

Malware Tales: Sodinokibi

Archived: 2026-04-05 22:22:06 UTC

Hi everyone! Today we are looking at a threat that appeared recently: a new ransomware called Sodinokibi.

Summary

1. *The Threat*
2. *The Loader*
3. *Mutex and Configuration*
4. *Machine information recovery*
5. *Encryption preparation inspired by GandCrab*
6. *Ransomware attack*
7. *C2 Registration*
8. *Conclusion*

1. The Threat

The first noteworthy appearance was at the end of April ([Talos Research](#)).

Then, at the start of this month, we gathered different reports of this threat being spread in Italy (eg: [JAMESWT MHT's tweet](#)), both via malspam and known server vulnerabilities.

Also, there was the announcement of the shutdown of the GandCrab Operation [Bleeping Computer](#), just some days earlier.

Coincidence? We'll see.

Our guess is that this new payload could be used as a replacement of **GandCrab** in the RAAS (Ransomware-as-a-service) panorama.

Therefore, in order to protect our customers effectively, we went deep into the analysis of this ransomware.

Mainly we analyzed two different samples:

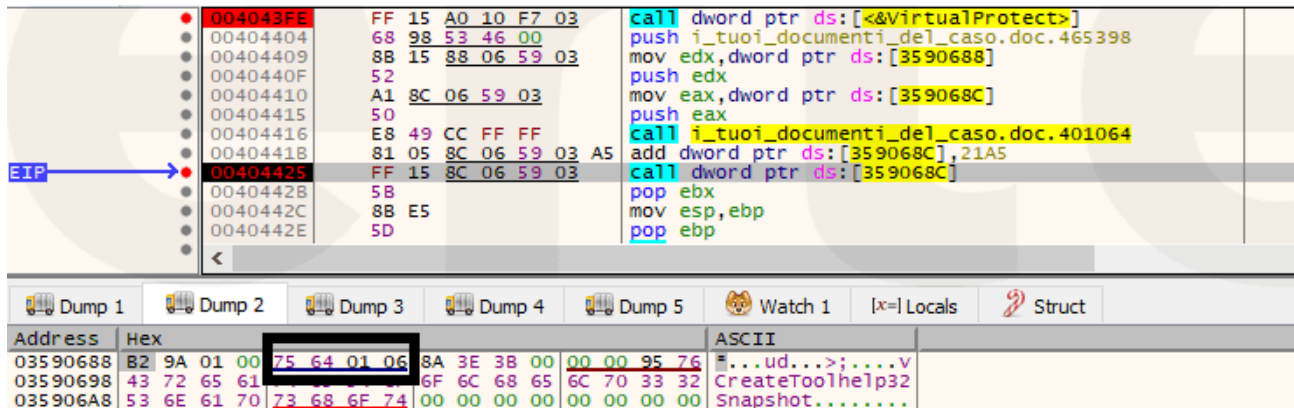
- version 1.01: md5: e713658b666ff04c9863ebecb458f174
- version 1.00: md5: bf9359046c4f5c24de0a9de28bbabd14

2. The Loader

Like every malware who deserves respect, **Sodinokibi** is protected by a custom packer that is different for each sample.

The method used by the version 1.01 sample to reconstruct the original payload is called “*PE overwrite*”.

To perform this technique, the malicious software must allocate a new area inside its process memory and fill it with the code that has the duty to overwrite the mapped image of the original file with the real malware payload. In this case, first the process allocates space in the Heap via LocalAlloc, then it writes the “*unpacking stub*” code, it signs that space as executable with VirtualProtect and finally it redirects the execution flow to the new memory space



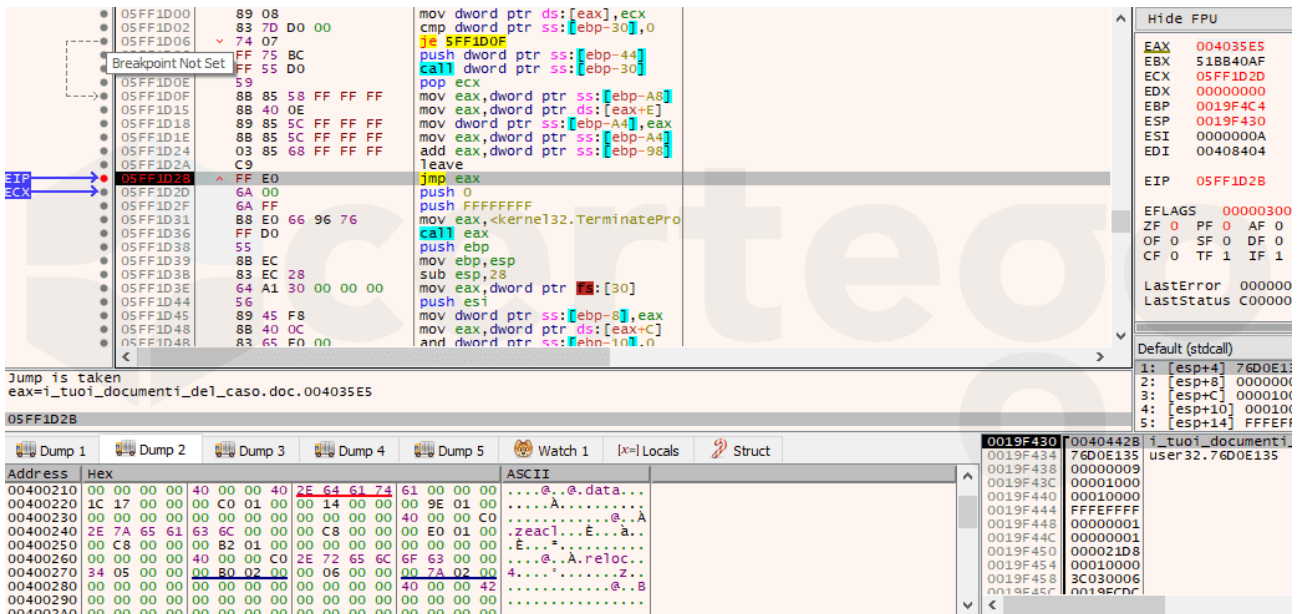
In order to slow the analysis, the loader contains a lot of junk code that will be never executed.



Also, in the following image, we can see that it tries to hide some important strings from the static analysis like “kernel32.dll”. It leverages “*stack strings*” plus the randomization of the order of the characters.

At this point, the unpacking stub resolves dynamically the functions that he needs like VirtualAlloc. Then it performs the overwrite of the original image base with the new decrypted payload.

Finally, it transfers the execution to the OEP (Original Entry Point) of the unpacked Sodinokibi payload.



3. Mutex and Configuration

Once unpacked, the sample tries to create a **mutex** object. It calls `CreateMutexW`, then, if there was an error, with `RtlGetLastWin32Error` it would extract the generated error. Indeed, if the mutex already existed, the error would have been “0xB7” (“`ERROR_ALREADY_EXISTS`” ref [docs](#)). In that case a function is called that terminates the process.

We found that the mutex name is different for each sample but following this pattern: “**Global{UUID}**”. Therefore it’s a method to detect the malware or to vaccinate the endpoint ([Zeltser blog](#)) that is reliable only for a specific sample.

Going forward, we found the configuration in an encrypted form in the section “**.zeacl**” for v.1.01 or “**.grrr**” for v.1.00. Once extracted, we noticed that it’s a JSON file.

These are the keys found in the configuration.

- “**pk**” -> base64 encoded key used to encrypt files
- “**pid**” -> personal id of the actor
- “**sub**” -> another id, maybe related to the specific campaign
- “**dbg**” -> debug mode
- “**fast**” -> fast mode
- “**wipe**” -> enable wipe of specific directories
- “**wht**” -> whitelist dictionary
- “**fld**” -> keyword in whitelisted directories
- “**fls**” -> whitelisted filenames
- “**ext**” -> whitelisted file extensions
- “**wfld**” -> directories to wipe
- “**prc**” -> processes to kill before the encryption
- “**dmn**” -> domains to contact after encryption

- “**net**” -> check network resources
- “**nbody**” -> base64 encoded ransom note body
- “**nname**” -> ransom note file name
- “**exp**” -> unknown, expert mode?
- “**img**” -> base64 encoded message on desktop background

If you are interested in manually checking the configuration files we have extracted in the samples we have analyzed, follow this link and download the archive (password:sodinokibi): [sodinokibi config files.zip](#)

4.Machine information recovery

Afterwards, Sodinokibi starts to gather information about the infected machine and builds another JSON structure that stores in an encrypted form in the “HKEY_LOCAL_MACHINE\SOFTWARE\recfg\stat” registry key.

Keys:

- “**ver**”: version (100 or 101)
- “**pid**”: previous config “pid”
- “**sub**”: previous config “sub”
- “**pk**”: previous config “pk”
- “**uid**”: user ID. It’s a 8 byte hexadecimal value generated with XOR encryption. First 4 bytes are created from the processor name, while the others are created from the volume serial number extracted with a “GetVolumeInformationW” API call.

- “**sk**”: secondary key, base64 encoded key generated at runtime
- “**unm**”: username
- “**net**”: hostname
- “**grp**”: windows domain

004056DC	55	push ebp	xor encryption
004056DD	8B EC	mov ebp,esp	
004056DE	8B 4D 08	mov ecx,dword ptr ss:[ebp+8]	
004056E2	8B 55 10	mov edx,dword ptr ss:[ebp+10]	
004056E5	F7 D1	not ecx	
004056E7	85 D2	test edx,edx	
004056E9	74 28	jne i_tuo1_documenti_de1_caso.doc.405716	
004056EB	56	push esi	
004056EC	8B 75 0C	mov esi,dword ptr ss:[ebp+C]	[ebp+C]:"Intel(R) Core(TM) i5-4200M CPU @ 2.50G
004056EF	57	push edi	
004056F0	0F B6 06	movzx eax,byte ptr ds:[esi]	
004056F3	4A	dec edx	
004056F4	6A 08	push 8	
004056F6	33 C8	xor ecx,eax	
004056F8	46	inc esi	
004056F9	5F	pop edi	
004056FA	8B C1	mov eax,ecx	
004056FC	D1 E9	shr ecx,1	
004056FE	83 E0 01	and eax,1	
00405701	F7 D0	not eax	
00405703	40	inc eax	
00405704	25 20 83 B8 ED	and eax,EDB88320	
00405709	33 C8	xor ecx,eax	
0040570B	83 EF 01	sub edi,1	
0040570E	75 EA	jne i_tuo1_documenti_de1_caso.doc.4056FA	
00405710	85 D2	test edx,edx	
00405712	75 DC	jne i_tuo1_documenti_de1_caso.doc.4056F0	
00405714	5F	pop edi	
00405715	5E	pop esi	
00405716	F7 D1	not ecx	
00405718	8B C1	mov eax,ecx	
0040571A	5D	pop ebp	
0040571B	C3	ret	

add esp,10		
push eax		eax: L"%08X%08X"
lea eax,dword ptr ss:[ebp-18]		eax: L"%08X%08X"
push eax		
push edi		
call dword ptr ds:[<wsprintfW>]		
add esp,10		
mov eax,edi		eax: L"%08X%08X"
pop esi		
pop edi		
mov esp,ebp		
pop ebp		
ret		

- “Ing”: language

push ebx		
push dword ptr ss:[ebp+C]		[ebp+C]:L"SYSTEM\\CurrentControlSet\\services\\Tcpip\\Parameters
mov esi,ebx		
push dword ptr ss:[ebp+8]		
call dword ptr ds:[<RegOpenKeyExW>]		
test eax,eax		
jne i_tuo1_documenti_de1_caso.doc.4046A9		
push edi		
mov edi,dword ptr ss:[ebp+18]		
push edi		
push ebx		
push dword ptr ss:[ebp+14]		
push ebx		
push dword ptr ss:[ebp+10]		[ebp+10]:L"Domain"
push dword ptr ss:[ebp-4]		
call dword ptr ds:[<RegQueryValueExW>]		

- “bro”: brother? Sodinokibi retrieves the keyboard language with GetKeyboardLayoutList. Then it implements an algorithm that gives “True” as value for this key only if the nation code ends with a byte between 0x18 and 0x2c. It’s not odd that inside this range there are the majority of the East-Europe language codes, like Russian, Cyrillic and Romanian. It’s a clear indication of the origin of the malware authors.

0040463F	6A 01	push 1	
00404641	53	push ebx	
00404642	FF 75 0C	push dword ptr ss:[ebp+C]	[ebp+C]:L"Control Panel\\International"
00404645	8B F3	mov esi,ebx	
00404647	FF 75 08	push dword ptr ss:[ebp+8]	
0040464A	FF 15 E8 CA 41 00	call dword ptr ds:[<RegOpenKeyExW>]	
00404650	85 C0	test eax,eax	
00404652	75 55	jne i_tuo1_documenti_de1_caso.doc.4046A9	
00404654	57	push edi	
00404655	8B 7D 18	mov edi,dword ptr ss:[ebp+18]	
00404658	57	push edi	
00404659	53	push ebx	
0040465A	FF 75 14	push dword ptr ss:[ebp+14]	
0040465D	53	push ebx	
0040465E	FF 75 10	push dword ptr ss:[ebp+10]	[ebp+10]:L"LocaleName"
00404661	FF 75 FC	push dword ptr ss:[ebp-4]	
00404664	FF 15 EC CA 41 00	call dword ptr ds:[<RegQueryValueExW>]	

- “os”: full OS name

```

push dword ptr ss:[ebp+C]
mov esi,ebx
push dword ptr ss:[ebp+8]
call dword ptr ds:[<&RegOpenKeyExW>]
test eax,eax
jne i_tuoi_documenti_dei_caso.doc.4046A9
push edi
mov edi,dword ptr ss:[ebp+18]
push edi
push ebx
push dword ptr ss:[ebp+14]
push ebx
push dword ptr ss:[ebp+10]
push dword ptr ss:[ebp-4]
call dword ptr ds:[<&RegQueryValueExW>]
[ebp+C]:L"SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion"
[ebp+10]:L"productName"

```

- “bit”: Sodinokibi extracts this value from “GetNativeSystemInfo” then it compares with 9 that corresponds to the x64 architecture. Further processing will generate “40” if the architecture is 64bit, “56” otherwise.

```

004043D2 83 EC 24      sub esp,24
004043D5 8D 45 DC      lea eax,dword ptr ss:[ebp-24]
004043D8 50           push eax
004043D9 FF 15 F0 C9 41 00 call dword ptr ds:[<&GetNativeSystemInfo>]
004043DF 33 C0        xor eax,eax
004043E1 66 83 7D DC 09 cmp word ptr ss:[ebp-24],9
004043E6 0F 94 C0      sete al
004043E9 8B E5        mov esp,ebp
004043EB 5D           pop ebp
004043EC C3           ret

```

- “dsk”: base64 encoded value generated based on the drives found on the machine.
- “ext”: new in 1.01. The random extension used for encrypted files.

5. Encryption preparation inspired by GandCrab

At this time, before performing the encryption, Sodinokibi replicates a behavior that is very similar to what GandCrab performs, suggesting that Sodinokibi authors learned from GandCrab ones or that they are strictly related.

Sodinokibi extracts the running processes with the combination of CreateToolhelp32Snapshot, Process32First and Process32First and checks if they match the names in the configuration. In that case, those processes are killed. The reason is that these programs could hold write access on files and therefore they could not allow the ransomware to encrypt them.

00401974	56		push esi
00401975	E8 4B 32 00 00		call i_tuoi_documenti_dei_caso.doc.4048C5
0040197A	56		push esi
0040197B	A3 80 D6 41 00		mov dword ptr ds:[41D680],eax
00401980	E8 E5 21 00 00		call i_tuoi_documenti_dei_caso.doc.40386A
00401985	E8 45 2A 00 00		call i_tuoi_documenti_dei_caso.doc.4043CF
0040198A	F7 D8		neg eax
0040198C	1B C0		sbb eax,eax
0040198E	83 E0 EA		and eax,FFFFFFEA
00401991	83 C0 56		add eax,56
00401994	A3 00 D7 41 00		mov dword ptr ds:[41D700],eax

Hide FPU	
EAX	00000056 'v'
EBX	0041C040 i_tu
ECX	FDE01E8C
EDX	0009E658
EBP	0019F414
ESP	0019F2AC
ESI	07F55D68
EDI	00000001

The list of the version 1.00 contains only the “mysql.exe” process, while the list of the version 1.01 is a lot longer and almost matches the ones used by GandCrab (source: [Symantec](#)).

00405311	33 D2	xor edx,edx	
00405313	F7 71 04	div dword ptr ds:[ecx+4]	
00405316	8B 41 08	mov eax,dword ptr ds:[ecx+8]	
00405319	8B 34 90	mov esi,dword ptr ds:[eax+edx*4]	
0040531C	EB 14	jmp i_tuoi_documenti_del_caso.doc.405332	esi:&"sqlbrowser.exe", [eax+
0040531E	FF 75 0C	push dword ptr ss:[ebp+C]	[ebp+C]:L"services.exe"
00405321	FF 76 04	push dword ptr ds:[esi+4]	[esi+4]:L"sqlbrowser.exe"
00405324	E8 68 FC FF FF	call i_tuoi_documenti_del_caso.doc.404F91	
00405329	59	pop ecx	
0040532A	59	pop ecx	
0040532B	85 C0	test eax,eax	
0040532D	74 0C	je i_tuoi_documenti_del_caso.doc.405338	
0040532F	8B 76 08	mov esi,dword ptr ds:[esi+8]	esi:&"sqlbrowser.exe"
00405332	85 F6	test esi,esi	esi:&"sqlbrowser.exe"
00405334	75 E8	jne i_tuoi_documenti_del_caso.doc.40531E	
00405336	33 C0	xor eax,eax	
00405338	5E	pop esi	esi:&"sqlbrowser.exe"
00405339	5D	pop ebp	
0040533A	C3	ret	

Another check done by the ransomware is for available network resources with WNetOpenEnumW e WNetEnumResourceW with the aim to find other files to encrypt.

```
cmd /c vssadmin.exe Delete Shadows /All /Quiet & bcdedit /set {{default}} r
ecoveryenabled No & bcdedit /set {{default}} bootstatuspolicy ignoreallfail
ures
```

Last operation before the encryption is to find all the directories with a name that matches the configuration key “wfld” and to wipe them. In this case, the list contains only “backup”. So, for example, Sodinokibi deletes Windows Defenders updates backups.

00406548	EB 75	jmp i_tuoi_documenti_del_caso.doc.40658F	
0040654A	53	push ebx	
0040654B	56	push esi	
0040654C	8D 45 F4	lea eax,dword ptr ss:[ebp-C]	esi:&L"Microsoft Windows Network"
0040654F	50	push eax	
00406550	57	push edi	
00406551	8D 45 FC	lea eax,dword ptr ss:[ebp-4]	
00406554	50	push eax	
00406555	FF 75 F8	push dword ptr ss:[ebp-8]	[ebp-8]:L"prov"
00406558	FF 15 7C CA 41 00	call dword ptr ds:[&WNetEnumResourcew]	
0040655E	89 45 F0	mov dword ptr ss:[ebp-10],eax	

6.Ransomware attack

Finally (or not?) Sodinokibi starts to iterate over the available directories with FindFirstFile and FindNextFile.

It skips files and directories that match conditions on the whitelist configuration. The others are encrypted by the ransomware that adds the random generated key as extension to the name.

0040646A	8B EC	mov ebp,esp	
0040646C	FF 75 0C	push dword ptr ss:[ebp+C]	[ebp+C]:L"\\\\?\\c:\\BOOTNXT.k12mqqhh"
0040646F	FF 75 08	push dword ptr ss:[ebp+8]	[ebp+8]:L"\\\\?\\c:\\BOOTNXT"
00406472	FF 15 68 CA 41 00	call dword ptr ds:[&MoveFilew]	

In each directory the malware also write the ransom note “{ext}.readme.txt” extracted from the configuration and a lock file.

Then it creates a file with a random name “{random}.bmp” in the %TEMP% which contains the image that will be put as a background with the help of DrawTextW and FillRect functions.

0040646A	8B EC	mov ebp,esp	
0040646C	FF 75 0C	push dword ptr ss:[ebp+C]	[ebp+C]:L"\\\\?\\c:\\BOOTNXT.k12mqqhh"
0040646F	FF 75 08	push dword ptr ss:[ebp+8]	[ebp+8]:L"\\\\?\\c:\\BOOTNXT"
00406472	FF 15 68 CA 41 00	call dword ptr ds:[&MoveFilew]	

7.C2 Registration

Once the encryption is finished, Sodinokibi starts to iterate through a giant list of domains hardcoded in the configuration (about 1k). These domains are the same across the samples we analyzed but they are ordered differently in order to mislead the analysis.

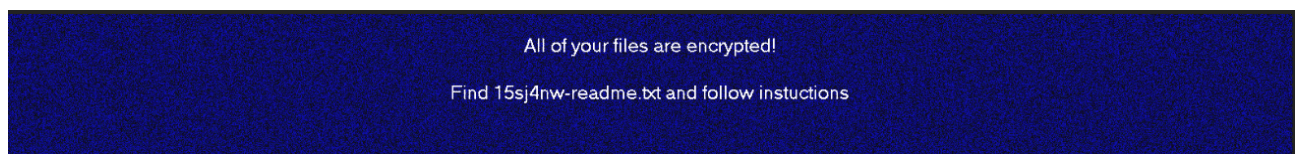
At a first glance, these domains seem legit and most of them are correctly registered.

This is not a classic DGA but the result is almost the same because the purpose is to hide the real C&C Server used by cyber criminals.

For each domain listed, Sodinokibi generates a random URI. Then it uses the winhttp.dll library functions to perform HTTPS POST requests with the created URLs.

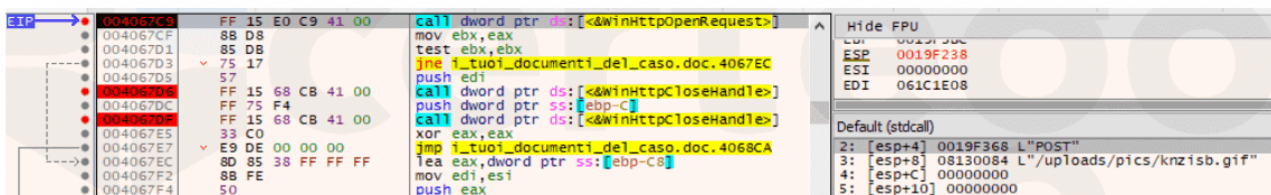
The data sent with the POST request is an encrypted form of the JSON configuration saved on the “HKEY_LOCAL_MACHINE\SOFTWARE\recfg\stat” registry key and described on the “Machine information recovery” section. In this way, malicious actors can collect important information of the infected machine.

The following are examples of some of these URLs:



Looking at an analysis of this sample in a sandbox ([AnyRun](#)), we noticed that HTTPS requests were not correctly listed. The malware can avoid traffic interception by proxies like Fiddler or Mitmproxy that are used for manual or automatic analysis.

```
hxxps://schluessedienste-hannover.de/admin/images/dcnzfpph.jpg
hxxps://alpesiberie.com/admin/tmp/sxuuaygb.png
hxxps://bratek-immobilien.de/uploads/image/bsxdfx.jpg
hxxps://bcmets.info/content/image/tjknaqfkuzxny.jpg
```



I suggest to read the following blog post where it’s further explained how these URLs are generated and why also this routine is inspired by GandCrab code: [Tesorion analysis](#)

8.Conclusion

Sodinokibi could be the *heir* of **GandCrab**. It's still at version 1.01 so maybe it's not mature yet but is actively developed and updated

Malicious actors have started to use **Sodinokibi** to generate profit, even in Italy.

It's important to continuously **monitor** your own assets, both on a network and an endpoint level, to fight against these kind of threats.

Certego Threat Intelligence Team has been studying upcoming cyber threats for years in order to provide the best protection to their customers.

IOC

```
HKEY_LOCAL_MACHINE\SOFTWARE\recfg\pk_key  
decryptor[.]top  
ap1ebzu47wgazapdqks6vrcv6zcnjppkxbr6wketf56nf6aq2nmyoyd[.]onion
```

Author:

Matteo Lodi, Threat Intelligence Team Leader

Source: <https://www.certego.net/en/news/malware-tes-sodinokibi/>