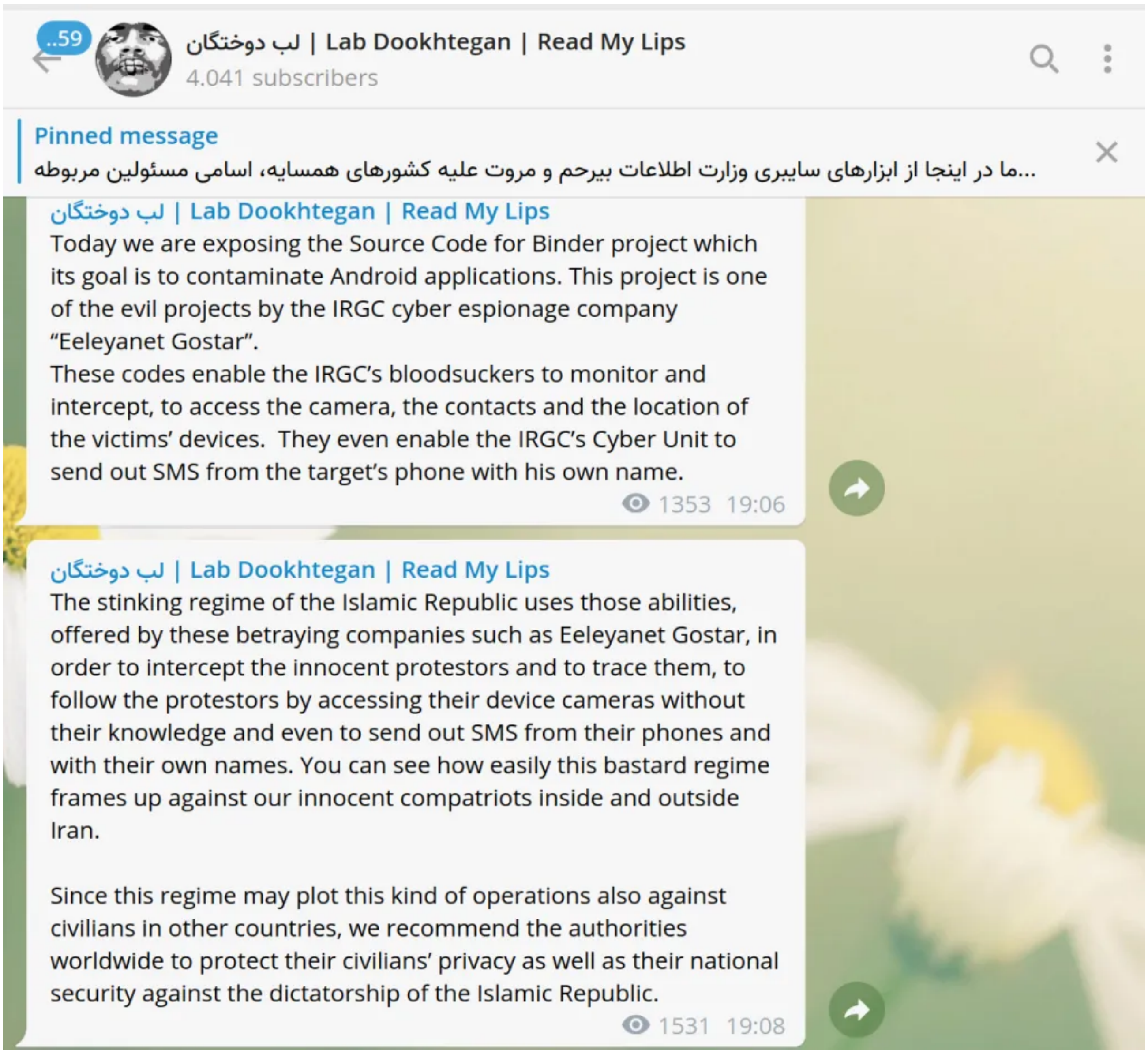
MuddyWater: Binder Project (Part 1)

marcoramilli.com/2021/05/01/muddywater-binder-project-part-1/

View all posts by marcoramilli May 1, 2021

According to **Lab Dookhtegan**, which you migth remeber him/their from HERE, HERE and HERE, `Binder` is a project related to IRGC cyber espionage group build for trojenize google apps (APK). The application "trojenization" is a well-known process which takes as input a good APK and a code to inject (a RAT, for example). The system is able to unbuild the original APK and to inject the RAT into the "good application". The result is a Trojan which could compromize an unaware target. Indeed unaware users by opening up the trojenized APK will run the desired application as wel as the RAT in a background process by starting up the infection chain. MuddyWater is notoriusly alledged linked to IRGC (HERE) as main contractors so what this blog post. According with ClearSky (Report HERE) Iranian cyber espionage forces are increasing their mobile abilities by meaning they are investing in Mobile (RAT andTrojan) development on either iOS and Android operative systems. All these information are rolling on the same direction and they are building up a concrete base to start to analyze what **Lab Dookhtegan** released over the past weeks.


..59

لب دوختگان | Lab Dookhtegan | Read My Lips
4.041 subscribers

Pinned message
...ما در اینجا از ابزارهای سایبری وزارت اطلاعات بیرحم و مروت علیه کشورهای همسایه، اسامی مسئولین مربوطه

لب دوختگان | Lab Dookhtegan | Read My Lips
Today we are exposing the Source Code for Binder project which its goal is to contaminate Android applications. This project is one of the evil projects by the IRGC cyber espionage company "Eeleyanet Gostar".
These codes enable the IRGC's bloodsuckers to monitor and intercept, to access the camera, the contacts and the location of the victims' devices. They even enable the IRGC's Cyber Unit to send out SMS from the target's phone with his own name.
1353 19:06

لب دوختگان | Lab Dookhtegan | Read My Lips
The stinking regime of the Islamic Republic uses those abilities, offered by these betraying companies such as Eeleyanet Gostar, in order to intercept the innocent protestors and to trace them, to follow the protestors by accessing their device cameras without their knowledge and even to send out SMS from their phones and with their own names. You can see how easily this bastard regime frames up against our innocent compatriots inside and outside Iran.

Since this regime may plot this kind of operations also against civilians in other countries, we recommend the authorities worldwide to protect their civilians' privacy as well as their national security against the dictatorship of the Islamic Republic.
1531 19:08

Message from Lab Dookhtegan on March 15th 2021

Indeed **Lab Dookhtegan** leaked some source code allegedly belonging to `Binder` project on his/their Telegram channel. Let's check some interesting points that we might deduce from that code without performing a complete source code analysis.

Source Code Highlights

The first file that I'd like to point out is named: `action.aspx.cs` . First of all we can deduce that `Binder` is a web application. We have no idea at this stage if there is a GUI involved or simple API calls, but let's analyze the source that we've got from telegram.

A first observation comes form authenication methods. As you might see from the following snip, before getting inside the main loop – which loops for `tasks` to be executed – the user need to be authenticated. The authentication, comes from the `Authorization` HTTP Headers. After that, we see exactly what we were actually thinking about trojenize applications. In other words taking `apk_path` and `rat_path` and building a resulting `apk` to be inoculated to victims.

```
[...]
if (st.get_apiToken("user", "pass").Equals(Request.Headers.Get("Authorization")))
            {
                LB_Log.Items.Add("connected ... ");
                string res = ch.getTask();
                if (!res.Equals("non"))
                {
                    var tasks = JArray.Parse(res);
                    for (int i = 0; i < tasks.Count; i++)
                    {
                        JObject jObject = JObject.Parse(tasks[i].ToString());
                        JObject aJson = jObject.GetValue("apk_path").ToObject<JObject>();
                        JObject rJson = jObject.GetValue("rat_path").ToObject<JObject>();
                        string id = jObject.GetValue("id").ToObject<string>();
                        string apkDotApk = aJson.GetValue("path").ToObject<string>();
                        string apkName = apkDotApk.Remove(apkDotApk.LastIndexOf("."), 4);
                        string apkId = aJson.GetValue("id").ToObject<string>();
                        string ratDotApk = rJson.GetValue("path").ToObject<string>();
                        string ratName = ratDotApk.Remove(ratDotApk.LastIndexOf("."), 4);
                        string ratId = rJson.GetValue("id").ToObject<string>();
                        int bMethod = jObject.GetValue("bmethod").ToObject<int>();
                        int status = jObject.GetValue("status").ToObject<int>();
                        int getPerm = jObject.GetValue("get_perm").ToObject<int>();
                        string projectPath = AppDomain.CurrentDomain.BaseDirectory + "binder\\";
                        string ratPath = projectPath + "apks\\" + id + "\\" + ratName;
                        string apkPath = projectPath + "apks\\" + id + "\\" + apkName;
[...]
```

Snippet from action.aspx.cs

From the 61th line of code we might understand why `binder` is the project name: `string projectPath = AppDomain.CurrentDomain.BaseDirectory + "binder\";` Finally from line `232` we experience a writing mistake, it is hard to call this mistake a typo, since letter `i` and letter `e` aren't close in english keyboard layouts, so probably we are reading a non english speaker developer.

```
[...]
  {
     Log.Items.Add("Your authentication is depricated ... ");
  }
     flag = true;
[...]
```

Snippet from action.aspx.cs. Not English speaker.

A second interested leaked file is `RedLogClass.cs`. First of all we might appreciate the `namespace` which is confirming the project name `Binder` (following snip for details), but even more interesting we might find how the attacker authenticate the client before processing requests.

```
[...]
 private bool enableSending = true;
        private string url = "http://192.168.20.106/api/add_logs";
        private string token =
"LEcTqrnm6ySmU4NdccUapeJRt9a6GYmrtSKilRNtCQnaWz4IfzxHFmbR7YDdMmtZCZyh55vwdbRWDe1TIFEdqkuNQdfhr7TpzBRA";
        public bool sendLog(string project_name, string category_name, string content, [CallerLineNumber] int
lineNumber = 0, [CallerMemberName] string caller = null)
        {
            if (enableSending)
            {
                string response = string.Empty;
                try
                {
                    var client = new RestClient(url);
                    client.Timeout = -1;
                    var Request = new RestRequest(Method.POST);
                    Request.AddHeader("Accept", "application/json");
                    Request.AddHeader("Authorization", "Bearer " + token);
                    //Request.AddHeader("Content-Type", "application/json; CHARSET=UTF-8");
                    //Request.AddJsonBody();
[...]
```

Snippet from ReadLogClass.cs

Indeed the attacker uses Authorization HTTP Header in the following format in order to authenticate the HTTP request. The authentication is performed by concatenating the word `Bearer` with the `token` (line 38). In this specific case we also have the value of such a token:

`LEcTqrnm6ySmU4NdccUapeJRt9a6GYmrtSKilRNtCQnaWz4IfzxHFmbR7YDdMmtZCZyh55vwdbRWDe1TIFEdqkuNQdfhr7TpzBRA` which represents a valid authenticator token. Another intresting observation comes from the `url` variable. It contains the path `/api/add_logs` . It might be used as network signature to detect such an malicious implant. As a bonus track we know the attacker deployed such a tool into a private LAN: `http://192.168.20.106` . This would be interesting later on, let's keep it in mind.

Let's move to another file the `BinderClass.sln` which highlights the used VisualStudio version `14.0.23107.0` and the minimal visual studio version compatible with: `10.0.40219.1` .

```
Microsoft Visual Studio Solution File, Format Version 12.00
# Visual Studio 14
VisualStudioVersion = 14.0.23107.0
MinimumVisualStudioVersion = 10.0.40219.1
Project("{FAE04EC0-301F-11D3-BF4B-00C04F79EFBC}") = "BinderClass", "BinderClass.csproj", "{648562EB-D95C-4C9E-
A7D5-7EDAE84E27AC}"
EndProject
Global
	GlobalSection(SolutionConfigurationPlatforms) = preSolution
		Debug|Any CPU = Debug|Any CPU
		Release|Any CPU = Release|Any CPU
	EndGlobalSection
	GlobalSection(ProjectConfigurationPlatforms) = postSolution
		{648562EB-D95C-4C9E-A7D5-7EDAE84E27AC}.Debug|Any CPU.ActiveCfg = Debug|Any CPU
		{648562EB-D95C-4C9E-A7D5-7EDAE84E27AC}.Debug|Any CPU.Build.0 = Debug|Any CPU
		{648562EB-D95C-4C9E-A7D5-7EDAE84E27AC}.Release|Any CPU.ActiveCfg = Release|Any CPU
		{648562EB-D95C-4C9E-A7D5-7EDAE84E27AC}.Release|Any CPU.Build.0 = Release|Any CPU
	EndGlobalSection
	GlobalSection(SolutionProperties) = preSolution
		HideSolutionNode = FALSE
	EndGlobalSection
	GlobalSection(ExtensibilityGlobals) = postSolution
		SolutionGuid = {2B405E88-E8FA-4807-A5CF-F8CD4AAB89D6}
	EndGlobalSection
EndGlobal
```

Snipped from BinderClass.sln

We are now reading a quite old version of Microsoft VisualStudio previous to 2017, which according to Microsoft Visual Studio release note it has been released before 2017 (HERE). Now, or we are reading a quite old source code (by menaing this project is up and running from years) or we are reading source code from developer/s who are developing since years without updating their principl development tool. While it could be quite unusual keeping an old Visual Studio version, Microsoft introduced many "cloud based" features into their recent Visual Studio platform, including the ability to recognize patterns and malicious code which might be not interesting if you are developing Malware. So, I am not saying this is what happened but I know developers that uses old Visual Studio versions for developing simple PoC and RedTeam scripts, so I believe both iphothesis would be concrete.

Conclusions

Source code analysis is insanely helpful to map how attackers are evolving. In this quick post I began reading the Binder source code allegedly attributed to MuddyWater in order to better understand capabilities, modus operandi and structures. A second part will get into additional project source code helping communities to better map and classify MuddyWater APT.

Follows the reading on Part2 (HERE)