Defense Evasion Techniques

Last updated: November 1st, 2021

Written by: Ariel silver

Analyzing, detecting, and preventing modern day techniques that malwares deploy to bypass security tools and professionals.

Introduction

Today's adversaries and attackers are no longer focused simply on doing the maximum possible damage; they're looking to remain undetected as long as possible. Malware authors go to great lengths to research new or overlooked mechanisms that let them blend with day-to-day machine operations. These methodologies (also known as "defense evasion techniques") seek to help malwares bypass defensive tools' detection.

Surprisingly, most of these techniques don't involve malicious code, but rather use Microsoft distributed files and objects. These strategies, known as "living of the land", use legitimate mechanisms for malicious purposes.

Luckily for us, Cynet 360 monitors evasion technique action points and add a layer of detection mechanisms to its robust knowledge base.

In the scope of this article, we will cover in detail few interesting and powerful evasion techniques that Cynet's CyOps team have faced lately.

This is part of an extensive series of guides about Malware Protection

Disclaimer

In the examples below, you will see several videos of simulated attack scenarios that produce an alert. In some of these scenarios the attack was successfully completed even though an alert was generated.

It's important to note that for these simulations I have disabled Cynet's prevention capabilities and left only detections enabled. You should not attempt this in any shape or form. This should only be done in a lab isolated from the corporate network. If the prevention mechanisms are enabled, none of the scenarios could be successful.

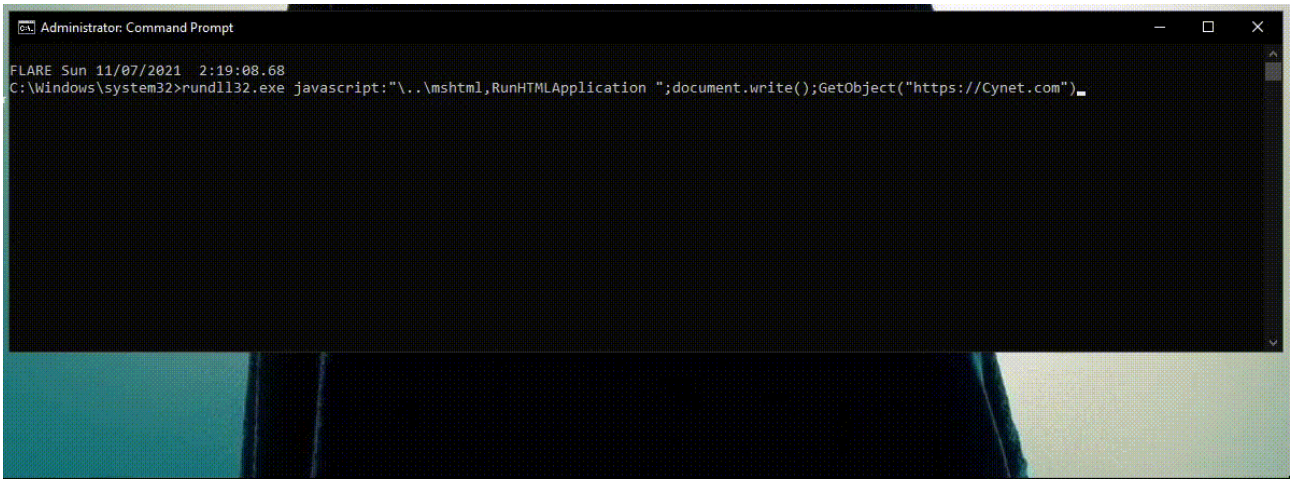
Breaking The Chains

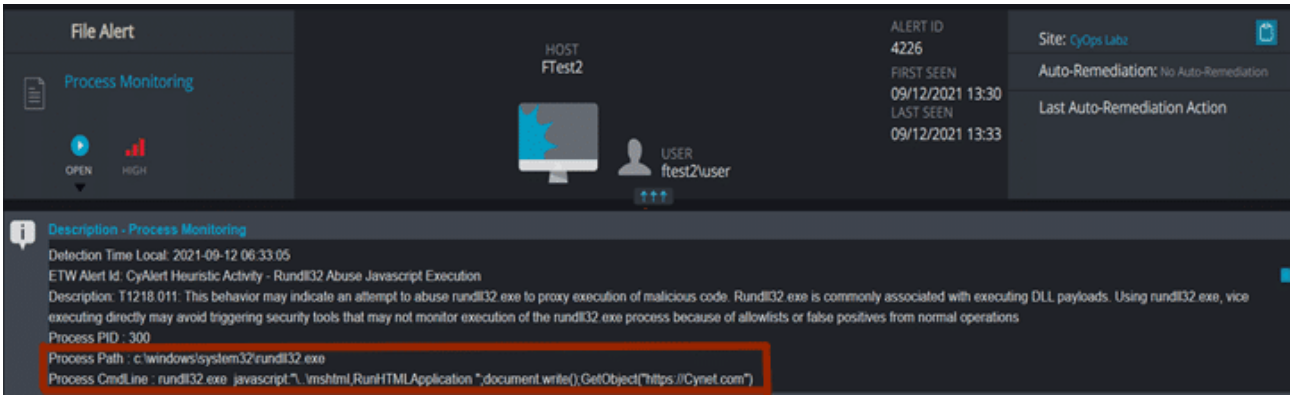
Most next-gen XDR agents continuously examine process actions and process trees to spot any suspicious behavior. Let's look at a common example of a suspicious process tree: Excel document initiates a mshta.exe instance. This behavior indicates malicious behavior 90% of the time and hopefully triggers an alert with any EDR system.

To break these process trees and avoid triggering predefined “child process” rules, attackers will add another legitimate process in the middle to proxy the execution of the child process and break the process chain.

One of the most powerful executables for this job is Rundll32.exe.
Per Microsoft, the original use of Rundll32.exe is that it “Loads and runs 32-bit dynamic-link libraries”, but in the Cyber community this executable has become so much more.
With the use of DLL’s shell32.dll, Rundll32.exe can proxy the execution of almost everything.

1) Rundll32.exe executes JavaScript code.

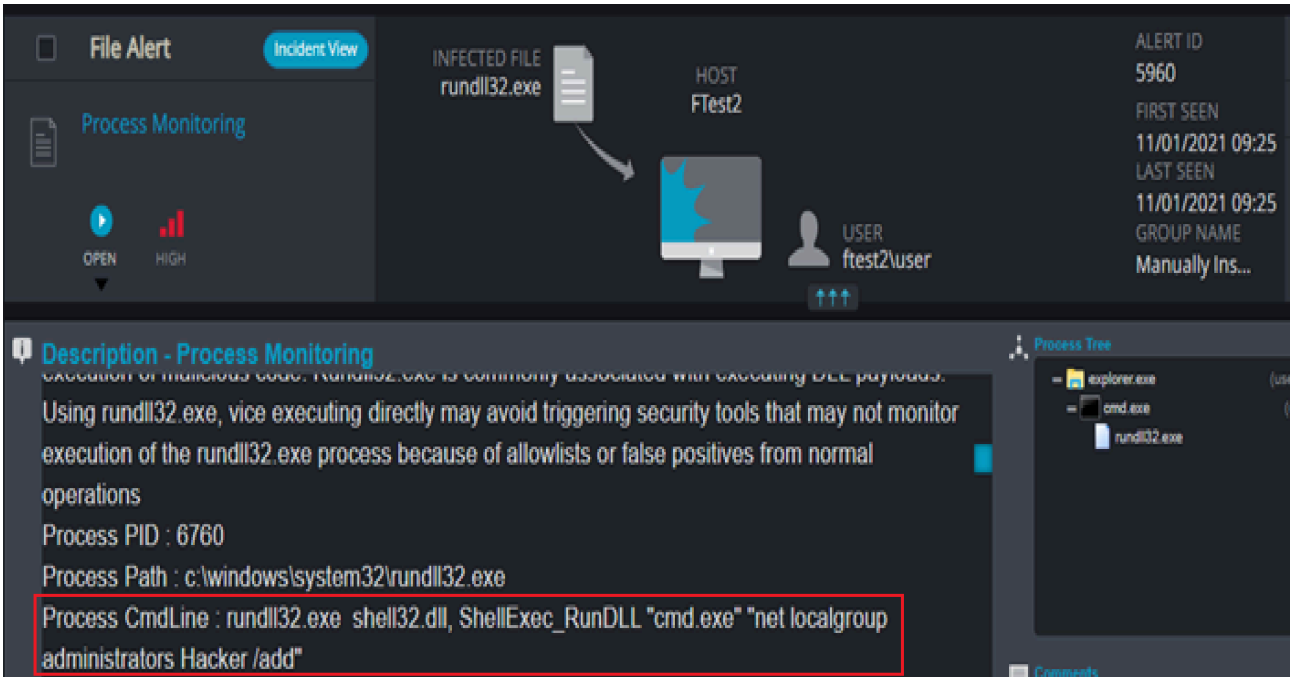




In this scenario, the attacker tried to use Rundll32.exe to proxy the execution of JS code.

This detection is unpassable, as it doesn't analyze the JavaScript code to determine the legitimacy of this behavior, but examines the process behavior.
As indicated before: "Rundll32 Loads and runs 32-bit dynamic-link libraries" so no JavaScript execution should be legitimate.

2) Rundll32.exe proxies shell commands.



In this scenario, the attacker tried to abuse Rundll32.exe to proxy the execution of shell commands using CMD. The use of shelldll32.dll with different arguments is something common with Rundll32.exe.

Once again this detection recognizes the process behavior as malicious and doesn't analyze the shell command.

Dot Com Bubble

The next scenarios relate to COM object hijacking. These are sophisticated techniques that provide the attacker two crucial elements: security tools evasion and persistence.

Admittedly, comparing this to the Dot Com crash might be exaggerating – it did take MSFT shares 17 years to bounce back – but this subject has certainly created massive headaches for Microsoft developers and security teams.

Before diving into the attacks and detections, we must establish a technical baseline:

CLSID – a serial number that represents a unique ID for any application component in Windows
CLSID's location and structure:
You can find CLSID under the following registry keys**:**
HKEY_CURRENT_USER\Software\Classes\CLSID
HKEY_LOCAL_MACHINE\Software\Classes\CLSID
HKEY_CLASSES_ROOT\CLSID\

HKEY_CURRENT_USER + HKEY_LOCAL_MACHINE CLSID's form the HKEY_CLASSES_ROOT which is the one
that the machine actually queries most of the time.
if there is a conflict between the 2 Hives CLSID's, HKEY_CURRENT_USER Overrides HKEY_LOCAL_MACHINE

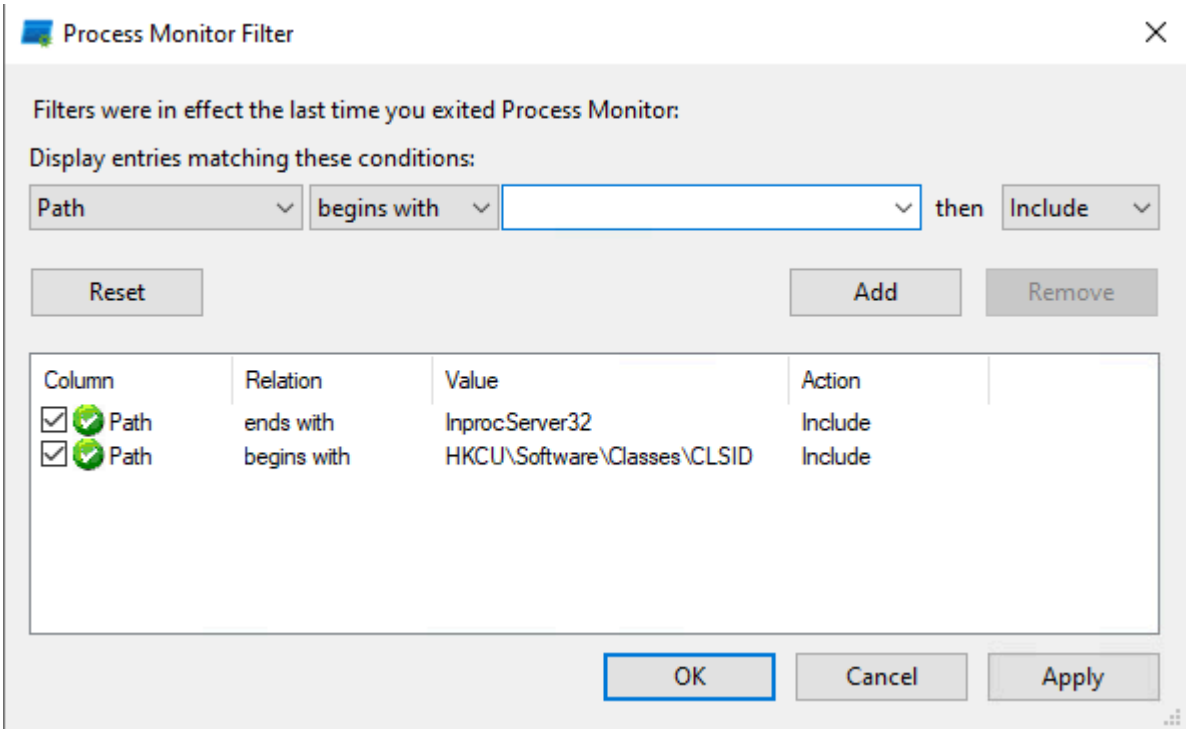
COM objects – components that allow inter-process communication and code reuse

COM object keys – Sub keys of the CLSID's. depends on the COM object key, but most of them point to files or other CLSID's

1) Abusing "TreatAs" subkey

With this technique, we are hijacking a COM object.
We do it by searching for CLSIDs that known softwares call but are unable to find as it doesn't exist. This is when we create this CLSID with any values and data we want, so next time the software calls the CLSID our hijacked COM object will be "executed".

I used Procmon to find a vulnerable COM object like the ones described above which operate everyday applications and see which one will pay dividends.

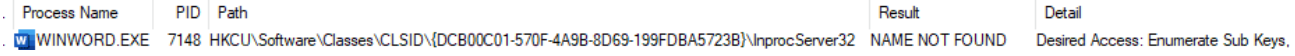


As you see I'm searching for keys under the HKEY_CURRENT_USER hive, that end with "InProcServer32".

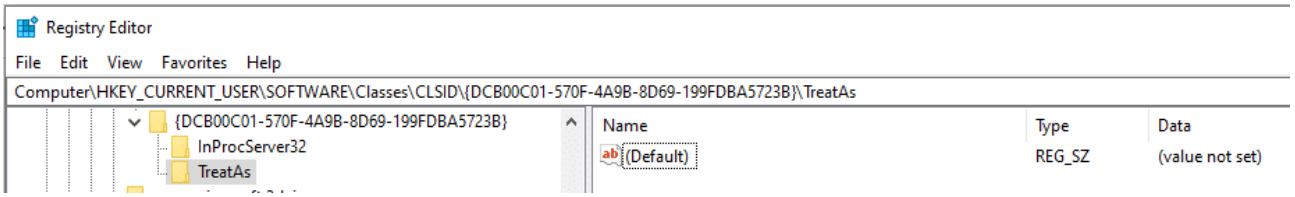
"InProcServer32" – A subkey that indicates where a COM library is located on the disk, defines the threading model, and points to a dll.

Once we finish the initial search, we need to find an instance in which the software tried to access an InProceServer32 subkey of an inexistent CLSID.
Using Procmon we can find it when searching for a result that equals "NAME NOT FOUND".

The COM object I found was referred to by WinWord.exe but didn't exist. Once again, a perfect candidate for a hijack.

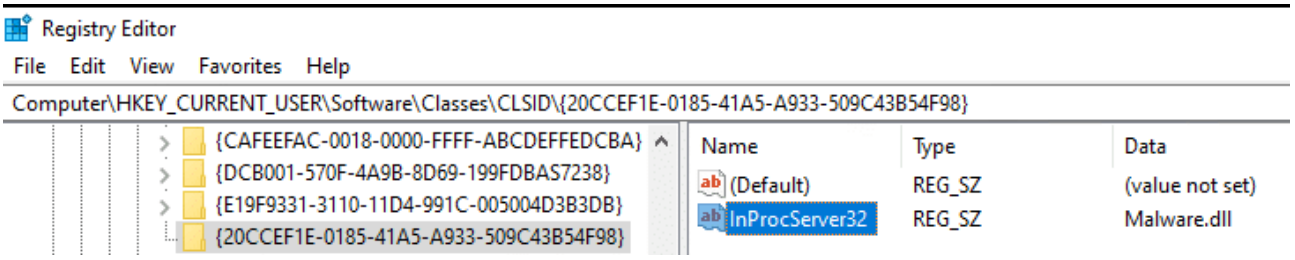


I used the Registry editor to create this registry key and under it created two subkeys: "InProcServer32" and "TreatAs"

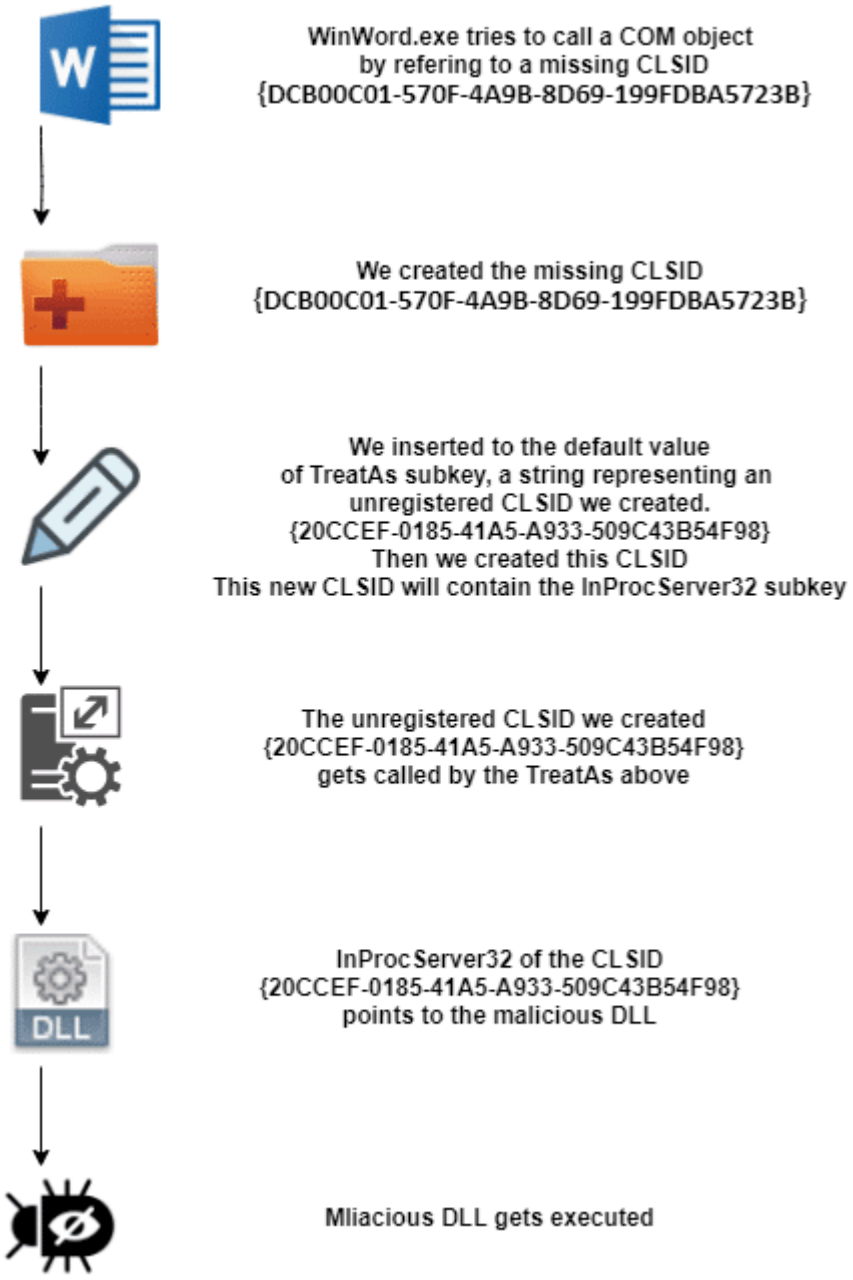


"TreatAs" – Subkey of a CLSID. "TreatAs" contains a value named "(Default)" which can contain a string of another CLSID number and spawn it as well when its own CLSID is called.
Usually, the additional CLSID that is called by "TreatAs" will have a "InProcServer32" subkey that points to a

malicious DLL. For example:

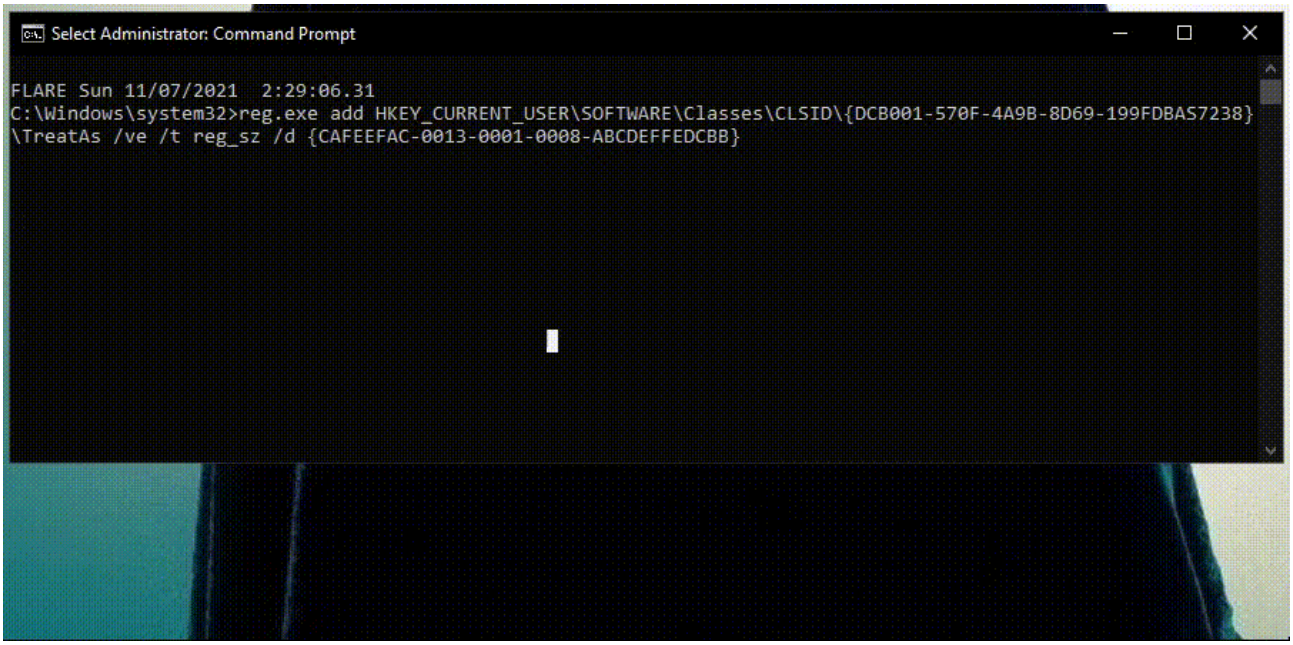


Attackers love this mechanism because it lets them create the following flow:

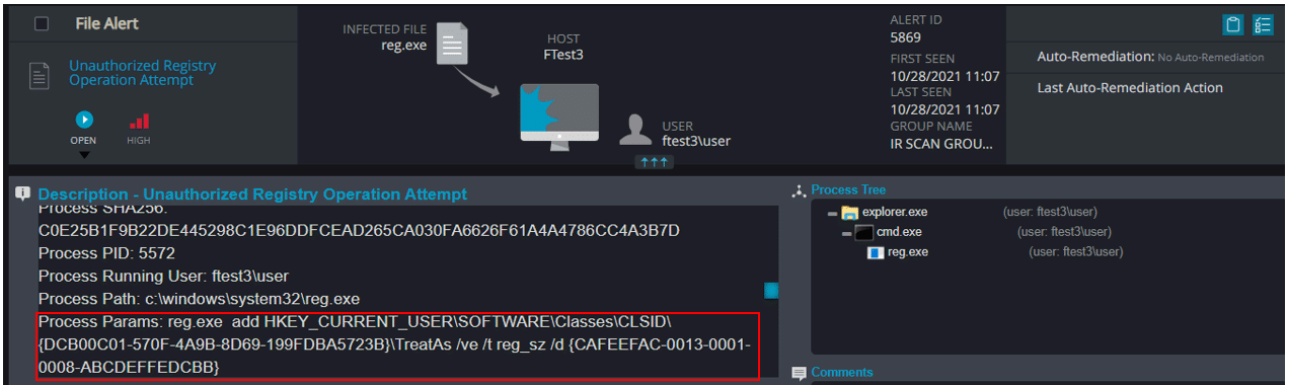


Cynet goes one step further to prevent this attack.

Instead of monitoring all the “TreatAs” subkeys and waiting for them to call an unregistered CLSID, Cynet just detects and prevents the adversary from modifying the data of “(Default)” value under the “TreatAs” subkey.



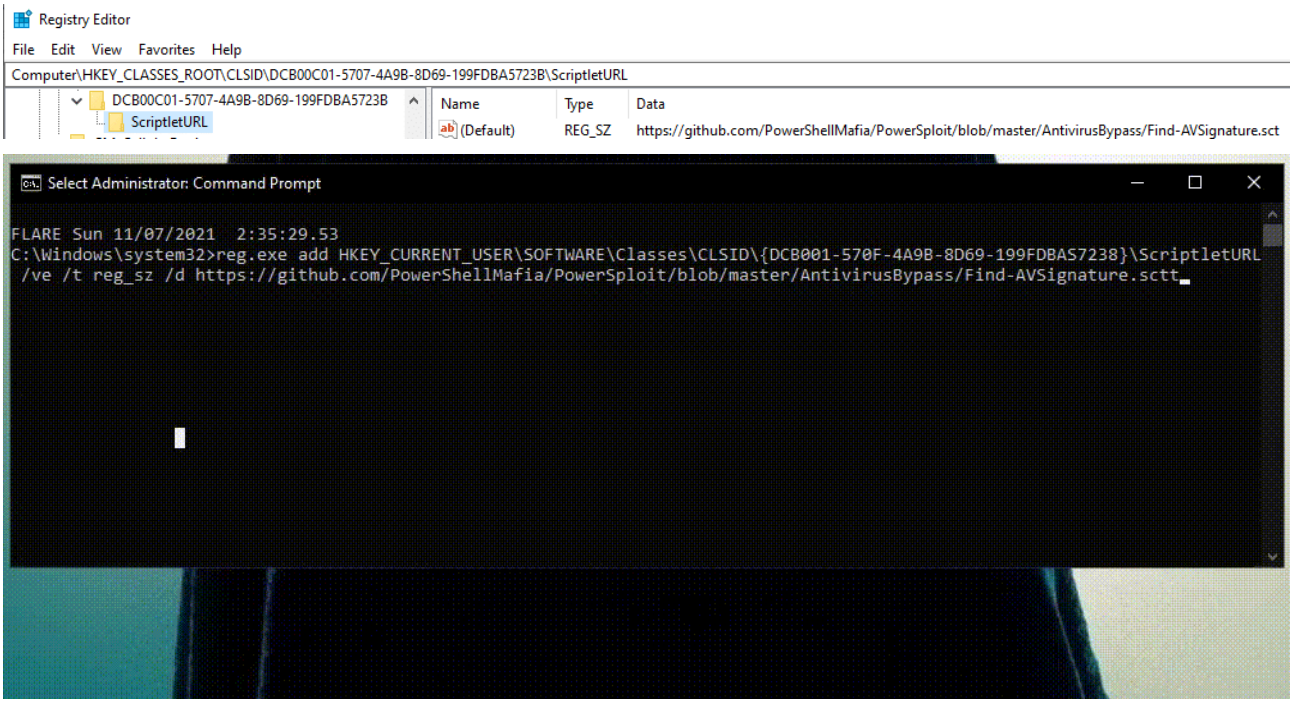
As shown in the video and the alert below, Cynet immediately detects the attempt to modify a “TreatAs” subkey of a registered CLSID.



2) Abusing “ScriptletURL” subkey

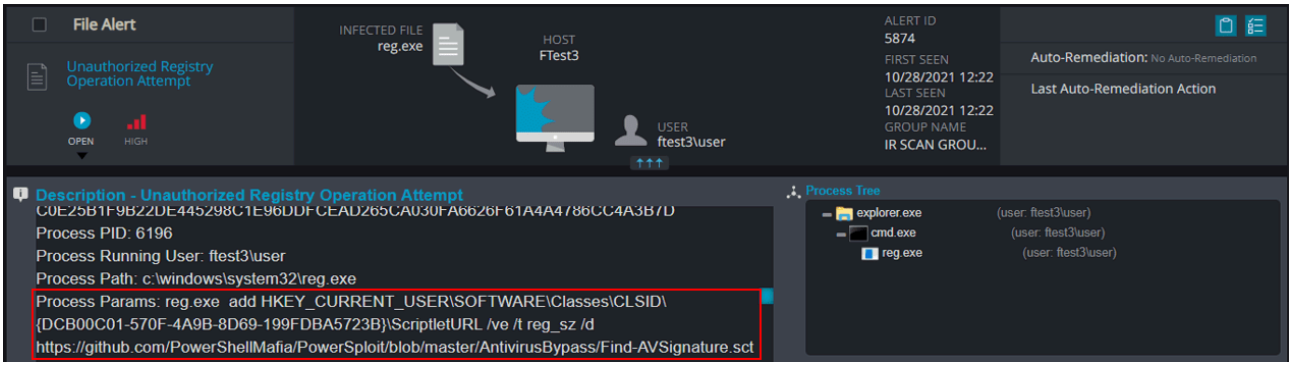
If you thought “TreatAs” is dangerous, the next sub-CLSID will look almost too powerful to be true.
The steps to find and hijack the missing CLSID are identical to the “TreatAs” flow.
For that reason, I’ve decided to further abuse the same CLSID as before: {DCB00C01-570F-4A9B-8D69-199FDBA5723B}. This time we are adding the “ScriptletURL” sub key.

This COM object allows software to download scripts from a remote URL and execute them without the script file ever touching the disk. Similar to Regsvr32.exe (which you will see in the following technique), “ScriptletURL” uses Scrobj.dll for these actions.



As shown in the video above, this time I attempted to complete the scenario using the CMD.
This was done to assure our customers that these types of operations trigger an alert, regardless of the method used.

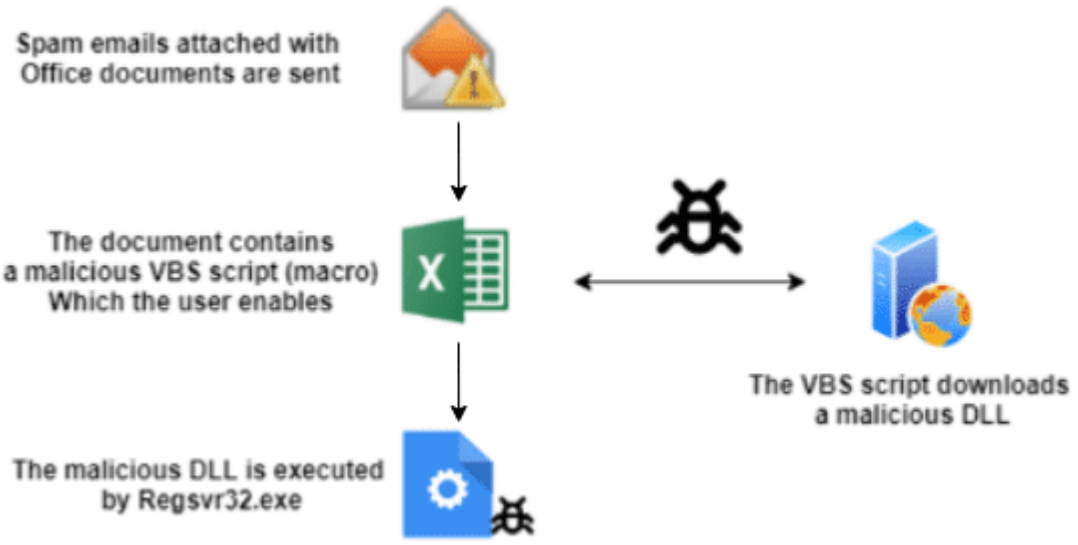
Once again Cynet detects and prevents the attempt to modify the registered CLSID's "ScriptletURL" sub key.
In this case the operation was detected and not prevented. Therefore, the next time WinWord.exe calls this COM object, it will download a PowerSploit script from GitHub and execute it.



Qakbot's BFF

The CyOps team has encountered Qakbot multiple times in recent months. Qakbot malware started as a banking trojan in 2007. Its goal was to retrieve banking credentials and financial information. In later stages, it can be used for a much wider range of activities, including spreading a ransomware attack.

Qakbot's attack flow first stages:

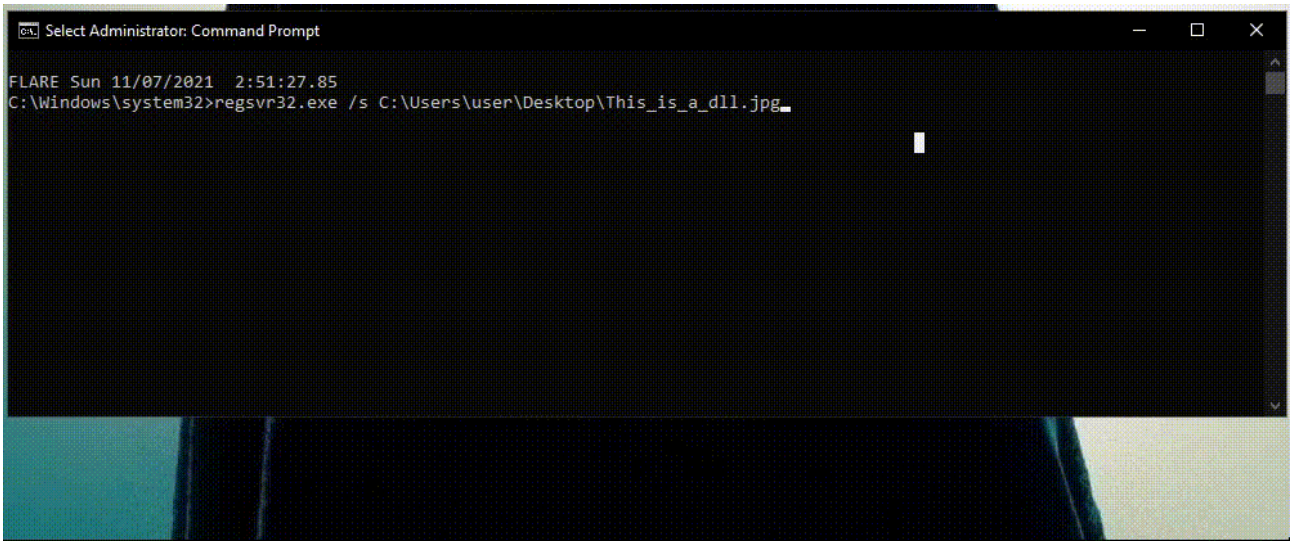


The important thing to notice in the above methodology is the use of Regsvr32.exe. Regsvr32.exe is a well-known and widely used system executable that has some documented functionalities regarding DLLs. Its documented capabilities essentially add up to initiating DLLs, or in Microsoft's words, it "registers .dll files as command components in the registry."

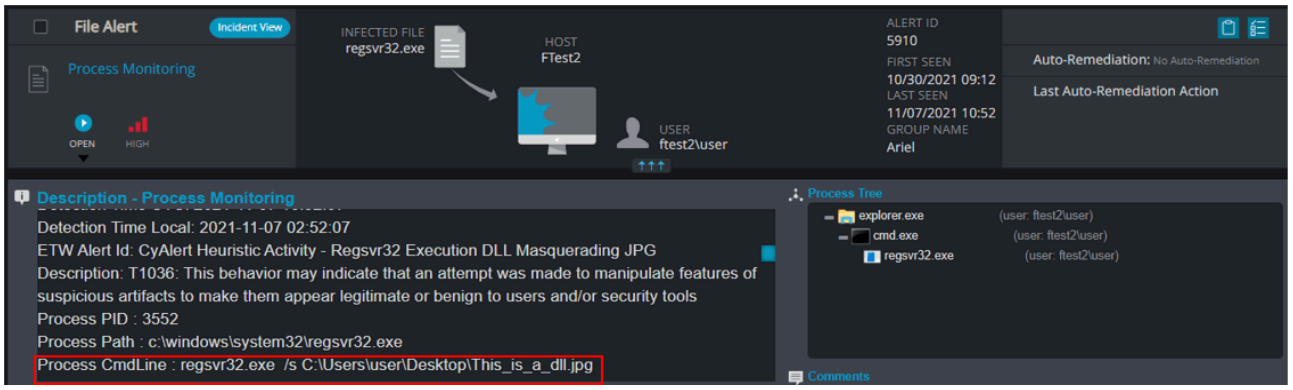
So how can we monitor and prevent the use of Regsvr32.exe without causing BSOD to our machine? First, we don't prevent any "clean" use of Regsvr32 such as executing a proper DLL file. Even so, we can monitor and prevent the following Regsvr32.exe abusing techniques:

1) Masquerading DLLs

One commonly used technique is to disguise malicious DLLs as different file types with various extensions. For example, an attacker can download a malicious DLL, change its extension to .jpg and use Regsvr32.exe to execute it as a DLL. Not many security tools monitor calls to a picture file.



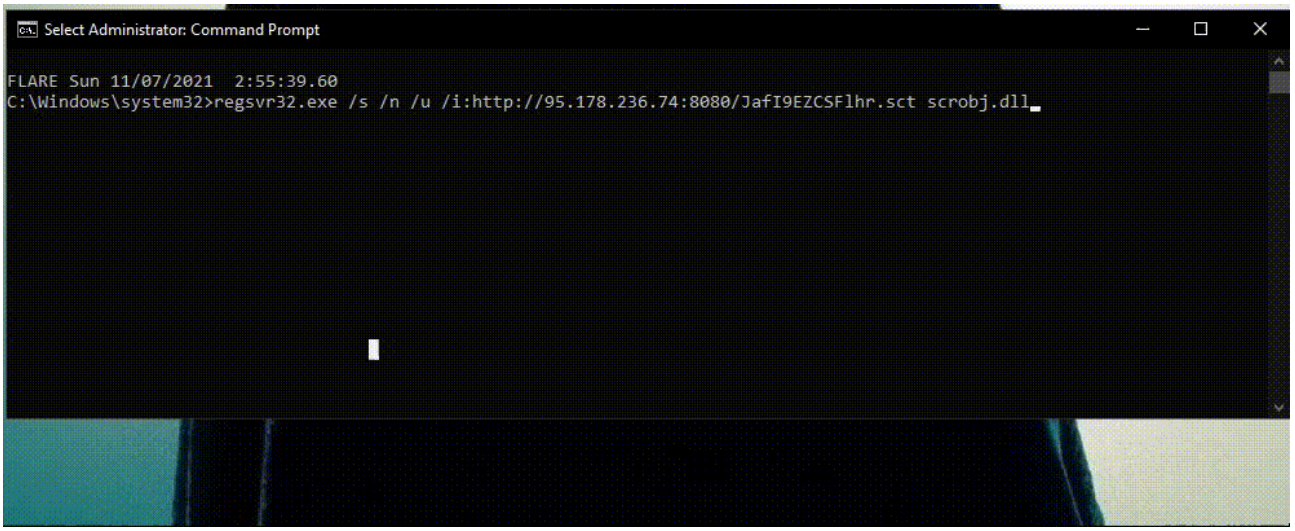
In above detection, Cynet looks at each and every instance of regsvr32.exe and the extension of every file being executed by regsvr32.exe.
If the file extension isn't .dll , Cynet understands that it's looking at a masquerading technique and prevents it.



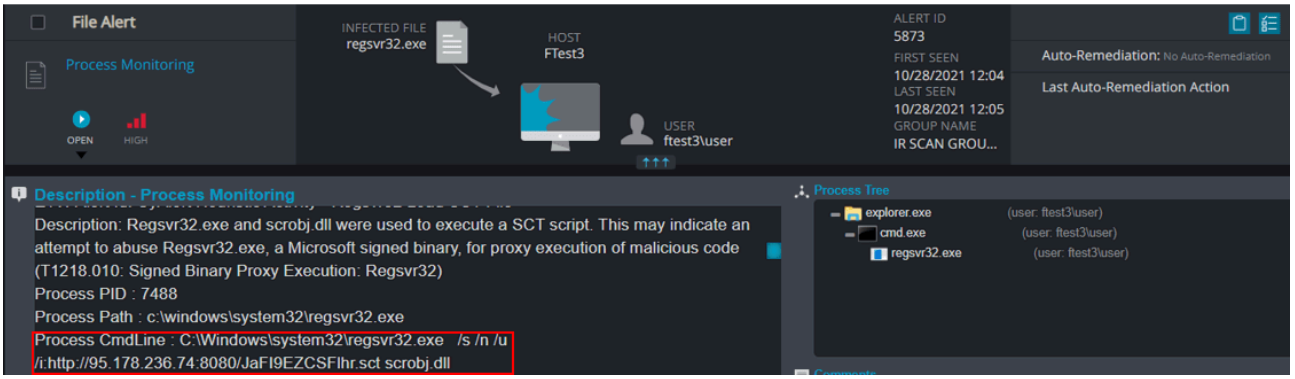
2) **Squiblydoo**

Squiblydoo is the term name for methods using Scrobj.dll for LOLBin attacks. Scrobj.dll can load a COM scriptlet directly from the internet and execute it.

As promised, in the ScriptURL explanation paragraph, I will break down another overpowered evasion technique to conduct fileless attacks using Scrobj.dll.



The command above attempts to abuse Regsvr32.exe to proxy the execution of a remote scriptlet.
The Squiblydoo methods initiate Fileless attacks which receive malicious strings on HTTP and HTTPS and execute them right on the RAM without ever touching the disk.
Cynet will alert and prevent this type of malicious behavior



Sorry… I don't speak base64

This method is used to bypass the not-so-good and very obsolete antivirus agents. To understand how it works, it's critical to first understand how AV categorizes files as malicious. AVs store a massive database of signatures that are searched against files. If a signature is matched, the file is labelled malicious. These signatures are built from combinations of strings and hexes that were found in malwares but are unique enough to not generate many false positives. To provide an example we're looking at Yara rules. These operate on the same concept as AV platforms, but shouldn't be mistaken for AV.

First I dumped them in a hex editor to search for any unique strings.

```
000162f0 86 70 8d 44 24 10 50 8d 4c 24 28 51 8d 54 24 30 .p.D$.P.L$(Q.T$0
00016300 52 8d 46 3c 50 e8 86 96 f1 ff 85 c0 8b 44 24 14 R.F<P........D$.
00016310 75 11 c7 44 24 10 ff ff ff 68 75 74 61 78 6f 67 u..D$....hutaxog
00016320 69 79 75 6b 69 68 6f 68 00 3b 44 24 2c 89 44 24 iyukihoh.;D$,.D$
00016330 14 0f 8c 6c ff ff ff 2b 7e 30 8d 46 3c c1 ff 02 ...l...+~0.F<...
00016340 50 89 7e 34 e8 b7 90 f1 ff 8b 44 24 20 2b 44 24 P.~4......D$ +D$
00016350 18 2b 6c 24 1c 89 46 64 8b 46 34 85 c0 8b dd 74 .+l$..Fd.F4....t
00016360 11 85 db 7d 02 33 db 83 c0 ff 3b d8 7e 07 8b d8 ...}.3....;.~...
00016370 58 61 70 75 7a 6f 6a 20 72 6f 76 6f 74 00 00 00 Xapuzoj rovot...
00016380 83 ff ff 74 3a 8b 46 08 8d 54 24 30 52 50 ff 15 ...t:.F..T$0RP..
00016390 f0 03 4b 00 8b 6e 54 e8 74 d6 fe ff 8b 56 08 2b ..K..nT.t....V.+
000163a0 f8 0f af fd 03 7e 5c 6a 00 8d 4c 24 34 51 03 ef .....~\j..L$4Q..
000163b0 52 89 7c 24 40 89 6c 24 48 ff 15 a0 03 4b 00 83 R.|$@.l$H....K..
000163c0 fb ff 89 5e 60 0f 84 7d 00 00 00 8b 4e 08 8b 2d ...^`..}....N..-
000163d0 f0 03 4b 00 8d 44 24 30 50 51 ff d5 8b 7e 54 e8 ..K..D$0PQ...~T.
000163e0 2c d6 fe ff 8b 4e 08 8b d0 8b c3 2b c2 0f af c7 ,.....N.....+....
000163f0 03 46 5c 6a 00 03 f8 89 44 24 38 8d 44 24 34 50 .F\j....D$8.D$4P
00016400 51 89 7c 24 48 ff 15 a0 03 4b 00 e8 00 d6 fe ff Q.|$H....K......
00016410 8b f8 3b df 7c 2a 8b 46 08 8d 54 24 30 52 50 ff ..;.|*.F..T$0RP.
00016420 d5 8b 44 24 3c 2b 46 5c 2b 44 24 34 99 f7 7e 54 ..D$<+F\+D$4..~T
00016430 83 e8 01 79 02 33 c0 8d 0c 38 3b d9 7e 0a 2b d8 ...y.3...8;.~.+.
00016440 53 8b c6 e8 08 e1 ff ff 83 7e 34 00 75 07 8b c6 S........~4.u...
00016450 e8 56 6f 67 69 64 75 6b 61 20 70 61 77 61 76 61 .Vogiduka pawava
00016460 00 00 8b 1d 28 03 4b 00 8b f8 74 0e 6a 00 68 d8 ....(.K...t.j.h.
00016470 27 00 00 57 ff d3 6a 00 eb 0c 6a 03 68 d8 27 00 '..W..j...j.h.'.
00016480 00 57 ff d3 6a 03 68 dc 27 00 00 57 ff d3 e8 9d .W..j.h.'..W....
00016490 dd ff ff 8b c6 e8 76 f0 ff ff 5f 5d c7 46 10 01 ......v... ].F..
```

Then I extracted the unique strings and created the following Yara rule:

```
rule test_1 {
    meta:
        author = "Ariel Silver"
    strings:
        $s1 = "hutaxogiyukihoh$" nocase
        $s2 = "Xapuzoj rovot" nocase
        $s3 = "Vogiduka pawava" nocase
    condition:
        (uint16be(0) == 0x4D5A and 1 of them) or 3 of them
}
```

This is a simple Yara rule created after analyzing a Trojan.

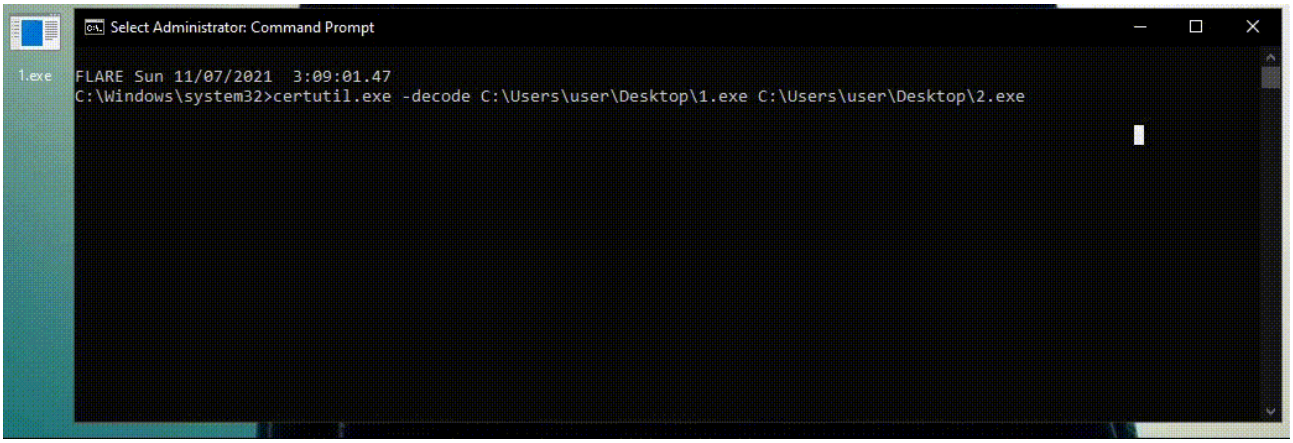
We found three unique strings and created a Yara signature that looks for executables containing one of these strings or a random file containing all three of the strings, Case-insensitive. Once again, this is not an AV signature, but just an example of the concept.

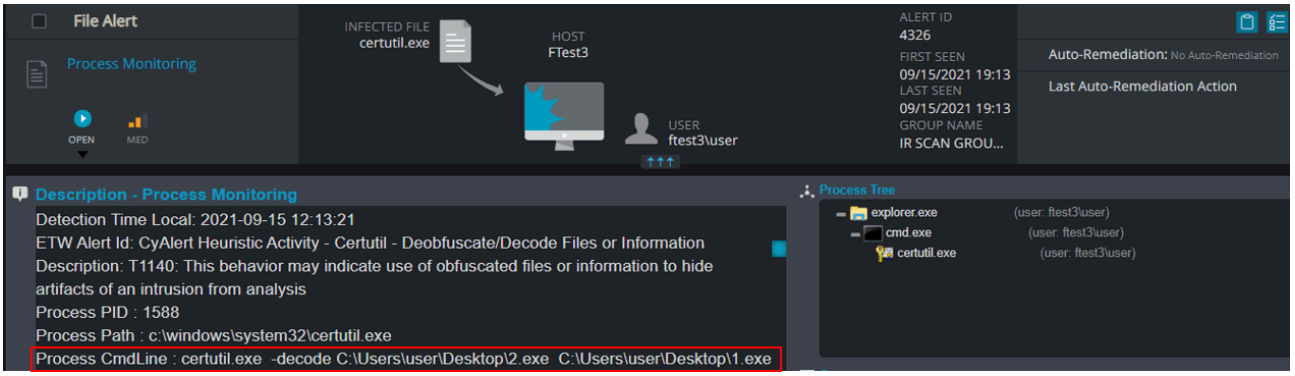
1) Certutil – decode

Let’s run a scenario. After compromising the system with a malicious macro, the attacker wants to transfer additional tools into the system. The adversary knows the company has a Firewall and IDS with a built-in AV, which means they need to bypass it. The attacker can convert their malware to base 64, and once inside the system convert it back into normal and execute it.

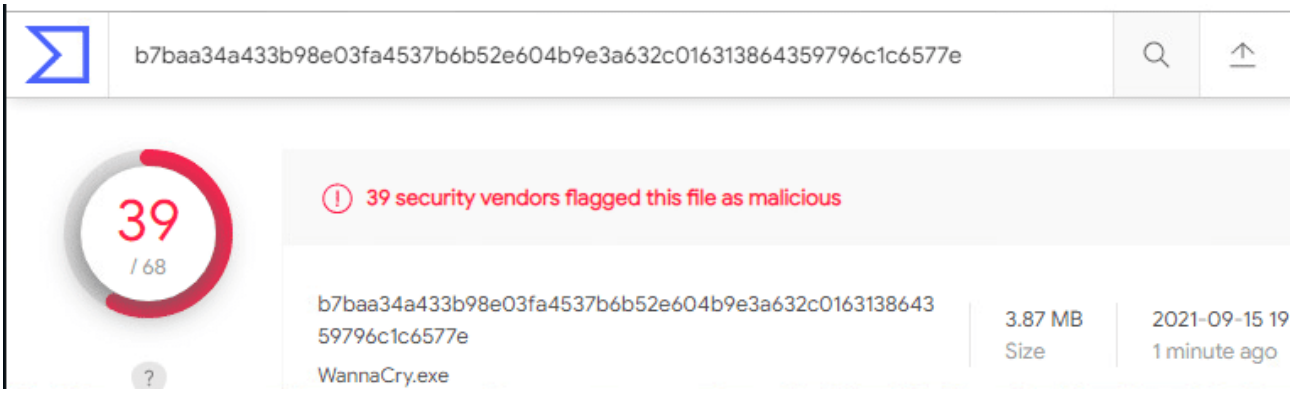
The malware can be decoded by a well-known system utility called Certutil. Certutil is a LOLBin attackers constantly use as it’s signed and has many capabilities.
Certutil -decode, is a way to decode a base64 file back to its functional form.

Luckily, Cynet monitors Certutil.exe and alerts on this behavior and many more.

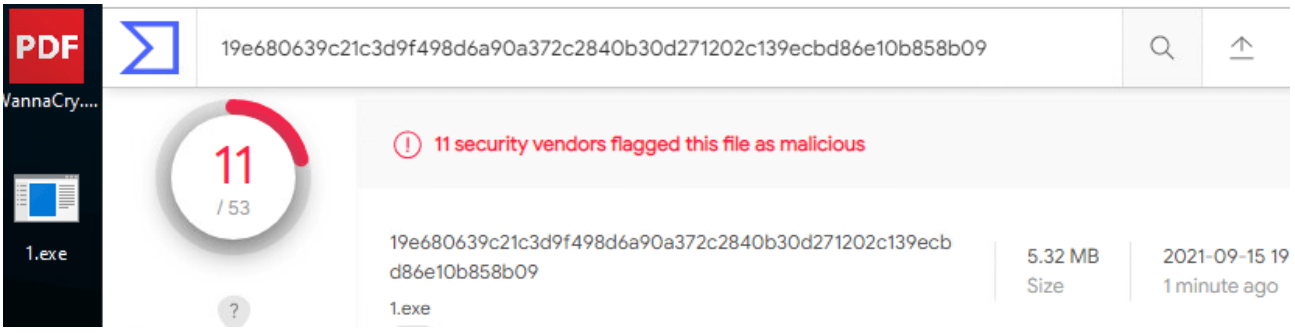




Just to prove that Antiviruses can't handle an easy bypass technique like this, I took a well-known ransomware variant called "WannaCry" and uploaded it twice to VirusTotal.
First is pre-encoding:



Second is post-encoding:



More than 66% of the Antiviruses couldn't keep up with such a simple technique.

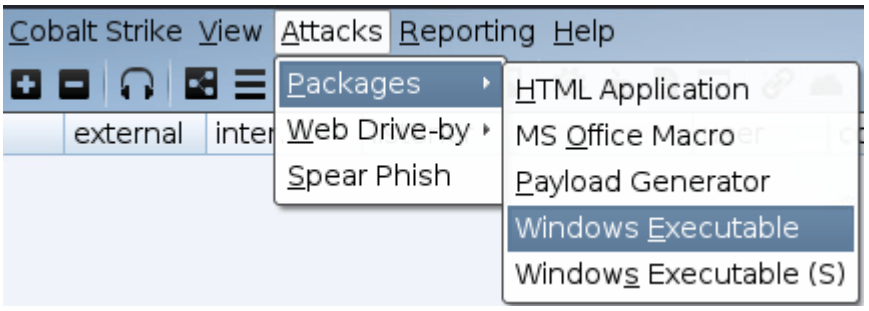
COVID Vaccine

Injection is one of the scariest phrases for any security analyst. Analyzing process injections requires ignoring the process tree and the vast majority of security tools to focus instead on process handled and API functions.

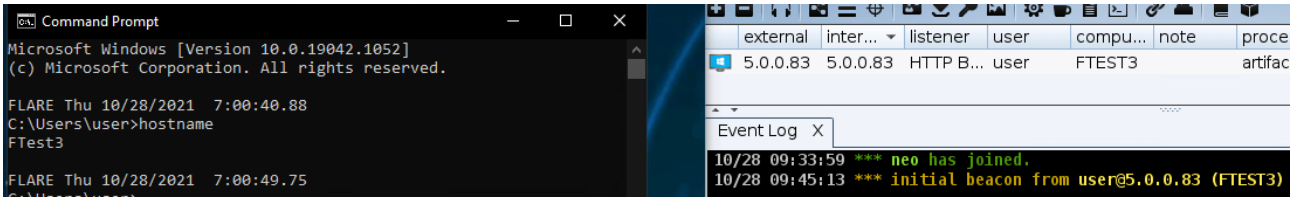
Let's take a look at Cobalt Strike. This malware started off as a red-team tool but has become a go-to platform for many attackers. Cobalt Strike comes with a user-friendly GUI that allows adversaries to understand, document, and further damage machines they've exploited. Additionally, it comes pre-loaded with several post-exploitation modules that can be initiated with a simple command.

Finally, there are attackers that use Cobalt Strike modules throughout the attack chain – from vulnerability scans to malware delivery or data exfiltration. Even though Cobalt Strike has endless attacking capabilities, the most commonly noted is its capability for process injection. I have simulated a Cobalt attack ending with an injection in our labs, to see what API functions were used by the malicious file.

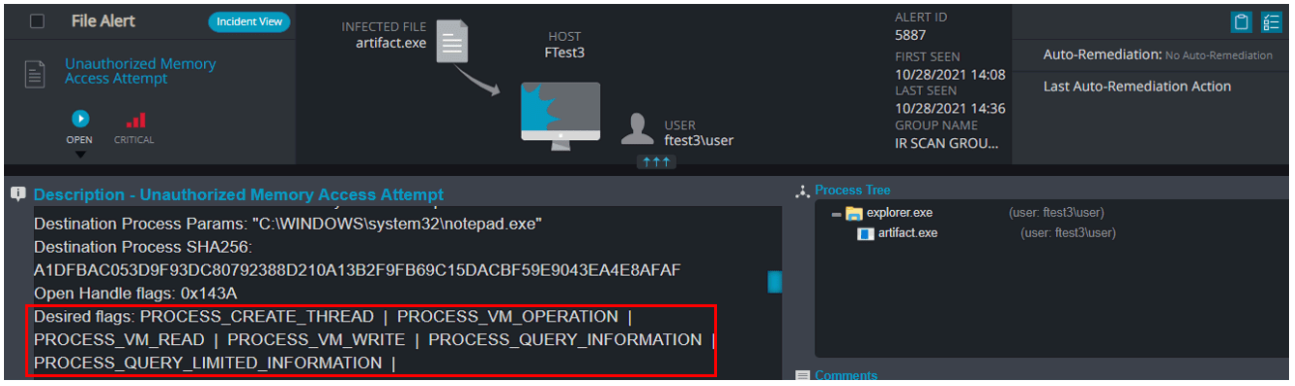
First we created a .exe file called “artifact.exe”, that we will plant on the target machine:



When executed by the target, the malicious file will create a reverse shell between the target and the C2:



Lastly used the cobalt injection command while specifying the PID of notepad.exe.
This generated the following alert on the target machine:



By looking at the requested flags, we see that the artifact.exe tried to create a remote thread on notepad.exe and write to its memory in order to execute malicious code on its behalf.
Cynet constantly monitors the API used between processes to find any anomalies that don’t align with the process permissions.

Conclusion

The evidence I’ve laid out in this article is only the tip of the iceberg when it comes to security tool evasion techniques. This is an endless game of cat and mouse, with both sides constantly evolving. We’ve broken down some common adversary techniques to remain undetected, and how Cynet can detect them. The Cynet research team is continuously searching for, analyzing, and building mechanisms to detect these methods and prevent them.

Additionally, the CyOps team is constantly learning about new techniques and things to look out for when monitoring our customers' networks, whether its analyzing suspicious files or dealing with incident response.
