}, Landroid/widget/Toast; ->show
132
133     .end method
```

Step 2: Modify the condition statement as illustrated in the below picture.

Modified_MainActivity.smali (other) X

```
.locals 3

.line 1
new-instance v0, Lcom/scottyab/rootbeer/b;

invoke-direct {v0, p0}, Lcom/scottyab/rootbeer/b;
invoke-virtual {v0}, Lcom/scottyab/rootbeer/b; ->
move-result v0

const/4 v1, 0x1

if-nez v0, :cond_0

const-string v0, "ERR:1 , Can't run this app."

.line 2
invoke-static {p0, v0, v1}, Landroid/widget/Toas
move-result-object v0
```

Step 3: Now save the file and re-build the application, don't forget to sign the apk file.

```

~ [REDACTED] > apktool b [REDACTED] -o test.apk
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
I: Using Apktool 2.4.1
I: Checking whether sources has changed...
I: Smaling smali folder into classes.dex...
I: Checking whether sources has changed...
I: Smaling smali_classes3 folder into classes3.dex...
I: Checking whether sources has changed...
I: Smaling smali_classes2 folder into classes2.dex...
I: Checking whether resources has changed...
I: Copying raw resources...
I: Copying libs... (/lib)
I: Copying libs... (/kotlin)
I: Copying libs... (/META-INF/services)
I: Building apk file...
I: Copying unknown files/dir...
I: Built apk...

~, [REDACTED] > d2j-apk-sign test.apk -o test_signed.apk
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
sign test.apk -> test_signed.apk

```

Re-packing the application

This will change the application logic and bypass the root detection.

4. Using Objection's Common method

I hope u guys are familiar with the objection tool, if not, pull it out using pip

pip3 install objection

Step 1: start the frida server on the android device/ Virtual device.

```

generic_x86_arm:/ $ su
generic_x86_arm:/ # cd /data/local/tmp
generic_x86_arm:/data/local/tmp # ./frida

```

Step 2: Now, launch the application with the following Objection command.

objection -g com.test.app explore

```
C:\Users\kishor>objection -g com.scottyab.rootbeer.sample explore
Using USB device `SM M105F`
Agent injected and responds ok!

[. . . . .]
|____|(object)inject(ion) v1.11.0

Runtime Mobile Exploration
by: @leonjza from @sensepost

[tab] for command suggestions
com.scottyab.rootbeer.sample on (samsung: 9) [usb] #
```

Step 3: now the application will be launched on your device, then execute the following objection command.

```
[tab] for command suggestions
com.scottyab.rootbeer.sample on (samsung: 9) [usb] # android root disable
(agent) Registering job 442663. Type: root-detection-disable
com.scottyab.rootbeer.sample on (samsung: 9) [usb] # (agent) [442663] File existence check for /
(agent) [442663] File existence check for /data/local/su detected, marking as false.
(agent) [442663] File existence check for /data/local/bin/su detected, marking as false.
(agent) [442663] File existence check for /data/local/xbin/su detected, marking as false.
(agent) [442663] File existence check for /sbin/su detected, marking as false.
(agent) [442663] File existence check for /system/bin/su detected, marking as false.
(agent) [442663] File existence check for /system/bin/failsafe/su detected, marking as false.
(agent) [442663] File existence check for /system/sd/xbin/su detected, marking as false.
(agent) [442663] File existence check for /system/xbin/su detected, marking as false.
```

This module will modify the values of the class methods of the root detection library in order to bypass the root detection

Step 4: If the root detection prompt is still there on the application, just press the back button and exit from the application (don't kill the app from background) and open it again.

The root detection will disappear :)

5. Bit Manual with Objection tool

This method is a kinda tricky.

Following are my steps.

Step 1: First convert the apk file into class files using dex2jar

Step 2: Analyse the class files and identify which library is being used for the root detection

The screenshot shows the JD-GUI interface. On the left, the package structure is displayed with a red box highlighting the `de.cyberkatze.iroot` package. Inside this package, the `IRoot.class` file is selected. On the right, the decompiled Java code for `IRoot` is shown:

```

public class IRoot extends CordovaPlugin {
    private boolean checkBuildTags() {
        String str = Build.TAGS;
        return (str != null && str.contains("test-keys"));
    }

    private boolean checkFilePath() {
        for (int i = 0; i < 8; i++) {
            (new String[8])[0] = "/sbin/su";
            (new String[8])[1] = "/system/bin/su";
            (new String[8])[2] = "/system/xbin/su";
            (new String[8])[3] = "/data/local/xbin/su";
            (new String[8])[4] = "/data/local/bin/su";
            (new String[8])[5] = "/system/sd/xbin/su";
            (new String[8])[6] = "/system/bin/failsafe/su";
            (new String[8])[7] = "/data/local/su";
            if ((new File((new String[8])[i])).exists())
                return true;
        }
        return false;
    }

    private boolean checkSuperUserApk() {
        return (new File("/system/app/Superuser.apk")).exists();
    }

    private boolean isDeviceRooted() {
        return (checkBuildTags() || checkSuperUserApk() || checkFilePath());
    }
}

```

Step 3: Now connect the app with objection (objection -g pkg_name/processID explore)

Step 4: Execute the following command

android hooking list class_methods <root detection class>

```

Found 1 classes
a [REDACTED] on (samsung: 9) [usb] # android hooking list class_methods de.cyberkatze.iroot.IRoot
private boolean de.cyberkatze.iroot.IRoot.checkBuildTags()
private boolean de.cyberkatze.iroot.IRoot.checkFilePath()
private boolean de.cyberkatze.iroot.IRoot.checkSuperUserApk()
private boolean de.cyberkatze.iroot.IRoot.isDeviceRooted()
public boolean de.cyberkatze.iroot.IRoot.execute(java.lang.String,org.json.JSONArray,org.apache.cordova.CallbackContext)

Found 5 method(s)
a [REDACTED] on (samsung: 9) [usb] #

```

Step 5: Now we can identify that the following boolean method `isDeviceRooted()` (may differ in other libraries) is in charge of root detection.

```
de.cyberkatze.iroot.IRoot.checkBuildTags()
private boolean de.cyberkatze.iroot.IRoot.checkFilePath()
private boolean de.cyberkatze.iroot.IRoot.checkSuperUserApk()
private boolean de.cyberkatze.iroot.IRoot.isDeviceRooted()
public boolean de.cyberkatze.iroot.IRoot.execute(java.lang.String, or

Found 5 method(s)
```

Step 5: We can change its return value `true` to `false` with the following command:

android hooking set return_value <root_detect_class.method> false

```
Found 5 method(s)
a [REDACTED] on (samsung: 9) [usb] # android hooking set return_value de.cyberkatze.iroot.IRoot.isDeviceRooted false
(agent) Attempting to modify return value for class de.cyberkatze.iroot.IRoot and method isDeviceRooted.
(agent) Hooking de.cyberkatze.iroot.IRoot.isDeviceRooted()
(agent) Registering job 133872. Type: set-return for: de.cyberkatze.iroot.IRoot.isDeviceRooted
      on (samsung: 9) [usb] # (agent) [133872] Return value was not false, setting to false.
(agent) [133872] Return value was not false, setting to false.
```

Step 6: Thats it, here we have changed the boolean value returned by the root detection class method, and this will change the application logic for root detection process and bypass the root detection.

6. Using Frida scripts

Here comes the most and widely used root detection bypass method.

Step 1: Make sure the frida server is running on your device and the USB cable is properly connected (if it is a physical device)

Step 2: Also make sure that your PC and the device have established a proper frida connection.

Step 3: You can verify the frida connection by using the command, shown in the below image.

```
D:\rootAVD>frida-ls-devices
Id          Type      Name
-----
local       local     Local System
emulator-5554  usb      Android Emulator 5554
socket      remote    Local Socket
```

Step 4: Now we have to find a good and popular frida scripts from the Codeshare or any other git repositories

My fav one is:

- i) <https://codeshare.frida.re/@dzoncerzy/fridantiroot/>

Step 5: Now connect your application with frida and load the frida script with the command, shown in below image.

```
D:\rootAVD>frida -U --codeshare dzoncerzy/fridantiroot -f com.████████ --no-pause
Frida 15.1.13 - 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/
Spawned `com.████████` Resuming main thread!
[SM M105F:::com.████████]-> message: {'type': 'send', 'payload': 'Loaded 12115 classes!'} data: None
message: {'type': 'send', 'payload': 'loaded: -1'} data: None
message: {'type': 'send', 'payload': 'ProcessManager hook not loaded'} data: None
message: {'type': 'send', 'payload': 'Bypass return value for binary: Superuser.apk'} data: None
message: {'type': 'send', 'payload': 'Bypass return value for binary: su'} data: None
message: {'type': 'send', 'payload': 'Bypass return value for binary: su'} data: None
message: {'type': 'send', 'payload': 'Bypass return value for binary: su'} data: None
message: {'type': 'send', 'payload': 'Bypass return value for binary: su'} data: None
message: {'type': 'send', 'payload': 'Bypass return value for binary: su'} data: None
message: {'type': 'send', 'payload': 'Bypass return value for binary: su'} data: None
message: {'type': 'send', 'payload': 'Bypass return value for binary: su'} data: None
message: {'type': 'send', 'payload': 'Bypass return value for binary: su'} data: None
```

7. Medusa Framework

Similar to Objection tool, MEDUSA is also a flexible framework that automates the procedures and methods used in the dynamic analysis of Android applications.

Git repo:

[Ch0pin/medusa](https://github.com/Ch0pin/medusa): Binary instrumentation framework based on Frida (github.com).

Step 1: Launch medusa

```
[root@kali:~/medusa]# python3 medusa.py
[i] Loading modules...
[i] Done....
Welcome to:

A large, pixelated green logo for "MEDUSA" is displayed. The letters are formed by a grid of green squares, with some squares being solid green and others having a diagonal dot pattern. The logo is centered on the screen below the command-line interface.

Type help for options

Available devices:

0) Device(id="local", name="Local System", type='local')
1) Device(id="socket", name="Local Socket", type='remote')
2) Device(id="5200dcd78dfa685", name="SM M105F", type='usb')

Enter the index of the device to use: 2

Device properties:

[ro.product.manufacturer]: [samsung]
[ro.product.name]: [m10ltedd]
```

Step 2: We have to use the `helpers/anti_debug` module for bypassing the root detection

Step 3: Follow the steps illustrated in the following screenshots.

```
medusa> search anti_debug
helpers/anti_debug
medusa> use helpers/anti_debug

Current Mods:
0) helpers/anti_debug

medusa> compile

Script is compiled

medusa> run -f ae.[REDACTED].jar
Spawned package : [REDACTED] on pid 26917
[in-session] [c:clear | e:exit | r:reload |
| rs:reset scratchpad | i:info | t:trace | ?:help |:
-----Application Info-----
- Application Name: android.app.Application
- Files Directory: /data/user/0/[REDACTED]/r/files
- Cache Directory: /data/user/0/[REDACTED]/r/cache
- External Cache Directory: /storage/emulated/0/Android/data/[REDACTED]/r/cache
- Code Cache Directory: /data/user/0/[REDACTED]/r/code_cache
- Obb Directory: /storage/emulated/0/Android/obb/[REDACTED]
- Package Code Path: /data/app/[REDACTED]-yLTMCanVbK7g-bdT-YkiAw==/base.apk
```

-----EOF Application Info-----

-----LOADING ANTI ROOT DETECTION SCRIPT-----
Loaded 11616 classes!
loaded: -1
ProcessManager hook not loaded
[!] overwriting is debugger connected
Bypass test-keys check
Bypass return value for binary: Superuser.apk
Bypass return value for binary: su
Bypass test-keys check
Bypass return value for binary: Superuser.apk
Bypass return value for binary: su
Bypass return value for binary: su

Bypassing root detection during the run time

That's it peeps, I'm sure there is a 90% possibility that any of these methods will help you to get rid of the root detection problem.

See you on next write up

References:

[Ch0pin/medusa: Binary instrumentation framework based on FRIDA \(github.com\)](#)

[topjohnwu/Magisk: The Magic Mask for Android \(github.com\)](#)

[frida/frida: Clone this repo to build Frida \(github.com\)](#)

Cybersecurity

Android

Infosec

Bug Bounty

Pentesting



Following



Written by Kishor balan

418 Followers

Security Analyst || eWPTXv2 || eJPT

More from Kishor balan

iPhone 7 32GB Black PC Charging(2W) 4%

iOS Version	13.3.1 (17D50)
Jailbroken	Yes Install AFC2
Activated	Yes
Product Type	iPhone9,1 (A1779)
Sales Model	MNCE2 J/A
IMEI	[REDACTED]
Serial No.	[REDACTED] G7X
ECID	[REDACTED] 026
Verify UDID	Yes
UDID	[REDACTED] B11B
View Verification Report	
View iDevice Details	

Hard Disk Capacity 11.65 GB / 29.79 GB

System Apps Photos Media UDisk Used Free

Reboot Turn Off Refresh

Kishor balan

Start your first iOS Application Pentest with me.. (Part- 1)

Hola Heckers,

6 min read · Jan 14, 2023

240 3



```
(iPhone: 16.4.1) [usb] # env
-----
r/containers/Bundle/Application/73335472-A448-4367-B0E5-1607F438F81D/DamnVulnerableIOSApp.app
/Containers/Data/Application/8722EAC6-06EC-4665-A854-24DBC5D1FD6/Library/Caches
/Containers/Data/Application/8722EAC6-06EC-4665-A854-24DBC5D1FD6/Documents
/Containers/Data/Application/8722EAC6-06EC-4665-A854-24DBC5D1FD6/Library
(iPhone: 16.4.1) [usb] # cd /var/mobile/Containers/Data/Application/8722EAC6-06EC-4665-A854-24DBC5D1FD6/Library/Caches
(iPhone: 16.4.1) [usb] # cd com.highaltitudehacks.dvia
(iPhone: 16.4.1) [usb] # ls
action      Read  Write  Owner       Group      Size   Creation
-----  -----
ilFirstUserAuthentication  True   True   mobile (501)  mobile (501)  128.0 B  2023-08-13 13:54:59
ilFirstUserAuthentication  True   True   mobile (501)  mobile (501)  64.0 B   2023-08-13 13:54:59
ilFirstUserAuthentication  True   True   mobile (501)  mobile (501)  48.0 KiB 2023-08-13 12:42:57
ilFirstUserAuthentication  True   True   mobile (501)  mobile (501)  0.0 B   2023-08-13 12:42:57
ilFirstUserAuthentication  True   True   mobile (501)  mobile (501)  32.0 KiB 2023-08-13 12:42:57
```

Kishor balan

iOS Pentesting Series Part 2- Into The Battlefield..

Hola Peeps,

6 min read · Aug 14, 2023

 44 2

...

```
on="1.0" encoding="utf-8"?>
<security-config>
<pin-config>
  main includeSubdomains="true">example.com</domain>
  <pin expiration="2018-01-01">
    <pin digest="SHA-256">7HIpactkIAq2Y49orF00QKurWxmmSFZhBCoQYcRhJ3
    <!-- backup pin -->
    <pin digest="SHA-256">fwza0LRMXouZHRC8Ei+4PyuldPDcf3UKg0/04cDM1c
  </pin>
</pin-set>
<pin-config>
<security-config>
```

 Kishor balan

It's all about Bypassing Android SSL Pinning and Intercepting Proxy Unaware applications.

Hola H3ckers,

6 min read · Nov 27, 2022

 243 2

...

```
[root] ~ % com.hightaltitudehacks.dvia on (iPhone: 16.4.1) [usb] # ios hooking generate simple JailbreakDetectionVC
var target = ObjC.classes.JailbreakDetectionVC;

Interceptor.attach(target['- isJailbroken'].implementation, {
  onEnter: function (args) {
    console.log('Entering - isJailbroken!');
  },
  onLeave: function (retval) {
    console.log('Leaving - isJailbroken');
  },
});

Interceptor.attach(target['- readArticleTapped:'].implementation, {
  onEnter: function (args) {
    console.log('Entering - readArticleTapped:');
  },
  onLeave: function (retval) {
    console.log('Leaving - readArticleTapped:');
  },
});

Interceptor.attach(target['- jailbreakTest1Tapped:'].implementation, {
```

 Kishor balan

iOS Pentesting Series Part 3- The Ceasefire

Hola mates,

7 min read · Aug 19, 2023

 22 



...

See all from Kishor balan

Recommended from Medium

 Aimar Sechan Adhitya

Bypassing Root Detection the Universal Way

Getting tired of reverse engineering obfuscated codes to bypass root detection? Time to learn the better way!

4 min read · Jan 12, 2024

 41  1

...

 Pawan Jaiswal

Android Penetration Testing with Frida: A Comprehensive Guide with Examples

As mobile devices become increasingly integral to our daily lives, securing Android applications against potential vulnerabilities is of...

4 min read · Jan 27, 2024



19



...

Lists



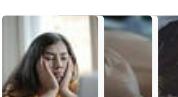
Tech & Tools

16 stories · 212 saves



Medium's Huge List of Publications Accepting Submissions

285 stories · 2528 saves



Staff Picks

630 stories · 922 saves



Natural Language Processing

1407 stories · 905 saves

Android



Sandeep Vishwakarma in InfoSec Write-ups

A step-by-step Android penetration testing guide for beginners

Greetings fellow hackers, my name is Sandy, Security Analyst and Bug bounty hunter.

18 min read · Nov 3, 2023

678

4



...



Sahil Choudhary

SSL Pinning Bypass on Flutter-Based Android Apps

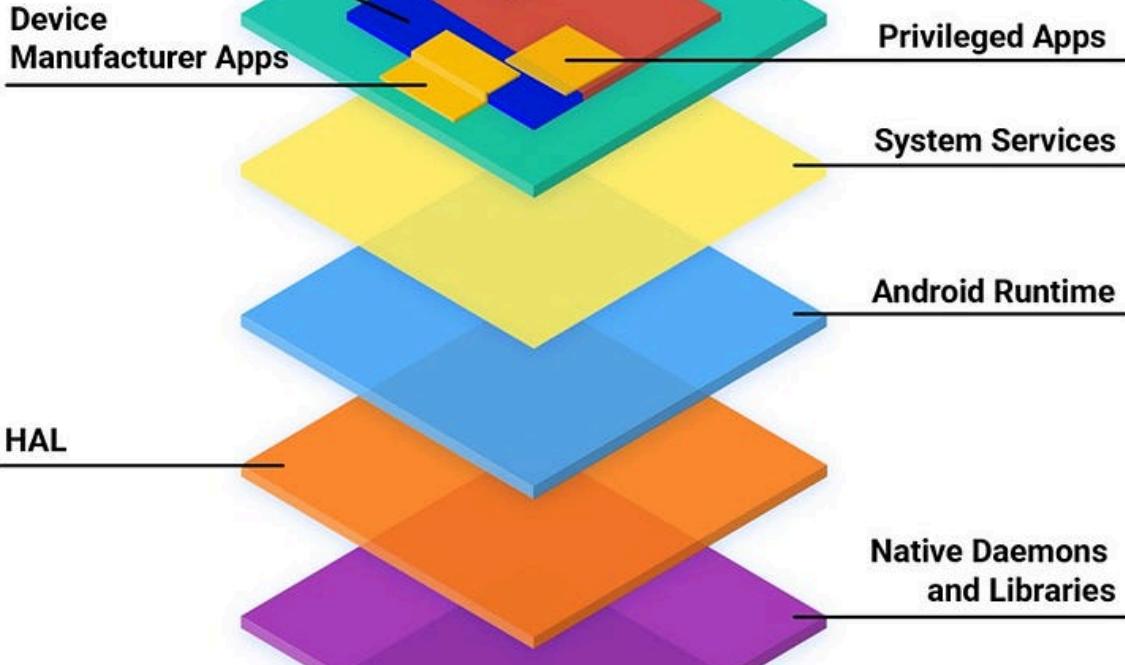
In the ever-evolving landscape of mobile application security, SSL pinning has become a common defense mechanism to protect sensitive data...

2 min read · Jan 3, 2024

13



...



 Rudrani Maity

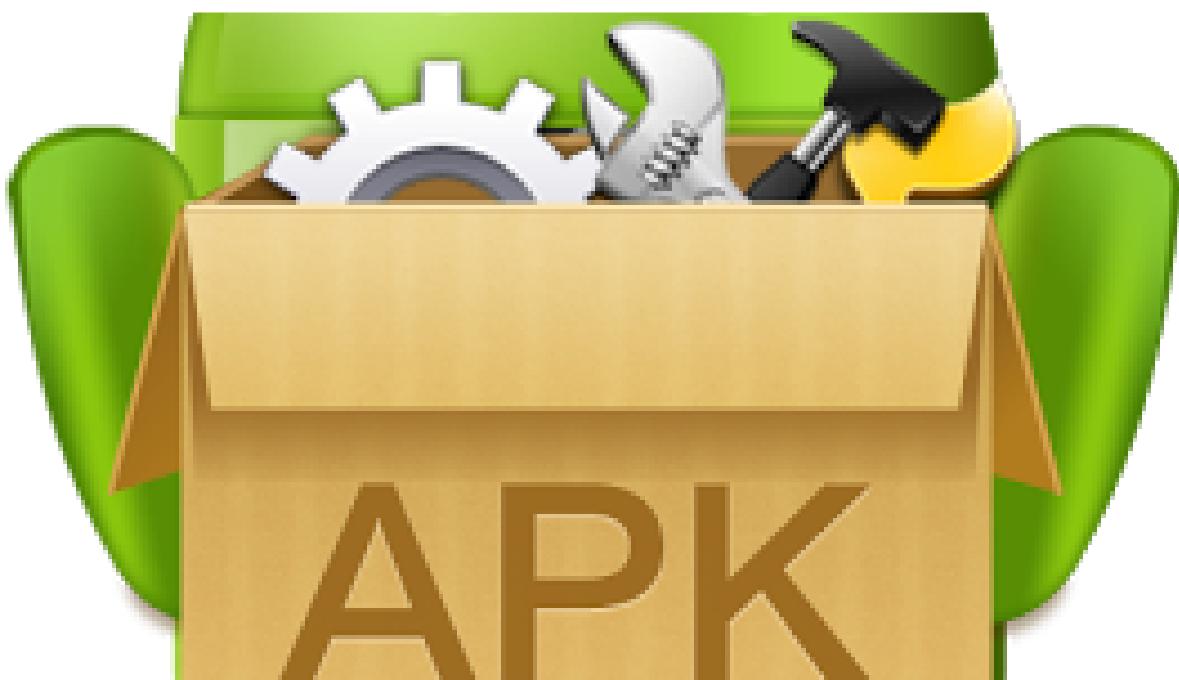
Network Traffic Analysis of Android apps using MITMproxy and Frida

Abstract: With the overwhelmingly increasing number of applications in android marketplace, i.e., google play store; over time with the...

15 min read · Nov 22, 2023



...



 Hacker's Dump

Diving Deep: A Comprehensive Guide to Android Penetration Testing—Part 3

Cracking the Code: A Beginner's Guide to Decoding Android Apps

8 min read · Nov 19, 2023



...

See more recommendations