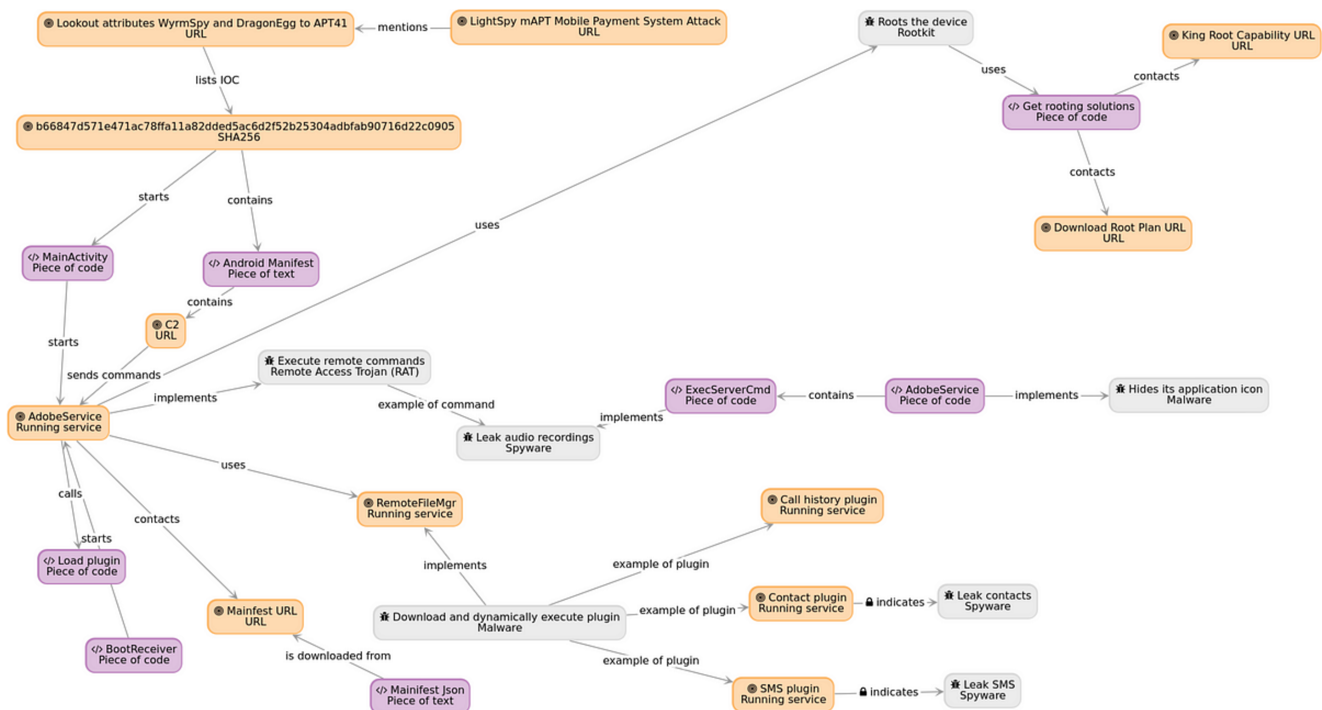
Organizing malware analysis with Colander: example on Android/WyrmSpy

@cryptax December 19, 2023



@cryptax

--

When I analyze a malware, I keep side by side an ugly text file where I write down my reversing notes. Unfortunately, my notes are usually totally cryptic — even by me — a few weeks later.
Update December 19, 2023: searching the cases is already possible, my error: I had missed the functionality (see Conclusion).

This is taken from my old notes on the analysis of a sample of Android/GodFather
To tackle this problem, I have been using Colander now and then for a few months. Colander is an open-source “incident response and knowledge management platform”. It is multi-purpose, i.e not specific to Android malware analysis, but I find it quite easy to use to organize my reverse engineering notes.

As an example, we'll analyze a sample of Android/WyrmSpy sha256 `b66847d571e471ac78ffa11a82dded5ac6d2f52b25304adbfab90716d22c0905`.

> What does this article cover? (1) Use of Colander, (2) Analysis of a sample of WyrmSpy. Enjoy!

Colander

Colander uses Incident Response terminology with artifacts, devices, threats, observables, events and data. Basically, you create items in these "categories" and then you can create *relations* between them, and Colander automatically generates a graph to display it all.

LookOut's report contains lists of WyrmSpy IOCs. So, I created a URL item for the report, a SHA256 observable for the sample I want to look into, and assigned the relation "lists IOCs" between both entities. Actually, it would probably be wiser to upload the sample as an artifact, but I was lazy and didn't want to upload any file.
The main activity is referenced in the Android Manifest, and fortunately, the sample is not packed, so we can decompile the activity. I always like to keep track of where the main entry point is, so I create a "fragment of data" below for the main activity.

Fragment of data for the Main activity of the malware.
The malware starts a `FakeActivity`, which starts a so-called `AdobeService`, which processes malicious commands. For instance, the following code will upload raw audio recording to the remote C2.

If at some point the malware processed a message "Calls" or "audio", the malware retrieves the audio file (if necessary converts from RAW format to MP3) and uploads the file to the remote C2.
In a few weeks, I won't remember where this piece of code is, so in Colander, I create (1) a threat "Leaks audio recordings", (2) a piece of code — in that case it is in `ExecServerCmd` of `com.flash18.AdobeService` — and (3) creates a relation "implements" between the first 2.

Creating a threat to memorize the malware does this + easily find again where this happens in the code
The initialization of the malware is done in a method called `InitAppConfig` of `com.flash18.Config`. 👀 It's actually the first time I see that: **the C2's URL is kept in a meta-data item of the Android Manifest** ! Not very stealthy 👓, but probably easy to customize for APT 41.

```
public static Boolean InitAppConfig(Context context) {
    try {
         Bundle meta =
context.getPackageManager().getApplicationInfo("com.sec.android.provide.badge",
0x80).metaData;
         Config._server_url = meta.getString("u");
         Config._password = meta.getString("p");
         Config._version = meta.getString("v");
         if(Config._interval <= 0) {
             Config._interval = 30;
         }

         Config._bind = meta.getString("kb");         Config._CustomId =
meta.getString("CustomId");         Config._uid = Utils.GetAndroidId(context);...
```

```
<meta-data android:name="p" android:value="password"/><meta-data android:name="v"
android:value="5.0707.1"/><meta-data android:name="u"
android:value="hXXps://116.205.4.18:33889/control/"/><meta-data android:name="uh"
android:value="LwMzVmkWVoyIe2oQ/F38OUa/lQjdPjaCpS/Fp8tC7VZbfJml8JdeuIMcmYPSW5SpWreW1yQ
<meta-data android:name="kb" android:value="no"/><meta-data android:name="CustomId"
android:value="mx000002"/>
```

Rooting status

WyrmSpy is known to test for and install **rooting** applications, which isn't altogether that common in malware. So, I want to look into that.

The `AdobeService` call `uploadRootInfo`, which sends the rooting status of the device: it says if the device can be rooted with **King Root**, and if there is yet another rooting solution on the device. To check if King Root can be used, the malware probably uses a non-identified (yet) third party SDK which contacts `pmir.3g.qq.com` and expects rooting solutions in return.

Later in the code, the malware roots the device if it isn't rooted yet. As expected, it will use King Root if it can. Otherwise, it tries "Ivy Root" which is also referred to in the code as `tryOurSolutions`. The later root zip is downloaded from the C2, unzipped and executed. Default download URLs are found in methods named `DownRootPlan` and `DownRootPlan2`.

Colander's graph

After a while, my brains are exploding and it is time to use Colander's graph capability. This is what the graph looks like at first.

Colander initial graph for Android/WyrmSpy shows relationships between entities. It's pretty good already but we see there are a few unrelated groups (in red)
I forgot several relations, which is why some nodes appear as isolated in the analysis. Let's fix this. For example: *who contacts the URL to get the* `mainfest`/`mainifest` (both typos are present in the malware)? It's the `AdobeService`. We can directly add the relationship from the

graph view.

Improving the graph: Adding relations of “AdobeService” to point to “Mainfest URL”
Relations are fully editable from the graph. For example, I want to correct the fact that the sample does not directly start the AdobeService. Rather, the sample begins with the MainActivity, which starts a FakeActivity (not represented), which starts the AdobeActivity. To do so, I delete the relation and create the correct ones.

Final version of Android/WyrmSpy analysis

After a little edition, this is the final graph I get to. Honestly, I didn’t have much to do and only moved a few nodes to reach this state.

Final graph reprensenting my analysis of WyrmSpy.
I can guarantee this is way easier to understand than my ugly notes.

The graph is just an entry point to get more details on each node. For instance, I clicked on RemoteFileMgr and can read / update what this class does in the malware.

Conclusion

- 🎅. I see it more as a very helpful tool to your analysis.
- The 😍. I played with several editors in the past and my graphs usually ended up like a plate of spaghettis. It’s not the case with Colander where I find the resulting graph readable (of course, it depends the level of detail you include)
- There is very: of my sample, that’s all and did not have to waste time “understanding how to make Colander do it”. To be honest, sometimes, I probably did not use it in an optimal manner: I’m a bit lost in the list of possible observables, threats and artifacts. Fortunately, it didn’t matter too much and I managed to get Colander do what was helpful to . . That’s versatility.
- My advice is to of relationships . Actually, it’s the same for any node label. If you want to say something long, put it in the description, not in the title.
- Next feature I’d love is for cases to be and , so that in 8 months I can quickly search through my cases to find again that sample which was doing XYZ.
- Android/ keeps the URL of its C2 in the `AndroidManifest.xml` file!
- Android/WyrmSpy implements several downloadable “plugins” which implement malicious features like reading and uploading all contacts.
- Android/WyrmSpy wants to operate on a rooted device. It will try and root the device several different ways.

— Cryptax