Leonardo S.p.A. Data Breach Analysis

reaqta.com/2021/01/fujinama-analysis-leonardo-spa

ReaQta Threat Intelligence Team identified the malware used in an exfiltration operation against the defence contractor Leonardo S.p.A. The analysis of the malware, which we dubbed **Fujinama**, highlights its capabilities for data theft and exfiltration while maintaining a reasonably low-profile, despite a lack of sophistication, mostly due to the fact that the malicious vector was manually installed by an insider.

Data Breach

Leonardo S.p.A. (formerly *Finmeccanica*) is the 8th largest **defence contractor**. Partially owned by the Italian government, the company is widely known, among other things, for their *AgustaWestland* Helicopters, major contributions to the *Eurofighter* project, development of naval artillery, armoured vehicles, underwater systems, implementation of space systems, electronic defence and more.

On the 5th of December 2020 the CNAIPIC (*National Computer Crime Center for Critical Infrastructure Protection*), a unit specialized in computer crime, part of the Polizia di Stato (the Italian Police), reported the arrest of 2 individuals in relation to a data theft operation, identified for the first time in January 2017, against Leonardo SpA's infrastructure. The anomalous activity was identified by the company's security unit and quickly reported to the authorities that started an extensive investigation.

Though the company’s initial report identified the leak to be negligible in volume, the CNAIPIC’s investigation found the amount to actually be significant, with **100.000** files exfiltrated for a total of **10Gb** of data from **33** devices in a single location and tracking the final infection to a total of **94** different devices. The attack was considered an *APT* by the Italian Police, carried out by a single person whom manually installed a custom malware on each targeted machine.

Physical attacks are hard to detect, as any local access to the device can help to mitigate on-device detections, this is especially true when the attacker is, like in Leonardo’s case, part of the company’s *security unit*.

A physical attack carried out by a person with high-level access is a **worst-case scenario** for any company or agency but, as we will see later, things might have taken a different turn if the malware involved was actually sophisticated.

Fujinama First Detection

In January 2017, Leonardo’s Cyber Security Unit reported anomalous traffic from a number of endpoints operating in the *Pomigliano D’Arco* (Naples) office, the offending application name *c**ft**mon.exe* was a twist of a well-known Windows component *c**tf**mon.exe*. The application was not recognized as malicious by the security solutions in use, but the network traffic was indeed highly anomalous. As we will see in the analysis, while the attacker was certainly persistent, the sophistication was also lacking, in fact the type of traffic generated led eventually to the identification of the threat.

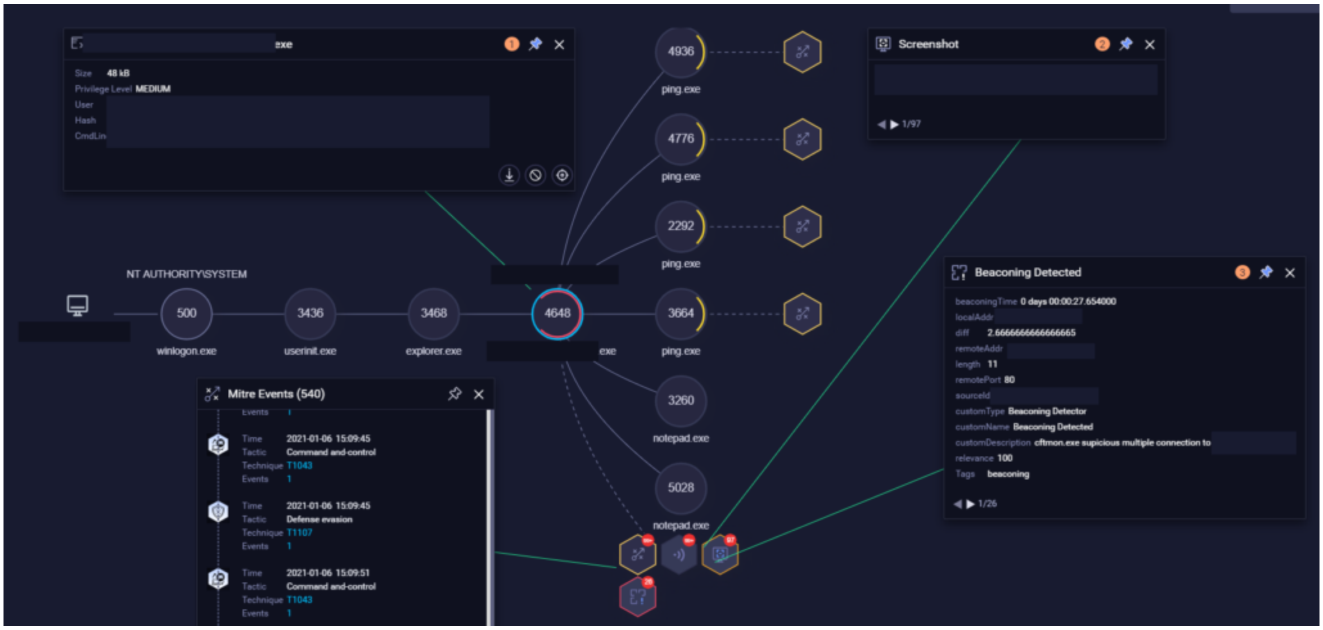
Unfortunately the CNAIPIC didn’t release any information on the threat, except for its filename and the C2 address used: *www[.]fujinama[.]altervista.org* though this was enough to threat hunt in our dataset looking for traces of this malware.

Hunting Down Fujinama

The hunt for *Fujinama* started shortly after CNAIPIC’s bulletin was published. Our *Threat Intelligence team* managed to find samples that reached our sensors network from 2018. From that point, we managed to pivot on a third sample that appears to be related to a different operation.

Two of the three samples share the same keylogging capabilities but they point at two different C2. A third sample, pointing to the *Fujinama* C2, is in all likelihood an evolution of the previous version that includes *screenshots capabilities*, *exfiltration* and *remote execution*. This specific sample, labeled *Sample 2* in the article, will be the focus of our behavioural analysis.

ReaQta-Hive Analysis



Behavioural Tree from a running instance on ReaQta-Hive

Fujinama was written in **Visual Basic 6** and it tries to mimic an internal Windows tool: *cftmon.exe* (as mentioned above, a twist on the legitimate *ctfmon.exe*).

Main Flow

The sample adopts a very simple sandbox evasion technique, sleeping for 60 seconds before activating the malicious flow that consists of:

- **Every 60 seconds**: capturing a **screenshot** of the Desktop and uploading it to the C2
- Installing a **keylogger** on the victim machine that sends all keystroke to the C2

- **Every 5 minutes**: checking on the C2 for the presence of a command used either to **execute** an application or to **exfiltrate** a specific file

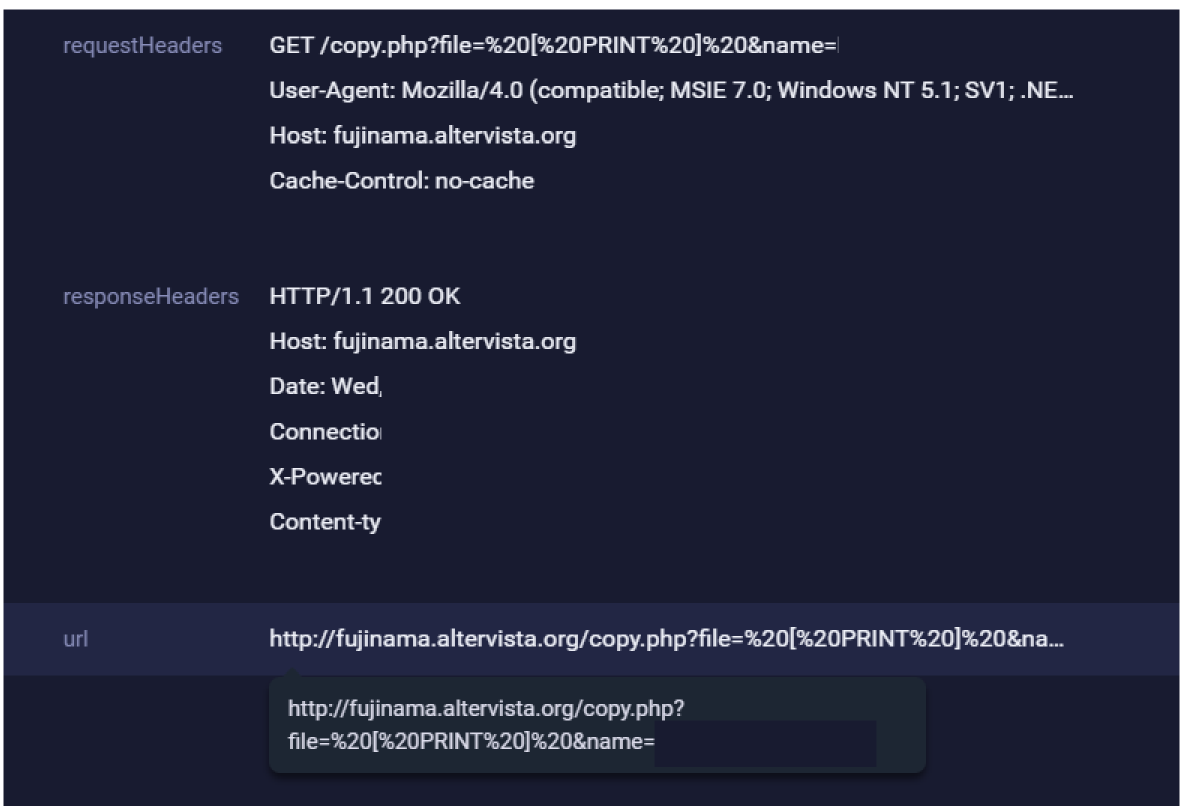
Screenshots

The *Screenshot routine* simulates a keypress on the *PrtScn* button to capture the image of the desktop. The screen content is then saved from the clipboard to a *jpg* file in a temporary folder. Finally Fujinama uploads the newly created image to its C2, using a http *POST* request with *content-type multi-part* before deleting the file from the victim's device.

The entire flow of the screenshot routine: from capture to upload and deletion

Keylogger

The *keylogging routine* simply waits for the user input, once a keystroke has been typed it is immediately uploaded to the C2. Surprisingly the keystroke is transferred using a simple *GET* request, this approach – although ignored by the local antivirus – is both visible and noisy, most likely this is what gave up the presence of the malware on its first detection.



Keylogger transferring the keystroke via GET

The keylogger routine is quite common, as shown in the API list below.

```
Private Declare Function GetWindowText Lib "user32" Alias "GetWindowTextA" (ByVal hwnd As Long, ByVal lpString As String, ByVal cch As Long) As Long
'VA: 402D88
Private Declare Function GetForegroundWindow Lib "user32" Alias "GetForegroundWindow" () As Long
'VA: 402D3C
Private Declare Function GetAsyncKeyState Lib "user32" Alias "GetAsyncKeyState" (ByVal vKey As Long) As Integer
```

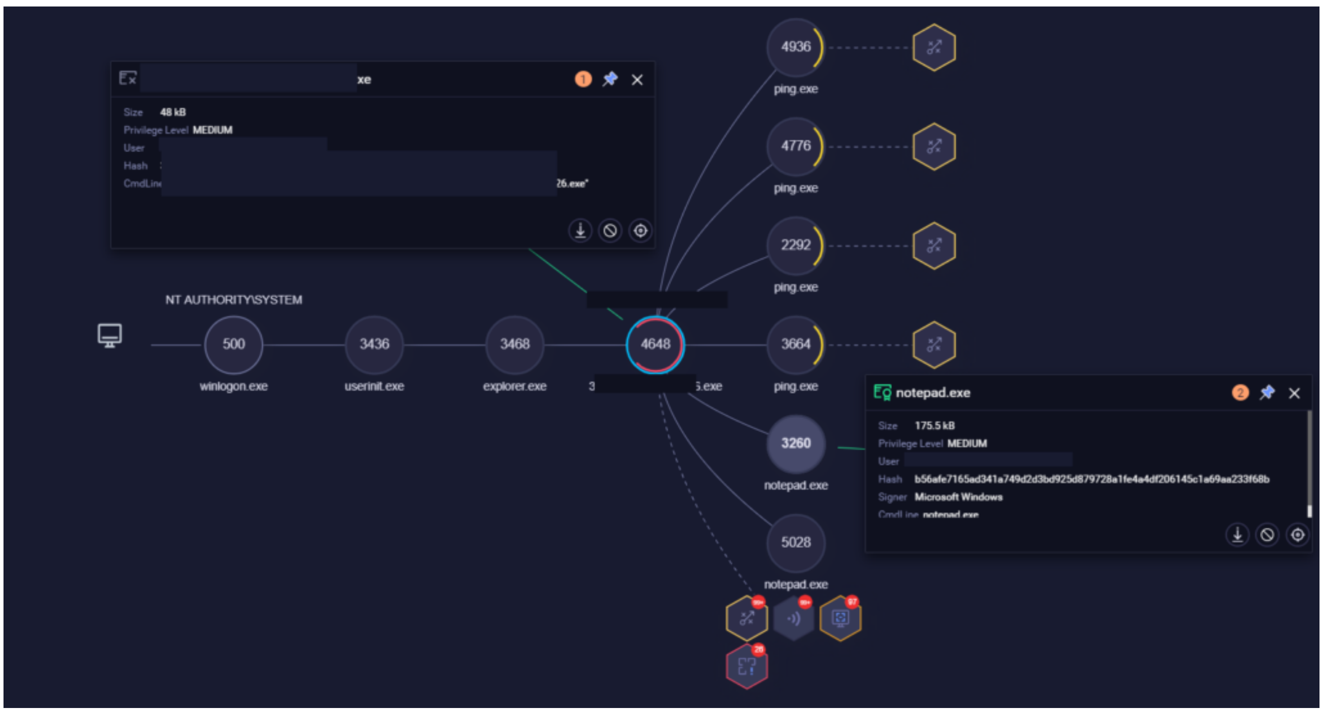
Windows APIs used by the keylogger

C2 Commands

An interesting part of Fujinama is the ability to execute custom commands and custom exfiltrations as instructed by the C2. Every 5 minutes a *configuration file* stored on the C2 for **each** infected endpoint, is polled. The samples we have analyzed support 2 commands:

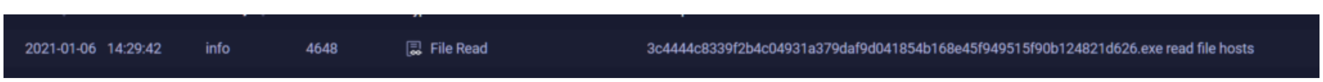
- **CMD:** contains the commandline to execute on the infected endpoint
- **SND:** exfiltrates a specific file from the endpoint

Below we show how it is possible to run custom commands from the C2.



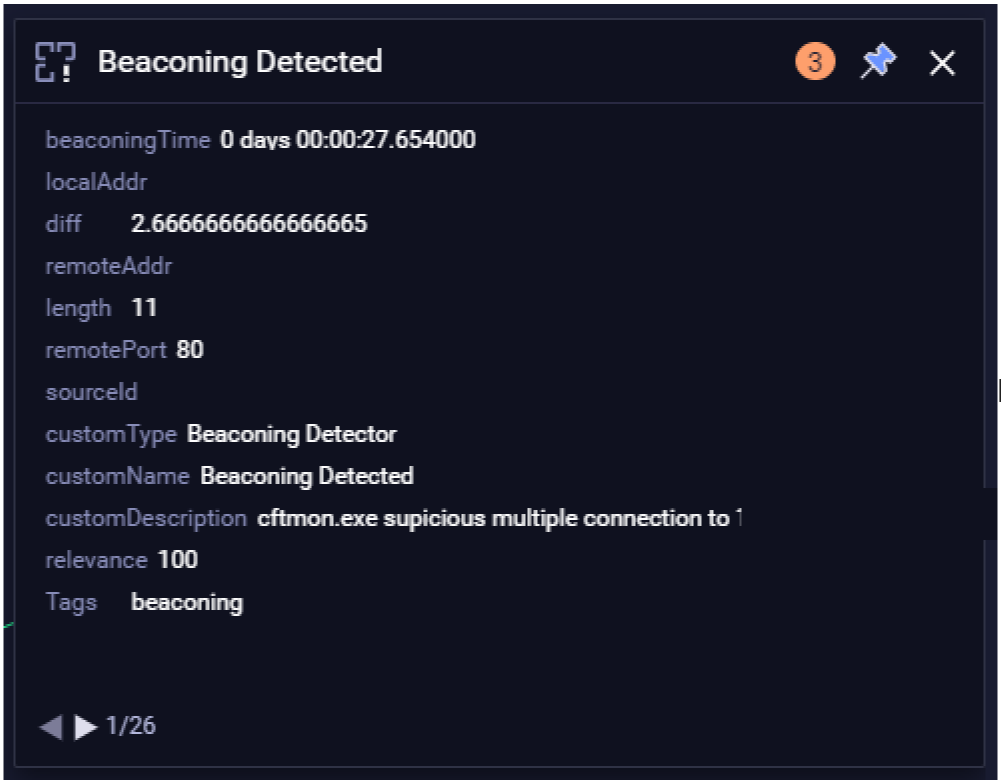
Process creation after the C2 instructed Fujinama via CMD command

Exfiltration is also confirmed using ReaQta-Hive, showing below Fujinama as it captures the *hosts* file from the infected endpoint, before delivering it to the C2.



File read of the *hosts* file after sending the SND command

The RAT's beaconing is automatically detected and alerted by ReaQta-Hive's engines. Below is a screenshot of Fujinama *phoning home* at regular intervals (with minor drifting due to other parts of the RAT contacting the C2).



Fujinama Beaconing

Variants

ReaQta has so far identified 3 different Fujinama samples, 2 of them certainly used on Leonardo's infrastructure while the third appears to be part of a different project.

- *Sample 1*: used on Leonardo (Keylog)
- *Sample 2*: used on Leonardo (Keylog, Screenshots, Remote commands)
- *Sample 3*: under investigation (Keylog)

The analysis shown above has been run on *Sample 2*, both *Sample 1* and *Sample 2* share the same C2 infrastructure. *Sample 2* is at all effects an evolution of *Sample 1* that acquired new capabilities (Screenshots and Remote command support) that were not present in the previous version.

(Sample 1) cftmon v3.3	(Sample 2) cftmon v3.5
Keylogging	Keylogging
	Screenshots
	Remote Commands

Capabilities for Sample 1 and Sample 2

We measured both the code similarity (95%) and behavioral similarity (99%) between *Sample 1* and *Sample 3,* confirming they're an almost exact match. *Sample 3* shares without doubts the same codebase used for *Sample 1*, with a few notable differences:

- *Sample 1* and *Sample 2* appear to be compiled on the same machine, *Sample 3* seems to have been compiled on a different device
- *Sample 1* and *Sample 2* and *Sample 3* share the same *language id* (Italian) though at least one keylogger string has been translated back to english
- *Sample 1* and *Sample 2* use the same C2, *Sample 3* uses a different one
- *Sample 1* and *Sample 2* share the same original filename: *cftmon.exe*, *Sample 3* uses a different one: *igfxtray.exe*

(Sample 1) cftmon v3.3	(Sample 3) dllhost.exe
fujinama[.]altervista.org	xhdyeggeeefeew[.]000webhostapp.com
“/copy.php? file=” & var_3C & “&name=”	“/XdffCcxuiusSSxvbZz.php? ZmlsZQo=” & var_3C & “&bmFtZQo=”
var_24 & “[INVIO]”	var_24 & “[RETURN]”

Code differences between Sample1 and Sample3

Lastly *Sample 3* appears to have been compiled quite recently compared to the other samples.

Sample	Compilation Time
Sample 1	2015-05-28 08:02:25
Sample 2	2015-07-14 12:33:39
Sample 3	2018-09-22 23:10:46

Compilation time for the various samples

Given the timeline, the third sample could not be used on Leonardo. We haven’t yet found traces of how, or where, the third sample was used but it’s possible that the malware project was shared with a third party that managed to alter a few parts.

Detection

ReaQta-Hive natively detects Fujinama, so no actions or updates are required from our customers and partners.

We can’t share the samples yet, although the article provides enough data to allow researchers to find them. Nevertheless given the presence of the third sample and the slim – but not negligible – possibility that someone is still maintaining this project, we would like to share with the community a Yara rule to identify Fujinama variants.

```
rule Fujinama {
    meta:
        description = "Fujinama RAT used by Leonardo SpA Insider Threat"
        author = "ReaQta Threat Intelligence Team"
        ref1 = "https://reaqta.com/2021/01/fujinama-analysis-leonardo-spa"
        date = "2021-01-07"
        version = "1"
    strings:
        $kaylog_1 = "SELECT" wide ascii nocase
        $kaylog_2 = "RIGHT" wide ascii nocase
        $kaylog_3 = "HELP" wide ascii nocase
        $kaylog_4 = "WINDOWS" wide ascii nocase
        $computername = "computername" wide ascii nocase
        $useragent = "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1; .NET
CLR 2.0.50727)" wide ascii nocase
        $pattern = "'()*+,G-./0123456789:" wide ascii nocase
        $function_1 = "t_save" wide ascii nocase
        $cftmon = "cftmon" wide ascii nocase
        $font = "Tahoma" wide ascii nocase
    condition:
        uint16(0) == 0x5a4d and all of them
}
```

Conclusions

As we have just shown, the malware is not particularly sophisticated but it certainly reached its goal. Sending data in clear with a simple *GET* is a major oversight for an actor that, supposedly, wants to remain undetected. The frequent beaconing and the absence of all kinds of hiding/evasion mechanisms (with the exception of a basic sandbox evasion technique) shows either a lack of care, or a lack of structure. One factor that contributed to the success of the attack was that the installation was performed manually, thus not requiring sophisticated evasion techniques. At the same time, the level of security expected from a major defence contractor should have pushed a sophisticated attacker toward a very different *modus operandi*.

In our view the attack has been built opportunistically over time, with incremental enhancements, lacking the structure of a real APT and certainly the sophistication. Unless the attacker could leverage on his position within the company, to make sure he couldn’t be detected, we can’t see any reason why he was expected to otherwise remain hidden. In this regard, the code changes only show an increase in exfiltration capabilities, while completely neglecting the detection aspect.

Interested to try out ReaQta-Hive? Schedule a free trial *here*