Bancontact Payconiq Company

SEPA Rulebooks
Scheme Manuals
Remote Domain
46D0 – Schedules 1, 2, and 3 – News 64
Mobile App Security Guidelines

Android, iOS
Newsletter 64

Bancontact
Payconiq
Company

Copyright

Authors

This monthly newsletter is written by NVISO Labs, experts in mobile security, on behalf of Bancontact Company sa/nv.

About NVISO

NVISO is a consultancy firm exclusively focusing on IT security. NVISO has a very clear sector focus with several references in the financial and governmental sectors. The Research and Development department of NVISO is NVISO Labs, whose goals are to allow our people to increase their skills and knowledge, to come up with innovative service offerings, to contribute to the security community, and to give valuable insights to our clients.
The fundamental values of NVISO are client satisfaction, focus, entrepreneurship, innovation, and ability to adapt. Our mission is to be an innovative and respected partner for our clients. For more information, we are happy to refer you to our website: https://www.nviso.be.
If you want to stay up to date with our latest research and other activities of NVISO Labs, we refer you to our blog: https://blog.nviso.be

Table of Contents

1 Summary of security impacts

January was quite a month for mobile security. Starting with Android and the Google Play Store.

Two malicious apps on the Google Play Store have been found that are using the Anubis Banking Malware and a clever way to evade detection.

Spyware has also been found on several apps in the store stealing clipboard items, call logs and even SMS conversations.

Some beauty camera applications have also been reported to send users to pornographic websites and stealing their pictures.

Finally concluding the Android section of this month's newsletter, researches have illustrated that out of 100 tested phones using facial recognition, 42 of them could be beaten by simply using a printed photograph.

In the camp of Apple this month, a bug in facetime was discovered, allowing users to spy on other users even before they answer the call.

The iPhones of activists, diplomats and rival foreign leaders have been hacked by a team of former U.S. government intelligence operatives working for the United Arab Emirates with the help of a sophisticated spying tool called Karma.

Apple has also released its first patch of 2019 fixing a decent number of CVE's.

In the case study this month, we are pushing our reverse-engineering skills to the limit as we crack OWASP's Level 3 CrackMe. This CrackMe contains multiple anti-reversing and anti-instrumentation controls that should make it difficult to figure out the correct flag. We have already looked at Level 1 (Newsletter 53) and Level 2 (Newsletter 61) in previous newsletters.

2 Malware and vulnerability details

2.1 iOS

FaceTime bug [1.N64.1]

Overall risk	Medium	**Impact**	Medium	**Likelihood**	Low

Summary

A flaw in FaceTime allowed an attacker to place a call to someone and listen to their microphone even if they did not pick up. It also allowed an attacker to capture video from the device's camera and was not just affecting iPhone and iPad but also Macs.

Details

Figure 1 - FaceTime Group Chat (source)

The bug relied entirely on a feature of iOS 12.1 called Group FaceTime, older versions of iOS or macOS were not affected by this bug.

The exploit followed the following chain of events:

1. The attacker places a call to the victim
2. While the call was still ringing, the attacker would need to bring up the "Add Person" screen
3. The attacker adds himself to the call a second time; doing this invokes the Group FaceTime, and caused the microphone of the victim to activate, even if the victim did not pick up.

To capture video of the target, there were two known techniques involved:

- One technique was to hope that the recipient pressed the power button on the phone to "decline" the call, in which case the camera would turn on.
- The other technique was to join the call from another device, which would also turn on the victim's camera.

Judging by the nature of this exploit, we can assume that FaceTime is already sending data when the FaceTime request is received. This is most likely to make FaceTime have a fast response time, because all the buffers are filled with audio and video. But by implementing this system, there was a disregard of potential security consequences.

The flaw essentially boils down to the fact that there was a logical error in the system which tricks FaceTime into thinking that the victim and the attacker are the same person (since the attacker joins the conversation twice)

At most however, the attacker would get about a minute of spying, since facetime does not ring forever. The attack was also not stealthy, since it required the attacker to call the victim, thus announcing himself to the victim.

The bug has been disclosed publicly on twitter by the user @BmManski, who is not a cybersecurity specialist, but a regular user, which makes this extra painful for Apple since the issue has not been disclosed through the dedicated channels.

Mitigation

Apple has temporarily disabled Group facetime functionalities to make sure the exploit cannot be abused. A statement has been released that Apple aims to fix the issue in iOS patch 12.1.4

Source:

https://techcrunch.com/2019/02/07/update-to-ios-12-1-4-to-re-enable-group-facetime/

https://blog.malwarebytes.com/101/2019/01/apples-facetime-privacy-bug-allowed-possible-spying/

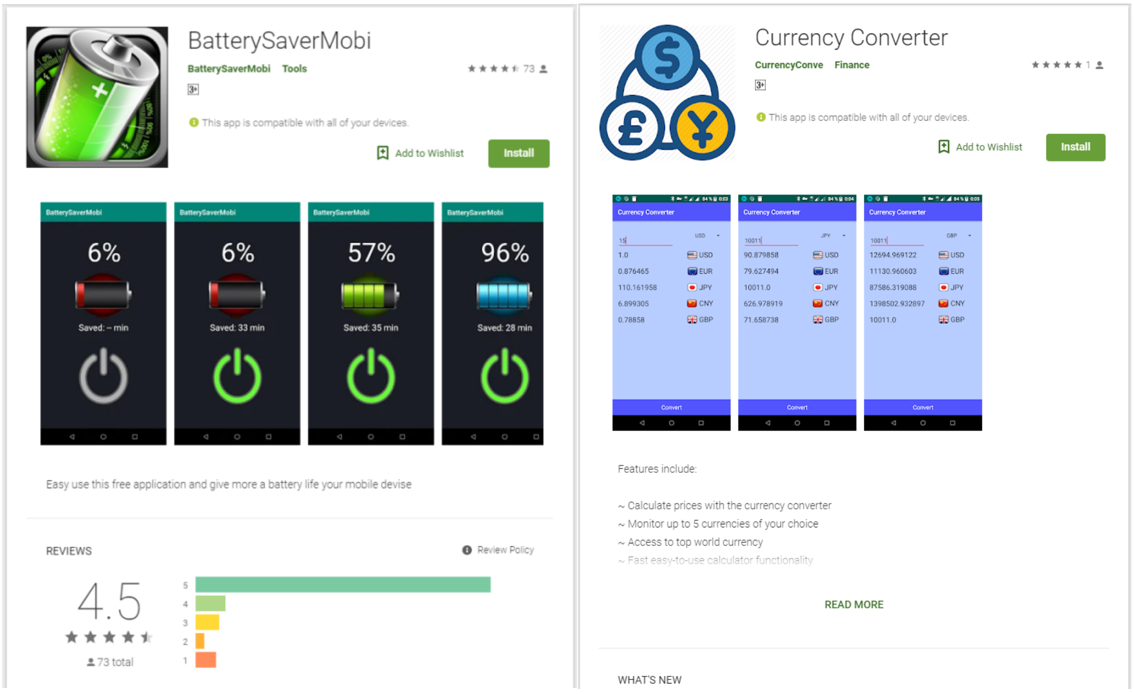
2.2 Android

ANDROIDOS_ANUBISDROPPER [2.N64.1]

Overall risk Low **Impact** High **Likelihood** Low

Summary

Two malicious applications have been found in the Google Play Store recently, called "Currency Converter" and "BatterySaveMobi". The two applications are disguised as useful applications but are carriers of the "Anubis" payload, a well-known banking malware.



Easy use this free application and give more a battery life your mobile devise

Features include:

~ Calculate prices with the currency converter
~ Monitor up to 5 currencies of your choice
~ Access to top world currency
~ Fast easy-to-use calculator functionality

Figure 2 - The Malicious apps (Source)

The battery app had more than 5000 downloads before it was taken down and the currency converter was downloaded for about 500 times.

Details

Anubis is a very disrupting banking malware which does not rely on a fake overlay like the traditional overlays but has a built in keylogger to capture keystrokes. Anubis could also be used as a ransomware. It uses the aserogeege.space domain as command and control server, but besides aserogeege, Anubis also uses eighteen other domains.

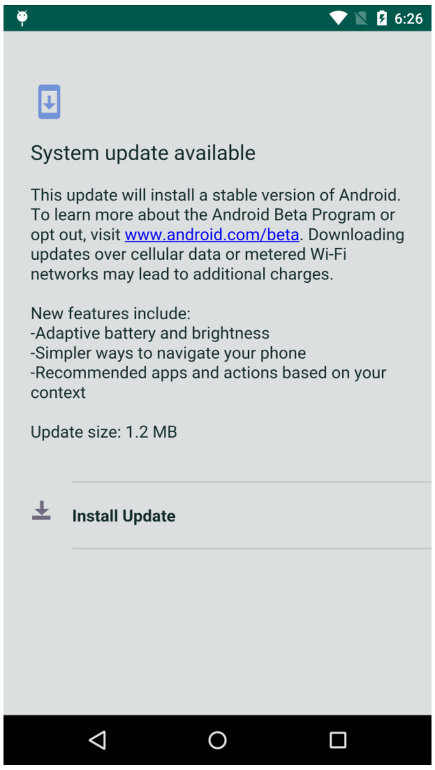
The domains that are being used have switched IP addresses six times since October 2018 (which was the last Anubis campaign). This shows us just how active this malware is.

The apps use a clever way to try and evade detection. As a user moves, their device generates some amount of motion data. The developer of these malicious apps used this knowledge to try and fool emulator sandbox environments, commonly used to detect malware, since these typically don't generate such motion data.

```
public void onSensorChanged(SensorEvent arg10) {
    this.k.registerListener(((SensorEventListener)this), this.l, 3);
    Sensor v0 = arg10.sensor;
    this.k.registerListener(((SensorEventListener)this), v0, 3);
    if(v0.getType() == 1) {
        float[] v10 = arg10.values;
        float v0_1 = v10[0];
        float v1 = v10[1];
        float v10_1 = v10[2];
        long v2 = System.currentTimeMillis();
        if(v2 - this.m > 100) {
            long v4 = v2 - this.m;
            this.m = v2;
            if(Math.abs(v0_1 + v1 + v10_1 - this.n - this.o - this.p) / (((float)v4)) * 10000f > 600f) {
                this.a();  // save step
            }

            this.n = v0_1;
            this.o = v1;
            this.p = v10_1;
        }
    }
}
```

Figure 3 - The motion tracking algorithm (Source)

If the apps don't detect any motion data, the malicious code will simply not run, thus evading detection. If the apps detect motion data, they assume that they're running on a genuine device, and their malicious code will run. When running, the apps will try to trick users into downloading the payload APK with a fake system update.


6:26

System update available

This update will install a stable version of Android. To learn more about the Android Beta Program or opt out, visit www.android.com/beta. Downloading updates over cellular data or metered Wi-Fi networks may lead to additional charges.

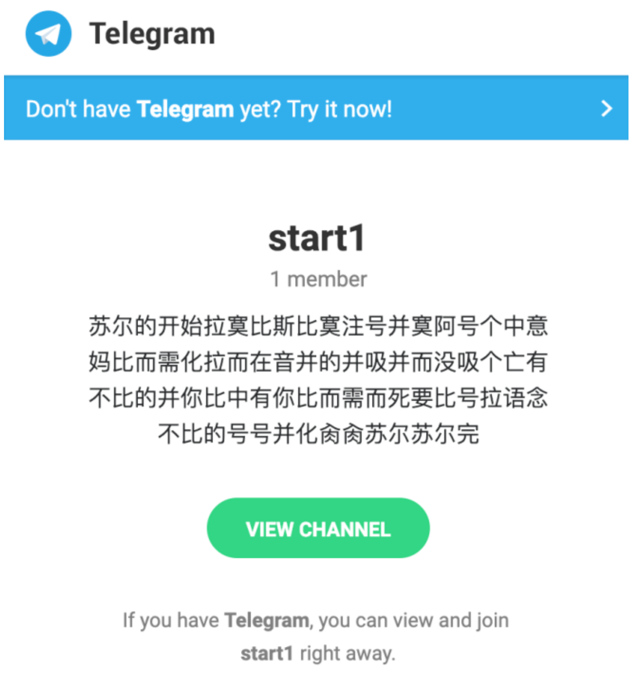
New features include:
-Adaptive battery and brightness
-Simpler ways to navigate your phone
-Recommended apps and actions based on your context

Update size: 1.2 MB

Install Update

Figure 4 - The fake system update (Source)

The developers of these applications attempt to hide the command and control server by encoding it in Telegram and Twitter webpage requests. By parsing the HTML content of the response, the device gets registered on the server. The server will then react to the application with an APK command which will drop the Anubis payload in the background.



start1

1 member

苏尔的开始拉寞比斯比寞注号并寞阿号个中意
妈比而需化拉而在音并的并吸并而没吸个亡有
不比的并你比中有你比而需而死要比号拉语念
不比的号号并化甸甸苏尔苏尔完

If you have **Telegram**, you can view and join **start1** right away.

Figure 5 - The encoded server URL (Source)

The latest data from TrendMicro shows that Anubis has been distributed to ninety-three countries and targets over three hundred financial apps to farm account details. Anubis can use the victims contact list to send spam messages and trick others to install the application, thus creating the means to spread itself.

Mitigation

The infected applications have been removed from the Google Play Store.

In order to prevent Anubis from stealing credentials, we advise developers to take biometric security into account such as fingerprint access. Additionally, adding strong device binding can help prevent easy account takeovers.

End-users should be cautious of any application that asks for banking credentials and be sure that they are legitimately linked to their bank.

More information can be found on the TrendMicro blog:

https://blog.trendmicro.com/trendlabs-security-intelligence/google-play-apps-drop-anubis-banking-malware-use-motion-based-evasion-tactics/

3 Case study – Attacking OWASP's CrackMe Level 3

In order to provide training material for Mobile Security consultants, OWASP has created a series of CrackMe's with increasing difficulty. The CrackMe's are applications that ask the user to enter a password and it will tell the user if that password is correct or not. The goal is thus to reverse engineer the application and figure out the correct password.

3.1 Introduction

This month, we are looking at the most difficult OWASP CrackMe: Level 3. This CrackMe contains multiple anti-reversing and anti-instrumentation checks that should make it difficult to figure out the correct flag. We have already looked at Level 1 (Newsletter 53) and Level 2 (Newsletter 61) and while those CrackMe's did give us a few hurdles to get around, they weren't too difficult yet. However, with the same toolset and a little bit more time, we will be able to extract the correct password from the Level 3 app.

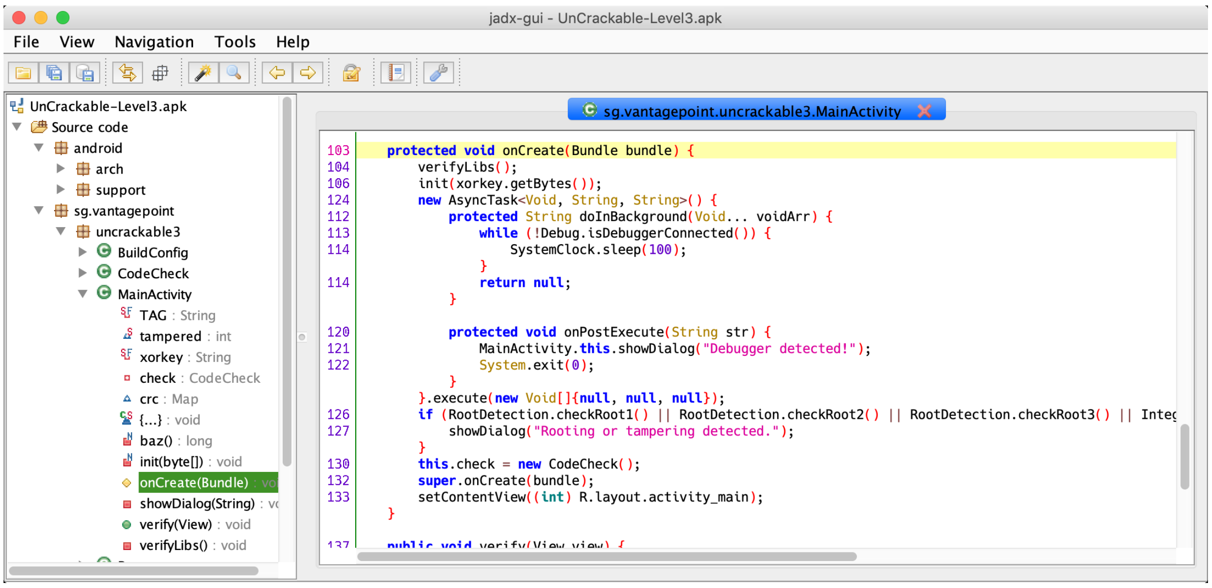
As always, our trusted setup is the following:

- Nexus 5 with LineageOS
- Magisk as the rooting method
- Frida installed through MagiskFridaServer

The CrackMe itself can be found on the OWASP MSTG GitHub and can be installed on the device through *adb install*.

3.2 Analyzing the application

Opening the APK in JadX-GUI gives us a quick overview of the application. We can see that there is not a lot of Java code, and that the application itself is not obfuscated.



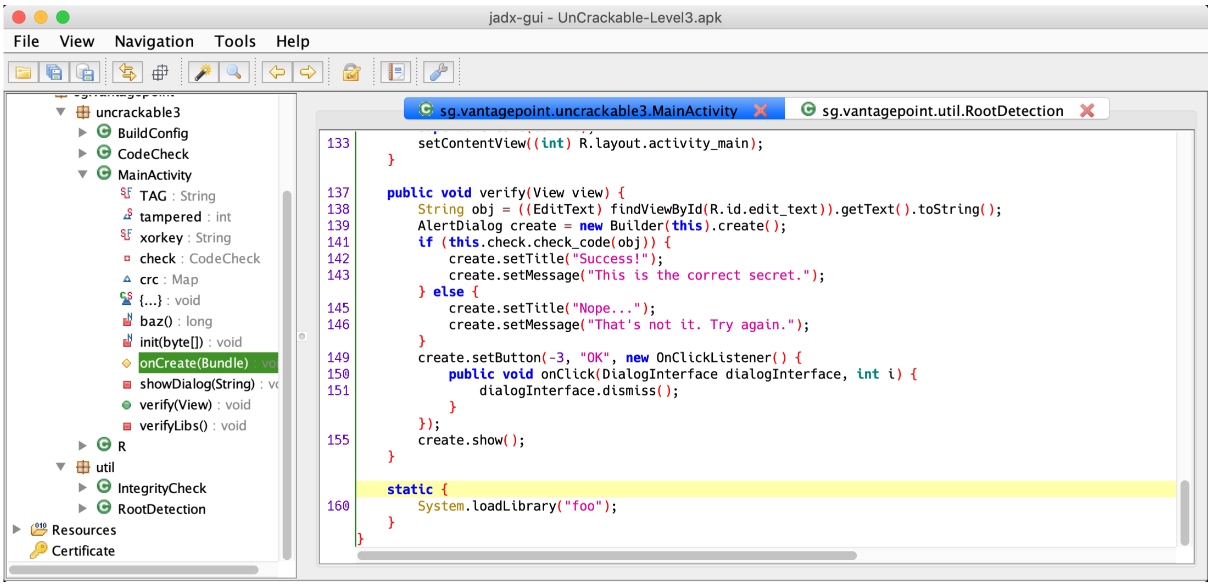
```
protected void onCreate(Bundle bundle) {
    verifyLibs();
    init(xorkey.getBytes());
    new AsyncTask<Void, String, String>() {
        protected String doInBackground(Void... voidArr) {
            while (!Debug.isDebuggerConnected()) {
                SystemClock.sleep(100);
            }
            return null;
        }

        protected void onPostExecute(String str) {
            MainActivity.this.showDialog("Debugger detected!");
            System.exit(0);
        }
    }.execute(new Void[]{null, null, null});
    if (RootDetection.checkRoot1() || RootDetection.checkRoot2() || RootDetection.checkRoot3() || Integ
        showDialog("Rooting or tampering detected.");
    }
    this.check = new CodeCheck();
    super.onCreate(bundle);
    setContentView((int) R.layout.activity_main);
}

public void verify(View view) {
```

Figure 6 - an overview of the application

The entry method (onCreate) performs the following actions:

1. Call verifyLibs()
2. Call init() with a password
3. Start an AsyncTask that keeps running until a debugger is detected. When the task stops, the application exits.
4. Check if the application is rooted
5. Create a new CodeCheck instance. This class is used in the verify() method to see if the entered password is correct.

Additionally, hidden away at the end of the decompiled file, we can see that a static constructor is used to load a native library ("foo"). This code will be run even before the onCreate method is executed.



```
        setContentView((int) R.layout.activity_main);
    }

    public void verify(View view) {
        String obj = ((EditText) findViewById(R.id.edit_text)).getText().toString();
        AlertDialog create = new Builder(this).create();
        if (this.check.check_code(obj)) {
            create.setTitle("Success!");
            create.setMessage("This is the correct secret.");
        } else {
            create.setTitle("Nope...");
            create.setMessage("That's not it. Try again.");
        }
        create.setButton(-3, "OK", new OnClickListener() {
            public void onClick(DialogInterface dialogInterface, int i) {
                dialogInterface.dismiss();
            }
        });
        create.show();
    }

    static {
        System.loadLibrary("foo");
    }
}
```

Figure 7 - loading foo

Let's look at the different controls implemented in the onCreate method. At the very least, there is root detection, debugger detection and resource integrity protection

Debugger detection

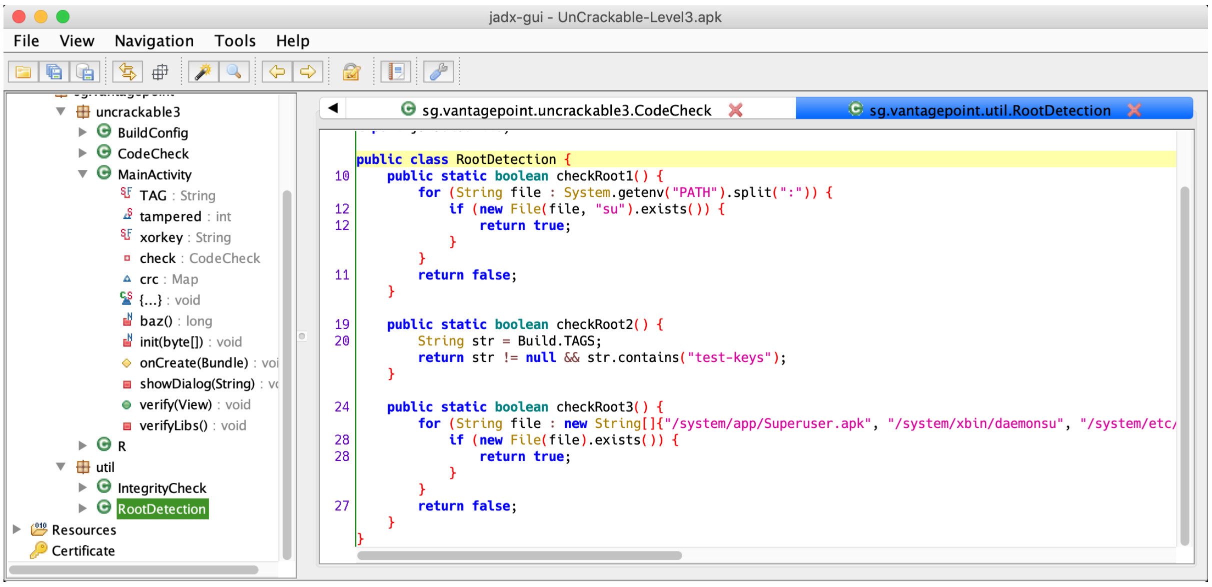
The debugger detector uses a continuously running AsyncTask instead of a one-time-check. The advantage with this type of check is that we can't attach a debugger after the application has performed its startup routine. Of course, we can still get around this by using the techniques we used for the Level 2 app: Changing the SMALI code or hooking Debug.isDebuggerConnected with Frida. However, the anti-debugging code shouldn't hinder our analysis, as Frida doesn't use the Java debugger functionality.

This control is a basic way of implementing 8.3 from OWASP's MASVS:

8.2	The app prevents debugging and/or detects, and responds to, a debugger being attached. All available debugging protocols must be covered.	✓

Root detection control

The root detection looks basic:



```java
public class RootDetection {
    public static boolean checkRoot1() {
        for (String file : System.getenv("PATH").split(":")) {
            if (new File(file, "su").exists()) {
                return true;
            }
        }
        return false;
    }

    public static boolean checkRoot2() {
        String str = Build.TAGS;
        return str != null && str.contains("test-keys");
    }

    public static boolean checkRoot3() {
        for (String file : new String[]{"/system/app/Superuser.apk", "/system/xbin/daemonsu", "/system/etc/
            if (new File(file).exists()) {
                return true;
            }
        }
        return false;
    }
}
```

Figure 8 - The root detection method

There's a test for the "su" binary being on the path of the application, a test for test-keys in the build tags and some file checks for typical files belonging to popular rooting methods. Magisk should easily be able to get past these checks. However, even after enabling Magisk Hide, the application shows a popup about root being detected, and the application is automatically closed:

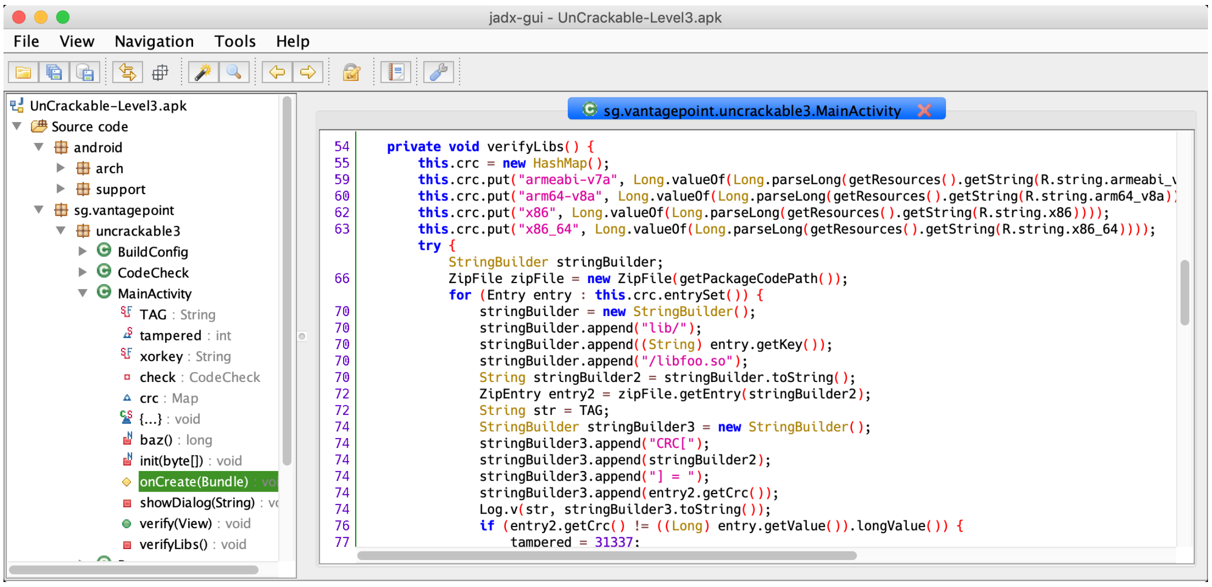
This is unexpected behavior, and we will have to find out what exactly is going wrong.

Root detection is covered in 8.1 of OWASP's MASVS:

8.1	The app detects, and responds to, the presence of a rooted or jailbroken device either by alerting the user or terminating the app.	✓

Resource Integrity Control

JadX-GUI does a good job of reversing the verifyLibs method, and much information can easily be deduced:



```
    private void verifyLibs() {
        this.crc = new HashMap();
        this.crc.put("armeabi-v7a", Long.valueOf(Long.parseLong(getResources().getString(R.string.armeabi_v
        this.crc.put("arm64-v8a", Long.valueOf(Long.parseLong(getResources().getString(R.string.arm64_v8a))
        this.crc.put("x86", Long.valueOf(Long.parseLong(getResources().getString(R.string.x86))));
        this.crc.put("x86_64", Long.valueOf(Long.parseLong(getResources().getString(R.string.x86_64))));
        try {
            StringBuilder stringBuilder;
            ZipFile zipFile = new ZipFile(getPackageCodePath());
            for (Entry entry : this.crc.entrySet()) {
                stringBuilder = new StringBuilder();
                stringBuilder.append("lib/");
                stringBuilder.append((String) entry.getKey());
                stringBuilder.append("/libfoo.so");
                String stringBuilder2 = stringBuilder.toString();
                ZipEntry entry2 = zipFile.getEntry(stringBuilder2);
                String str = TAG;
                StringBuilder stringBuilder3 = new StringBuilder();
                stringBuilder3.append("CRC[");
                stringBuilder3.append(stringBuilder2);
                stringBuilder3.append("] = ");
                stringBuilder3.append(entry2.getCrc());
                Log.v(str, stringBuilder3.toString());
                if (entry2.getCrc() != ((Long) entry.getValue()).longValue()) {
                    tampered = 31337;
```

Figure 9 - The verifyLibs method

Based on the names of the variables and the strings from the error message, we can guess that the application calculates a Cyclic Redundancy Check (CRC) which is a basic hashing method to verify that the resources are unaltered. This CRC code is calculated for all the included binaries, and the classes.dex file itself. This means that if we have to modify classes.dex (SMALI code) or one of the shared libraries (ARM code), we will also have to modify this verifyLib method, as the application won't accept any modified resources. This control is part of OWASP's MASVS[1]:

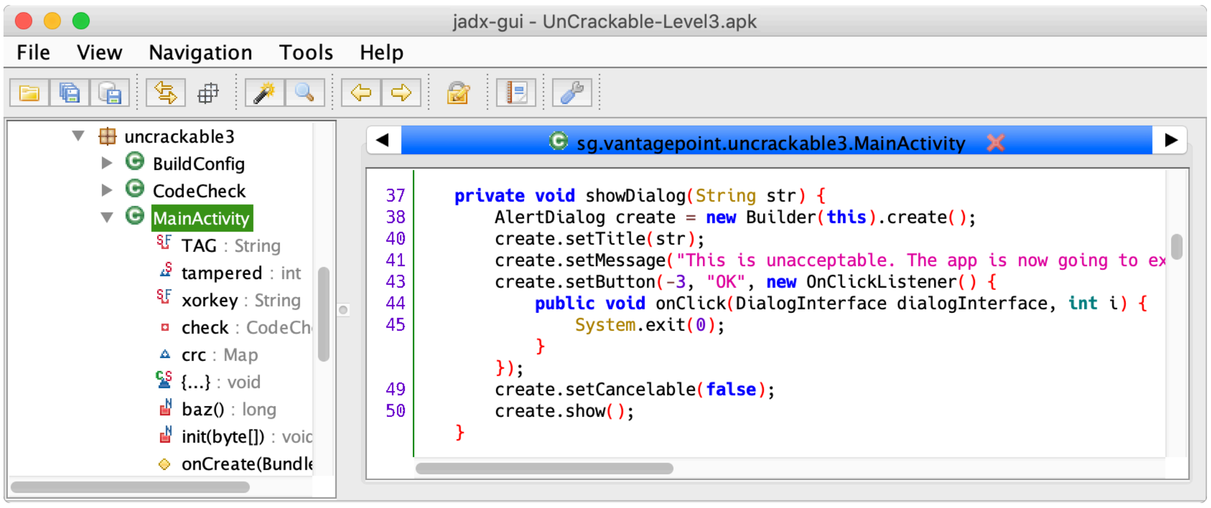
8.3	The app detects, and responds to, tampering with executable files and critical data within its own sandbox.	✓

Since the verifyLibs() code (which is part of classes.dex) actually verifies its own integrity, it would be impossible to include the correct CRC directly in the code, as this would create a loop. Updating the CRC string in the code would modify the CRC of classes.dex, which would have to be updated in the code, …

To prevent this from happening, the code takes the CRC code from the APK file itself and compares it to the CRC code of the unpackaged classes.dex file on the file system. This means that this check will only fail if the classes.dex or one of the SO files is modified directly on the file system. If we would repackage the application, the checks would not be triggered, since the CRC code of the APK itself would automatically be updated.

Surprisingly, the application is complaining that the CRC of the classes.dex file is not correct, even though we didn't modify it. This is unexpected behavior, but we can get around this by modifying the behavior of the showDialog method. After all, this method forces the application to close, but if we just make sure that the showDialog method doesn't do anything, nothing will happen when either the root detection or the CRC checks fail.

[1] https://github.com/OWASP/owasp-masvs/blob/master/Document/0x15-V8-Resiliency_Against_Reverse_Engineering_Requirements.md#impede-dynamic-analysis-and-tampering



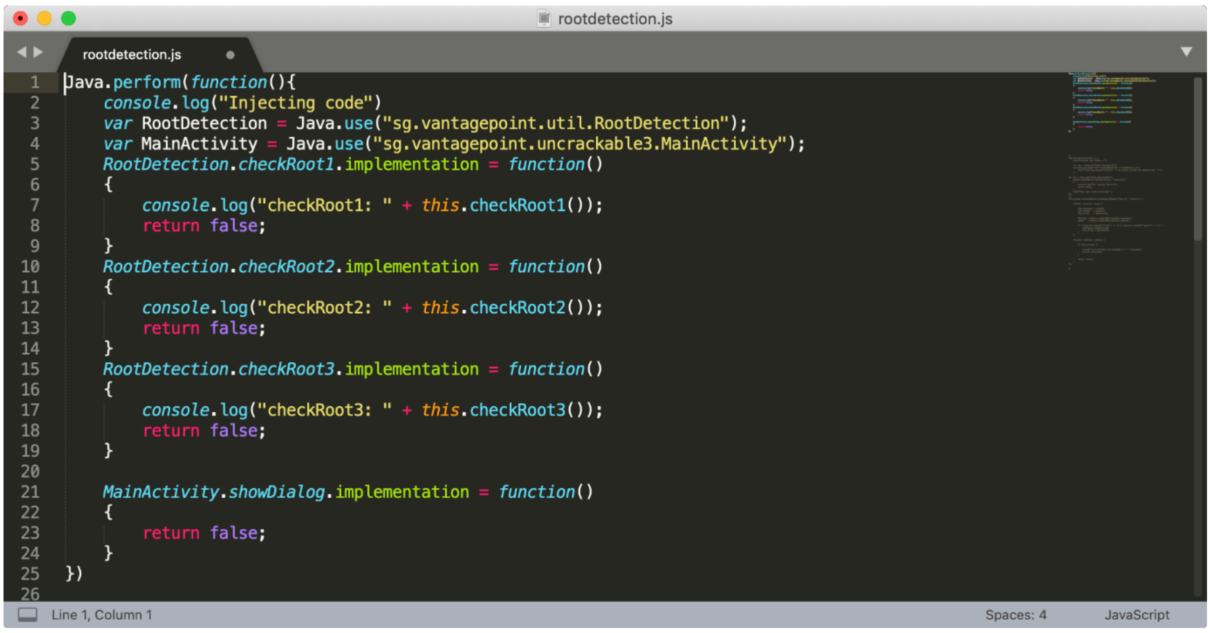
```java
    private void showDialog(String str) {
        AlertDialog create = new Builder(this).create();
        create.setTitle(str);
        create.setMessage("This is unacceptable. The app is now going to ex
        create.setButton(-3, "OK", new OnClickListener() {
            public void onClick(DialogInterface dialogInterface, int i) {
                System.exit(0);
            }
        });
        create.setCancelable(false);
        create.show();
    }
```

Figure 10 - The showDialog function causes the application to exit

3.3 Instrumenting the application

Disabling tamper detection

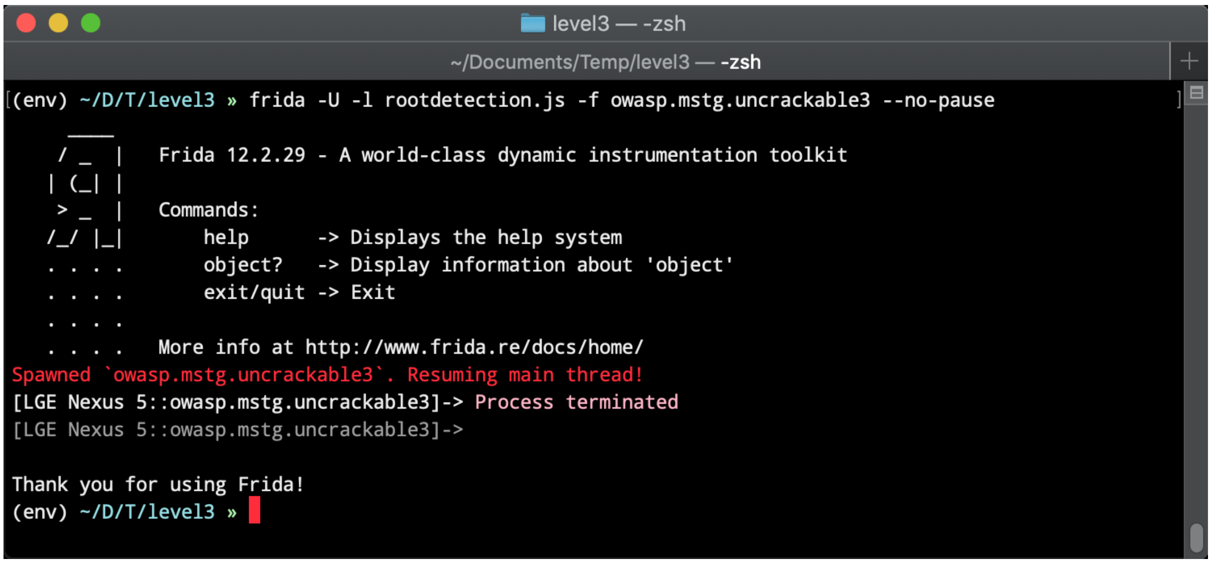
Let's hook the rooting methods with Frida and do the same for the showDialog function so that it doesn't do anything:



```javascript
Java.perform(function(){
    console.log("Injecting code")
    var RootDetection = Java.use("sg.vantagepoint.util.RootDetection");
    var MainActivity = Java.use("sg.vantagepoint.uncrackable3.MainActivity");
    RootDetection.checkRoot1.implementation = function()
    {
        console.log("checkRoot1: " + this.checkRoot1());
        return false;
    }
    RootDetection.checkRoot2.implementation = function()
    {
        console.log("checkRoot2: " + this.checkRoot2());
        return false;
    }
    RootDetection.checkRoot3.implementation = function()
    {
        console.log("checkRoot3: " + this.checkRoot3());
        return false;
    }

    MainActivity.showDialog.implementation = function()
    {
        return false;
    }
})
```

Figure 11 - Altering the application behaviour using Frida hooks

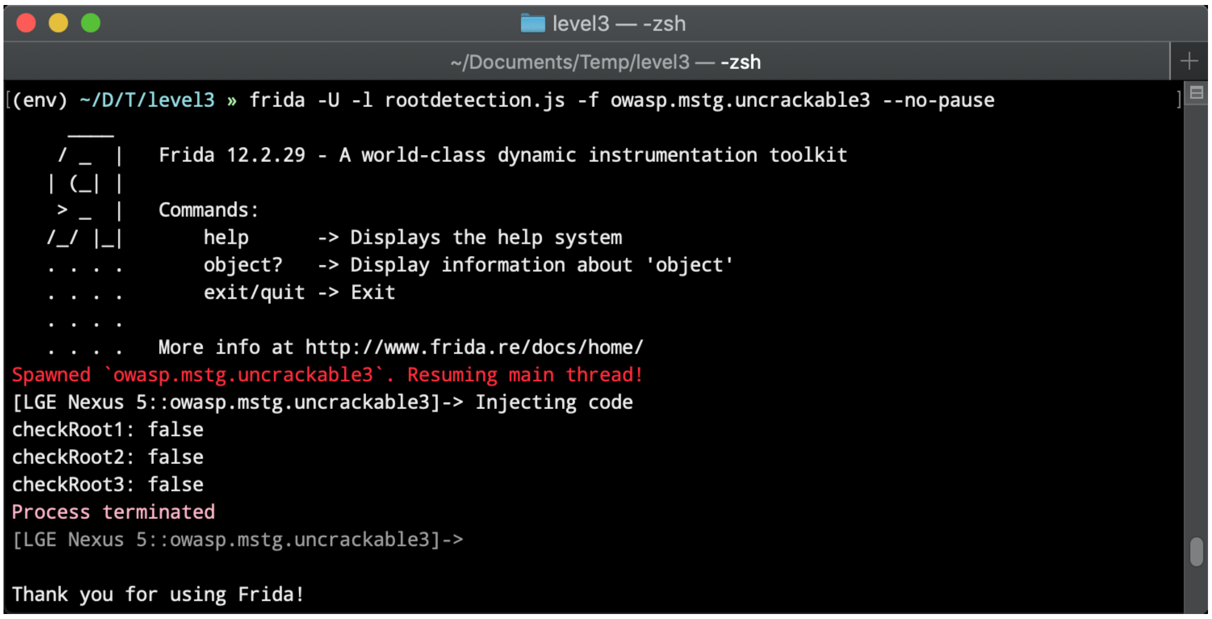
Running the application with the injected code does not have the expected result, as it crashes even before our Frida code is executed:



```
level3 — -zsh
~/Documents/Temp/level3 — -zsh
[(env) ~/D/T/level3 » frida -U -l rootdetection.js -f owasp.mstg.uncrackable3 --no-pause
     ____
    / _  |   Frida 12.2.29 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at http://www.frida.re/docs/home/
Spawned `owasp.mstg.uncrackable3`. Resuming main thread!
[LGE Nexus 5::owasp.mstg.uncrackable3]-> Process terminated
[LGE Nexus 5::owasp.mstg.uncrackable3]->

Thank you for using Frida!
(env) ~/D/T/level3 »
```

Figure 12 - The app crashes before Frida gets a chance

Running the same command a few times eventually prints out our Hooks, but even then, the application crashes:



```
level3 — -zsh
~/Documents/Temp/level3 — -zsh
[(env) ~/D/T/level3 » frida -U -l rootdetection.js -f owasp.mstg.uncrackable3 --no-pause
     ____
    / _  |   Frida 12.2.29 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at http://www.frida.re/docs/home/
Spawned `owasp.mstg.uncrackable3`. Resuming main thread!
[LGE Nexus 5::owasp.mstg.uncrackable3]-> Injecting code
checkRoot1: false
checkRoot2: false
checkRoot3: false
Process terminated
[LGE Nexus 5::owasp.mstg.uncrackable3]->

Thank you for using Frida!
```

Figure 13 - The hooks are printed after a few attempts

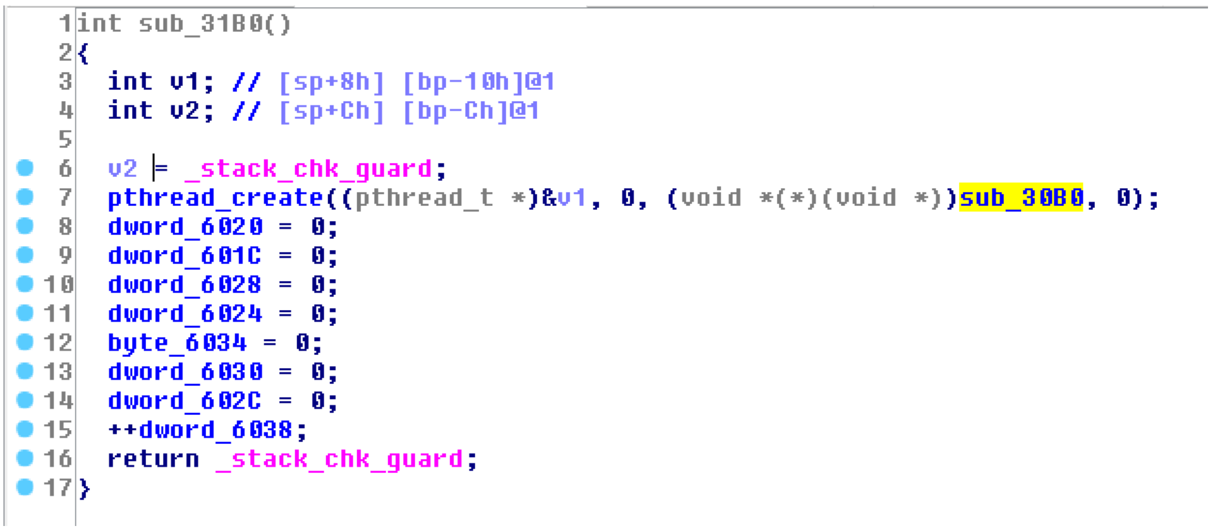
This could mean that there is a race condition between Java and the native library that is loaded, but it's still not possible to say if the application crashes on purpose as a result of it detecting Frida, or if we are facing some kind of bug related to Frida. After all, dynamic instrumentation is pretty complex, and many things can go wrong.

On the upside, we can see that the checkRoot methods did not trigger the popup window, so the CRC check is probably responsible for the root checks.

Examining the native library

The application is built for all platforms, so we can choose which native library we want to examine. For this case study, we will take the x86 version and load it into IDA Pro. After using apktool to decompose the APK, we can find the x86 library in the /lib/x86 folder.

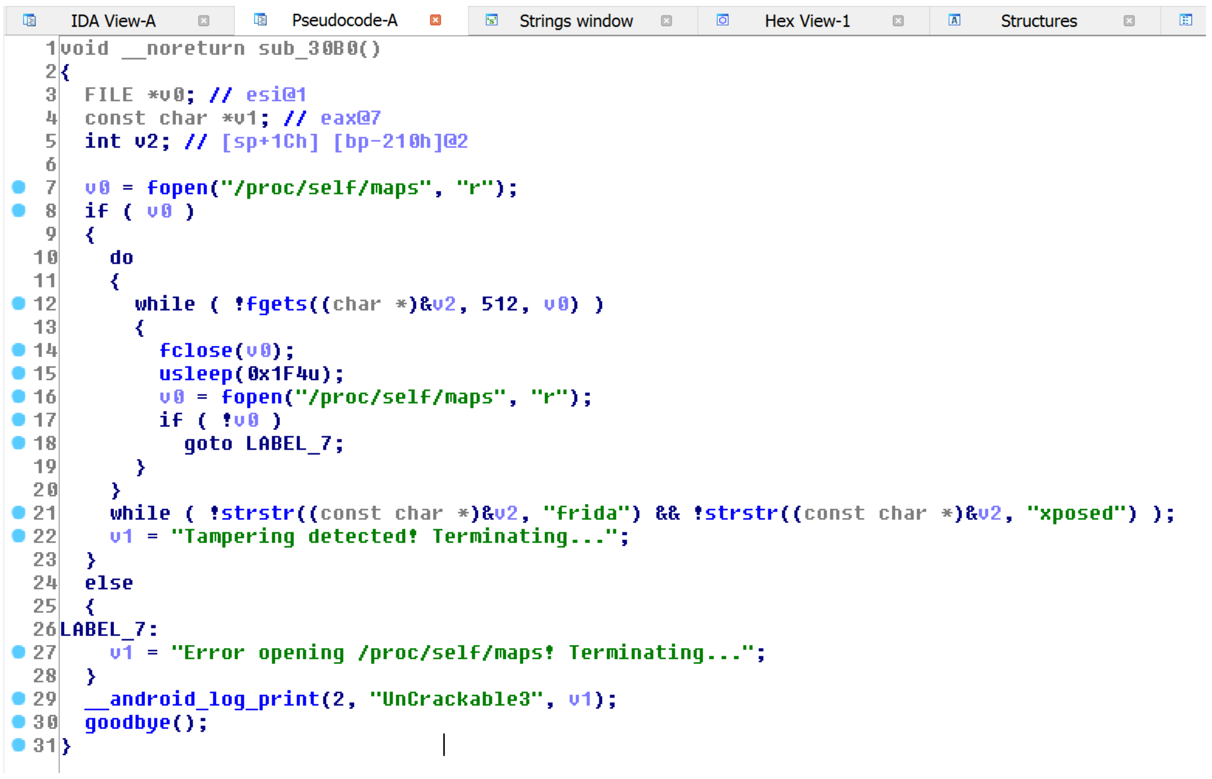
Browsing around the .so file, we can see that there is a static initializer method that gets called when the library is loaded:



```
int sub_31B0()
{
  int v1; // [sp+8h] [bp-10h]@1
  int v2; // [sp+Ch] [bp-Ch]@1

  v2 = _stack_chk_guard;
  pthread_create((pthread_t *)&v1, 0, (void *(*)(void *))sub_30B0, 0);
  dword_6020 = 0;
  dword_601C = 0;
  dword_6028 = 0;
  dword_6024 = 0;
  byte_6034 = 0;
  dword_6030 = 0;
  dword_602C = 0;
  ++dword_6038;
  return _stack_chk_guard;
}
```

Figure 14 - The initializer of the shared library

This method initializes some variables to 0 and spawns a new thread which will run sub_30B0. Let's take a closer look at that method:

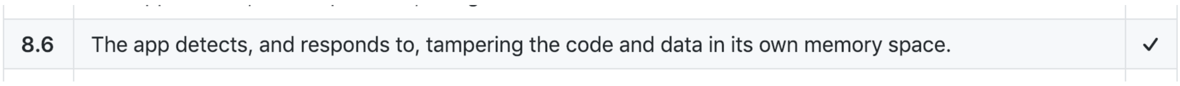


```
void __noreturn sub_30B0()
{
  FILE *v0; // esi@1
  const char *v1; // eax@7
  int v2; // [sp+1Ch] [bp-210h]@2

  v0 = fopen("/proc/self/maps", "r");
  if ( v0 )
  {
    do
    {
      while ( !fgets((char *)&v2, 512, v0) )
      {
        fclose(v0);
        usleep(0x1F4u);
        v0 = fopen("/proc/self/maps", "r");
        if ( !v0 )
          goto LABEL_7;
      }
    }
    while ( !strstr((const char *)&v2, "frida") && !strstr((const char *)&v2, "xposed") );
    v1 = "Tampering detected! Terminating...";
  }
  else
  {
LABEL_7:
    v1 = "Error opening /proc/self/maps! Terminating...";
  }
  __android_log_print(2, "UnCrackable3", v1);
  goodbye();
}
```

Figure 15 - sub_30B0 method

It's easy to follow the logic of this method. It opens /proc/self/maps, and searches for any line containing either 'frida' or 'xposed'. The maps file contains all the mapped memory regions of the application, and any injected framework is likely to be included in the list. Since we are using Frida, it will no doubt detect the injected library and quit the application through the goodbye() method.

This control is mentioned as 8.6 in the MASVS:



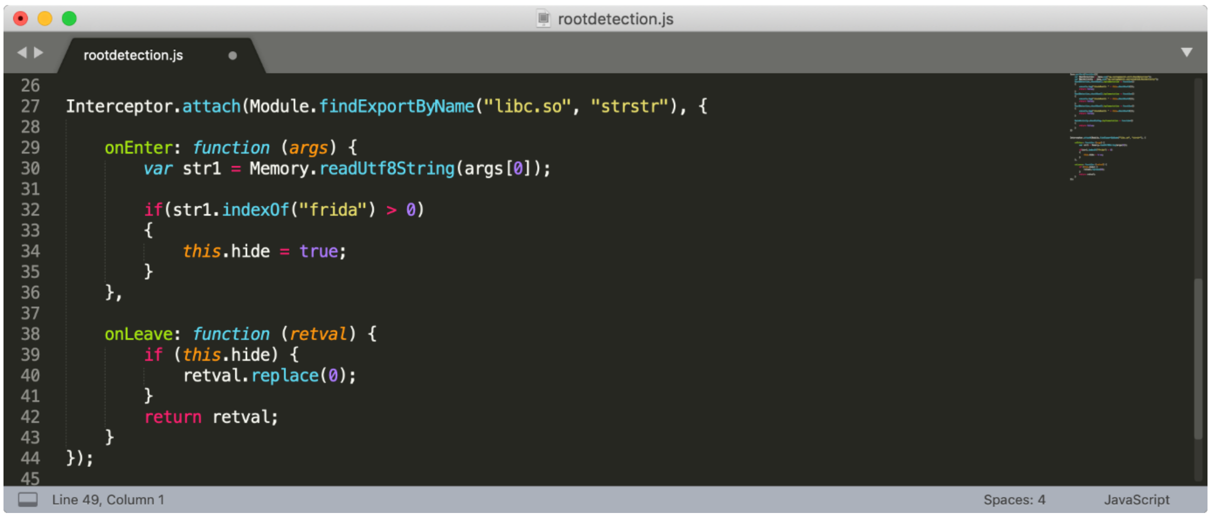
8.6	The app detects, and responds to, tampering the code and data in its own memory space.	✓

There are three basic options to get around this control:

- We modify the name of the Frida library
- We modify the code and repackage the application
- We change the code at runtime using Frida.

Renaming the library should be the easiest approach, but Frida doesn't support automated renaming yet, and the library's name is currently hardcoded in a few locations, making it an inconvenient solution. The next best thing is using Frida to modify the behavior at runtime and using Frida to get around anti-Frida behavior sounds like fun.

We can hook into the "strstr" function, which is part of libc and figure out if it is detecting the Frida library somewhere:



```javascript

Interceptor.attach(Module.findExportByName("libc.so", "strstr"), {

    onEnter: function (args) {
        var str1 = Memory.readUtf8String(args[0]);

        if(str1.indexOf("frida") > 0)
        {
            this.hide = true;
        }
    },

    onLeave: function (retval) {
        if (this.hide) {
            retval.replace(0);
        }
        return retval;
    }
});

```

Figure 16 - Using Frida to hook into the strstr function

Using Frida's Interceptor class, we can have a function called when entering and exiting the strstr method. We can keep track of when the method would return true, and then change the return value once the method returns.

After this, we can finally use the application and test some input:

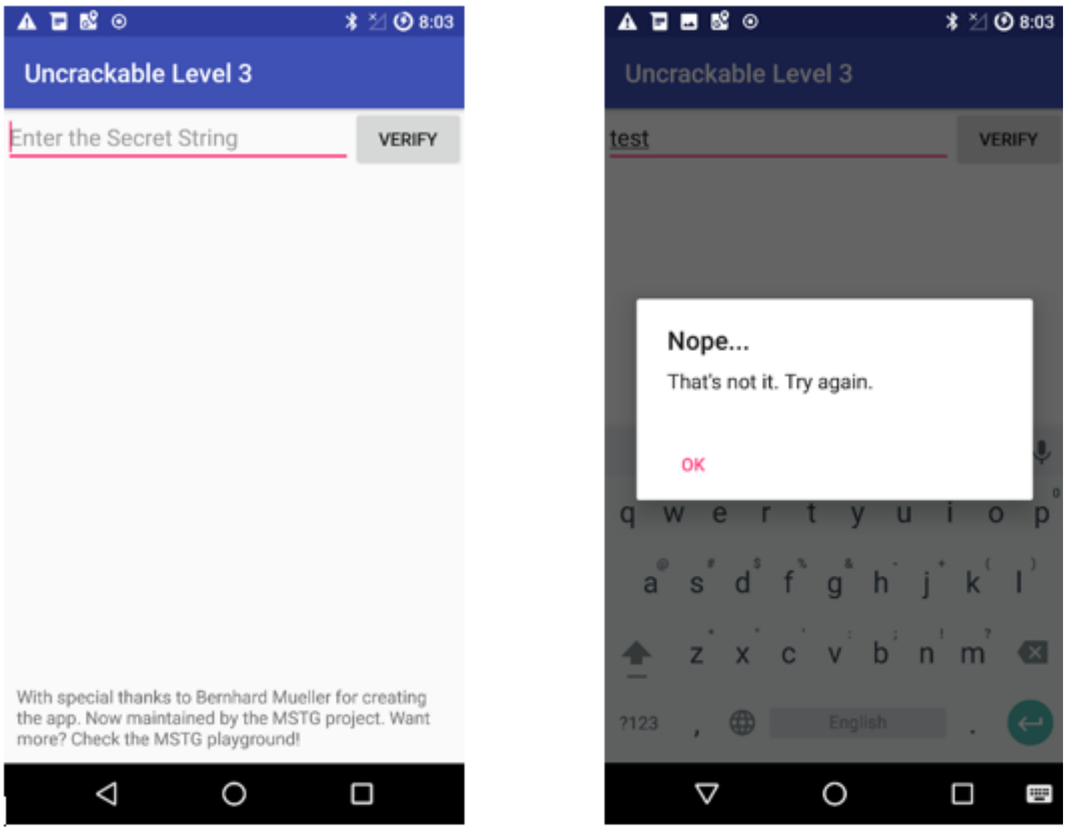
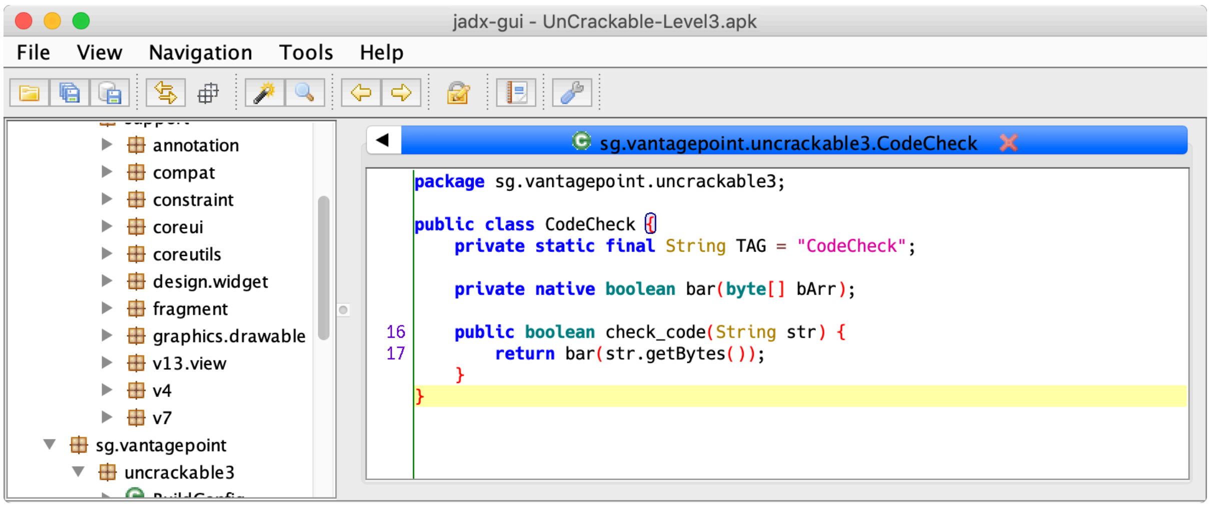

Uncrackable Level 3
Enter the Secret String
VERIFY
With special thanks to Bernhard Mueller for creating the app. Now maintained by the MSTG project. Want more? Check the MSTG playground!
test
Nope...
That's not it. Try again.
OK

Figure 17 - The application functions again

Extracting the password

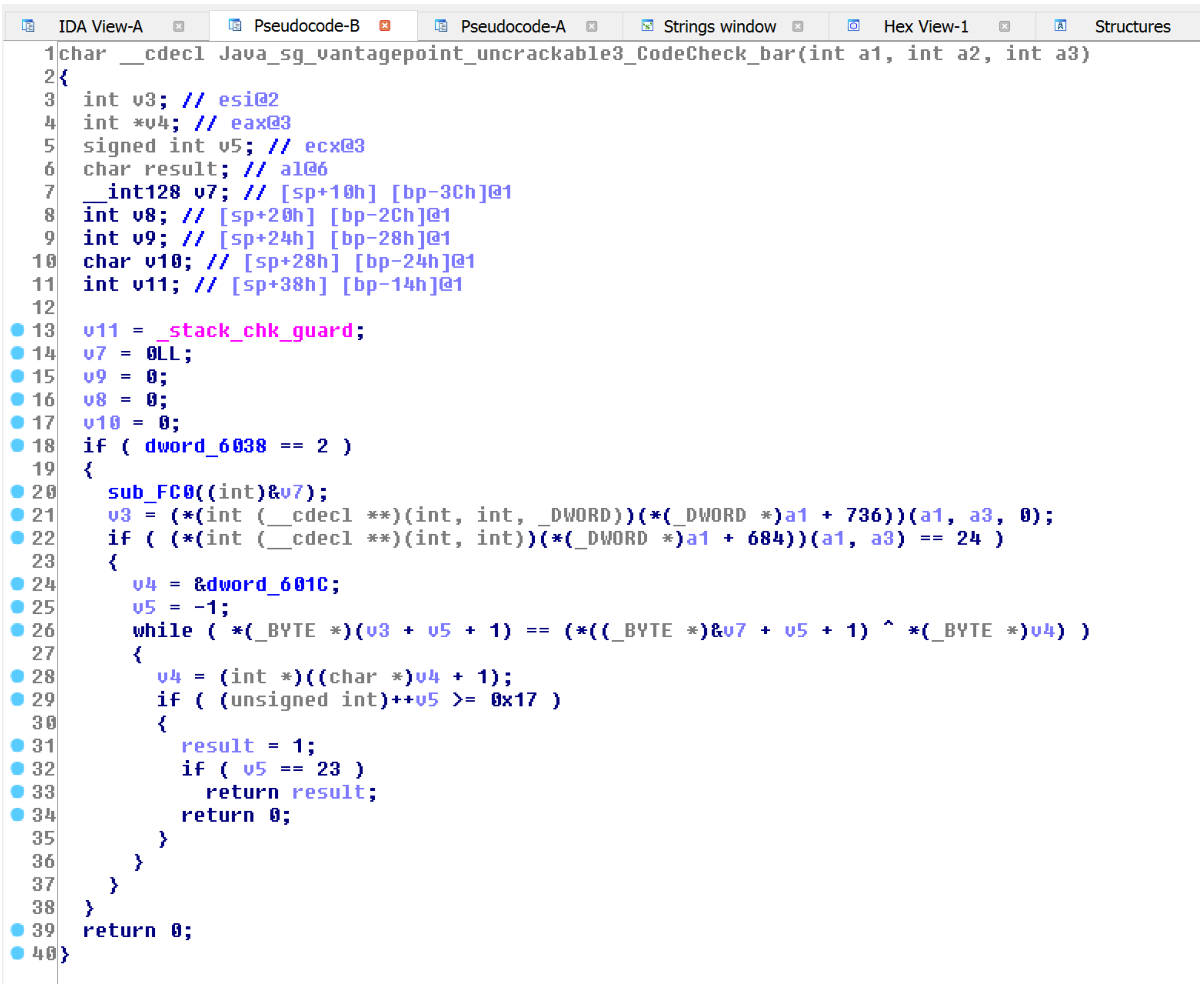
The entered password is passed to CodeCheck.check_code() which then passes it to the native bar method:



```java
package sg.vantagepoint.uncrackable3;

public class CodeCheck {
    private static final String TAG = "CodeCheck";

    private native boolean bar(byte[] bArr);

    public boolean check_code(String str) {
        return bar(str.getBytes());
    }
}
```

Figure 18 - The check_code method

In IDA, the CodeCheck.bar method looks as follows:



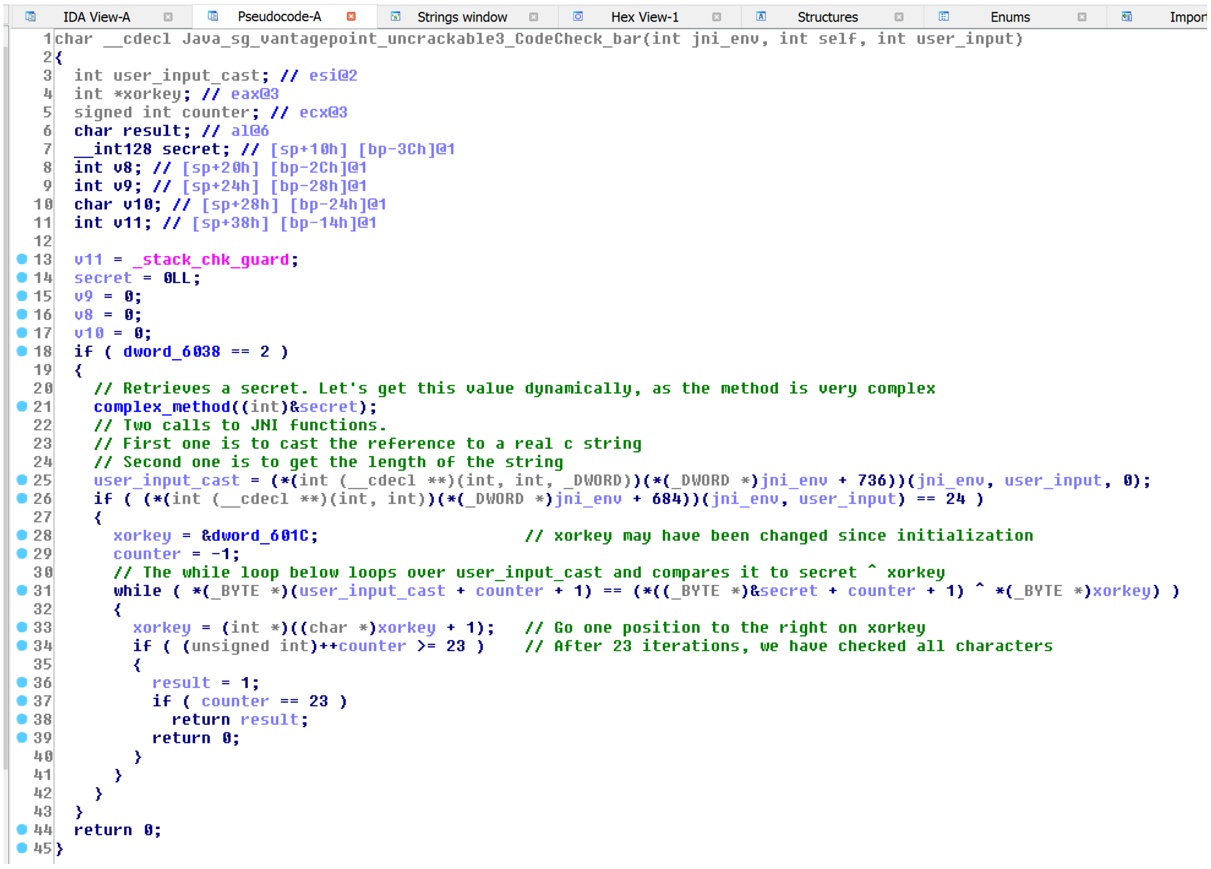
```
char __cdecl Java_sg_vantagepoint_uncrackable3_CodeCheck_bar(int a1, int a2, int a3)
{
  int v3; // esi@2
  int *v4; // eax@3
  signed int v5; // ecx@3
  char result; // al@6
  __int128 v7; // [sp+10h] [bp-3Ch]@1
  int v8; // [sp+20h] [bp-2Ch]@1
  int v9; // [sp+24h] [bp-28h]@1
  char v10; // [sp+28h] [bp-24h]@1
  int v11; // [sp+38h] [bp-14h]@1

  v11 = _stack_chk_guard;
  v7 = 0LL;
  v9 = 0;
  v8 = 0;
  v10 = 0;
  if ( dword_6038 == 2 )
  {
    sub_FC0((int)&v7);
    v3 = (*(int (__cdecl **)(int, int, _DWORD))(*(_DWORD *)a1 + 736))(a1, a3, 0);
    if ( (*(int (__cdecl **)(int, int))(*(_DWORD *)a1 + 684))(a1, a3) == 24 )
    {
      v4 = &dword_601C;
      v5 = -1;
      while ( *(_BYTE *)(v3 + v5 + 1) == (*((_BYTE *)&v7 + v5 + 1) ^ *(_BYTE *)v4) )
      {
        v4 = (int *)((char *)v4 + 1);
        if ( (unsigned int)++v5 >= 0x17 )
        {
          result = 1;
          if ( v5 == 23 )
            return result;
          return 0;
        }
      }
    }
  }
  return 0;
}
```

Figure 19 - The native Codecheck_bar method

This is a little bit cryptic, but with a little bit of deduction, we can figure some stuff out:

- The arguments for a JNI method are a pointer to the JNI environment, the object and any additional arguments that the method receives
- The v5 variable starts at -1 and is updated each iteration in the while loop
- There is a check for something to equal 24, and then there is a while loop that stops when the v5 counter has reached 24. Most likely, the correct password is 24 characters long.
- In the while loop, the applied logic is v3 == v7 ^ v4, for each character in the variables.
- v4 is a reference to the pizza variable that has been configured by the application earlier through the MainActivity.init method. The variable might have undergone a transformation, so let's find out the actual value when the password is compared
- The v7 value is populated from within the sub_FC0 function.

With a little bit of renaming, we end up at the following:

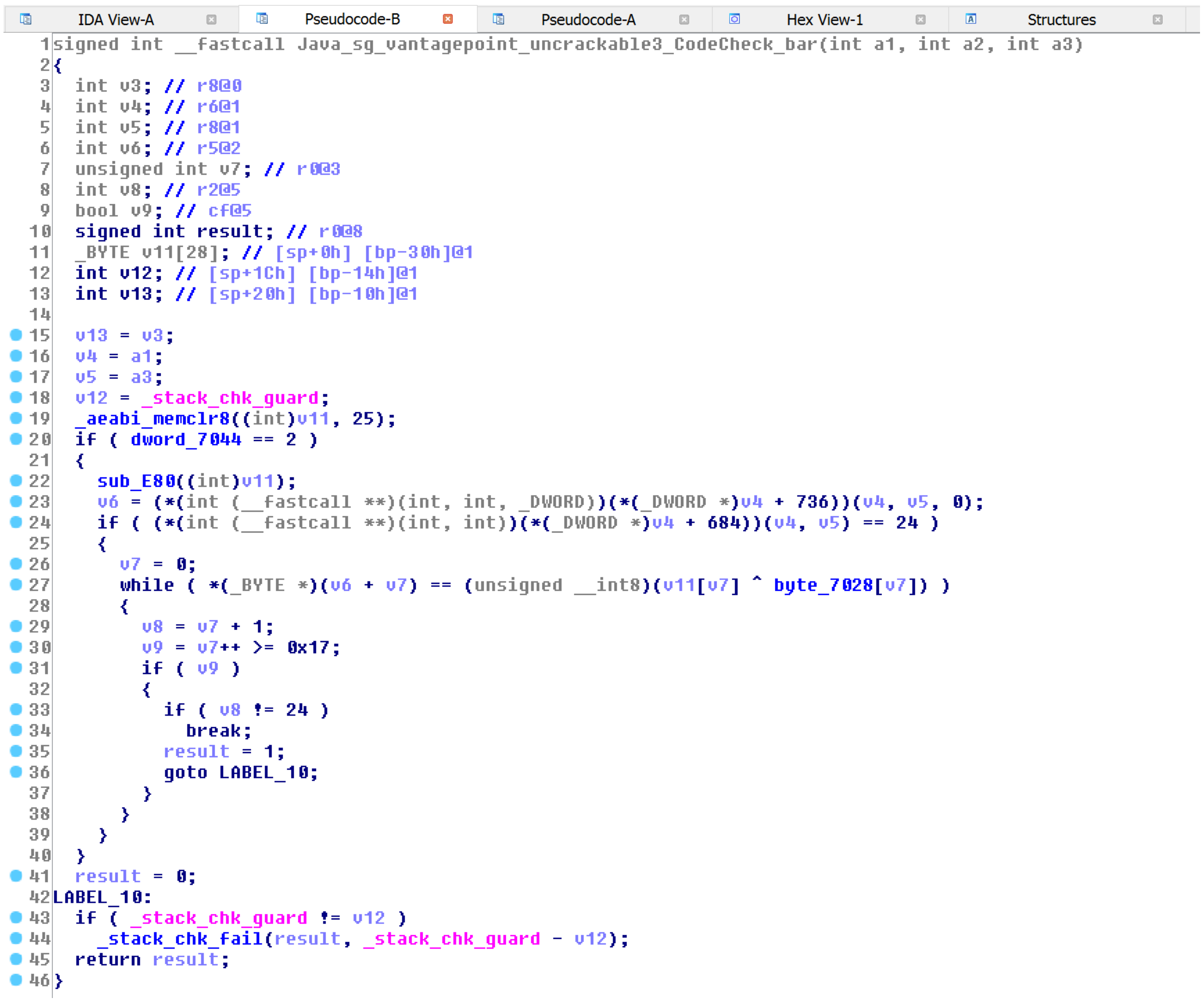


```
char __cdecl Java_sg_vantagepoint_uncrackable3_CodeCheck_bar(int jni_env, int self, int user_input)
{
  int user_input_cast; // esi@2
  int *xorkey; // eax@3
  signed int counter; // ecx@3
  char result; // al@6
  __int128 secret; // [sp+10h] [bp-3Ch]@1
  int v8; // [sp+20h] [bp-2Ch]@1
  int v9; // [sp+24h] [bp-28h]@1
  char v10; // [sp+28h] [bp-24h]@1
  int v11; // [sp+38h] [bp-14h]@1

  v11 = _stack_chk_guard;
  secret = 0LL;
  v9 = 0;
  v8 = 0;
  v10 = 0;
  if ( dword_6038 == 2 )
  {
    // Retrieves a secret. Let's get this value dynamically, as the method is very complex
    complex_method((int)&secret);
    // Two calls to JNI functions.
    // First one is to cast the reference to a real c string
    // Second one is to get the length of the string
    user_input_cast = (*(int (__cdecl **)(int, int, _DWORD))(*(_DWORD *)jni_env + 736))(jni_env, user_input, 0);
    if ( (*(int (__cdecl **)(int, int))(*(_DWORD *)jni_env + 684))(jni_env, user_input) == 24 )
    {
      xorkey = &dword_601C;                      // xorkey may have been changed since initialization
      counter = -1;
      // The while loop below loops over user_input_cast and compares it to secret ^ xorkey
      while ( *(_BYTE *)(user_input_cast + counter + 1) == (*((_BYTE *)&secret + counter + 1) ^ *(_BYTE *)xorkey) )
      {
        xorkey = (int *)((char *)xorkey + 1);   // Go one position to the right on xorkey
        if ( (unsigned int)++counter >= 23 )    // After 23 iterations, we have checked all characters
        {
          result = 1;
          if ( counter == 23 )
            return result;
          return 0;
        }
      }
    }
  }
  return 0;
}
```

Figure 20 - The modified method with some comments to keep track of the code

To solve this, we need to figure out the xorkey and the secret that is generated by the complex_method. We can use the same Interceptor class to hook onto the complex_method, but since it's not exported, we must figure out where it is loaded into memory. Unfortunately, we can't use the offsets of the x86 binary, as the application will load the armv7 binary on the device, resulting in different offsets

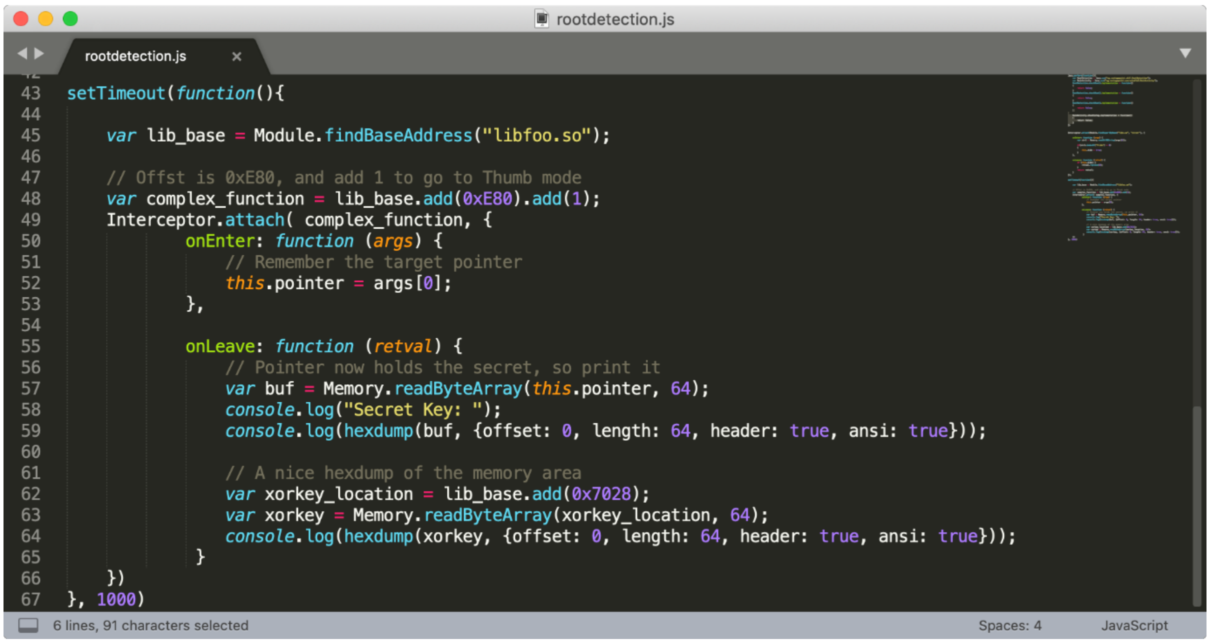
The same CodeCheck_bar function in the armv7 binary looks as follows:



```
signed int __fastcall Java_sg_vantagepoint_uncrackable3_CodeCheck_bar(int a1, int a2, int a3)
{
  int v3; // r8@0
  int v4; // r6@1
  int v5; // r8@1
  int v6; // r5@2
  unsigned int v7; // r0@3
  int v8; // r2@5
  bool v9; // cf@5
  signed int result; // r0@8
  _BYTE v11[28]; // [sp+0h] [bp-30h]@1
  int v12; // [sp+1Ch] [bp-14h]@1
  int v13; // [sp+20h] [bp-10h]@1

  v13 = v3;
  v4 = a1;
  v5 = a3;
  v12 = _stack_chk_guard;
  _aeabi_memclr8((int)v11, 25);
  if ( dword_7044 == 2 )
  {
    sub_E80((int)v11);
    v6 = (*(int (__fastcall **)(int, int, _DWORD))(*(_DWORD *)v4 + 736))(v4, v5, 0);
    if ( (*(int (__fastcall **)(int, int))(*(_DWORD *)v4 + 684))(v4, v5) == 24 )
    {
      v7 = 0;
      while ( *(_BYTE *)(v6 + v7) == (unsigned __int8)(v11[v7] ^ byte_7028[v7]) )
      {
        v8 = v7 + 1;
        v9 = v7++ >= 0x17;
        if ( v9 )
        {
          if ( v8 != 24 )
            break;
          result = 1;
          goto LABEL_10;
        }
      }
    }
  }
  result = 0;
LABEL_10:
  if ( _stack_chk_guard != v12 )
    _stack_chk_fail(result, _stack_chk_guard - v12);
  return result;
}
```

Figure 21 - The CodeCheck_bar function in the amv7 binary

The offset of complex_function is now 0xE80 (hence the name 'sub_E80') and the xorkey, *byte_7028*, is located at offset 0x7028. We can use this knowledge to print out the content of v11 after sub_E80 has been called, and the value of byte_7028 once the code is executed:



```
setTimeout(function(){

    var lib_base = Module.findBaseAddress("libfoo.so");

    // Offst is 0xE80, and add 1 to go to Thumb mode
    var complex_function = lib_base.add(0xE80).add(1);
    Interceptor.attach( complex_function, {
            onEnter: function (args) {
                // Remember the target pointer
                this.pointer = args[0];
            },

            onLeave: function (retval) {
                // Pointer now holds the secret, so print it
                var buf = Memory.readByteArray(this.pointer, 64);
                console.log("Secret Key: ");
                console.log(hexdump(buf, {offset: 0, length: 64, header: true, ansi: true}));

                // A nice hexdump of the memory area
                var xorkey_location = lib_base.add(0x7028);
                var xorkey = Memory.readByteArray(xorkey_location, 64);
                console.log(hexdump(xorkey, {offset: 0, length: 64, header: true, ansi: true}));
             }
    })
}, 1000)
```

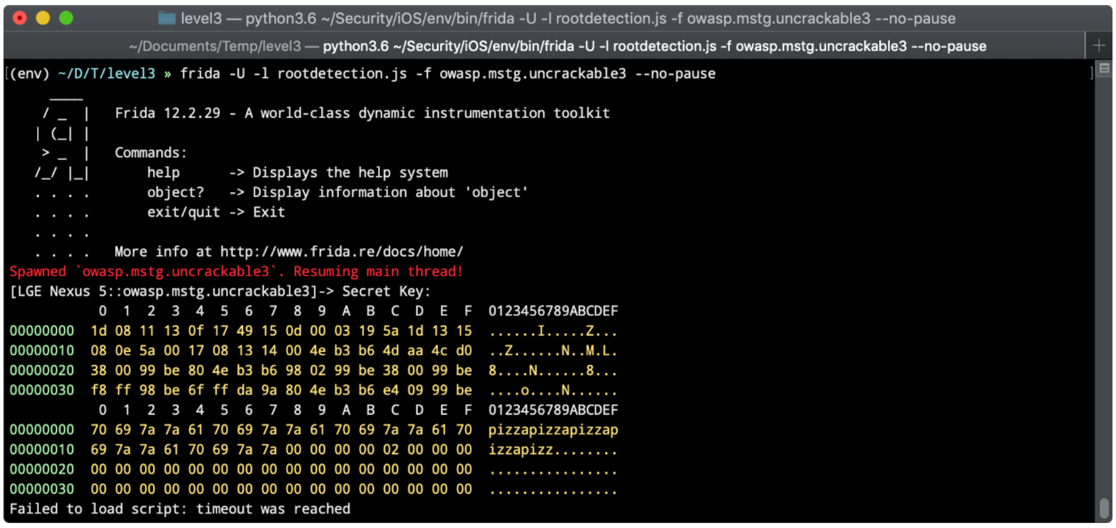
Figure 22 - Attaching to the complex_function

When the complex_function is called, we remember the location of the given pointer. When the function exits, the pointer will contain the correct secret and we can print it, together with the xorkey.

One little trick here is that we actually have to use the address 0xE81, as the least significant bit of the address determines whether the code is in Thumb or ARM mode. Since the code is in Thumb mode, we have to set it to 1[2].

Finally, there is a setTimeout around the Interceptor code, as we need to wait for the library to be loaded, otherwise we won't get a valid value from Module.findBaseAddress.

Running this hook, together with all the other ones results in the following execution:



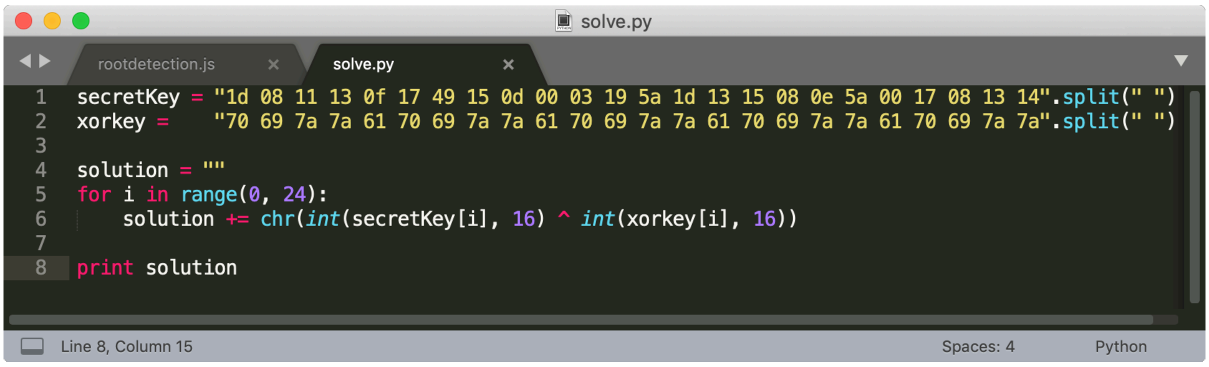
```
level3 — python3.6 ~/Security/iOS/env/bin/frida -U -l rootdetection.js -f owasp.mstg.uncrackable3 --no-pause
~/Documents/Temp/level3 — python3.6 ~/Security/iOS/env/bin/frida -U -l rootdetection.js -f owasp.mstg.uncrackable3 --no-pause
(env) ~/D/T/level3 » frida -U -l rootdetection.js -f owasp.mstg.uncrackable3 --no-pause
     ____
    / _  |   Frida 12.2.29 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at http://www.frida.re/docs/home/
Spawned `owasp.mstg.uncrackable3`. Resuming main thread!
[LGE Nexus 5::owasp.mstg.uncrackable3]-> Secret Key:
           0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
00000000  1d 08 11 13 0f 17 49 15 0d 00 03 19 5a 1d 13 15  ......I.....Z...
00000010  08 0e 5a 00 17 08 13 14 00 4e b3 b6 4d aa 4c d0  ..Z......N..M.L.
00000020  38 00 99 be 80 4e b3 b6 98 02 99 be 38 00 99 be  8....N......8...
00000030  f8 ff 98 be 6f ff da 9a 80 4e b3 b6 e4 09 99 be  ....o....N......
           0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
00000000  70 69 7a 7a 61 70 69 7a 7a 61 70 69 7a 7a 61 70  pizzapizzapizzap
00000010  69 7a 7a 61 70 69 7a 7a 00 00 00 00 02 00 00 00  izzapizz........
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
Failed to load script: timeout was reached
```

Figure 23 - Printing the xorkey and the secret key

[2] https://www.frida.re/docs/javascript-api/#interceptor

We can see that the xorkey is still a repetition of the word “pizza”, so it hasn’t been altered since it was initialized. The secret key is a random hexadecimal string without any other obvious meaning.

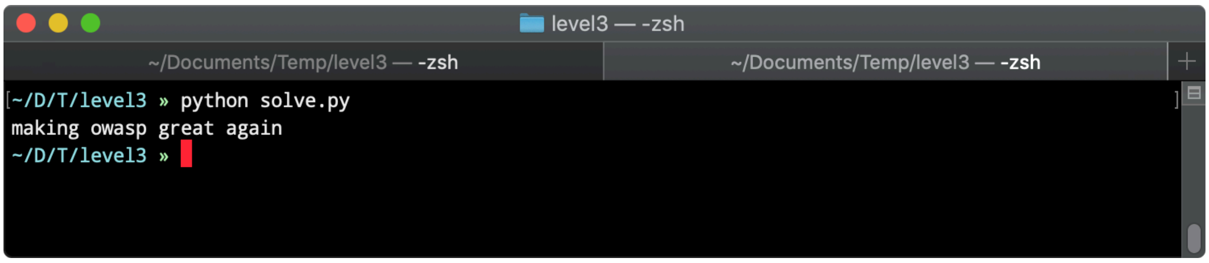
As a final step, we have to XOR the xorkey with the secret key in order to get the actual password:



```python
secretKey = "1d 08 11 13 0f 17 49 15 0d 00 03 19 5a 1d 13 15 08 0e 5a 00 17 08 13 14".split(" ")
xorkey =    "70 69 7a 7a 61 70 69 7a 7a 61 70 69 7a 7a 61 70 69 7a 7a 61 70 69 7a 7a".split(" ")

solution = ""
for i in range(0, 24):
    solution += chr(int(secretKey[i], 16) ^ int(xorkey[i], 16))

print solution
```

Figure 24 - XOR the xorkey and secret key

Running this script prints the flag:



```
[~/D/T/level3 » python solve.py
making owasp great again
~/D/T/level3 »
```

Figure 25 - The resulting password of the previous operation

3.4 Final thoughts

This application has incorporated much more anti-reverse engineering controls than the Level 2 version. In fact, the application has also implemented additional security controls which we have not circumvented because they did not interfere with our specific approach towards extracting the password. For example, the AsyncTask described in the beginning of this case study was never triggered, since we were never using the any Java debugging techniques.

If it is important to impede the comprehension of your application, it is important to cover a vast array of controls, each targeting different possible entry points into the application. Having strong anti-reverse engineering controls comes down to:

- Having many of them. The more hurdles, the more likely the attacker will give up
- Being creative. Implementing custom or lesser known controls will for example prevent default Frida scripts from working.

Of course, obfuscation and anti-reverse engineering controls are never a substitute for real security and their usage depends much on the type of application. A good example of this is Pokemon Go, which has implemented very strong controls to prevent people from cheating in the game and using bots to gather Pokemon. As it is nearly impossible to separate a normal user from a bot based on the data received by the backend, Niantec has put much effort into preventing users from understanding how the communication with the backend works.

More information on the different controls implemented by the Level 3 Uncrackable can be found in chapter 8 of OWASP’s MASVS.

4 Operating systems updates

4.1 iOS security update

Apple released one security software update for its mobile devices this month, iOS 12.1.3

iOS 12.1.3 bundles patches for Apple KeyStore, Bluetooth, Kernel, Web Kit, Facetime… and fixes a multitude of vulnerabilities including remote code execution and escalation of privilege attacks.

Some of the noteworthy vulnerabilities fixed are listed below:

- Apple KeyStore CVE-2019-6235: A sandboxed process may be able to circumvent sandbox restrictions
- Bluetooth CVE-2019-6200: An attacker in a privileged network position may be able to use Bluetooth for code execution
- Core Media CVE-2019-6202/6221: Malicious applications may be able to do privilege escalation
- Keyboard CVE-2019-6206: Password autofill may fill in passwords after they were cleared manually
- Safari Reader CVE-2019-6228: maliciously crafted web content could lead to cross site scripting

The full list can be found in the patch notes via the following URL:

https://support.apple.com/en-us/HT209443

4.2 Android security update

The Android security team fixed 27 vulnerabilities in their January security updates. Two of the bugs were assigned a critical risk, one is a system RCE flaw which could enable remote attackers to execute arbitrary code within the context of privileged processes by using specially crafted files. The second critical risk vulnerability was found in a closed-source component of Qualcomm but could not be directly exploited.

Google did not find any indication that the bugs have been exploited or abused in the wild.

All other 25 bugs were classified as High and have been patched. Most of the vulnerabilities were found in system and kernel components.

Out of all these bugs, 13 were privilege escalations, 8 were information disclosures and the remainders are part of closed source components and could not be exploited.

More information can be found on the January 2019 security bulletin:

https://source.Android.com/security/bulletin/2019-01-01

5 Mobile security news

5.1 Mobile operating system security

Beauty Camera apps send users Pornographic content and collect their pictures.

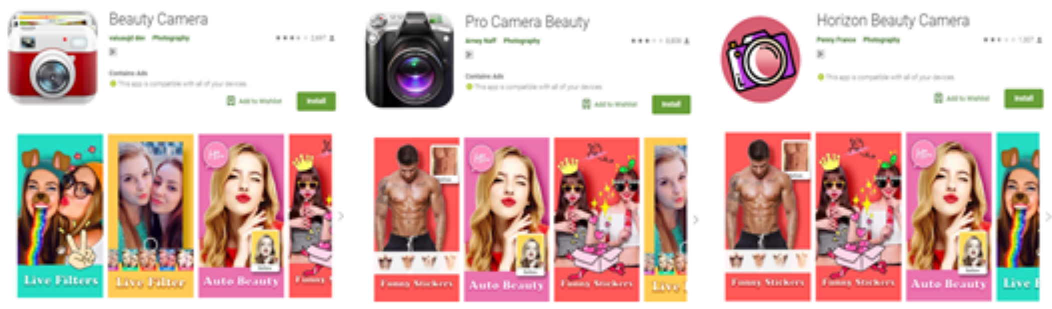
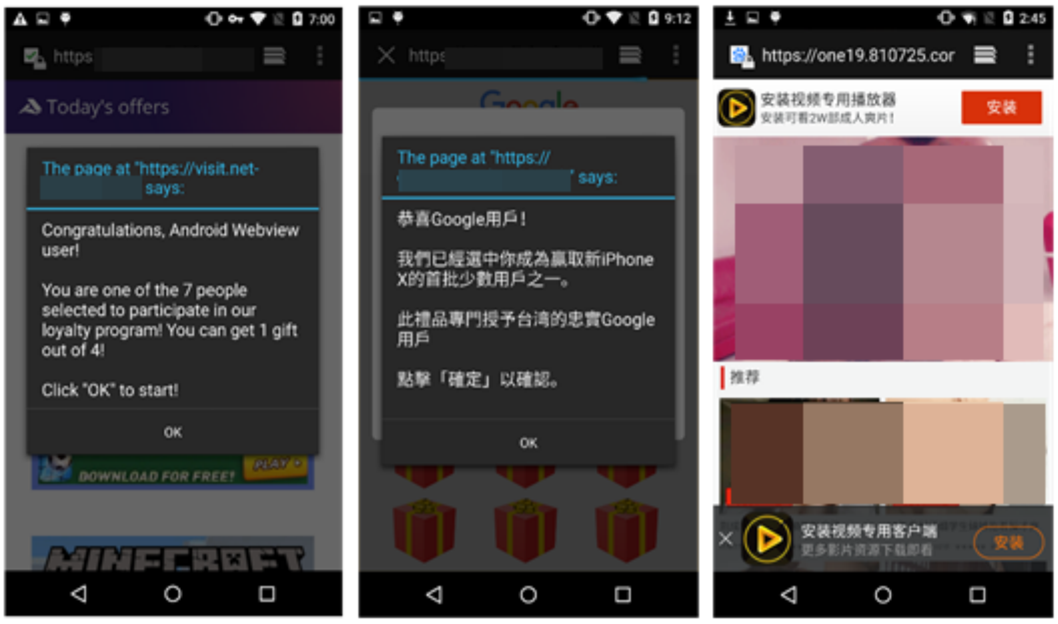


Figure 26 - The malicious camera applications (Source)

TrendMicro has discovered that in several Android beauty camera apps (some of which have been downloaded millions of times) there is an option built in to access remote ad configuration servers which means the application can change behaviour on the fly.

The apps push several full screen advertisements to the victim when they unlock their devices, including malicious ads such as pornography and fraudulent content. Some of these apps also redirect to phishing websites that ask the user for personal information such as phone numbers or addresses.



https [redacted]

Today's offers

The page at "https://visit.net-[redacted] says:

Congratulations, Android Webview user!

You are one of the 7 people selected to participate in our loyalty program! You can get 1 gift out of 4!

Click "OK" to start!

OK

https [redacted]

The page at "https://[redacted]" says:

恭喜Google用戶！

我們已經選中你成為贏取新iPhone X的首批少數用戶之一。

此禮品專門授予台湾的忠實Google用戶

點擊「確定」以確認。

OK

https://one19.810725.cor

安装视频专用播放器
安装可看2W部成人爽片！
安装

推荐

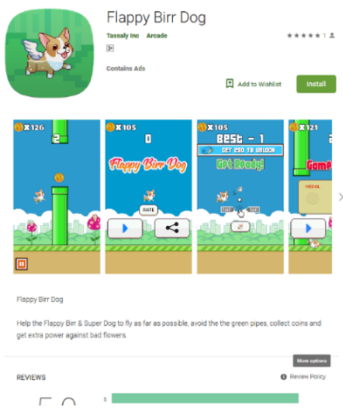
安装视频专用客户端
更多影片资源下载即看
安装

Figure 27 - Example of the pop ups mentioned (Source)

For more information please see TrendMicro's blogpost:

https://blog.trendmicro.com/trendlabs-security-intelligence/various-google-play-beauty-camera-apps-sends-users-pornographic-content-redirects-them-to-phishing-websites-and-collects-their-pictures/

Spyware disguises as Android applications on Google Play.



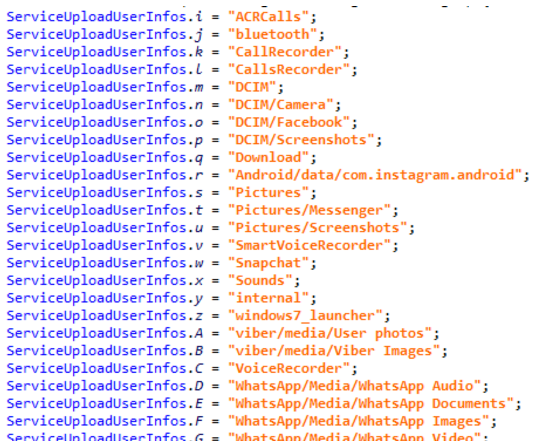
Help the Flappy Birr & Super Dog to fly as far as possible, avoid the the green pipes, collect coins and get extra power against bad flowers.

Figure 28 - One of the infected applications (Source)

TrendMicro has discovered spyware (ANDROIDOS_MOBSTSPY) which was embedded in some games that were on the Google play store in 2018. Some of these games have been downloaded over 100.000 times from users all over the world. Google has already removed the malicious games from the Play Store.

The apps that were infected are Flappy Birr Dog, FlashLight, HZPermis Pro Arabe, Win7imulator, Win7Launcher and Flappy Bird

The Spyware was capable of stealing information such as the user's location, SMS conversations, call logs and clipboard items. When the application is launched, the malware checks if the device has internet access, which will then parse an XML configuration file from its command and control server.



```
ServiceUploadUserInfos.i = "ACRCalls";
ServiceUploadUserInfos.j = "bluetooth";
ServiceUploadUserInfos.k = "CallRecorder";
ServiceUploadUserInfos.l = "CallsRecorder";
ServiceUploadUserInfos.m = "DCIM";
ServiceUploadUserInfos.n = "DCIM/Camera";
ServiceUploadUserInfos.o = "DCIM/Facebook";
ServiceUploadUserInfos.p = "DCIM/Screenshots";
ServiceUploadUserInfos.q = "Download";
ServiceUploadUserInfos.r = "Android/data/com.instagram.android";
ServiceUploadUserInfos.s = "Pictures";
ServiceUploadUserInfos.t = "Pictures/Messenger";
ServiceUploadUserInfos.u = "Pictures/Screenshots";
ServiceUploadUserInfos.v = "SmartVoiceRecorder";
ServiceUploadUserInfos.w = "Snapchat";
ServiceUploadUserInfos.x = "Sounds";
ServiceUploadUserInfos.y = "internal";
ServiceUploadUserInfos.z = "windows7_launcher";
ServiceUploadUserInfos.A = "viber/media/User photos";
ServiceUploadUserInfos.B = "viber/media/Viber Images";
ServiceUploadUserInfos.C = "VoiceRecorder";
ServiceUploadUserInfos.D = "WhatsApp/Media/WhatsApp Audio";
ServiceUploadUserInfos.E = "WhatsApp/Media/WhatsApp Documents";
ServiceUploadUserInfos.F = "WhatsApp/Media/WhatsApp Images";
ServiceUploadUserInfos.G = "WhatsApp/Media/WhatsApp Video";
```

Figure 29 - File Upload options of the malware (Source)

The malware will then collect device information such as language, country and device manufacturer, after which the device is registered on the Server and waits for instructions. The C&C can then send many different commands, such as collecting call logs, monitoring SMS conversations, or collecting other forms of sensitive information, which can then be uploaded to the C&C.

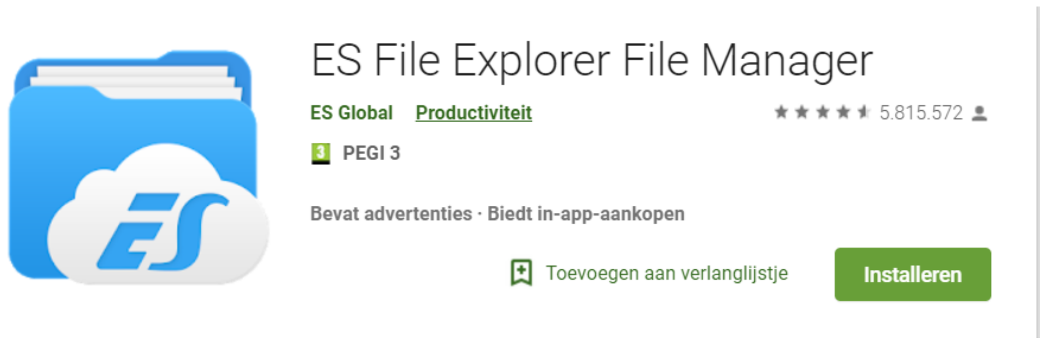
The malware is also capable of displaying phishing screens for Facebook and Google to attempt to trick the user into giving up their credentials

For more information please see the TrendMicro blogpost:

https://blog.trendmicro.com/trendlabs-security-intelligence/spyware-disguises-as-Android-applications-on-google-play/

Flaw found in the popular application ES File Explorer for Android

A security researcher nicknamed “fs0c131y” has found a flaw in the app called ES File Explorer, which has been downloaded over 500 million times.



ES File Explorer File Manager

ES Global Productiviteit ★★★★★ 5.815.572

PEGI 3

Bevat advertenties · Biedt in-app-aankopen

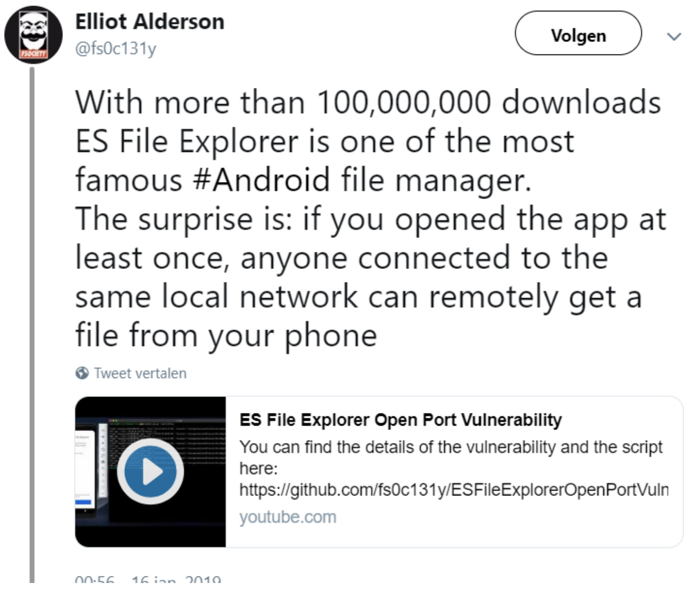
Figure 30 - The vulnerable application (Source)

Every time a user launches the application, it starts a local HTTP server on port 59777, which is publicly accessible. Through this endpoint, an attacker can send a JSON payload to the target cell phone, which will trigger remote code execution in the context of the application. As the application is a File Manager, this means the attacker would have access to any file that is world-readable on the device.

```
curl --header "Content-Type: application/json" --request POST --data
'{"command":"[my_awesome_cmd]"}' http://192.168.0.8:59777
```

example of JSON payload as part of the proof of concept on github.

The vulnerability has been patched by the developer after it caught the developer’s attention via a twitter post:



Elliot Alderson
@fs0c131y

With more than 100,000,000 downloads ES File Explorer is one of the most famous #Android file manager.
The surprise is: if you opened the app at least once, anyone connected to the same local network can remotely get a file from your phone

ES File Explorer Open Port Vulnerability
You can find the details of the vulnerability and the script here: https://github.com/fs0c131y/ESFileExplorerOpenPortVuln
youtube.com

00:56 - 16 jan. 2019

Figure 31 - The twitter post disclosing the vulnerability (Source)

The full exploit can be found on the following website:

https://github.com/fs0c131y/ESFileExplorerOpenPortVuln

Three years overdue: Google patches Chrome for Android to fix vulnerability.

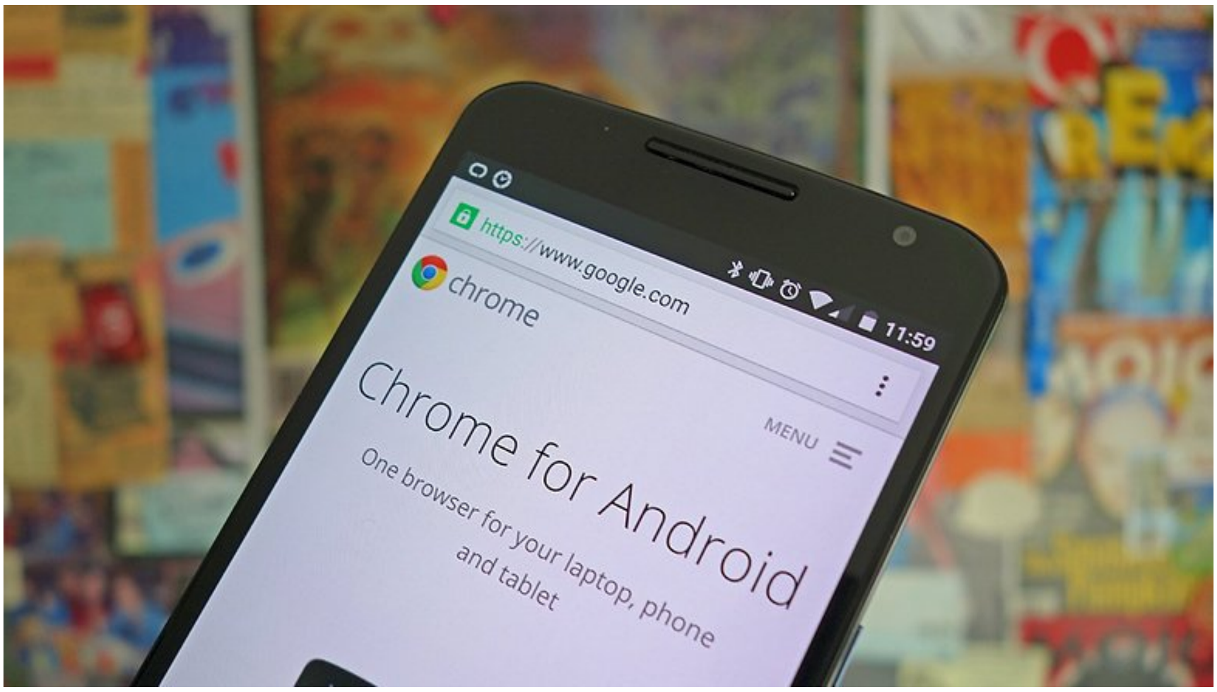
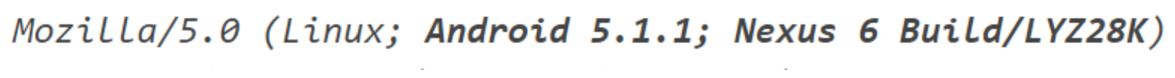


Figure 32 - Chrome for Android (Source)

Google has finally gotten around to patching a three-year-old vulnerability in its Chrome for Android browser. The vulnerability was an information disclosure vulnerability which revealed the phone model and build.

The security researchers at Night watch Cybersecurity identified the vulnerability back in May 2015 but Google did not address the threat.

The information disclosed is in the user-agent header, which is used whenever a user visits a website



```
Mozilla/5.0 (Linux; Android 5.1.1; Nexus 6 Build/LYZ28K)
```

Figure 33 - User-agent header of Nexus 6 (Source)

The fact that the user-agent header identifies the operating system and its version is not unique. This follows generally what many other browsers have been doing on desktop and mobile. The problem is in the build tag. A build tag can identify both the device name and its firmware build. For many devices, this can be used to gather information not only about the device itself, but also the carrier on which it is running and from what country the user is, which then could be used to identify the device security patch level, providing insight into which exploits could potentially be used against the targeted device.

Source:

https://wwws.nightwatchcybersecurity.com/2018/12/25/chrome-browser-for-Android-reveals-hardware-information/

Hackers could potentially abuse Siri shortcuts, introduced in iOS 12

Siri shortcuts, a new function introduced by Apple in iOS 12, could be abused by hackers, according to security researchers at IBM.

The shortcuts are meant to provide faster access to applications and features, automating common tasks, and can be enabled by third party developers. Because of the apps' capability to perform complex tasks, it also introduced potential security risks according to John Kuhn, senior threat researcher at IBM.

Using out of the box shortcut functionality, code can be executed to have Siri tell the victim that their information has been collected and that a ransom needs to be paid. Furthermore, an attacker could automate data collection from the device, such as stored videos, pictures, IP address and potentially even geolocation.

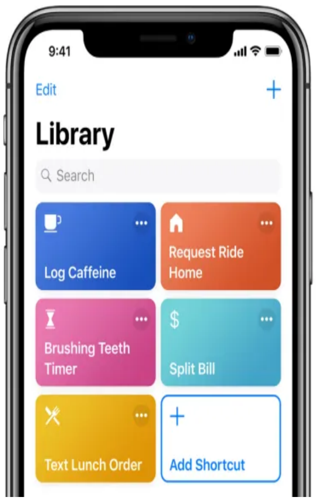


Figure 34 - Siri shortcuts (Source)

One more harmful attack would be to use the shortcut app as a spambot, automating a message to be sent to the victim's entire contact list, containing a download with the same malicious shortcut, causing a spread of the malware.

The entire blogpost can be found on the following URL:

https://securityintelligence.com/hey-siri-get-my-coffee-hold-the-malware/

Facebook and Google use Certificate loophole on iOS to launch data gathering apps.

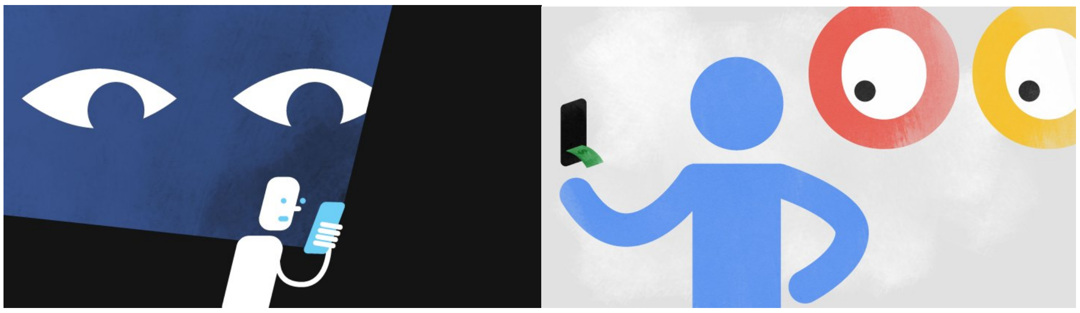


Figure 35 – Facebook and Google are watching (Source)

Apple's Developer Enterprise Program (ADEP) has been abused by Google and Facebook in order to install invasive data gathering applications outside of the respective organisations.

The program allows developers to distribute applications used in the corporation internally without having to pass the App Store review process, which means the application can be launched with more permissions than would be allowed by other applications downloaded from the App Store.

TechCrunch revealed that users starting from just 13 years old got compensated via gift cards to install a so-called research application from Facebook, which could collect messages as well as continuously track the location of users.

Facebook has shut down the iOS version of the application since the reveal, but that did not stop Apple from revoking all of the enterprise certificates of both Facebook and Google, which also stopped the deployment of legitimate internal apps, as they use the same certificates.

Facebook is using the same tactics on Android, but Google has not responded to the question of whether the application breaks any of Googles policies.

In a recent article, also by TechCrunch, it was reported that Google has shut down its own research app called “Screenwise Meter” which used the same loophole as the Facebook app.

The TechCrunch posts can be found here:

https://techcrunch.com/2019/01/30/googles-also-peddling-a-data-collector-through-apples-back-door/

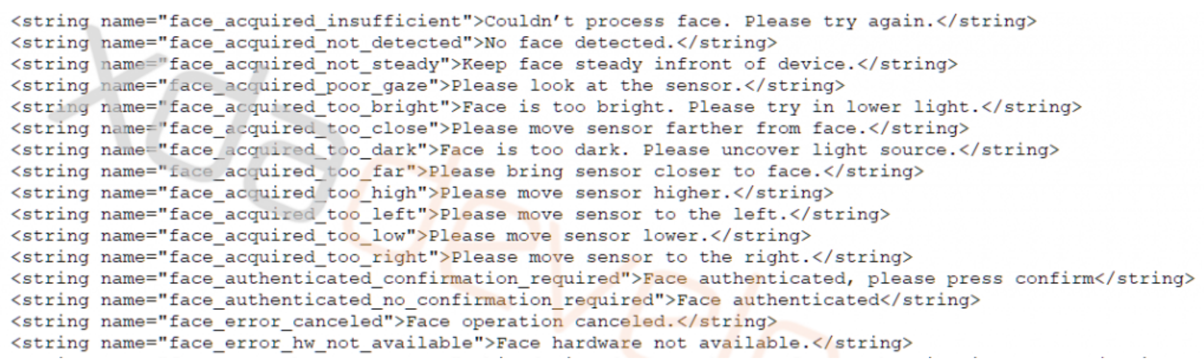
https://techcrunch.com/2019/01/29/facebook-project-atlas/?guccounter=1

Google developing native biometric authentication in Android Q

Figure 36 - Android Q Biometrics(Source)

According to the XDA-Developers portal, Google is developing native support for a secure biometric authentication feature, closely resembling Apple’s face ID.

The feature is rumoured to be released in Android Q. Mishaal Rahmad points to “dozens of strings and multiple methods, classes and fields related to facial recognition in the framework, System UI and Settings APKs" of a leaked Android Q build



```
<string name="face_acquired_insufficient">Couldn’t process face. Please try again.</string>
<string name="face_acquired_not_detected">No face detected.</string>
<string name="face_acquired_not_steady">Keep face steady infront of device.</string>
<string name="face_acquired_poor_gaze">Please look at the sensor.</string>
<string name="face_acquired_too_bright">Face is too bright. Please try in lower light.</string>
<string name="face_acquired_too_close">Please move sensor farther from face.</string>
<string name="face_acquired_too_dark">Face is too dark. Please uncover light source.</string>
<string name="face_acquired_too_far">Please bring sensor closer to face.</string>
<string name="face_acquired_too_high">Please move sensor higher.</string>
<string name="face_acquired_too_left">Please move sensor to the left.</string>
<string name="face_acquired_too_low">Please move sensor lower.</string>
<string name="face_acquired_too_right">Please move sensor to the right.</string>
<string name="face_authenticated_confirmation_required">Face authenticated, please press confirm</string>
<string name="face_authenticated_no_confirmation_required">Face authenticated</string>
<string name="face_error_canceled">Face operation canceled.</string>
<string name="face_error_hw_not_available">Face hardware not available.</string>
```

Figure 37 - Strings found in the leaked build of Q (Source)

The leaked build also includes a lot of other features, such as a system-wide dark theme, which will work with apps that don’t come with a native dark mode.

Source :

https://www.xda-developers.com/Android-q-face-id-rumor/

5.2 General security news

Top malware threats for December 2018

Check Point published their “Global Threat Index” for December 2018.The index investigates the most common malware variants and trends of cyber criminals.

Researchers saw “Smoke Loader” rise to the top 10 after a boost in activity mainly in Japan and Ukraine, Smoke Loader is commonly used as an enabler for other malware such as Panda Banker, AZORult Info stealer, …

Despite cryptocurrencies becoming less valuable in 2018, crypto mining makes up half of the top 10 list and fills the top 4 positions with Coinhive remaining the most prevalent malware for over one year now, impacting 12% of organizations worldwide.

The top 10 list is as follows:

1. Coinhive
2. XMRig
3. Jsecion
4. Cryptoloot
5. Emotet
6. Nivdort
7. Dorkbot
8. Ramnit
9. Smokeloader
10. Authedmine

The full list can be found here:

https://blog.checkpoint.com/2019/01/14/december-2018-most-wanted-malware-smokeloader-crypto-malware-ransomware/

38% of smartphones using facial recognition are unlockable using a photo.

The Dutch consumer confederation “Consumentenbond” has tested 110 smartphones using facial recognition software to unlock the phone. 42 of them could be unlocked using a photo of the user’s face.

Samsung, Huawei, Sony these are just a few of the brands that were vulnerable to this exploit, it’s worth noting that not all the devices produced by these manufacturers can be exploited by this flaw, it usually involves the low/mid-grade smartphones.

The full results can be found on the following website:

https://www.consumentenbond.nl/veilig-internetten/gezichtsherkenning-te-hacken#no1

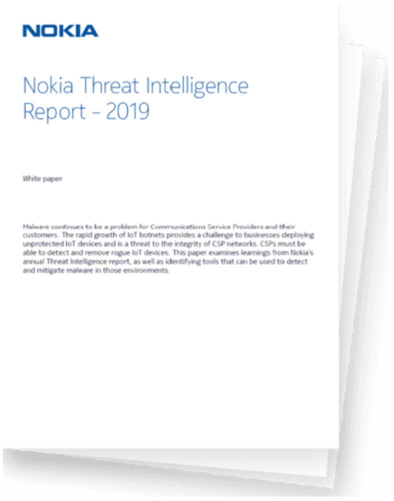
Figure 38- Facial recognition in a picture (Source)

Android 50 times more likely to be infected with malware compared to iOS

Nokia's latest threat intelligence report has revealed that Android devices are 50 times more likely to be infected with malware than Apple devices. According to the whitepaper, Android is responsible for about 47% of all the observed malware infections, Windows pc's make up about 36% and IoT 16%. Apple's iPhone wins the race with a share of less than 1%.

The full report can be found here:

https://pages.nokia.com/T003B6-Threat-Intelligence-Report-2019.html



NOKIA

Nokia Threat Intelligence Report – 2019

White paper

Malware continues to be a problem for Communications Service Providers and their customers. The rapid growth of IoT botnets provides a challenge to businesses deploying unprotected IoT devices and is a threat to the integrity of CSP networks. CSPs must be able to detect and remove rogue IoT devices. This paper examines learnings from Nokia's annual Threat Intelligence report, as well as identifying tools that can be used to detect and mitigate malware in those environments.

Figure 39 - Nokia's Threat Report (Source)

Malware dubbed Redaman uses debt to prey on banking victims in Russia

Russian speakers beware, since a new malware campaign is targeting you by using threats of debt and missing payments.

The round of attacks, researched by the Palo Alto security team, was tracked over the last four months of 2018.

The attack vector is large and involves mass distribution of spam and phishing mails using titles that could cause panic such as "Debt due Wednesday". The message keeps changing but they all have a common theme; they want to trick the recipient into opening the attached archive.

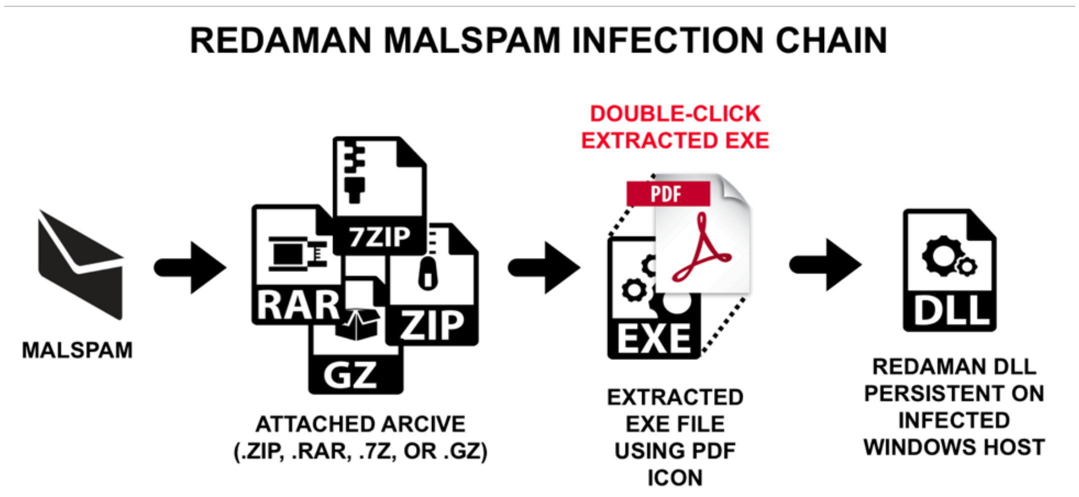
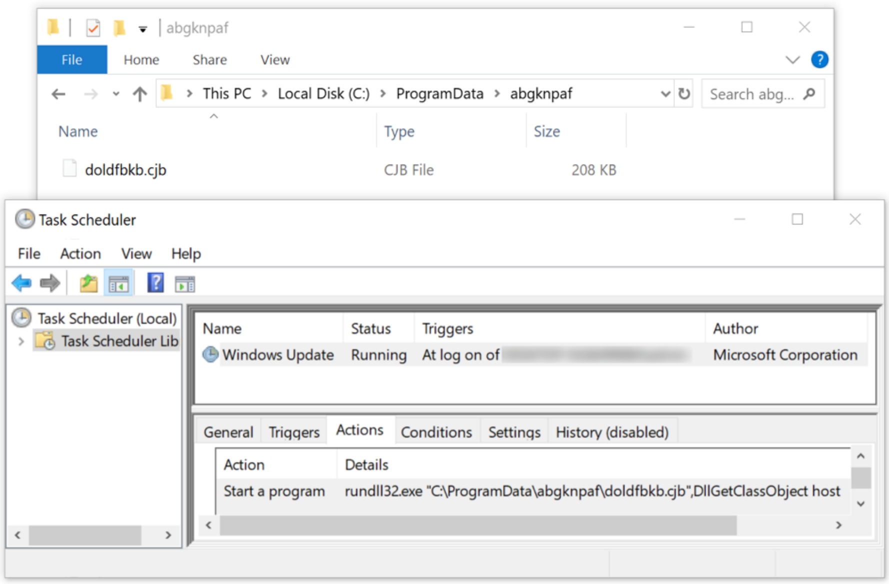

REDAMAN MALSPAM INFECTION CHAIN
MALSPAM
7ZIP
RAR
ZIP
GZ
ATTACHED ARCIVE (.ZIP, .RAR, .7Z, OR .GZ)
DOUBLE-CLICK EXTRACTED EXE
PDF
EXE
EXTRACTED EXE FILE USING PDF ICON
DLL
REDAMAN DLL PERSISTENT ON INFECTED WINDOWS HOST

Figure 40 - Flow chart for infections from Redaman banking malware from September through December of 2018. (Source)

Upon execution a Trojan will launch a scan to check if the program is running in a sandbox or not. If the malware uncovers files or directories that suggest virtualization or sandboxing, the executable exits. If the machine appears to be legitimate, the windows executable will drop a DLL file in the PC's temp folder and create a randomly named folder in the ProgramData directory, it will then shift the DLL to this folder and use a random file name. The DLL will create a windows task which triggers every time the user logs into the machine to maintain persistence.



Name	Type	Size
doldfbkb.cjb	CJB File	208 KB

Name	Status	Triggers	Author
Windows Update	Running	At log on of [redacted]	Microsoft Corporation

Action	Details
Start a program	rundll32.exe "C:\ProgramData\abgknpaf\doldfbkb.cjb",DllGetClassObject host

Figure 41 - The Windows task (Source)

Redaman uses a hooking system to monitor browsing activity. The goal is to steal banking credentials and other data which could compromise the victim's funds or enable the attacker to commit identity theft.
Source:
https://unit42.paloaltonetworks.com/russian-language-malspam-pushing-redaman-banking-malware/

6 Mobile platform statistics

6.1 OS market shares

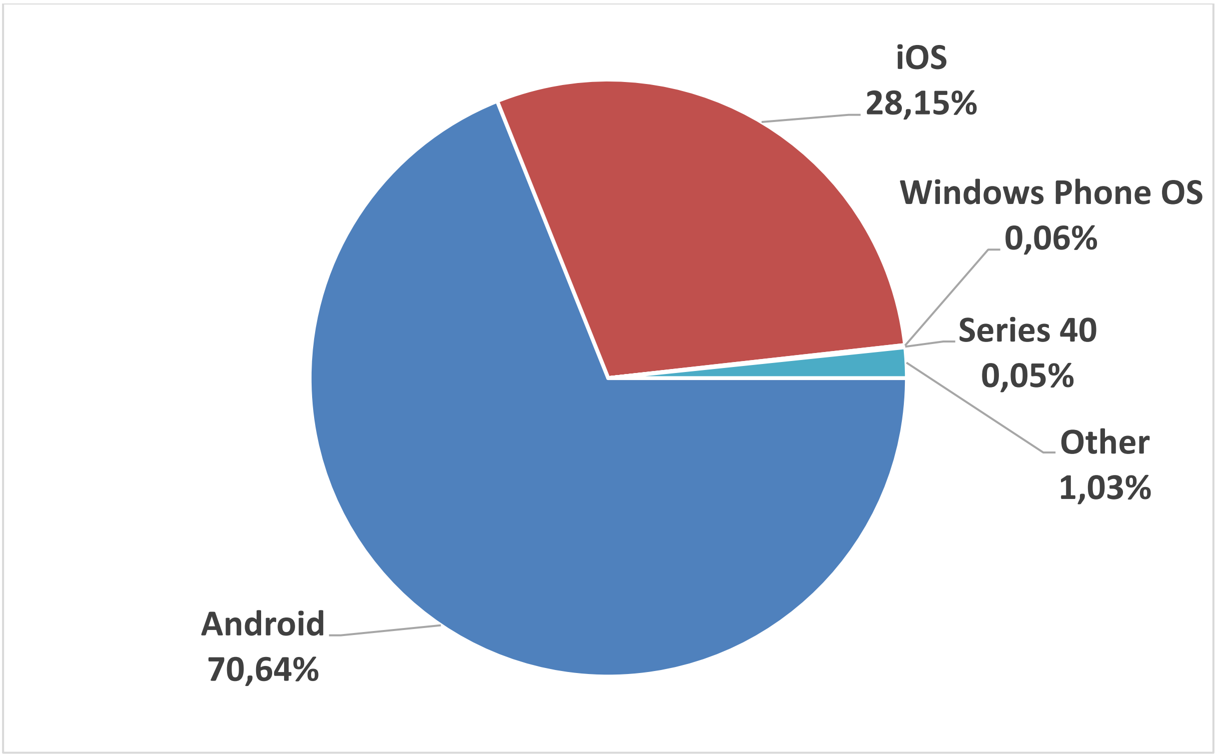
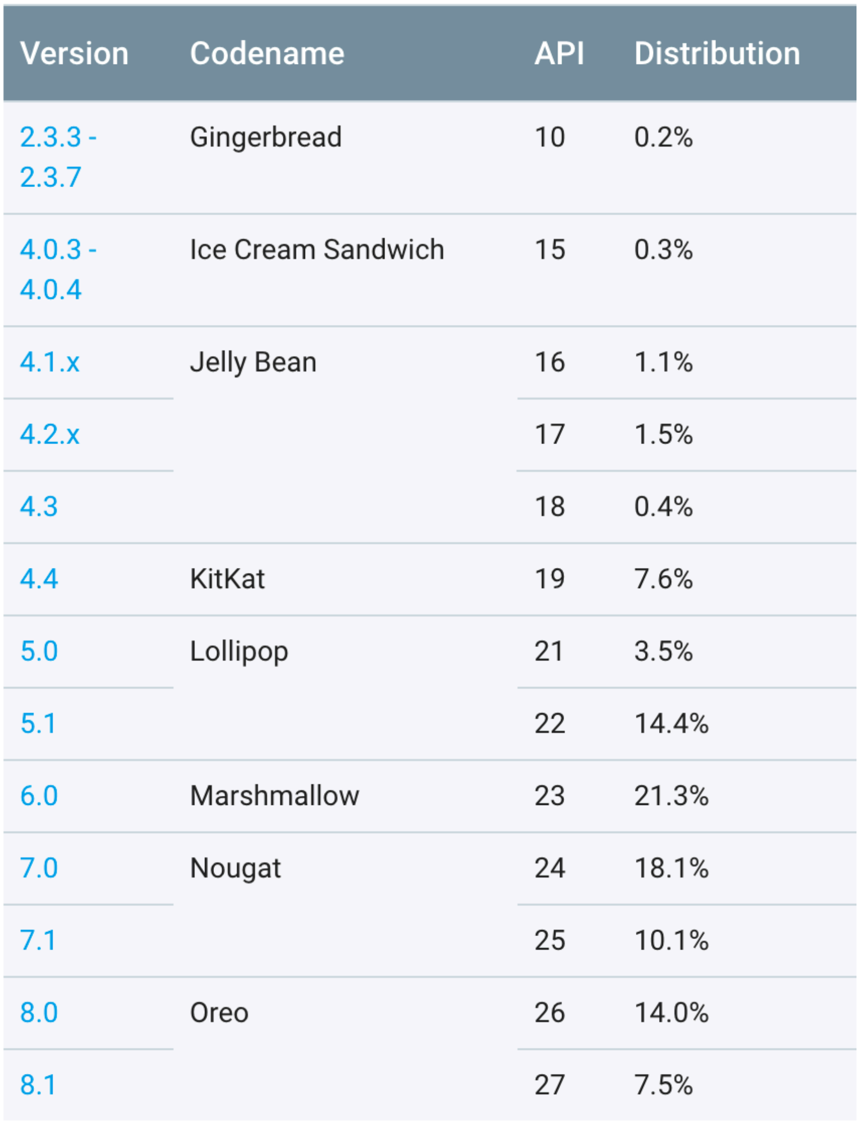


Figure 42: Mobile OS market shares for January 2019. (Source)

Compared to last month's OS market share numbers, we can see a small move towards Android adoption, away from iOS. The Android adoption rates have increased with 1,71 percentage points, while the iOS adoption rates have decreased by 1,14 percentage points. The difference between the two previous differences can be seen in the "Other" operating systems, which has slightly increased compared to last month. Among the "Other" operating systems are Linux, Bada and Symbian. The shares of the Windows phone OS and Series 40 OS systems remain mostly unchanged this month.

6.2 Android



Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.2%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.3%
4.1.x	Jelly Bean	16	1.1%
4.2.x		17	1.5%
4.3		18	0.4%
4.4	KitKat	19	7.6%
5.0	Lollipop	21	3.5%
5.1		22	14.4%
6.0	Marshmallow	23	21.3%
7.0	Nougat	24	18.1%
7.1		25	10.1%
8.0	Oreo	26	14.0%
8.1		27	7.5%

Figure 43: Android OS fragmentation of October 2018. (Source)

No new information on Android OS fragmentation has been published by Google since the three previous newsletters. The data displayed above was collected during a 7-day period ending on October 26, 2018.

User adoption rates of the older Android versions remain fairly stable from September to October. The number of devices running Android versions lower than 5.0 all changed no more than 0.2%. It is worth noting that all changes of these versions are decreases of user adoption rates.

Looking at the more recent versions of Android, starting at version 5.1, we see that users continue to migrate to the newest versions. Adoption rates of Android Oreo increased with a total of 2,3% while the adoption rates of Android Marshmallow, Nougat and older dropped with the same amount.

6.3 iOS

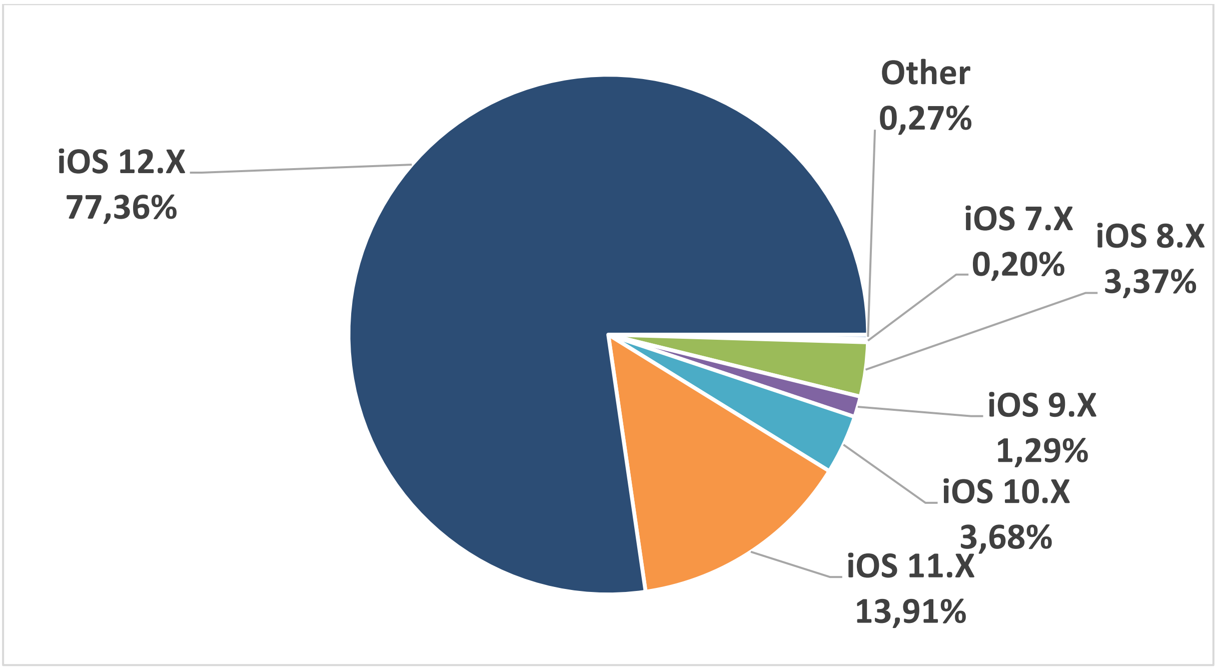


Figure 44: iOS fragmentation in January of 2019. (Source)

Similar to the previous three months, iOS users continue to shift towards the newest iOS 12 versions. Since the stable release of iOS 12 in mid-September, almost three quarters of all iOS users have installed it. iOS 12.x has seen an increase this month of 4,36%

Many of the users currently operating on iOS 12 have switched over from iOS 11, as can be noticed in the iOS 11 adoption rates. iOS 11 is down from 17,37% to 13,91% this month

The number of users operating on the older iOS versions have remained fairly stable compared to last month.

(End of document)