Threat Spotlight: Solarmarker

By Chris Neal

Published: 2021-07-29 · Archived: 2026-04-05 18:01:18 UTC

Thursday, July 29, 2021 13:00

By Andrew Windsor, with contributions from Chris Neal.

Executive summary

- Cisco Talos has observed new activity from Solarmarker, a highly modular .NET-based information stealer and keylogger.

- A previous staging module, "d.m," used with this malware has been replaced by a new module dubbed "Mars."
- Another previously unreported module named "Uranus" has been identified.
- The threat actor behind Solarmarker continues to evolve while remaining relatively undetected.
- Cisco Talos has created full coverage in response to this evolving threat, protecting Cisco customersTalos is actively tracking a malware campaign with the Solarmarker information-stealer dating back to September 2020. Some DNS telemetry and related activity even point back to April 2020. At the time, we discovered three primary DLL components and multiple variants utilizing similar behavior.

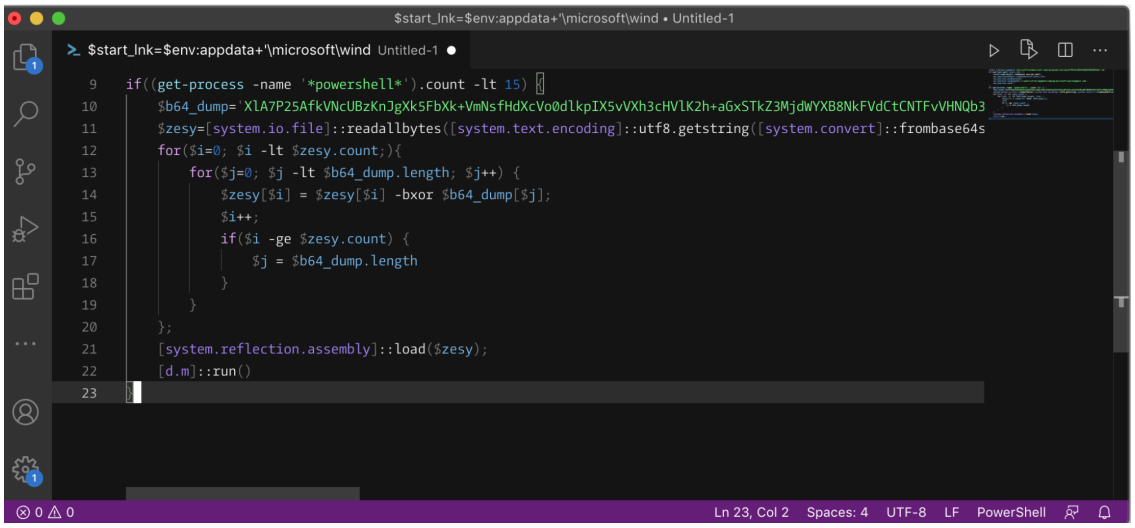
First, the initial malicious executable injects the primary component, typically named "d.m." This serves as a stager on the victim host for command and control (C2) communications and further malicious actions. The second component, commonly referred to as "Jupyter," was observed being injected by the stager and possesses browser form and other information-stealing capabilities. Another secondary module, named "Uran" (likely in reference to Uranus), is a keylogger and was discovered on some of the older campaign infrastructure. Uran was previously undiscovered despite deep analysis on Solarmarker and the Jupyter module.

Modules

To provide a more comprehensive description of Solarmarker, we'll break down known and unreported modules. Information regarding coverage and defense are detailed at the end of this blog post.

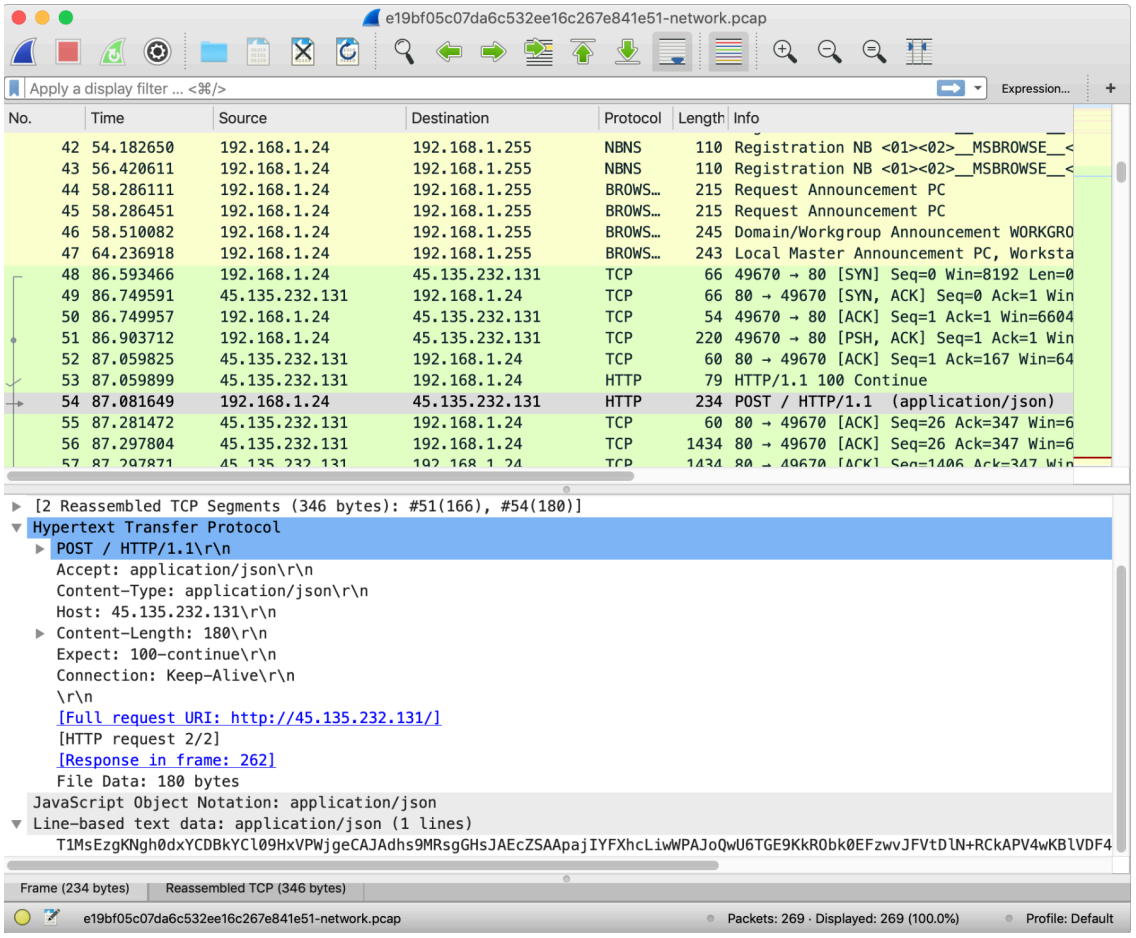
Staging module "d.m"

The staging component of Solarmarker serves as the central execution hub, facilitating initial communications with the C2 servers and enabling other malicious modules to be dropped onto the victim host. Within our observed data, the stager is deployed as a .NET assembly named "d" and a single executing class named "m" (referred to jointly in this analysis as "d.m"). The malware extracts a number of files to the victim host's "AppData\Local\Temp" directory on execution, including a TMP file with the same name as the original downloaded file, and a PowerShell script file (PS1), from which the rest of the execution chain spawns. The TMP file executing process issues a PowerShell command that loads the content of the dropped PS1 script and runs it before deleting the loaded file. The resulting binary blob is then decoded by XORing its byte array with a hardcoded key and injected into memory through reflective assembly loading. The "run" method of the contained module "d.m" is then called to complete the initial infection.



Reconstructing the malicious DLL "d.m" and invoking it through reflection. The module "d.m" acts as a system profiler and staging ground for additional action by the actor. One particularly interesting operation (as well as the namesake of the campaign) is the file write of "AppData\Roaming\solarmarker.dat," which serves as a victim host identification tag. When the class method "GetHWID" is called, the sample checks if "solarmarker.dat" exists already on the host. If it does not, a randomly generated 32-character string will be written to the file "solarmarker.dat" at that path. It should be noted that some of the older variants of Solarmarker don't actually use this file for storing the victim ID and instead use varying forms of concatenating and encoding the collected system information strings.

The newly created or existing string is then returned back as the value for the "hwid" field in a JSON object also containing fields for collected system information. This object is subsequently encoded by XORing the byte array with a hardcoded key, similar to how the first stage PowerShell script constructed the malicious DLL. This new byte array is then further encoded in base64 in preparation for transmission to a hardcoded C2 IP address — 45.135.232[.]131 at the time in many of our samples — through HTTP POST requests.

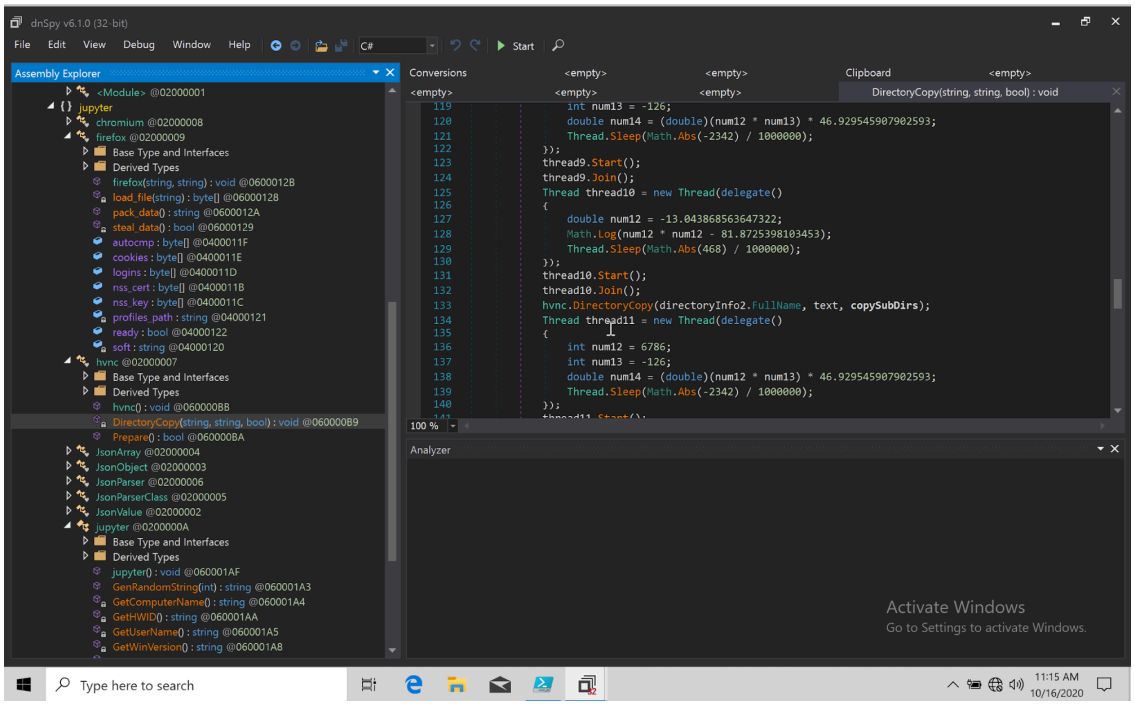


HTTP POST communications with the C2 using encoded JSON.

The stager also possesses the ability to remotely receive execution commands by dropping new PS1 or PE files onto the victim system at "\AppData\Local\Temp\" directory with a filename of 24 randomly generated alphanumeric characters and executing them either through PowerShell or process hollowing, or through directly receiving PowerShell text commands.

Jupyter information-stealing module

The Jupyter information stealer is Solarmarker's second most-dropped module. During the execution of many of the Solarmarker samples, we observed the C2 sending an additional PS1 payload to the victim host. Responses from the C2 are encoded in the same manner as the JSON object containing the victim's system information. After reversing the base64 and XOR encoding, it writes this byte stream to a PS1 file on disk, runs it, and subsequently deletes the file. This new PowerShell script contains a base64-encoded .NET DLL, which was also injected through .NET's reflective assembly loading. Analysis of the DLL module, named "Jupyter," shows that it contains capabilities to steal personal information, credentials, and form submission values from the victim's Firefox and Chrome installation and user directories.



Decompiled source of one version of the Jupyter DLLs.

Like the "d.m" DLL, this module sends information to its C2 server through HTTP POST requests, using a similarly encoded JSON stream containing additional system information and stolen data. However, the previous version of the Jupyter DLL we analyzed opts to use a hardcoded domain, "vincentolife[.]com" in one case, rather than use the same hardcoded IP address as the stager component. The Jupyter module also adds a POST request at the end of its execution loop, which queries the C2 domain for additional information and/or commands that could add to the data gathered from its primary execution loop.

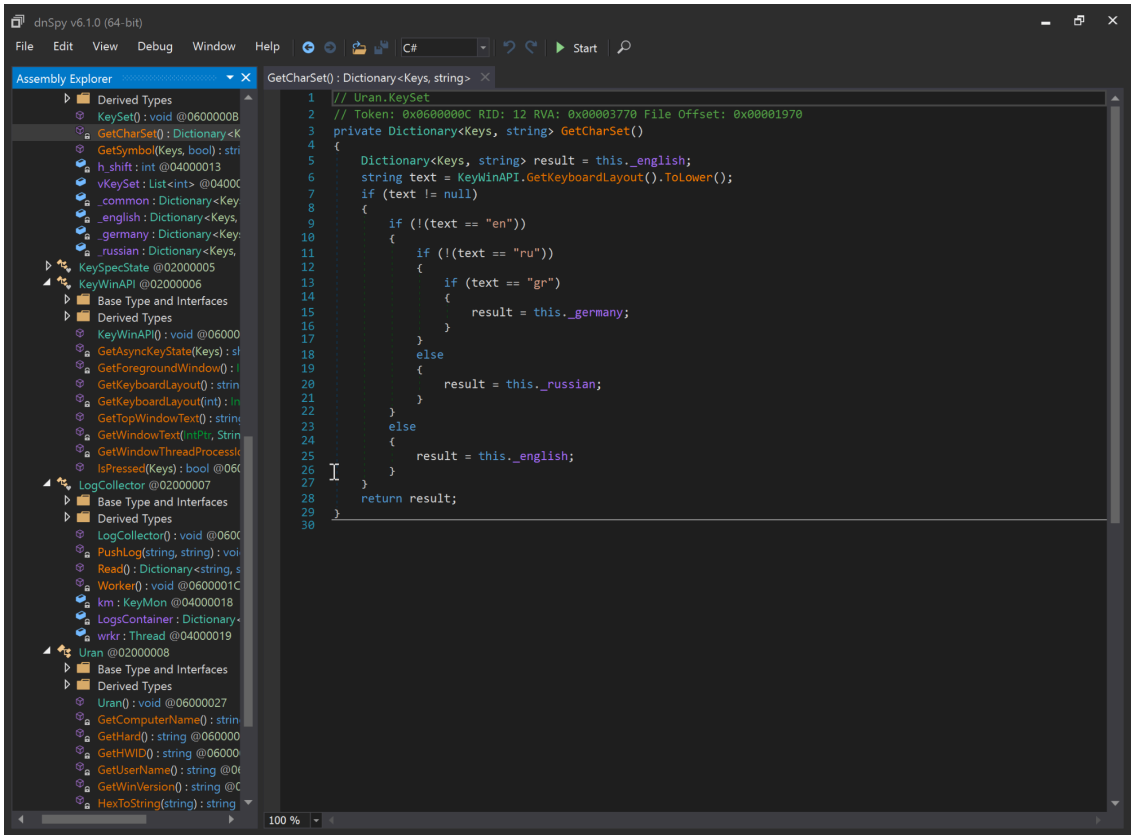
Unlike the "d.m" DLL, rather than use a similar XOR encoding scheme, the versions of Jupyter we analyzed use the .NET "ProtectedData" class to encrypt the data being sent to Solarmarker's C2 servers. This keeps the encoding scheme less exposed to static and manual analysis by using .NET's native APIs, in contrast to the hard-coded key values and custom encoding logic in the PowerShell scripts and "d.m" DLL. Additionally, by including the "CurrentUser" flag for the data protection scope argument in the "Unprotect" method call, the encryption key is effectively binded to a specific user. This complicates any attempts at outside decryption and analysis of the raw data being communicated between the victim host and the C2 server, even with possession of the "Jupyter" DLL itself.

Previously Unreported Module: Uran/Uranus

Further research into the actor's infrastructure revealed a previously unreported second potential payload, "Uranus," keeping in line with Solarmarker's penchant for space-themed names. It is derived from the file "Uran.PS1" (Uran is the Russian word for Uranus) hosted on Solarmarker's infrastructure at "on-offtrack[.]biz/get/uran.ps1." Like our observed instances of the Jupyter module being dropped, Uranus starts off as a PS1 file served through the staging DLL. Similar to the other components, the PS1 file was encoded through a simple bitwise XOR of each byte in the file. Going back through some of the encode-decode class methods found in both the variants of the staging component and Jupyter module allowed us

to match Uranus with the correct decoding method. Upon execution, the PowerShell script reflectively loads the DLL class, "Uran," on the victim host and initiates its "Run" method.

Analysis of the Uranus module reveals it to be a form of keylogging malware. When the "Run" method is executed, the malware instantiates a "LogCollector" class object. Its functionality is relatively straightforward, but makes impressive use of a variety of tools within the .NET runtime API. The "LogCollector" makes use of other utility classes contained within the Uranus DLL to capture the user's keystrokes and relevant metadata in order to contextualize and sort the actions better. For example, it will look for available input languages and keyboard layouts installed on the victim host and attach their two letter ISO codes as additional attributes to the keylogging data collected. Interestingly, in this case, the actor checks specifically for German and Russian character sets, before defaulting to an English label. The current foreground window titles are also used as sorting keys in the dictionary object that the "LogCollector" class uses to store its captured input text.



Decompiled source of the Uranus keylogger DLL.

Extraction is set to occur every 10,000 seconds using a thread sleep call to delay Uranus' event loop. This module also uses HTTP POST requests as its primary method of communications with Solarmarker's C2 infrastructure. This version of the DLL has the hostname "hxxp://spacetruck[.]biz:82/postk?q=" hard-coded as the destination for the data held by "LogCollector." The query string consists of two components. First, a grouping of system metadata, such as the victim ID, version string, and computer name are constructed as a JSON object and converted to base64. Unlike the variants that used the "solarmarker.dat" file filled with random characters as a victim identifier, this version of the module instead uses a combined string of the victim host's username, computer name and hard drive serial number. The byte representation of this string is then run through a byte-shift and XOR algorithm to produce the victim identification string. Secondly, the dictionary object containing the

keylogging data created by "LogCollector" is also converted to base64 and appended to the query string text. Finally, the POST request is sent and the event loop begins again.

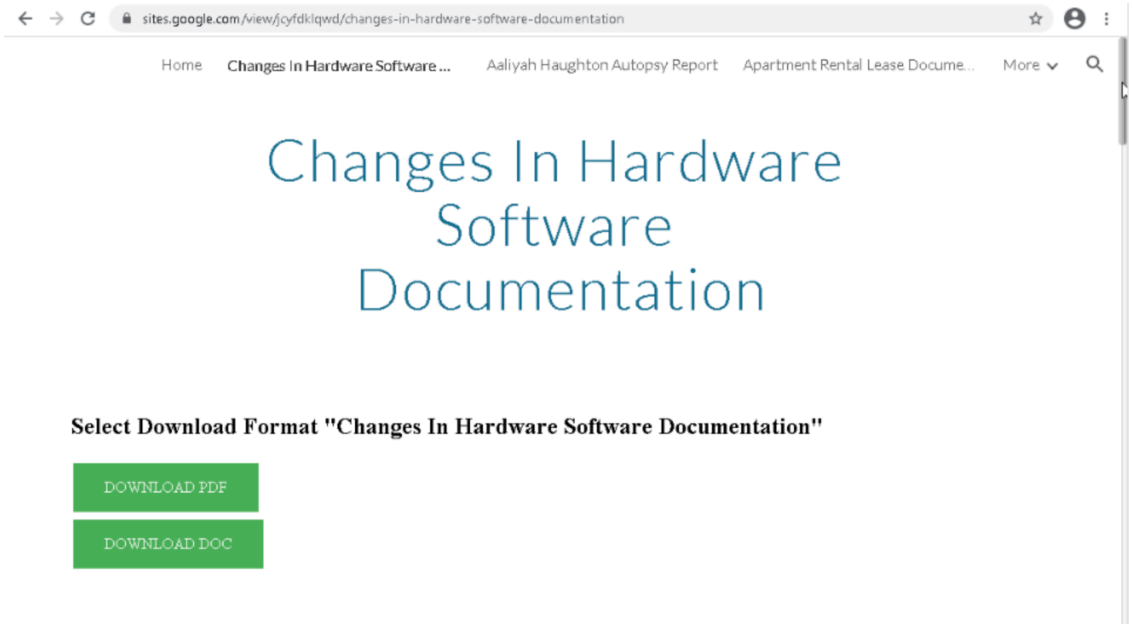
Previously unreported activity

Solarmarker's general execution flow hasn't significantly changed between versions, updates and variants. The actor wants to install a backdoor in memory on the victim host so that it may be used as a staging ground for further payloads, such as Jupyter. However, around the end of May and beginning of June 2021, Talos began observing surges of new Solarmarker activity in our telemetry. In these recent iterations, the actor tweaked the download method of the initial parent dropper and has made noticeable upgrades to the staging component, now called "Mars."

During our research on earlier campaign activity, Talos initially believed that victims were downloading Solarmarker's parent malicious PE files through generic-looking, fake file-sharing pages hosted across free site services, but many of the dummy accounts had become inactive between the time we found the filenames used by Solarmarker's droppers in our telemetry and attempting to find their download URLs. This method of delivery was later corroborated by malware analysts at CrowdStrike in their own reporting on Solarmarker. For example, we saw several download pages being hosted under suspicious accounts on Google Sites. In these cases, the URL for a given file took the form of: `hxxps://sites[.]google[.]com/view/{random characters}/{file name}`

These links direct the victim to a page offering the ability to download the file as either a PDF or Microsoft Word file. Following the download link sends the victim through multiple redirects across varying domains before landing on a final download page.

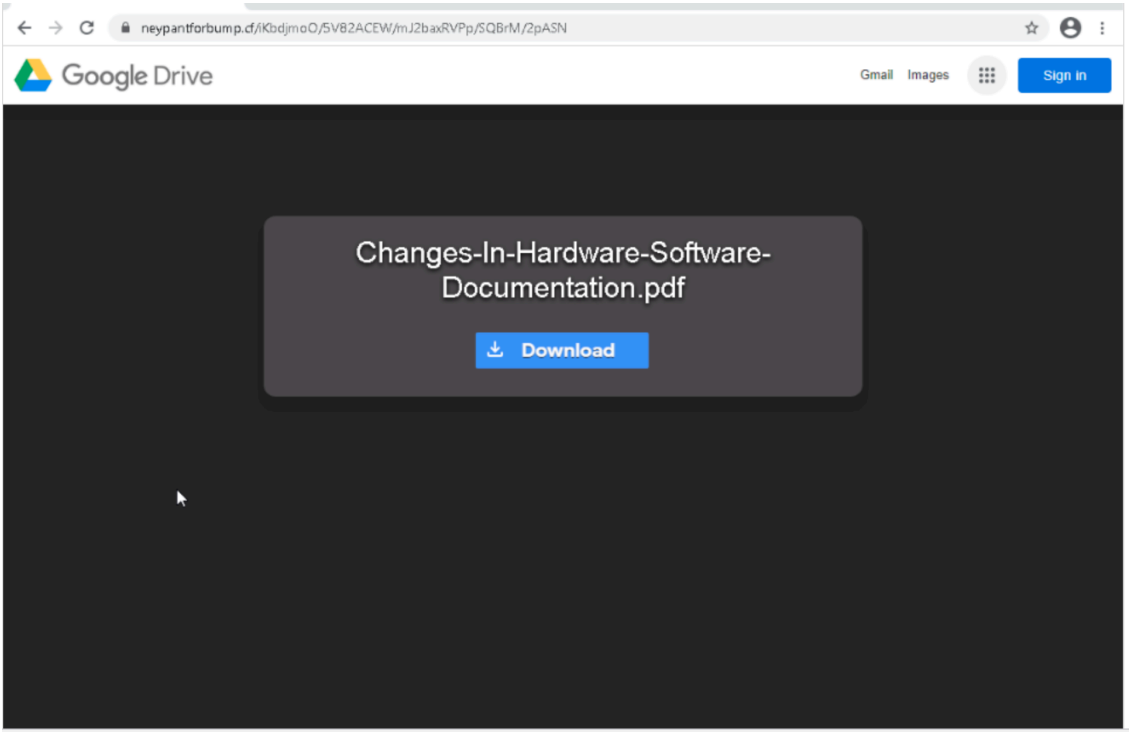
This general methodology hasn't changed, many of the parent file names found in our telemetry can be found on suspicious web pages hosted on Google Sites, although the actor has changed their final lure pages a bit.



Example of one of the initial landing pages hosted on Google Sites.

Previously, the Solarmarker samples from our telemetry would be downloaded from a final page with an overly

generic header name, "PdfDocDownloadsPanel." In this more recent campaign, Solarmarker's operators have put some additional effort into making their final download page more legitimate looking, as well as differentiating it from the older pages exposed in public reporting. The final download page now attempts to mimic a download file request from Google Drive, although the page is hosted under a rotating actor-controlled domain.

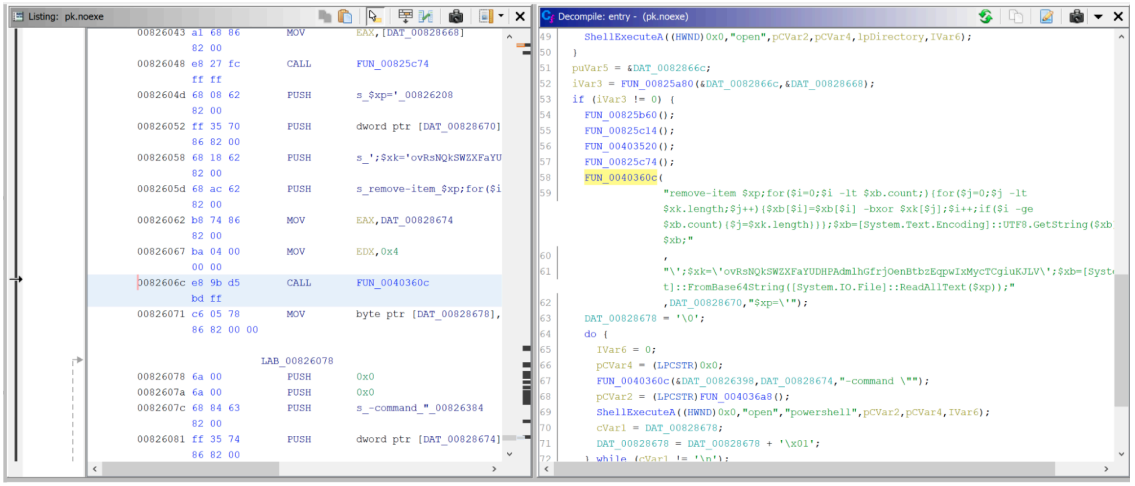


Example of a final lure page meant to mimic Google Drive.

Mars staging module

Updated Solarmarker variants possessing the newly observed Mars DLL, which took the place of the original stager component "d.m," infect a victim host through two currently known execution chains. The first uses a second-stage dropper after initial execution of the parent file that contains a decoy program PDFSam as well as the Mars DLL bytes and subsequently executed PowerShell to complete Solarmarker's later stages and persistence measures. The second version has the same overall execution chain as the first, but hardcodes all of the data and instructions from the second-stage dropper into the parent file itself, cutting out the need to write an additional file to disk and execute it. These are based on observations within available telemetry. Given the actor's penchant for deploying many different versions of Solarmarker and its modules, it is likely that the adversary added variations on the execution chain and different paths and names for the dropped files.

Using the version with the second-stage dropper as an example, the main file drops a second stage PE, named "pk.exe," which acts as the springboard for the rest of this Solarmarker variant's execution chain. It writes a copy of the utility program PDFSam to "\AppData\Local\Temp\gkNjg0918jkd.exe." This program is executed in tandem with the rest of Solarmarker's initialization to act as misdirection for the victim by attempting to look like a legitimate document. The main PowerShell script that will serve as the next execution stage is also written to disk at "\AppData\Local\Temp\tskAjbflM1jj." The PE then opens a shell and executes a PowerShell constructed from inside the binary).



Powershell embedded in pk.exe used in the next stage of execution.

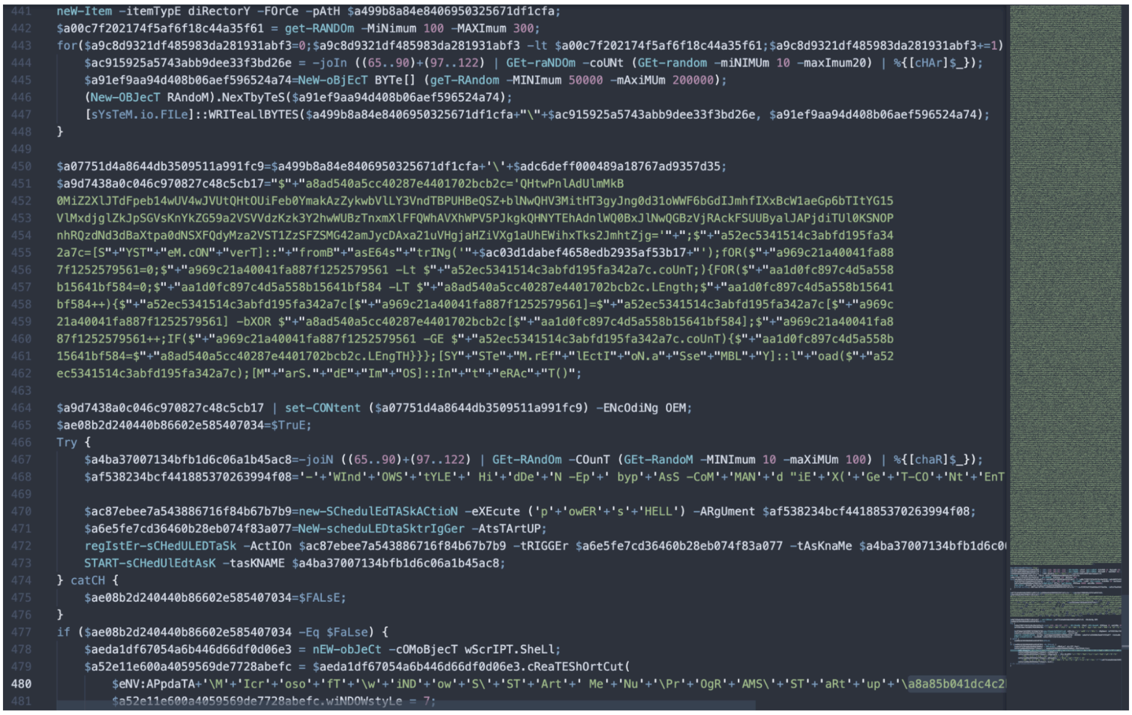
The PowerShell command executed will read in the base64-encoded text from the "tskAjbflM1jj" file and subsequently delete this file from the victim's hard drive. Next, it will XOR the decoded bytes using a hardcoded alphanumeric key string, in this case: xk='ovRsNQkSWZXFaYUDHPAdmlhGfrjOenBtbzEqpwIxMycTCgiuKJLV.'

After the bytes are XOR'd, the resulting string is another PowerShell script, which is then executed through the Invoke-Expression cmdlet.

This second PowerShell script contains the Mars DLL hardcoded within it, along with the final stages of Solarmarker's initial execution. The DLL file is written to disk at:

"%USER%\APPDATA\ROAMING\MiCrOSofT\<Random string of 5-20 characters>\<Random string of 10-20 characters>"

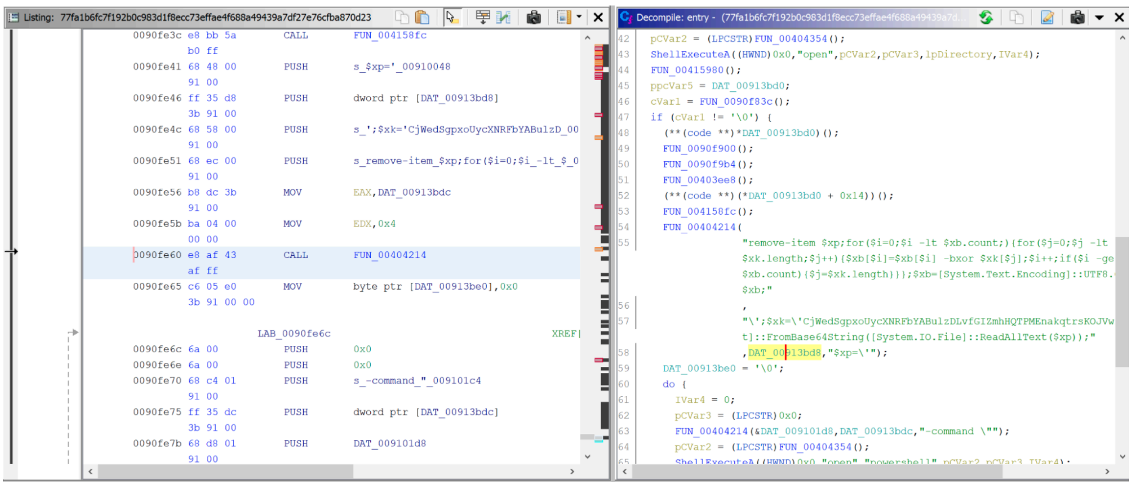
The actual file write is done by looping through the Mars DLL byte array while injecting random bytes into the file so that the actual malicious DLL doesn't sit on the victim's disk. The script will then construct a PowerShell command that reads the newly written file and reverses the byte manipulation, as well as performing byte XORing with another hardcoded key, similar to the previous PowerShell command. The result is the actual Mars DLL, which is loaded through assembly reflection and executed by calling the "Interact" method from the "Mars.Deimos" class. Finally, the script sets up persistence measures by creating a startup task that copies the final PowerShell command and re-runs the execution of the Mars DLL. It will try to create this through the PowerShell Register-ScheduledTask cmdlet, and upon that action failing, will instead create a WScript object containing the PowerShell command to load and execute the Mars DLL and write it to a shortcut file in the victim's startup folder named "a8a85b041dc4c2bfc00abf64d546d.LnK".



PowerShell script setting up persistence measures and invoking the Mars DLL.

After the dropper's intermediate stages of execution are completed, the Mars module is reconstructed from a file written to disk after being run through an XOR algorithm in PowerShell, similar to the campaign's previous execution chains. The resulting malicious DLL file is loaded through .NET's assembly reflection using PowerShell and invoked by calling the "Interact" method of the "Mars.Deimos" class.

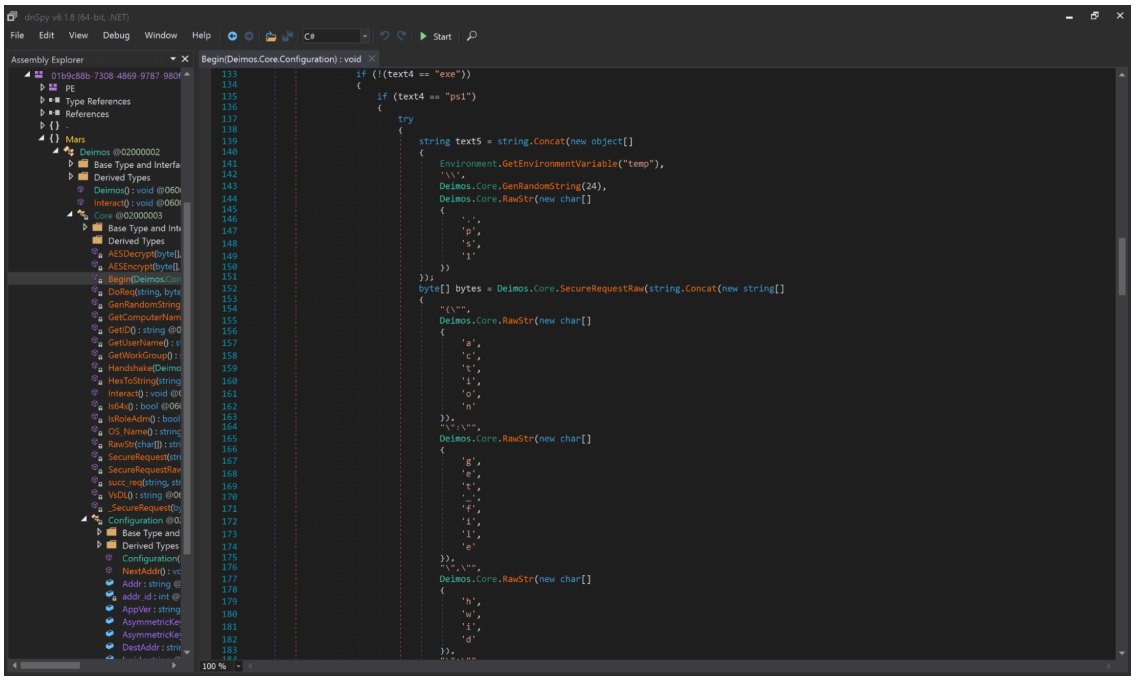
As mentioned above, the second version of the execution chain forgoes the secondary dropper file and instead has the information contained in "pk.exe" encoded in the parent PE file. In these samples, the decoy program PDFSam was written to the hardcoded location of "%USER%\AppData\Local\Temp\CMmnnjAi1984unbd.exe," while the encoded PS1 file is written to disk at "%USER%\AppData\Local\Temp\FkJB11kdJJhbdDl." The remainder of the execution is similar to the one stemming from "pk.exe."



Second stage PowerShell code now embedded in the parent PE file.

Decompilation of the DLL shows that it first gathers some initial system information about the victim host, such as the Workgroup name, is the user a system administrator, system name, current user, and Windows version. This

information is collected through execution calls to different methods inside of the Deimos class and the results are collected into a JSON text string and sent to Solarmarker's C2 server through HTTP POST requests. Previous components would alternate between using hostnames and IP addresses to contact the C2 server. In this case, these versions of the Mars module use a hardcoded IP value: 46[.]102[.]152[.]102. This C2 host had not been observed in older versions of Solarmarker's components previously analyzed.



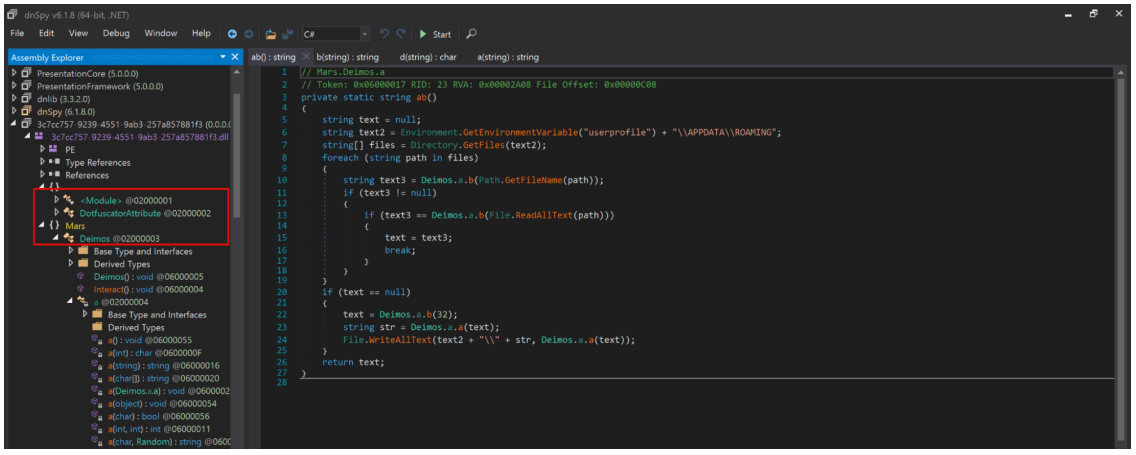
Decompiled code from a non-obfuscated Mars DLL.

Like the previous stager, the Mars module also remotely receives execution commands by dropping new files onto the victim system and spawning processes to run them, or through direct PowerShell text commands. During the execution of the DLL's "Begin" method, if the "flag" boolean variable is set to "false," the malware on the victim host is set to receive a file or command response from the C2. For file responses, the sample expects either a PS1 or PE file based on the values associated with the "status," "task_id" and "type" keys in the JSON object. The received bytes are written to the "\AppData\Local\Temp\" directory with a filename of 24 randomly generated alphanumeric characters. If Mars receives a PowerShell command string, the command is executed directly without writing anything to disk. These files are then subsequently deleted from the victim's hard drive. In inspecting network communication from the earlier Solarmarker campaign, we observed the Jupyter module dropping through a PS1 file.

The Mars module also uses a more complex encryption scheme than its predecessor to better obfuscate the data sent to and from the C2 server. Previously, Solarmarker would use a combination of base-64 encoding and a bitwise XOR algorithm, or a simple implementation of .NET's default encryption scheme, to encode the JSON being sent back and forth in the HTTP POST traffic. The actor now uses a 16 byte, randomly generated AES symmetric key to encrypt the data sent to and from the C2 host, which is created during the Mars DLL's initial execution. This key is sent with the victim identification string back to the actor's C2 in the first HTTP request, which is itself encrypted using an RSA asymmetric key pair. The public key is hardcoded within the Mars DLL, making the symmetric key, and thus, a readable version of the POST JSON data much more difficult to obtain in general practice.

Like the "d.m" and Jupyter modules, the Mars DLL has a few different versions. At the time of writing, five different variants are seen in our telemetry and VirusTotal data. These possessed the identifiers: "IN-1," "IN-2," "IN-3," "IN-9" and "IN-10." Most of Solarmarker's variants have left the "solarmarker.dat" artifact on disk that makes it easy to identify the actor's activity. As previously mentioned, some of the older variants of Solarmarker don't actually use this file for storing the victim ID and instead use varying forms of concatenating and encoding the collected system information strings, but for most of the observed instances between October 2020 and June 2021 this file is present. However, Solarmarker's operators appear to have noticed this and have attempted to further obfuscate their modules, making it more difficult for defensive measures to detect specifically on the "solarmarker.dat" file.

The higher versioned variants of Mars, "IN-9" and "IN-10," no longer use the "solarmarker.dat" file to house the victim's identification string. Instead, they use the randomly generated 32-character string as the text file name and the file's contents. The check to see if "solarmarker.dat" already exists on a victim has also been changed to cycle through all the file names in the path "%USER%\APPDATA\ROAMING" and look for a file whose name is equivalent to the extracted text within it. These versions also attempt to obfuscate the C# code itself by using the Dotfuscator packer, thereby frustrating efforts at analyzing its decompiled form.



Mars DLL that has been obfuscated through Dotfuscator.

Targets

At its core, the Solarmarker campaign appears to be conducted by a fairly sophisticated actor largely focused on credential and residual information theft. The specific extraction of browser forum captures, cookie data, and credential database files in the Jupyter module, along with the keylogging component of the Uranus module lends further credence to a specific interest in credential harvesting. The targeted language component of the keylogger suggests that the actor could have a heightened geographical interest in Europe, or it does not have the ability to process text in languages outside of English, German or Russian. During initialization, the class object responsible for monitoring the victim's keypresses checks specifically whether the current host's keyset language code is one of those three languages. If the returned language code is none of these, it defaults to labeling its captured text as English. This really only attaches specificity to German or Russian-speaking victims, as it indiscriminately mixes other language sets that use

variants of the Latin script and non-Latin languages, such as the CJK group, all under the umbrella of "English." Regardless, they are not particular or overly careful as to which victims are infected with their malware. This is evidenced by the large variety of composition for the parent file names and the absence of similar geographical checks on the victim systems before dropping and executing any of the later-stage modules.

During this recent surge in the campaign, Talos observed the health care, education, and municipal governments verticals being targeted the most often. These sectors were followed by a smaller grouping of manufacturing organizations, along with a few individual organizations in religious institutions, financial services and construction/engineering. Despite what appears to be a concentration of victimology among a few verticals, we assess with moderate confidence that this campaign is not targeting any specific industries, at least not intentionally. The variety of naming in Solarmarker's lure files is too great to be of any specific relevance to a targeted industry. To this point, we originally suspected that the actor may have been scraping document file names across the web to use as file names for its dropper. Researchers at Microsoft have also stated they believe the Solarmarker campaign is using SEO poisoning in order to make their dropper files highly visible in search engine results. This could also potentially skew what types of organizations are likely to come across the malicious files depending on what is topically popular at the time.

Organizations should be particularly concerned about the modular nature and information stealing capabilities of this malware family. Solarmarker has so far shown to split, not only its staging activity, but also final payloads of varying functionality into separate DLL components. Using the staging DLL, the malware can then execute whichever payload module they choose, some of which may be previously undiscovered. The modules already observed make potential victims vulnerable to having sensitive information stolen, including sensitive browser usage, such as if they enter their credit card number or other personal information. These attackers may also look to steal login credentials, which could then be used for lateral movement into other systems or to access and steal even more enticing data, such as a customer or patient medical information database.

Threat actor

There are indicators and evidence suggesting that Russian language speakers created Solarmarker, or the creators at least designed it to look that way. Static and dynamic analysis of Solarmarker's droppers revealed attributes in the executables' resource section that indicate the files were created on a system with Russian language support. The keylogging module's name, "Uran," is a transliteration of the Russian word for Uranus, "Уран," and Russian was listed as a targeted language in the module itself. The name of the browser credential and information stealer, "Jupyter," could also be an attempted transliteration or translation of the Russian equivalent for Jupiter, "Юпитер," albeit an odd one. A more commonly accepted transliteration for a native Russian speaker would be "Yupiter." The names for the "Mars" module and its class object "Deimos" work both as direct English words and possible transliterations of their Russian equivalents, "Марс" and "Деймос."

Character use and substitution doesn't have universally applied rules in Russian when using the Latin alphabet. Therefore, the naming scheme is not all that odd within the given context. Security researchers from Morphisec, in

their analysis of the Jupyter module, also pointed out similar indicators of the actor potentially being Russia-based, such as possibly misspelling the English "Jupiter" and the hardcoded IP addresses being hosted on "SELECTEL," a Russian ASN, which was also true for IP addresses found in other modules and Solarmarker variants during earlier stages of the campaign. While it is interesting to highlight this pattern, and other circumstantial evidence, it is important to note that this is not sufficient to assign any high-confidence attribution. All of these components are fairly easy to manipulate into arbitrary values by the actor, and thus, without additional evidence, could just as well be false flags.

Conclusion

The actor behind the Solarmarker campaign possesses moderate to advanced capabilities. Maintaining the amount of interconnected and rotating infrastructure and generating a seemingly limitless amount of differently named initial dropper files requires substantial effort. The actor also exhibits determination in ensuring the continuation of their campaign, such as updating the encryption methods for the C2 communication in the Mars DLL after researchers had publicly picked apart previous components of the malware, in addition to the more typical strategy of cycling out the C2 infrastructure hosts.

There are several proactive and remedial actions to mitigate the risk of infection, data theft, and potential further action post-infection by Solarmarker. Initial execution of Solarmarker is seemingly reliant on the user willingly downloading and opening its parent file. Organizations need to educate users on the risks of downloading and executing files from unvetted or suspicious sources on the internet.

There are other aspects to the malware that are more susceptible to active defense measures. Restricting the use of administrative tools, in this case PowerShell, can help prevent any of Solarmarker's numerous scripts from executing. Solarmarker's observed C2 components are hardcoded into their respective DLLs. Monitoring for any traffic to either the IP address or domain name can reveal which endpoints have been affected on a network and how far the infection has progressed. Blocking traffic to and from both of these indicators effectively hamstrings Solarmarker's ability to receive further instructions from its operators and to exfiltrate any collected data from the victim host. Using multi-factor authentication can help prevent immediate account compromise if a victim believes their information and credentials have already been stolen. This also provides cover time to cycle out the credentials for any affected accounts. Finally, ensuring that sensitive systems are properly segmented, have correct privilege requirements, and are not unnecessarily facing the internet can prevent lateral movement by the actor using stolen credentials and data exfiltration to their C2 server.

Solarmarker's ongoing campaign and associated family of malware are concerning. It was initially able to operate and evolve over a significant amount of time while remaining relatively undetected. Its ability to receive and drop additional files, in addition to executing PowerShell commands from the actor's C2, opens up the possibility of future updates and evolutions of Solarmarker's child modules, as well as additional functionalities through new malicious components being developed by the actor. We expect the actor behind Solarmarker to continue to refine their malware and tools, as well as alternate their C2 infrastructure, in order to prolong their campaign for the foreseeable future. As stated above, Solarmarker's operators have already shown a willingness and ability to update their tactics, techniques, and procedures (TTPs) based on points of exposure in previous iterations.

Coverage

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

Product	Protection
Cisco Secure Endpoint (AMP for Endpoints)	✓
Cloudlock	N/A
Cloud Web Security	✓
Cisco Secure Email	N/A
Cisco Secure Firewall/Secure IPS (Network Security)	✓
Cisco Secure Network Analytics (Stealthwatch)	N/A
Cisco Secure Cloud Analytics (Stealthwatch Cloud)	N/A
Cisco Secure Malware Analytics (Threat Grid)	✓
Umbrella	✓
Cisco Secure Web Appliance (Web Security Appliance)	✓

Cisco Secure Endpoint is ideally suited to prevent the execution of the malware detailed in this post. New users can try Cisco Secure Endpoint for free here.

Cisco Secure Web Appliance web scanning prevents access to malicious websites and detects malware used in these attacks.

Cisco Secure Firewall and Meraki MX can detect malicious activity associated with this threat.

Cisco Secure Malware Analytics helps identify malicious binaries and build protection into all Cisco Security products.

Cisco 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.

Additional protections with context to your specific environment and threat data are available from the Cisco Secure Firewall Management Center.

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

The following ClamAV signatures have been released to detect this threat as well as tools and malware related to these campaigns:

Win.Trojan.Solarmarker-9832983-0
Win.Dropper.SolarMarker-9867952-0
Win.Trojan.Jupiter-9858780-0

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

The following SIDs have been released to detect this threat: 57973-57974

Cisco Secure Endpoint (AMP) users can use Orbital Advanced Search to run complex OSqueries to see if their endpoints are infected with this specific threat. For specific OSqueries on this threat, click here

IOCs IPs:

45.135.232[.]131
46.102.152[.]102

Hostnames:

hxxp://on-offtrack[.]biz/get/uran.ps1
hxxp://spacetruck[.]biz:82
hxxp://vincentolife[.]com

SHA256:

99e5a83a5f8e9fb7850922aed9a44bbc2b2a03f5fdda215db4d375bd1952536d - parent dropper
3b79aab07b9461a9d4f3c579555ee024888abcda4f5cc23eac5236a56bf740c7 - parent dropper
77fa1b6fc7f192b0c983d1f8ecc73effae4f688a49439a7df27e76cfba870d23 - parent dropper
032b09bbf1c63afc06afb011d69bafc096d7d925d99e24e3785db5a2957358ec - parent dropper
a871b7708b7dc1eb6fd959946a882a5af7dafc5ac135ac840cfbb60816024933 - parent dropper
187e204c5c30b9b56ccc82df510c4c215cdfd37b475d1edba9a0631a4d82ae2e - parent dropper
1252f2bf3817714a8303f7e448930d6d4f797a70ec7effc80f2b1db5e49b9077 - parent dropper
7240e51c027e498e71da4006e261409fcf3a5cf5b912d493a9da91199c840059 - parent dropper
3039dddbcfb26326ba2e8692316ca674753fa6e1d141c887dc1e9f1ae546ca17 - "pk.exe" second stage dropper
056f373077ca5b6a070975b22839d6f427cbcaeaec4dc31df86231cd3757f7e3 - Mars DLL
10fc8f8cf1b45a6a6b2b929414a84fc513f80d31b988c3d70f9a21968e943bf2 - Mars DLL
1bd39e3ee0d59ac691cc644c026b6c5bb3d0713f6e4e2136b94f161558a4444f - Mars DLL
e7d165f3728b96921b43984733a92a51148ec87aec900c519a547c470e2a12d9 - Mars DLL
eebaec0f28addac39549ebf9be659e105046b1c9801381d4c4dd07cfc74d25d6 - Mars DLL
870f691ec9a83e9c4acce142e0acbf110260e6c8e707410c23c02076244f3973 - Mars DLL
573631c92d34e7ca2dfc1a590606bd66194d517f8d97dd319169262ac46b3e2e - Mars DLL construction PS1 script
b29e6b570a96224c874b9fae36b65db9e9b8c6e1081f18cf87e56bd996470c22 - Mars DLL construction PS1 script
0a11002a0f390fb6e4933fead0521032f8a9ede7ced27f75011d8a30c5772376 - Uran DLL
721bb4554f8e6615dd9e10e28765cf42882c00e1d04f808d8f766b51fdf4d7e6 - Uran construction PS1 script

83b50b18ebfa4402b2c0d2d166565ee90202f080d903fd15cccd1312446a636e - "CMmnnjAi1984unbd.exe" and "gkNjg0918jkd.exe" decoy program "PDFSam"
