unit42.paloaltonetworks.com /unusual-malware/

Off the Beaten Path: Recent Unusual Malware

Dominik Reichel :: 3/14/2025

Executive Summary

Recently, we discovered several new malware samples with unique characteristics that made attribution and function determination challenging. While many threat actors will strictly use tools released by the offensive security community, we also encounter novel, custom-built malware – sometimes with new tricks and techniques. This article describes three particularly unusual malware examples we came across last year.

- The first malware sample is a passive Internet Information Services (IIS) backdoor developed in C++/CLI, a programming language very rarely used by malware authors.
- The second sample is a bootkit that uses an unsecured kernel driver to install a GRUB 2 bootloader for a rather unusual purpose.
- The third sample is a Windows implant of a cross-platform post-exploitation framework developed in C++.

Although the last example is a red team tool that doesn't use any novel methods, we believe it is worth reviewing due to significant deviation from other post-exploitation frameworks we've seen during the past year.

Palo Alto Networks customers are better protected from these malware samples through Advanced WildFire, with its different memory analysis features.

Cortex XDR and XSIAM are designed to prevent the execution of known malicious malware, and also prevent the execution of unknown malware using Behavioral Threat Protection and machine learning based on the Local Analysis module.

If you think you might have been compromised or have an urgent matter, contact the Unit 42 Incident Response team.

Related Unit 42 Topics **Malware**, Backdoor

Example 1: C++/CLI IIS Backdoor

The C++/CLI programming language is an extension of the C++ programming language that can be used to write mixed-mode .NET applications. These mixed assemblies can have managed code (C#) and unmanaged code (C++). Analyzing these files is challenging because it can be hard to read their interoperation code in existing .NET decompilers.

This programming language is very rare among malware authors, likely because C++/CLI is poorly documented compared to other languages. One of the first public mentions of a malware sample coded in C++/CLI was a module of a toolkit that Positive Technologies described in 2018. However, this module and all other C++/CLI malware we have come across so far is not as complex as this particular IIS backdoor.

We found two versions of this passive IIS malware uploaded to VirusTotal, both submitted from Thailand. The later version, compiled on May 9, 2023, differs from the earlier one, compiled on April 28, 2023, in its approach to handling external commands. It uses a custom cmd.exe wrapper tool, as opposed to the earlier version which uses just the cmd.exe tool. This change was likely implemented to create less monitorable activity, as spawning cmd.exe directly from an IIS process could raise suspicion. These samples have been referred to as “Detele” [PDF] during a presentation by John Southworth at the LABScon conference in 2024.

Technical Analysis of the C++/CLI IIS Backdoor

The two samples of this passive IIS malware were compiled with different Visual C++/CLI compiler versions and also differ slightly in functionality.

- The newer and bigger assembly (SHA256 hash: 15db49717a9e9c1e26f5b1745870b028e0133d430ec14d52884cec28ccd3c8ab) is internally named proxyxml_v4 (described as version 2). This newer sample uses more AMSI/ETW patching and has a different implementation for the non-self-contained command-line features.
- The slightly older one (SHA256 hash: 8571a354b5cdd9ec3735b84fa207e72c7aea1ab82ea2e4ffea1373335b3e88f4) is named IISShellModule (described as version 1).

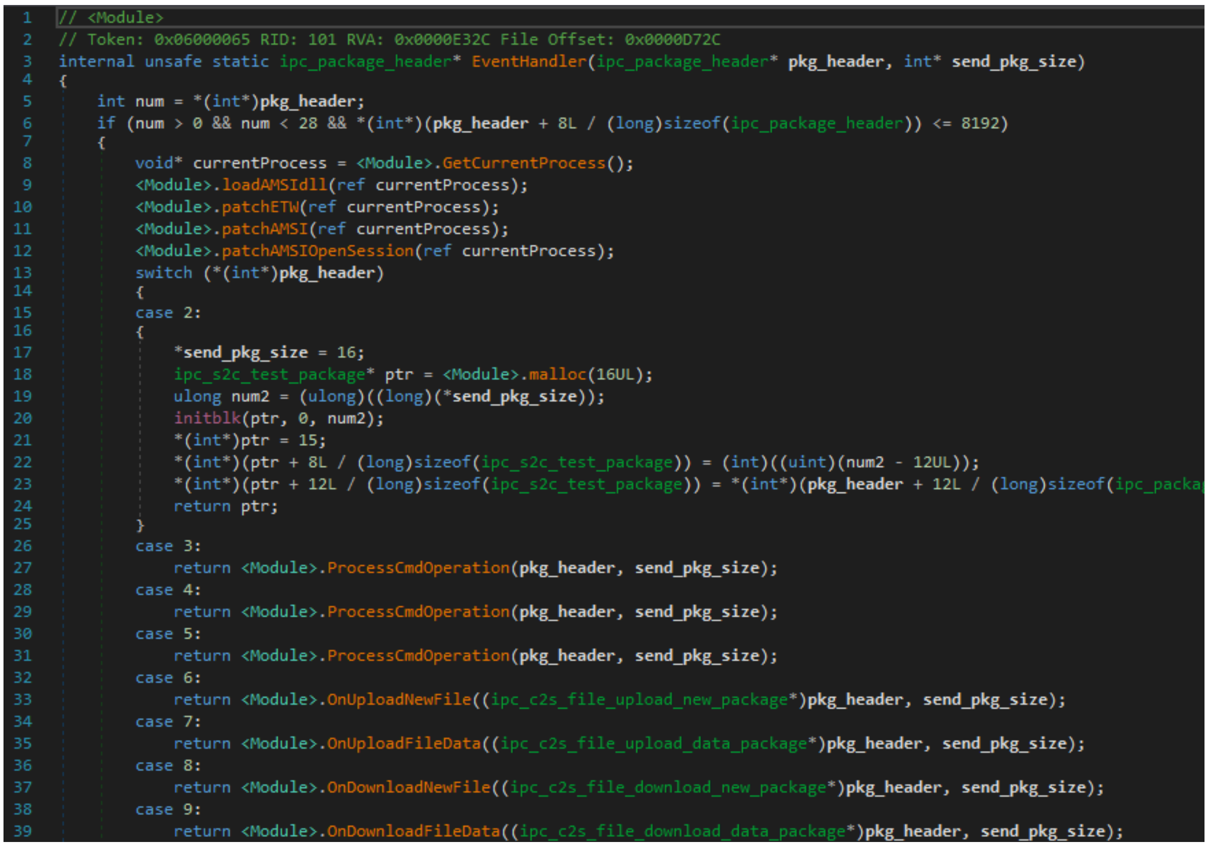
The author created the backdoor as an IIS module that uses the exported function RegisterModule to register itself for RQ_SEND_RESPONSE event notifications. Therefore, whenever the IIS server sends an HTTP response, it will call the backdoor’s registered OnSendResponse method. For callback traffic, the backdoor’s OnSendResponse method filters on the incoming HTTP request having the following attributes before calling its event handler:

- Request type: HTTP POST
- Request header field and value: X-ZA-Product : AbJc123!@#45!!
- Request header field and value: X-ZA-Platform : <any>

The custom HTTP request header field named X-ZA-Product is internally reassigned to PWD_HEADER, and its value AbJc123!@#45!! is reassigned as PWD_VALUE. This PWD_VALUE is encrypted using AES with a key of AQJBdmin!@#45!@## (internally called KEY) and the result is Base64-encoded.

The other HTTP request header field named X-ZA-Platform is processed by the malware as CMD_HEADER, and the CMD_VALUE represents the actual command data. This CMD_VALUE is also encrypted using AES with the same KEY as the PWD_VALUE and the result is also Base64-encoded.

The backdoor has an event handler that processes the data from X-ZA-Platform to parse the included commands. Figure 1 shows the event handler code that processes the implemented commands.



```
// <Module>
// Token: 0x06000065 RID: 101 RVA: 0x0000E32C File Offset: 0x0000D72C
internal unsafe static ipc_package_header* EventHandler(ipc_package_header* pkg_header, int* send_pkg_size)
{
    int num = *(int*)pkg_header;
    if (num > 0 && num < 28 && *(int*)(pkg_header + 8L / (long)sizeof(ipc_package_header)) <= 8192)
    {
        void* currentProcess = <Module>.GetCurrentProcess();
        <Module>.loadAMSIdll(ref currentProcess);
        <Module>.patchETW(ref currentProcess);
        <Module>.patchAMSI(ref currentProcess);
        <Module>.patchAMSIOpenSession(ref currentProcess);
        switch (*(int*)pkg_header)
        {
        case 2:
        {
            *send_pkg_size = 16;
            ipc_s2c_test_package* ptr = <Module>.malloc(16UL);
            ulong num2 = (ulong)((long)(*send_pkg_size));
            initblk(ptr, 0, num2);
            *(int*)ptr = 15;
            *(int*)(ptr + 8L / (long)sizeof(ipc_s2c_test_package)) = (int)((uint)(num2 - 12UL));
            *(int*)(ptr + 12L / (long)sizeof(ipc_s2c_test_package)) = *(int*)(pkg_header + 12L / (long)sizeof(ipc_packa
            return ptr;
        }
        case 3:
            return <Module>.ProcessCmdOperation(pkg_header, send_pkg_size);
        case 4:
            return <Module>.ProcessCmdOperation(pkg_header, send_pkg_size);
        case 5:
            return <Module>.ProcessCmdOperation(pkg_header, send_pkg_size);
        case 6:
            return <Module>.OnUploadNewFile((ipc_c2s_file_upload_new_package*)pkg_header, send_pkg_size);
        case 7:
            return <Module>.OnUploadFileData((ipc_c2s_file_upload_data_package*)pkg_header, send_pkg_size);
        case 8:
            return <Module>.OnDownloadNewFile((ipc_c2s_file_download_new_package*)pkg_header, send_pkg_size);
        case 9:
            return <Module>.OnDownloadFileData((ipc_c2s_file_download_data_package*)pkg_header, send_pkg_size);
```

Figure 1. IIS backdoor event handler as shown by dnSpyEx.

At first, the handler patches AMSI and ETW routines for the current process (copied and pasted from GitHub). Afterwards, the handler utilizes the X-ZA-Platform command data to extract the specific commands and corresponding data for each implemented command feature.

Table 1 shows the list of available commands in version 2 of this malware.

Command	Internal Term	Description
2	-	Reply with a test HTTP request.
3 / 4 / 5	ProcessCmdOperation	Write the embedded cmd.exe wrapper application (internally termed BackendIPCServer) to %PUBLIC%\VC_REDIST_CONFIG_X64.TXT and create a process for it. Redirect any command-line commands from the C2 server to this wrapper app via a named pipe \.\pipe\pipename_isudbvvws and also return the result via the pipe.
6	OnUploadNewFile	Create an empty file with a given file (absolute path) if not already present.
7	OnUploadFileData	Write data to a given file (absolute path). Most likely used in combination with OnUploadNewFile.
8	OnDownloadNewFile	Checks the file size of a given file (absolute path).
9	OnDownloadFileData	Return data of a given file (absolute path). Most likely used in combination with OnDownloadNewFile.
10	OnUploadMemoryData	Create a memory buffer and write the given shellcode, .NET assembly or PowerShell code to it. The shellcode is used in exec_builtin_cmd_inject, the .NET assembly in exec_builtin_cmd_net and the PowerShell code in exec_builtin_cmd_pscript.
14	-	This is the self-contained command line in contrast to the external command line via the wrapper app. This contains sub-values listed below.
14 - 0	exec_builtin_cmd_pwd	Return the current directory path.
14 - 1	exec_builtin_cmd_ls	Return the names, sizes, types and last modified times of all files in the current directory.
14 - 2	exec_builtin_cmd_cat	Return the data of a given file (absolute path).
14 - 3	exec_builtin_cmd_rm	Remove a given file (absolute path).
14 - 4	exec_builtin_cmd_process	Get names, PIDs, architectures and users of all running processes.
14 - 5	exec_builtin_cmd_sysinfo	Get detailed system information such as: • Current username • IIS information (major/minor version) • Windows OS information ◦ Product name ◦ Major/minor version ◦ Build number ◦ Platform ID ◦ Architecture • Current time and time zone • External IP address (api.ipify[.]org) • Internal IP address • Gateway IP address • DNS addresses • ARP table data • Adapter addresses • Environment variables
14 - 6	exec_builtin_cmd_exec	Create a process of a given file (absolute path).
14 - 7	exec_builtin_cmd_ps	Execute a given PowerShell code in its own run space.
14 - 8	exec_builtin_cmd_pscript	Execute a given PowerShell code from the memory buffer from OnUploadMemoryData in its own run space.
14 - 9	exec_builtin_cmd_net	The first option creates a new process, patches AMSI/ETW, creates a buffer in the process and reflectively loads the assembly from OnUploadMemoryData. The second option executes the assembly from OnUploadMemoryData in the current process via CLR hosting (CLRCreateInstance, …).
14 - 10	exec_builtin_cmd_inject	Inject the shellcode from OnUploadMemoryData into a new (remote thread injection), existing (remote thread injection) or the current process (new thread).

Table 1. Implemented commands in malware version 2.

The wrapper application (SHA256 hash: a28d0550524996ca63f26cb19f4b4d82019a1be24490343e9b916d2750162cda) used in ProcessCmdOperation is embedded in the .rdata section.

To load an assembly into a new process as part of the exec_builtin_cmd_net command, a small embedded loader DLL (SHA256 hash: aa2d46665ea230e856689c614edcd9d932d9edad0083bf89c903299d148634a2), also embedded in the .rdata section, is loaded into the process that in turn reflectively loads the assembly.

The returned result of each command (which can also be debug information in case of an error) is then AES-encrypted and Base64-encoded.

Malware version 1 has a slightly different implementation in functionality. It patches AMSI and ETW routines only in the routine that executes a .NET assembly in a new process and not at the beginning of the command data event handler like in version 2. Also, version 1 does not use an external command-line wrapper application for commands 3-5. Instead, it uses different implementations for these commands as shown in Table 2.

Command	Internal Term	Description
3	ExecuteCmd	Execute a given command-line command by spawning a child cmd.exe process and redirecting the result to a pipe.
4	GetExecutionResult	Read the command-line command result from the pipe from ExecuteCmd.
5	StopCmdChildProcess	Terminate the cmd.exe child process and close the pipe from ExecuteCmd.

Table 2. Different commands in malware version 1 in comparison to version 2.

While using native Windows API functions is not mandatory for C++/CLI applications, this malware extensively uses them for all of its features. Overall, this malware appears to be coded by a seasoned, old-school Windows developer. The author uses the classic Hungarian notation throughout the code. For example the malware uses the following variable names:

- wszExe
- pszArg
- pNetExeBuffer
- dwNetExeBufferSize
- uiBaseAddress
- strCmdValueEncrypted
- g_hBackendIPCServer
- g_aryBackendIPCServer

This malware has some inconsistent notations, debug messages and a few typos throughout the code that indicate the malware was not created by someone who speaks English as a first language. For example, the following list shows an excerpt of the debug strings used in the malware:

- [+] PID:
- [-] Exec Failed.
- [+] Inject Succeed
- [-] Inject Failed
- [+] .Net Exec Succeed
- [-] .Net Exec Failed
- [-] .Net Exec Timeout (>20s). Result Maybe Incomplete
- [-] Cat File Left Content Failed
- [-] Cat File 0 size
- [-] Cat File Failed
- [+] Detele Succeed
- [-] Detele Failed
- unknow

The above list contains misspellings of the words unknown and delete. We also find inconsistent use of tense, where Succeed is present tense, while Failed is past tense. Also using Result Maybe Incomplete, where the proper spelling should be Result May Be Incomplete.

Summary of C++/CLI IIS Backdoor

This passive IIS backdoor written in C++/CLI has numerous functionalities and is likely under active development. All network traffic is encrypted and encoded. Even though it has been professionally created, there appear to be weak spots that facilitate detection and analysis. All (debug) strings are stored in cleartext, making analysis easier. Additionally, the malware uses hard-coded passwords and keys for authentication.

We assess this malware is quite uncommon, because we have not yet discovered any other comparable samples. This rarity indicates the malware could have been used in a targeted attack, especially with its unusual development language and sophisticated nature. However, we cannot yet attribute this malware to any known threat actor.

Example 2: A Dixie-Playing Bootkit

What started as an analysis of a possible new implant from the Equation Group turned out to be one of the most peculiar threats we saw in 2024 in terms of its behavior.

At a first glance, the sample looked similar to previous malware attributed to the Equation Group. This sample has the typical exported function name dll_u, it uses multiple API functions from msvcrt.dll, and it abuses a third-party driver to gain access to kernel-mode. All these characteristics have been seen in EquationDrug and SlingShot samples too. Additionally, some security vendors classify this as a new EquationDrug sample.

This sample is also interesting because of its associated VirusTotal submission data. The sample was submitted from Oxford, Mississippi. It was uploaded with the file name w32analytics.dll to VirusTotal from the directory path C:\Windows\System32. This at least indicates it's from an actual ITW infection of a real victim, as this directory is reserved for the Windows operating system and commonly abused by malware. Beginning with Windows Vista, administrative privileges are required to write a file to the system32 directory. It indicates that this malware was placed there by an individual with admin privileges or another unidentified related malware that had administrative privileges. We have not found any other similar samples at this time.

This sample was compiled with MinGW and is signed by the University of Mississippi with an invalid certificate, with the issuer being it@olemiss[.]edu. These characteristics have not been seen in any previous samples from the stated threat actor. Finally, the malware's behavior is the main reason the sample most likely has nothing to do with the Equation Group.

Technical Analysis of a Dixie-Playing Bootkit

The sample (SHA256 hash: 950243a133db44e93b764e03c8d06b99310686d010b52b67f4effa57f0d72e04) is a 64-bit DLL and has two exported functions, dll_u and install.

Invoking the install export deletes any previous installations of the malware and creates a new scheduled task for persistence by using the following command:

- schtasks /create /tn w32analytics /sc ONCE /st 07:00 /ru SYSTEM /tr \"rundll32 w32analytics.dll,dll_u\"

This creates a scheduled task named w32analytics that is set to run once at 7:00 AM under the SYSTEM account. When triggered, this task executes the exported function dll_u from w32analytics.dll using the rundll32 command.

The dll_u function first uses zlib to decompress an embedded payload into memory. The decompressed payload is a 35 MB disk image. This image is a hybrid GRUB 2 bootloader designed to be compatible with both BIOS and UEFI systems.

The image is made of the following:

- A GRUB 2 master boot record (MBR)
- A BIOS boot partition that is the second stage of a GRUB 2 BIOS bootloader
- An EFI system partition (ESP) that contains the necessary data and files to run on a UEFI system

The threat then installs the bootloader on every physical disk with one of two options depending on the Windows OS version.

For Windows Vista and above, it drops a legitimate signed kernel driver named ampa.sys (SHA256 hash: 01D51DF682136CCE453BB1DA8964073E6BC7297CE4DAE7301C753BB618A69469) to disk, which is embedded in the resource section. The driver is later abused for the installation of the GRUB 2 bootloader disk image.

The installation procedure is as follows:

1. Create the driver file in C:\Windows\System32\ampa.sys
2. Adjust the process token with SeLoadDriverPrivilege privilege
3. Create the driver service in the Windows registry and set the needed values under HKLM\System\CurrentControlSet\Services\ampa
4. Load the driver with NtLoadDriver
5. Delete the driver service in the registry

The malware installs the driver programmatically by dynamically resolving and executing the following API functions:

- NtLoadDriver
- NtUnloadDriver
- RtlInitAnsiString
- RtlAnsiStringToUnicodeString
- RtlFreeUnicodeString
- LookupPrivilegeValueA
- OpenProcessToken
- AdjustTokenPrivileges

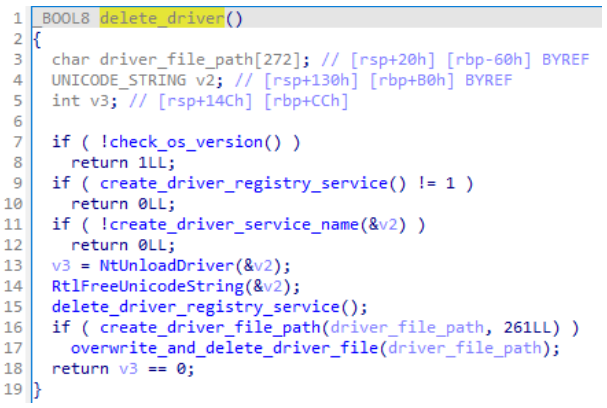
- RegOpenKeyExA
- RegCloseKey
- RegCreateKeyExA
- RegDeleteKeyA
- RegQueryValueExA
- RegSetValueExA

Now that the driver is loaded into kernel space, it abuses its write dispatch routine to write the bootloader into the first sector of each disk with the help of the drivers' symbolic link \\.\wowrt\DR\DISK%u.

When the malware is executed on a Windows version earlier than Vista, it uses the \.\PhysicalDrive%u symbolic link to install the bootloader.

After the bootloader is installed, it again creates the driver service in the registry to unload the driver from kernel space with NtUnloadDriver. When the driver is unloaded, it additionally overwrites the driver file on disk with zero bytes before it finally deletes it with DeleteFile.

Figure 2 shows the driver deletion routine.



```
_BOOL8 delete_driver()
{
  char driver_file_path[272]; // [rsp+20h] [rbp-60h] BYREF
  UNICODE_STRING v2; // [rsp+130h] [rbp+B0h] BYREF
  int v3; // [rsp+14Ch] [rbp+CCh]

  if ( !check_os_version() )
    return 1LL;
  if ( create_driver_registry_service() != 1 )
    return 0LL;
  if ( !create_driver_service_name(&v2) )
    return 0LL;
  v3 = NtUnloadDriver(&v2);
  RtlFreeUnicodeString(&v2);
  delete_driver_registry_service();
  if ( create_driver_file_path(driver_file_path, 261LL) )
    overwrite_and_delete_driver_file(driver_file_path);
  return v3 == 0;
}
```

Figure 2. Kernel driver deletion procedure as shown by IDA Pro's decompiler.

Finally, the malware tries to get SeShutdownPrivilege token rights to force a system reboot with the ExitWindowsEx function to trigger the bootloader execution.

When rebooted, the GRUB 2 bootloader shows an image and periodically plays Dixie through the PC speaker. This behavior could indicate that the malware is an offensive prank. Notably, patching a system with this customized GRUB 2 bootloader image of the malware only works on certain disk configurations.

We performed multiple tests on various Windows 10 virtual machines (VM) using both BIOS and UEFI firmware options during installation. Table 3 shows the results of execution on those test VMs along their corresponding partition configurations and firmware versions.

Partition structure (first partition on the left and last partition on the right, visually divided by "|")	**Firmware option used during installation**	**BIOS boot successful**	**UEFI boot successful**	**UEFI Secure boot successful**
| ESP (100 MB) | Windows (60 GB, NTFS) | System Recovery (550 MB) |	UEFI	No	No	No
| System Reserved (50 MB, NTFS) | Windows (60 GB, NTFS) | System Recovery (550 MB) |	BIOS	Yes	Yes	No
| Empty partition (1 GB, NTFS) | ESP (100 MB) | Windows (59 GB, NTFS) |	UEFI (with custom partition structure)	Yes	Yes	Yes

Table 3. Test results of malware executed on different Windows 10 systems.

This code was found in the GRUB 2 image extracted from its configuration file:

```
function load_video {
  if [ x$feature_all_video_module = xy ]; then
    insmod all_video
  else
    insmod efi_gop
    insmod efi_uga
    insmod ieee1275_fb
    insmod vbe
    insmod vga
    insmod video_bochs
    insmod video_cirrus
  fi
}

set linux_gfx_mode=
export linux_gfx_mode
load_video

insmod gfxterm
insmod png
terminal_output gfxterm

background_image /image.png

echo
sleep 60
play /dixie.play
configfile /grub2/grub.cfg
```

The function load_video checks the availability of all video modules. If no video modules are available, it loads specified video modules.

The commands set linux_gfx_mode= and export linux_gfx_mode set and export the variable for the Linux graphics mode, while the load_video function call loads video modules. Modules for the graphics terminal and PNG images are loaded through insmod gfxterm and insmod png respectively.

The output of the terminal is set to the graphics terminal through the command terminal_output gfxterm. An image is set as a background image for the GRUB menu using the command background_image /image.png. The GRUB menu is paused for 60 seconds using the commands echo and sleep 60. The Dixie audio file is played during this pause using the command play /dixie.play. Lastly, the location of the main GRUB configuration file is specified through the command configfile /grub2/grub.cfg.

Summary of a Dixie-Playing Bootkit

To our knowledge, this is the first malware that installs a GRUB 2 bootloader. While having a few characteristics of previous Equation Group samples, we do not believe this malware is connected to this threat actor. We believe this malware is a PoC created by somebody from the University of Mississippi and they might have dropped it on a campus computer.

While the abused third-party driver was later also found to be vulnerable by Northwave Cyber Security, this malware merely abused it to write the bootloader to disk, because this driver is also unsecured. There is no exploit used, but it rather abuses the driver's unsecured write dispatch routine. The usual term "bring your own vulnerable driver" (BYOVD) wouldn't really fit in this case.

Example 3: A Red Team Framework Named ProjectGeass

This stood out from the various red team tools we came across in 2024 because it seems to be a new multi-platform post-exploitation framework written from scratch and still in development. This malware is named ProjectGeass and is a self-described beacon Windows sample. The term beacon commonly describes the agent of a post-exploitation toolkit.

This sample was submitted to VirusTotal from Singapore as the only file from that source.

This ProjectGeass sample was developed in C++ and contains several debug messages and artifacts with some indicators of other beacons for Android and Unix/Linux. The sample has the OpenSSL and Boost.Asio libraries statically linked, making it quite large at 6 MB.

Interestingly this tool uses the term “maneuver” for the execution of third-party files, indicating that this framework could have been used for a red team/blue team test.

Technical Analysis of a Red Team Framework Named ProjectGeass

The ProjectGeass beacon sample is a 64-bit Windows executable (SHA256 hash: cca5df85920dd2bdaaa2abc152383c9a1391a3e1c4217382a9b0fce5a83d6e0b) that was compiled on Oct. 31, 2023, with Microsoft Visual Studio C++. It has multiple project paths left as debug artifacts, giving a good impression of the inner structure of the project:

```
D:\source\repos\ProjectGeass\beacon\CommandExecute\CommandExecuteWindows.cpp
D:\source\repos\ProjectGeass\beacon\Config\AppConfigurator.cpp
D:\source\repos\ProjectGeass\beacon\EndpointInformation\EndpointInfoCollectorBase.cpp
D:\source\repos\ProjectGeass\beacon\EndpointInformation\EndpointInfoCollectorOnWindows.cpp
D:\source\repos\ProjectGeass\beacon\ExecuteThirdPartyFiles\ExecuteThirdPartyFilesBase.h
D:\source\repos\ProjectGeass\beacon\ExecuteThirdPartyFiles\ExecuteThirdPartyFilesOnWindows.cpp
D:\source\repos\ProjectGeass\beacon\FileManager\FilesManagerCommon.h
D:\source\repos\ProjectGeass\beacon\FileManager\DownloadFileManager.cpp
D:\source\repos\ProjectGeass\beacon\FileManager\FileManager.cpp
D:\source\repos\ProjectGeass\beacon\FileManager\UploadFileManager.cpp
D:\source\repos\ProjectGeass\beacon\KeyLogger\KeyLoggerImpl.cpp
D:\source\repos\ProjectGeass\beacon\ListDirectory\ListDirectoryCrossPlatform.cpp
D:\source\repos\ProjectGeass\beacon\Network\Packet.cpp
D:\source\repos\ProjectGeass\beacon\Network\TCPClient.cpp
D:\source\repos\ProjectGeass\beacon\Network\TCPSession.cpp
D:\source\repos\ProjectGeass\beacon\Online\OnlineAndHeartbeat.cpp
D:\source\repos\ProjectGeass\beacon\ProcessManage\ProcessManageOnWindows.cpp
D:\source\repos\ProjectGeass\beacon\ListDirectory\ListDirectoryBase.h
D:\source\repos\ProjectGeass\beacon\CommandExecute\CommandExecuteBase.h
D:\source\repos\ProjectGeass\beacon\ProcessManage\ProcessManageBase.h
D:\source\repos\ProjectGeass\beacon\TaskManage\TaskHandler.cpp
D:\source\repos\ProjectGeass\beacon\TaskManage\TaskProcessor.cpp
```

We can use a tool like SusanRTTI and GraphWiz to visualize the C++ Run-time type information (RTTI) to get a better understanding of the code structure. Figure 3 shows an excerpt of the class inheritances in this ProjectGeass sample.

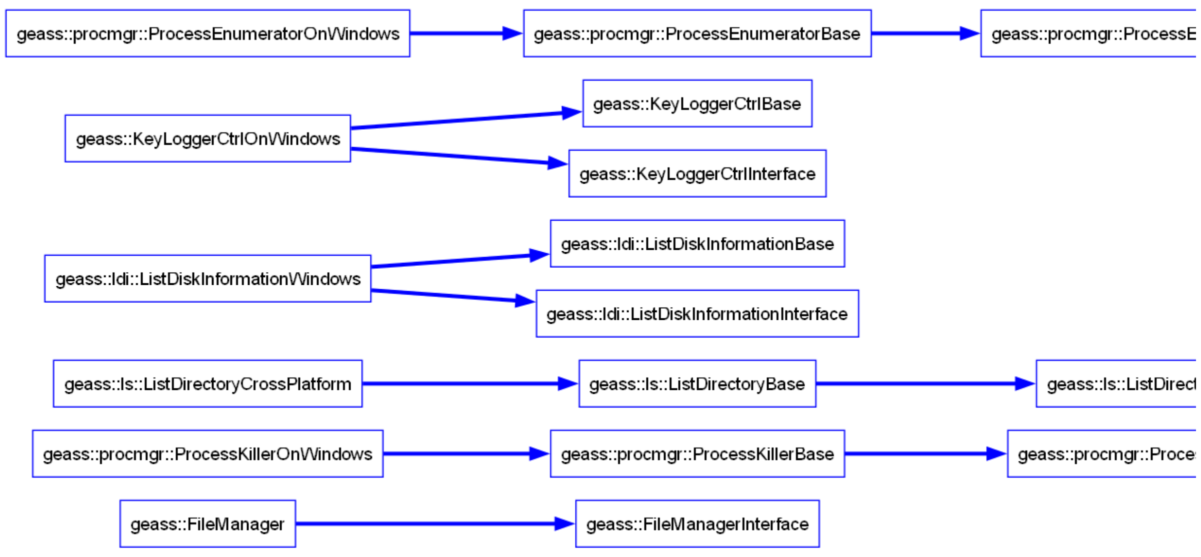

geass::procmgr::ProcessEnumeratorOnWindows
geass::procmgr::ProcessEnumeratorBase
geass::procmgr::ProcessE
geass::KeyLoggerCtrlOnWindows
geass::KeyLoggerCtrlBase
geass::KeyLoggerCtrlInterface
geass::ldi::ListDiskInformationWindows
geass::ldi::ListDiskInformationBase
geass::ldi::ListDiskInformationInterface
geass::ls::ListDirectoryCrossPlatform
geass::ls::ListDirectoryBase
geass::ls::ListDirect
geass::procmgr::ProcessKillerOnWindows
geass::procmgr::ProcessKillerBase
geass::procmgr::Proce
geass::FileManager
geass::FileManagerInterface

Figure 3. ProjectGeass class inheritances based on the C++ RTTI information.

As noted in Figure 3, the authors named multiple classes Windows or OnWindows, which implies there are other classes with the same purpose but for different operating systems. This ProjectGeass sample also contains a class named ListDirectoryCrossPlatform that hints at support for other platforms. Also, as part of the endpoint collection routines, this sample tries to figure out if the platform it's executed on is Windows, Android, Unix or Linux. All these indicators suggest that ProjectGeass is a multi-platform post-exploitation framework supporting multiple operating systems.

The ProjectGeass beacon has the following features:

- File upload/download
- Execute Windows commands
- Get/set heartbeat data
- Sleep time adjustment
- Enumerate processes
- Start/stop keylogger
- Process listing/termination
- File manager (e.g., create/list/rename/delete directories, files, attributes)
- Receive and execute payloads
- Get endpoint information (e.g., network, disk, user)

While most strings are stored in cleartext, some are encrypted with a simple XOR-based algorithm with each string having its own key. Table 4 shows the decrypted strings with their connected features.

Decrypted Strings	**Used To**
"cmd.exe /C"	Create a process from pipe data as part of the self-contained commands feature
"Administrators"	Get network user information as part of the endpoint information collection feature
"ROOT\CIMV2", "SELECT UUID FROM Win32_ComputerSystemProduct", "WQL", "UUID"	Get OS information as part of the endpoint information collection feature
"S-1-5-18"	Process token adjustment
"The operating system is: %WINDOWS_LONG%", "winbrand.dll", "BrandingFormatString"	Used to get the Windows version string (described here: How to tell the "real" version of Windows your app is running on?)
"MyWindowClass"	Dummy window for the keylogger
"ROOT\SecurityCenter", "SELECT * FROM AntiVirusProduct", "DisplayName"	Endpoint antivirus information collection via WMI
"http", "ipv4.renfei.net", "GET / HTTP/1.0", "Host: ipv4.renfei.net", "Accept: text/plain", "Connection: close", "Invalid response", "Response returned with status code: "	Get an external IP address as part of the endpoint network information collection
"SOFTWARE\Microsoft\Cryptography", "MachineGuid"	Cryptographic related information

Table 4. Decrypted strings and their purposes.

The configuration data is located in the .data section and is RC4-encrypted. This data is implemented as a structure with the decryption key in cleartext (F5g3dsriT05L5RuTfHZlJX4dJfOVRJlsWjLC) followed by the encrypted configuration data.

Table 5 shows the decrypted configuration data.

Information	Decrypted Data
Server address	10.4.7[.]149
Server port	7515
Server certificate	-----BEGIN CERTIFICATE----- MIID6zCCAlOgAwIBAgIQOIFwtYsC2Pu4YtNz3mOGBzANBgkqhkiG9w0BAQsFADAO MQwwCgYDVQQDEwNucGQwHhcNMjMxMDI2MDYyNTA4WhcNMzMxMDI3MDYyNTA4WjAO MQwwCgYDVQQDEwNucGQwggGiMA0GCSqGSIb3DQEBAQUAA4IBjwAwggGKAoIBgQC/ j31oOFSGU7Vb/cpv39AMxFBewosWGOAmg+qtSBsz1o0gj/nLKuGquYgYCvfzla4B sLOpbk32Zh32KtOnq+vvQ4d/iK2yFLc6hWD24hGsNQ1uIyFPbnmQ+Xu6hJ9SNv5m WUIo9sxNQCobBS1dEl/n7FN9nX/XGO2ydBRPMJ9ppyrGjY7a9deITgNcqajgUJuW OTq2m4D7T2O8Lgon28tLf5ETiJIrnw+RH+ezt7jiF5oqd+W6hVSmtk57RQHD/u+h bA9u+j6J45gtikeD70kibZ4X3fzv3UNRSj93ubCx/i+H2MdKbvhDULjo83cLlhqj iHZp3wfRO4GeG9i96HANCr7w5o3Cw37fDBYGDJs9KUFeqKAeKLM5xTlh4+A4m+aF herWmRuX6sQnQSkifPdF44gymbYQTs+pWFSwNsoS6jZ+X5kX3Ddr/B07uOqPqaGZ olSjwzGqIB2cOgb7/RotLb7W9dvhhwKlmX11BdQpD0daRPYeXLcuXaS4Fp9nV40C AwEAAaNFMEMwDgYDVR0PAQH/BAQDAgIEMBIGA1UdEwEB/wQIMAYBAf8CAQAwHQYD VR0OBBYEFBSvya4X86b9540iQiX5x+0eGqWqMA0GCSqGSIb3DQEBCwUAA4IBgQAT zWrz+ZfpSpsydRW1LRtCx1FCh6bGlRCJZokiETh4l9G526X413SsUcclhJ5ykbIE vCQPZbhixiUloLCczFUvT2Ey1h5zvABE9ah1iB1CAYzukrS4/TXrkLIBa+UazjIG NKS2favWTH1rv719dh4/YvgatNAXi7TA66k9ji57ojf2DgIzwEV0Sk16seeWqqGs eeHATMkx05kvUTdsdKO4ElzsX4qsfIIzPEe18mL4x0sns40o05b1oMnGFYXtbYV8 4sOB4GfubU+PQBOBzYI1U7RZip+OpHgLTntLLSrbyemKklhcivlTLmI4Vg4uWZw1 pMcd9IQieNWLmesJS8FKDxf9BT0PXrAstNKZ8nx3BZqy3KkdC9CHI9DKDuIqilV2 gMxncdDuTdGV1mgfUrW92fjO08DerfyMv7xhIKTpBYjkek+Y09oVC1OnSC3lVc6I SfelPFioCvBF0lpevtlR/L61Q0qIxOk+o41infeZGS1QmBmE6gvlbtH1C9yZ/RQ= -----END CERTIFICATE-----
Proxy address	-
Proxy port	-
Proxy username	-
Proxy password	-
Project ID	1726486365509521408
Mutex ID	1726489580380622848
Online time point	-
Sleep duration	-
Verify certificate	1 (True)

Table 5. Decrypted beacon configuration data.

Summary of a Red Team Framework Named ProjectGeass

ProjectGeass is a post-exploitation framework that appears to have been developed for a professional or commercial purpose. As we have not yet found any other similar samples, this may be a private or non-public project. We cannot attribute it to any known company or organization. Since this malware uses a Chinese site (ipv4.renfei[.]net) to check its host's external IP address, the creator might be Chinese. However, that is all conjecture.

Conclusion

A number of new and interesting types of malware appeared in the past year, each using strategies that had not been reported before. This article reviewed three examples.

The first piece of malware we examined is a passive IIS backdoor that showed indications that attackers used it in targeted attacks. It was also developed in a programming language rarely used for malware, C++/CLI.

The second sample uses a third-party kernel driver to install a GRUB 2 bootloader, which we have not seen before.

The third sample, named ProjectGeass, appears to be a new post-exploitation framework in development. This may have been created for professional or commercial purposes, and possibly developed by a Chinese speaker.

Palo Alto Networks customers are better protected from these malware samples through Advanced WildFire, with its different memory analysis features.

Cortex XDR and XSIAM are designed to prevent the execution of known malicious malware, and also prevent the execution of unknown malware using Behavioral Threat Protection and machine learning based on the Local Analysis module.

If you think you may have been compromised or have an urgent matter, get in touch with the Unit 42 Incident Response team or call:

- North America: Toll Free: +1 (866) 486-4842 (866.4.UNIT42)
- UK: +44.20.3743.3660
- Europe and Middle East: +31.20.299.3130
- Asia: +65.6983.8730
- Japan: +81.50.1790.0200
- Australia: +61.2.4062.7950
- India: 00080005045107

Palo Alto Networks has shared these findings with our fellow Cyber Threat Alliance (CTA) members. CTA members use this intelligence to rapidly deploy protections to their customers and to systematically disrupt malicious cyber actors. Learn more about the Cyber Threat Alliance.

Indicators of Compromise

IIS Backdoor

SHA256 hash: 15db49717a9e9c1e26f5b1745870b028e0133d430ec14d52884cec28ccd3c8ab

- File size: 238,592 bytes
- File name: proxyscrape.dll
- File name internal: proxyxml_v4.dll
- File type: 64-bit Windows DLL
- Description: Main module version 2

SHA256 hash: aa2d46665ea230e856689c614edcd9d932d9edad0083bf89c903299d148634a2

- File size: 15,360 bytes
- File name: -
- File name internal: ReflectiveDLL.dll
- File type: 64-bit Windows DLL
- Description: Reflective loader embedded in main module version 2

SHA256 hash: a28d0550524996ca63f26cb19f4b4d82019a1be24490343e9b916d2750162cda

- File size: 19,456 bytes
- File name: VC_REDIST_CONFIG_X64.TXT
- File name internal: -
- File type: 64-bit Windows EXE
- Description: Wrapper application for cmd.exe embedded in main module version 2

SHA256 hash: 8571a354b5cdd9ec3735b84fa207e72c7aea1ab82ea2e4ffea1373335b3e88f4

- File size: 191,488 bytes
- File name: proxyxml.dll
- File name internal: IISShellModule.dll
- File type: 64-bit Windows DLL
- Description: Main module version 1

SHA256 hash: 94017628658035206820723763a2a698a4fd7be98fc2c541aad6aa0281ef090e

- File size: 14,848 bytes
- File name: -
- File name internal: ReflectiveDLL.dll

- File type: 64-bit Windows DLL
- Description: Reflective loader embedded in main module version 1

Bootkit

SHA256 hash: 950243a133db44e93b764e03c8d06b99310686d010b52b67f4effa57f0d72e04

- File size: 6,444,544 bytes
- File name: w32analytics.dll
- File name internal: loader.dll
- File type: 64-bit Windows DLL

ProjectGeass

SHA256 hash: cca5df85920dd2bdaaa2abc152383c9a1391a3e1c4217382a9b0fce5a83d6e0b

- File size: 6,040,576 bytes
- File name: -
- File name internal: -
- File type: 64-bit Windows EXE