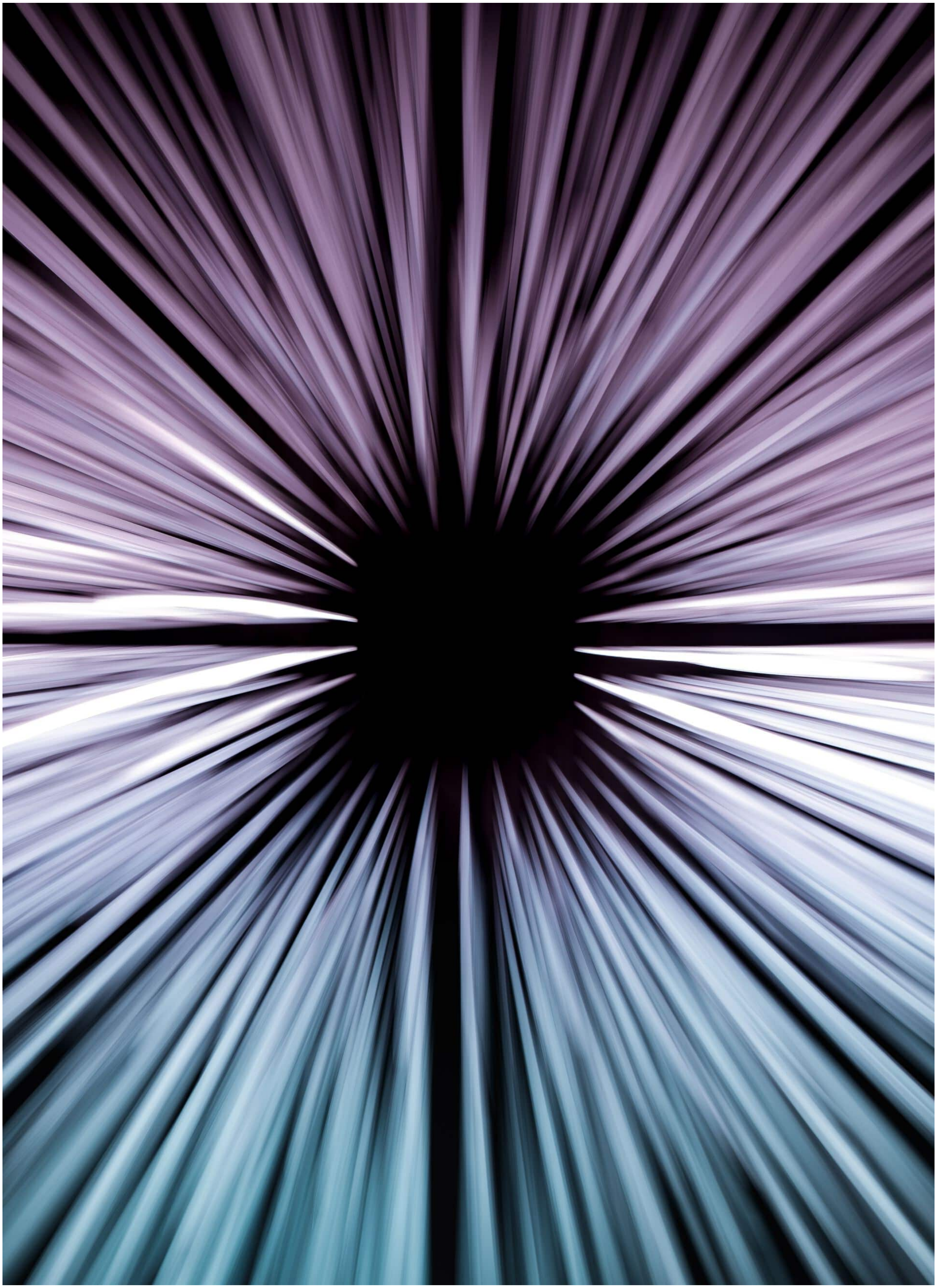
Detecting Sodin

hatching.io/blog/ransomware-part2



2019-12-18

- triage
- ransomware

Written by
Pete Cowman

Sodin - also known as Sodinokibi or REvil - is a successful ransomware family which often employs advanced evasion techniques to avoid notice until the right time. It is developed and operated as ransomware-as-a-service (RaaS), meaning that threat actors can pay to make use of the software to run their campaigns.

Active from early 2019, Sodin has rapidly become a dominant force in ransomware activity, quickly filling the gap left by the end of Gandcrab being available as a service. There are many good writeups of the Sodin family available online, and in this blog post, we are not doing a full analysis of the sample. Instead, we will breakdown the main ways in which Sodin is detectable and identifiable within a dynamic sandbox environment, aiming to give examples of some of the techniques covered in part 1 of this miniseries.

Analyzed Samples

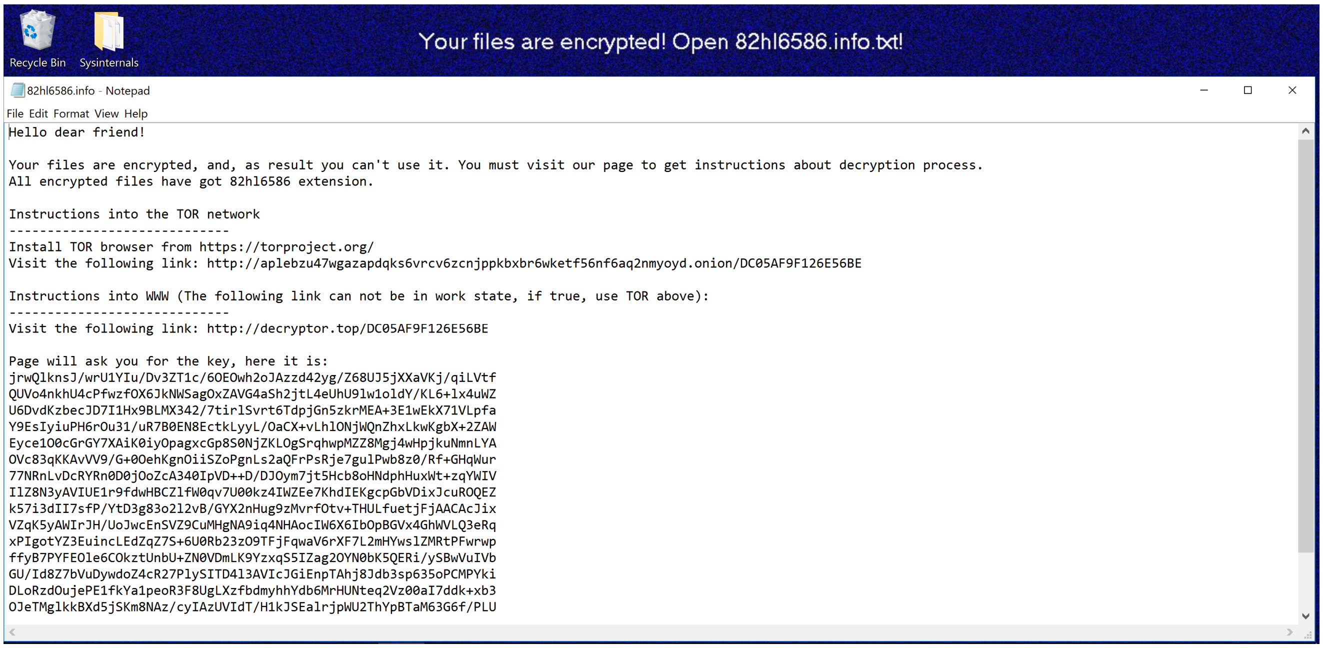
SHA256	tria.ge Analysis
e5d23a3bb61b99e227bb8cbfc0e7f1e40fe a34aac4dcb80acc925cfd7e3d18ec	https://tria.ge/reports/191216-pg5st7zccj/
06b323e0b626dc4f051596a39f52c46b35f 88ea6f85a56de0fd76ec73c7f3851	https://tria.ge/reports/191216-4zdx1n374x/
0fa207940ea53e2b54a2b769d8ab033a6b2 c5e08c78bf4d7dade79849960b54d	https://tria.ge/reports/191216-8bfljdyw2s/
139a7d6656feebe539b2cb94b0729602f62 18f54fb5b7531b58cfe040f180548	https://tria.ge/reports/191216-4rcmytrrka/

Ransomware Information

Ransomware is unusual among malware as it has no interest in hiding its infection from the user once it has carried out its task. Sodin is no exception to this, dropping ransom notes to the directories in which it has encrypted files and changing the wallpaper to point the victim towards these files.

Ransom Note

We took a look at ransom notes in the last blog post in this series so we won’t go over the details again here, but in summary, they contain instructions for the victim on how to pay the ransom and decrypt their files.




```
Hello dear friend!

Your files are encrypted, and, as result you can't use it. You must visit our page to get instructions about decryption process.
All encrypted files have got 82hl6586 extension.

Instructions into the TOR network
-----------------------------
Install TOR browser from https://torproject.org/
Visit the following link: http://aplebzu47wgazapdqks6vrcv6zcnjppkbxbr6wketf56nf6aq2nmyoyd.onion/DC05AF9F126E56BE

Instructions into WWW (The following link can not be in work state, if true, use TOR above):
-----------------------------
Visit the following link: http://decryptor.top/DC05AF9F126E56BE

Page will ask you for the key, here it is:
```
jrwQlknsJ/wrU1YIu/Dv3ZT1c/6OEOwh2oJAzzd42yg/Z68UJ5jXXaVKj/qiLVtf
QUVo4nkhU4cPfwzfOX6JkNWSagOxZAVG4aSh2jtL4eUhU9lw1oldY/KL6+lx4uWZ
U6DvdKzbecJD7I1Hx9BLMX342/7tirlSvrt6TdpjGn5zkrMEA+3E1wEkX71VLpfa
Y9EsIyiuPH6rOu31/uR7B0EN8EctkLyyL/OaCX+vLhlONjWQnZhxLkwKgbX+2ZAW
Eyce1O0cGrGY7XAiK0iyOpagxcGp8S0NjZKLOgSrqhwpMZZ8Mgj4wHpjkuNmnLYA
OVc83qKKAvVV9/G+0OehKgnOiiSZoPgnLs2aQFrPsRje7gulPwb8z0/Rf+GHqWur
77NRnLvDcRYRn0D0jOoZcA340IpVD++D/DJOym7jt5Hcb8oHNdphHuxWt+zqYWIV
IlZ8N3yAVIUE1r9fdwHBCZlfW0qv7U00kz4IWZEe7KhdIEKgcpGbVDixJcuROQEZ
k57i3dII7sfP/YtD3g83o2l2vB/GYX2nHug9zMvrfOtv+THULfuetjFjAACAcJix
VZqK5yAWIrJH/UoJwcEnSVZ9CuMHgNA9iq4NHAocIW6X6IbOpBGVx4GhWVLQ3eRq
xPIgotYZ3EuincLEdZqZ7S+6U0Rb23zO9TFjFqwaV6rXF7L2mHYwslZMRtPFwrwp
ffyB7PYFEOle6COkztUnbU+ZN0VDmLK9YzxqS5IZag2OYN0bK5QERi/ySBwVuIVb
GU/Id8Z7bVuDywdoZ4cR27PlySITD4l3AVIcJGiEnpTAhj8Jdb3sp635oPCMPYki
DLoRzdOujePE1fkYa1peoR3F8UgLXzfbdmyhhYdb6MrHUNteq2Vz00aI7ddk+xb3
OJeTMglkkBXd5jSKm8NAz/cyIAzUVIdT/H1kJSEalrjpWU2ThYpBTaM63G6f/PLU

Example ransom note created by Sodin infection

In the case of the Sodin family, this note contains several URLs which the victim is meant to visit to receive the instructions and see how much is being demanded in ransom. No bitcoin address or information regarding the ransom is available directly in this file, which is unfortunate (extracting these can be useful for tracking campaigns/threat actors), but these URLs are still worth extracting for further investigation.

The name of the ransom note is defined by the `nname` field within the configuration discussed later in this blog post and will include the file extension given to all encrypted files. For example, where `{"EXT"}` is replaced by the extension `{"EXT"}.info.txt` .

Analyzing Ransom Notes via tria.ge

The tria.ge sandbox includes detections for many aspects of ransomware but there will always be data we don’t include in the final report for one reason or another. However, our kernel driver maintains a record of almost all activity on the VM which can be accessed directly via the Triage API, enabling users to run custom parsers/extractors over the data. In this section, we will do a quick example on fetching and processing this information to extract basic information from a Sodin ransom note - although many other uses are also possible.

Documentation for the API can be found at https://tria.ge/docs/.

Onemon, the kernel driver used in Triage, records a log of system events for each task within an analysis (a 'task' is a specific VM instance run during analysis - for example a sample run on Windows 7 and Windows 10 will have 2 tasks, 1 for each VM). This data can be downloaded in JSON format via the following command

curl -H 'Authorization: Bearer <YOUR_API_KEY>' 'https://api.tria.ge/v0/samples/{sampleID}/{taskID}/logs/onemon.json'

Note that you must pass your API key to access Triage endpoints. This can be found on your [account page][acount].

The `taskID` field should simply be replaced by a string like `task1` referencing the exact analysis you would like to access - for reference, the taskID value can be seen in the final part of the URL when viewing a report online.

As an example, requesting the `onemon.json` file for the first sample linked in this blogpost looks as follows:

```
curl -H 'Authorization: Bearer <YOUR_API_KEY>' \
  'https://api.tria.ge/v0/samples/191216-pg5st7zccj/task1/logs/onemon.json'
```

The file received from the API contains 1 JSON object per line, each representing an event within the system. The results below have been prettified for readability.

```
{
    "kind": "onemon.File",
    "event": {
        "dstpath": "",
        "flags": "NoFileFlags",
        "id": 844424930209781,
        "kind": "CreateModify",
        "pid": 1444,
        "srcpath": "C:\\Users\\Admin\\AppData\\Local\\Temp\\st748h0795z.bmp",
        "status": 0,
        "ts": 36769
    }
}
{
    "kind": "onemon.Registry",
    "event": {
        "kind": "SetValueKeyStr",
        "path": "\\REGISTRY\\USER\\S-1-5-21-1774239815-1814403401-2200974991-
1000\\Control Panel\\Desktop\\Wallpaper",
        "pid": 1444,
        "status": 0,
        "ts": 36769,
        "valued": null,
        "valuei": 0,
        "values": "C:\\Users\\Admin\\AppData\\Local\\Temp\\st748h0795z.bmp"
    }
}
{
    "kind": "onemon.NetworkFlow",
    "event": {
        "dstip": 134744072,
        "dstport": 53,
        "pid": 284,
        "proto": 17,
        "srcip": 285214474,
        "srcport": 60531,
        "ts": 36894
    }
}
```

Example events in onemon.json

Each line includes a 'kind' tag which defines the structure contained within the following 'event' tag. There are many different 'kind' definitions, but likely the main ones which will be of interest are `onemon.Registry`, `onemon.Process`, and `onemon.File`. We can combine the 'kind' tag and the 'status' tag within the event structure to form filters using grep, e.g., to show only events where a write operation took place in the registry, we could use `grep onemon.Registry | grep SetValueKeyStr`.

In this example, we are interested in the ransom notes dropped by Sodin during analysis. By default, Triage dumps the contents of all .txt files created by a sample so that automated processing can be carried out on the files, e.g., to extract URLs. These dumps are (currently) stored as FileContents blocks within the onemon log:


```
{
    "kind": "onemon.FileContents",
    "event": {
        "buf":
"SABlAGwAbABvACAAZABlAGEAcgAgAGYAcgBpAGUAbgBkACEADQAKAA0ACgBZAG8AdQByACAAZgBpAGwAZQBzA

        "id": 844424930209780,
        "pid": 1444,
        "ts": 33462
    }
}
```


Sodin ransom note in onemon.json

The `onemon.FileContents` blocks are linked to standard `onemon.File` events where the sample performed a CreateFile operation for a .txt file. These blocks contain metadata useful for relating the file dumps to their original file names and paths, the ID value shown above acts as the reference. Note the `"flags": "DumpContents"` field in this event type


```
{
    "kind": "onemon.File",
    "event": {
        "dstpath": "",
        "flags": "DumpContents",
        "id": 844424930209780,
        "kind": "CreateModify",
        "pid": 1444,
        "srcpath": "C:\\Users\\Public\\Videos\\Sample Videos\\43s40i71l.info.txt",
        "status": 0,
        "ts": 33462
    }
}
```


If we are looking for a specific file, we can grep through the json for the file name and then pick out the specific FileContents block that we were looking for based on the ID value.

The `"buf"` section of the FileContents block shown above is a base64-encoded representation of the data written to the file, extracted by intercepting the API call for the file write and dumping the buffer contents. In order to retrieve the original contents, simply decode the value.

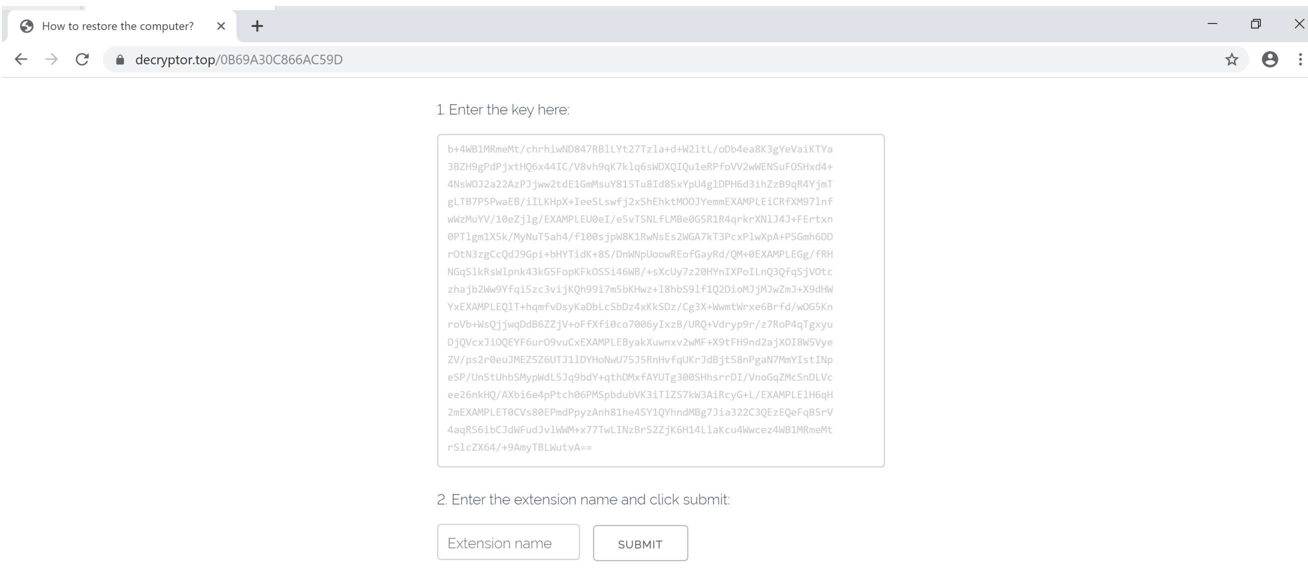
At this point the entire contents of the file are available in plaintext for any further processing desired - extracting contact URLs/email addresses, bitcoin wallets, personal identifier codes etc.

We have produced a simple Python script which, when passed a Triage analysis ID and your API key, will perform the process outlined above and dump every .txt file found in the onemon.json to the current directory. It will generate a number of `.txt` files. You can find the script here. To use it, you'll need the Triage ID, the task ID of the analysis you want to examine, and your API key for tria.ge. Usage may look as follows:

```
python3 triage-ransomnote.py <API-Key> 191216-4rcmytrrka task1
```

Ransom Portal Overview

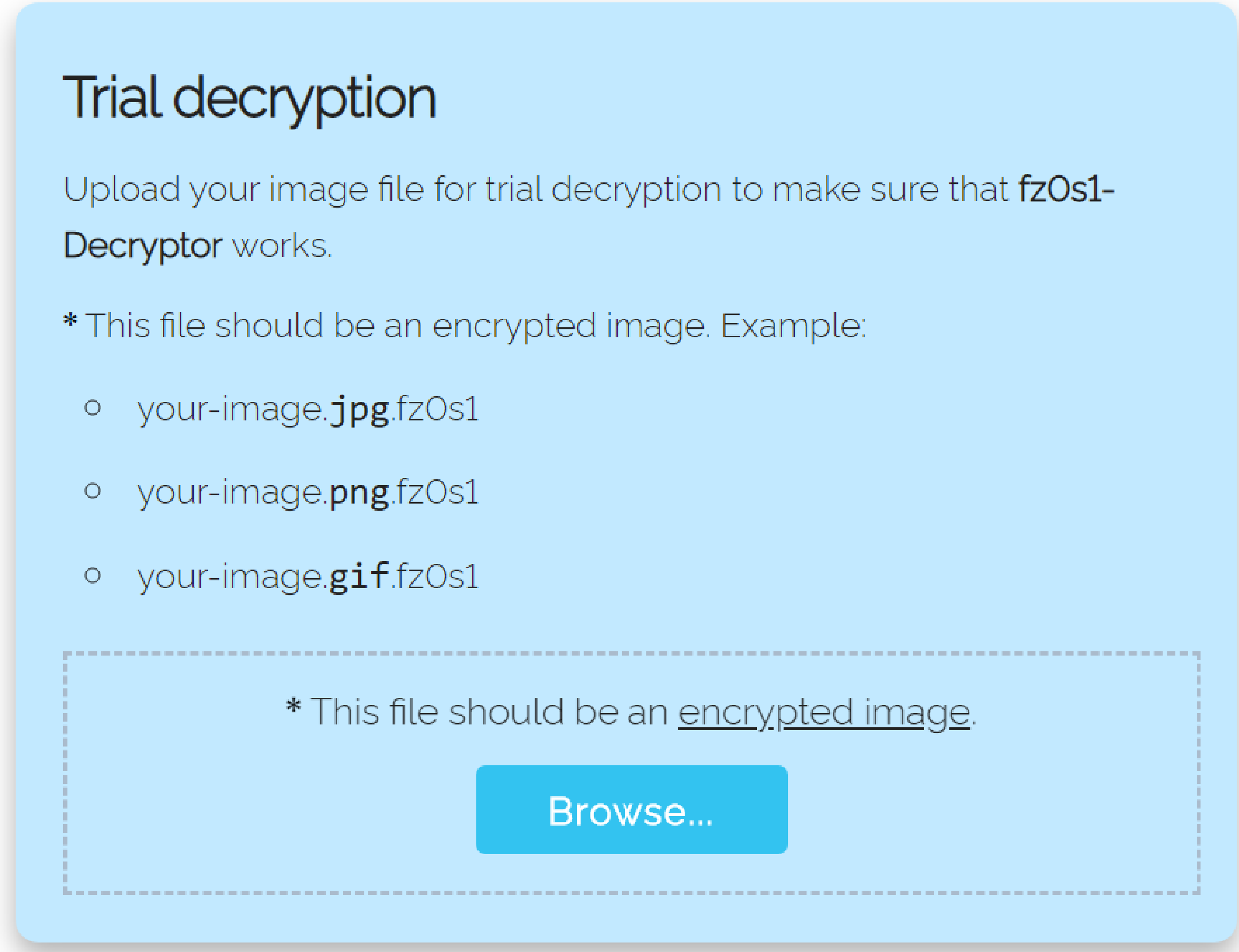
The URLs contained within the ransom note lead to a web portal customized to the victim based on a generated ID value, which is the final part of the address (see ransom note above). To make it harder for analysts to use automated tools to gather information, victims must enter some basic information when accessing their portal - a generated 'key' visible in the ransom note and the extension used when encrypting files (this can differ based on the configuration settings).



Ransom portal landing page

After submitting this information, the victim can then access payment details and other information, including guides on using Bitcoin; a trial decryptor that can be used on a single file; and even a chat support feature to get assistance from the malware operators.

Main elements of Sodin ransom portal



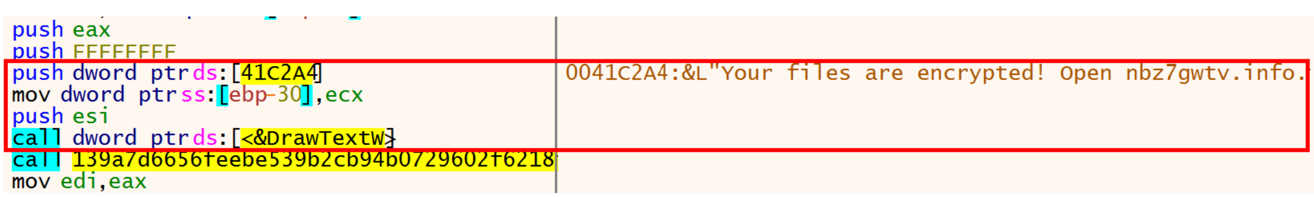
Main elements of Sodin ransom portal

The most useful information on this page is the ransom price and the Bitcoin address which is to be used for payment - using a framework like Selenium it would be possible to automate the gathering of these addresses for tracking.

Desktop Wallpaper Change

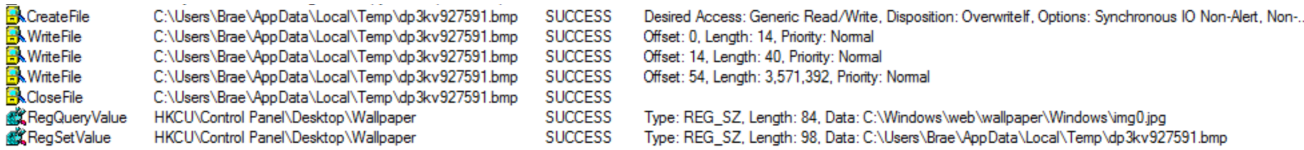
To make sure the victim is aware of the infection, Sodin also generates a new desktop background image and makes it active via the registry.

The image is a bitmap created through the DrawTextW function in User32.dll. The text to be written is defined within the malware's JSON config section (discussed later in this post).



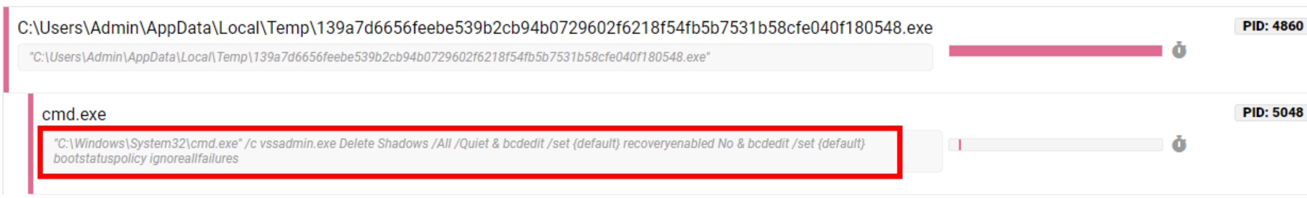
Sample passes string into DrawTextW to create background image

The image is saved to the user's Local AppData directory with a random file name, and the registry key `HKCU\\Control Panel\\Desktop\\Wallpaper` is set to point to the new file.



Preventing System Recovery

Like many other ransomware families, Sodin attempts to make recovery of the infected machine more difficult by disabling or removing some Windows features. It achieves this using common `vssadmin` and `bcdedit` commands.



Brief descriptions of these commands are provided below for reference. In a dynamic analysis environment, these are reasonably clear indicators of maliciousness, as there are few legitimate reasons for an application to delete backups or interfere with boot settings.

Command	Description
vssadmin Delete Shadows /All /Quiet	Delete system shadow copies
bcdedit /set {default} recoveryenabled No	Disables Windows Error Recovery on startup
bcdedit /set {default} bootstatuspolicy ignoreallfailures	Sets system to ignore errors and boot as normal (NB: this is also the default Windows setting)

Configuration

To enable threat actors to customize their Sodin campaign, the family includes a configuration file embedded within the executable. This is packaged as a PE section with a distinct name - the 2 variants we have examined for this blogpost used `.grrr` and `.zeacl`.

```
Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0000a154  040a1000  040a1000  00001000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .rdata        0000f658  040ac000  040ac000  0000c000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  2 .data         0000171c  040bc000  040bc000  0001c000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  3 .zeacl        0000c800  040be000  040be000  0001e000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  4 .reloc        00002000  040cb000  040cb000  0002b000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00009974  00401000  00401000  00000400  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .rdata        0000f760  0040b000  0040b000  00009e00  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  2 .data         00001200  0041b000  0041b000  00019600  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  3 .grrr         0000c800  0041d000  0041d000  0001a800  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  4 .reloc        0000050c  0042a000  0042a000  00027000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
```

PE sections from unpacked Sodin samples

These sections are a JSON configuration file encrypted using RC4 which contain a large amount of information about the particular campaign the sample belongs to

```
{
    "pk": "GadtWz2QBTacskL+55Wpo65IkwY28qJOxHoe4Xte81M=",
    "pid": "10",
    "sub": "7",
    "dbg": false,
    "fast": true,
    "wipe": true,
    "wht": {
        "fld": ["appdata", "google", "msocache", "mozilla", "program files",
"windows", "perflogs", "application data", "windows.old", "system volume
information", "program files (x86)", "$windows.~ws", "intel", "$recycle.bin",
"$windows.~bt", "programdata", "boot", "tor browser"],
        "fls": ["ntuser.dat.log", "bootsect.bak", "ntuser.dat", "iconcache.db",
"ntldr", "autorun.inf", "boot.ini", "bootfont.bin", "desktop.ini", "thumbs.db",
"ntuser.ini"],
        "ext": ["ldf", "msi", "nomedia", "msu", "wpx", "ani", "shs", "theme", "386",
"adv", "icns", "lnk", "ico", "ics", "rom", "sys", "mod", "cur", "com", "scr", "cpl",
"diagcfg", "lock", "diagcab", "msstyles", "idx", "msc", "icl", "rtp", "exe", "drv",
"hta", "nls", "deskthemepack", "cmd", "hlp", "themepack", "dll", "mpa", "msp", "ps1",
"prf", "ocx", "bat", "diagpkg", "cab", "bin", "spl", "key"]
    },
    "wfld": ["backup"],
    "prc": ["mysql.exe"],
    "dmn": "lyricalduniya.com;theboardroomafrica.com;chris-
anne.com;ownidentity.com;web865.com;[...]",
    "net":true,
    "nbody
":"SABlAGwAbABvACAAZABlAGEAcgAgAGYAcgBpAGUAbgBkACEADQAKAA0ACgBZAG8AdQByACAAZgBpAGwAZQB

    "nname": {"EXT"}.info.txt,
    "exp":false,

"img":"WQBvAHUAcgAgAGYAaQBsAGUAcwAgAGEAcgBlACAAZQBuAGMAcgB5AHAAdABlAGQAIQAgAE8AcABlAG4

}
```


There are quite a few useful fields in here - these are outlined in the table below.

Field Name	Description
pk	Base64-encoded public key used for file encryption
pid	Only used if `net` field is also set, sent to C2 servers. Likely related to campaign identifier etc.
sub	See `pid`
dbg	Enable/disable debug mode (for the malware author)
fast	Boolean value which changes how large files are encrypted

Field Name	Description
wipe	Boolean. If set, sample will try to erase contents of folders blacklisted in the `wfld` field
wht	Defines whitelists for encryption process. Contains 3 sections:
	1. `fld` : folders to ignore
	2. `fls` : specific files to ignore
	3. `ext` : whitelisted file extensions
wfld	List of folder names. If the `wipe` field is set to true then the malware will attempt to erase the contents of all folders
prc	List of process names the malware will try to terminate before carrying out file encryption
dmn	List of domain names which the malware will attempt to contact to use as C2
net	Boolean value. Sets whether sample should send host information to C2 servers listed in `dmn` key
nbody	Base64-encoded version of the ransom note dropped to the file system after encryption
nname	Filename of the ransom note
exp	Boolean value. Defines whether or not the malware will use an exploit to try and escalate privileges on the system
img	Base64-encoded version of the text shown in the wallpaper background set by the malware.

Configuration field details

When delivered to a new system Sodin samples are packed using a custom algorithm, hiding the existence of the specially named resource sections. The malware uses a PE overwrite approach to its unpacking mechanism - it allocates heap space using LocalAlloc, writes the unpacker stub to it, and then passes execution to that area after marking it RWX with VirtualProtect.

```
call    ds:VirtualProtect
push    offset unk_465398
mov     edx, dwSize
push    edx
mov     eax, lpAddress
push    eax
call    sub_401064
add     lpAddress, 21A5h
call    lpAddress
pop     ebx
mov     esp, ebp
pop     ebp
retn
```

Passes execution to unpacker

Create and call unpacker stub

The unpacker stub then writes the new PE to the address space in which the original file was mapped, clearing the content of the entire region before writing the new executable and jumping back to the updated Entry Point. We can now dump the executable and examine the sections, as shown previously.

The configuration itself is still encrypted at this stage, but in its unpacked form it is possible to analyze the decryption process and recreate it. The algorithm is RC4 - we can clearly see the SBox creation and swapping operations

> but some minor changes have been made to prevent easy decryption with standard RC4 tools.

Examining the executable we can see that the key for the decryption is the first 32 bytes of the resource section.

```
loc_4019FB:                             ; CODE XREF: sub_4019D8+1D↑j
                push    esi
                push    ds:dword_41D024
                call    sub_40352C
                mov     esi, eax
                pop     ecx
                test    esi, esi
                jz      short loc_401A27
                push    esi
                push    ds:dword_41D024
                push    edi
                push    20h             ; keylen 32
                push    offset config_crypted ; .grrr resource section
                call    DecryptConfig
                add     esp, 14h
                mov     eax, esi
```

Preparing keylength and ciphertext parameters and passing them to the configuration decryption function

With this information and analysis of the decryption function, it was possible to build a configuration extractor to parse and report details from these configurations during dynamic analysis of the samples. In Hatching Triage, we have implemented a system that enables taking process memory dumps when particular conditions are met during an analysis. Using this, we can obtain the unpacked executable and run processing on it to include the information in the analysis report and enable easy identification of particular campaigns or actors.

The Sodin extractor is not available on Triage yet while testing and improvements to the memory dumping methodology are implemented, but keep an eye on our Twitter account for updates when that is released.

File Encryption

Sodin can encrypt files on local storage or any mapped network shares, overwriting the original files and renaming them with an extension generated on a per-infection basis. This process is highly customizable through the embedded configuration section, allowing for certain files/folders to be whitelisted and protected from the encryption process. The Sodin executable also accepts a command-line parameter - by passing the value `-nolan`, one can disable the encryption of mapped network shares and limit the effects to only the infected machine.

```
"wht": {
    "fld": ["appdata", "google", "msocache", "mozilla", "program files", "windows",
"perflogs", "application data", "windows.old", "system volume information", "program
files (x86)", "$windows.~ws", "intel", "$recycle.bin", "$windows.~bt", "programdata",
"boot", "tor browser"],
    "fls": ["ntuser.dat.log", "bootsect.bak", "ntuser.dat", "iconcache.db", "ntldr",
"autorun.inf", "boot.ini", "bootfont.bin", "desktop.ini", "thumbs.db", "ntuser.ini"],
    "ext": ["ldf", "msi", "nomedia", "msu", "wpx", "ani", "shs", "theme", "386",
"adv", "icns", "lnk", "ico", "ics", "rom", "sys", "mod", "cur", "com", "scr", "cpl",
"diagcfg", "lock", "diagcab", "msstyles", "idx", "msc", "icl", "rtp", "exe", "drv",
"hta", "nls", "deskthemepack", "cmd", "hlp", "themepack", "dll", "mpa", "msp", "ps1",
"prf", "ocx", "bat", "diagpkg", "cab", "bin", "spl", "key"]
}
```


Whitelist configuration section

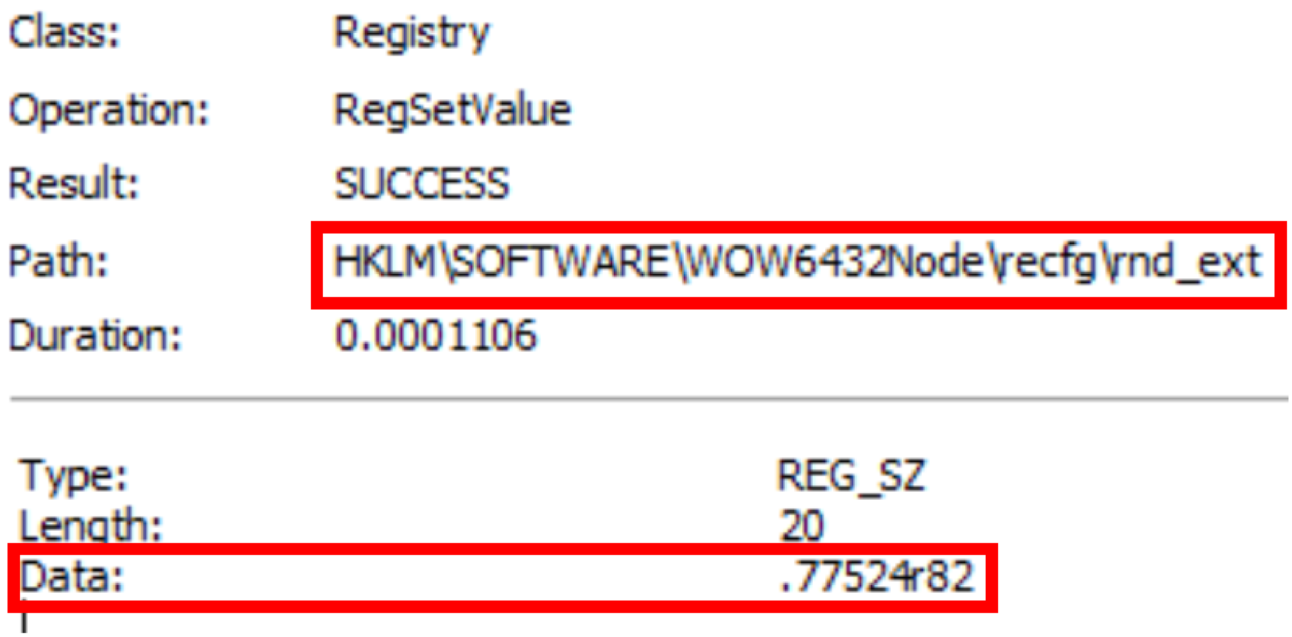
The keys specify whitelists for:

- `fld` - folder names
- `fls` - specific file names
- `ext` - file extensions

The malware iterates through every directory and file on the system, checking them against these configuration values and queuing them up for encryption if they are not excluded. Once encrypted, the files are renamed with a new extension.

Encrypted File Extension

Before starting the encryption process, Sodin generates the random file extension which is applied to every encrypted file. The extension is a string of letters and numbers from 5 to 10 characters long, which is generated and saved to the registry along with other information gathered at various stages (discussed below).

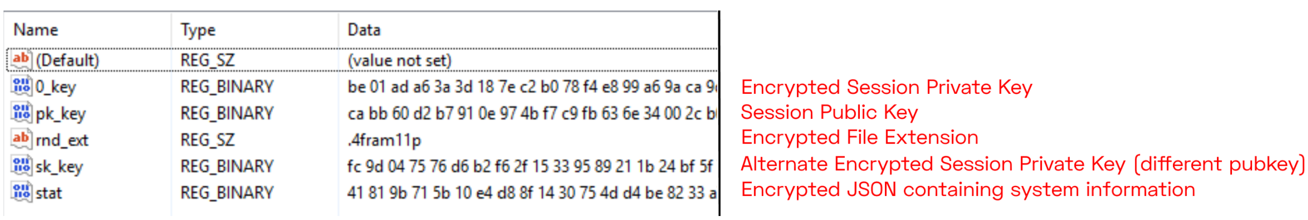


File extension saved to SOFTWARE\\WOW6432Node\\recfg\\rnd_ext

The extension itself is tricky to accurately identify as Sodin-related due to the generic nature of it, but the registry path is relatively specific and is readily detectable in a sandbox.

Other Registry Changes

As well as the file extension, Sodin saves a few other bits of data to the `Software\\Wow6432Node\\recfg registry key`.

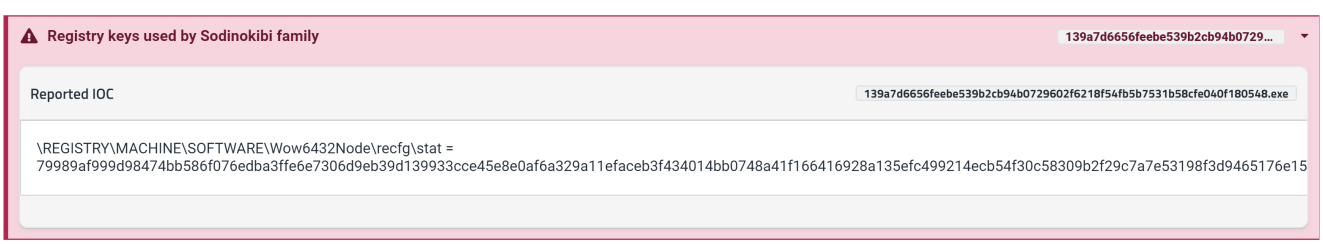


Name	Type	Data	
(Default)	REG_SZ	(value not set)	
0_key	REG_BINARY	be 01 ad a6 3a 3d 18 7e c2 b0 78 f4 e8 99 a6 9a ca 9	Encrypted Session Private Key
pk_key	REG_BINARY	ca bb 60 d2 b7 91 0e 97 4b f7 c9 fb 63 6e 34 00 2c b	Session Public Key
rnd_ext	REG_SZ	.4fram11p	Encrypted File Extension
sk_key	REG_BINARY	fc 9d 04 75 76 d6 b2 f6 2f 15 33 95 89 21 1b 24 bf 5f	Alternate Encrypted Session Private Key (different pubkey)
stat	REG_BINARY	41 81 9b 71 5b 10 e4 d8 8f 14 30 75 4d d4 be 82 33 a	Encrypted JSON containing system information

Values within the recfg registry key

None of the values assigned to these keys are static, but all of the key names are common across Sodin samples. We won't go into the details of these here, but the image above gives a basic outline of the contents.

In a dynamic environment, this sort of registry structure is ideal for identifying a family.



Tria.ge signature output for registry keys

Conclusion

Sodin is a complex family with far more functionality than we have covered here, but this has outlined the main indicators which are useful for identifying samples within a dynamic environment like Triage. References to a number of samples used as source material for this post can be found in the header at the top of the page.

We'll be continuing to expand coverage over the coming weeks for ransomware families with ransom note and configuration extractors. In the next post in this mini-series we'll cover another family which poses some different challenges to analysis and present ways to solve them.

Until next time, Happy Holidays!