Reversing Emotet: the annihilator of hopes, dreams, and time

Reverse Engineering

Michael Bruce Ades

Matrix

13 July 2025

Help Received and Academic Integrity Statement:

In accordance with the ACE core values, I have neither given nor received unauthorized aid on this paper.

Kade explained how to use Ghidra, but I did not use it :(

Josh explained what ProcDot does and actually told

Executive Summary

Our agency discovered a malware sample on the Enterprise Network (EN) and requires us to report our findings to USCyberCom. We must reverse engineer the sample named *267.bin* with the Message Digest Algorithm 5 (MD5) hash: *f3f48c57c38bff2ddd220f20569e1ee6*. The analysis requires a mix of both static and dynamic analysis. The agency provides the sample in a folder zipped with the password *infected*. We must add .exe to the sample for when we're ready to execute.

We cannot reconstruct how the 267.bin first reached the endpoint device, whether through spoofing, malicious documents, or removable media, so we assume that one legitimate user double-clicked the file, which led to the initial compromise on the EN. We focus our efforts on the analysis of post-initial compromise rather than the initial compromise vector. We further cannot guarantee that any hard-coded Internet Protocol (IP) addresses embedded in the sample remain active or respond on the network, so we assume each endpoint resolves correctly and answers Hypertext Transfer Protocol (HTTP) requests as intended by the malware creator and that the author only utilizes HTTP requests and no other forms of communication through covert channels. We assume that our Windows 10 environment mirrors the target environment's default configuration and that the malware operates as intended. We utilize this to avoid false negatives caused by environmental discrepancies and artifacts left behind by our sandbox.

We computed the hash, queried VirusTotal, and identified the 267.bin as Emotet. Strings.exe revealed a packed stub, and we unpacked the payload. The sample contained Command and Control (C2) Internet Protocols (IP) correlated with Emotet indicators. In our sandbox, ProcMon and What's Running showed the loader drop classtangent.exe into WOW64.dll, and install a Windows service named classtangent for persistence.

1. Problem Statement

Our agency discovered a malware sample on the Enterprise Network (EN) and requires us to report our findings to USCyberCom in a detailed, comprehensive report. The agency provides the sample in a folder zipped with the password *infected*. We must reverse engineer the sample named *267.bin* with the Message Digest Algorithm 5 (MD5) hash: *f3f48c57c38bff2ddd220f20569e1ee6*. The analysis requires a mix of both static and dynamic analysis.

For our static analysis, the report requires calculations of the MD5 and Secure Hash Algorithm 256-bit (SHA-256) hashes, file size, and critical data from VirusTotal (first detection date, top detection vendors, and reported file names). The report requires any packers or obfuscators used by the malware, along with anti-disassembly measures, with the use of online relevant sources. In dynamic analysis, our agency requires that we trace registry modifications (keys added, modified, or deleted), file creations and modifications (paths and names), and network connections (destination Internet Protocols (IP) and Uniform Resource Locator (URL) requests). The agency requires us to enumerate significant Dynamic Link Libraries (DLL) and dependencies that the malware loads or modifies, and record any services it creates or alters. They also require us to inspect kernel-level hooks or rootkit components used.

Our final report will include all file hashes, VirusTotal and AnyRun links, malware family classifications, indicators of compromise (IoC), packer details, and references. We ensure that USCyberCom receives a thorough analysis of 267.bin with a key assessment of the sample's threat.

2. Assumptions

We cannot determine the initial compromise vector, whether the 267.bin arrived through a phishing attack or removable media, so we assume a user-initiated execution. We presume the user initiated launch of the malware in order to focus on unpacking the malware's features past the initial compromise. We do not concern ourselves with the reconstruction of the exploit delivery or mechanisms.

We cannot verify the reachability of the attacker's command and control infrastructure, and we therefore assume each hardcoded domain and IP resolves and responds over HTTP as intended. We document each outbound request and response and map the malware's network protocol without reverse-engineering other custom transport features over covert channels.

We assume our Windows 10 x64 analysis lab mirrors the target environment's default configuration (registry policies, installed executables, and security controls). We treat our sandbox as an accurate stand-in for all information and assume that the malware leaves all notable artifacts required for analysis. We further avoid false negatives caused by environmental discrepancies and concentrate on the malware's logic rather than the sandbox's artifacts.

3. Background

We group malicious software into families based on behavior and tooling. Our investigation of 267.bin benefits from a quick background of the inner workings of malware.

3.1 Malware

Malicious code infiltrates hosts through social-engineering tactics, fake downloads, or lateral movement, then executes commands that steal data, disrupt services, or recruit devices for

broader campaigns. We outline four malware concepts that frame our analysis: Trojans, botnets, command-and-control (C2), and packers.

3.1.1 Trojan

A Trojan horse acts as a type of malware that disguises itself as legitimate software to deceive users into running it [1]. It relies on the target to execute a seemingly harmless file that conceals malicious intent [1]. Once the target launches the decoy application, the hidden payload executes the attacker's intended tasks. CrowdStrike notes that when a Trojan runs, it exfiltrates sensitive information, spawns secondary implants, and gives attackers persistent remote access to the system [2].

3.1.2 Botnet

Botnets use a collection of a lot of compromised hosts in a centrally managed swarm. Each bot silently awaits commands from a controller to perform coordinated malicious activities. Palo Alto Networks describes bots that poll a controller, accept commands, and participate in distributed-denial-of-service (DDOS) or credential-stuffing campaigns [3].

3.1.3 C2

A command-and-control framework (C2) generates agents that run commands on target hosts and relay results to the C2 server. The C2 deploys listeners that accept incoming sessions from those agents. Operators queue commands, transfer files, and pivot with agents. [4]

3.1.4 Packers

Packers compress or encrypt an executable, wrap the original code within a stub, and unpack the payload only after launch. ANY.RUN analysts note that this technique makes static scanners come to no conclusion because the on-disk file lacks readable strings and imports [2]. Companies typically use packers for software protection or compression, but malware authors leverage them as an anti-reverse-engineering measure [2]. Common packers include UPX and custom encryption routines. A prerequisite for understanding a malware's true behavior involves unpacking the malware sample, either via automated tools or manual debugging. In this case, initial static analysis of 267.bin suggested it was packed due to high entropy and unreadable strings, which prompted us to use an unpacking tool (see Section 5).

3.2 Emotet

The Emotet malware family began as a banking Trojan and evolved into a modular botnet loader [5]. Over its lifetime, Emotet's developers upgraded it, adding new features and distribution methods. Emotet primarily spreads through massive spam email campaigns, and victims receive phishing emails with malicious Word/Excel attachments or links that use macros, a feature to run commands with those attachments, and a hidden Emotet DLL is downloaded and executed on their system [6].

Emotet functions as a botnet. Infected machines, known as bots, contact C2 servers for commands. Emotet's controllers often use the botnet to deliver payloads. Once Emotet establishes a foothold, it can drop other malware such as TrickBot or beacons, used for ransomware attacks [6]. Emotet malware often acts as an early step in a multi-layer attack, leading to data theft. Emotet uses a modular architecture with plugins for specific tasks, such as

email harvesting, lateral movement, and credential stealing. Emotet follows a polymorphic nature, in which it frequently changes code to avoid detection and uses packers to make it hard to detect and remove [6].

In early analyses between 2018-2020, Emotet used randomly generated process names for persistence and added unique registry keys for identification [6]. The later versions from 2022 and beyond adopted 64-bit code and changed infection techniques to use Excel macros or OneNote attachments, and continued to add more to the codebase [6]. The sample 267.bin appears as an Emotet variant active around mid-2023 (see Section 5 for VirusTotal results). We expect it to exhibit Emotet's main behaviors of process injection or duplication, establishing persistence, and reaching out to a list of remote hosts over HTTP.

4. Tools and Techniques

To analyze the 267.bin sample, we utilized a combination of static and dynamic analysis tools. These tools helped us unpack the malware, inspect its code, and observe runtime behavior, while maintaining a controlled environment. Below, we list the key tools and techniques used.

4.1 Static Analysis

Static analysis involves examining the malware binary without executing it. We gather file metadata, scan for readable strings, check for packer signatures, and attempt to identify functionality from the file's structure. Below, we list our static analysis toolkit for the 267.bin sample.

4.1.1 Strings.exe

Strings.exe from Sysinternals scans binaries for ASCII or Unicode text and reveals embedded URLs, Windows API calls, and suspicious commands. In our case, *strings 267.bin* showed long stretches of random characters and only a few recognizable names, which indicates a strong chance that the sample used a packer [7].

4.1.1 Dependencies.exe

Dependencies.exe from Sysinternals provides insights into the imports and exports of a. Executable. We utilize the dependencies tool to understand the various Windows API imports 267.bin uses for its operation. Furthermore, we correlate our statically analyzed dependencies with the dependencies we locate through dynamic analysis [7].

4.1.1 Certutil

We used the built-in certutil utility to compute cryptographic hashes of the file. Running *certutil -hashfile 267.bin MD5* and then *SHA256* gave us the unique identifiers needed to query threat intelligence [8].

4.1.1 Detect-It-Easy (DiE)

We used DiE to detect packers and measure entropy. The tool identifies known packers used by executables. DiE did not identify a known packer for the sample, which reinforces the possibility of a custom packer stub. However, DiE's entropy analysis showed the binary's entropy ~6 bits per byte, which leans on the higher side of entropy. A uniformly high entropy across large sections means the file's contents are largely mixed, consistent with packing. We

interpreted this result as evidence that 267.bin is indeed packed, but not with a packer that DiE recognizes [7].

4.2 Dynamic Analysis

Dynamic analysis involves running the malware in an isolated environment to observe its behavior. We prepared a Windows 10 Virtual Machine with monitoring tools and reverted to a snapshot to ensure a clean start. Below, we use the following tools for dynamic analysis.

4.2.1 Procmon

We launched ProcMon to capture real-time file system, registry, process, and thread events during malware execution. We set a filter to focus on events from 267.exe to record only relevant actions. ProcMon allowed us to see which files the malware created or modified, which registry keys it accessed, any subprocesses spawned, and any IP addresses communicated with [7].

4.2.2 What's Running

What's Running lists every thread, module, and socket, letting us spot rogue processes that standard Task Manager often misses. It also keeps track of these processes through a record [7].

4.2.2 ProcDot

ProcDot ingests Procmon output plus packet captures, then draws an interactive graph that connects executable event flow [7].

4.2.3 FakeNet

FakeNet-NG emulates common Internet services and captures outbound traffic, allowing us to observe C2 requests without leaking data to external hosts [7].

4.2.3 Regshot

Regshot snapshots the registry before and after execution, producing a diff that highlights new persistence keys or altered security settings [7].

4.3 VirusTotal

VirusTotal submits the hash or sample to more than seventy antivirus engines and multiple static analyzers, and provides reputation scores and related samples that speed the identification process [7].

4.4 Any.Run

Beyond execution of the malware, ANY.RUN allows memory dumping and packet replay. It provides insight into the initial techniques the malware utilizes and serves as a good base for initial triage [7].

4.5 Unpacker - Mal_Unpack

Mal_Unpack launches the packed binary inside a sacrificial process, waits for self-decryption during the unpack process, dumps the restored PE carved from memory, and terminates the host process. It then provides the clean sample for further reversing [9].

4.6 IPInfo

IPInfo's API enriches destination IPs with each identifiable location, along with a map that plots the IPs [10].

4.6 VirtualBox

VirtualBox snapshots let us roll the lab VM back to a clean state after execution, removing the need for lengthy rebuilds while allowing for repeatable tests [11].

5. Problem Solution

The solution below follows the flow of a walkthrough of our analysis from initial identification, static analysis, and dynamic analysis. We then dive deeper into the malware strain and compare things we discover with what we researched.

5.1 Initial Identification of Malware

In Figure 1, we start with the extraction of our malware with the password of "infected" on the zip file.

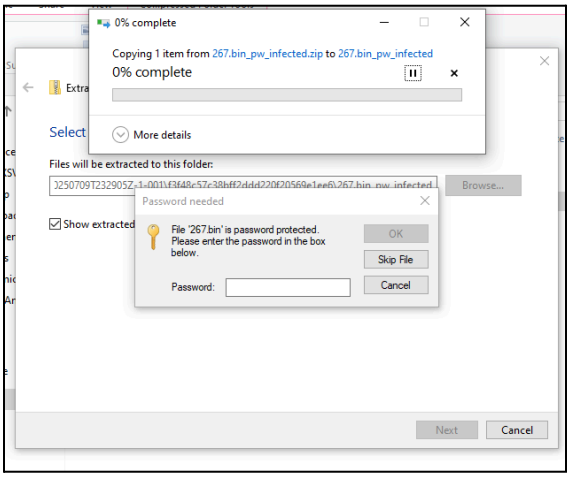

Figure 1. Extraction of our 267.bin malware used for analysis

Once we extracted the sample named 267.bin, we took a snapshot of the virtual machine on VirtualBox. The snapshot provides a save that we recover to in the event of a failure when we work with the malware. We now start with the initial collection of information. In Figure 2, we identify the size of 267.bin.

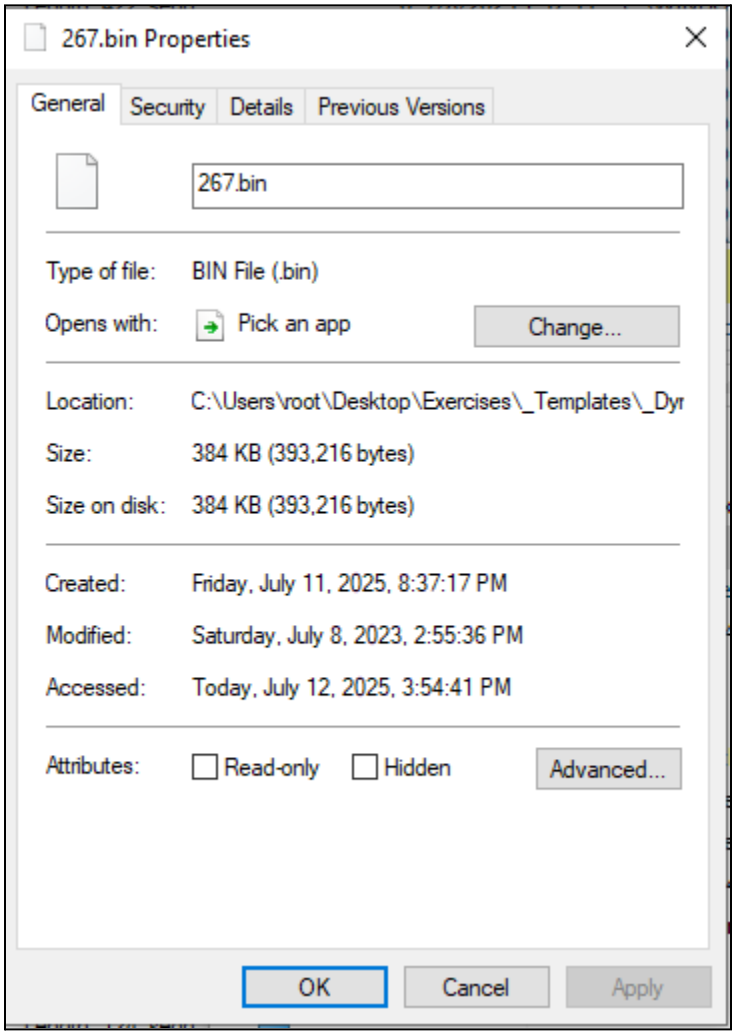

Figure 2. Size of 267.bin

We require the MD5 and SHA-256 hash of the sample. The hash provides a unique identification of the sample, and we use it for further reconnaissance in VirusTotal. In Figures 3 and 4, we use certutil and identify its hash.

```
C:\Users\root\Desktop\Exercises\_Templates\_Static Analysis Template>certutil -hashfile 267.bin MD5
MD5 hash of 267.bin:
f3f48c57c38bff2ddd220f20569e1ee6
CertUtil: -hashfile command completed successfully.
```

Figure 3. MD5 hash of 267.bin

The MD5: f3f48c57c38bff2ddd220f20569e1ee6 hash of the sample.

```
C:\Users\root\Desktop\Exercises\_Templates\_Static Analysis Template>certutil -hashfile 267.bin SHA256
SHA256 hash of 267.bin:
b1cad1540ecb290088252635f8e130022eed7486eb128c0ca3d676945d60a9fc
CertUtil: -hashfile command completed successfully.
```

Figure 4. SHA-256 hash of 267.bin

The SHA-256: b1cad1540ecb290088252635f8e130022eed7486eb128c0ca3d676945d60a9fc hash of the sample. We create two hashes for redundancy and ensure that the MD5 and SHA-256 correspond to the same malware sample. We now identify if the sample existed previously through VirusTotal. In Figure 5, we pass in the MD5 hash into VirusTotal and identify the sample as “trojan.emotet/zusy.”

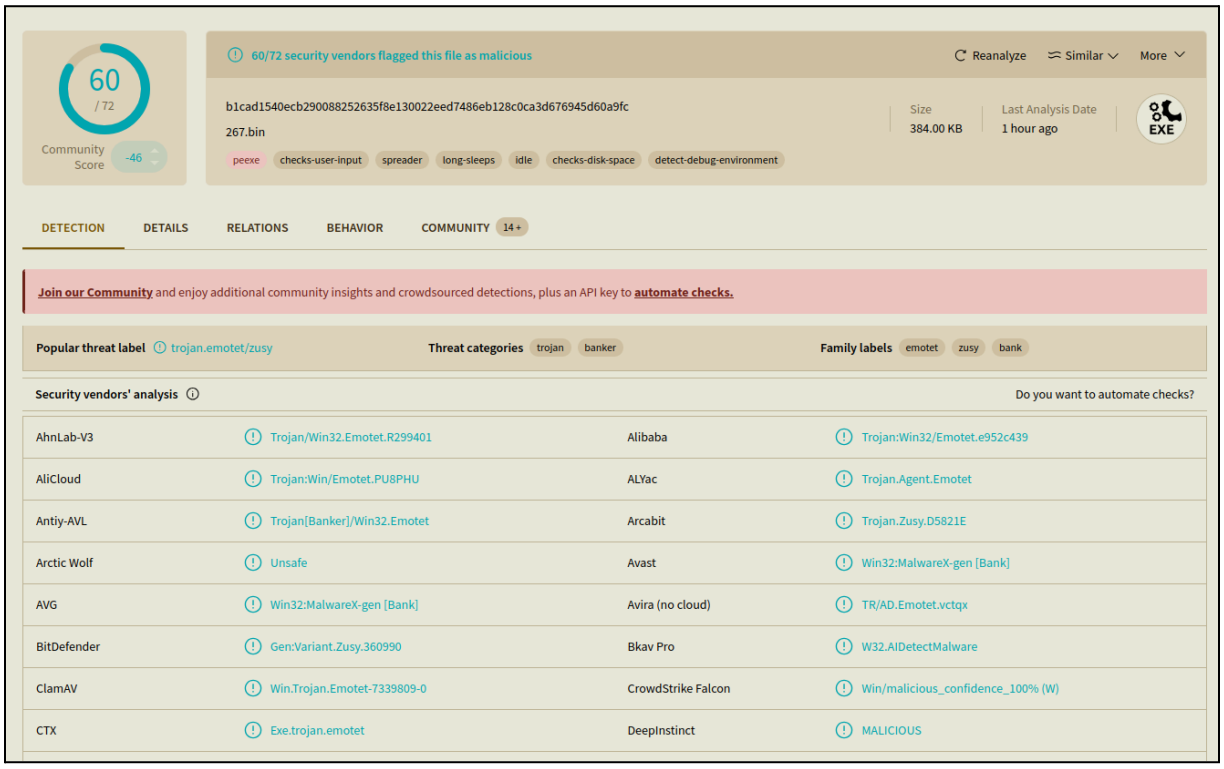

Figure 5. VirusTotal of 267.bin

We now know that our sample belongs to the Emotet/Zusy family, and numerous threat detectors flag the executable as malicious. In Figure 6, we identify the first occurrence of the sample in the wild along with the other aliases used for the same sample.

Basic properties

MD5	f3f48c57c38bff2ddd220f20569e1ee6
SHA-1	0421127f1bcca91a6ab2a570a47f8159101b751a
SHA-256	b1cad1540ecb290088252635f8e130022eed7486eb128c0ca3d676945d60a9fc
Vhash	035056555d1d5517z70057mz17ez1
Authentihash	a3a64d72a568e549fa5f437b371f418445ee04c6ff6636a24318a543c23398f3
Imphash	efe1c3568d5733ccb1e9d2b524c47cea
Rich PE header hash	24ae9e717eb5d67a8cc4a0b4b12ce081
SSDEEP	3072:iYylxN7LMWf+GPBLi21ocO2jytUkU4uDQUiysA+30Sor6KH7j0m43ayYZt:Glx5MKQUJkqDDj+xW6KH7luN
TLSH	T1A884B00532A4D4B2D89701BF8D03C33556B6F0A45B269BC377D40D9E9BA46E1BA3B3C9
File type	Win32 EXE executable windows win32 pe peexe
Magic	PE32 executable (GUI) Intel 80386, for MS Windows
TrID	Win32 Executable MS Visual C++ (generic) (47.3%) | Win64 Executable (generic) (15.9%) | Win32 Dynamic Link Library (generic) (9.9%) | Win16 NE executable (generic) (7..
DetectItEasy	PE32 | Compiler: EP:Microsoft Visual C/C++ (2005) [EXE32] | Compiler: Microsoft Visual C/C++ (14.00.50727) [C++/book] | Linker: Microsoft Linker (8.00.50727) | Tool: Vi..
Magika	PEBIN
File size	384.00 KB (393216 bytes)

History

Creation Time	2019-10-06 18:38:52 UTC
First Seen In The Wild	2019-10-13 14:16:20 UTC
First Submission	2019-10-13 08:27:20 UTC
Last Submission	2025-07-12 20:14:31 UTC
Last Analysis	2025-07-12 19:32:56 UTC

Names

267.bin
classtangent.exe
267.exe
b1cad1540ecb290088252635f8e130022eed7486eb128c0ca3d676945d60a9fc.exe
zararli.exe
olqXjben
tabletthemes_exe
267.bin.exe
b1cad1540ecb290088252635f8e130022eed7486eb128c0ca3d676945d60a9fc 2.exe
magnifybackup.exe

Figure 6. Identification of 267.bin history and aliases

We proceed and use Any.Run and identify how the sample operates. In Figure 7, we observe that the sample drops a clone of itself in a different location and creates a child process.

Figure 7. Behavior of 267.bin

The behavior indicates a packed malware sample that unpacks itself. Now that we have an initial idea of the sample, we start with static analysis.

5.1 Static Analysis

For static analysis, we must always start with strings, no matter what. Strings provide a lot of insight into the behavior of the malware. In Figure 8, we start our process with the command *strings.exe 267.bin* and find a long section of randomness.

Figure 8. Strings on 267.bin

The section of randomness indicates a packed sample, because strings typically reveal a lot of human-readable literals (file paths, version strings, and API names); however, a long continuous stream of data indicates an encrypted payload hidden in the executable. In Figure 9, we identify various API calls indicative of packing.

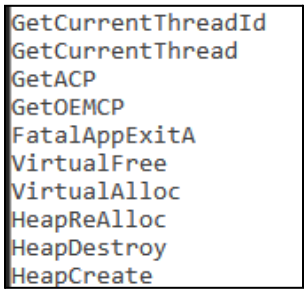

Figure 9. Strings on 267.bin

In Figure 10, we identify the CRYPT32.dll in strings as well.

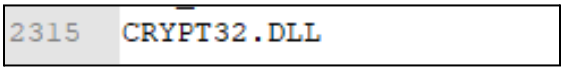

Figure 10. Strings on 267.bin

The presence of the API calls suggests that the binary lives as a tiny loader stub rather than a full application. A packer embeds its payload as an encrypted or compressed blob inside the PE file with CRYPT32.dll, and we assume that the stub uses VirtualAlloc and HeapAlloc to carve out an

executable memory region and decrypt the payload into that buffer. In Figure 11, we further confirm our assumption through our next tool, Detect-It-Easy, which calculates the entropy of a sample.

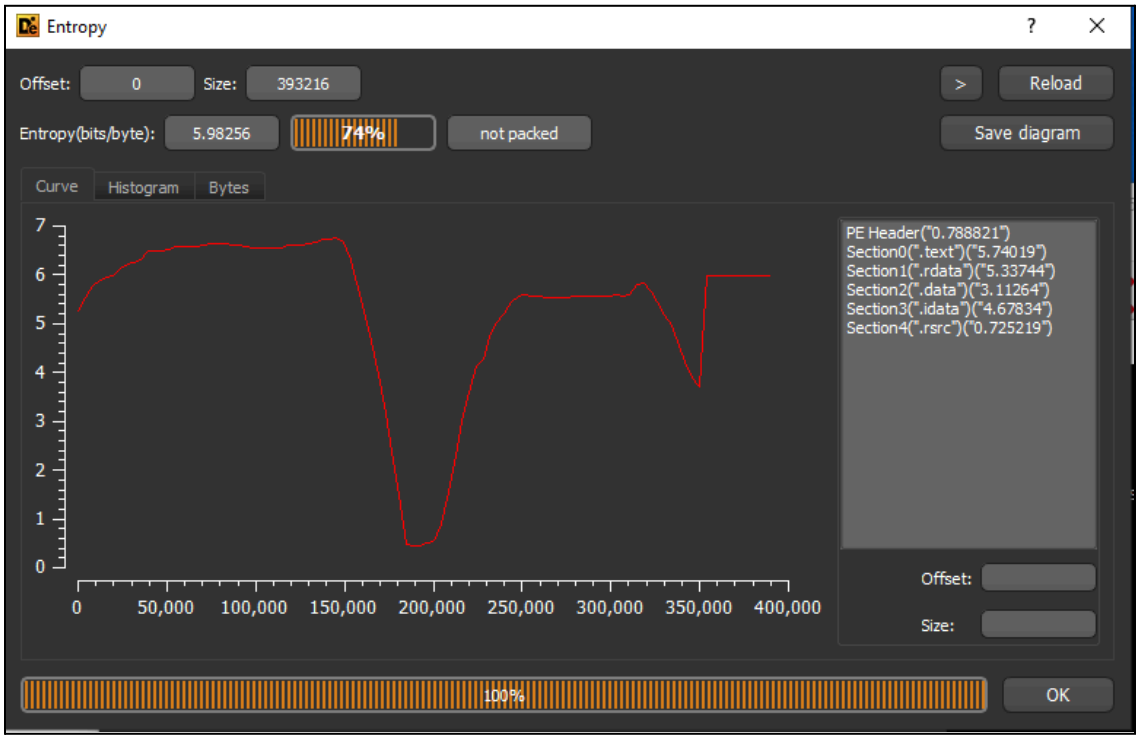

Figure 11. Entropy of 267.bin

Detect-It-Easy does not identify a packer. This leads us to the belief of a custom packer. The overall entropy of the sample hovers around 5.98256, which tells us that the binary isn't uniformly random. When we derive randomness, the higher the file entropy leads to a greater indication of a malicious sample, because copious amounts of encryption signify intentions to hide its own functionality.

We use the mal_unpack tool and set a wait time of 3000ms, because it takes a while for the sample to finish unpacking itself. In Figure 12, we utilize the tool mal_unpack, and we successfully unpack the 267.bin with the command *mal_unpack /exe 267.bin /timeout 3000*.

```
Microsoft Windows [Version 10.0.19045.4651]
(c) Microsoft Corporation. All rights reserved.

C:\Users\root\Desktop\Exercises\_Templates\_Static Analysis Template>mal_unpa
ck /exe 267.bin /timeout 3000
[-] Could not set debug privilege
[*] Cache is Disabled!
Starting the process: 267.bin
With commandline: ""
Exe name: 267.bin
Root Dir: 267.bin.out
[*] Watch respawns from main EXE file: 267.bin
Module Path retrieved: C:\Users\root\Desktop\Exercises\_Templates\_Static Ana
lysis Template\267.bin
Scanning...
Scanning...
Scanning...
Scanning...
Found suspicious: 2920
Suspicious detected, breaking!
Unpacked in: 1484 milliseconds; 4 attempts.
WARNING: 1 of the related processes are not killed

C:\Users\root\Desktop\Exercises\_Templates\_Static Analysis Template>
```

Figure 12. mal_unpack on the 267.bin

We label the unpacked sample *unpacked_267.bin.exe*. We run strings on the new file and see that the strings dump shows a full suite of high-level WIN32 API calls in contrast with the previous packed sample.

```
826  IsProcessorFeaturePresent
827  lstrlenW
828  WaitForSingleObject
829  GetFileAttributesW
830  DeleteFileW
831  lstrcpyW
832  HeapFree
833  GetLastError
834  GetTickCount
835  lstrlen
836  GetCommandLineW
837  GetFileSize
838  LoadLibraryW
839  WTSGetActiveConsoleSessionId
840  GetProcAddress
841  HeapReAlloc
842  LoadLibraryA
843  SignalObjectAndWait
844  lstrcpynW
845  SetFileAttributesW
846  VirtualAlloc
847  GetCurrentProcessId
848  GetTempFileNameW
849  CreateProcessW
850  GetLocalTime
```

Figure 13. strings on unpacked_267.bin.exe

We see that the stub imports GetTickCount and Sleep, which may measure execution delays, and if a debugger single-steps or breaks, the elapsed time may cause the malware to abort or switch behavior. We identify this as one of the anti-debugging techniques.

When we observed the original dependencies for the unpacked malware, we noticed 3 DLLS loaded. In Figure 14, we identify KERNEL32.dll with the specific loaded WinAPI calls used by the malware.

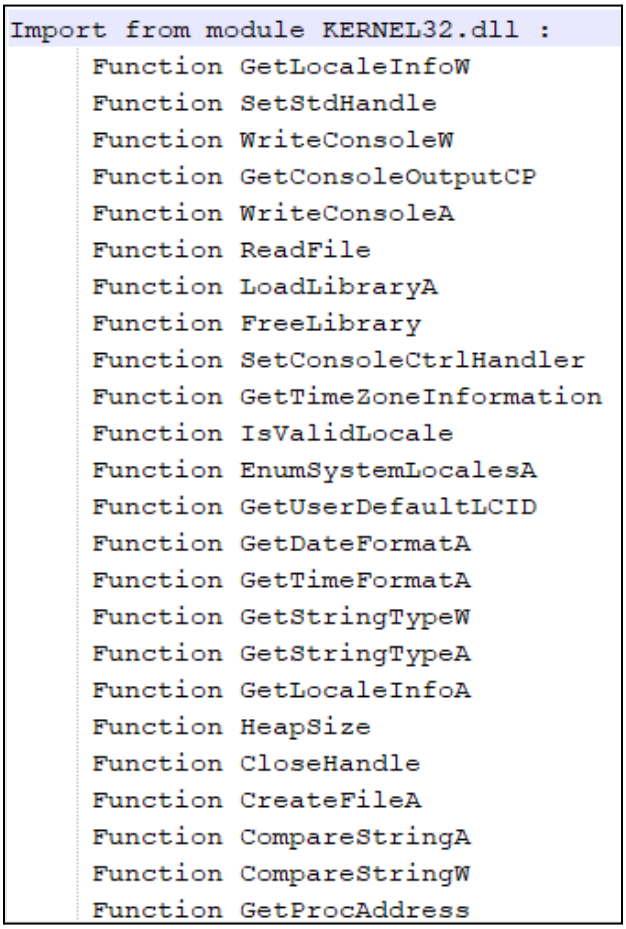

Figure 14. KERNEL32.dll

In Figure 15, we identify USER32.dll with the specific loaded WinAPI calls used by the malware.

```
Import from module USER32.dll :
     Function LoadImageA
     Function GetWindowLongA
     Function SetWindowLongA
     Function BeginPaint
     Function GetClientRect
     Function EndPaint
     Function LoadIconA
     Function LoadCursorA
     Function RegisterClassExA
     Function GetSystemMetrics
     Function CreateWindowExA
     Function ShowWindow
     Function UpdateWindow
     Function GetWindowRect
     Function DefWindowProcA
     Function PostQuitMessage
     Function DestroyWindow
     Function UnregisterClassA
     Function GetMessageA
     Function TranslateMessage
     Function DispatchMessageA
     Function TranslateAcceleratorA
     Function LoadStringW
```

Figure 15. USER32.dll

In Figure 16, we identify GDI32.dll with the specific loaded WinAPI calls used by the malware.

```
Import from module GDI32.dll :
     Function GetObjectA
     Function DeleteObject
     Function CreateCompatibleDC
     Function SelectObject
     Function BitBlt
     Function DeleteDC
     Function GetStockObject
[-] Import listing done
```

Figure 16. GDI32.dll

When we observe the unpacked malware, we identify only the KERNEL32.dll and the NTDLL.dll. We found no direct references to crypt32.dll or other essential DLLs. Instead, we believe the unpacked payload calls LoadLibrary and GetProcAddress at runtime to fetch crypt32.dll and any additional modules dynamically, indicative of other Emotet samples [6]. This measure prevents signature-based scanners and debuggers from setting breakpoints on those APIs before the malware unpacks itself. It defers the DLL links until after the decryption stub completes, and the malware adds another obstacle that hides its cryptographic and network routines behind dynamic loading.

5.2 Dynamic Analysis

Before execution of the malware, we took a Regshot baseline and started ProcMon with filters for 267.exe. We also ensured FakeNet was running to catch network traffic. We then set up our sample for execution by modifying the 267.bin to 267.exe. We then execute our 267.exe. In Figure 17, we start up ProcMon and filter for processes named 267.exe.

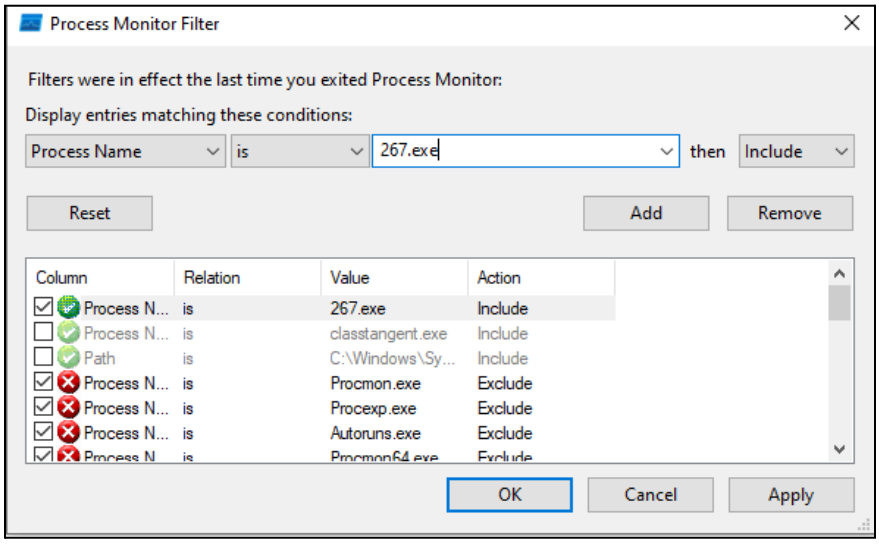

Figure 17. strings on unpacked_267.bin.exe

Upon launching 267.exe, it quickly spawned a new process. In our test, the new process was created at path: C:\Windows\SysWOW64\classtangent.exe. The malware created a subdirectory in this location, copied itself (or decrypted payload) to that location as an EXE, and executed it. We observed in ProcMon that 267.exe wrote a file "classtangent.exe" to the above path (with the same file size as the original, which indicates it was the payload dropped). In Figure 18, we use What's Running and identify the classtangent.exe process, which correlates with our results found in ProcMon.

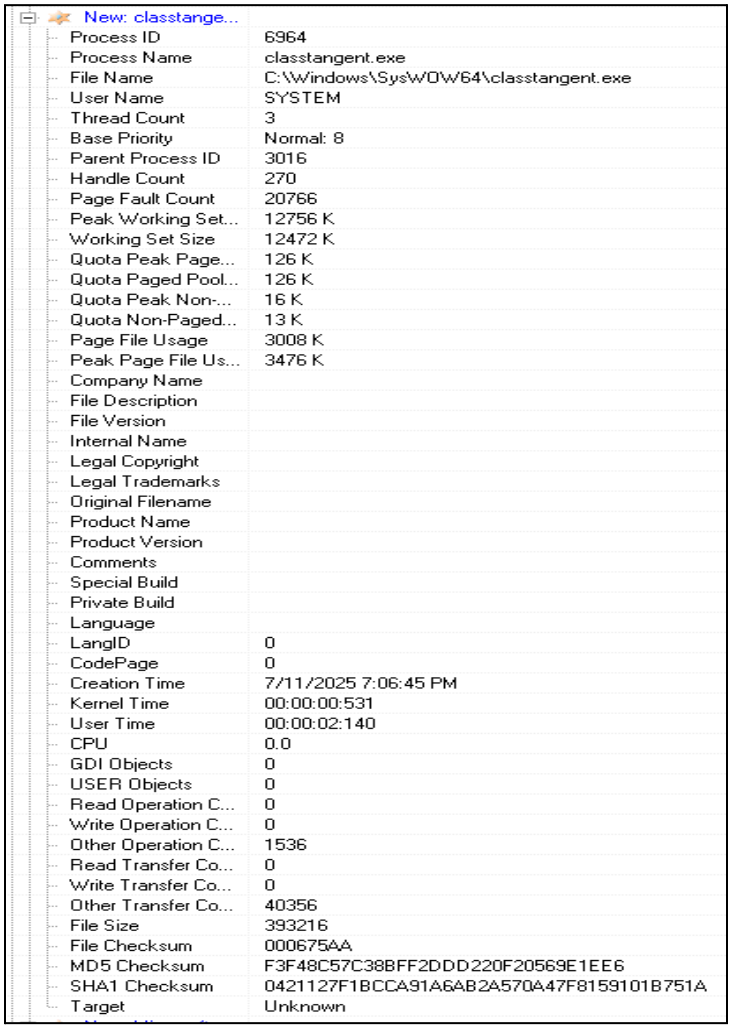

Figure 18. classtangent.exe process

In Figure N, we further notice the creation of the classtangent service through What's Running. The service was named "classtangent," which matched the dropped EXE name. The malware invoked the Service Control Manager through WinAPI to create a new service entry pointing to the classtangent.exe executable, and then started that service. This results in classtangent.exe running with SYSTEM privileges, assuming it was invoked with those permissions, and ensures it will run on every boot (as long as the service startup type is auto). We confirmed the service creation by checking the registry logs from regshot. In Figure 19, the new registry key *HKLM\SYSTEM\CurrentControl\Set\Services\classtangent* was present, with an ImagePath value of the classtangent.exe file.

```
HKLM\SYSTEM\ControlSet001\Services\classtangent<BR>
HKLM\SYSTEM\WPA\8DEC0AF1-0341-4b93-85CD-72606C2DF94C-7P-37<BR>
HKLM\SYSTEM\CurrentControlSet\Services\classtangent<BR>
```

Figure 19. registry keys in regshot for the classtangent service

The use of a service for persistence is a key finding, which means removal of the malware requires deleting that service entry and the file. Below we provide Figure 20, where we first identified the classtangent service.

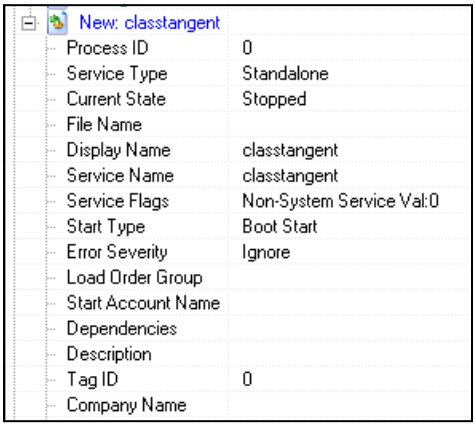

Figure 20. classtangent.exe service

In Figure 21, we check the classtangent services property and we see the startup type set to Automatic, which acts as a persistent mechanism for the malware.

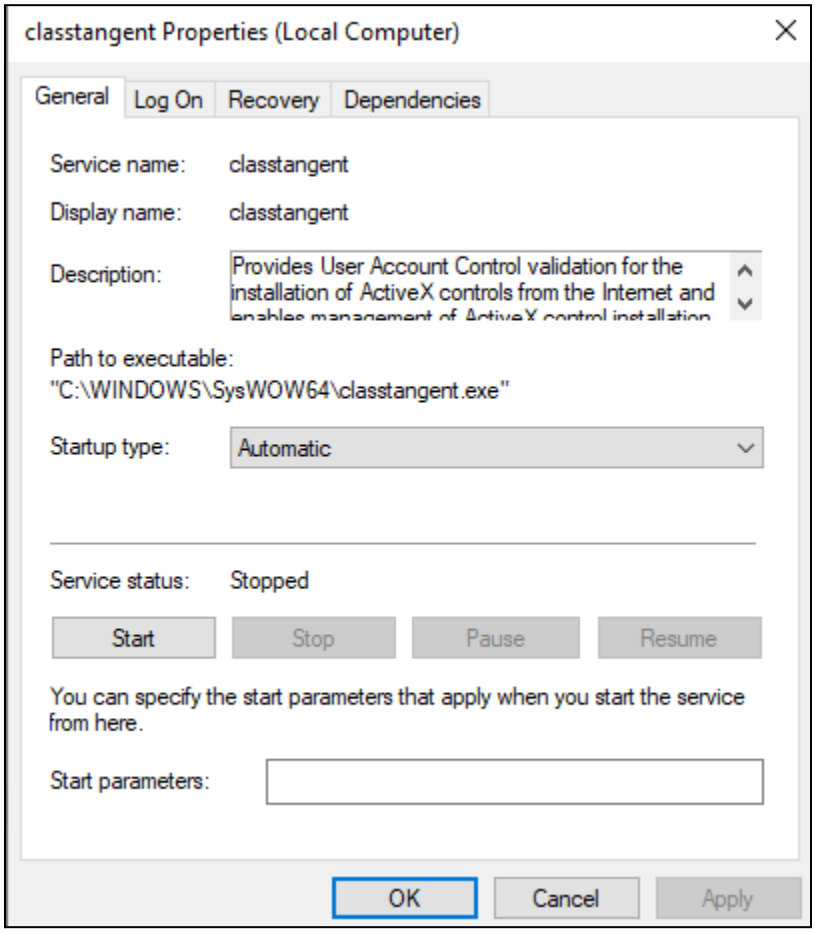

Figure 21. classtangent.exe service automatic startup

In Figure 22, we further noticed registry key modification that set up various Tcpip interfaces, which indicated the malware used network connectivity.

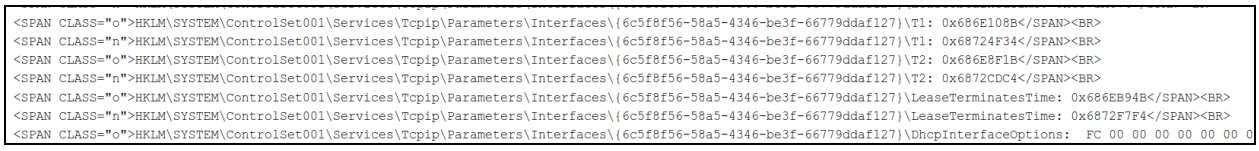

Figure 22. registry key modifications for TcpIp interfaces

When we look into What's Running we see IP connections established under the classtanget.exe process. In Figure 23, we provide one of the IP connections reached by the malware.

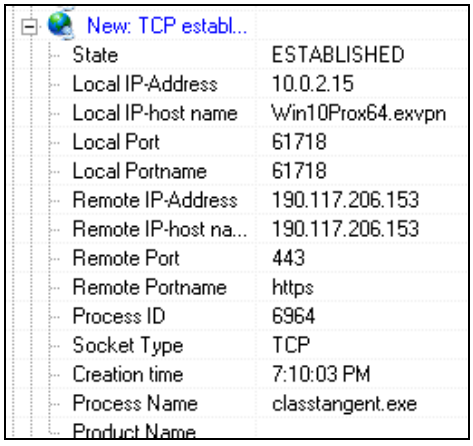

Figure 23. ip connection established by the malware

We do further reconnaissance of the IP connections the malware attempts to reach out to via ProcMon, because it logs every event the malware conducts. In Figure 24, we filter for processes with the name classtangent.exe, because this executable establishes those TCP connections.

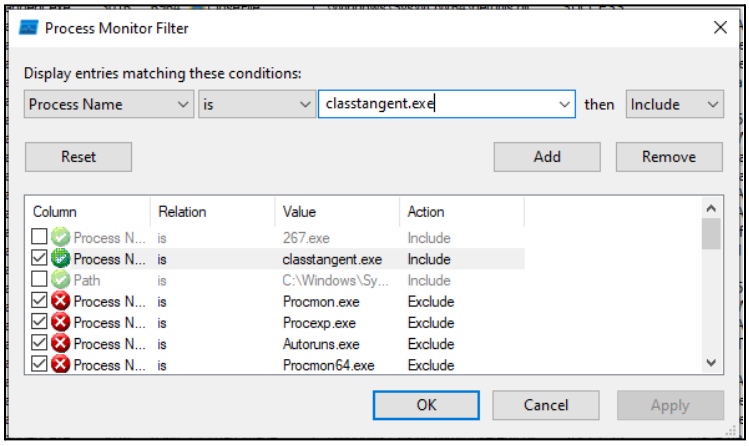

Figure 24. filter set for procmon with classtangent.exe

In Figure 25, we identify 25 TCP connections in ProcMon and put them into a CSV for further analysis.

	Time of Day	Process Name	PID	Operation	Path	Result
2	1:56:12.5452739 PM	classtangent.exe	6964	TCP Connect	Win10Prox64.exvpn:56899 -> 190.117.206.153:https	SUCCESS
3	1:56:12.5553604 PM	classtangent.exe	6964	TCP Send	Win10Prox64.exvpn:56899 -> 190.117.206.153:https	SUCCESS
4	1:56:12.5588284 PM	classtangent.exe	6964	TCP Disconnect	Win10Prox64.exvpn:56899 -> 190.117.206.153:https	SUCCESS
5	1:56:15.6395306 PM	classtangent.exe	6964	TCP Connect	Win10Prox64.exvpn:56901 -> 203.99.187.137:https	SUCCESS
6	1:56:15.6543091 PM	classtangent.exe	6964	TCP Send	Win10Prox64.exvpn:56901 -> 203.99.187.137:https	SUCCESS
7	1:56:15.6570946 PM	classtangent.exe	6964	TCP Disconnect	Win10Prox64.exvpn:56901 -> 203.99.187.137:https	SUCCESS
8	1:56:18.7787750 PM	classtangent.exe	6964	TCP Connect	Win10Prox64.exvpn:56902 -> 200.55.168.82:ftp-data	SUCCESS
9	1:56:18.7869043 PM	classtangent.exe	6964	TCP Send	Win10Prox64.exvpn:56902 -> 200.55.168.82:ftp-data	SUCCESS
10	1:56:18.8118673 PM	classtangent.exe	6964	TCP Send	Win10Prox64.exvpn:56902 -> 200.55.168.82:ftp-data	SUCCESS
11	1:56:18.8928623 PM	classtangent.exe	6964	TCP TCPCopy	Win10Prox64.exvpn:56902 -> 200.55.168.82:ftp-data	SUCCESS
12	1:56:18.8928821 PM	classtangent.exe	6964	TCP Receive	Win10Prox64.exvpn:56902 -> 200.55.168.82:ftp-data	SUCCESS
13	1:56:18.9080710 PM	classtangent.exe	6964	TCP TCPCopy	Win10Prox64.exvpn:56902 -> 200.55.168.82:ftp-data	SUCCESS
14	1:56:18.9080980 PM	classtangent.exe	6964	TCP Receive	Win10Prox64.exvpn:56902 -> 200.55.168.82:ftp-data	SUCCESS
15	1:56:18.9257653 PM	classtangent.exe	6964	TCP Disconnect	Win10Prox64.exvpn:56902 -> 200.55.168.82:ftp-data	SUCCESS
16	1:56:22.3053961 PM	classtangent.exe	6964	TCP Connect	Win10Prox64.exvpn:56904 -> 70.32.94.58:8080	SUCCESS

Figure 25. tcp connections identified in ProcMon of the classtangent.exe

In Table N, we make a list of all the IP addresses, ports, and communication types identified.

IP Address	Port	Communication Type
190.117.206.153	443	HTTPS (encrypted HTTP)
203.99.187.137	443	HTTPS
200.55.168.82	20	FTP Data Channel
70.32.94.58	8080	HTTP (proxy)
213.138.100.98	8080	HTTP
144.76.62.10	8080	HTTP
203.99.182.135	443	HTTPS
176.58.93.123	80	HTTP
192.241.220.183	8080	HTTP
94.177.253.126	80	HTTP
216.75.37.196	8080	HTTP
95.216.207.86	7080	HTTP (alt port)
78.109.34.178	443	HTTPS
113.52.135.33	7080	HTTP
216.70.88.55	8080	HTTP
138.197.140.163	8080	HTTP
83.169.33.157	8080	HTTP
212.112.113.235	80	HTTP
143.95.101.72	8080	HTTP
190.13.146.47	443	HTTPS
178.249.187.150	7080	HTTP
157.7.164.178	8081	HTTP (alt port)
5.189.148.98	8080	HTTP
51.38.134.203	8080	HTTP
93.78.205.196	443	HTTPS

Table 1. ip connections from classtangent.exe

The malware contained a targeted list of IP addresses. The malware used no hostnames/DNS queries. The ports used included 443, 8080, 7080, 8081; these are nonstandard for HTTP (except 443). However, we assume they only use HTTP for communication and no covert channels. In Figure 26, we filter into one of the individual IP addresses (190.117.206.153) in Wireshark and identify one of the malware's HTTP requests. We provide the pcap collected by Fakenet.

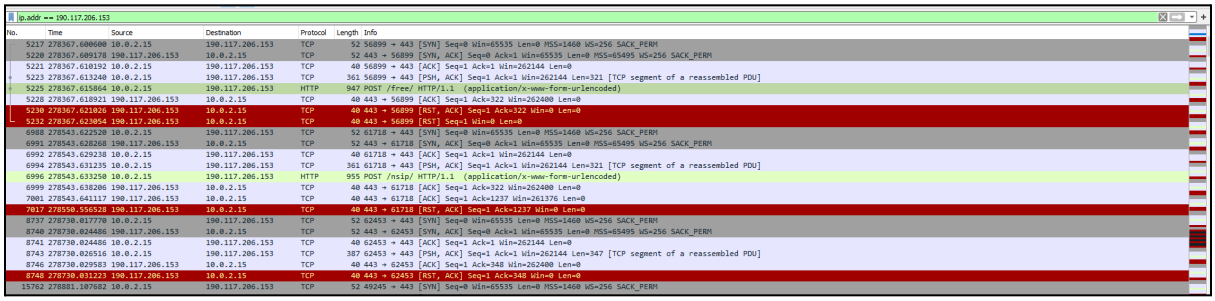

Figure 26. filter set for procmon with classtangent.exe

Each request had a URL path that looked somewhat benign. In Figure 27, we provide one of the HTTP responses with a redirect to the IP of the path /free/.

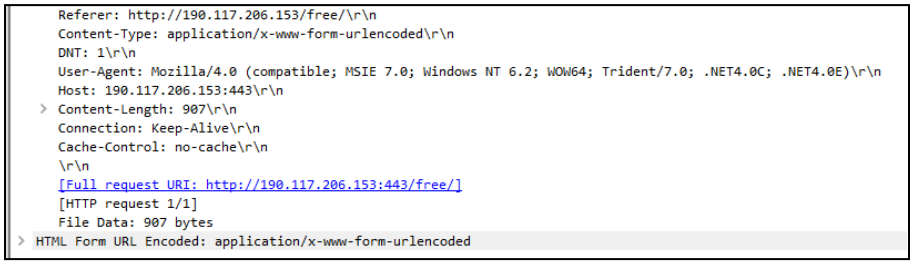

Figure 27. HTTP response to botnet IP

We identify the path /free/ as part of Emotet's protocol, essentially random directory names used as endpoints. The POST requests contained encrypted data, not plaintext HTTP payloads. This points to a bot sending a "check-in" message containing info about the infected

machine. Each of those servers presumably is an Emotet C2 node and acts as a botnet. We noticed the malware iterated through the 25 IPs and then seemed to stop, maybe waiting to retry later for communication back to the central C2. In Figure 28, we map out the entire botnet through IPInfo with our collected IPs.

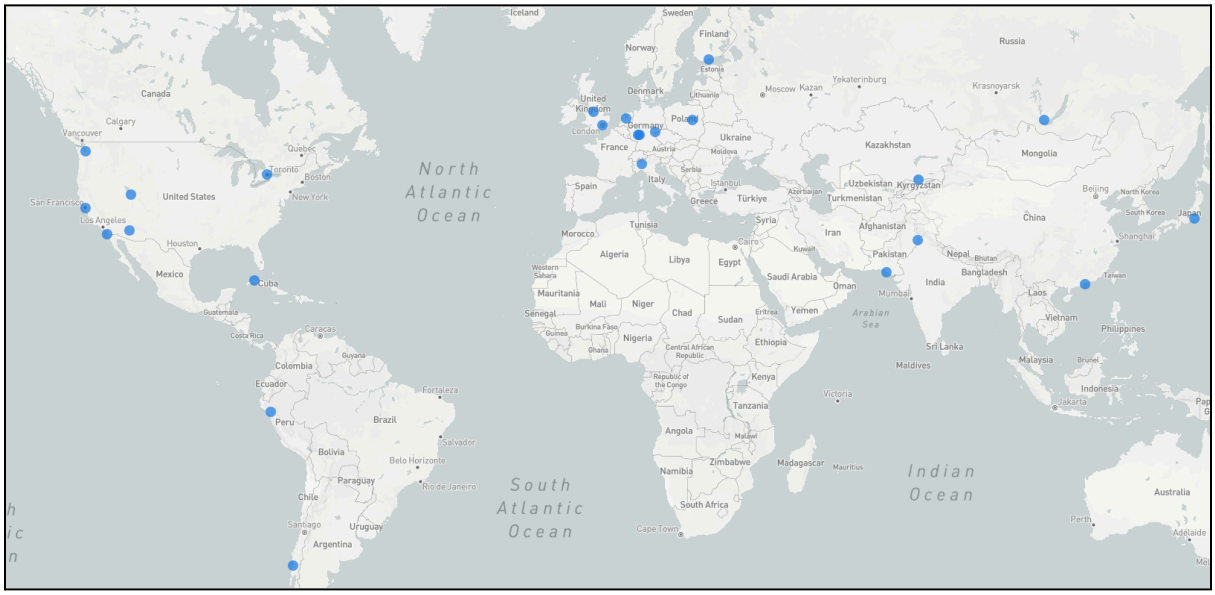

Figure 28. map of the botnet IP's used by our Emotet sample

We decided to look into Procmon and see the dynamically loaded DLL's to confirm our hypothesis from static analysis. In Figure 29, we identify multiple DLLs loaded by classtangent.exe

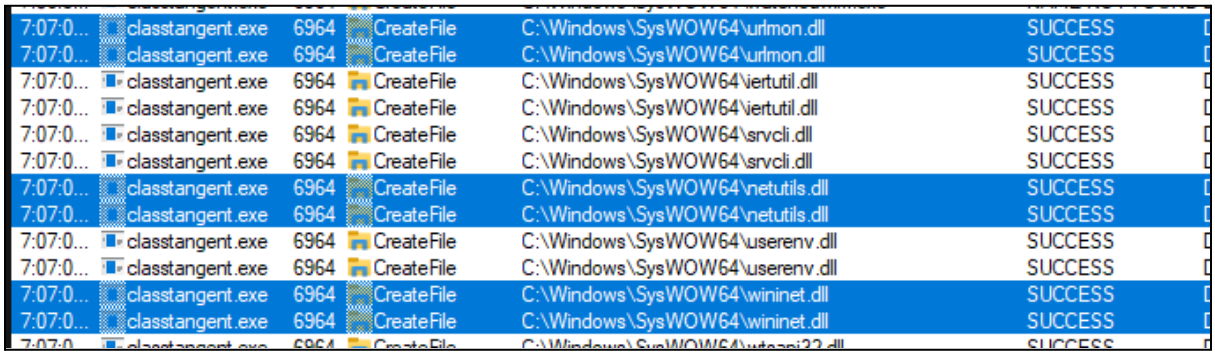

Figure 29. Dynamically added DLLs

We identify the Wininet.dll, which provides functions to issue HTTP requests, download payloads, and exfiltrate data over standard web ports [12]. The Urlmon.dll provides functions to

fetch files or scripts from remote URLs, using the same routines that Internet Explorer employs for downloads [13]. The Iertutill.dll may leverage browser components for the sample [14]. The srvcli.dll exposes RPC-based network shares and remote connection functions [15]. The malware may call Netutils.dll to modify network interfaces, such as the IP interfaces created above [16]. The userenv.dll allows the malware to apply group policy settings as a potential way to gain more access to the system [17].

For our last part of the analysis, we use ProcDot to make a process flow diagram of 267.exe. We pass in the downloaded CSV generated from ProcMon with a filter set for that specific process name. In Figure 30, we provide the diagram.

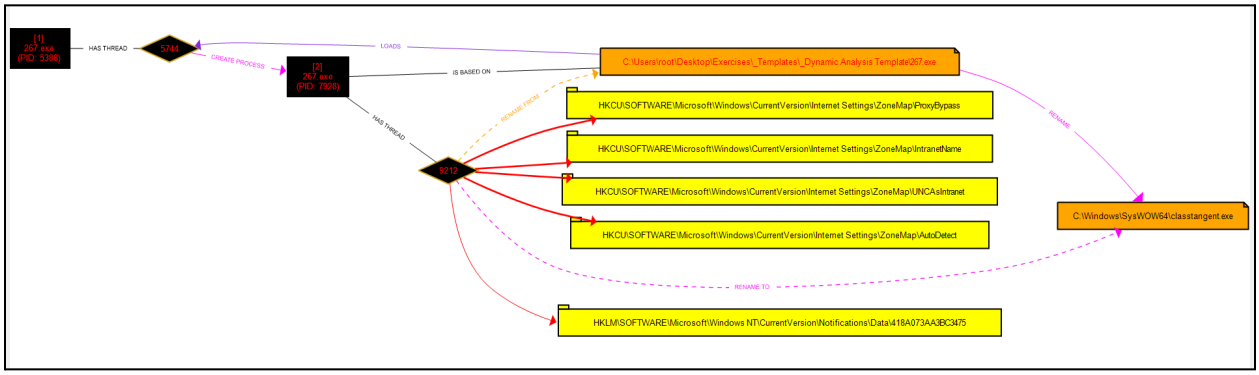

Figure 30. ProcDot graph of 267.exe

ProcDot provides a high-level overview of the 267.exe executable. It creates a duplicate child process, invokes multiple Windows API calls for network functionality, and creates the classtangent.exe executable. In Figure 31, we do the same and outline the high-level overview for the classtangent.exe executable.

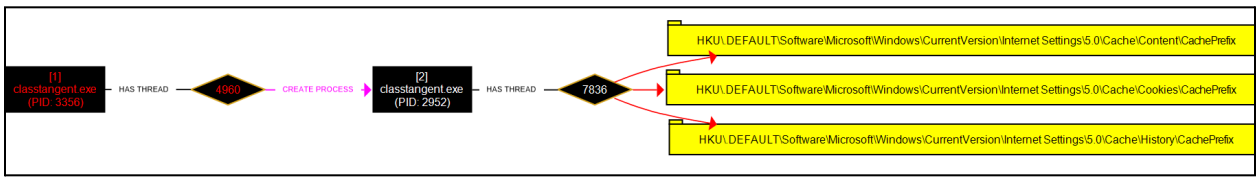

Figure 31. ProcDot graph of 267.exe

The classtangent.exe executable invokes itself and spins up a process, and then executes multiple registry key permissions for internet settings. The malware seems to tweak cache behavior on the system.

Finally, after compiling all these dynamic and static findings, we correlate them with Emotet's known profile: drop a file in WOW64, create a service, attempt network C2, and wait for further commands. Our sample matches all of these. We have analyzed 267.bin, identified it as Emotet, unpacked it, and observed its malicious activities. We provide our fully documented report in the appendix. In the next section, we assess the risks we outlined based on our assumptions for the sample and environment.

6. Risk Assessment

We assume that one legitimate user double-clicked the 267.bin sample for the initial compromise vector. If 267.bin exploited an unpatched service or exploited a vulnerability in an application, we may miss critical exploit chains and pre-execution unpackers. Furthermore, we may miss loader stubs that alter the sample's behavior in memory based on its environment. If we fail to report these stubs, it could mean we miss persistence mechanisms or environment-specific evasion techniques. Lastly, if multiple users across the Enterprise Network executed the sample, we would have no reliable way to pinpoint every infection. Our dynamic analysis would only capture one endpoint's behavior.

We assume each endpoint resolves correctly and answers HTTP requests as intended by the malware creator and that the author only utilizes HTTP requests and no other forms of communication through covert channels. If the real C2 infrastructure uses non-standard ports, mutual-TLS, or DNS tunneling, our sandbox may not observe real callbacks or how the malware

fetches payloads. We would underestimate the full capabilities of the malware, including the data-exfiltration methods and second-stage downloads.

We assume that our Windows 10 environment mirrors the target environment's default configuration and that the malware operates as intended. If the real network enforces custom group-policy configurations, antivirus measures, and non-standard DLL versions, the sample might detect or adapt to those defenses, and our sample may mask persistent mechanisms or anti-debug checks, and we won't see them in our analysis.

7. References

[1] Fortinet, "What Is a Trojan Horse Virus?," *Fortinet Cyber Glossary*, Available: https://www.fortinet.com/uk/resources/cyberglossary/trojan-horse-virus#:~:text=What%20Is%20A%20Trojan%20Horse,Virus (accessed Jul. 13, 2025).

[2] I. Jack, "Malware packers – Anti-Reverse Engineering (Part 3)," *Medium*, Available: https://medium.com/@iramjack8/malware-packers-9dcf6fa2f0e7#:~:text=While%20some%20packers%20have%20legitimate,We%20need%20to%20go%20deeper (accessed Jul. 13, 2025).

[3] Palo Alto Networks Unit 42, "What Is a Botnet?," *Palo Alto Networks Cyberpedia*, Available: https://www.paloaltonetworks.com/cyberpedia/what-is-botnet#:~:text=A%20botnet%20,data%20theft%2C%20and%20spam%20distribution (accessed Jul. 13, 2025).

[4] K. Yasar, "What Is a Command-and-Control Server (C&C Server)?," *TechTarget WhatIs*, Available: https://www.techtarget.com/whatis/definition/command-and-control-server-CC-server#:~:text=A%20command,carry%20out%20an%20extortion%20scheme (accessed Jul. 13, 2025).

[5] Cybersecurity and Infrastructure Security Agency (CISA), "AA20-280A: Emotet Malware," *CISA Cybersecurity Advisory*, Oct. 1, 2020. Available: https://www.cisa.gov/news-events/cybersecurity-advisories/aa20-280a (accessed Jul. 13, 2025).

[6] The Hacker News, "All You Need to Know About Emotet in 2022," Nov. 26, 2022. Available: https://thehackernews.com/2022/11/all-you-need-to-know-about-emotet-in.html#:~:text=It%20se

arches%20for%20email%20addresses,attacks%20that%20lead%20to%20ransomware (accessed Jul. 13, 2025).

[7] Lt. Col. S. Sonya, *Reverse Engineering* [Slides], U.S. Air Force AFRL/RIG course, 2025.

[8] M. C. B. Systems, "Use CertUtil to Get File Hash," *MCBSYS Blog*, Mar. 2017. Available: https://www.mcbsys.com/blog/2017/03/use-certutil-to-get-file-hash/ (accessed Jul. 13, 2025).

[9] Hasherezade, "mal_unpack," GitHub, Available: https://github.com/hasherezade/mal_unpack (accessed Jul. 13, 2025).

[10] IPInfo, "IPInfo Map Tool," *IPInfo.io*, Available: https://ipinfo.io/tools/map (accessed Jul. 13, 2025).

[11] Oracle, "Oracle VM VirtualBox," *VirtualBox*, Available: https://www.virtualbox.org/ (accessed Jul. 13, 2025).

[12] Microsoft, "InternetSetCookieW function (wininet.h)," *Microsoft Docs*, Available: https://docs.microsoft.com/windows/desktop/api/wininet/nf-wininet-internetsetcookiew (accessed Jul. 13, 2025).

[13] Microsoft Answers, "urlmon.dll," *Windows Forum*, Available: https://answers.microsoft.com/en-us/windows/forum/all/urlmondll/7736927e-2f69-44f2-bd4f-2191dc6dc78a (accessed Jul. 13, 2025).

[14] File.net, "iertutil.dll process," Available: https://www.file.net/process/iertutil.dll.html (accessed Jul. 13, 2025).

[15] HijackLibs, “srvcli.dll library,” Available: https://hijacklibs.net/entries/microsoft/built-in/srvcli.html (accessed Jul. 13, 2025).

[16] MalwareTips, “Netutils.dll – What It Is & How to Fix Errors,” *MalwareTips Blog*, Available: https://malwaretips.com/blogs/netutils-dll-what-it-is-how-to-fix-errors/ (accessed Jul. 13, 2025).

[17] Microsoft, “CreateProfile function (userenv.h),” *Microsoft Docs*, Available: https://learn.microsoft.com/windows/win32/api/userenv/nf-userenv-createprofile (accessed Jul. 13, 2025).

8. Appendix

OVERVIEW		
Name	267.bin; unpacked_267.bin	
Download Location	https://drive.google.com/drive/folders/1f8TJMxagrDEqAdBaHXC39jJdHFXiEfu-	
Size	384kb (packed binary); 93kb (unpacked binary)	
Hash	MD5	**SHA-256**
	267.bin: f3f48c57c38bff2ddd220f20569e1ee6 Unpacked_267.bin: 8db2aa9a70da99f0337ce90bd9ab58ac	267.bin: B1cad1540ecb290088252635f8e130022eed7486eb128c0ca3d676945d60a9fc Unpacked_267.bin: A9c84dd3c20a9ecebdbcbd7f28e40a17160e718b0d8d7dab67f959482f92ab9d
Virus Total Link	267.bin: VirusTotal - File - b1cad1540ecb290088252635f8e130022eed7486eb128c0ca3d676945d60a9fc https://www.virustotal.com/gui/file/b1cad1540ecb290088252635f8e130022eed7486eb128c0ca3d676945d60a9fc Unpacked_267.bin: https://www.virustotal.com/gui/file/a9c84dd3c20a9ecebdbcbd7f28e40a17160e718b0d8d7dab67f959482f92ab9d?nocache=1	
Any.run Link	267.bin:	

	https://app.any.run/tasks/2acaea41-e35d-4227-9128-5ab673c1c087 Unpacked_267.bin: https://app.any.run/tasks/3071478f-c396-4529-8dbb-e0afb25d0f0d	
Detection Date	**Packed version:** Creation Time 2019-10-06 18:38:52 UTC First Seen In The Wild 2019-10-13 14:16:20 UTC First Submission 2019-10-13 08:27:20 UTC Last Submission 2025-07-10 01:10:15 UTC Last Analysis 2025-07-09 22:44:38 UTC	
Main Detection Vendors	267.bin Alibaba: Trojan:Win32/Emotet.e952c439 Google: Detected AVG: Win32:MalwareX-gen [Bank] Microsoft: Trojan:Win32/Emotet.PG!MTB Symantec: Trojan.Gen.MBT MalwareBytes: Malware.AI.4218689116	Unpacked_267.bin Google: Detected AVG: Win32:trojan-gen Microsoft: Trojan:Win32/Emotet.DHF!MTB Symantec:ML.Attribute.HighConfidence MalwareBytes: Emotet.Trojan.Dropper.DDS Avast: Win32:Trojan-gen

Reported File Names	**Packed version:** 267.bin classtangent.exe 267.exe b1cad1540ecb290088252635f8e130022eed7486eb128c0ca3d676945d60a9fc.exe zararli.exe oIqXjben tabletthemes_exe 267.bin.exe b1cad1540ecb290088252635f8e130022eed7486eb128c0ca3d676945d60a9fc 2.exe magnifybackup.exe
Malware Family Class	trojan.emotet
Online References	https://medium.com/threat-intel/emotet-dangerous-malware-keeps-on-evolving-ac84aadbb8de
Definite Indications of System Infection	HOST INDICATIONS
	Creation of a service called “classtangent”
	Creation of a file called “classtangent.exe” in the C:\WINDOWS\SysWOW64\ directory
	NETWORK INDICATIONS
	Attempted TCP connections to 90.117.206.153, 203.99.187.137, 200.55.168.82, and 70.32.94.58.

STATIC ANALYSIS		
HEADER	**DATA**	**COMMENTS**
Packer Used	Custom	May use VirtualAlloc

	PATH	COMMENTS
Significant DLLs (Dependencies)	Kernel32.dll, user32.dll	Contains the sleep and getTickCount functions commonly used for anti-debugging mechanisms.
	urlmon.dll, netutils.dll, and wininet.dll	These dll files contain various functions that enable network communication.

DYNAMIC ANALYSIS		
HEADER	**PATH**	**COMMENTS**
Significant Folder(s) Created	C:\WINDOWS\SysWOW64\	Folder used to store classtangent.exe malware
Significant File(s) Created/Modified	classtangent.exe	Executable that acts as a callback for the botnet
	267.exe	Child process created by the original 267.exe

HEADER	**PATH**	**HASH**
DLL's Created	None identified	N/A
DLL's Modified	None identified	N/A
Significant Registry Key Creation(s)	**LOCATION**	
	HKLM\SYSTEM\ControlSet001\Services\classtangent HKLM\SYSTEM\WPA\8DEC0AF1-0341-4b93-85CD-72606C2DF94C-7P-37 HKLM\SYSTEM\CurrentControlSet\Services\classtangent 	
Significant Registry Key Modification(s)	**DATA**	**COMMENTS**
	HKU\.Default\Software\Microsoft\Windows\CurrentVersion\Internet	

	Settings\5.0\Cache\Content\CachePrefix	
	HKU\.Default\Software\Microsoft\Windows\CurrentVersion\Internet Settings\5.0\Cache\Cookies\CachePrefix	
	HKU\.Default\Software\Microsoft\Windows\CurrentVersion\Internet Settings\5.0\Cache\Content\History\CachePrefix	
	HKLM\SYSTEM\ControlSet001\Services\Tcpip\Parameters\Interfaces\{6c5f8f56-58a5-4346-be3f-66779ddaf127}\T1 0x686E10B8 → 0x68672434 HKLM\SYSTEM\ControlSet001\Services\Tcpip\Parameters\Interfaces\{6c5f8f56-58a5-4346-be3f-66779ddaf127}\T2 0x686E8F1B → 0x6872CDC4 HKLM\SYSTEM\ControlSet001\Services\Tcpip\Parameters\Interfaces\{6c5f8f56-58a5-4346-be3f-66779ddaf127}\LeaseTerminatesTime 0x686E894B → 0x6872F7F4	Used to create various TCP parameters for the botnet

	HKLM\SYSTEM\ControlSet001\Services\Tcpip\Parameters\Interfaces\{6c5f8f56-58a5-4346-be3f-66779ddaf127}\DhcpInterfaceOptions FC 00 00 00 00 00 00 00 (no prior value)	
Services Created	classtangent	
Persistence/Startup	The classtangent service is configured to start automatically.	
Kernel Hooks/Rootkit Operations	None identified	
Software Protection / Anti-Debugging Techniques	The original and unpacked binaries pass execution to child processes so they can terminate themselves after startup. The unpacked binary calls the sleep and getTickCount functions which are commonly used for anti-debugging features. It loads DLLs dynamically.	
	ADDRESS	**COMMENTS**
URL Requests	190.117.206.153 203.99.187.137 200.55.168.82 70.32.94.58 213.138.100.98 144.76.62.10	Suspected IPs of C2 servers

	203.99.182.135 176.58.93.123 192.241.220.183 94.177.253.126 216.75.37.196 95.216.207.86 78.109.34.178 113.52.135.33 216.70.88.55 138.197.140.163 83.169.33.157 212.112.113.235 143.95.101.72 190.13.146.47 178.249.187.150 157.7.164.178 5.189.148.98 51.38.134.203 93.78.205.196	

Malware is a part of the Emotet family.
N/A

TOOL	COMMENTS
Certutil	Computed MD5 and SHA-256 hashes for VirusTotal and IOC gathering.
VMWare	Virtual Machine “Blast Chamber”
VirtualBox	Virtual Machine “Blast Chamber”
What’sRunning	Compare snapshots of live system state before and after infection, view active TCP and Binary information
Mal_Unpack	Automated runtime unpacking of the encrypted payload to disk for deeper static analysis.
RegShot	Compare registry snapshots to determine new additions after infection

Procmon	Monitor file system and registry changes in real-time by each process
Strings	Retrieve textual data embedded in the sample
DependencyWalker (Depends.exe)	Static Analysis of Imported DLLs and Functions
Wireshark	Analyze network traffic
VirusTotal	Queried the sample's hashes to confirm malware family classification, review AV detections, and collect additional IOC details.
FakeNet-NG	Simulated Internet services locally and intercepted HTTP POSTs to C2 servers without touching the real network.
ANY.RUN	Performed an independent interactive sandbox run to verify process behavior, service creation, and network callbacks.
ProcDot	Visualized ProcMon logs into a process-flow graph showing drop-and-run behavior and registry modifications.
IPInfo	Looked up geolocation and ownership for observed C2 IP addresses to contextualize the attacker infrastructure.
Detect-It-Easy (DiE)	Measured file entropy and attempted to identify known packers.