

MoonWalk: A deep dive into the updated arsenal of APT41 | Part 2

Yin Hong Chang, Sudeep Singh :: 7/11/2024



Technical Analysis

Attack chain

The focus of this blog post is the second half of the attack chain that begins with the in-memory execution of MoonWalk backdoor. Once the MoonWalk backdoor is successfully loaded by DodgeBox, the malware decrypts and reflectively loads two embedded plugins (C2 and Utility). The C2 plugin uses a custom encrypted C2 protocol to communicate with the attacker-controlled Google Drive account.

A figure depicting the attack chain used to deploy MoonWalk with the DodgeBox loader is shown below.

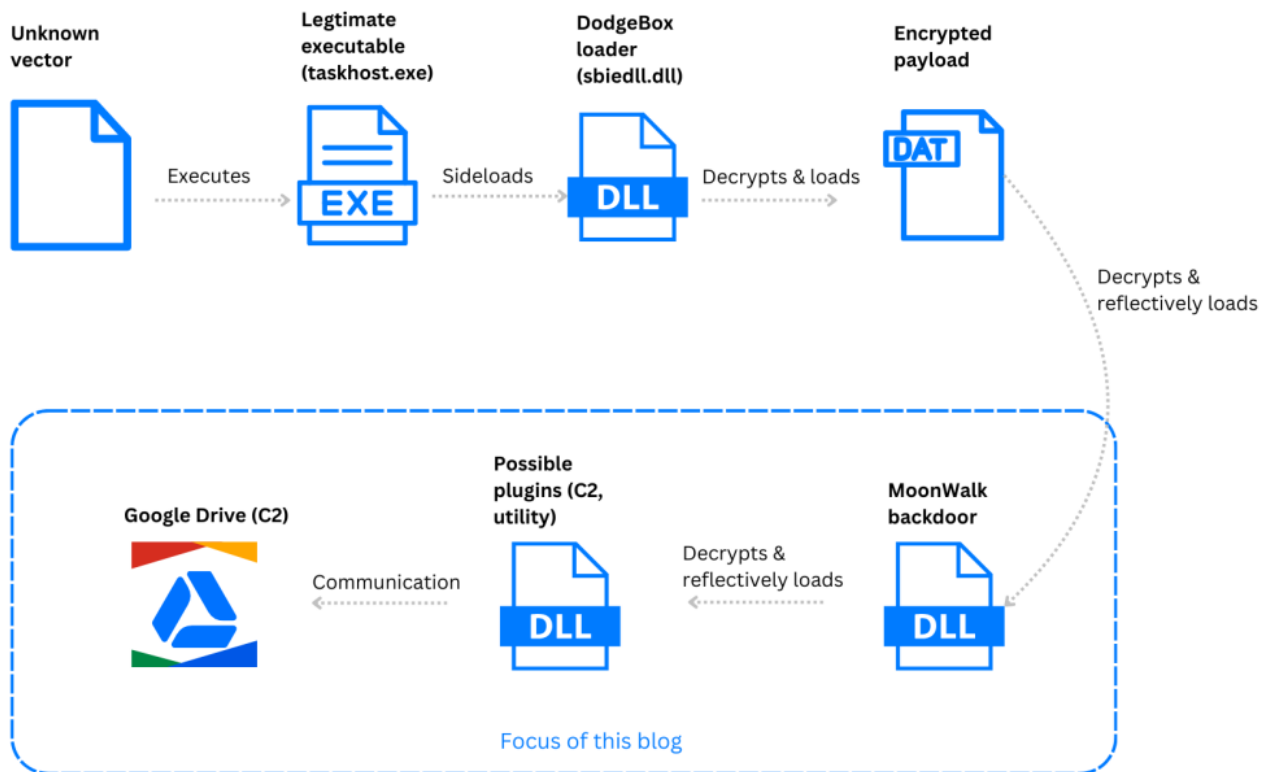


Figure 1: Attack chain used to deploy the DodgeBox loader and MoonWalk backdoor.

MoonWalk analysis

MoonWalk is a malware backdoor written in C that shares many code similarities with DodgeBox, suggesting a common development toolkit. It incorporates many evasion related functions from DodgeBox, including those related to the following:

- DLL hollowing
- Import resolution
- DLL unhooking
- Call stack spoofing

Additionally, MoonWalk utilizes the same DLL blacklist as DodgeBox.

ThreatLabz analysis reveals MoonWalk's modular design, allowing it to load different plugin components as needed. The sample examined by ThreatLabz contains two embedded plugins, a C2 plugin for C2 communication, and a utility plugin that provides functionality related to compression and public-key cryptography. This modular architecture makes MoonWalk highly adaptable, enabling attackers to customize its functionality for different scenarios.

In the section below, we will highlight several notable capabilities of MoonWalk.

Unloading the DodgeBox loader

When MoonWalk first initializes, it resolves its imports using the same algorithms as DodgeBox. Then, depending on the DodgeBox configuration parameter `Config.fShouldUnloadStealthVector`, MoonWalk unloads the DodgeBox DLL from memory and unlinks it from the Process Environment Block (PEB). This reduces MoonWalk's in-memory footprint, and obfuscates its origins, hindering memory forensic analysis.

Using Windows Fibers

Next, MoonWalk initializes global structures used to manage Windows Fibers. Windows Fibers are a lightweight threading mechanism, available in the Windows operating system since Windows NT SP5. Unlike traditional threads, which are scheduled by the operating system, fibers are cooperatively scheduled by the application itself. This allows developers to tune an application's performance for a specific workload. However, due to the complexity of utilizing Windows Fibers, and performance improvements of computer hardware, Windows Fibers were not widely adopted, and remains an obscure feature.

However, with the increased focus on cybersecurity in recent years, there has been an uptick in interest in Windows Fibers from the research and red-teaming community. Multiple research papers ([1](#), [2](#), [3](#)) and open-sourced [proof of concepts](#) (POCs) have been published, abusing Windows Fibers to evade AVs/EDR solutions.

APT41 may have been following these developments, as they have incorporated Windows Fibers into the MoonWalk backdoor. At a high level, MoonWalk maintains a global array of fibers. When a function needs to be executed as a fiber, a fiber is created using the `CreateFiber` API. This fiber is then packaged together with the address of the function and its arguments and other metadata, and inserted into the global array. The main fiber then schedules these fibers for execution. This use of Windows Fibers helps MoonWalk evade AVs and EDRs which do not support the scanning of Windows Fibers, and also makes analysis challenging by breaking up the control flow.

Configuration

MoonWalk decrypts its configuration, which is hard-coded within its `.lrsrc` section. Like DodgeBox, MoonWalk uses MD5 for configuration validation and AES Cipher Feedback (AES-CFB) for decryption.

However, MoonWalk's configuration is more complex, featuring nested structures and arrays. This configuration contains various execution parameters including the following:

- Mutex names
- A fiber configuration
- Heartbeat intervals
- Encryption keys
- C2-related data

In the sample we analyzed, MoonWalk's configuration (referred to as `Config`) included OAuth secrets used to authenticate with the attacker-controlled Google Drive account, and other notable fields as shown below:

Config.szClientID:

XXXXXXXX3108-0pm3bsjc0mto2e1k4kp2u8817lgk3e3v.apps.googleusercontent.com

Config.szClientSecret:

XXXXXXXXBiuo8VPZUH1dBHkv86mC1xFU_Z3

Config.szRefreshToken: XXXXXXXXiYDPmH9cCgYIARAAGAkSNwF-

L9IrcM7YiuxWrNuyIfKINyNc_pEVytGNNK750ZyyIm32qH6Wh3dGIBTvdPJ2v92xAohHwWw

Config.rgbXorKey:

a8e6bd132daf0360b1af1f5eea15e42f8c6f1dcd7d34376ae4e83a1a4f5907c0

Config.szMutexName:

Global\ctXjvsAxpzyqElmk

Config.szName:

default

After loading the default configuration, MoonWalk searches for a new configuration file at `C:\ProgramData\[MD5(Config.rgbIDBytes)]`. If found, the malware decrypts and loads this file. A sample of MoonWalk's decrypted configuration is available in the Appendix of this blog for reference.

Unpacking and loading plugins

MoonWalk then extracts embedded plugins from the `.lrsrc` section. In the MoonWalk sample we analyzed, there were two plugins embedded within this section: one plugin for C2, and another plugin which provides utility functions such as public key cryptography and compression.

Each plugin in the `.lrsrc` section is prefixed with 72 bytes of metadata, which includes AES-CFB secrets, an MD5 checksum, and plugin type information. The plugin type information fields provide information about the features of a plugin. These fields help identify whether a plugin serves as a command handler, C2, or utility. More details about the structure of plugin metadata can be found in the Appendix section.

MoonWalk organizes these plugins by registering them in a global linked list. MoonWalk then goes through this list to load the C2 plugin and its dependencies, such as the utility plugin, using DLL hollowing. This process is similar to what we previously described in [Part 1](#) for DodgeBox. Like DodgeBox, this MoonWalk sample stores a copy of the host DLL

in `C:\Windows\Microsoft.NET\assembly\GAC_MSIL\System.Data.Trace.`

Network Communication

After loading the C2 plugin, MoonWalk is prepared to establish communication with the C2 server. MoonWalk utilizes Google Drive for C2 communications. This helps MoonWalk evade detection, as traffic to and from reputable cloud services are less likely to raise suspicion, especially if a target is already using this service. Strangely, MoonWalk uses the string `curl/7.54.0` as its User-Agent when making HTTP requests, even though it does not utilize libcurl in its C2 plugin, and uses the WinHTTP family of APIs instead.

At a high level, MoonWalk communicates over Google Drive in the following manner:

Step	Description
1 - Initialization	<p>MoonWalk obtains an access token from the Google Authorization Server, by utilizing the OAuth secrets in its configuration (<code>Config.szClientID</code>, <code>Config.szClientSecret</code> and <code>Config.szRefreshToken</code>).</p> <p>MoonWalk generates 16 random bytes, and hex-encodes them, resulting in a string such as: <code>f137da1a9019849fbc2aac49a4b6f2c3</code>. We will reference this string as <code>SessionID</code>.</p> <p>MoonWalk uses the Google Drive APIs to retrieve the ID for the <code>/data</code> directory.</p> <p>MoonWalk retrieves the ID for the <code>/data/temp</code> directory.</p> <p>MoonWalk searches <code>/data/temp</code> for a file named after the generated <code>SessionID</code> (i.e. <code>f137da1a9019849fbc2aac49a4b6f2c3</code>). If the file is not found, MoonWalk generates and uploads a file <code>/data/temp/[SessionID]</code> to initiate a cryptographic handshake and exchange AES keys with the server.</p>
2 - Cryptographic Handshake (Client Hello and Server Hello)	<p>MoonWalk then looks for the <code>/data/[SessionID]</code> directory, and its subdirectory <code>/data/[SessionID]/s1</code>. The directory titled <code>s[number]</code> seems to serve as the designated location where MoonWalk will retrieve and download forthcoming C2 instructions.</p> <p>Lastly, MoonWalk searches for the <code>/data/[SessionID]/s1/1</code> file. As it becomes available, MoonWalk downloads and processes it, and completes the cryptographic handshake.</p>
3 - Information Gathering	<p>MoonWalk then checks for the existence of the directory <code>/data/[SessionID]/c1</code>, and creates it if it does not exist. Then, MoonWalk gathers information such as the computer name, user name, and OS version, and uploads this to the file <code>/data/[SessionID]/c1/1</code>.</p> <p>MoonWalk then proceeds to send heartbeats regularly to the C2 server by updating a file named <code>"temp.txt"</code> with the current Unix timestamp as a string.</p>
4 - Heartbeat	<p>MoonWalk also regularly polls the <code>/data/[SessionID]/s1</code> directory for new files. If a new file is found, MoonWalk processes it and uploads its response in the <code>/data/[SessionID]/c1</code> directory. During our analysis of MoonWalk, only ping commands were observed, where MoonWalk responded by uploading encoded files to the <code>/data/[SessionID]/c1</code> directory, containing the current Unix timestamp.</p>

Table 1: High-level view of the MoonWalk C2 communication protocol using Google Drive.

Cryptographic Handshake (Client Hello)

During the cryptographic handshake phase, MoonWalk exchanges AES keys with the server using a custom protocol. Because of this, it becomes very difficult or impossible to decode encrypted C2

messages without access to these AES keys, which exist only in MoonWalk's process memory.

The process begins with MoonWalk generating a 32-byte AES key (`rgbClientAESKey`) and a 16-byte initialization vector (IV) (`rgbClientAESIV`) using a custom random number generator. The AES key is then treated as an Elliptic-curve Diffie-Hellman (ECDH) private key, to generate the corresponding ECDH public key (`rgbECDHPublicKey`) using the `curve25519_donna` function.

MoonWalk then encodes the ECDH public key and AES IV by XORing them with the XOR key from MoonWalk's configuration (`Config.rgbXorKey`). A checksum is created by performing an MD5 hash on the concatenation of `Config.rgbXorKey`, ECDH public key, and AES IV, and then taking the hash's first four bytes. Finally, MoonWalk uploads this data to Google Drive at the path `/data/temp/[SessionID]`.

The figure below shows content of an uploaded file:

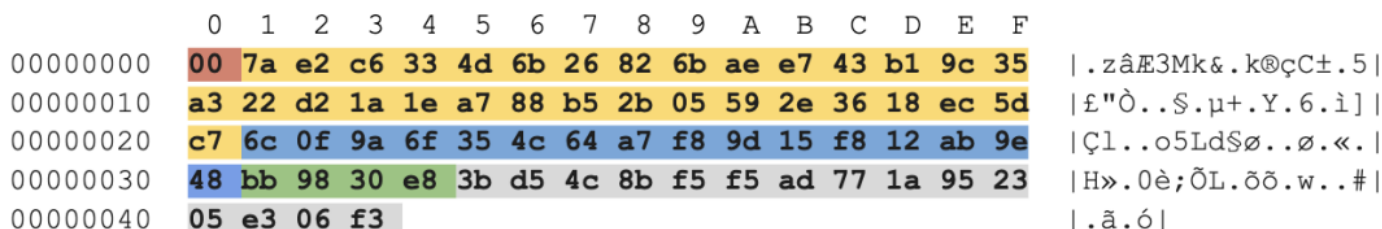


Figure 2: Contents of a MoonWalk Client Hello key exchange message.

The table below provides a description of the various fields contained within the uploaded file:

Offset	Size in bytes	Description
0x00	1	Unknown field, possibly a message type enum. <code>rgbECDHPublicKey XORed with Config.rgbXorKey</code>
0x01	32	<code>rgbECDHPublicKey</code> before the XOR operation is: d2 04 7b 20 60 c4 25 e2 da 01 f8 1d 5b 89 d1 8c ae bd 07 d3 da bc 82 41 e1 b1 14 2c 57 b5 5a 07 <code>rgbClientAESIV XORed with Config.rgbXorKey</code>
0x21	16	<code>rgbClientAESIV</code> before the XOR operation is: c4 e9 27 7c 18 e3 67 c7 49 32 0a a6 f8 be 7a 67
0x31	4	First four bytes of MD5 (<code>Config.rgbXorKey rgbECDHPublicKey rgbClientAESIV</code>)
0x35	15	Unknown bytes.

Table 2: Description of MoonWalk Client Hello key exchange message.

Cryptographic Handshake (Server Hello)

MoonWalk then downloads the file located at `/data/[SessionID]/s1/1`. This file contains the server's response to MoonWalk's handshake above.

This file, and all subsequent uploaded or downloaded files, are encoded using a custom scheme. Here, we walk through the decoding process of this scheme, using the Server Hello file as an example.

The figure below is an example of the overall layout of the encoded Server Hello file:

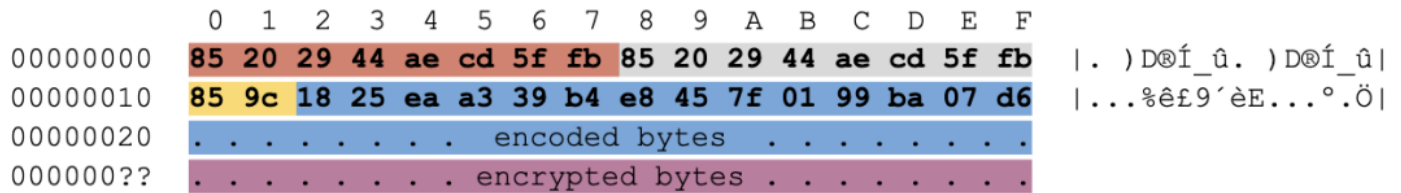


Figure 3: MoonWalk Server Hello message format.

A description of these fields is shown in the following table.

Offset	Size in bytes	Description
0x00	8	<code>rgbFileXorKey</code> The XOR key used to decode <code>rgbEncodedBytes</code> .
0x08	8	Unknown, potentially a message type field.
0x10	2	<code>dwNumEncodedBytes</code> The number of encoded bytes that follows. This field is encoded with <code>rgbFileXorKey</code> . Decoding this field shows that there are 0xbc encoded bytes within this file. $85\ 20\ \text{XOR}\ 85\ 9c = 00\ bc$ <code>rgbEncodedBytes</code>
0x12	<code>dwNumEncodedBytes</code>	The encoded bytes within this file. These bytes appear to contain message metadata, such as Google Drive file IDs, message headers, or junk bytes. To decode these bytes, <code>rgbFileXorKey</code> is used, starting with the third byte of the XOR key.

```

18 25 ea a3 39 b4 e8 45 7f 01 99 ba 07 d6
XOR
29 44 ae cd 5f fb 85 20 29 44 ae cd 5f fb
=
31 61 44 6e 66 4f 6d 65 56 45 37 77 58 2d

```


Offset	Size in bytes	Description
		rgbEncryptedBytes
0x??	variable	The rest of the file is not encoded, because this section is typically encrypted with AES-CFB, using the AES keys exchanged during the cryptographic handshake phase.

Table 3: Description of the MoonWalk Server Hello message format.

The figure below shows the Server Hello file after decoding:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	85	20	29	44	ae	cd	5f	fb	00	00	00	00	00	00	00	00
00000010	00	bc	31	61	44	6e	66	4f	6d	65	56	45	37	77	58	2d	.¼1aDnfOmeVE7wX-
00000020	42	63	77	7a	2d	6e	70	68	47	34	47	58	79	2d	69	64	Bcwz-nphG4GXy-id
00000030	57	68	69	00	7e	ab	88	27	ff	95	aa	cf	53	d2	67	eb	Whi.~«. 'ÿ.ªİSÒgë
00000040	eb	91	1e	33	c3	56	a6	4d	66	17	b8	23	a7	b0	d4	54	ë..3ÄV!Mf.,#S°ÔT
00000050	cf	2e	9f	00	61	36	b0	61	f5	ca	62	1b	ba	f0	d4	ab	İ...a6°aðÊb.°ðÔ«
00000060	87	0f	82	15	b8	1d	5f	31	ac	21	25	a5	2f	77	fa	b4_1¬!%¥/wú´
00000070	d2	c9	86	20	60	9f	32	bc	92	c4	a3	19	4f	17	5d	6f	ÒÉ. `²¼.Ä£.O.]o
00000080	77	c3	66	37	8c	a0	16	9f	8c	e6	7c	51	bb	1e	9d	1f	wÄf7. ...æ Q»...
00000090	05	00	9c	b1	67	5a	0a	ef	ff	60	3e	61	24	d1	a0	9d	...±gZ.İÿ`>a\$Ñ .
000000a0	c9	32	02	29	b7	8c	f8	d3	67	7a	a2	69	af	a1	2d	c8	É2.) .øÓgzçî¯;-È
000000b0	af	eb	88	eb	2a	d1	db	5d	3e	60	53	7b	83	5e	03	18	¯ë.ë*ÑÛ]>`S{.^..
000000c0	91	fc	e7	6f	3b	74	ee	54	f9	b2	c3	a6	3a	b6	df	64	.üçø;tîTù²Ä:;¶ßd
000000d0	d9	00	29	b9	97	ba	84	7d	01	e6	cd	c2	d1	d0	8e	e5	Û.)¹.°.}.æÍÄÑÐ.â
000000e0	5a	48	2b	75	1e	31	2f	d3	12	38	bd	30	3a	fa	ad	9b	ZH+u.1/Ó.8½0:ú..
000000f0	6f	1e	08	27	98	17	5e	67	fb	c1	33	44	d1	82	05	f5	o..'..^gûÁ3DÑ..õ
00000100	65	bc	83	3b	a5	6f	ab	58	3b	b7	49	7b	ed	cf	0a		e¼.;¥o«X;·I{íİ.



Figure 4: Example contents of a decoded MoonWalk Server Hello message.

The decoded Server Hello fields are described in the table below.

Offset	Size in bytes	Description
		rgbFileXorKey
0x00	8	The XOR key, used to decode rgbEncodedBytes.
0x08	8	Unknown
0x10	2	dwNumEncodedBytes
		szHeartBeatFileID
0x12	variable	The Google Drive ID of the heartbeat file, temp.txt.
0x34	variable	Unknown

Offset	Size in bytes	Description
		Encoded buffer, XOR encoded with <code>Config.rgbXorKey</code> .
		After decoding, the following fields are revealed:
0x0ce	48	<code>rgbServerECDHBasePoint</code> - Used as the ECDH base point, which MoonWalk later uses to generate the shared AES key used by the server. 77 82 64 13 04 16 94 da 35 d2 1e b8 27 d7 35 ff 02 8a 47 85 56 41 29 5b cb 3b 28 22 f2 69 3d 3a The remaining bytes after decoding contain a checksum, and additional unknown bytes.
0x0fe	4	Checksum generated by MD5 (<code>rgbServerECDHBasePoint Config.rgbXorKey</code> .)
0x102	variable	Unknown

Table 4: Description of fields within a MoonWalk Server Hello message.

With this information, MoonWalk generates a public key (`rgbECDHServerPublicKey`) using the `curve25519_donna` function. Then, `rgbECDHServerPublicKey` is XORed against `Config.rgbXorKey` to generate the server AES key.

```
Curve25519_Donna (
    a1->rgbECDHServerPublicKey,
    // Public Key (out):
    // 000001e6`246391ec b5 8f a7 ee 0b da d6 79-79 60 85 79 bf 32 ad 91
    // 000001e6`246391fc 24 a3 39 66 4c 4b 49 97-6c 71 92 d3 55 45 4b 3e
    a1->rgbClientAESKey,
    // Private Key:
    // 000001e6`2463920c 54 be fd a7 f4 0f 62 15-fb 22 9a 48 04 e3 6e 90
    // 000001e6`2463921c 85 4b b9 c7 f2 5f de 57-65 59 9c 90 18 04 d9 d1
    a1->rgbECDHServerBasepoint);
    // Basepoint:
    // 000001e6`24639251 77 82 64 13 04 16 94 da-35 d2 1e b8 27 d7 35 ff
    // 000001e6`24639261 02 8a 47 85 56 41 29 5b-cb 3b 28 22 f2 69 3d 3a

    rgbServerAESKey = rgbECDHServerPublicKey ^ Config.rgbXorKey
    // 1d 69 1a fd 26 75 d5 19-c8 cf 9a 27 55 27 49 be
    // a8 cc 24 ab 31 7f 7e fd-88 99 a8 c9 1a 1c 4c fe
```

In this manner, MoonWalk exchanges AES keys with its C2, and thus concludes the cryptographic handshake.

Information gathering

During this phase, MoonWalk collects information about the environment and uploads it to Google Drive. The gathered data includes details such as the processor architecture, Windows product type, version and build numbers, computer and usernames, as well as IP addresses. This information is then compressed using LZ4. A checksum is then added, using the 32-bit MurmurHash2 algorithm, with a customized mixing constant where r is set to 15, and with the initial seed set to `0x12345678`. These bytes are then encrypted using AES-CFB with the server's AES key, and packaged using the custom scheme detailed above, before being uploaded to Google Drive.

More details of the environment information collected are provided in the Appendix of this blog.

Heartbeat

MoonWalk also regularly sends heartbeats to the server. It uploads the current Unix timestamp in plain text to a `temp.txt` file on Google Drive, using the file ID `szHeartBeatFileID` retrieved as part of the cryptographic handshake.

Backdoor capabilities

In our analysis of MoonWalk, we did not observe the C2 sending any other commands or plugins. If a command handler plugin (`dwPluginTypePart2 == 1` described in the Appendix) is not found, MoonWalk defaults to a built-in list of handlers. These handlers contain functionality, which include the following:

- Collect environment information (similar to the information gathering step above)
- Steal token (token impersonation)
- Create token (log on to the Windows machine using given credentials)
- Download new configuration
- Execute command line commands

Note: This list is not complete as further analysis is required.