In-depth Technical Analysis of Colibri Loader Malware

By No items found.

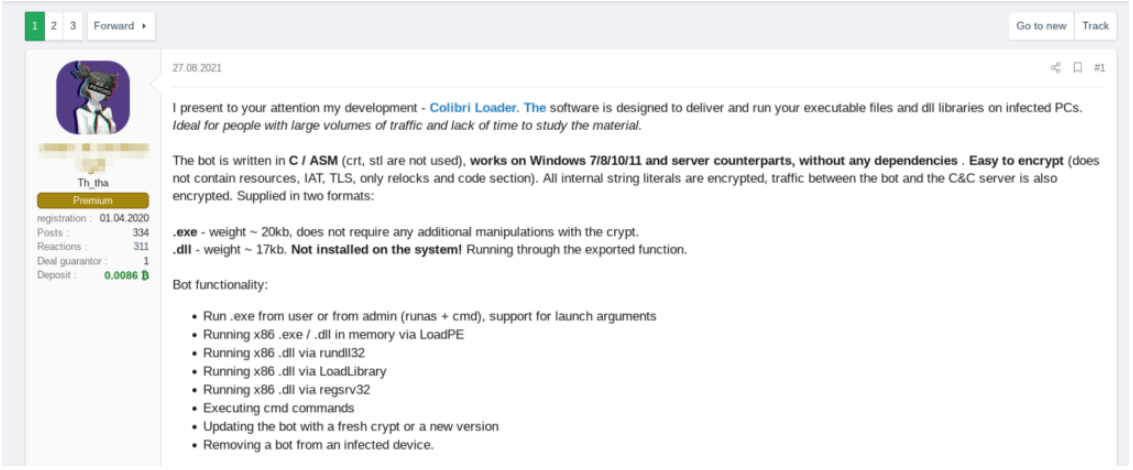
Published: 2025-08-21 · Archived: 2026-04-05 18:18:29 UTC

On 27 August 2021, cybersecurity researchers discovered a malware loader dubbed Colibri being sold on an underground Russian forum. The actors claim that the loader is stealthy and can be used to target Windows systems, to drop other malware onto the infected system.

Features of the Colibri loader malware

The features of the loader, as listed in the advertisement, include the following:

- The loader is written in C/ ASM.
- It works on Windows operating systems including Windows servers.
- The loader does not have dependencies, indicating that the loader works without relying on other entities from the system.
- The loader does not have an IAT (Import Address Table) that contains used WinAPI functions.
- Colibri loader has only two sections in the PE structure namely the ***".text"*** (code section) and the ***".reloc"*** (relocation section).
- All the strings in the loader as well as the connection channel between the C2 server and the loader are encrypted.



Threat actor's post on the cybercrime forum about the Colibri Malware Loader

Threat actor's post on the cybercrime forum

Technical Analysis of Colibri

Unpacking the loader

Colibri loader comes packed in a trojanized executable file. By using x64dbg (debugger) and putting breakpoints on the function *VirtualAlloc* we were able to extract the actual payload of the Colibri loader.

Packed sample	74c4f24e9c025d55c4dd8aca8b91fce3
Colibri unpacked sample	58FEE16BBEA42A378F4D87D0E8A6F9C8

The self-modifying code in the malware

By testing the extracted payload with PEStudio it is evident that the payload has only two sections, ***.text*** (Code Section) and ***.reloc*** (Relocation Section). The results of scanning the payload also show the existence of a self-modifying section in the code. This implies that the payload is capable of dynamically resolving other parts of the code that are not accessible through static analysis of the payload.

By running the payload in an IDA debugger we will be able to resolve the self-modifying code section of the payload.

property	value	value
name	.text	.reloc
md5	AC8635390715F3F3463E8F38...	4C726964B1A2C9FA5D3E8C...
entropy	6.182	4.398
file-ratio (70.83%)	66.67 %	4.17 %
raw-address	0x00001000	0x00005000
raw-size (17408 bytes)	0x00004000 (16384 bytes)	0x00000400 (1024 bytes)
virtual-address	0x036E1000	0x036E5000
virtual-size (15460 bytes)	0x00003A1C (14876 bytes)	0x00000248 (584 bytes)
entry-point	**0x0000460D**	-
characteristics	0xE0000020	0x42000040
writable	x	-
executable	x	-
shareable	-	-
discardable	-	x
initialized-data	-	x
uninitialized-data	-	-
unreadable	-	-
self-modifying	x	-
virtualized	-	-
file	n/a	n/a

Two sections that exist in the payload, besides the self-modifying property

```
.text:036E460D start:
.text:036E460D                 push    ebx
.text:036E460E                 push    esi
.text:036E460F                 push    edi
.text:036E4610                 jz      short near ptr loc_36E4614+1
.text:036E4612                 jnz     short near ptr loc_36E4614+1
.text:036E4614
.text:036E4614 loc_36E4614:                            ; CODE XREF: .text:036E4610↑j
.text:036E4614                                         ; .text:036E4612↑j
.text:036E4614                 mov     eax, 6DE8h
.text:036E4619                 add     [ebx+eax+75h], dh
.text:036E461D                 add     [eax-1CBE18h], edi
.text:036E4623                 jmp     fword ptr [eax-48h]
.text:036E4623 ; ---------------------------------------------------------------------------
.text:036E4626                 dw 0Bh
.text:036E4628                 dd 59026A00h, 309E8h, 0EDBA00h, 0C88B4F9Bh, 0FFD471E8h
.text:036E4628                 dd 0E8D0FFFFh, 0FFFFEF22h, 3B74C085h, 1750374h, 0DE7FE8B8h
.text:036E4628                 dd 374FFFFh, 0E8B80175h, 0FFFFF263h, 1750374h, 0F74FE8B8h
.text:036E4628                 dd 3068FFFFh, 6A000075h, 0C7E85902h, 0BA000002h, 4F9B00EDh
.text:036E4628                 dd 2FE8C88Bh, 0FFFFFFD4h, 5FD9EBD0h
.text:036E4684 ; ---------------------------------------------------------------------------
.text:036E4684                 pop     esi
.text:036E4685                 pop     ebx
.text:036E4686                 retn
```

The self-modifying code

```
.text:034B460D                 public start
.text:034B460D start:
.text:034B460D                 push    ebx
.text:034B460E                 push    esi
.text:034B460F                 push    edi
.text:034B4610                 jz      short loc_34B4615
.text:034B4612                 jnz     short loc_34B4615
.text:034B4612 ; ---------------------------------------------------------------------------
.text:034B4614                 db 0B8h ; .
.text:034B4615 ; ---------------------------------------------------------------------------
.text:034B4615
.text:034B4615 loc_34B4615:                            ; CODE XREF: .text:034B4610↑j
.text:034B4615                                         ; .text:034B4612↑j
.text:034B4615                 call    loadmodules
.text:034B461A                 jz      short loc_34B461F
.text:034B461C                 jnz     short loc_34B461F
.text:034B461C ; ---------------------------------------------------------------------------
.text:034B461E                 db 0B8h ; .
.text:034B461F ; ---------------------------------------------------------------------------
.text:034B461F
.text:034B461F loc_34B461F:                            ; CODE XREF: .text:034B461A↑j
.text:034B461F                                         ; .text:034B461C↑j
.text:034B461F                 call    createmutex
```

Dynamically resolved code section

IAT dynamic resolving

To avoid detection by AVs statically, the payload's author hashes all WinAPI functions, ignoring the Import Address Table (IAT), which aids in recognising the malware's activity statically. The payload resolves the function names dynamically using XOR and Shift operations. After resolving the function name, the address of the function is stored in eax register and a call function is created.

```
.text:034B4631                 mov     edx, 4F9B00EDh
.text:034B4636                 mov     ecx, eax
.text:034B4638                 call    getFunc
.text:034B463D                 call    eax             ; sleep
```

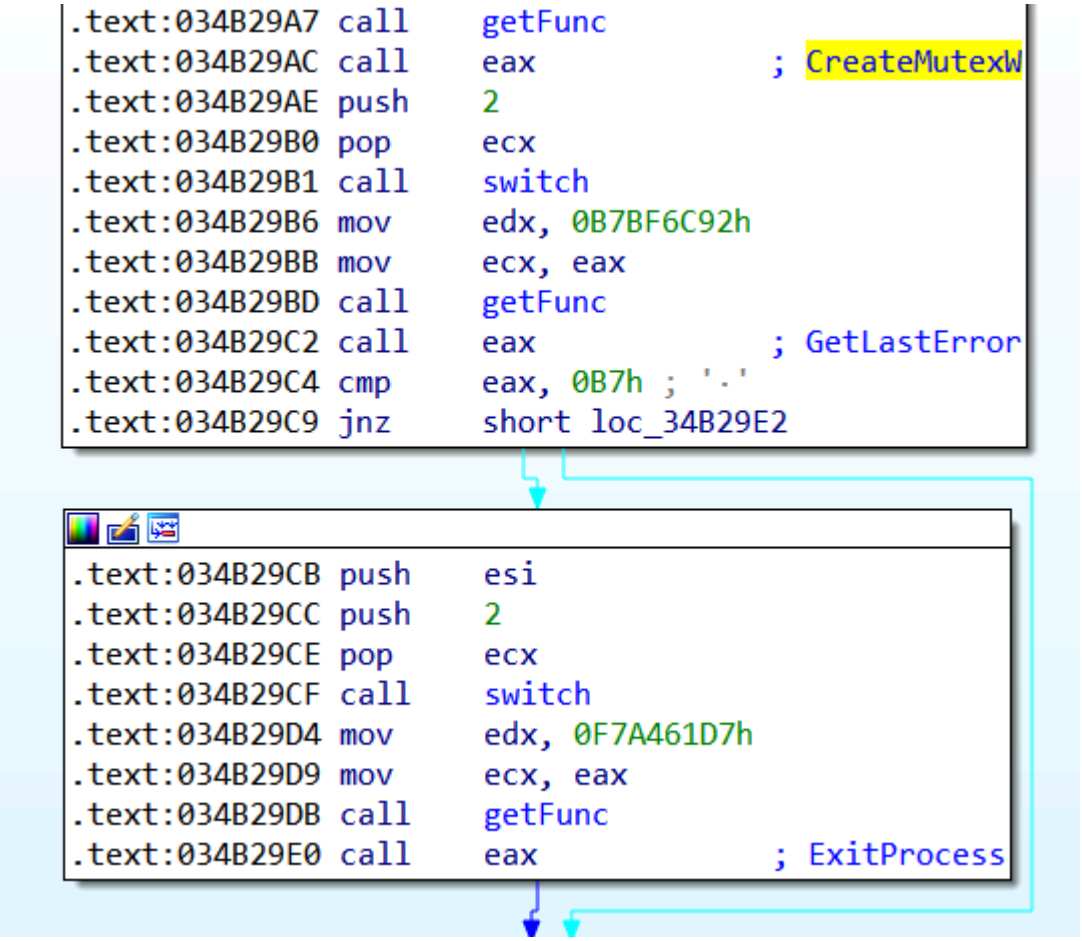
Dehashing the function name

```
v7 = pointertoFile + *(_DWORD *)(v4 + pointertoFile + 0x20);
v10 = v7;
v9 = pointertoFile + *(_DWORD *)(v4 + pointertoFile + 0x24);
if ( !v6 )
  return 0;
while ( dehashFunc((unsigned __int16 *)(pointertoFile + *(_DWORD *)(v7 + 4 * counter))) != hexvalue )
{
  v7 = v10;
  if ( ++counter >= v6 )
    return 0;
}
return pointertoFile + *(_DWORD *)(v6 + pointertoFile + 4 * *(unsigned __int16 *)(v9 + 2 * counter));
```

Dehashing the function names

Kill itself if there is already a running instance process of it

Before running on the system, the payload creates a mutex by calling the function CreateMutexW and then tests if there is an instance of the payload already running on the infected system. If there is an existing running process of payload on the system, the payload calls the ExitProcess function and exits the execution. If there is no instance of the payload running on the system, the payload continues the execution and calls the *Sleep* function to sleep for 3 seconds as a simple way to evade protection.



Check the existence of the payload on the system

The connection with the C2 server

To make the static analysis more difficult and to evade detection, the author of this malware has encrypted all the strings. After resolving the function names dynamically and using the debugger, the strings extracted from the payload will have the following artifacts:

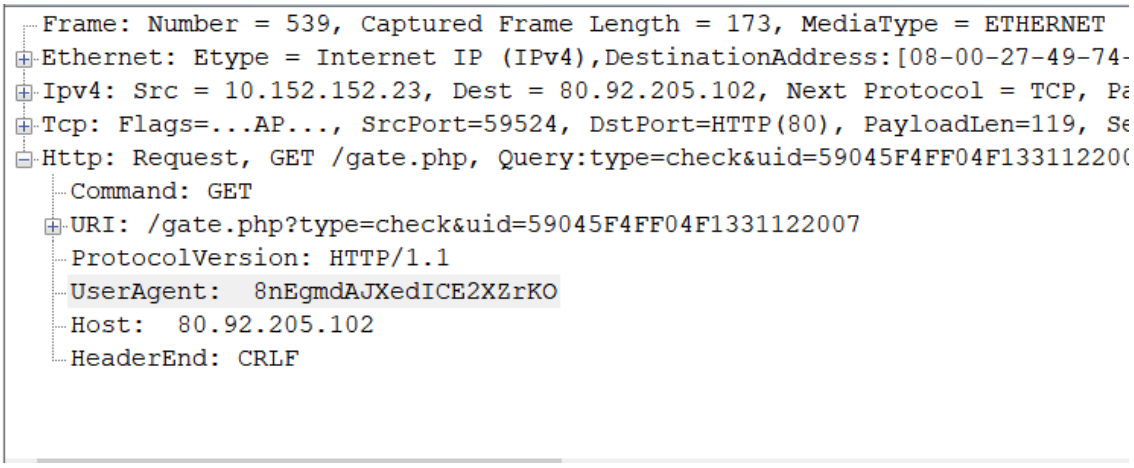
- IP address of the C2 server ***80.92.205.102***
- URL with another payload to be downloaded ***/gate.php***

The payload first initializes the use of WinInet functions by calling *InternetOpenW*, followed by which it opens the HTTP section with the function *InternetConnectW*. The payload creates an HTTP request by calling the function *HttpOpenRequestW* under the following parameters:

- The type of request: `/GET`
- Name of the target object: `/gate.php?type=check&uid=59045F4FF04F133112200`
- HTTP version to be used in the request: `HTTP/1.1`

After this, the payload sends the GET request to the server by calling the function HttpSendRequestW. Then, the payload calls the function `InternetQueryDataAvailable` to determine the amount of requested data. Based on the results of the previous function calls, the payload reads the data by calling the function `InternetReadFile`. Our assumption is that the payload requests the C2 server and downloads another payload on the system.

The payload calls the function `CryptStringToBinary`, to decrypt data after downloading the data from the C2 server, which indicates that the data could be encrypted.



GET request to the C2 server

Indicators of Compromise – Colibri Loader Malware

MD5	74c4f24e9c025d55c4dd8aca8b91fce3
58FEE16BBEA42A378F4D87D0E8A6F9C8	
IP	80.92.205.102
URL	80.92.205.102/gate.php? type=check&uid=59045F4FF04F133112200

Conclusion

Colibri loader is a type of malware that is used to load more types of malware into the infected system. This loader has multiple techniques that help avoid detection. This includes, omitting the IAT (Import Address Table) along with the encrypted strings to make the analysis more difficult. Like any other loader malware, the Colibri can be used to install information-stealing malware which may result in substantial loss of sensitive information. Thus, users should be wary of any unknown files on their systems.
