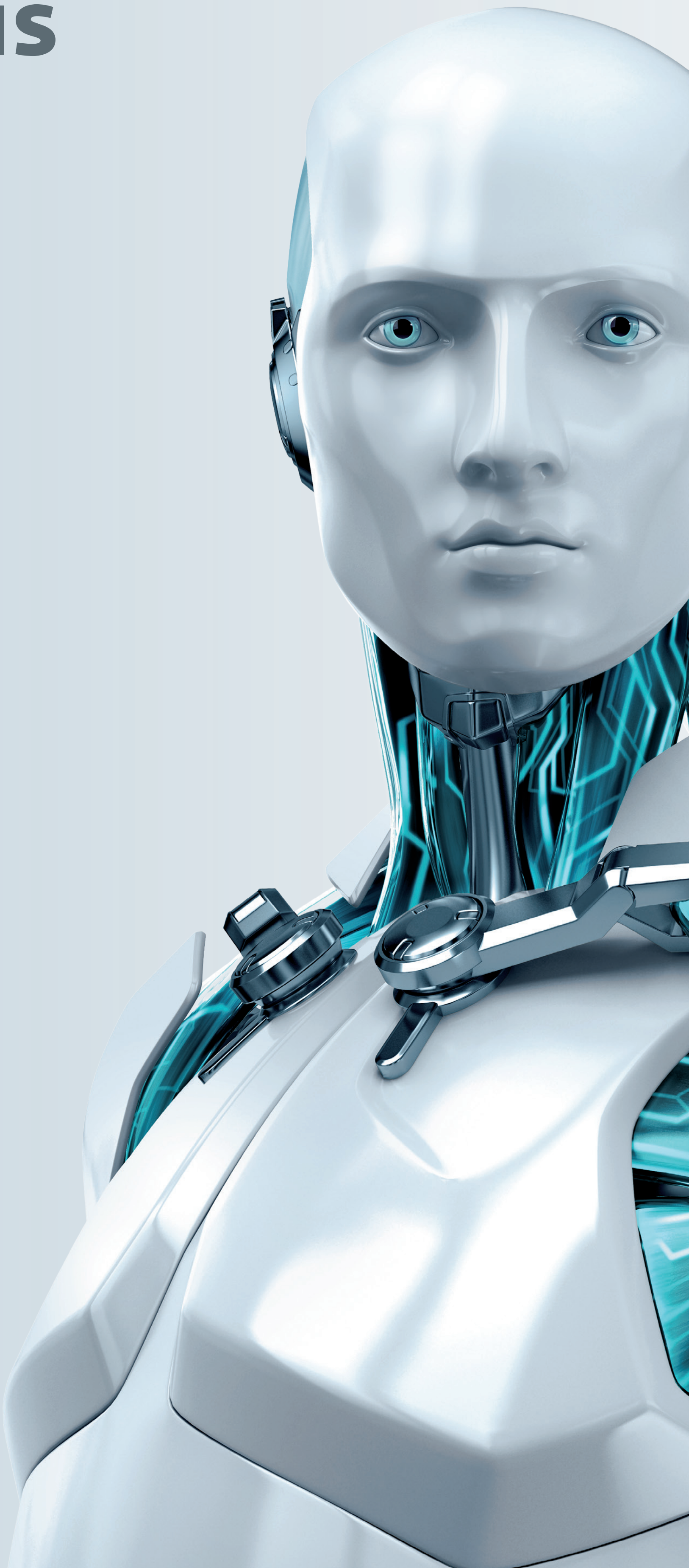
OceanLotus

Old techniques, new backdoor

ESET

March 2018



ENJOY SAFER TECHNOLOGY™

OceanLotus

Old techniques, new backdoor

ESET

March 2018

ENJOY SAFER TECHNOLOGY™

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

OCEANLOTUS: OLD TECHNIQUES, NEW BACKDOOR

The OceanLotus group, also known as APT32 and APT-C-00, is infamous for its campaigns targeting the eastern part of Asia. A great deal of research about this group was published last year, including papers such as those from CyberReason, a lengthy global view from FireEye and the watering-hole explanation from Volexity. We see that this group keeps updating their backdoors, infrastructure, and infection vectors.

OceanLotus continues its activity particularly targeting company and government networks in East-Asian countries. A few months ago, we discovered and analyzed one of their latest backdoors. Several tricks are being used to convince the user to execute the backdoor, to slow down its analysis and to avoid detection. These techniques will be discussed in detail in this blog post.

Distribution

Various methods are used to trick potential victims into running the malicious dropper. Our telemetry reveals that East-Asian countries such as Vietnam, the Philippines, Laos and Cambodia, are the most targeted.

Double extension and fake icon applications (Word, PDF, etc)

The droppers are probably attached to an email message. Some file names we have seen include:

- `Mi17 Technical issues - Phonesack Grp.exe`. Mi-17 is a common Russian helicopter.
- `Chi tiet don khieu nai gui saigontel.exe`, which translates from Vietnamese to "Details of the complaint sent to Saigontel". Saigontel is a telecommunication company in Vietnam.
- `Updated AF MOD contract - Jan 2018.exe`
- `remove_pw_Reschedule of CISD Regular Meeting.exe`
- `Sorchornor_with_PM_-_Sep_2017.exe`
- `20170905-Evaluation Table.xls.exe`
- `CV_LeHoangThing.doc.exe`. Fake résumé (CV) documents were also seen in Canada.

All these files have something in common: A decoy document is launched, but is password-protected. It's unclear whether this password is provided by any means, for example in the email it was attached to, or if the documents are simply not meant to work.

Fake installers

Several fake installers, claiming to be installers or updates for popular software were seen in watering hole campaigns. A good example of such an installer is the repackaged Firefox installer described by 360 Labs on Freebuf (Chinese language).

Another sample we saw had the name `RobototFontUpdate.exe`. It was also likely to have been distributed via compromised websites, but we do not yet have enough evidence to confirm this.

All the files described here, whether they were sent in an email or downloaded while visiting a compromised site, deliver the same (functionally equivalent) backdoor component. For the purpose of this article we will dissect the `RobototFontUpdate.exe` sample and show how it manages to execute its malicious payload on a system.

Technical analysis

The whole process of installation and execution relies heavily on multiple layers of obfuscation such as decryption of payloads, PE reconstruction and loading shellcode, and side-loading techniques. The last technique was previously described in a previous ESET research article about Korplug.

Execution flow overview

The attack is split in two parts: the dropper and backdoor launcher. Each step of each part of the process will be explained in detail in its respective section. The following two diagrams briefly summarize the general flow of execution of the malware.

The dropper part has the following flow of execution:

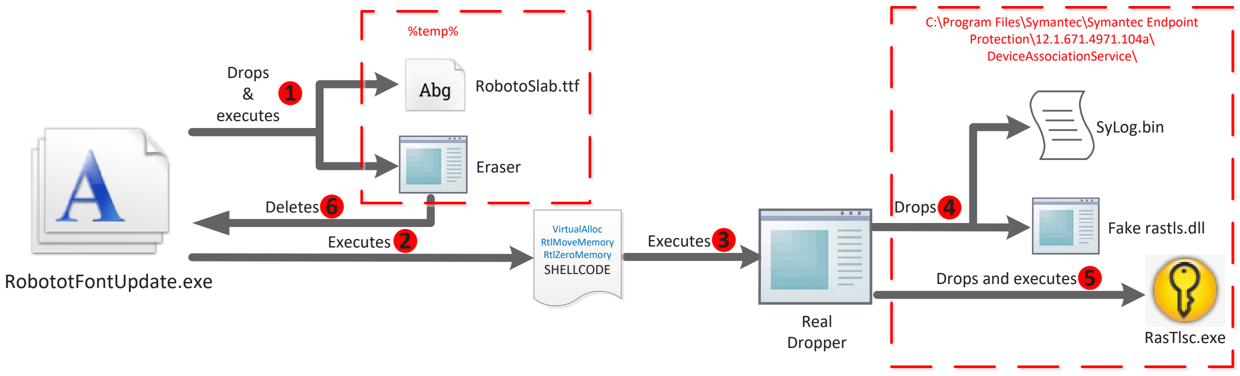


Figure 1 **Dropper execution flow**

The backdoor part has the following execution flow:

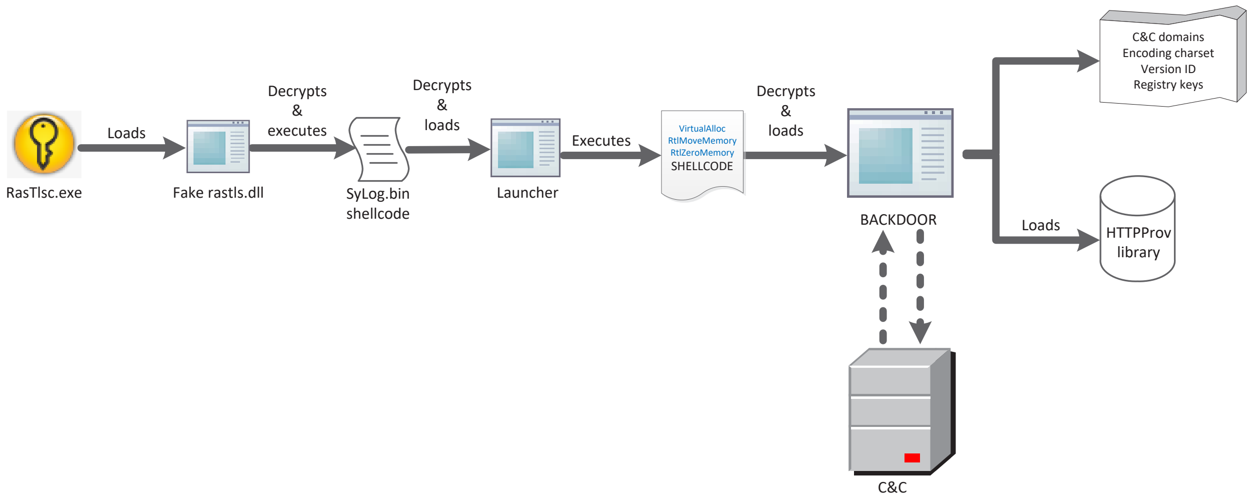
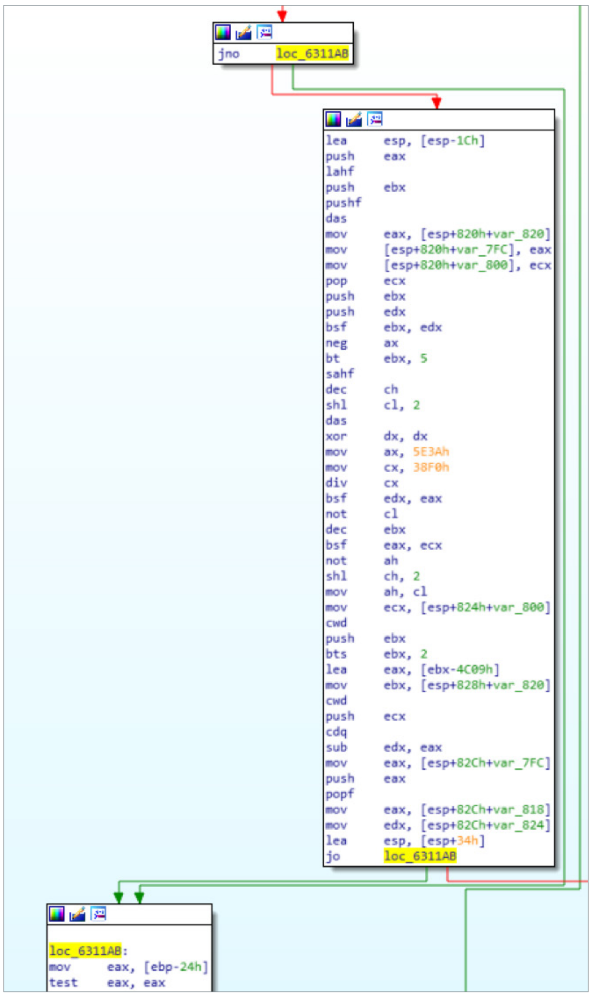


Figure 2 **Backdoor execution flow**

A quick word on control flow obfuscation

Almost all of these components are obfuscated. The obfuscation is based on pairs of complementary conditional jump instructions. Every form is used: `JZ/JNZ, JP/JNP, JO/JNO`, etc, each pair jumping to the same target. The sequence is interleaved with junk code, which makes use of the stack pointer, but does not change the conditional flag's value. This means it will always end up in the same branch. This causes problems during decompilation due to the use of positive stack pointer values.



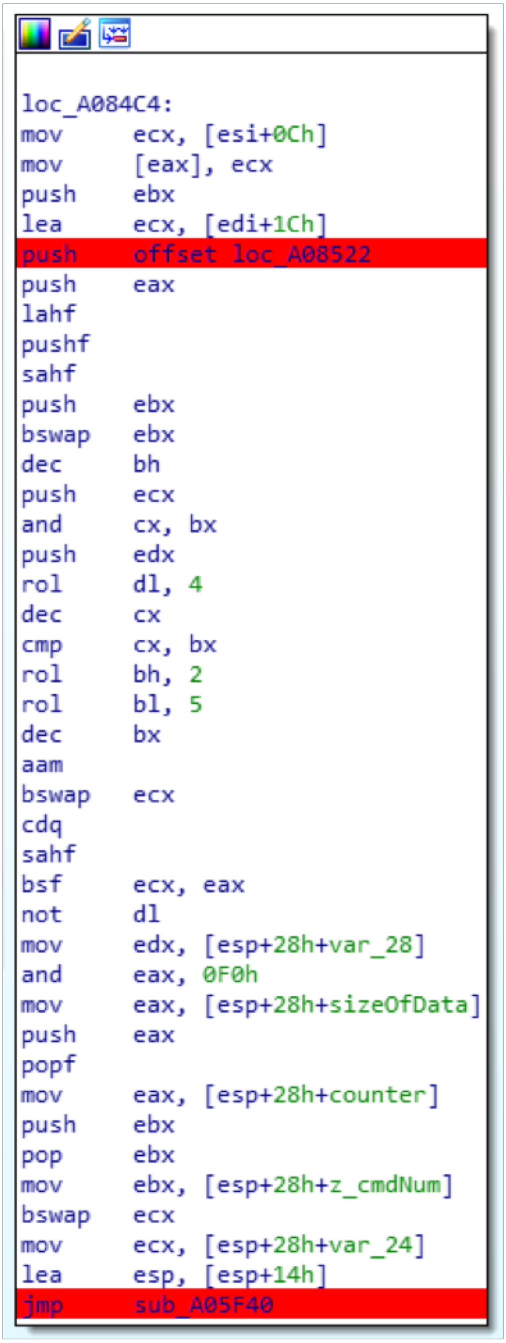
```
jno     loc_6311AB
```

```
lea     esp, [esp-1Ch]
push    eax
lahf
push    ebx
pushf
das
mov     eax, [esp+820h+var_820]
mov     [esp+820h+var_7FC], eax
mov     [esp+820h+var_800], ecx
pop     ecx
push    ebx
push    edx
bsf     ebx, edx
neg     ax
bt      ebx, 5
sahf
dec     ch
shl     cl, 2
das
xor     dx, dx
mov     ax, 5E3Ah
mov     cx, 38F0h
div     cx
bsf     edx, eax
not     cl
dec     ebx
bsf     eax, ecx
not     ah
shl     ch, 2
mov     ah, cl
mov     ecx, [esp+824h+var_800]
cwd
push    ebx
bts     ebx, 2
lea     eax, [ebx-4C09h]
mov     ebx, [esp+828h+var_820]
cwd
push    ecx
cdq
sub     edx, eax
mov     eax, [esp+82Ch+var_7FC]
push    eax
popf
mov     eax, [esp+82Ch+var_818]
mov     edx, [esp+82Ch+var_824]
lea     esp, [esp+34h]
jo      loc_6311AB
```

```
loc_6311AB:
mov     eax, [ebp-24h]
test    eax, eax
```

Figure 3 **Complementary conditional jump**

Moreover, some basic blocks push one address on the stack, then end with a `JMP/CALL` while other basic blocks push two addresses then end with a `RET` instruction. The second push is the function to call and the first one is the address of the next basic block to jump to. This creates basic blocks with no parents.



```
loc_A084C4:
mov     ecx, [esi+0Ch]
mov     [eax], ecx
push    ebx
lea     ecx, [edi+1Ch]
push    offset loc_A08522
push    eax
lahf
pushf
sahf
push    ebx
bswap   ebx
dec     bh
push    ecx
and     cx, bx
push    edx
rol     dl, 4
dec     cx
cmp     cx, bx
rol     bh, 2
rol     bl, 5
dec     bx
aam
bswap   ecx
cdq
sahf
bsf     ecx, eax
not     dl
mov     edx, [esp+28h+var_28]
and     eax, 0F0h
mov     eax, [esp+28h+sizeOfData]
push    eax
popf
mov     eax, [esp+28h+counter]
push    ebx
pop     ebx
mov     ebx, [esp+28h+z_cmdNum]
bswap   ecx
mov     ecx, [esp+28h+var_24]
lea     esp, [esp+14h]
jmp     sub_A05F40
```

Figure 4 **PUSH/JMP technique**

The resulting combination of these two obfuscation techniques creates "beautiful" graphs:

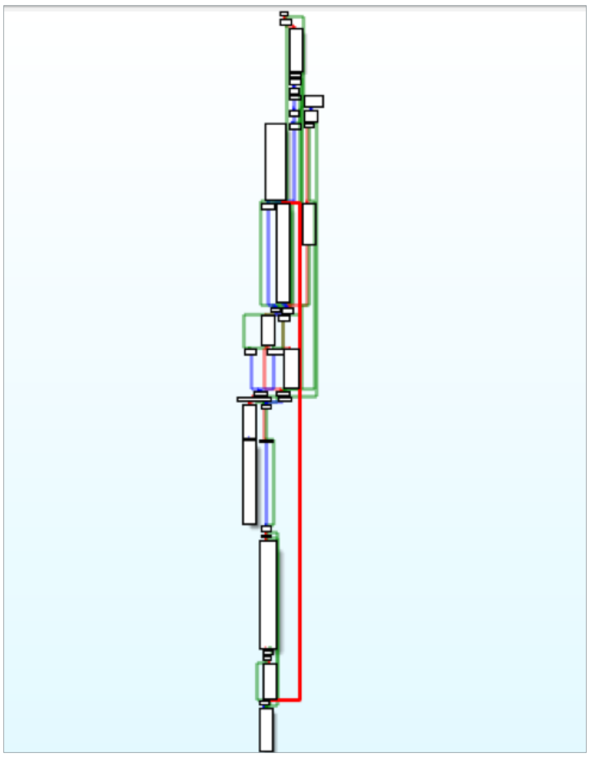


Figure 5 **Control flow obfuscation**

The junk code is pretty easy to spot and once the scheme is known, it can be ignored when analysing the samples.

The Dropper

Stage 1 The decoy document

Over the past few months, a number of decoy documents have been used. One of them was a fake TrueType font updater for the `Roboto Slab regular` font. This choice of font seems a bit odd since it does not support a lot of East-Asian languages.

Figure 6 **RobototFontUpdate icon**

When executed, this binary decrypts its resource (XOR with a 128-byte, hardcoded key) and decompresses the decrypted data (LZMA). The legitimate *RobotoSlab-Regular.ttf* (SHA1:`912895e6bb9e05af3a1e58a1da417e992a71a324`) file is written into the `%temp%` folder and run via Win32 API function `ShellExecute`.

The shellcode decrypted from the resource is executed. After its execution, the fake font updater drops another application whose sole purpose is to delete the dropper. This "eraser" application is dropped as `%temp%\[0-9].tmp.exe`.

Stage 2 The shellcode

The shellcode used is the same for every stage that uses shellcode

The shellcode is a custom PE loader. It recreates an executable in memory: it decrypts all the sections and computes the necessary relocations and other offsets. The shellcode retrieves three Windows API functions: `VirtualAlloc`, `RtlMoveMemory` and `RtlZeroMemory`.

The `RtlZeroMemory` function is heavily used to zero-out fields in the PE header. Relying on automatic memory dumping will not work since the MZ/PE headers are broken.

The shellcode calls the entry-point function of the decrypted PE and then the `DLLEntry` export function.

Stage 3 Real Dropper, `{103004A5-829C-418E-ACE9-A7615D30E125}.dll`

This executable decrypts its resource using the AES algorithm with CBC mode via the Windows API. The size of the hardcoded key is 256 bits. After decryption, the data are decompressed (LZMA algorithm).

If the process is running with administrator privileges, then the malware achieves persistence by creating a service, else the classic Windows "Run" registry key is used (`HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run;DeviceAssociationService;rastlsc.exe`).

If the dropper is executed with administrator privileges, then it tries to write the following files in the `C:\Program Files\Symantec\Symantec Endpoint Protection\12.1.671.4971.104a\DeviceAssociationService\` folder else it writes them in the `%APPDATA%\Symantec\Symantec Endpoint Protection\12.1.671.4971.104a\DeviceAssociationService\` folder:

- `rastlsc.exe` (SHA1:`2616da1697f7c764ee7fb558887a6a3279861fac`, copy of legitimate Symantec Network Access Control application, dot1xtra.exe)
- `SyLog.bin` (SHA1:`5689448b4b6260ec9c35f129df8b8f2622c66a45`, encrypted backdoor)
- `rastls.dll` (SHA1:`82e579bd49d69845133c9aa8585f8bd26736437b`,malicious DLL side-loaded by `rastlsc.exe`)

The path changes from sample to sample but the pattern is similar. Depending on its privileges, the malware drops the files in `%ProgramFiles%` or `%appdata%`. We've also seen:

- `\Symantec\CNG Key Isolation\`
- `\Symantec\Connected User Experiences and Telemetry\`
- `\Symantec\DevQuery Background Discovery Broker Tasks\`

These paths are used by various Symantec products.

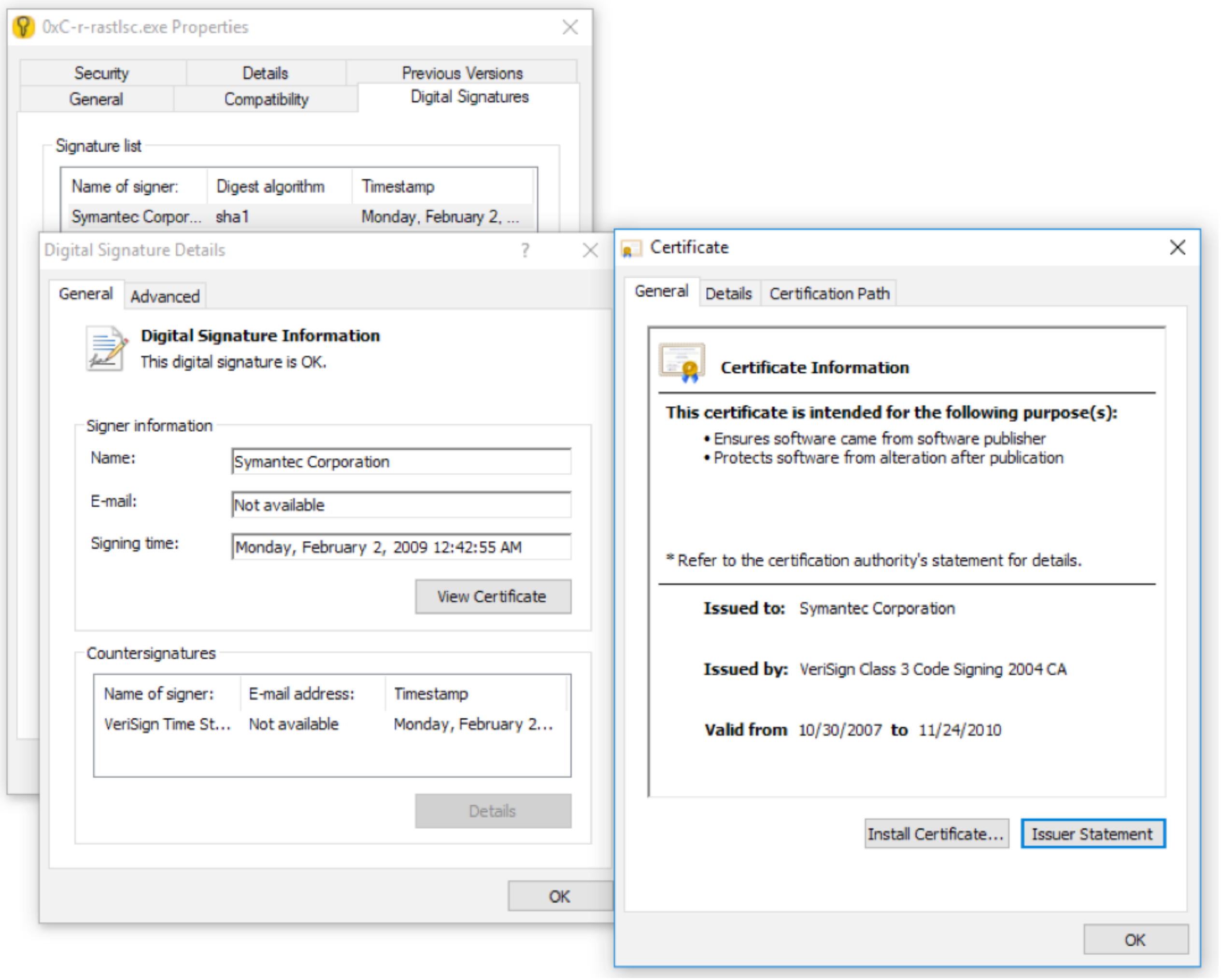
After achieving persistence and dropping the executable, the legitimate Symantec executable, `rastlsc.exe,` is executed using `CreateProcessW`.

We've also seen another version (`{BB7BDEC9-B59D-492E-A4AF-4C7B1C9E646B}.dll`), which executes `rastlsc.exe` with the parameter `krv`. Its meaning is discussed below.

Backdoor component: `rastlsc.exe` side-loading

The OceanLotus group uses an old and publicly known technique on one of the Symantec product's executable files. The trick, here, is to take advantage of the library loading process of a legitimate and signed executable by writing a malicious library inside the same folder. This way it will make malicious behaviors look legitimate because these actions are made by the trusted executable process.

As mentioned earlier, the legitimate executable `rastlsc.exe` is dropped and executed.
This executable imports the `rastls.dll` file, which in this case contains the malicious payload.


0xC-r-rastlsc.exe Properties

Security | Details | Previous Versions
General | Compatibility | Digital Signatures

Signature list

Name of signer:	Digest algorithm	Timestamp
Symantec Corpor...	sha1	Monday, February 2, ...

Digital Signature Details

General | Advanced

Digital Signature Information
This digital signature is OK.

Signer information
Name: Symantec Corporation
E-mail: Not available
Signing time: Monday, February 2, 2009 12:42:55 AM

Countersignatures

Name of signer:	E-mail address:	Timestamp
VeriSign Time St...	Not available	Monday, February 2...

Certificate

General | Details | Certification Path

Certificate Information

This certificate is intended for the following purpose(s):
• Ensures software came from software publisher
• Protects software from alteration after publication

* Refer to the certification authority's statement for details.

Issued to: Symantec Corporation

Issued by: VeriSign Class 3 Code Signing 2004 CA

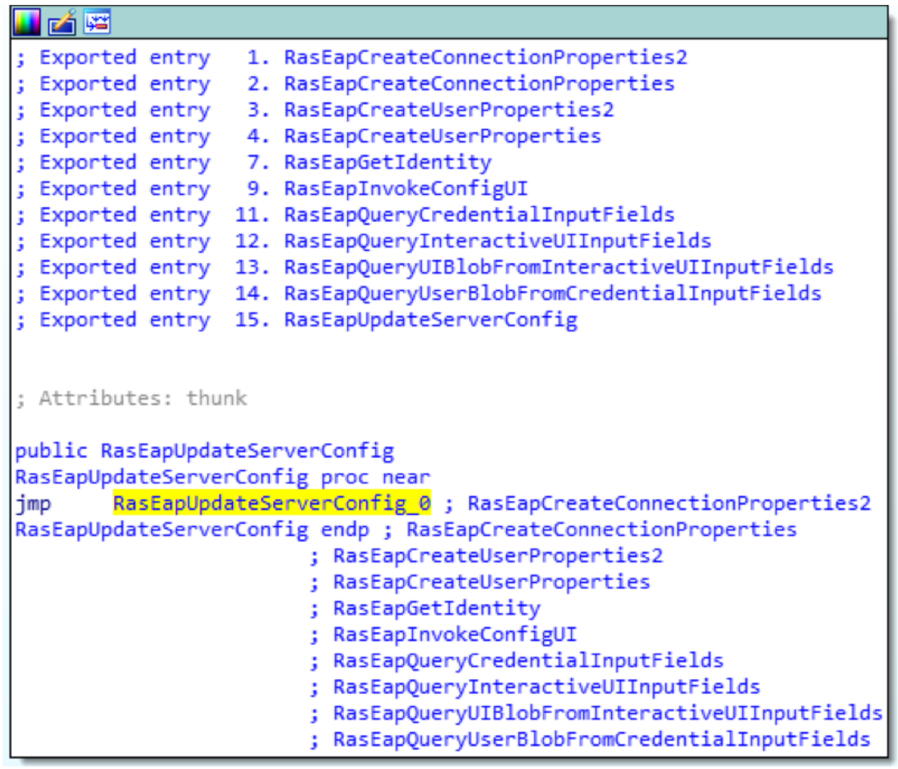
Valid from 10/30/2007 **to** 11/24/2010

Figure 7 **Symantec rastlsc.exe digital signature**

Side loading was also observed using other legitimate, signed executables including `mcoemcpy.exe` from McAfee, which loads `McUtil.dll`. This technique has also been used by PlugX before. This also got the attention of the Vietnam CERT (Vietnamese language).

Stage 1 Library side-loading, `rastls.dll`

The internal name of this dll is `{7032F494-0562-4422-9C39-14230E095C52}.dll` but we've seen other versions like `{5248F13C-85F0-42DF-860D-1723EEAA4F90}.dll`. All exported functions lead to the execution of the same function.



```
; Exported entry   1. RasEapCreateConnectionProperties2
; Exported entry   2. RasEapCreateConnectionProperties
; Exported entry   3. RasEapCreateUserProperties2
; Exported entry   4. RasEapCreateUserProperties
; Exported entry   7. RasEapGetIdentity
; Exported entry   9. RasEapInvokeConfigUI
; Exported entry  11. RasEapQueryCredentialInputFields
; Exported entry  12. RasEapQueryInteractiveUIInputFields
; Exported entry  13. RasEapQueryUIBlobFromInteractiveUIInputFields
; Exported entry  14. RasEapQueryUserBlobFromCredentialInputFields
; Exported entry  15. RasEapUpdateServerConfig


; Attributes: thunk

public RasEapUpdateServerConfig
RasEapUpdateServerConfig proc near
jmp     RasEapUpdateServerConfig_0 ; RasEapCreateConnectionProperties2
RasEapUpdateServerConfig endp ; RasEapCreateConnectionProperties
                        ; RasEapCreateUserProperties2
                        ; RasEapCreateUserProperties
                        ; RasEapGetIdentity
                        ; RasEapInvokeConfigUI
                        ; RasEapQueryCredentialInputFields
                        ; RasEapQueryInteractiveUIInputFields
                        ; RasEapQueryUIBlobFromInteractiveUIInputFields
                        ; RasEapQueryUserBlobFromCredentialInputFields
```

Figure 8 **All rasltls.dll exports lead to the same function**

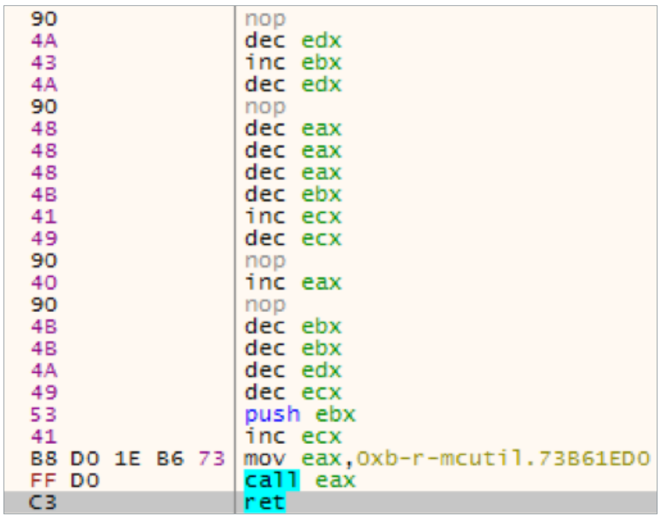
This export tries to read the `SyLog.bin` file located inside the same folder. Other versions tried to open the file `OUTLFLTR.DAT`. If that file exists, it is decrypted using AES in CBC mode with a hardcoded, 256-bit key and then decompressed (LZMA compression).

The `McUtil.dll` variant uses a different technique. At first glance, it looks as if like the main function does nothing malicious, but in fact it replaces the `.text` section of the legitimate `mcoemcpy.exe`, a side-loaded binary. It generates shellcode whose purpose is to call the function reading the encrypted stage-two shellcode in the `mcscentr.adf` file.

The following pseudocode is used to create the shellcode:

```
x = False
i = 0
buff = genRandom()
opc1 = [0x58,0x59,0x5a,0x5b]
opc2 = [0x50,0x51,0x52,0x53,0x54,0x55,0x56,0x57]
opc3 = [0x90,0x40,0x41,0x42,0x43,0x44,0x45,0x46,0x47,0x48,
        0x49,0x4a,0x4b]
while i < len(buff):
        currentChar = buff[i]
        if currentChar < 0xc8:
                buff[i] = opc1[currentChar % len(opc1)]
        else:
                if x:
                        buff[i] = opc2[currentChar % len(opc2)]
                else:
                        buff[i] = opc3[currentChar % len(opc3)]
                x = x == False
        i+=1
```

The result is the following assembly listing:



```
90                nop
4A                dec edx
43                inc ebx
4A                dec edx
90                nop
48                dec eax
48                dec eax
48                dec eax
4B                dec ebx
41                inc ecx
49                dec ecx
90                nop
40                inc eax
90                nop
4B                dec ebx
4B                dec ebx
4A                dec edx
49                dec ecx
53                push ebx
41                inc ecx
B8 D0 1E B6 73    mov eax,0xb-r-mcutil.73B61ED0
FF D0             call eax
C3                ret
```

Figure 9 **Generated shellcode**

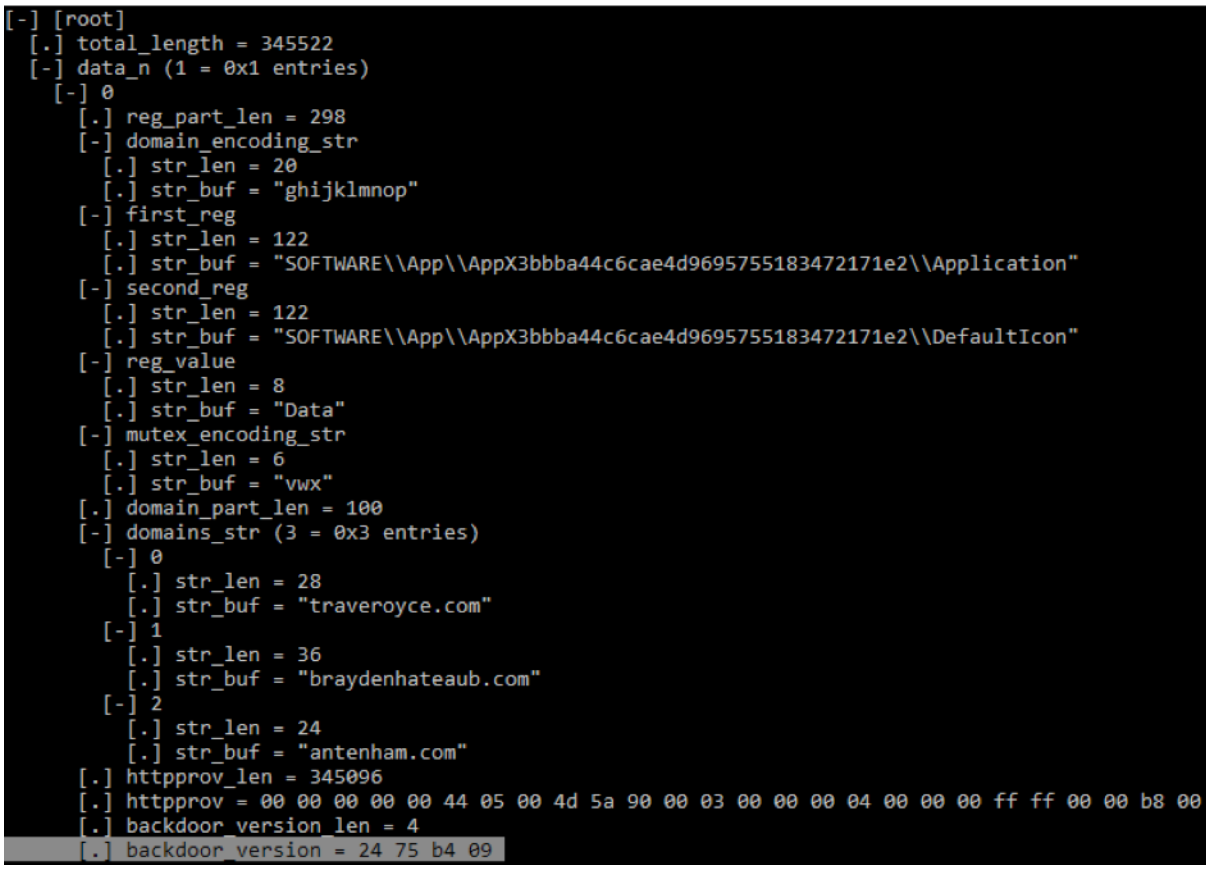
Stage 2 to Stage 4 Shellcode, Launcher and Shellcode again

The shellcode decrypts and loads the library `{E1E4CBED-5690-4749-819D-24FB660DF55F}.dll`. The library retrieves its resource and tries to start the service "DeviceAssociationService". The decrypted data also contains shellcode. The latter decrypts the final layer: the backdoor.

The variant `{92BA1818-0119-4F79-874E-E3BF79C355B8}.dll` checks whether `rastlsc.exe` was executed with `krv` as the first parameter. If so, then a job is created and `rastlsc.exe` is executed again, but without the parameter.

Stage 5 The backdoor, `{A96B020F-0000-466F-A96D-A91BBF8EAC96}.dll`

The malware first tries to retrieve its resource and decrypt it using RC4. The decrypted resource contains an interesting piece of information, used to configure the backdoor. The format of this configuration is straightforward to reverse. Using Kaitai struct and its structure dumper, the following representation can be displayed:



```
[-] [root]
  [.] total_length = 345522
  [-] data_n (1 = 0x1 entries)
    [-] 0
      [.] reg_part_len = 298
      [-] domain_encoding_str
        [.] str_len = 20
        [.] str_buf = "ghijklmnop"
      [-] first_reg
        [.] str_len = 122
        [.] str_buf = "SOFTWARE\\App\\AppX3bbba44c6cae4d9695755183472171e2\\Application"
      [-] second_reg
        [.] str_len = 122
        [.] str_buf = "SOFTWARE\\App\\AppX3bbba44c6cae4d9695755183472171e2\\DefaultIcon"
      [-] reg_value
        [.] str_len = 8
        [.] str_buf = "Data"
      [-] mutex_encoding_str
        [.] str_len = 6
        [.] str_buf = "vwx"
      [.] domain_part_len = 100
      [-] domains_str (3 = 0x3 entries)
        [-] 0
          [.] str_len = 28
          [.] str_buf = "traveroyce.com"
        [-] 1
          [.] str_len = 36
          [.] str_buf = "braydenhateaub.com"
        [-] 2
          [.] str_len = 24
          [.] str_buf = "antenham.com"
      [.] httpprov_len = 345096
      [.] httpprov = 00 00 00 00 00 44 05 00 4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00 b8 00
      [.] backdoor_version_len = 4
      [.] backdoor_version = 24 75 b4 09
```

Figure 10 **Configuration structure**

NOTES: *except for the string domain_encoding_str and the httpprov library, all these data change from one sample to another. The registry keys are almost the same but they follow a similar pattern:* `\HKCU\SOFTWARE\Classes\AppX[a-f0-9]{32}`*, which is quite generic.*

The malware retrieves the first 10 bytes of the username (UTF-16), XORs it with the 3-letter UTF-16 `mutex_encoding_str` string, and encodes it in hex. The result is used as a mutex name. For instance, for a user whose name starts with `abc` and the key being `vwx`, the following mutex will be created: `\Sessions\1\BaseNamedObjects\170015001b`.

The backdoor includes a PE loader that loads the library `HTTPProv.dll` in memory, calls its entry-point and then calls the export function named `CreateInstance`.

Communication

The backdoor uses a classic TCP communication protocol over port `25123`. In order to retrieve the server IP address, the backdoor first creates a particular DNS query.

The malware chooses between one of the three domains from the configuration and adds a custom sub-domain generated using two values. The first value is the computer name up to a length of 16 bytes. The second value is the 4-byte version ID. The following Python 2 code implements the encoding algorithm :

```
letters=domain_encoding_str # “ghijklmnop”
hex_pc_name=pc_name.encode(“UTF-16LE”).encode(“hex”)
s=’’
for c in hex_pc_name:
if 0x2f < ord(c) < 0x3a:
       s+=letters[ord(c) - 0x30]
else:
       s+=c
```

For instance, if the computer name is `random-pc` and the version ID is 0x0a841523 then the following domain could be created:

`niggmhggmeggmkggmfggmdggidggngggmjgg.ijhlokga.dwarduong[.]com`

The following regular expression could be used to flag a C&C server for this backdoor:

`[ghijklmnopabcdef]{4-60}\.[ghijklmnopabcdef]{8}\.[a-z]+\.[a-z]+`

If an IP address is resolved for this particular domain, then the malware tries to establish a connection on TCP port `25123`. Each sample has three different domain names it can use to find its C&C server.

All communication is encrypted using RC4 and compressed with LZMA. It is possible to decrypt the traffic because the key is prepended to the packets. The format is:

`[RC4 key (4 bytes)][encrypted data]`

Each byte of the key is generated using the `rand` function. Once the packet is decrypted and decompressed, the data follow the format:

`[dw:unknown][dw:unknown][dw:command number][dw:size of data][dw:unknown][dw:data]`

The first time the client connects to the server, a UUID is returned and used as a session ID. The latter is stored in the registry key as binary data: `HKCU\SOFTWARE\Classes\AppXc52346ec40fb4061ad96be0e6cb7d16a\DefaultIcon`

As mentioned earlier, the backdoor also contains a library called `HTTPprov`. This library is an alternative way, as a backup, to communicate with the server as a backup. This DLL sends a POST request over the HTTP protocol to communicate. It also supports HTTPS and the usage of a SOCKS5, SOCKS4a or SOCKS4 proxy. The library is statically linked with `libcurl`.

Once its initialization is done, the following registry key is created to instruct the backdoor to use HTTP in future communication with the C&C server: `HKCU\SOFTWARE\Classes\CLSID{E3517E26-8E93-458D-A6DF-8030BC80528B}`.

A generic user agent is used: `Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.0; Trident/4.0)`.

The most distinctive characteristic of this library is the custom URI encoding algorithm. The resource part of the URI is created using the following pseudocode:

```
buffEnd = ((DWORD)genRand(4) % 20) + 10 + buff;
while (buff < buffEnd){
        b=genRand(16);
        if (b[0] - 0x50 > 0x50)
                t=0;
        else
                *buf++= UPPER(vowels[b[1] % 5]);
        v=consonants[b[1]%21]);
        if (!t)
                v=UPPER(v);
        *buff++= v;
        if (v!='h' && b[2] - 0x50 < 0x50)
                *buff++= 'h';
        *buff++= vowels[b[4] % 5];
        if (b[5] < 0x60)
                *buff++= vowels[b[6] % 5];
        *buff++= consonants[b[7] % 21];
        if (b[8] < 0x50)
                *buff++= vowels[b[9] % 5];
        *buff++= '-';
};
*buff='\0';
```

NOTE: *For clarity, the length-checking part has been removed from the snippet.*

From this generated string, two numbers are computed based on the custom checksum to obtain a URI:

```
checksum=crc32(buff)
num2=(checksum >> 16) + (checksum & 0xffff) * 2
num1=(num2 ^ 1) & 0xf
URL=GENERATED_DOMAIN+ "/" + num1 + "/" + num2 + "-" + buff
```

By adding the URI generator of the `HTTPprov` library, the following URL could be generated:

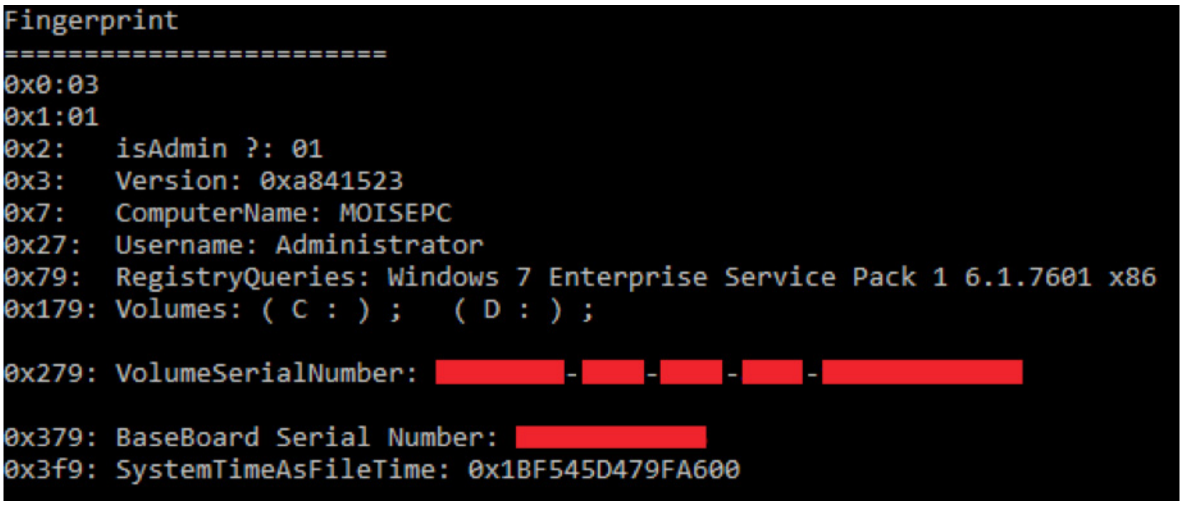
`hXXp://niggmhggmeggmkggmfggmdggidggngggmjgg.ijhlokga.aisicoin[.]com/13/139756-Ses-Ufali-L`

Commands

After receiving its SESSIONID, the backdoor does a fingerprint of the system. The packet is built in this fashion:

Offset in the packet	Description
0x000	byte: value varies in each version
0x001	0x01: hardcoded byte
0x002	bool: is elevation token present
0x003	dword: version ID
0x007	string (UTF-16), computer name (0x20 bytes max)
0x027	string (UTF-16), user name
0x079	registry query result of the `HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion` values: `ProductName, CSDVersion, CurrentVersion, ReleaseId, CurrentBuildNumber` and the result of the call to `IsWow64Process (x86|x64)`
0x179	Following format string "%s(%s);" replaced with (`GetVolumeInformationW:VolumeNameBuffer`),`VolumePathNames`
0x279	PhysicalDrive deviceIOControl 0x2D1400 (IOCTL_STORAGE_QUERY_PROPERTY) (VolSerialNumber)
0x379	wmi `SELECT SerialNumber FROM Win32_BaseBoard`
0x3f9	GetSystemTimeAsFileTime
0x400	bool: unknown
0x401	dword: obtained from the decryption of the resource

Here's an example of a fingerprint of a system:



```
Fingerprint
========================
0x0:03
0x1:01
0x2:   isAdmin ?: 01
0x3:   Version: 0xa841523
0x7:   ComputerName: MOISEPC
0x27:  Username: Administrator
0x79:  RegistryQueries: Windows 7 Enterprise Service Pack 1 6.1.7601 x86
0x179: Volumes: ( C : ) ;   ( D : ) ;

0x279: VolumeSerialNumber: [redacted]-[redacted]-[redacted]-[redacted]-[redacted]

0x379: BaseBoard Serial Number: [redacted]
0x3f9: SystemTimeAsFileTime: 0x1BF545D479FA600
```

Figure 11 **System fingerprint**

This is a full-featured backdoor that offers its operators many capabilities, such as file, registry and process manipulation, loading additional components, and performing a system fingerprint. Here's the list of supported commands:

Command number	Description
0	Fingerprint
1	Sets the session ID
2	Creates a process and gets the output (using pipes)
3	Sets the connection retry counter
4	Delays polling time
5	Reads a file or registry key and computes the MD5
6	Creates a process
7	Creates a file, a registry entry or a stream in memory
8	Writes to the registry
9	Queries the registry
10	Searches for files on the system
11	Moves files to another directory
12	Deletes files from the disk
13	List the drives mapped on the system using the `GetLogicalDriveStringW` function
14	Creates directory
15	Deletes directory
16	Reads a file from an offset
17	Calls the PE Loader (switch to `HTTPprov` communication)
18	[Unknown]
19	0: Query a value from the registry, 1:Drop and execute a program
20	Sets an environment variable
21	Runs shellcode in a new thread
22	Retrieves an environment variable
+23 in new version	Restarts itself if the "APPL" environment variable doesn't exist

CONCLUSION

Once again, OceanLotus shows that the team is active and continues to update its toolset. This also demonstrates its intention to remain hidden by picking its targets, limiting the distribution of their malware and using several different servers to avoid attracting attention to a single domain or IP address. The encryption of the payload, together with the side-loading technique – despite its age – is a good way to stay under the radar, since the malicious activities look like they come from the legitimate application.

IoCs

Samples

Table 1. **Initial Dropper**

SHA1-1	Filename	ESET Detection Name
fdcb35cd9cb8dc1474cbcdf1c9bb03200dcf3f18	RobototFontUpdate.exe	Win32/TrojanDropper. Agent.RUI
a40ee8ff313e59aa92d48592c494a4c3d81449af	Firefox Installer.exe	Win32/TrojanDropper. Agent.RUI
c2eb1033bc01ab0fd732a7ba4967be02c0690bf0	20170905-Evaluation Table.xls.exe	Win32/TrojanDropper. Agent.RUI
d35695f2366a43628231e73ffa83ca106306a8fa	CV_LeHoangThing.doc.exe	Win32/TrojanDropper. Agent.RUI
fe0161fb8a26a0bf4afad746c7ebf89499dcd3a7	Chi tiet don khieu nai gui saigontel.exe	Win32/TrojanDropper. Agent.RUI
032ef58b7978d079287874044dc516af624ae5f5	Mi17 Technical issues - Phonesack Grp.exe	Win32/TrojanDropper. Agent.RUI
2a387d7d47a63d6e47d9cc92d3dc69a53816c2c0	Sorchornor_with_PM_-_ Sep_2017.exe	Win32/TrojanDropper. Agent.RUI
7105caa6d4fd8a2c67523d385277528e556ae4f6	Updated AF MOD contract - Jan 2018.exe	Win32/TrojanDropper. Agent.RUI
f96bcd875836da89800912de1e557891697c7cf4	remove_pw_Reschedule of CISD Regular Meeting.ex_	Win32/TrojanDropper. Agent.RUI

Table 2. **Sideloaded libraries**

SHA1-1	Filename	ESET Detection Name
82e579bd49d69845133c9aa8585f8bd26736437b	rastls.dll	Win32/Salgorea.BD
202fb56edb2fb542e05c845d62ffbdcfbebed9ec	McUtil.dll	Win32/Korplug.MK

Network

IP addresses

```
46.183.220.81
46.183.220.82
46.183.222.82
46.183.222.83
46.183.222.84
46.183.223.106
46.183.223.107
74.121.190.130
74.121.190.150
79.143.87.230
```

```
79.143.87.233
84.38.132.226
84.38.132.227
149.56.180.243
158.69.100.199
164.132.45.67
192.34.109.163
192.34.109.173
198.50.191.194
198.50.191.195
198.50.234.96
198.50.234.111
```

Domain names

```
adineohler[.]com
aisicoin[.]com
alicervois[.]com
anessallie[.]com
antenham[.]com
arinaurna[.]com
arkoimmerma[.]com
aulolloy[.]com
avidilleneu[.]com
avidsontre[.]com
aximilian[.]com
biasatts[.]com
braydenhateaub[.]com
carosseda[.]com
chascloud[.]com
dreyoddu[.]com
dwarduong[.]com
eckenbaue[.]com
eighrimeau[.]com
errellawle[.]com
erstin[.]com
frahreiner[.]com
hieryells[.]com
hristophe[.]com
ichardt[.]com
icmannaws[.]com
iecopeland[.]com
irkaimboeuf[.]com
jamedalue[.]com
jamyer[.]com
jeanessbinder[.]com
jeffreyue[.]com
keoucha[.]com
laudiaouc[.]com
lbertussbau[.]com
loridanase[.]com
marrmann[.]com
meroque[.]com
moureuxacv[.]com
```

`myolton[.]com`
`nasahlaes[.]com`
`ntjeilliams[.]com`
`omasicase[.]com`
`onnaha[.]com`
`onteagle[.]com`
`orinneamoure[.]com`
`orresto[.]com`
`orrislark[.]com`
`rackerasr[.]com`
`rcuselynac[.]com`
`sanauer[.]com`
`stopherau[.]com`
`tefanie[.]com`
`tefanortin[.]com`
`tephens[.]com`
`traveroyce[.]com`
`tsworthoa[.]com`
`ucaargo[.]com`
`ucairtz[.]com`
`urnage[.]com`
`venionne[.]com`
`virginiaar[.]com`

Host-based

Windows registry keys

`HKCU\SOFTWARE\Classes\AppXc52346ec40fb4061ad96be0e6cb7d16a\`
`HKCU\SOFTWARE\Classes\AppX3bbba44c6cae4d9695755183472171e2\`
`HKCU\SOFTWARE\Classes\CLSID{E3517E26-8E93-458D-A6DF-8030BC80528B}\`
`HKCU\SOFTWARE\Intel\Display\igfxcui\igfxtray\;[NUMBER];[DWORD]` (set by command #19)

Last IoC update: 2018-02-22 18:30:46 Eastern Standard Time