REvil - Sodinokibi

Technical analysis and Threat Intelligence Report

Article written by:
Gianfranco Tonello
Michele Zuin
Federico Girotto

CTA-2019-06-24

Last revision: 2019-07-17

Summary

Introduction

Sodinokibi ransomware, also known as REvil, made it first appearance in April 2019, where it looks to exploit the Oracle WebLogic Server vulnerability to propagate itself.

C.R.A.M. (Research Center Anti-Malware) of TG Soft has analyzed ransomware evolution in the last few months.

In Italy it made first appearance in Mat 24th 2019, with a RDP attack, as we posted in the tweet of May 28th 2019:

The authors of Sodinokibi ransomware, even if they are the first versions of their creation, seem to have a long experience in this threats of cyber-crime.

Some researchers have identified the similarities with GandCrab ransomware, whose project was shut down in beginning June. It seems that Sodinokibi ransomware is the right candidate to fill the hole left behind GandCrab.

Infection Vector

Sodinokibi ransomware uses different methods of propagation:

- Oracle WebLogic Server Vulnerability
- RDP attacks
- Spam Campaigns
- Watering hole
- Exploit kit and malvertising

In Italy, we have observed that Sodinokibi ransomware used various methods of propagation. All such methods have been found in Italy except Oracle WebLogic Server vulnerability.

The first attack that we have record was on 24th May 2019, in this case the infection vector was through RDP attack. This kind of infection vector execute a brute force on credentials, it has already been used by other ransomware as Dharma.

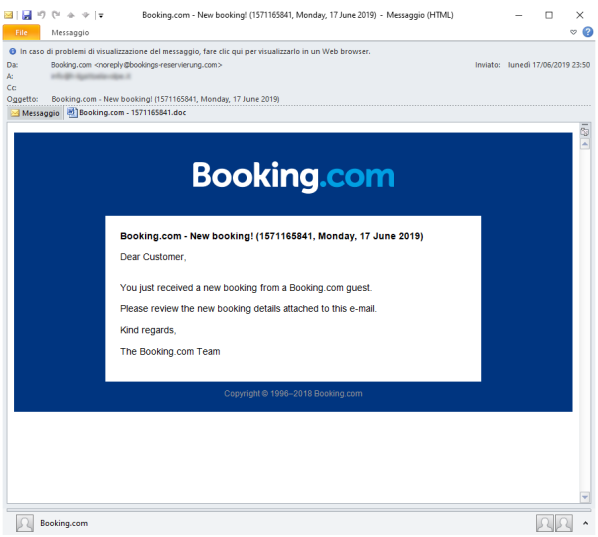
Interestingly, the IP 151.106.56[.]254 used by cyber criminal to access via RDP was the same IP identified in other RDP attacks in June of this year.

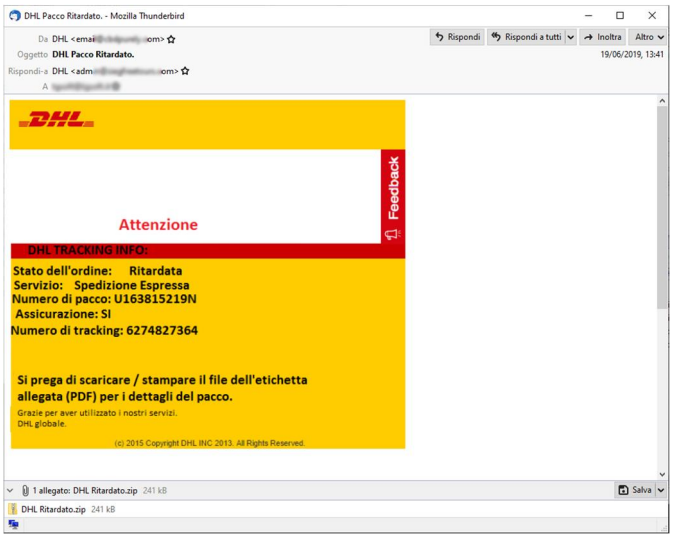
Affiliates have used spam campaigns to distributed Sodinokibi ransomware, that was recorded in June. A new campaign was discovered which deals:

- Booking.com
- DHL

"Booking.com" campaign in the summer months, is very apt choose with the summer holiday season approaches, it may induce the victims to open the attachment.

In the images below, we can see the two malspam campaigns of Sodinokibi.





In Italy the first case of watering hole was recorded on website “winrar.it” a distributor of WinRar in Italy. For the whole day on Wednesday the 19th June was downloaded Sodinokibi instead of setup of WinRar.

In 2016 “winrar.it” website was already attacked by APT StrongPity, here too this was watering hole attack, in which the setup of WinRar was modified to include and downloaded also StrongPity spy malware.

If in 2016 the attack on “winrar.it” was organized by a professional cyber-espionage organization, in the attack of this year the attackers have replaced the setup of WinRar with Sodinokibi. Who downloaded WinRar in the afternoon of 19th June, could find something strange in the downloaded file, the icons, actually, are not like the WinRar ones, as we can see in the figures below:

In addition, the execution of file does not downloaded WinRar, as has been the case of StronPity ransomware.

Attackers have poorly exploited the watering hole attack to winrar.it.

In other cases involving the spread of Sodinokibi , registered in Italy on 7th June 2019, were utilized malvertising attack .

The authors of Sodinokibi seem to be very active in spreading the ransomware.

Sodinokibi Ransomware Analysys

Then we analyze Sodinokibi version 1.1.

When the file infected from ransomware is executed, Sodinokibi generates a different mutex for each build, as an example :

`Global\D382D713-AA87-457D-DDD3-C3DDD8DFBC96`

A section of the file infected is decrypted with RC4, this section contains the configuration of the malware structured in this way:

```
{
    "pk": "",
    "pid": "",
    "sub": "",
    "dbg": ,
    "fast": ,
    "wipe": ,
    "wht": {
        "fld": [],
        "fls": [],
        "ext": []
    },
    "wfld": [],
    "prc": [],
    "dmn": "",
    "net": ,
    "nbody": "",
    "nname": "",
    "exp": ,
    "img": ""
}
```

In the table below we see the description of the fields:

Fields	Description
pk	Pubblic Key in base64
pid	Identifier
sub	Identifier
dbg	Debug: true/false
fast	True/False
wipe	True/False
wht -> fld	Folder exclusions
wht -> fls	Files exclusions
wht -> ext	Exclusion of the extension
wfld	Wipe folder
prc	Process to finish
dmn	Domains C2
net	Files encryption in the network: true/false
nbody	Instructions for payment
nname	{EXT}-readme.txt (dove EXT è l'estensione del file cifrato)
exp	Exploit True/False
img	Image contained in alert encryption on the desktop

If "exp" filed is "true" then a 32 or 64 bit shellcode is executed with the exploit CVE-2018-8453 through the elevation of privilege.

The next step is create a registry key **REcfg** if it is not already exist:

`HKEY_LOCAL_MACHINE\SOFTWARE\recfg`

If the key do not have permissions, it is created in HKEY_CURRENT_USER.

The following values are created within **REcfg**:

- pk_key
- sk_key
- 0_key
- rnd_ext
- stat

Calculate the private and public keys

Now the private and the public keys are calculated, as we can see in the figure:

```
loc_132388:
lea     eax, [ebp+var_88]
push    offset pk_key_14D5A0
push    eax
call    _Calcola_Key_Privata_Pubblica_1355B8 ; Calcola_Key_Privata_Pubblica (pKeyPrivata, pKeyPubblica)
push    20h
pop     ebx             ; ebx = 20h
lea     eax, [ebp+var_4]
mov     [ebp+var_C], ebx ;  20h
push    eax
push    ebx             ; ebx = 20h
lea     eax, [ebp+var_88]
push    eax
push    offset pk_config__14D580
call    sub_13597B      ; pBuff_Key = (key, buffer IN, size IN, size out)
mov     edi, eax        ; buffer output per sk_key
lea     eax, [ebp+var_8]
push    eax
push    ebx
lea     eax, [ebp+var_88]
push    eax
push    offset unk_14C020 ; master key pubblica
call    sub_13597B      ; pBuff_Key = (key, buffer IN, size IN, size out)
mov     esi, eax        ; buffer output 0_key
lea     eax, [ebp+var_88]
push    ebx
push    eax
call    _Wrp_ZeroMemory_135966
add     esp, 30h
test    edi, edi
jz      loc_1324F4
```

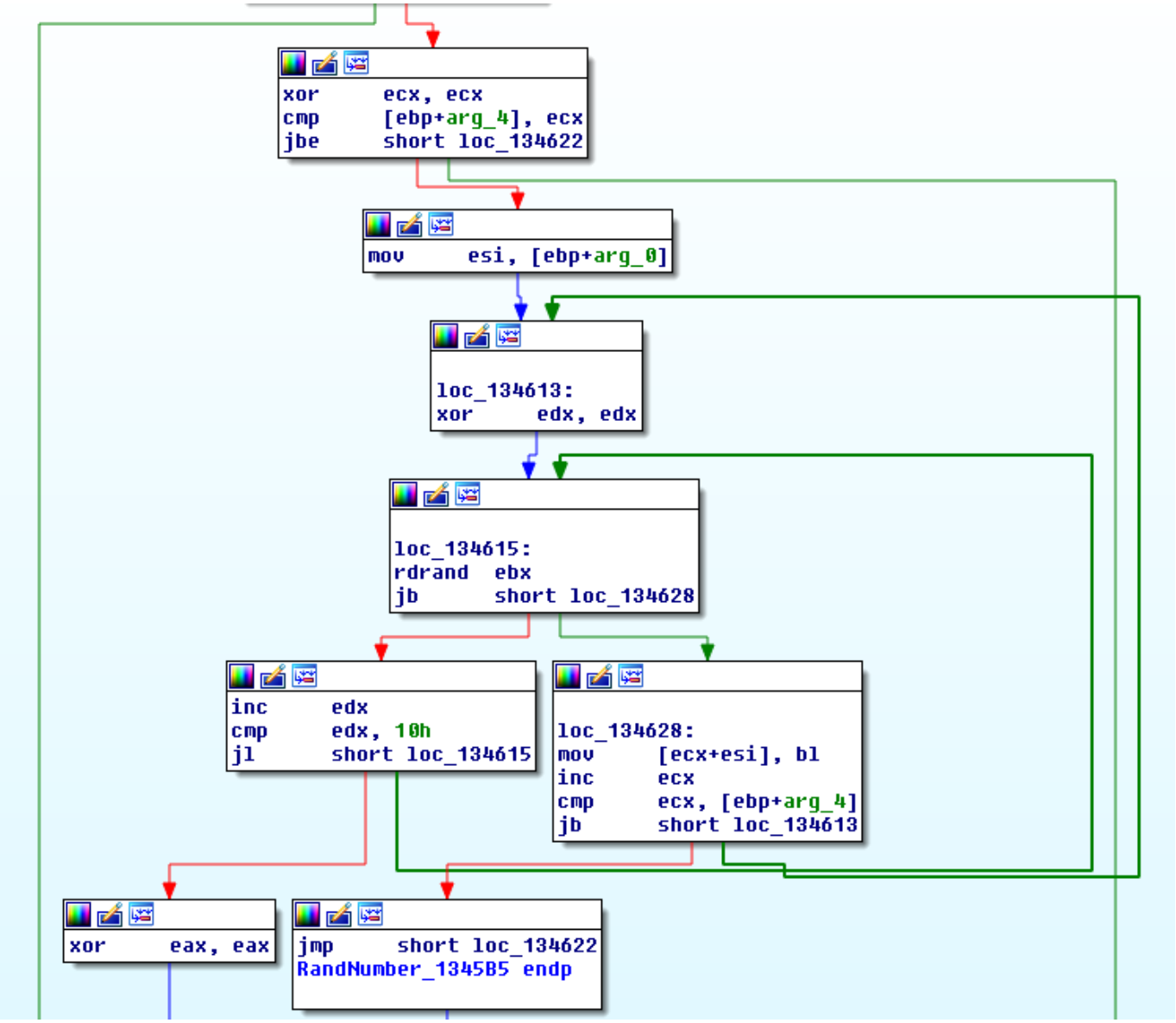
Private and public keys are calculated in this way:

```
; Calcola_Key_Privata_Pubblica (pKeyPrivata, pKeyPubblica)
; Attributes: bp-based frame

_Calcola_Key_Privata_Pubblica_1355B8 proc near

arg_0= dword ptr  8
arg_4= dword ptr  0Ch

push    ebp
mov     ebp, esp
push    [ebp+arg_0]     ; key privata
call    _Calcola_NumeroRandom_20h_13560B
pop     ecx
test    eax, eax
jnz     short loc_1355CA ; key pubblica
```

```
pop     ebp
retn
```

```
loc_1355CA:
push    [ebp+arg_4]     ; key pubblica
push    [ebp+arg_0]     ; key privata
call    sub_1355DC      ; (pKeyPrivata, pKeyPubblica)
pop     ecx
xor     eax, eax
pop     ecx
inc     eax
pop     ebp
retn
_Calcola_Key_Privata_Pubblica_1355B8 endp
```

The private key was generated from random number of 256 bit, from the figure we can see the random number generation subroutine PRNG (PseudoRandom Number Generators):



The function to generate PRNG use the hardware Intel Ivy Bridge, based on NIST's SP 800-90 guidelines, through the call to assembly **rdrand** instruction**.**

The random number generated, before it becomes private key, is elaborated in this way:

```
mov     al, [esi+1Fh]
and     byte ptr [esi], 0F8h
and     al, 3Fh
or      al, 40h
mov     [esi+1Fh], al
xor     eax, eax
inc     eax
```

At this point, starting from private key was generated public key. The private and public keys are generated using ECC (Elliptic Curve Cryptography).

The keys (private and public) are both two numbers of 256 bit, which define two points on the elliptic curve.

The Exchange of the keys is made with the "E*lliptic Curve Diffie-Hellman*" (ECDH) method, where:

$$d_A P_B = d_B P_A$$

Given G a fixed point of the curve, where:

- d_A = private key of A (secret random number)
- $P_A = G*d_A$ = public key of A (G multiplied by d_A)
- d_B = private key of B (secret random number)
- $P_B = G*d_B$ = public key of B

Sodinokibi use elliptic curve "Curve25519", in which G={9}, developed by Dan Bernestein, as supposed in the post of Eric Klonowski (@noblebarstool) on Twitter.

After Sodinokibi has generated the ECC pair of keys in the memory, which we call dk_key (private key) and pk_key (public key), the public key is stored in the `recfg` regisry key inside of the value `pk_key`:

`HKEY_LOCAL_MACHINE\SOFTWARE\recfg`

`[pk_key]` = Public Key

sk_key Data Structure

At this point sk_key data structure is generated by the call to Sub_13597B subroutine:

```
pBuff_sk_key = Sub_13597B (key_pubblica_json, key_privata, size IN, size out)
```

The Sub_13597B aims to encrypt the private key generated inside sk_key data structure.

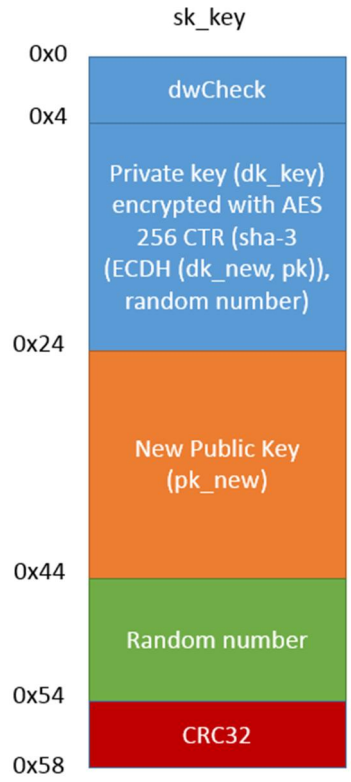
The Sub_13597B takes 4 input parameters:

- key_pubblica_json: public key "pk" inside the json configuration section
- key_privata: private key generated "dk"
- size IN: size of private key "dk"
- size out: size of sk_key structure

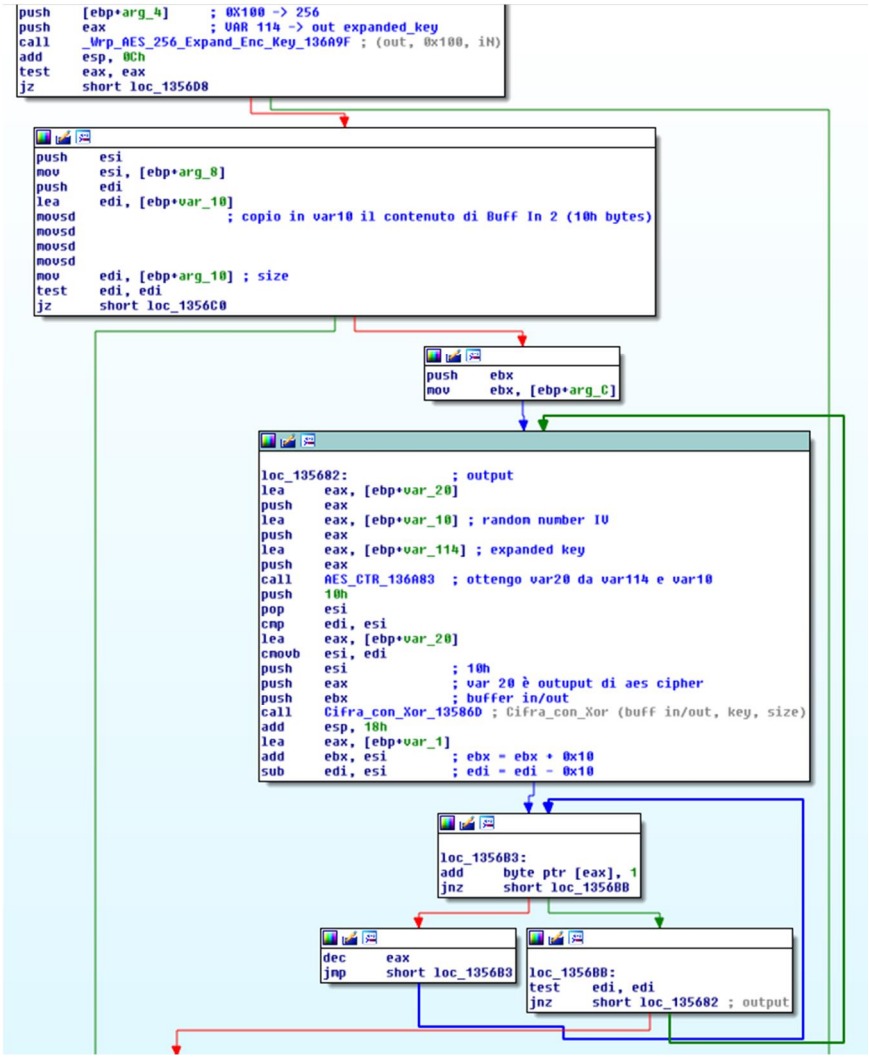
Sub_13597B subroutine execute the following steps:

1. Allocate a buffer of 0x58 byte and copy the private key (*dk_key*) "key_privata" from offset 0x4 into buffer
2. Calculate a new pairs of ECC keys, one private (*dk_new*) and one public (*pk_new*)
3. Calculate *dk_new*pk -> shared_key_new* (whre *pk* is public key inside the json configuration section) and the result is "hashed" with SHA-3.
4. Calculate a random number of 16 byte -> *random_16,* it will be used as IV (initialization vector for AES)
5. Encrypts the buffer allocated from 0 to 0x24 via AES-256 CTR through the IV initialization vector and SHA-3 (*shared_key_new*)
6. Copy the public key *pk_new* into buffer allocated at offset 0x24
7. Copy the random number *random_16* into buffer allocated at offset 0x44
8. Calculate the CRC32 of the buffer allocated from 0 to 0x24 and save the result at offset 0x54

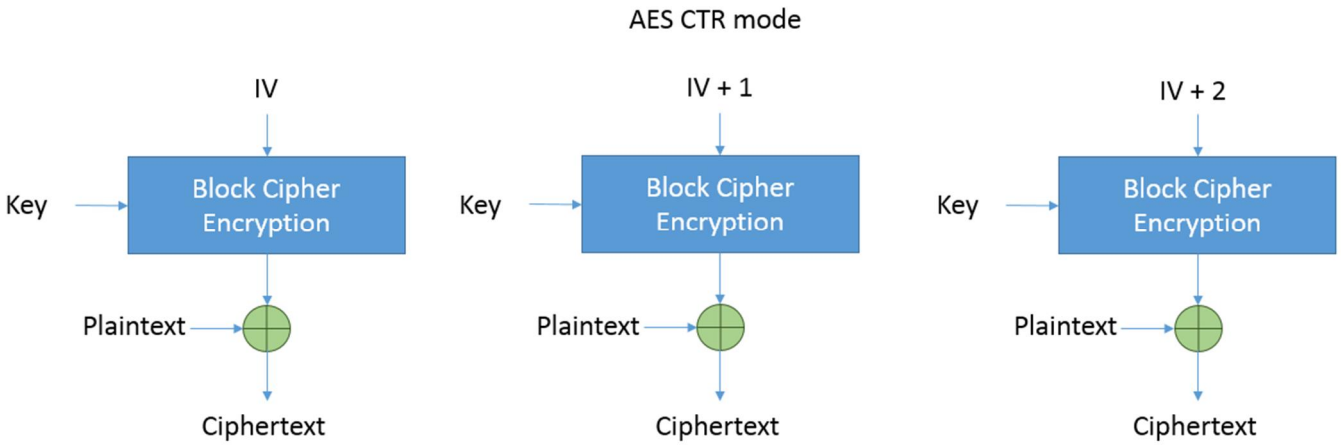
Sub_13597B subroutine returns the pointer to buffer that is allocated to of 0x58 byte inside the sk_key data structure.
sk_key data structure, as we see on the right figure, will be stored in the registry under the same name.



We can see the call to AES-256 in CTR mode, in the figure below:



AES CTR takes the following scheme:



0_key Data Structure

0_key data structure is generated in a similar way, by the call to Sub_13597B subroutine:

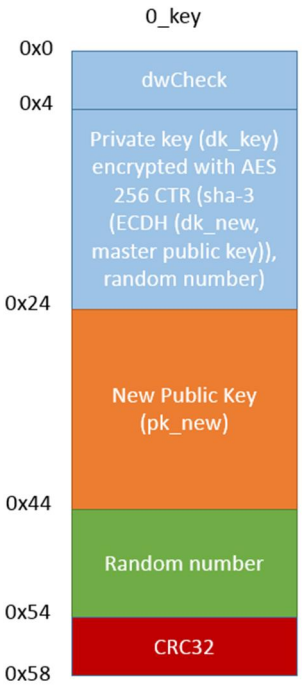
```
pBuff_0_key = Sub_13597B (master_key_pubblica, key_privata, size IN, size out)
```

The procedure for generation of 0_key data structure is similar to that of sk_key data structure, in this case it is used a "master public key" stored inside an executable file instead of the public key pk (the one inside the json configuration section).

The "embedded" master public key is:

```
79 CD 20 FC E7 3E E1 B8 1A 43 38 12 C1 56 28 1A
04 C9 22 55 E0 D7 08 BB 9F 0B 1F 1C B9 13 06 35
```

Inside the 0_key data structure we have the dk private key encrypted through the "master public key".
0_key data structure, as we see in the figure below, will be saved in the registry under the same name.



Registry Key "rnd_ext"

The value "rnd_ext" is stored inside the registry key **REcfg,** it contains the encrypted file extension randomly calculated.

Registry Key "stat"

The value "stat" is stored inside the registry key **REcfg,** it contains the following string formatted:

```
{"ver":%d,"pid":"%s","sub":"%s","pk":"%s","uid":"%s","sk":"%s","unm":"%s","net":"%s","grp":"%s","lng":"%s","bro":%s,"os":"%s","bit":%d,"dsk":"%s","ext":"%s"}
```

It is stored in "stat" in encrypted and base64 encoded form.

Name	Description
ver	Version of Sodinokibi
pid	PID of json
sub	SUB of json
pk	PK of json
uid	CRC32 di "processor brand string" e Volume Serial Number (8 bytes)
sk	sk_key in BASE64
unm	Username
net	Name of computer
Grp	Name of workgroup or domain
lng	ID language
bro	True / False if ID language is a "friend"
Os	Operating System
Bit	Value: 86 or 64
Dsk	Information of disk in base 64 (drive and free space)
Ext	Extension of encrypted file

Countries considered "friends" on the basis of the "bro" value:

- Romania
- Russia
- Ukraine
- Belarus
- Estonia
- Latvia
- Lithuania
- Tajikistan
- Iran
- Armenia
- Azerbaijan
- Georgia
- Kazakistan
- Kyrgyzstan
- Turkmenistan
- Uzbekistan

The Sodinokibi ransomware ends the current process if the keyboard language belong to the list of countries considered "friends".

The “stat” formatted string is encrypted with a master public key stored inside a executable file.

The master public key “embedded” is:

```
36 7D 49 30 85 35 C2 C3 68 60 4B 4B 7A BE 83 53
AB E6 8E 42 F9 C6 62 A5 D0 6A AD C6 F1 7D F6 1D
```

Ransom instruction

Ransom instruction are prepared from the body, which is extracted from the “nbody” field of the json configuration.

The body is formatted with the following value:

- uid
- rnd_ext
- stat on base 64

The “uid” is the user ID calculated from CRC of “processor brand string” and Volume Serial Number, which is used to compose the URL where to make the ransom payment:

- http://aplebzu47wgazapdqks6vrcv6zcnjppkbxbr6wketf56nf6aq2nmyoyd.onion/<uid>
- http://decryptor.top/<uid>

Terminate Processes and delete Shadow Copy

The processes listed in the JSON configuration under “prc” are killed and the Windows Shadow copy with the following command are deleted:

```
cmd.exe /c vssadmin.exe Delete Shadows /All /Quiet & bcdedit /set {default}
recoveryenabled No & bcdedit /set {default} bootstatuspolicy
ignoreallfailures
```

Wipe

Then the malware checks the "wipe" value in the JSON configuration and if set to true it deletes all the files contained in the folders that correspond to the "wfld" value of the JSON configuration.

File encryption

A Thread is created which is pending on function “GetQueuedCompletionStatus”.

Files on local disk and network folder are numbered (if the “net” parameter of JSON configuration is a “true” value) then proceed with file encryption.

In every folder is created a .lock file and the instructions regarding the ransom with name *{random extension}-readme.txt*.

Files and folders that correspond to the JSON "wht" field containing the subfields "fld", "fls" and "ext", which are respectively for "folder", "files" and "extension" are excluded from encryption.

Here is an example:

```
"wht": {

        "fld": ["google", "mozilla", "$windows.~bt", "programdata",
"$recycle.bin", "program files (x86)", "appdata", "msocache", "program
files", "windows.old", "$windows.~ws", "application data", "perflogs",
"windows", "boot", "intel", "system volume information", "tor browser"],

        "fls": ["bootsect.bak", "autorun.inf", "ntldr", "ntuser.dat.log",
"ntuser.ini", "boot.ini", "ntuser.dat", "bootfont.bin", "desktop.ini",
"thumbs.db", "iconcache.db"],

        "ext": ["exe"]

    }
```

For each file intended to encryption is generated a Salsa20 key, as follows:

```
push    eax             ; var_20
call    _Calcola_Key_Privata_Pubblica_1355B8 ; Calcola_Key_Privata_Pubblica (pKeyPrivata, pKeyPubblica)
lea     eax, [ebp+var_40]
push    eax
lea     eax, [ebp+var_20]
push    offset pk_key_14D5A0 ; pk_key del registro
push    eax             ; var 20
call    _Calcola_Var40_135822 ; (Buffer IN, Key, Buffer OUT)
lea     eax, [ebp+var_20]
push    20h
push    eax
call    _Wrp_ZeroMemory_135966
mov     esi, [ebp+arg_0] ; struttura dati
lea     eax, [ebp+var_40] ; key di cifratura che viene copiata nella tabella master di Salsa20
push    40h
push    100h
push    eax
lea     edi, [esi+108h]
push    edi
call    __Set_Salsa_Tabella_136EA3
lea     eax, [ebp+var_40]
push    20h
push    eax
call    _Wrp_ZeroMemory_135966
add     esi, 0F8h
push    8               ; size vettore
push    esi             ;  Buffer Vettore Inizializzazione
call    _Calcola_RandomNumber_13578B ; _Calcola_RandomNumber (PBuffer, dwSize)
push    esi             ;  puntatore al Vettore di Inizializzazione IV
push    edi             ; edi punta alla struttura Dati offset 0x108 Tbl Master Salsa
call    _Set_IV_Tabella_Salsa_136E85
add     esp, 44h
push    20h             ;  size
push    ebx             ;  buffer
push    0
call    _CRC32_1356DC   ; calcola in eax il CRC32 (val, buffer, size)
mov     ecx, [ebp+arg_0] ; struttura dati
add     esp, 0Ch
pop     edi
mov     [ecx+100h], eax ; crc32 del buffer D8
mov     eax, dword_14D714
pop     esi
mov     [ecx+104h], eax
pop     ebx
mov     esp, ebp
pop     ebp
retn
```

Encryption algorithm used by Sodinokibi is Salsa20.

The encryption key for Salsa20 is obtained in this way:

1. Calculate a new pairs of ECC private/public keys (dk_new_file, pk_new_file)
2. Calculate SHA-3 (*dk_new_file*pk_key) -> shared_key_salsa* (where *pk_key* is a public key stored inside registry under pk_key voice). In *shared_key_salsa* we will obtained the key which is plugged in Salsa20 master table.
3. Calculate a random number of 8 byte for the initialization vector of the Salsa20 master table.
4. Composes the Salsa20 master table.

It is created in memory a data structure that holds:

- Handle of the file to be encrypted
- Sk_key
- 0_key
- pk_new_file
- Initialization vector of Salsa20
- The CRC32 of pk_new_file
- Master table of Salsa20

This data structure is passed to the Thread created previously through the API functions:

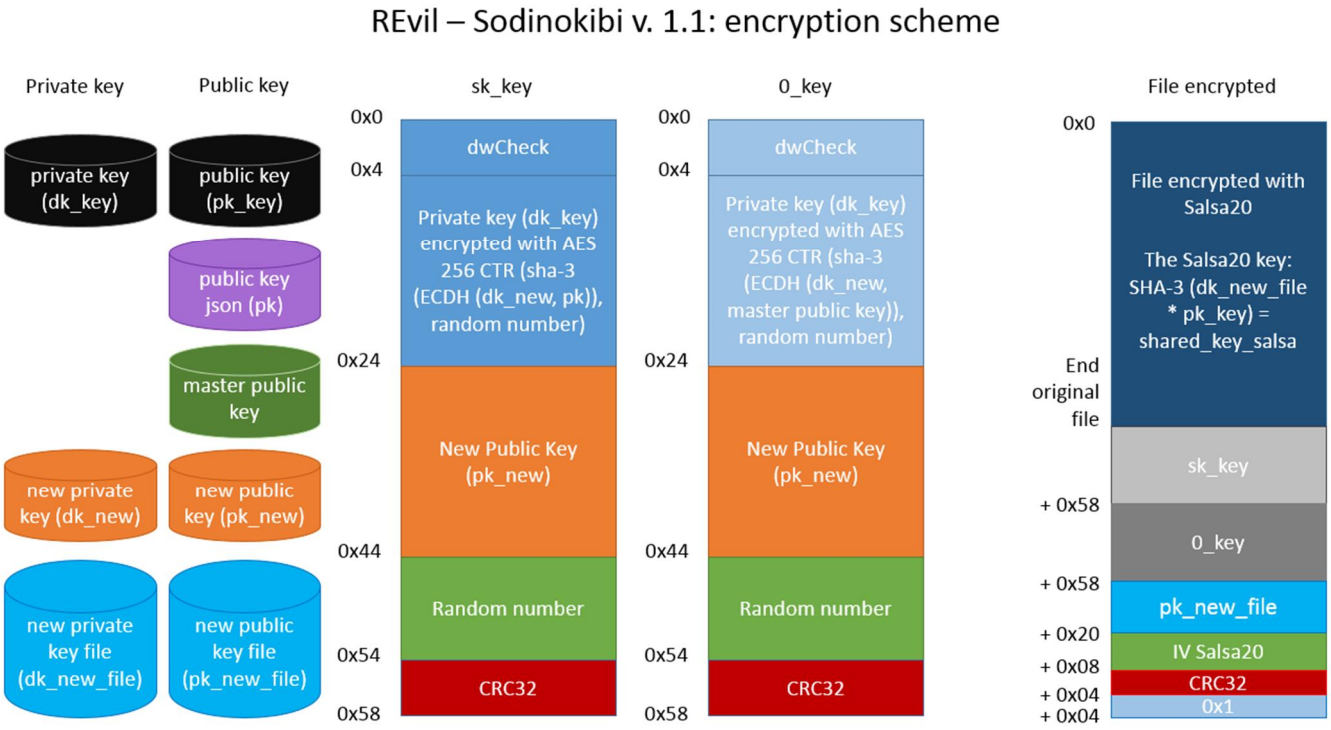
- CreateIoCompletationPort
- PostQueuedCompletionStatus

The thread is pending on the GetQueuedCompletionStatus API function, when it receives a new call it starts the file encryption phase through the Salsa20 algorithm and then the following fields are saved in the data structure :

- Sk_key
- 0_key
- pk_new_file
- Initialization vector of Salsa20
- The CRC32 of pk_new_file

The size of the hanging part varies depending on Sodinokibi version. In versions 1.0 and 1.1 the length is 0xE0 bytes while in version 1.2 it is 0xE4 bytes.

In the figure we can see the encryption scheme of Sodinokibi version 1.1:



Desktop image

At the end of the files encryption, the next step is to modify the desktop image , which we can see in the figure below:

The image is generated using API functions for the graphics and the text is inserted using “DrawText” function, that is loaded in “img” field through JSON configuration.

C2 Server

We find a list of 1079 domains inside the JSON configuration. Sodinokibi makes a connection with each domain of this list generating a URL through a DGA algorithm using the following terms:

Terms	Extension
• wp-content • pictures • news • pics • admin • data • temp • graphic • game • static • assets • tmp • uploads • images • include • image • content	• jpg • gif • png

https://<host>/<term 1>/<term 2>/<random chars>.<extension>

Some examples:

- https://stagefxinc.com/wp-content/pictures/pmkapi.jpg
- https://birthplacemag.com/admin/pictures/hpxxqbak.gif
- https://clemenfoto.dk/news/pics/ohxkyt.gif
- https://wineandgo.hu/admin/pics/ahlpbrzo.jpg
- https://lexced.com/data/temp/hpttgdyg.png

Sodinokibi transmits through a "POST" to each domain of the list the "stat" data structure in encrypted form . From our analysis only the following domains responded with "HTTP / 1.1 200 OK":

www.zuerich-umzug.ch	geitoniatonaggelon.gr
belofloripa.be	insane.agency
www.soundseeing.net	acb-gruppe.ch
utilisacteur.fr	www.cardsandloyalty.com
www.airserviceunlimited.com	www.sbit.ag
www.mediahub.co.nz	yourhappyevents.fr
www.irizar.com	tieronechic.com
www.cleanroomequipment.ie	mariajosediazdemera.com
www.pinkxgayvideoawards.com	www.skyscanner.ro
www.rhino-turf.com	11.in.ua
mike.matthies.de	funworx.de
drbenveniste.com	www.omnicademy.com
scotlandsroute66.co.uk	www.bratek-immobilien.de
m2graph.fr	metroton.ru

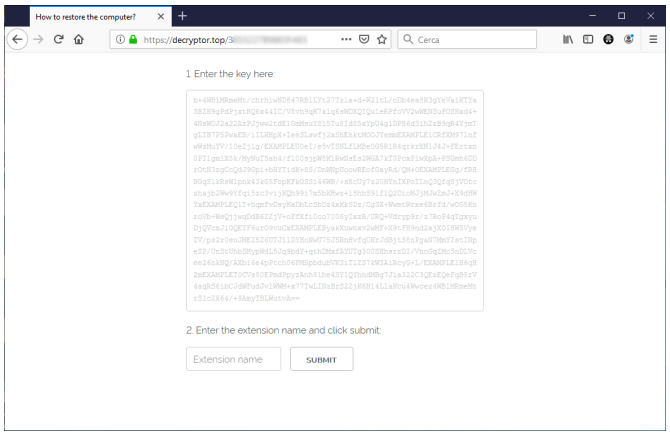
But this does not mean that one of these domains is that of Sodinokibi C2 Server.

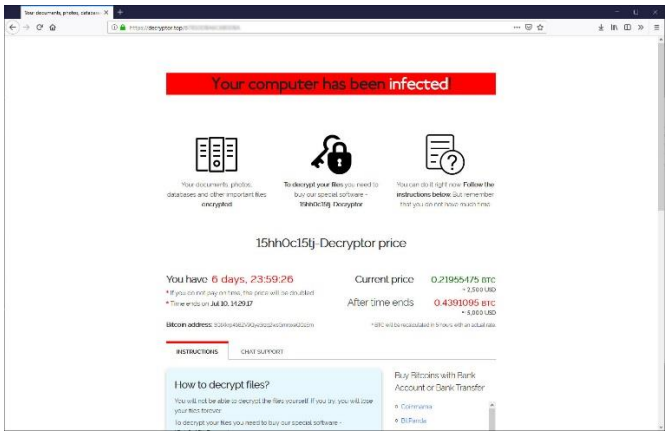
Ransom payment

According to the ransom instructions, the victim have to connect to the following domains for the payment methods:

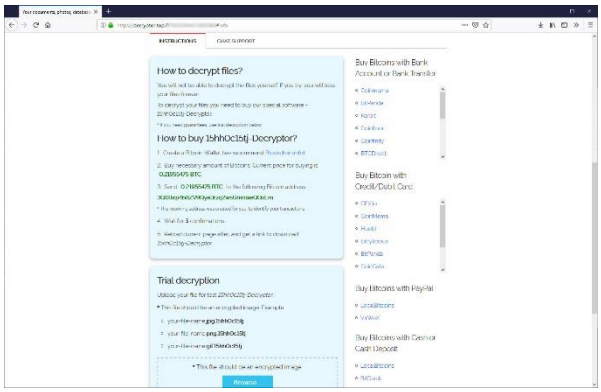
- http://aplebzu47wgazapdqks6vrcv6zcnjppkbxbr6wketf56nf6aq2nmyoyd.onion/<uid>
- http://decryptor.top/<uid>

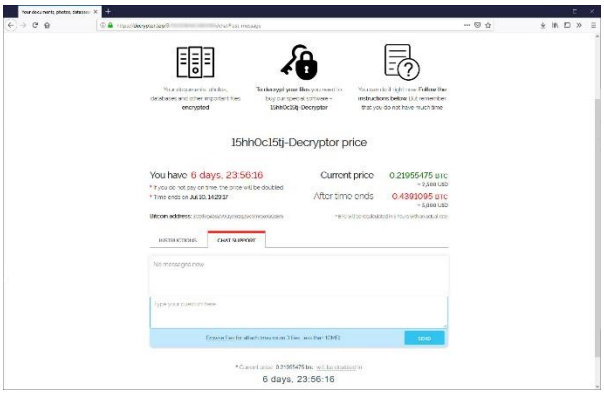
Victims are requested to enter first thing, the random extension and the “Key” value contained in ransom instructions (it is the “stat” version encrypted on base 64).





When victims input this data the payment amount is generated and are provided information on how to purchase BitCoin, and in addition a support chat is included, as we can see in the following images:





The wallet for payment is generated automatically for each victim, the ransom price is $ 2,500 it doubles to $ 5,000 if payment is not made within 7 days.

How does decryption work?

The only way to recover the encrypted files by Sodinokibi is with a “dk_key” private key. The decryption key is encrypted inside “sk_key” and “0_key”.

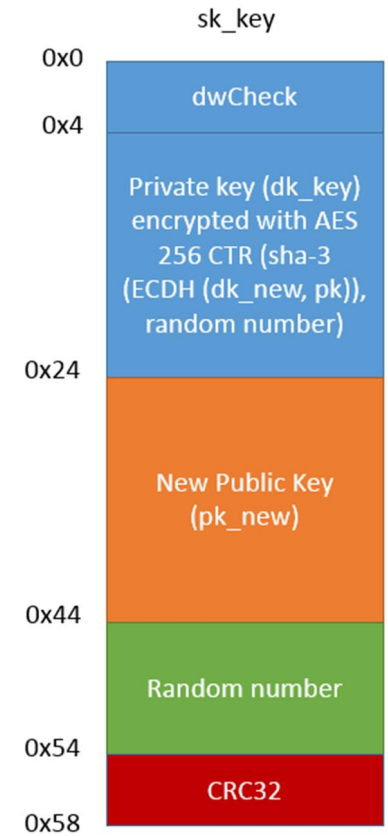
The attacker recovered “dk_key” in these ways:

1. Decrypting sk_key
2. Decrypting 0_key

Now in order to decrypt “sk_key” the attacker use a secret key, the private key “dk” , which only they know. The private key “dk” is the symmetric key of the public key “pk” stored in the json configuration.
The public key “pk_new” is put in unencrypted way inside “sk_key” structure.
It is calculated the value: dk * pk_new = *shared_key_new*
The “*shared_key_new*” is the same as: dk_new*pk.

The private key (`dk_key`) is encrypted with AES-256 CTR through the "`SHA-3 (shared_key_new`" and the `random number` (IV) which is on offset 0x44.
Decrypting the buffer from 0x4 to 0x24 with AES-256, through "SHA-3 (shared_key_new)" and the `random number` you get "`dk_key`".

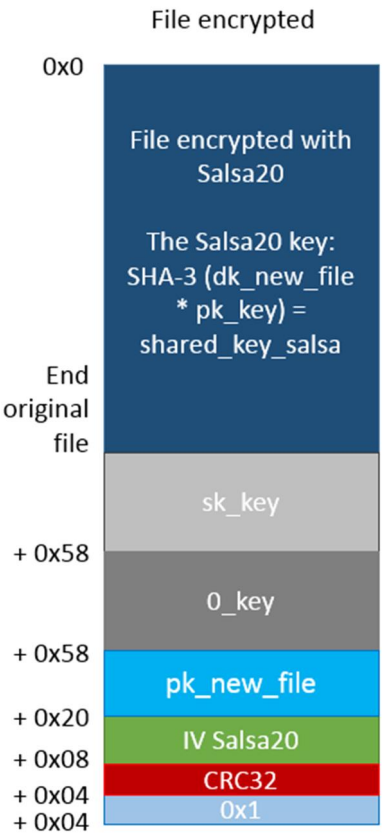
Now the same procedure can be performed to decrypted “0_key”, in this case is used the master private key, which only the authors of Sodinokibi know, to get “dk_key”.

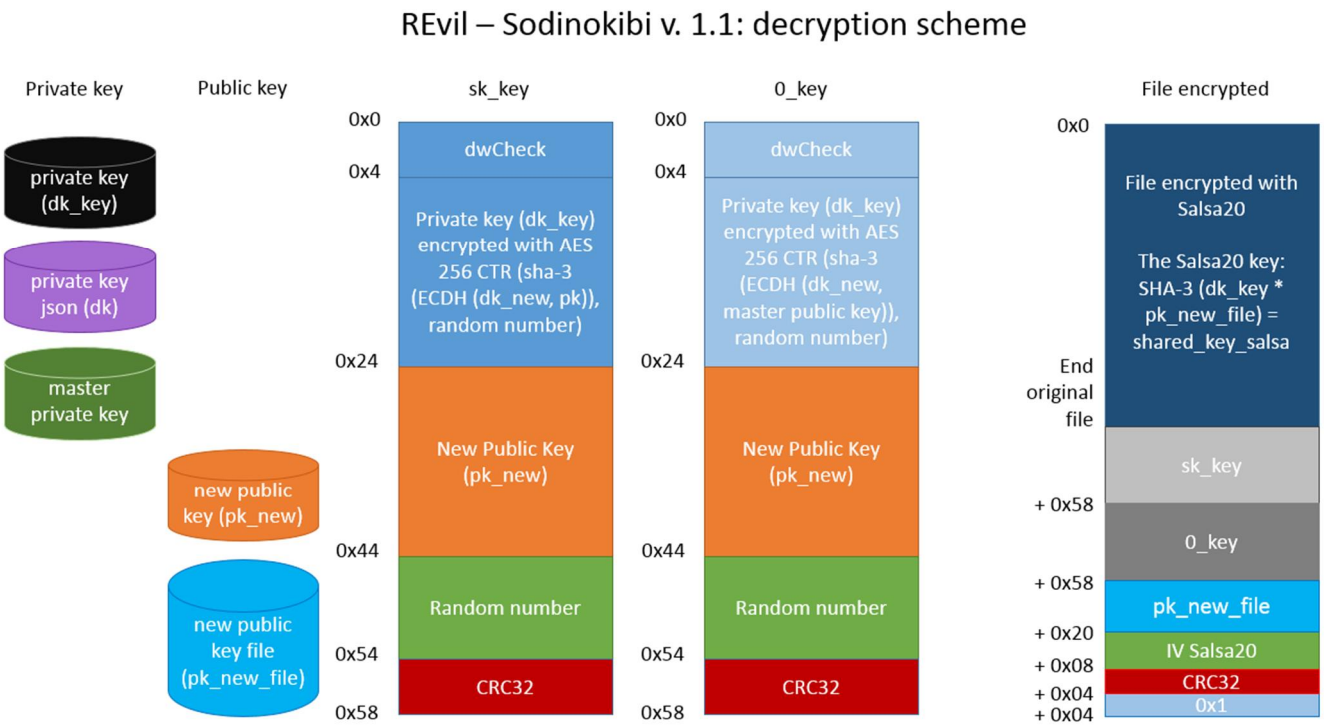


Now we know dk_key so to determinate the encryption key used in Salsa20 we execute the following operation:
SHA-3 (dk_key *pk_new_file) = shared_key_salsa
Where the public key pk_new_file is put in unencrypted way at the end of the encrypted file.

shared_key_salsa is also equals to SHA-3 (*dk_new_file*pk_key)*

In shared_key_salsa we will have the key that is inserted in the Salsa20 master table.
Now it is possible to decrypt the files through shared_key_salsa.





Versions

The authors of Sodinokibi have developed the following versions:

Version	Date	Size appending data
1.0a	2019-04-23	0xe0
1.0b	2019-04-27	0xe0
1.0c	2019-04-29	0xe0
1.1	2019-05-05	0xe0
1.2	2019-06-10	0xe4
1.3	2019-07-08	0xe4

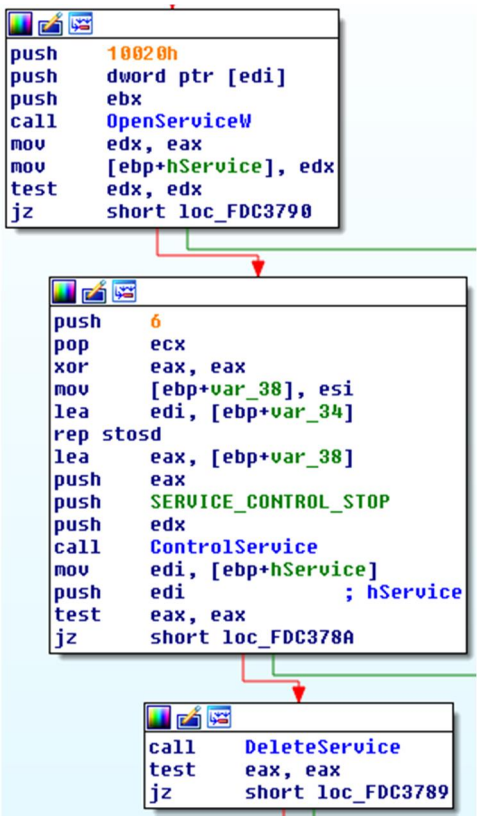
Version 1.2

In version 1.2 the registry key "`sub_key`" has been added which contains the public key of the json configuration (`pk`) and the data size in the encrypted files is 0xe4 bytes, where an additional control dword with value 0 has been added.

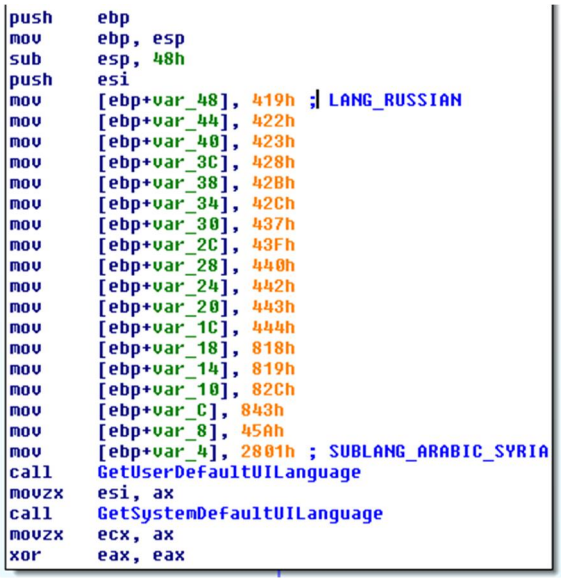
In version 1.3 there is a new field inside to json configuration called “svc”, the field contains the list of services to stop.

Version 1.3

In this version has been added a field called “svc” in the json config. This field contains a list of services to delete, as we can see in the figure.



Furthermore to verify if the victim is from a “friend” country, in addition to check of language of keyboard has been added checks on the default language and on system language, as we can see in the figure.



It uses WQL to determinate the creation of processes:

```
SELECT * FROM __InstanceCreationEvent WITHIN 1 WHERE TargetInstance ISA
'Win32_Process'
```

Furthermore it uses a new key of registry instead of “**REcfg**”:

- HKEY_LOCAL_MACHINE\SOFTWARE\QtProject\OrganizationDefaults

Inside to QtProject\OrganizationDefaults are saved the following values:

- pvg
- sxsP
- BDDC8
- f7gVD7
- Xu7Nnkd
- sMMnxpgk

Table of comparison for the version 1.2 and 1.3:

Vers. 1.2: REcfg	**Vers. 1.3: QtProject\OrganizationDefaults**
sub_key	pvg
pk_key	sxsP
sk_key	BDDC8
0_key	f7gVD7
rnd_ext	Xu7Nnkd
stat	sMMnxpgk

Telemetry

The trend of Sodinokibi malware campaigns has been monitored between April and July 2019.

In the table below we can see the campaigns monitored:

Data Campagna	Campagna	PK	PID	SUB	Versione	Data compilazione
25/04/2019	Oracle Weblogic	nAjfiPcolyelwwCkM1hLhXo5HUQMtrAB+7m8eHzerho=	7	3	1.0a	2019-04-23 18:21:53
25/04/2019	Oracle Weblogic	nAjfiPcolyelwwCkM1hLhXo5HUQMtrAB+7m8eHzerho=	7	3	1.0a	2019-04-23 18:21:53
25/04/2019	Oracle Weblogic	nAjfiPcolyelwwCkM1hLhXo5HUQMtrAB+7m8eHzerho=	7	3	1.0a	2019-04-23 18:21:53
25/04/2019	Oracle Weblogic	nAjfiPcolyelwwCkM1hLhXo5HUQMtrAB+7m8eHzerho=	7	3	1.0a	2019-04-23 18:21:53
25/04/2019	Oracle Weblogic	nAjfiPcolyelwwCkM1hLhXo5HUQMtrAB+7m8eHzerho=	7	3	1.0a	2019-04-23 18:21:53
		a54FxmOM4c90SBAgCVw4ykJv62ImcbOvaHKwO8OKegI=	19	29	1.0b	2019-04-27 18:11:51
		a54FxmOM4c90SBAgCVw4ykJv62ImcbOvaHKwO8OKegI=	19	29	1.0c	2019-04-29 19:06:06
		a54FxmOM4c90SBAgCVw4ykJv62ImcbOvaHKwO8OKegI=	19	29	1.0c	2019-04-29 19:06:06
		N3IqbCUZr/g/XgALTUaGw7K8E5UvA+CcRa5zto0xg0A=	20	45	1.0c	2019-04-29 19:06:06
		TmrkEVU29HHz1nfhwl0C6p4U5syGzUCmcyAJQnZSHyY=	8	10	1.0c	2019-04-29 19:06:06
		4hKQrOidB69uTPA/7uaOuTipRsh2y956X1K+jyyLUjA=	17	11	1.1	2019-05-05 17:38:48
		eYI9jfld2wfrBiZk/ABspJesaySH6q+XbmHRQ55NBkE=	19	100	1.1	2019-05-19 18:08:46
		w0qhPcoO83YCbvmGI4ySs7ZiTUaT5YAk0DXIM/hOnjQ=	20	44	1.1	2019-05-22 18:42:29
		Xew60HCSStmaZwEnoW4XuhBiy5I3SyKugEH5PM4P7RA=	15	19	1.1	2019-05-22 18:42:29
		io3clxJXtLLzcA1anNSmn//tKeId5pGV/mVuqwvms3g=	20	46	1.1	2019-05-22 18:42:29
		eYI9jfld2wfrBiZk/ABspJesaySH6q+XbmHRQ55NBkE=	19	100	1.1	2019-05-22 18:42:29
		ClwOJSOhyaamJ5eplhJrLN5UJdwH29Ky8t+Yn3WeLzg=	30	128	1.1	2019-05-24 14:41:21
		duPwGxBEa19yzAI27JhOVXw155oZWe3CWVbWJ7uwhBU=	16	165	1.1	2019-05-24 14:41:21
24/05/19	RDP	2Dj6WyDEOKff6CVJadXjX+ogDuXN/XnIdrVWffa6/B0=	19	36	1.1	2019-05-24 14:41:21
03/06/19	Malpam	pzprC6xbhNFhM/+qJI6gCrd2pnCgyRdai+B89OUhWAw=	30	97	1.1	2019-05-24 14:41:21
		m7cFgORjIUsRFy4odzcrLk+3iOTw9TNGLdSy6RjQImQ=	19	96	1.1	2019-06-03 18:09:45
		pzprC6xbhNFhM/+qJI6gCrd2pnCgyRdai+B89OUhWAw=	30	97	1.1	2019-06-03 18:09:51
		U5gGGTWKYrgvh5QFI+53Jc7aj8ntwjj0C4ai0/2A+jg=	34	298	1.1	2019-06-03 18:09:51
		N9tiPqA45L8cXACRHIBdJFayV8M5MEF4JjppDRO+oHU=	30	113	1.1	2019-06-03 18:09:51
		pzprC6xbhNFhM/+qJI6gCrd2pnCgyRdai+B89OUhWAw=	30	97	1.1	2019-06-03 18:09:51
		p+iVJIiHGF12r1Q7fPSAF3Y36m0DmS4bbOtZMLKszAI=	16	288	1.1	2019-06-03 18:09:51
		1LSb3+cEvUYZYvzU06n8wFiQCczYZ0MrZwUCy0HN7TY=	34	295	1.1	2019-06-03 18:09:51
		fXWQXz0Or53eh4p5JZnqYIiIQ+tPjrrni5z6Y+Ocvw0=	16	267	1.1	2019-06-03 18:09:51
		F5YmiEk1fBN5E7SkF7sRqBE5+QRpLLYtk0ONcITtzWM=	16	250	1.1	2019-06-03 18:09:51
		KtKn8udbrebS5jbzcimIkGAbGMIwX9Ks85rOWrmJ23Q=	28	285	1.2	2019-06-10 15:29:32
		jD6pLfwUHIEoWBKadIZ4A78CLm8I0UKIzdzW7XautWE=	33	357	1.2	2019-06-10 15:29:32
		w2TWFCLDTFMuBv5VN6eA5NHyUM7SRRLt+hIuKWXk8mE=	40	450	1.2	2019-06-10 15:29:32
		PdQtqjCAKZmIJn1Fbw1ZGic+XVzOOTwt4Gm1gdXGsXg=	16	314	1.2	2019-06-10 15:29:32
		X5KVRMdkoLhmeigRMY9Ve4j+/3uVeOOjDgMAM4V22mA=	12	313	1.2	2019-06-10 15:29:32
		Xew60HCSStmaZwEnoW4XuhBiy5I3SyKugEH5PM4P7RA=	15	19	1.2	2019-06-10 15:29:32
		/SvNLPYVd04yhjQWFntNHZ0bsHYz2DzRIF+HjkQuTmE=	33	331	1.2	2019-06-10 15:29:32
18/06/19	Malspam – Booking	Js9mSQ5X8GfxGiHDyNSEBzRCDIONrR0tet7eKc6ptCk=	27	439	1.2	2019-06-10 15:29:32
19/06/19	Malspam – DHL	ClwOJSOhyaamJ5eplhJrLN5UJdwH29Ky8t+Yn3WeLzg=	30	128	1.2	2019-06-10 15:29:32
		Js9mSQ5X8GfxGiHDyNSEBzRCDIONrR0tet7eKc6ptCk=	27	439	1.2	2019-06-18 19:36:45
		w6mw66IFMUJDfNK5Y4RQDLCGX6MPgfNXIaY42EhURkM=	17	538	1.2	2019-06-18 19:36:45
		vYOXI2Z84mknj8GgTaOG/tyi9eAg0Kv8cTvqCPE3Jkg=	7	474	1.2	2019-06-18 19:36:45
19/06/19	Winrar	vYOXI2Z84mknj8GgTaOG/tyi9eAg0Kv8cTvqCPE3Jkg=	7	474	1.2	2019-06-18 19:36:45
19/06/19	Winrar	vYOXI2Z84mknj8GgTaOG/tyi9eAg0Kv8cTvqCPE3Jkg=	7	474	1.2	2019-06-18 19:36:45
24/06/19	RigEK	qmLSnN9s+6ZosKo1tV0sbdd6RjBKuJ4pkq66+7tRWHY=	35	531	1.2	2019-06-18 19:36:45
26/06/19	Targeting South Korea	w6mw66IFMUJDfNK5Y4RQDLCGX6MPgfNXIaY42EhURkM=	17	538	1.2	2019-06-18 19:36:45
25/06/19	Malspam – Booking	RJLY2jLnGa3qAJx5s3sIwl0flZjJFSxHjZgDYwHKaBI=	27	564	1.2	2019-06-24 15:53:35
01/07/19	RigEK	Zrui05IT0bzVjJv7WuNIq6PZyXjBMEStA2eSxQT8TjY=	22	607	1.2	2019-06-24 15:53:35

The fields from the table are the following:

1. Campaign Date
2. Type of Campaign
3. PK (public key inside the JSON configuration)
4. PID present in JSON configuration
5. SUB present in JSON configuration
6. Sodinokibi version
7. Date the master file of Sodinokibi is compiled

PID field identify the group has acquired the service Sodinokibi ransomware (RAAS). SUB field probably identify "SUBSCRIPTION" that is the period of validity of the service.

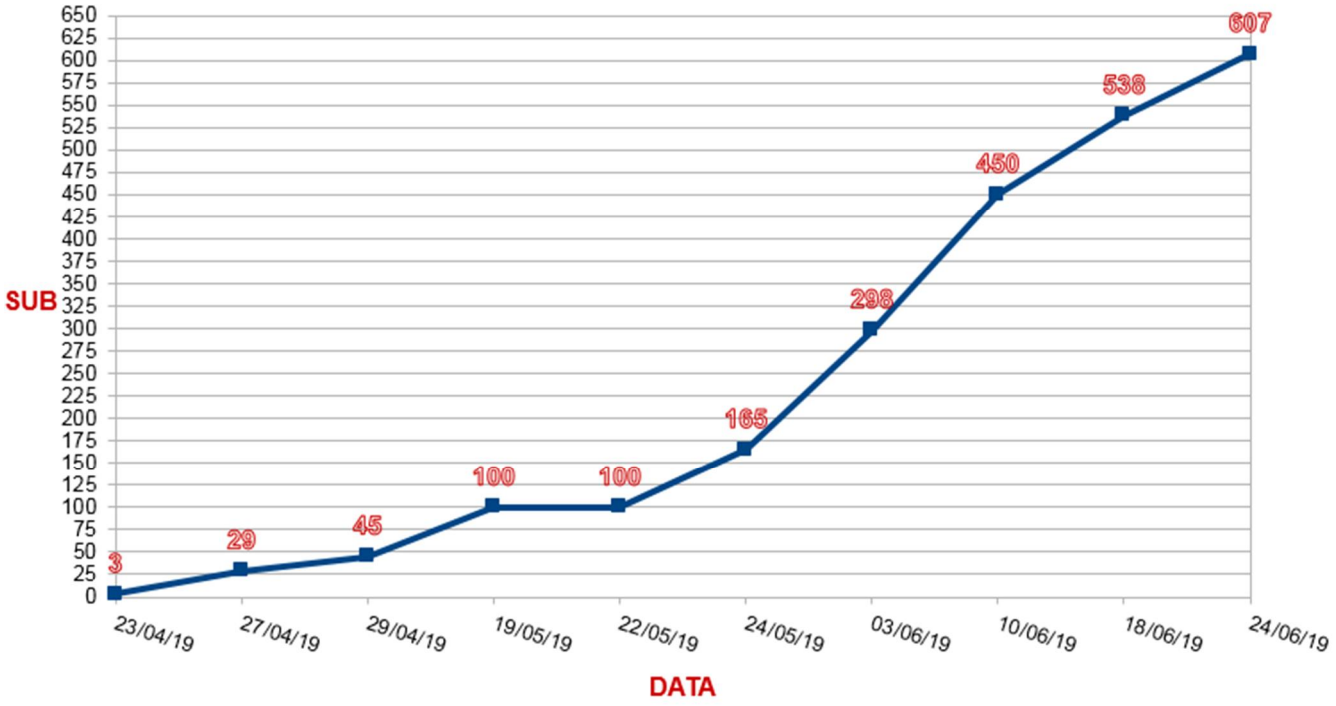
The pairs of PID & SUB with identical value have the same public key (PK), how we can see in the case of PID:7 and SUB: 3.

The campaign with PID 7 was the first to use Oracle Weblogic vulnerability to distribute the ransomware on 25 April 2019 (SUB:3), the same group seems to be associated with the Watering Hole attack campaign to distributor of WinRar in Italy on 19th June 2019 with a new SUB: 474.

As we can see, the group with PID: 7 has purchased more subscription periods. Using the three parameters PID-SUB-PK, one can identify the campaign associated with the same actor.

Until early July of this year, the PID 40 was the highest value, this suggests that there are at least 40different groups. The highest value of SUB was 607 which could indicate that at least 607 subscription periods have been purchased.

We compare in the graphic here below, the date of compilation of the malware and the SUB value present in json configuration. It is possible to see how the curve growth strongly suggesting that the Sodinokibi CryptoMalware is distributed with the "as-a-service" method.



Conclusion

The authors of Sodinokibi are individuals with a certain level of technical knowledge and probably this ransomware is not their first creation and it is actively developed.

This project is developed to be distributed with model RaaS (Ransomware-as-a-Service).

Sodinokibi ransomware uses for file encryption the algorithm Salsa20 with a key exchange method based on ECDH.

Sodinokibi operation spreads wide in the last month, through a different methods to distribute the ransomware via Malspam, RigEK, RDP attacks, etc. The attackers with the recent decision to shutting down GandCrab Ransomware operation left a hole, that seem to exploited by Sodinokibi.

IOC

MD5:

```
DB42F17991A7BA10218649B978D78674
E713658B666FF04C9863EBECB458F174
16863F6727BC5DD44891678EBCA492D2
FD3F3AF76D31D8F134E2E02463D89D29
6E543C13594F987A6051BC3D9456499F
CCFDE149220E87E97198C23FB8115D5A
FB68A02333431394A9A0CDBFF3717B24
692870E1445E372DDD82AEDD2D43F9B8
DB6D3A460DEDE97CA7E8C5FBFAEF3A72
48A673157DA3940244CE0DFB3ECB58E9
79F2341510D9FB5291AEFC3E69D18253
3DF42FA9732864A9755F5C8FB7ED456A
```

URL:

```
aplebzu47wgazapdqks6vrcv6zcnjppkbxbr6wketf56nf6aq2nmyoyd.onion
decryptor.top
```