

NANOREMOTE, cousin of FINALDRAFT

 elastic.co/security-labs/nanoremove

Daniel Stepanic

December 11, 2025

The fully-featured backdoor we call NANOREMOTE shares characteristics with malware described in REF7707 and is similar to the FINALDRAFT implant.

 24 min read  [Malware Analysis](#)

 NANOREMOTE, cousin of FINALDRAFT

Introduction

In October 2025, Elastic Security Labs discovered a newly-observed Windows backdoor in telemetry. The fully-featured backdoor we call NANOREMOTE shares characteristics with malware described in [REF7707](#) and is similar to the [FINALDRAFT](#) implant.

One of the malware's primary features is centered around shipping data back and forth from the victim endpoint using the [Google Drive API](#). This feature ends up providing a channel for data theft and payload staging that is difficult for detection. The malware includes a task management system used for file transfer capabilities that include queuing download/upload tasks, pausing/resuming file transfers, canceling file transfers, and generating refresh tokens.

This report aims to enhance awareness among defenders and organizations regarding the threat actors we are monitoring and their evolving capabilities.

Key takeaways

- Elastic Security Labs discovers a new Windows backdoor
- NANOREMOTE likely developed by espionage threat actor linked to FINALDRAFT and REF7707
- NANOREMOTE includes command execution, discovery/enumeration and file transfer capabilities using Google Drive API
- The backdoor integrates functionality from open-source projects including Microsoft Detours and libPeConv
- Elastic Defend prevents the NANOREMOTE attack chain through behavioral rules, machine learning classifier, and memory protection features

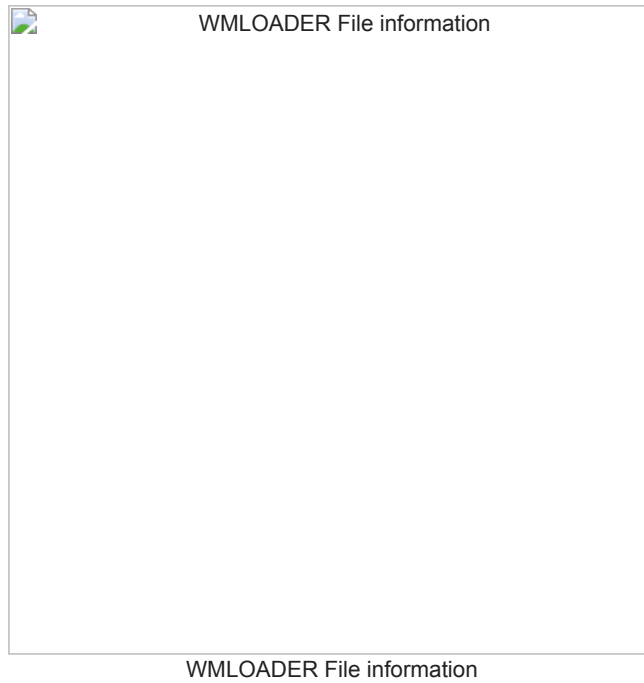
NANOREMOTE analysis

WMLOADER

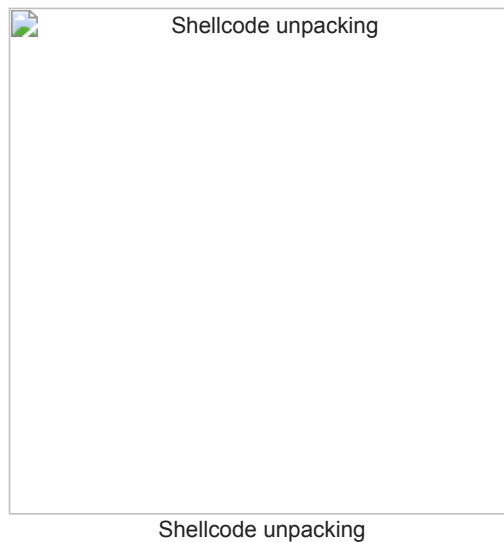
The observed attack chain consists of two primary components, a loader (WMLOADER) and payload (NANOREMOTE). Although this report focuses on NANOREMOTE, we will describe the loader to provide context on the overall infection flow.



[WMLOADER](#) masquerades as a Bitdefender Security program (**BDReinit.exe**) with an invalid digital signature.



After execution, the program makes a large number of calls to Windows functions ([VirtualAlloc](#) / [VirtualProtect](#)), preparing the process to host embedded shellcode stored within the file. The shellcode is located at RVA ([0x193041](#)) and decrypted using a rolling XOR algorithm.



This shellcode looks for a file named [wmsetup.log](#) in the same folder path as WMLOADER then starts decrypting it using AES-CBC with a 16-byte ASCII key ([3A5AD78097D944AC](#)). After decryption, the shellcode executes the in-memory backdoor, NANOREMOTE.

Based on the previous shellcode decryption routine, we can identify other related samples targeting Bitdefender and Trend Micro products when searching in VirusTotal.



NANOREMOTE

NANOREMOTE is a fully-featured backdoor that can be used to perform reconnaissance, execute files and commands, and transfer files to and from victim environments. The implant is a 64-bit Windows executable written in C++ without obfuscation.

NANOREMOTE Configuration

The NANOREMOTE sample we observed was preconfigured to communicate with a hard-coded non-routable IP address. We believe the program was generated from a builder as we do not see any cross-references pointing to a configuration setting.

For the Google Drive API authentication, NANOREMOTE uses a pipe-separated configuration that can use multiple clients. The `|*|` separator splits the fields used by a single client and the `|-|` is used as a marker to separate the clients. There are three fields per client structure:

- Client ID
- Client Secret
- Refresh Token

Below is an example of the format:

```
Client_ID_1|*|Client_Secret_1|*|Refresh-Token_1|-|Client_ID_2|*|Client_Secret_2|*|Refresh-Token_2
```

The developer has a fallback mechanism to accept this configuration through an environment variable named `NR_GOOGLE_ACCOUNTS`.

Interface/Logging

NANOREMOTE provides a detailed console displaying the application's real-time activity, including timestamps, source code locations, and descriptions of its behaviors.



Console output

Console output

A new Windows directory is created in the same location where NANOREMOTE was executed, the folder is called **Log**.



Log folder storing log file

Log folder storing log file

A newly created log file (`pe_exe_run.log`) is dropped in this folder containing the same output printed from the console.



Log file contents

Log file contents

Setup

There is an initial setup routine by NANOREMOTE before the main worker loop starts. The malware generates a unique GUID via `CoCreateGuid` then hashes the GUID using the [Fowler-Noll-Vo \(FNV\)](#) function. This GUID is used by the operator to identify individual machines during each request.



The malware developer has a process-wide crash handler to create a Windows minidump of the running process when an unhandled exception occurs, this is most likely being used to triage program crashes.



TopLevelExceptionFilter function

TopLevelExceptionFilter function

The exception will produce the dump before terminating the process. This is a pretty standard practice although the `MiniDumpWithFullMemory` might be considered less common in legitimate software as it could end up producing larger sized dumps and contain sensitive data.



Minidump generation

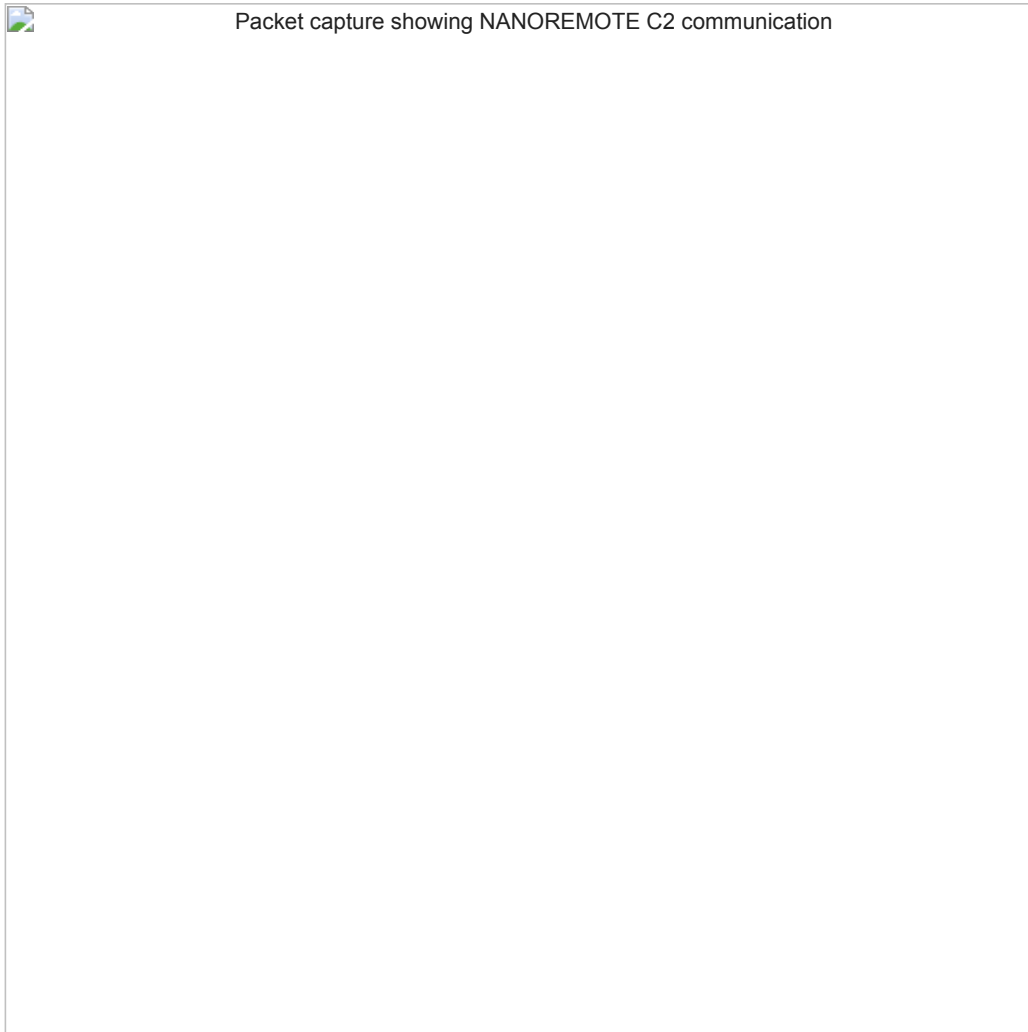
Minidump generation

A quick Google search using the same string formatter for the dump file (`%d%02d%02d%02d%02d_sv.dmp`) listed only 1 result from a Chinese-based software development [website](#).



Network Communication

As mentioned previously, NANOREMOTE's C2 communicates with a hard-coded IP address. These requests occur over HTTP where the JSON data is submitted through POST requests that are Zlib compressed and encrypted with AES-CBC using a 16-byte key (558bec83ec40535657833d7440001c00). The URI for all requests use /api/client with User-Agent (NanoRemote/1.0).



Packet capture showing NANOREMOTE C2 communication

Below is the CyberChef recipe used for the C2 encryption/compression:



Each request prior to encryption, follows a schema consisting of:

- **Command ID:** Associated command handler ID
- **Data:** Command-specific object containing key/value pairs required by the corresponding handler
- **ID:** Unique machine identifier assigned to the infected host

Below is an example of a request that triggers execution of whoami via the command key inside the data object:

```
{
  "cmd": 21,
  "data": {
    "command": "whoami"
  },
  "id": 15100174208042555000
}
```

Each response follows a similar format using the previous fields along with two additional fields.

- **Output:** Contains any output from the previously requested command handler
- **Success:** Boolean flag used to determine if command was successful or not

Below is an example of the response from the previous whoami command:

```
{
  "cmd": 21,
  "data": 0,
  "id": 17235741656643013000,
  "output": "desktop-2c3iqho\\rem\\r\\n",
  "success": true
}
```

Command Handlers

NANOREMOTE's main functionality is driven through its 22 command handlers. Below is a control-flow graph (CFG) diagram showcasing the switch statement used to dispatch the different handlers.



Below is the command handler table:

Command ID	Description
#1	Collect host-based information
#2	Modify beacon timeout
#3	Self-termination
#4	List folder contents by path
#5	List folder contents by path and set working directory
#6	Get storage disk details
#7	Create new directory
#8 #9	Delete directory/files
#10 #11	Teardown (Clear cache, cleanup)
#12	PE loader - Execute PE from disk
#13	Set working directory
#14	Get working directory
#15	Move file
#16	Queue download task via Google Drive
#17	Queue upload task via Google Drive
#18	Pause download/upload transfer
#19	Resume download/upload transfer
#20	Cancel file transfer
#21	Command execution
#22	PE loader - Execute PE from memory

Handler #1 - Collect host-based information

This handler enumerates system and user details to profile the victim environment:

- Uses `WSAIoct1` with `SIO_GET_INTERFACE_LIST` to retrieve internal and external IP address
- Grabs username via `GetUserNameW`
- Retrieves the hostname via `GetComputerNameW`
- Checks if current user is member of Administrator group via `IsUserAnAdmin`
- Retrieves the process path used by the malware using `GetModuleFileNameW`
- Retrieves operating-system information (product build) from the registry using the `WinREVersion` and `ProductName` value names
- Gets process ID of running program via `GetCurrentProcessID`

Below is an example of the data sent to the C2 server:

```
{
  "cmd": 1,
  "data": {
    "Arch": "x64",
    "ExternalIp": "",
    "HostName": "DESKTOP-2C3IQH0",
    "ID": 8580477787937977000,
    "InternalIp": "192.168.1.1",
    "OsName": "Windows 10 Enterprise ",
    "ProcessID": 304,
    "ProcessName": "pe.exe",
    "SleepTimeSeconds": 0,
    "UID": 0,
    "UserName": "REM *"
  },
  "id": 8580477787937977000
}
```

Handler #2 - Modify beacon timeout

This handler modifies the beacon timeout interval for NANOREMOTE's C2 communication, the malware will sleep based on the number of seconds provided by the operator.



Beacon timeout adjusted

Beacon timeout adjusted

Below is an example of this request where NANOREMOTE uses the key (*interval*) with a value (*5*) to modify the beacon timeout to 5 seconds.

```
{
  "cmd": 2,
  "data": {
    "interval": 5
  },
  "id": 15100174208042555000
}
```

Handler #3 - Self-termination

This handler is responsible for setting a global variable to 0 effectively signaling the teardown and process exit for NANOREMOTE.

Handler #4 - List folder contents by path

This handler lists the folder contents using a provided file path from the operator. The listing for each item includes:

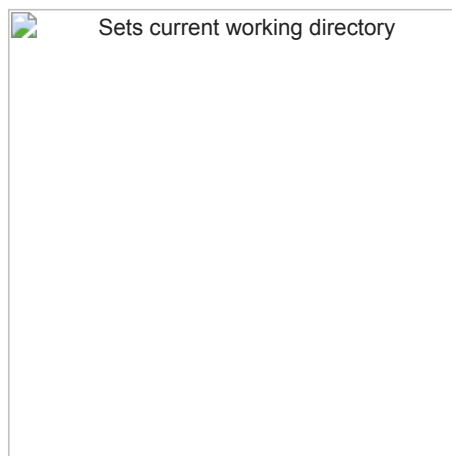
- Whether the item is a directory or not
- Whether the item is marked as hidden
- Last modified date
- File name
- Size



Folder directory enumeration

Handler #5 - List folder contents and set working directory

This handler uses the same code as the previous handler (#4), the only difference is that it sets the current working directory of the process to the provided path as well.



Sets current working directory

Handler #6 - Get Storage Disk Info

This handler uses the following Windows API functions to collect storage disk information from the machine:

- GetLogicalDrives
- GetDiskFreeSpaceExW
- GetDriveTypeW
- GetVolumeInformationW



Storage disk info printed from console

Below is an example of the request in JSON showing the data returned:

```
{
  "cmd": 6,
  "data": {
    "items": [
      {
        "free": 26342813696,
        "name": "C:",
        "total": 85405782016,
        "type": "Fixed"
      }
    ]
  },
  "id": 16873875158734957000,
  "output": "",
  "success": true
}
```

Handler #7 - Create new folder directory

This command handler creates a new directory based on a provided path.

Handler #8, #9 - Delete file, directory

This handler supports both #8 and #9 command ID's, the branching is dynamically chosen based on the provided file path. It has the ability to delete files or a specified folder.

Handler #10, #11 - Teardown/Cleanup

These two handlers call the same teardown function using different arguments to recursively release heap allocations, internal C++ objects, and cached data associated with the malware's runtime. This purpose is to clean up the command structures and prevent memory leaks or instability.

Handler #12 - Custom PE Loader - Execute PE from disk

This handler includes a custom PE loading capability for files that exist on disk. This functionality leverages standard Windows APIs along with helper code from library [libPeConv](#) to load PE files from disk without using the traditional Windows loader.

In short, it will read a PE file from disk, copy the file into memory, manually map the sections/headers, preparing the file before finally executing it in memory. This implementation is a deliberate technique for stealth and evasion bypassing user-mode hooking and traditional visibility. As one example, when a file is executed through this technique, there is no trace of this executable being launched using procmon.



NANOREMOTE loading PE file from disk into memory

NANOREMOTE loading PE file from disk into memory

Below is the following input for this handler where the local file path is provided under the key (args):

```
{
  "cmd": 12,
  "data": {
    "args": "C:\\tmp\\mare_test.exe"
  },
  "id": 15100174208042555000
}
```

The following screenshot shows successful execution of our test executable using this technique:



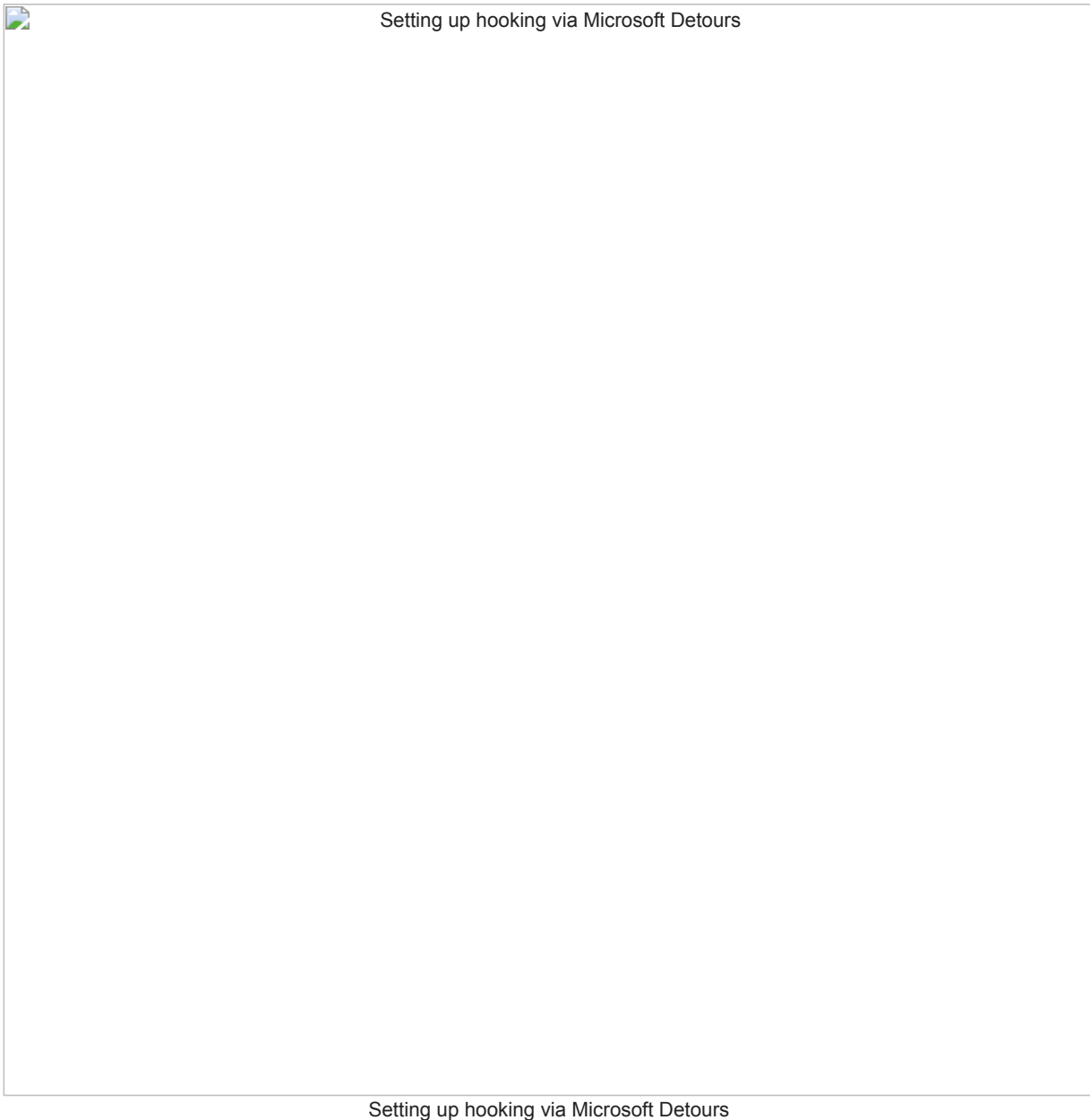
During this analysis, one interesting note is the adoption of the libPeConv [library](#), this is a great and useful project that we ourselves use internally for various malware-related tasks. The developer of NANOREMOTE uses several functions from this library to simplify common tasks related to manually loading and executing PE files in memory. Below are the functions used by the library found in NANOREMOTE:

- `default_func_resolver`: Resolves functions in a PE file by dynamically loading DLLs and retrieving the addresses of exported functions.
- `hooking_func_resolver`: Retrieve the virtual address of a function by name from a loaded DLL.
- `FillImportThunks`: Populates the import table by resolving each imported function to its actual address in memory.
- `ApplyRelocCallback`: Applies base relocations when a PE file is loaded at an address different from its preferred base.

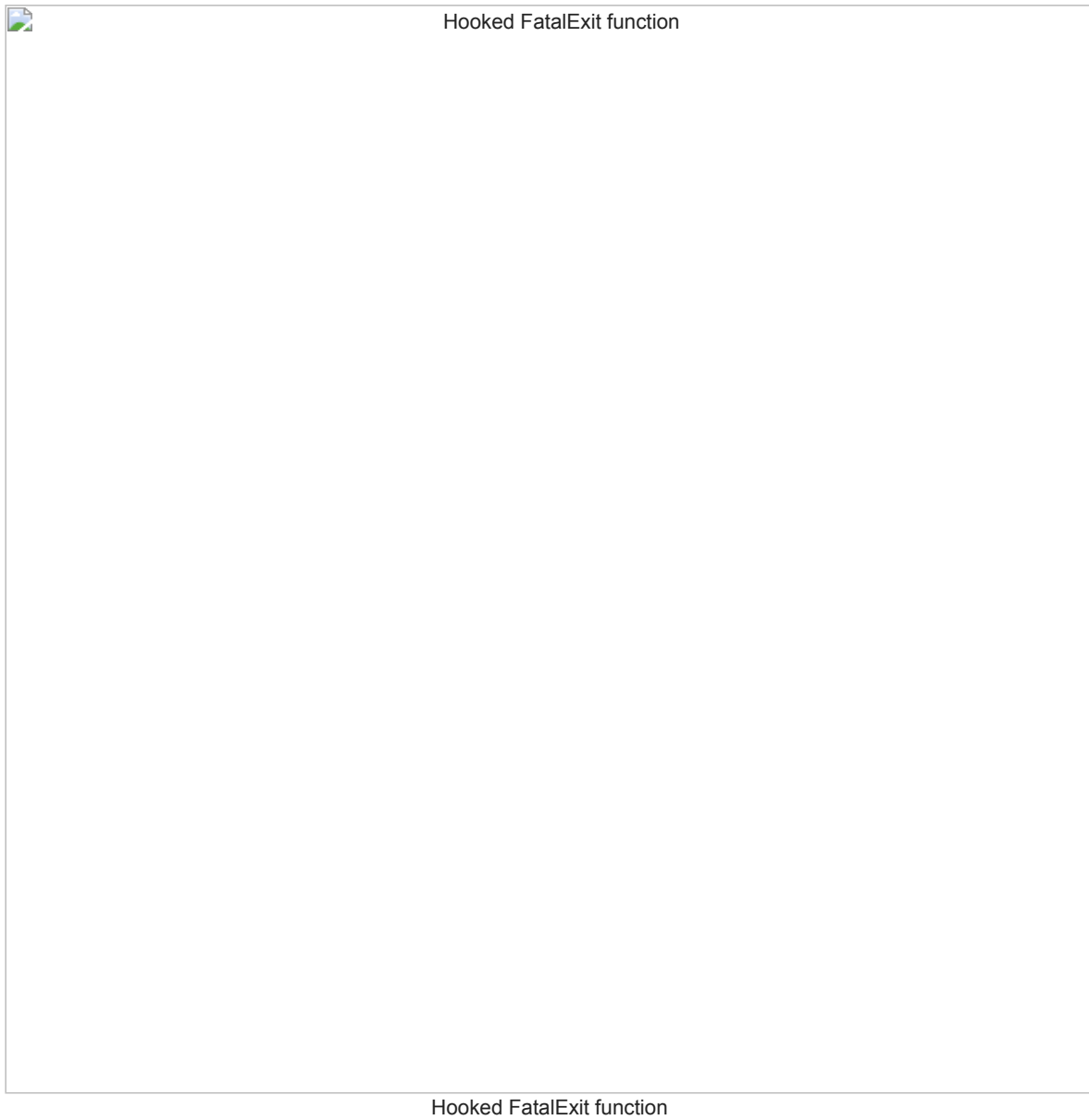


Another notable observation in this handler is the use of the open-source hooking library, [Microsoft Detours](#). This library is used to intercept the following Windows functions:

- GetStdHandle
- RtlExitUserThread
- RtlExitUserProcess
- FatalExit
- ExitProcess



This runtime hooking routine intercepts termination-related functions to enforce controlled behavior and improve resiliency. For example, NANOREMOTE prevents a failure in a single worker thread from terminating the entire NANOREMOTE process.



Hooked FatalExit function

Handler #13 - Set working directory

This handler sets the working directory to a specific directory using the key (path). Below is an example request:

```
{
  "cmd": 13,
  "data": {
    "path": "C:\\tmp\\Log"
  },
  "id": 1510017420804255000
}
```

Handler #14 - Get working directory

This handler retrieves the current working directory, below is an example response after setting the directory with previous handler (#13).

```
{  
  "cmd": 14,  
  "data": 0,  
  "id": 11010639976590963000,  
  "output": "[+] pwd output:\r\nC:\\\\tmp\\\\Log\r\n",  
  "success": true  
}
```

Handler #15 - Move File

This handler allows the operator to move files around the victim machine using MoveFileExW with two arguments (old_path, new_path) moving the file to a different folder by performing a copy and delete file operation.



Moving files via MoveFileExW

Moving files via MoveFileExW

Handler #16 - Queue Download Task

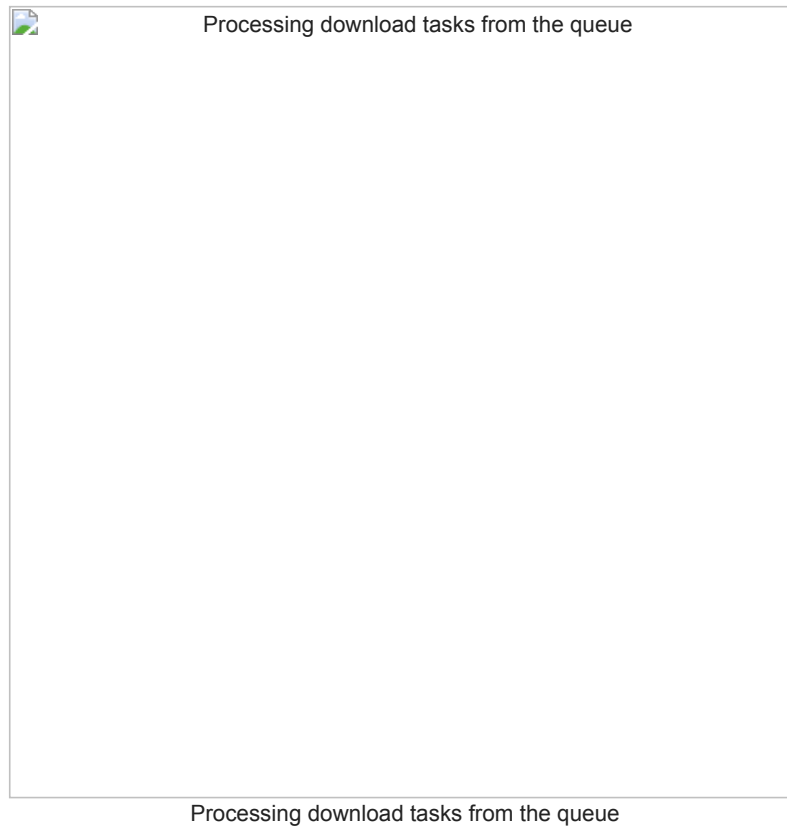
This handler creates a download task object with a provided task_id then enqueues the task into the download queue. This implementation uses OAuth 2.0 tokens to authenticate requests to the Google Drive API. This functionality is used by the threat actor to download files to the victim machine. The encrypted communication to Google's servers makes this traffic appear legitimate, leaving organizations unable to inspect or differentiate it from normal use.



Task added to queue

Task added to queue

Inside the main worker thread, there is a global variable used to track queue objects and process the awaiting tasks by the malware.



A task is processed using various fields provided by the C2 server:

- type
- task_id
- file_id
- target_path
- file_size
- md5

When a download task is processed, NANOREMOTE will retrieve the size of the file hosted on Google Drive using the file ID (1BwdUSlyA3WTUrpAEEDhG0U48U9hYPcy7). Next, the malware will download the file via WinHttpRequest then use WinHttpWriteData to write the file on the machine.



Fiddler session showing download from Google Drive

Below is the console output showing this download process:



Downloading file from Google Drive using tasking system

This malware feature poses a unique challenge for organizations as threat groups continue to abuse trusted cloud platforms for data exfiltration and payload hosting. This traffic without any context can easily blend in with legitimate traffic making detection difficult for defenders who rely on network visibility.

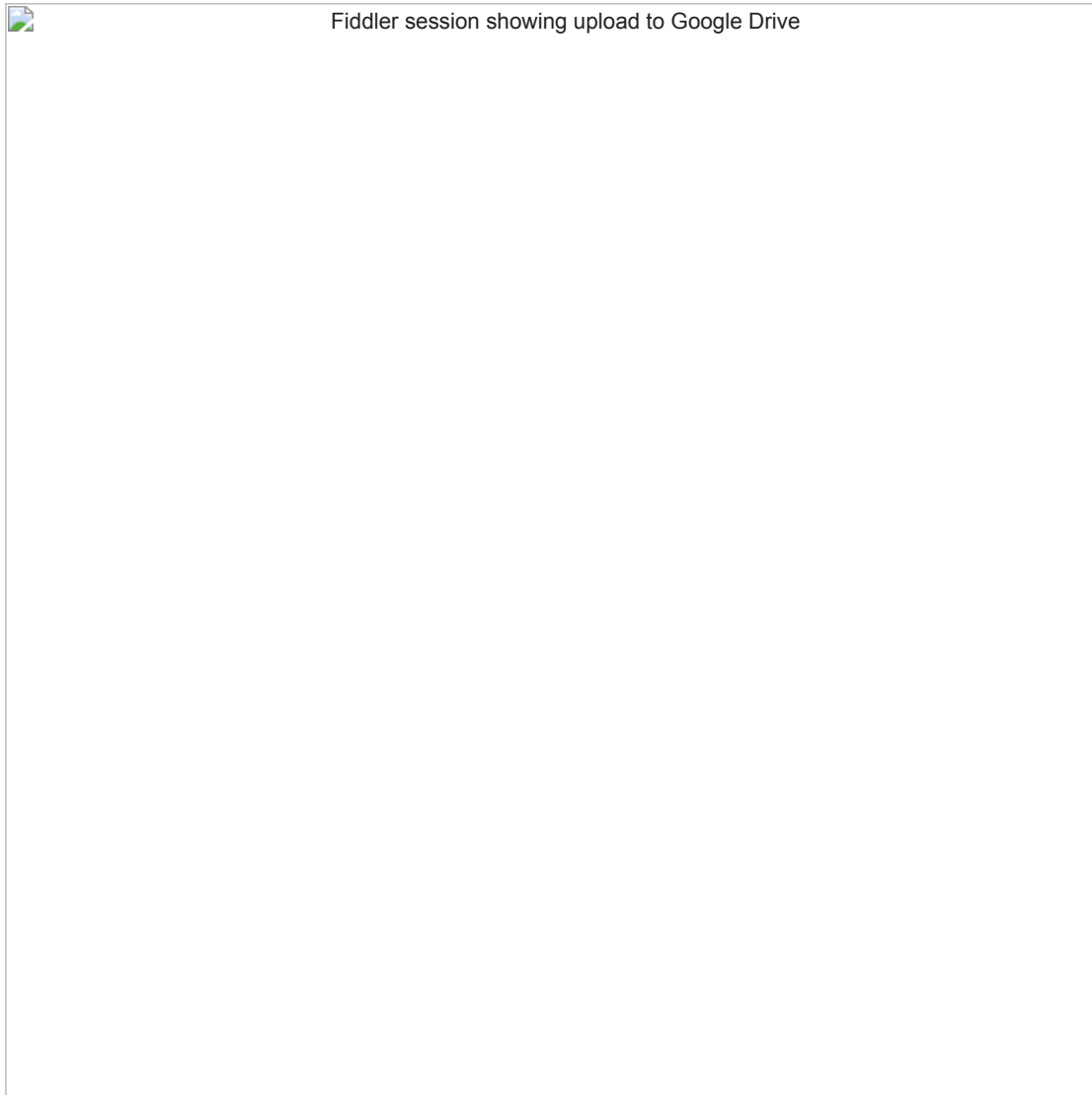
Handler #17 - Queue Upload Task

This handler works in similar fashion as the previous handler (#16), instead it is creating an upload queue task and enqueueing the task into the upload queue. This handler is used by the threat actor to upload files from the victim machine to the adversary's controlled Google Drive account.

The following fields are provided by the operator through the C2 server:

- `type`
- `task_id`
- `upload_name`
- `source_path`
- `file_size`
- `md5`

Below is the network traffic generated by the malware when uploading a test file via the Google Drive API (`/upload/drive/v3/files`).



Fiddler session showing upload to Google Drive

The below figure shows the console during this upload process.



Console output showing file upload success

Below is a screenshot of the previous demonstration using the file upload feature with our own Google Drive test account.



Below is the response from this handler:

```
{
  "cmd": 17,
  "data": {
    "file_id": "1qmP4TcGFfE2xbjYSlV-AVCRA96f6Kp-V7",
    "file_name": "meow.txt",
    "file_size": 16,
    "md5": "1e28c01387e0f0229a3fb3df931eaf80",
    "progress": 100,
    "status": "uploaded",
    "task_id": "124"
  },
  "id": 4079875446683087000,
  "output": "",
  "success": true
}
```

Handler #18 - Pause download/upload transfer

This handler allows the operator to pause any download and upload tasks managed by NANOREMOTE by passing the `task_id`.



Upload transfer paused



Upload transfer paused

Handler #19 - Resume download/upload transfer

This handler allows the operator to resume any paused download or upload tasks managed by NANOREMOTE using the task_id.



Upload transfer resumed

Handler #20 - Cancel file transfer

This handler allows the operator to cancel any download/upload tasks managed by NANOREMOTE through the task_id.



Cancelled transfer

Handler #21 - Command Execution

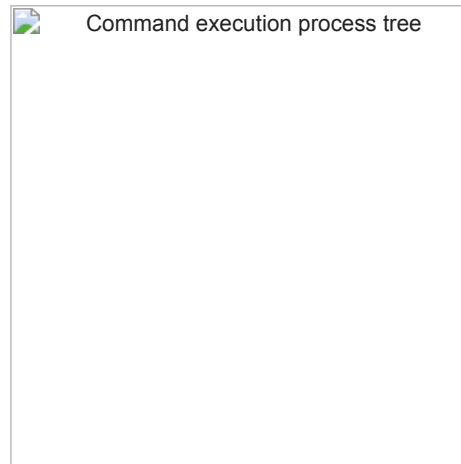
This is the main handler used by the adversary for command execution on the victim machine. It works by spawning new processes and returning the output through Windows pipes. This is a core feature found in most backdoors used by adversaries for direct access to enumerate the environment, perform lateral movement, and execute additional payloads.



Command execution

Command execution

The figure below shows NANOREMOTE's process tree when this handler is invoked. The malware spawns `cmd.exe`, which in turn launches the specified command—in this case, `whoami.exe`.



Command execution process tree

Handler #22 - Execute encoded PE from memory

This handler loads and executes a Base64 encoded PE file inside the existing NANOREMOTE process. The encoded PE file is provided by the C2 server using the `pe_data` field. If the program requires command-line arguments, the key `(arguments)` is used.

Below is an example showing the console output using test program:



Similarity to FinalDraft

There is overlap between FINALDRAFT and NANOREMOTE from both code similarity and behavioral perspectives.

Many functions exhibit clear code re-use across the two implants. For example, both follow the same sequence of generating a GUID via CoCreateGuid, hashing it with the [Fowler-Noll-Vo \(FNV\)](#) function and performing identical heap-validation checks before freeing the buffer.



A good portion of the HTTP-related code used to send and receive requests suggests similarity as well. Below is an example of a control flow-graph showing the setup/configuration of an HTTP request used by both malware families.



During our analysis, we observed that the WMLOADER decrypts the corresponding payload from a hard-coded file named wmsetup.log – the same file name that was used by PATHLOADER to deploy FINALDRAFT that we [published](#) earlier in the year.

Another interesting finding is that we discovered a [sample](#) (wmsetup.log) from VirusTotal that was recently uploaded from the Philippines on 2025-10-03.





VirusTotal file submission for wmsetup.log in October 2025

We downloaded the file, placed it alongside WMLOADER, then executed the loader. It successfully decrypted the `wmsetup.log` file, revealing a FINALDRAFT implant.



YARA showing FINALDRAFT from wmsetup.log

YARA showing FINALDRAFT from wmsetup.log

Below is a side-by-side graphic showing the same AES key is used to successfully decrypt both FINALDRAFT and NANOREMOTE.



Same AES used to decrypt both NANOREMOTE and FINALDRAFT

Our hypothesis is that WMLOADER uses the same hard-coded key due to being part of the same build/development process that allows it to work with various payloads. It's not clear why the threat group behind these implants are not rotating the key, it's possibly due to convenience or testing. This appears to be another strong signal suggesting a shared codebase and development environment between FINALDRAFT and NANOREMOTE.

NANOREMOTE through MITRE ATT&CK

Elastic uses the [MITRE ATT&CK](#) framework to document common tactics, techniques, and procedures that threats use against enterprise networks.

Tactics

Tactics represent the why of a technique or sub-technique. It is the adversary's tactical goal: the reason for performing an action.

- [Collection](#)
- [Command and Control](#)
- [Defense Evasion](#)
- [Discovery](#)
- [Execution](#)
- [Exfiltration](#)

Techniques

Techniques represent how an adversary achieves a tactical goal by performing an action.

- [Account Discovery](#)

- [Local Storage Discovery](#)
- [Exfiltration Over Web Service: Exfiltration to Cloud Storage](#)
- [Masquerading: Invalid Code Signature](#)
- [Command and Scripting Interpreter: Windows Command Shell](#)

Mitigating NANOREMOTE

Within a lab environment executing NANOREMOTE, there were many different alerts triggered using [Elastic Defend](#).



One of the main behaviors to validate for defenders is the abuse of using legitimate services such as Google Drive API. Below is an example alert triggered with the Connection to Commonly Abused Webservices rule when interacting with the Google API for both the download and upload of files using NANOREMOTE.





Connection to Commonly Abused Webservices

The PE loading technique using the Base64 encoded file from the C2 server was also detected via **Memory Threat Detection Alert: Shellcode Injection** alert.



Detection/Prevention

- [Potential Evasion with Hardware Breakpoints](#)
- [Potential Evasion via Invalid Code Signature](#)
- [Unbacked Shellcode from Unsigned Module](#)
- [Shellcode Execution from Low Reputation Module](#)
- [Image Hollow from Unusual Stack](#)
- [Connection to Commonly Abused Webservices](#)
- Memory Threat Detection Alert: Shellcode Injection

YARA

Elastic Security has created YARA rules to identify this activity.

```
rule Windows_Trojan_NanoRemote_7974c813 {
  meta:
    author = "Elastic Security"
    creation_date = "2025-11-17"
    last_modified = "2025-11-19"
    license = "Elastic License v2"
    os = "Windows"
    arch = "x86"
    threat_name = "Windows.Trojan.NanoRemote"

  strings:
    $str1 = "/drive/v3/files/%s?alt=media" ascii fullword
    $str2 = "08X-%04X-%04x-%02X%02X-%02X%02X%02X%02X%02X" ascii fullword
    $str3 = "NanoRemote/" wide
    $str4 = "[+] pwd output:" wide
    $str5 = "Download task %s failed: write error (wrote %llu/%zu bytes)"
    $seq1 = { 48 83 7C 24 28 00 74 ?? 4C 8D 4C 24 20 41 B8 40 00 00 00 BA 00 00 01 00 48 8B 4C 24 28 FF 15 ??
    ?? ?? ?? 85 C0 }
    $seq2 = { BF 06 00 00 00 89 78 48 8B 0D ?? ?? ?? ?? 89 48 ?? FF D3 89 78 78 8B 0D ?? ?? ?? ?? 89 48 7C FF
    D3 89 78 18 8B 0D }
    condition:
      4 of them
}

rule Windows_Trojan_WMLoader_d2c7b963 {
  meta:
    author = "Elastic Security"
    creation_date = "2025-12-03"
    last_modified = "2025-12-03"
    license = "Elastic License v2"
    os = "Windows"
    arch = "x86"
    threat_name = "Windows.Trojan.WMLoader"
    reference_sample = "fff31726d253458f2c29233d37ee4caf43c5252f58df76c0dced71c4014d6902"

  strings:
    $seq1 = { 8B 44 24 20 FF C0 89 44 24 20 81 7C 24 20 01 30 00 00 }
    $seq2 = { 41 B8 20 00 00 00 BA 01 30 00 00 48 8B 4C C4 50 FF 15 }
    condition:
      all of them
}
```

Observations

The following observables were discussed in this research.

Observable	Type	Name	Reference
fff31726d253458f2c29233d37ee4caf43c5252f58df76c0dced71c4014d6902	SHA-256	BDReinit.exe	WMLOADER
999648bd814ea5b1e97918366c6bd0f82b88f5675da1d4133257b9e6f4121475	SHA-256	ASDTool.exe	WMLOADER
35593a51ecc14e68181b2de8f82dde8c18f27f16fcebedbbdac78371ff4f8d41	SHA-256	mitm_install_tool.exe	WMLOADER
b26927ca4342a19e9314cf05ee9d9a4bddf7b848def2db941dd281d692eaa73c	SHA-256	BDReinit.exe	WMLOADER
57e0e560801687a8691c704f79da0c1dbdd0f7d5cc671a6ce07ec0040205d728	SHA-256	NANOREMOTE	

Jump to section

- [Introduction](#)
- [Key takeaways](#)
- [NANOREMOTE analysis](#)
- [WMLOADER](#)
- [NANOREMOTE](#)
- [NANOREMOTE Configuration](#)
- [Interface/Logging](#)
- [Setup](#)
- [Network Communication](#)
- [Command Handlers](#)

Elastic Security Labs Newsletter

[Sign Up](#)