Malware Analysis Journey

# Intro

This blog post aims to inform others about the process of malware analysis while documenting my own journey into this area. A number of guides about malware analysis can be found on internet, but not many focus on a beginner’s attempt of just sitting down and attempting to do it. I’m addressing that by writing about my mistakes and successes for others to learn from, as well as my thought process and explanations into my actions. The journey itself is actually quite long, and while I would like to explain everything I learned, this article will primarily focus on interesting things I found and the journey as a whole. We will go through:

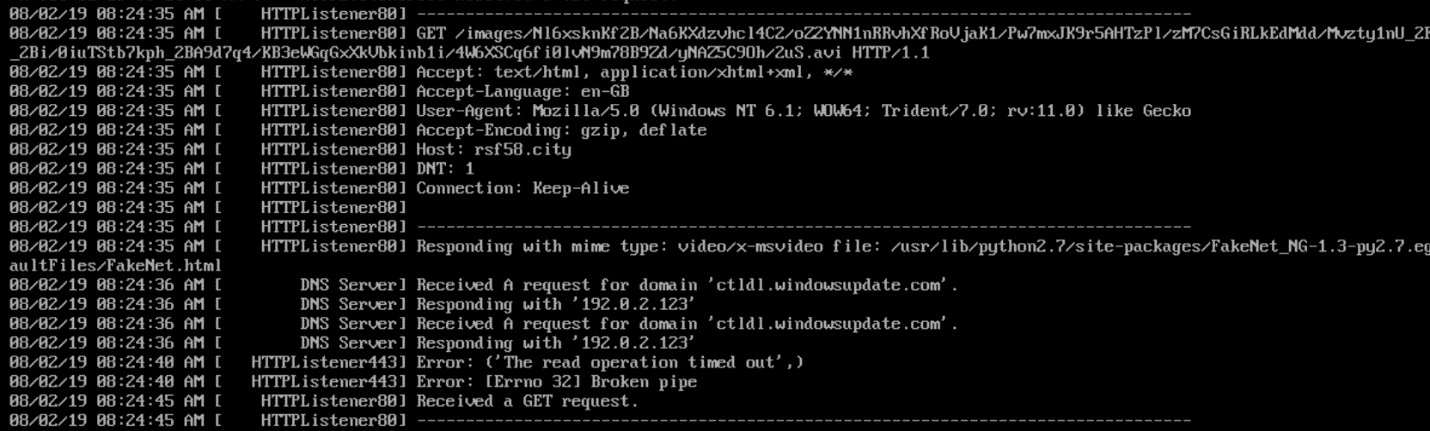
* Word Macros
* Unpacking malware
* Discovering malware capabilities
* Malware obfuscation techniques

The article will combine both technical and non-technical portions, as well as some bonus resources for people who want to explore a bit more in the orange sections.

# 1.1 Malware Background

Most malware write-ups focus on well-known malwares (e.g. Emotet or NotPetya), and this malware is no exception; except that I got it from my parents. My mom was emailed a word document from her boss’s hacked email account and after opening it figured something was wrong. She sent the document to me figuring I could help her find out what it does. The first issue I ran into was the difficultly of sending malware via email, as Outlook will automatically remove known viruses and Google Drive will not allow you to open known viruses. I found the easiest way to avoid this was to zip the file because the compression would bypass Google’s virus detection. I also applied the best practices which state the zip file should be a password protected and the content’s file extension removed (.doc) and replaced with .txt so it isn’t accidentally executed.

Now that I actually had the malware, it was time to setup the analysis environment. I was fortunate enough to be using a specially designed testing environment provided by Context, but in general you would need to set up your own. Most environments will be similar to Context’s, involving a virtual machine, usually on its own network, which contains all the necessary tools to analyze malware. A Virtual Machine (VM) is a computer that runs on top of your own computer. It uses the same resources as your computer except they are partitioned to be separate. Virtual machines are generally used because of their “snapshot” feature and their separation from our own computer, mitigating damage if our malware is destructive (e.g. ransomware). The “snapshot” feature is a really useful virtual machine function that acts like a save point, capturing the state of the machine and allowing you to return back to that point of time when you wish. While these are all good reasons to use virtual machines, it should be noted that sophisticated malware can detect if it is in a VM and either breakout onto your main computer or act differently.

Another feature of Context’s environment is FakeNet. Generally environments are setup without internet connection for isolation purposes, however some malware can detect this behavior and terminate. Fakenet is a solution to this problem by basically acting as a fake internet. The program runs on another server and sends acknowledgements to any web requests that the virtual environment might make. Now that we have covered the background, let’s begin analysis.

(An example of FakeNet responding to HTTP requests)

[Emotet Malware: https://www.malwarebytes.com/emotet/

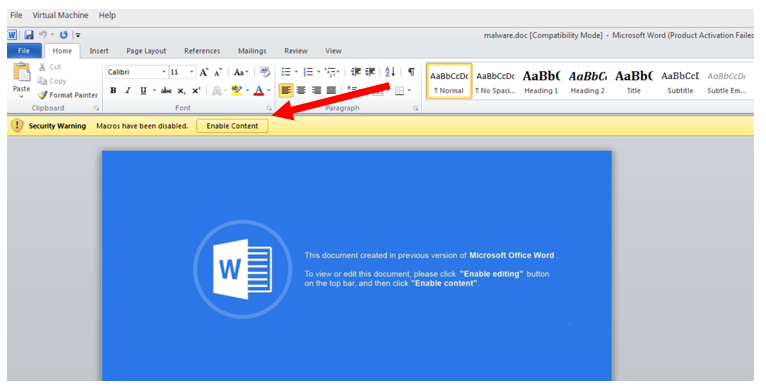
Petya and WannaCry: https://en.wikipedia.org/wiki/WannaCry\_ransomware\_attack

If you are curious about NotPetya, the malware analyst who stopped it has now been sentenced to jail: https://krebsonsecurity.com/2019/04/marcus-malwaretech-hutchins-pleads-guilty-to-writing-selling-banking-malware/

Creating your own environment: https://www.malwaretech.com/2017/11/creating-a-simple-free-malware-analysis-environment.html

Ransomware: <https://www.us-cert.gov/Ransomware>]

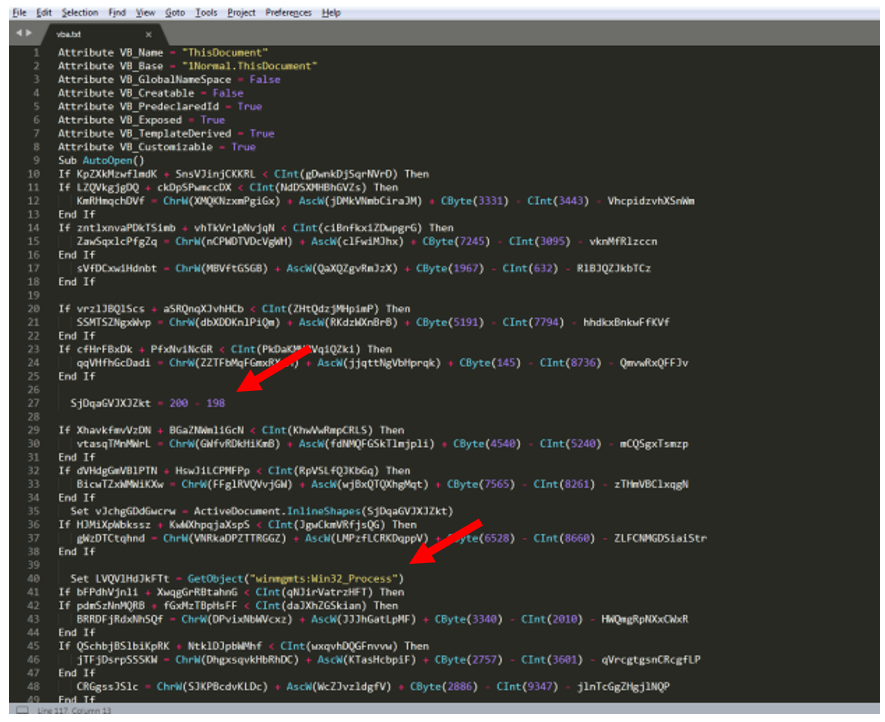
# 2.0 Word Document



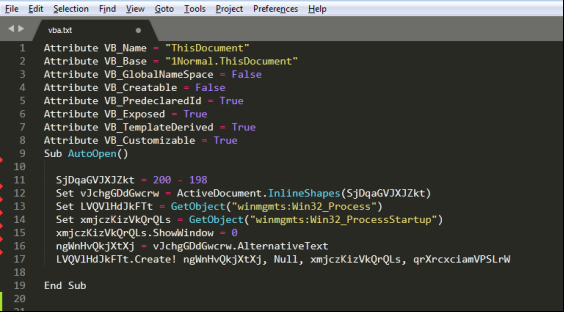
(Opening page of the document)

Opening up the Word document in our VM, we are greeted with a page asking for macros to be enabled in order to view the content. Clicking enable content will run the macro and execute the malicious code on our system. This is because a macro is essentially a script that runs Visual Basic for Applications (VBA) code. Macros are normally used to automate tasks in Word, such as adding a letterhead to a document , but malicious users can weaponize this feature. A common technique used by malicious users is to embed a macro into a word document that, when run, downloads and executes malware. By default, Word disables macros from untrusted documents (i.e. downloaded from the internet), as there shouldn’t be downloaded documents automatically running code upon opening. However, the technique has a high success rate as not many people know what a word macro is. To get an idea of what this macro does, we need to open it up.

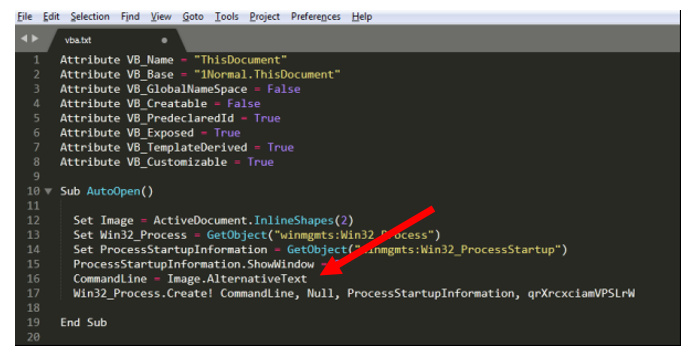
## 2.1 Word Macro



Opening up the macro in Sublime Text (a text editor), we can see the code is obfuscated, meaning it has been made deliberately difficult to read. Using our coding knowledge, we can see that the random variables names in the if statements don’t seem do anything. Scrolling the macro manually and searching for multiple uses of variables, I deleted all of the if statements and ended up with the following.



Formatting the macro by naming variables and moving code around produces this.

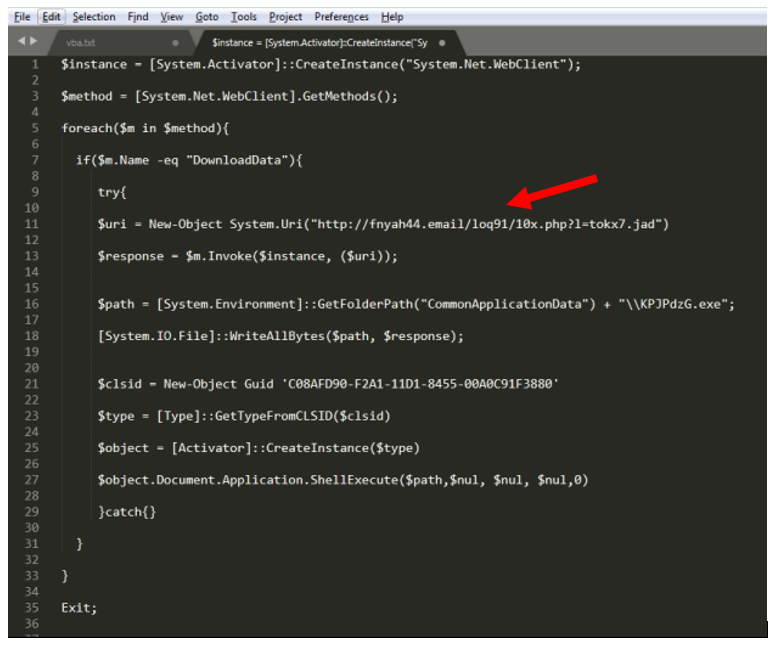


Looking at the code, we need to know what command line is being executed in order to figure out what the macro does. We can see that the command line is hidden in the alternative text of an image, which is the text that is displayed instead of an image if the image can’t load. We should be able to view this either by opening the alt-text of the image or by printing it out when the macro runs which is what I will be doing.

[AutoOpen: https://support.microsoft.com/en-gb/help/286310/description-of-behaviors-of-autoexec-and-autoopen-macros-in-word

Win32\_process: https://docs.microsoft.com/en-us/windows/win32/cimwin32prov/create-method-in-class-win32-process]

## 2.2 PowerShell

Great, our command seemed to work and we have successfully printed out the command line (shown in the debug window). Reading the command line, it seems to execute a PowerShell command which is Base64 encoded. I won’t go into detail about Base64, but it is a simple encoding algorithm that can be easily decoded. Decoding this command and formatting it nicely we get the following.

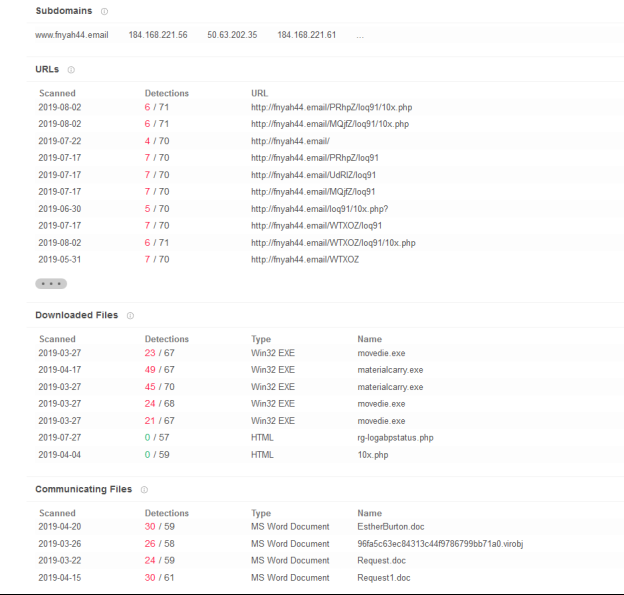
Just looking at the code, we can see that it is trying to download data from a URL called fnyah44.email and is trying to store the data in C:\ProgramData\ and execute it. This confirms our suspicion that the word document was simply a dropper and the downloaded data is most likely our malware. If the malware domain was still up and running, I would have grabbed the malware from the folder and begun analyzing. However, by the time I had received the malware, the domain had expired and I was forced to use other methods.

[/c: https://stackoverflow.com/questions/515309/what-does-cmd-c-mean

Base64: https://en.wikipedia.org/wiki/Base64

PowerShell: https://docs.microsoft.com/en-us/powershell/scripting/overview?view=powershell-6]

## 2.3 VirusTotal

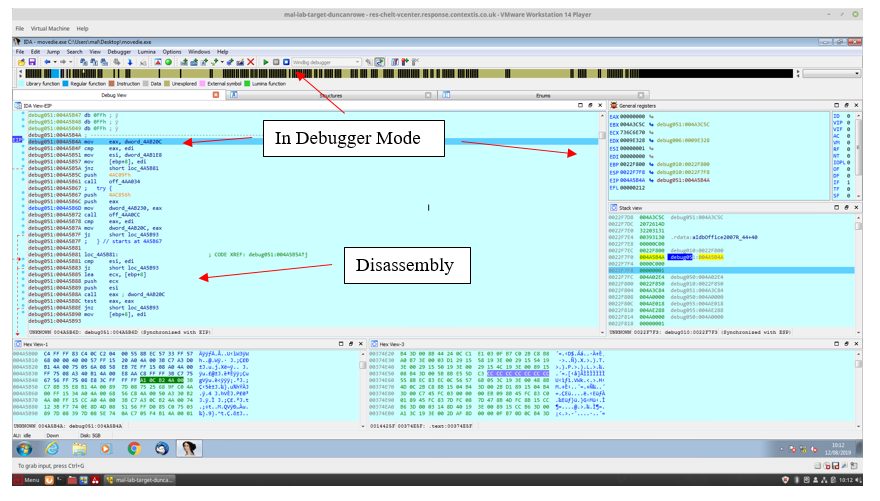
One of the ways we can download malware, besides visiting a malicious site, is to look for it on websites that analyze and store malware data like VirusTotal. VirusTotal is a well-known website used to share malware research. Anyone can upload malicious files to their website and VirusTotal will automatically analyze and create a page on its findings. Other users can see and visit this page where they can start a discussion and share their own research. It is a useful tool to see how common the malware is and the analysis can help give a rough idea of what the malware does. For my purposes, I wanted to retrieve malware that was communicating with our domain fnyah44.email and download it for analysis.

(VirusTotal fnyah44.email page)Typing our URL into VirusTotal, we can see a page has already been created for our domain (shown above). Viewing it, we can see that the URL format fnyah44.email/loq91/10x.php matches other URL’s and, more interestingly, we can view the files that would have been downloaded if the malware domain was running. Unfortunately, in order to download malware from VirusTotal a vetted account is needed, but this wasn’t a problem as Context was able provided me with the files. Looking through the malicious files, the most interesting for me was going to be the 10x.php file, as that was what our malware was attempting to communicate with. However, it seemed to just be a 404 error so I moved onto the executables instead. Looking at the basic analysis that was performed, VirusTotal seem to believe they were trojans. A trojan is a type of malware that masquerades as legitimate software (like a trojan horse) and generally focuses on information, either stealing it, modifying it or deleting it. Looking at the analysis, two main executables were downloaded and one of them, movedie.exe didn’t have a full analysis available and a lower detection rate amongst anti-virus (AV) programs. This made me curious enough to choose movedie.exe to perform my analysis on.

[404 Error: https://www.prestashop.com/en/blog/404-not-found-error-how-to-fix-it

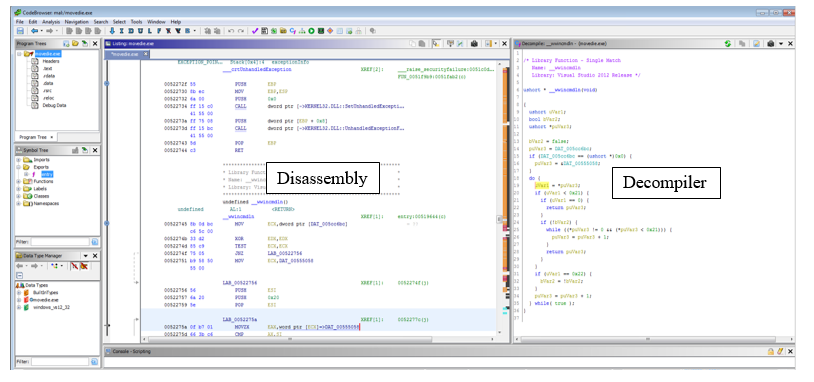
VirusTotal: https://www.virustotal.com/gui/domain/fnyah44.email/details]

# 3.0 MoveDie.exe

Ghidra and IDA Pro are toolkits that help reverse engineer software by disassembling them into the most basic human-readable language form, assembly. IDA Pro today is considered the industry standard toolkit because of its variety of features. It can correctly display assembly instructions for a program as well as create a kind of pseudo code (if hex-rays decompiler is installed). It also includes a debugger which allows you to step through the assembly and see the program change as you execute each instruction (used for dynamic analysis). There are also a bunch of plugins built by developers that allow many other features such as the disassembling of different architecture types, but all this comes at a hefty cost ($1200).

(IDA Pro View)

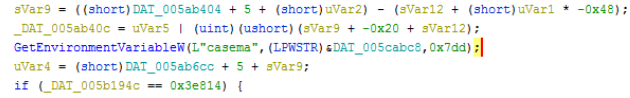
On the other side we have the NSA released tool, Ghidra, which is free and has these all of these features (except for debugging which is planned to be added). We will be using both of these programs for malware analysis.



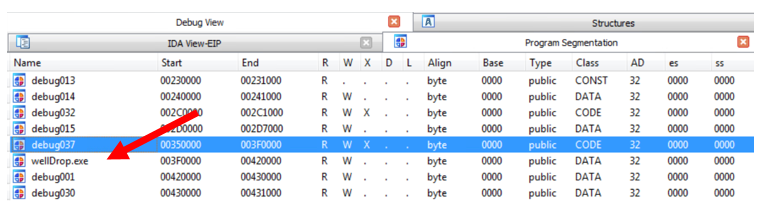
(Ghidra View)

## 3.1 Analysis

When I first started static analysis, I spent a while looking through the default setups of getting environment variables and computers names. However, after these default functions calls I managed to find an interesting tidbit, one of the reasons why I wanted to analyze movedie.exe.



Above is a snippet from the decompiler in Ghidra. As shown, the malware checks for an environment variable called ‘casema’. Casema seems to be a Dutch trading company and my first thought was that this was some sort of targeted malware where the malware authors only wanted it to execute on that specific company. However, one of my colleagues suggested that it was a kill switch instead, where if the environment variable was set to a certain value it wouldn’t run. Looking at the code it seems regardless of its value it continues execution. No clue what this does, but it made me curious as this was the only “malicious code” I could find in movedie.exe so I decided to change approach.

I opened movedie.exe up in IDA Pro to take a look at the malware dynamically. While my version of IDA didn’t have hex-rays decompiler paid for (meaning I couldn’t look at the decompiled code), it did have a debugger. This meant that I could step through the code and see real changes in the malware. After stepping through for a bit, I found a call to a module called welldrop.exe, which is essentially another program inside of movedie.exe. Here is the welldrop.exe segment in movedie.exe’s memory.

To enter welldrop.exe, the malware calls a register value (ESI) which points to a dynamic address inside of welldrop.exe and redirects the execution flow. This is why dynamic analysis is important as static analysis would not have known what that value was as it is dynamically created.

[Assembly: https://www.tutorialspoint.com/assembly\_programming/assembly\_quick\_guide.htm

Ghidra Update: https://www.cyberscoop.com/ghidra-nsa-new-version-black-hat-2019/]

3.2 WellDrop.exe

Looking through welldrop.exe, I originally thought it was the main bulk of malware. Continuing to step through it however, I couldn’t find anything malicious about it. After a while, I remembered some common malware techniques I learned while watching a YouTube video from OALabs. OALabs is a malware analysis channel and they had recently created a video on unpacking malware. Re-watching the video for a refresher, it turns out that most malware hide themselves in a type of program called a packer which helps obfuscate themselves so that anti-virus software have a harder time detecting them. This packer uses a variety of techniques to unpack the payload inside itself and run it. Essentially the program that I downloaded from VirusTotal is a dropper for the actual malware and we needed to dig a bit deeper.

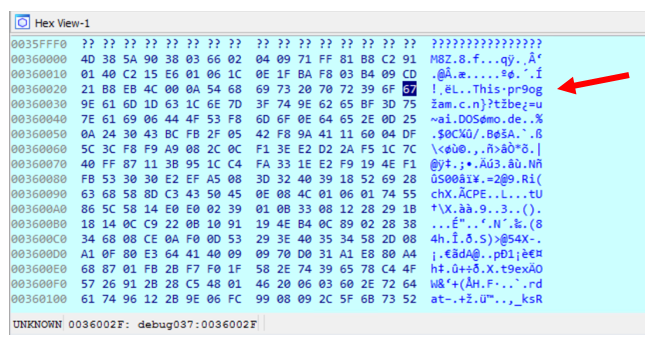
[OALabs: https://www.youtube.com/channel/UC--DwaiMV-jtO-6EvmKOnqg

Video: https://www.youtube.com/watch?v=YXnNO3TipvM

Article about the video which is a bit clearer: https://liveoverflow.com/unpacking-buhtrap-malware-basics-of-self-injection-packers-ft-oalabs-2/]

## 3.3 Packer

A packer, by definition, needs some way of unpacking the malware hidden inside itself in order to execute it. This can be done in multiple ways, but I’m going to focus on my particular packer’s technique known as self-injection. In essence the packer will unpack a “stub” into memory and then transfer execution to this stub. The stub then allocate the malware into a section of the packer’s memory and changes the permissions of this area to execution and executes it. One of the more common ways to do this is to use Windows API calls such as VirtualAlloc and VirtualProtect. Windows API consists of a series of DLL’s that are part of the operating system and have functions other programs can use. Since all Windows computers have these functions, the malware can use VirtualAlloc to allocate a region of pages with certain permissions and then uses VirtualProtect to change those permissions. Sure enough, stepping through welldrop.exe we can see VirtualAlloc being called three times. Our first call allocates some memory (A) then redirect the execution flow into location A and allocates two more sections (B and C). Based on our knowledge above we can see that memory A is the stub and the malware should be in memory B or C. Viewing these sections in IDA-Pro, neither memory B nor C look to contain any information. However upon closer inspection, the beginning of memory C looks like a weird MZ header.



Headers are at the beginning (head) of files and help identify what type of file they are to programs running them. They have a specific structure and therefore the ASCI representation of the hex characters “MZ” are always at the beginning of a PE (portable executable) header as well as the phrase “This program cannot be run in DOS mode”. However, in this case we can see it has an 8 in the middle of MZ and only parts of the DOS message. Some quick googling leads us to see that our malware is compressed with LZ-based compression and that there is a python tool that can help decompress it. After dumping the data from IDA-Pro and decompressing the malware we finally have our payload.

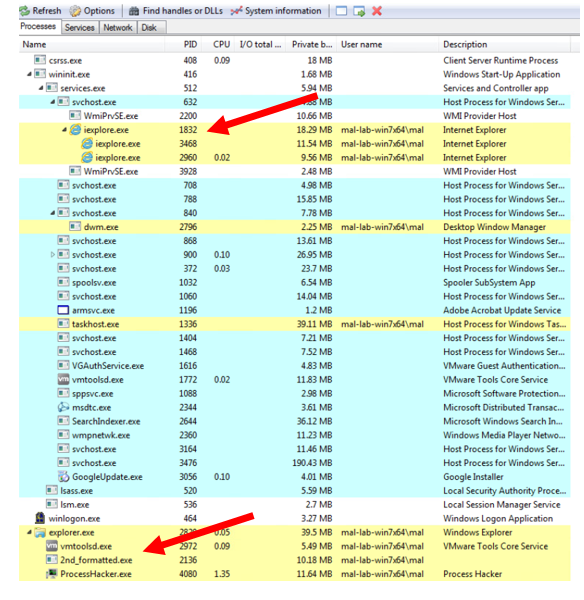
[Types of DLL Injections: https://www.endgame.com/blog/technical-blog/ten-process-injection-techniques-technical-survey-common-and-trending-process

Permissions: https://www.linux.com/learn/understanding-linux-file-permissions (it’s for files but the knowledge of the types of permissions are the same)

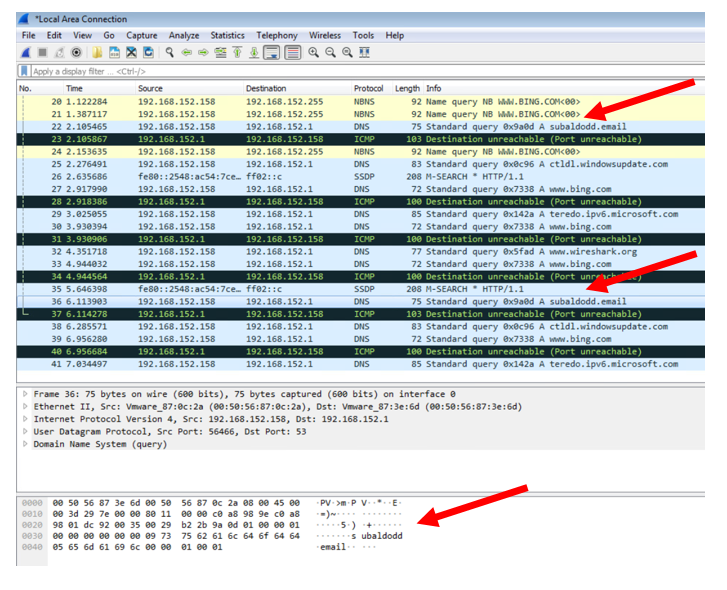
PE Header: https://docs.microsoft.com/en-us/windows/win32/debug/pe-format

LZ Decompression: https://github.com/herrcore/aplib-ripper]

# 4.0 Malware.exe

I decided at this point that it would be wise to take a break from reverse engineering and see what running the malware did. To help check what processes would be launched, I started a program called Process Hacker 2 and to see the outbound network traffic I launched Wireshark. Process Hacker 2 is shown below and is quite similar to task manager in that it lists all user processes currently running on the machine, but it organizes the information more clearly.

(Process Hacker View)

Looking at Process Hacker, I noticed that an Internet Explorer instance seemed to be running under a svchost.exe process which is quite suspicious. Svchost.exe is a system process that can host multiple Windows services. A DLL (Dynamically Linked Library) is a collection of functions that can be used by multiple programs concurrently. They are loaded by svchost.exe for the hosted application to use as DLL’s can’t be run by themselves. Normally Internet Explorer is launched from the desktop, so its parent process would be explorer.exe. However in this case, since it is under svchost.exe, we can conclude it wasn’t launched from a user but probably from our malware. More bizarrely, since it was launched from our malware, 2nd\_formatted.exe, it should appear as a child process but our binary has no child processes. I would later find out (through an article linked below) that this is because when we ran our PowerShell commands, they used WMI functions to call the process. This means that our process would run as the WMI executable, WmiPrvSE.exe which we see above. A good reason for malware to launch Internet Explorer is to send information over the network so I decided to check out Wireshark. Wireshark is just a packet sniffing program that records all of the traffic being sent from the machine to the internet and back.

As shown in the above Wireshark screenshot, the malware tries to resolve the following domain name:

* wrladolph.city
* rsf58.city
* subaldodd.email

Looking these domains up online, we see that they are well known command and control (C&C) servers where a C&C server is a centralized server that provides instructions to the malware for it to perform. Unfortunately again, the command and control servers were down when I reached this stage. Now that I had finally figured out what the malware was supposed to do, it was time to go back into reverse engineering and see if it did anything else.

[WMI: https://www.cbronline.com/news/ursnif-malware-c2]

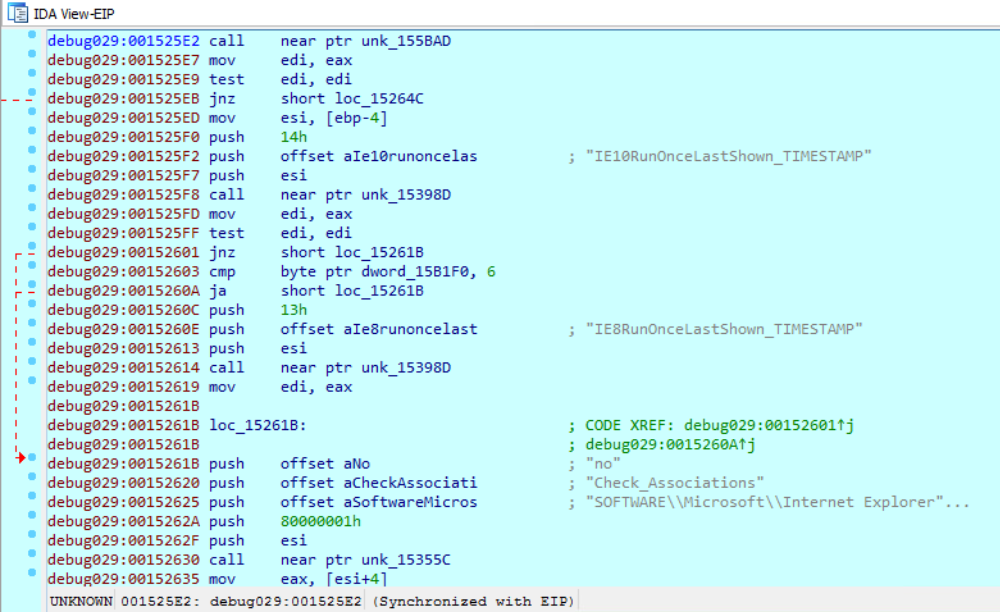
## 4.1 Domain Names

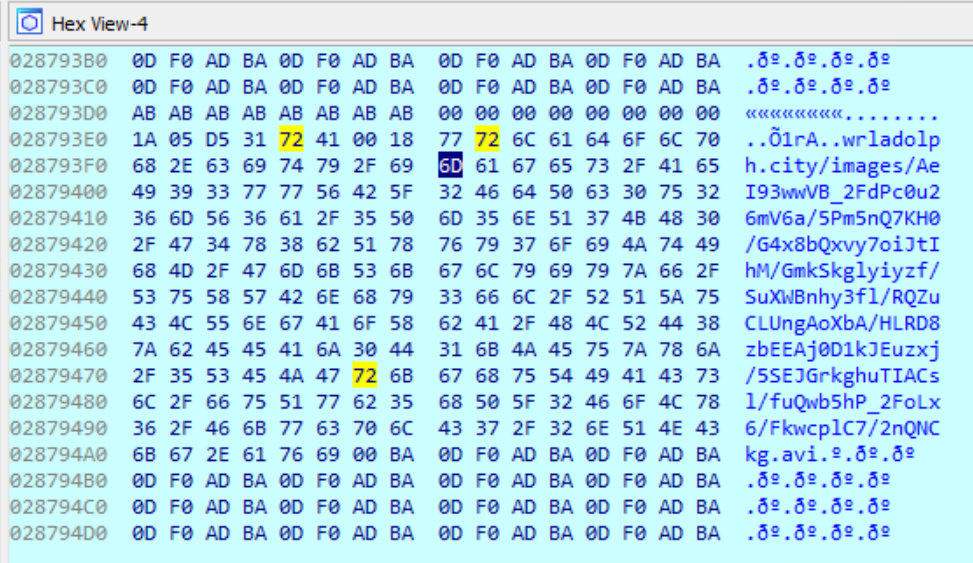
(Section of malware memory)

Stepping through the program, I found the three C&C server names in memory as well as other URL’s. One that caught my eye was constitution.org as URL’s containing the .org extension are normally official sites. I was curious what malware could be using this for and a quick google brought me to an interesting article. It stated that older malware use to grab words from the US Declaration of Independence and string them together to form random domains using a technique known as Domain Generation Algorithms (DGA). Through the use of DGA’s, malware is able to keep communicating with their C&C servers even if a C&C server goes down or is blocked. However, while this was interesting, it seemed that our malware wasn’t actually using the DGA as the malware didn’t reference any of the code. This makes me think that maybe the malware author used a tool to create the malware and that this was a setting in that toolkit, however I am uncertain. A quick visit to the website constitution.org shows us that the link is now broken and won’t work (it seems they removed the .txt file).

[Constitution.org: <https://unit42.paloaltonetworks.com/rovnix-declaration-generation-algorithm/>]

## 4.2 Internet Explorer

Further along in the malware, we can see a couple of registry values: IE10 and IE8 (Internet Explorer versions) as well as the setup for a Check Association’s call. Check Association’s is a Windows call that checks if the user has permission to access a certain registry subkey or entry.

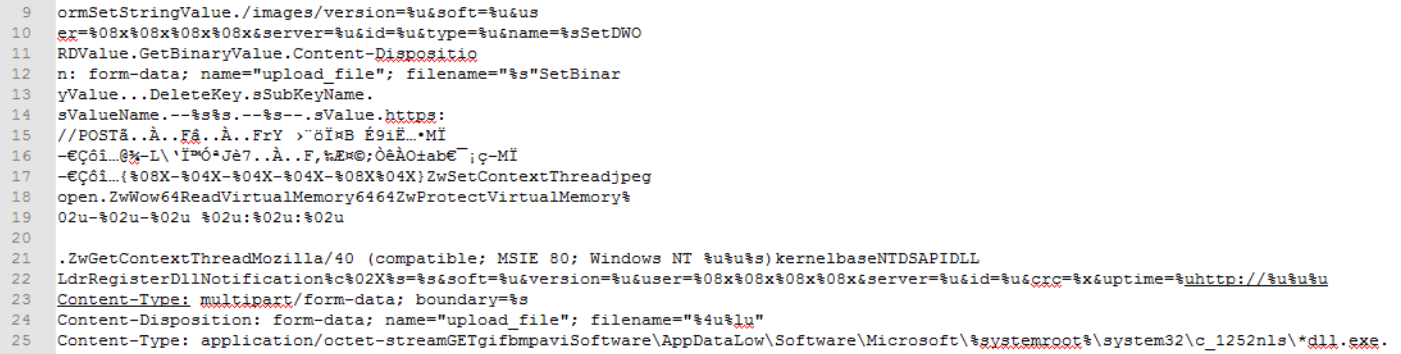
I believe this call is used to check if the malware has permission to access Internet Explorer. If it can, then the malware builds a link to one of the three C&C server’s for Internet Explorer to send a request to.

From memory, we can see with the link’s format, with first the domain followed by an images path and other random letters paths, ending with a file extension of .avi. The file extension is a bit strange, .avi is a type of video file and on its own isn’t malicious. There are some malware techniques where metadata can be written so that the .avi file will exploit something in the media player but I find this unlikely. The answer would come later as I continued my investigation, so let’s move on. Unfortunately again, the C&C server is down and we were unable to download a .avi file from VirusTotal as no other virus downloaded a .avi file. However, we can do some reconnaissance to gain insight into our malware’s capabilities.

## 4.3 Reconnaissance

In order to discover what the malware can do, we first need to finish reverse engineering it. Continuing to step through the program, we see that the malware receives the HTTP response to its download request in memory.

The response contains the default Internet Explorer response for no connection. The malware does not continue from beyond this point and due to the obfuscation and dynamic nature of the malware, I was unable to find what the malware would do with a valid response. Even changing the code to pass all of the branching statements was unhelpful as the program kept crashing after a certain point.

Another way I attempted to find out the malware’s capabilities was by viewing its HTTP request in memory. We already know some of the requests the malware sends due to FakeNet and Wireshark, as well as the strings the malware uses for those requests. If we find where the malware stores those strings in memory, maybe we could see some of the strings the malware would use in other requests. Searching around a bit, I found where the binary was retrieving those strings and copied them into notepad for formatting, as shown below.

We can see some of the strings that were used to build our initial request as well as some additional strings. We can see some that can point to a post request and some sort of upload functionality (line 12, 15, 24). It also looks like the malware can add a .gif or .bmp extension as well as .avi (line 25). While we can’t determine the nature of these strings, we have gained some insight. To get some more information, we can move onto the C&C side.

## 4.4 C&C Server Results

While our malware doesn’t respond to FakeNet’s responses, we are also unable to see its normal response to its C&C server as they have been shut down. However, looking at the C&C servers on VirusTotal, we see files that other successful malware have downloaded and base our malware’s capabilities off of their behavior.

Scrolling through the domains, there seem to be a lot of community comments, which can help tell us what some of the downloads do. Notable downloads include an advertisement tracking pixel, an html login page and another trojan. From this information, I would guess that this malware was a dropper and was used to establish a foothold on the computer. It would download more types of malware and do whatever the malicious users wanted. While our own analysis has been limited by the lack of an active C&C server, other researchers have been luckier.

## 4.5 Ursnif Malware Family

While reading an article about the bad naming policies of anti-virus detection programs, it stated that anti-virus results were better for unpacked malware. I decided to test this and uploaded my own version of the unpacked malware and saw that it was identified as part of the Ursnif Malware family. Looking back at the past VirusTotal results for the original packed malware, there were some AV programs that identified Ursnif, but many didn’t which created mixed results. However, in this case, it became clear that our malware was part of Ursnif once I researched it. The style of infection, the formatting of the C&C URL’s, as well as the binaries behavior all matched. This was very good news as now I could check my work against professionals and see how much I found (or missed).

Ursnif is a banking Trojan which attempts to steal bank credentials and online account credentials. Reading through the articles, it seems that there are many different strains of the malware and my version is not from the big strains. Some strains of the malware are marketed specifically to target Japanese computers while others target Italy and use steganography techniques (the hiding of information in audio or pictures). It seems to be rapidly evolving with multiple offshoots due to a public GitHub repo of its open source code and has apparently been prevalent in the wild since 2014. In general, after the malware’s initial communication with the C&C server, the malware uses a range of attacks to steal cryptocurrency off the machine and steals login and server details from web browsers and email clients. It then performs a man-in-the-browser attack which essentially lets the malicious user gather all data accessed through the web browser like banking credentials and sends it back to the C&C server. While we know don’t know if our malware uses these techniques, as it is part of the same malware family, it is quite possible. However using this research, we can see what was missed during my initial analysis.

[Bad Naming Practice: https://www.gdatasoftware.com/blog/2019/08/35146-taming-the-mess-of-av-detection-names

Malware File: https://www.virustotal.com/gui/file/fe94697f2aaba8807d425ef333e3b2a3d70a59f90fd51650adf7c90516608f95/detection

Japanese Ursnif: https://www.cybereason.com/blog/new-ursnif-variant-targets-japan-packed-with-new-features

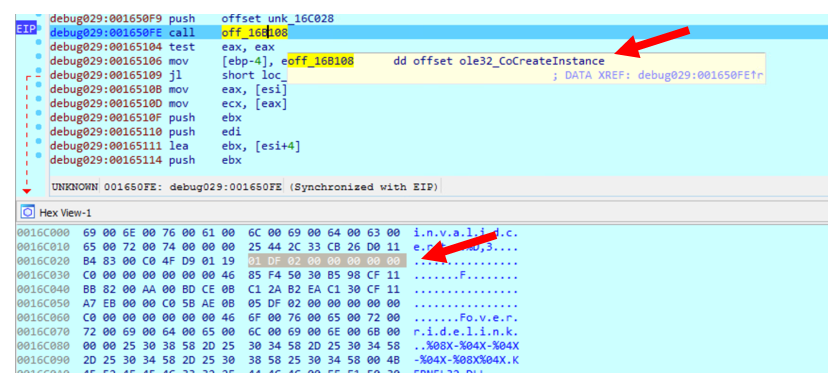
Italian Ursnif: https://www.difesaesicurezza.com/en/cyber-en/the-cybercrime-has-targeted-italy-with-necurs-botnet-and-ursnif-malware/

Italian Steganography Ursnif: https://blog.yoroi.company/research/ursnif-long-live-the-steganography/

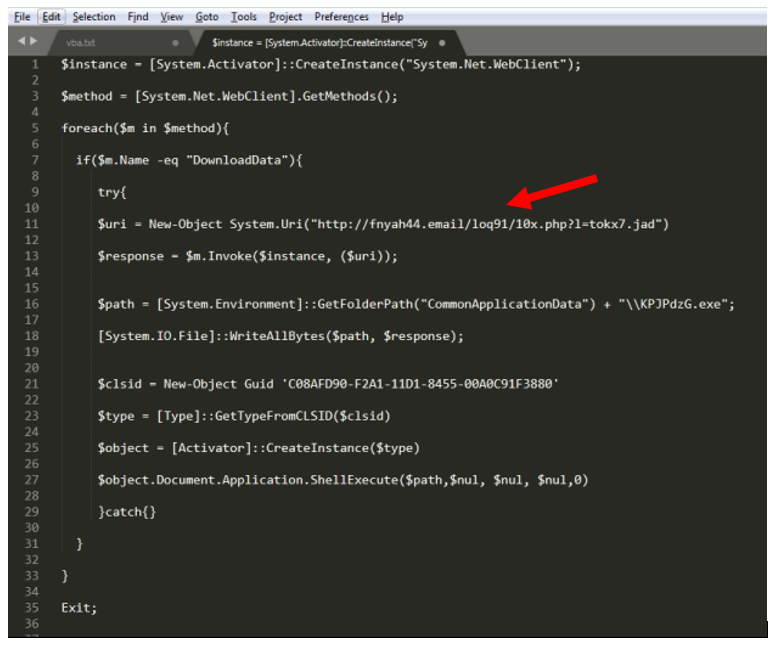
VMRay: https://www.vmray.com/cyber-security-blog/analyzing-ursnif-behavior-malware-sandbox/]

## 4.6 Corrections

While reading over other researchers analysis’s, there were a couple things that I had either misunderstood the behavior of or wasn’t able to explain the reasoning for. One of the thing I couldn’t understand was why iexplorer.exe or our PowerShell command was not running as a child process under 2nd\_formatted.exe. Our PowerShell command was explained above as using WMI calls, but iexplorer.exe also didn’t show up as a child process because it was created as a COM object with a call to CoCreateInstance.

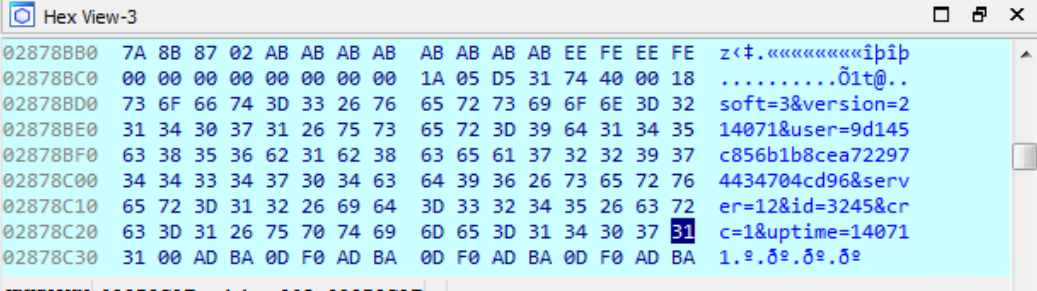
COM stands Computer Object Model, which essentially is a binary model for programs to create environmentally neutral objects that any program can use. A CLSID (class ID) is a GUID that references a COM object while a GUID is a number that identifies resources. When we run our COM object, it isn’t our malware that is running, but an environmentally neutral object that has the same properties as our malware. This is the reason iexplorer.exe doesn’t appear as a child process of our malware.

We can see that in the malware there is a call to CoCreateInstance and a CLSID passed is in, shown in the stack view underneath. While the argument is in hex and little-endian format (basically stored in 8 byte size chunks in reverse order), the value 0002DF01-0000-0000-C000-000000000046 is the CLSID of Internet Explorer. Internet Explorer is then created as a COM object which explains why it was spawned under svchost.exe.



We can see another use of a COM object in our original PowerShell code run by our macro. There is a CLSID referenced in $clsid (line 21) and a call to ShellExecute (line 27). The GUID referenced in line 21 (the string that says C08AF…) resolves to a CLSID that references the object ShellBrowserWindow. This object can execute new processes under explorer.exe and therefore those processes won’t appear as children of our process. In our macro’s case, it would execute the malware under explorer.exe instead of as a child process of Microsoft Word.

Lastly, one of the things I misunderstood was the URL encoding. The URL’s created were not just random strings as I had thought but instead details about the environment being sent back to the malware server. While I had found the strings being encoded in the malware and questioned where they were being used, originally thinking as parameters in the request, I failed to relate it to the URL. I did however think the URL might be the encoded version of something but didn’t pursue it far enough.



It turns out, the malware sends the information above to the C&C server as part of the URL, encoded in Base64 with some characters being replaced with their equivalent hex representation in order to fingerprint our computer.

While I did miss these important areas of the malware, I am quite happy with the overall picture I managed to find. I understand that my lack of experience lead to some of these mistakes, but am glad that there are other researchers who write about these topics and are willing to educate others on them.

[Other Analysis Article: https://www.fortinet.com/blog/threat-research/ursnif-variant-spreading-word-document.html

Bypassing child/parent analysis: https://www.countercept.com/blog/dechaining-macros-and-evading-edr/]

5.0 Take Away

Overall, I am quite happy with the result of my experiment with malware analysis. We were able to successfully extract and understand the macro VBA, download the packer, unpack the malware and understand its capabilities. Furthermore my first experience with “wild” malware resulted in the analysis of a malware family that I haven’t heard of before and was quite interesting to learn about. I also learned a lot about the Windows API, unpacking malware and enjoyed the entire process of learning new things while being uncomfortable with the material. Lastly, I hope you were able to take something away from this article and thanks for reading.