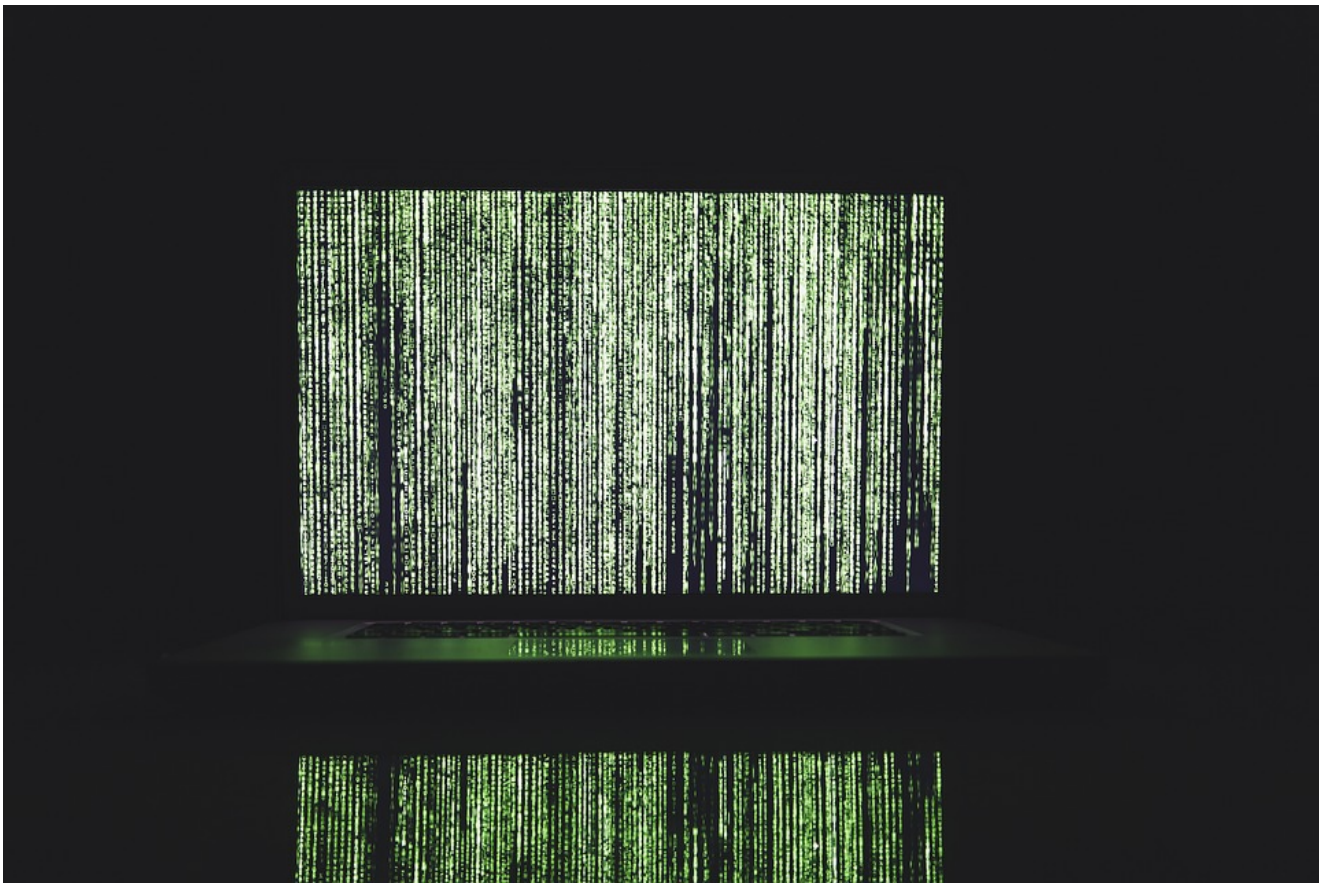
Ramnit – in-depth analysis | CERT Polska

cert.pl/en/news/single/ramnit-in-depth-analysis/



If we look on Ramnit's history, it's hard to exactly pin down which malware family it actually belongs to. One thing is certain, it's not a new threat. It emerged in 2010, transferred by removable drives within infected executables and HTML files.

A year later, a more dangerous version was released. It contained a part of recently leaked Zeus source code, which allowed Ramnit to become a banking trojan.

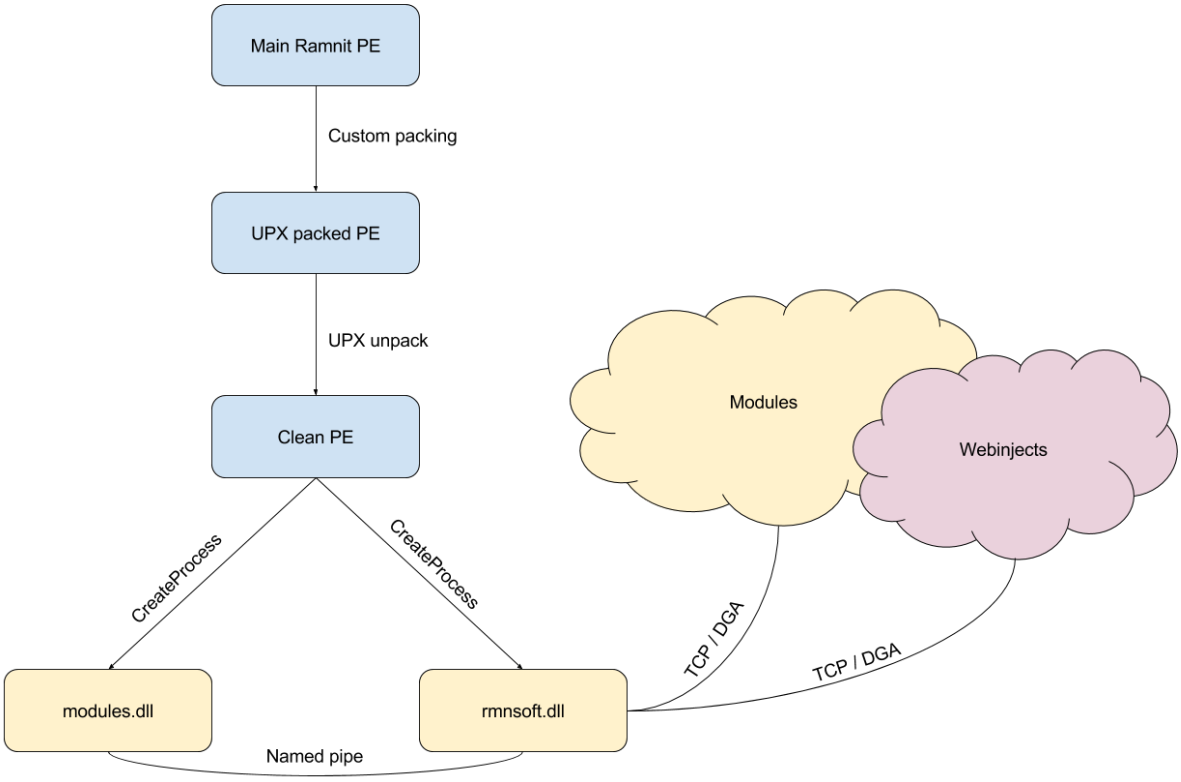
These days, it has become much more sophisticated by utilizing a number of malicious activities including:

- Performing Man-in-the-Browser attacks
- Stealing FTP credentials and browser cookies
- Using DGA (Domain Generation Algorithm) to find the C&C (Command and Control) server
- Using privilege escalation
- Adding AV exceptions
- Uploading screenshots of sensitive information

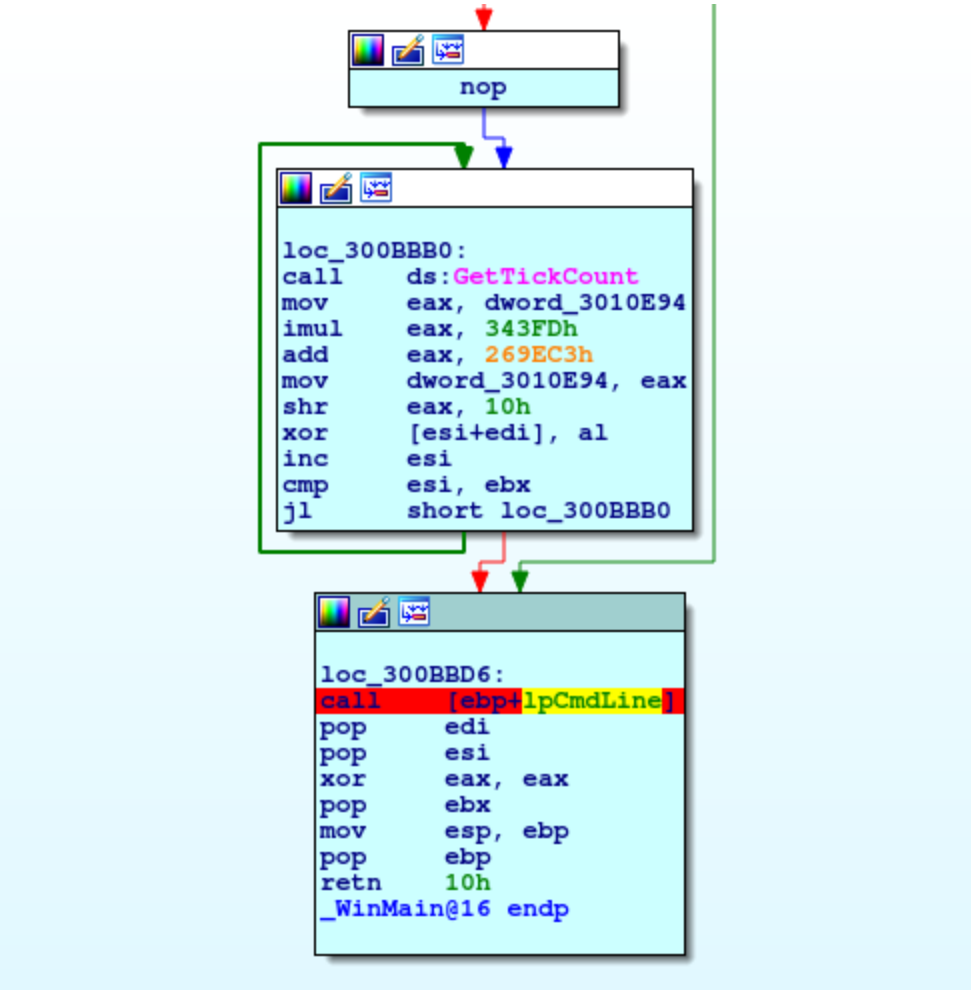
Despite Europol's shut down of 300 C&C servers in 2015, it's still going strong, recently being distributed by RIG EK via seamless gates.

Executable's analysis

The main binary is packed like a matryoshka – a custom packing method first and then UPX.



Despite being encrypted, extracting the binary from the packer is pretty straight-forward – all one needs to do is to set a breakpoint right after the binary decrypts the code and before it jumps into it.



And if we now navigate to the newly unpacked code section we'll find the binary right after the loader assembly:

```
debug021:0020685A mov     [ebp-0D4h], ecx
debug021:00206860 lea     edx, [ebp-0E0h]
debug021:00206866 push    edx
debug021:00206867 call    near ptr unk_2058F0
debug021:0020686C add     esp, 4
debug021:0020686F mov     esp, ebp
debug021:00206871 pop     ebp
debug021:00206872 retn
debug021:00206872 ; ---------------------------------------------------------------------------
debug021:00206873 db 0CCh ; ?
debug021:00206874 db 0CCh ; ?
debug021:00206875 db 0CCh ; ?
debug021:00206876 db 0CCh ; ?
debug021:00206877 db 0CCh ; ?
debug021:00206878 db 0CCh ; ?
debug021:00206879 db 0CCh ; ?
debug021:0020687A db 0CCh ; ?
debug021:0020687B db 0CCh ; ?
debug021:0020687C db 0CCh ; ?
debug021:0020687D db 0CCh ; ?
debug021:0020687E db 0CCh ; ?
debug021:0020687F db 0CCh ; ?
debug021:00206880 db  4Dh ; M
debug021:00206881 db  5Ah ; Z
debug021:00206882 db  90h ; ?
debug021:00206883 db    0
debug021:00206884 db    3
debug021:00206885 db    0
debug021:00206886 db    0
debug021:00206887 db    0
debug021:00206888 db    4
debug021:00206889 db    0
debug021:0020688A db    0
debug021:0020688B db    0
debug021:0020688C db 0FFh
debug021:0020688D db 0FFh
```

The unpacked binary (after UPX decompression) consists of 3 general functions:

- ApplyExploit
- CheckBypassed
- start

ApplyExploit

If the current user is not already an admin and the process is not running with admin privileges it tries to perform privilege escalation.

Malware contains exploits for CVE-2013-3660 (patched in MS13-053) and CVE-2014-4113 (patched in MS14-058) vulnerabilities, however before it actually tries to run the payload, registry checks are performed to make sure that the host system is indeed vulnerable to said CVEs:

If the exploits succeed or the program is already running with high privileges, a “TRUE” value is stored in a hardcoded random-looking registry key: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\jfghdug_ooetvtgk, which is later used in the CheckBypassed function.

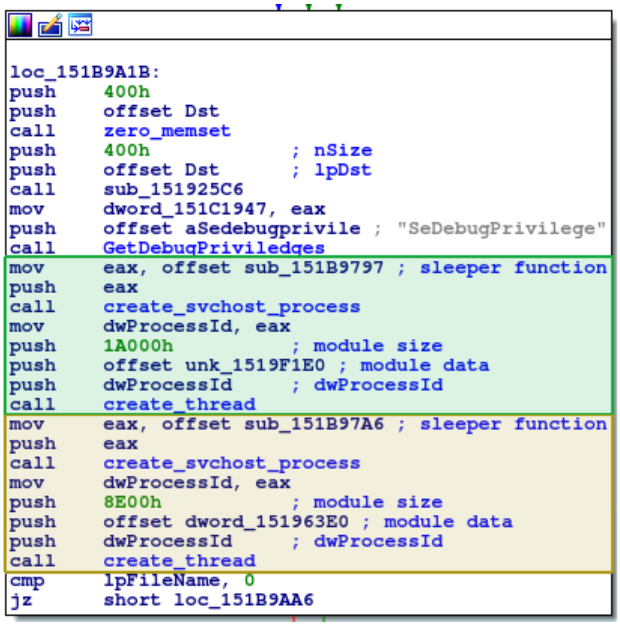
CheckBypassed

This function checks if previously mentioned registry key is set. If not and process has admin privileges, updates it. Assuming the exploit has worked, Ramnit then adds registry keys to evade Windows’ security systems detection (see Obfuscation/Evasion):

start routine

The routine coordinates ApplyExploit and CheckBypassed – if they both run successfully it creates two svchost.exe processes and writes rmnsoft.dll and modules.dll into them respectively.

Important detail: the binary executes CheckBypassed before ApplyExploit, so the binary has to be executed again in order to make any further progress. This trick outsmarts many single-run malware analysis systems, such as Cuckoo.



```
loc_151B9A1B:
push    400h
push    offset Dst
call    zero_memset
push    400h            ; nSize
push    offset Dst      ; lpDst
call    sub_151925C6
mov     dword_151C1947, eax
push    offset aSedebugprivile ; "SeDebugPrivilege"
call    GetDebugPrivileges
mov     eax, offset sub_151B9797 ; sleeper function
push    eax
call    create_svchost_process
mov     dwProcessId, eax
push    1A000h          ; module size
push    offset unk_1519F1E0 ; module data
push    dwProcessId     ; dwProcessId
call    create_thread
mov     eax, offset sub_151B97A6 ; sleeper function
push    eax
call    create_svchost_process
mov     dwProcessId, eax
push    8E00h           ; module size
push    offset dword_151963E0 ; module data
push    dwProcessId     ; dwProcessId
call    create_thread
cmp     lpFileName, 0
jz      short loc_151B9AA6
```

Module 1 - rmnsoft.dll

Module 2 - modules.dll

Static config

Ramnit encrypts its network communication using RC4 algorithm. Key for RC4 and botnet name are encrypted using xor with a hardcoded password.

XOR encryption is pretty standard, the only catch is that it skips key's first char and then reverses the key.

XOR function calls:

Ciphertext lengths are almost always too long and we have to rely on null termination:

DGA config seems to be always declared at the beginning of the data section:

```
.data:2002A000 ; Section 3. (virtual address 0001A000)
.data:2002A000 ; Virtual size                  : 00003327 (  13095.)
.data:2002A000 ; Section size in file          : 00003327 (  13095.)
.data:2002A000 ; Offset to raw data for section: 0001A000
.data:2002A000 ; Flags C0000040: Data Readable Writable
.data:2002A000 ; Alignment     : default
.data:2002A000 ; ===========================================================================
.data:2002A000
.data:2002A000 ; Segment type: Pure data
.data:2002A000 ; Segment permissions: Read/Write
.data:2002A000 _data           segment para public 'DATA' use32
.data:2002A000                 assume cs:_data
.data:2002A000                 ;org 2002A000h
.data:2002A000 ; int dga_domain_no
.data:2002A000 dga_domain_no   dd 15                   ; DATA XREF: sub_2001CEFD+73↑r
.data:2002A004 ; int domain_seed
.data:2002A004 domain_seed     dd 36F066Dh             ; DATA XREF: sub_2001CEFD+79↑r
.data:2002A008 magic_check     dd 1                    ; DATA XREF: sub_2001D735+EB↑r
.data:2002A00C                 dd 580F9D06h
.data:2002A010 dword_2002A010  dd 0                    ; DATA XREF: sub_2001D735+23A↑r
.data:2002A010                                         ; DllEntryPoint+1D2↑r
.data:2002A014 ; u_short hostshort
.data:2002A014 hostshort       dd 0                    ; DATA XREF: sub_2001CEB0+6↑r
.data:2002A014                                         ; sub_2001D735+251↑r
.data:2002A018 ; u_short port_443
.data:2002A018 port_443        dd 443                  ; DATA XREF: DGA+64↑r
.data:2002A018                                         ; sub_2001D166+10↑r ...
.data:2002A01C xor_secret_length dd 5                  ; DATA XREF: sub_2001CEFD+3A↑r
.data:2002A01C                                         ; init_md5s+34↑r ...
.data:2002A020 unknown_chunk   db 0F7h ; ÷             ; DATA XREF: sub_2001CEFD+4A↑o
.data:2002A021                 db 0B8h ; ¸
.data:2002A022                 db  5Fh ; _
.data:2002A023                 db 0E8h ; è
```

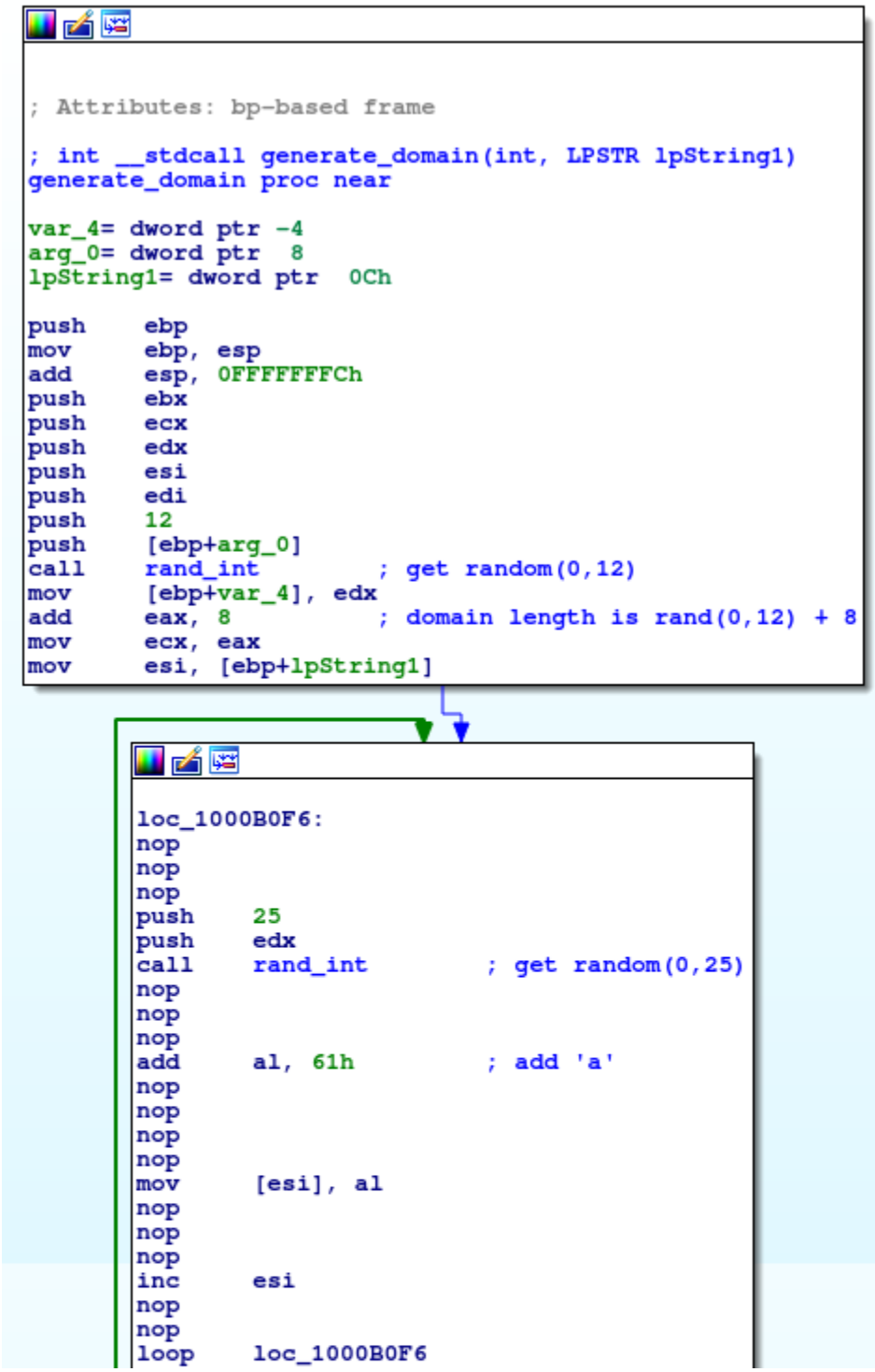
Persistence

Program copies itself into C:\Users\User\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\.

DGA

Ramnit generates a list of domains by using a LCG algorithm with a hardcoded seed:

Generating a domain:



```
mov     byte ptr [esi], 0
push    offset a_com    ; ".com"
push    [ebp+lpString1] ; lpString1
call    lstrcatA
xor     edx, edx        ; gen new seed
mov     eax, [ebp+arg_0]
mov     ebx, [ebp+var_4]
mul     ebx             ; x = eax * ebx
add     eax, edx        ; eax = low32(x) + high32(x)
pop     edi
pop     esi
pop     edx
pop     ecx
pop     ebx
leave
retn    8               ; return eax
generate_domain endp
```

DGA recreated in Python:

Communication

Ramnit connects to C&C servers through port 443, but don't let that fool you – it doesn't use HTTPS, but its own protocol instead:

Packet's structure:

Chunks' structures:

So if we'd like to send a packet containing some data, we would:

- encrypt large (>4bytes) chunk data using RC4 with a key recovered from the XOR decryption
- create packed chunks from data parts
- concatenate all chunks together
- wrap the output in packet layer

Traffic example:

Wireshark · Follow TCP Stream (tcp.stream eq 22) · dump

Magic Header Packet Length Command Data

```
00ff4b000000

e200200000005e654bf944b93632c607754f291e88c7f07f3e1613125ba71c94510bb42ebd1600200000005c614efc41b8603bc350704a7d148fc5a02d6a1644480daf41910258e47fe91e

00ff0100000001
00ff01000000
51
00ff860000005100800000058aa428e519aec96b58d683e6349a7c20aed60f1909ce6bd4eca3eaf437be1d8e032023b332145d4598c7af7b57d53e9166f9185a51cd234109b6a65e14a89f40013ab23b4a0a3d7da2f0a22ee3fa07118ef34fc72f3fb129c92bef679a95cc9beffdcd11b748981d7423d3e618f58603923a0c41839459030bb364f3be53055
00ff0d010000
f001000000000100000000010000000014c0d0000014c0d0000014f0d000000a7000000f4037e9a768f0203f434112ef931bff5931a5c2f74151be14c93014f802deb44eb710c14524e3010cfb5a4d0d2019314feb27d0dab7a0b370d2d3ea60834765c5c1deb44809c5e4e8407a307980f66874a893700b3ef11d5427698f2c131c9411532d94d0a11a8fbccc5e1394f01a7c28f68196531546383 2bd57ac3f35ca7c80fdb0c67b830fd4d525376bf3d66f95460b9fa01e990a64f3f3948354aac1ce14f03ff173c6c26002000000005e654bf944b93632c607754f291e88c7f07f3e1613125ba71c94510bb42ebd1600040000001c660dee010a0000000104000000010100000001000000000
00ff14000000f002dcc5000000000000010000000132000000
00ff01000000
18
00ff0b000000180 10a0000000178000000
00ff14000000
1a015f2fe5580004000001c660dee0100000000
00ff0b0000001a015f2fe5580000000000
00ff01000000
16
00ff860000001601 80510100
```

Some of available commands:

Command	Byte Value	Short Description
COMMAND_OK	0x01	Server's response that the command executed successfully
GET_DNSCHANGER	0x11	Get DNS-changer payload
GET_INJECTS	0x13	Get webinjects
UPLOAD_COOKIES	0x15	Upload stolen cookies (zip format)
GET_MODULE	0x21	Get a specific module
GET_MODULE_LIST	0x23	Get a list of downloadable modules
VERIFY_HOST	0x51	Check if the host is able to send a signed message
REGISTER_BOT	0xe2	Register bot (send two MD5s)
UPLOAD_INFO_GET_COMMANDS	0xe8	Upload detailed machine info

Bot registration

When a bot wants to register itself it sends two encrypted md5 hashes, the data structure of which is following:

Python code:

If C&C responds with a success packet (00ff0100000001), malware follows up with a empty 0x51 command. Signature from the response is verified using a hardcoded public RSA key. If there is a mismatch – the execution stops.

Modules

The program can request a list of modules and then download each one individually:

Antivirus Trusted Module v2.0

Adds exceptions to a fixed list of anti-virus software (AVG Anti-Virus, BitDefender, Avast, ESET NOD32 Antivirus, Norton AntiVirus)

Chrome reinstall module (x64-x86) v0.1

Uninstalls Google Chrome

and installs it again:

Cookie Grabber v0.2 (no mask)

Steals cookies from various hardcoded locations and sends a zip with results to the C&C through rmnsoft.dll.

Hooker

Used for performing Man-in-the-Browser attacks and hooking HTTP functions.

Webinjects

Webinjects are a relatively new addition to Ramnit. They utilize a standard Zeus format:

Obfuscation / Evasion

Ramnit attempts to hide itself from Windows Defender by adding following registry values:

'NOPs' are inserted in random functions, which makes them difficult to find using e.g. Yara rule:

```
push    ebp
mov     ebp, esp
add     esp, 0FFFFFFACh
push    44h
lea     eax, [ebp+StartupInfo]
push    eax
call    sub_151911BB
push    10h
lea     eax, [ebp+ProcessInformation]
push    eax
call    sub_151911BB
nop
nop
push    small [ebp+arg_4]
nop
nop
pop     small [ebp+StartupInfo.wShowWindow]
nop
nop
nop
nop
push    1
nop
nop
pop     [ebp+StartupInfo.dwFlags]
nop
nop
lea     eax, [ebp+ProcessInformation]
```

New variant

During writing of this article we've noticed a variation of Ramnit called clickbideu in an Italian spam campaign.

Its loader is completely different, but the communication module (rmnsoft.dll) has remained somewhat unchanged with only some minor differences:

DGA cycles between 3 hardcoded TLDs instead of just one:

Python implementation:

Also new version seems to be using different port – 8001, although we've also seen usage of port 442.

Additionally, a different value ("fE4hNy1O") is used for calculating the second md5.

Additional links

IoCs

Yara rules:

Samples analyzed:

- Main PE

  92460d8ac1d1e9f155ef2ca6dd7abb417df8900a17e95157d4372a2c846e829f
- rmnsoft.dll

  be2044fe6f0220dde12c51677f2ef4c45d9dea669073bd052695584e573629e0
- modules.fll

  96a10e07d092f6f429672ce2ca66528aae19de872bda39249135a82477d27a83
- Module Antivirus Trusted Module v2.0 (AVG, Avast, Nod32, Norton, Bitdefender)

  975ed0f933d4a22ca631c5ab77c765cd46c48511d43326b066b4505c6dc911de
- Module Cookie Grabber v0.2 (no mask)

  bc977a0f455fc747a7868a7940aa98af10c91c4aae7598310de8b78132436bee
- Module Hooker

  a88151b3bf825e26ded28f94addeada095d2cd13791b2153a9594b26d9cfb85e

Configs:

Loader sha256:

- d290225dde1b18bf68c4c42e06638a61fb336c91a2c4e6dd007bcbe7327fcbae
- c2cae7d9ef91dfcc1ae8f542e0ac64ce66c526d5a4154241855020612d358ee8
- 1f3fbca46a599b4f221ead7785606451365db45bbbc537ee0c4d019e8984d106
- 9d723bb1dc375834ebb907271b83dffab44e98b82fa73da6267037f019e4bc83
- f3567e2b5fc521987f0dd79aff6f3b1328db8e03fa825c3c030080a8b5819564
- 7689465ba010537b0c29cf18d32a25962bd1605b717733f5953eb1b1eb0a68c9
- f98ca50b7d07682ac359b97dd68eb924c4cbd825db72c1a132458e9bb765fa1e
- 4b00b0ece480267af051e7907458381d8a9e8506c7da67b8a8e1d74d45773d68
- 6ac47d82134385fa73386ff3cd7b2eb7008da2205b3f5af7b41fab45c63f9046
- 6a1fc689d2ef32ee6288498f8a875c6dc880d7494f46c05d25d0e1f627984e8e
- 522e935b91307b8c01e0ea8a724985f5b4e01227a761aeccb63b00f0d964f7e9
- b3e67b5ee899c53f90c9da772592a4709372192542e1297bbce4929a8e1d5c69
- 71d92cc6dc9273d162a969960b1021e5f18cf39b2c48043e5c5e49db5a58d955

- da15c2a89334496910b6d966bf91fa25a1c9526c53796e06d166416abe7cf2f4
- e4353bda9692581ea9743165dfd843238c23bb92e24b778983de80e90ac650a3

DGA domains for analyzed configs: