# The leap of a Cycldek-related threat actor

By Ivan Kwiatkowski

Published: 2021-04-05 · Archived: 2026-04-05 18:00:43 UTC

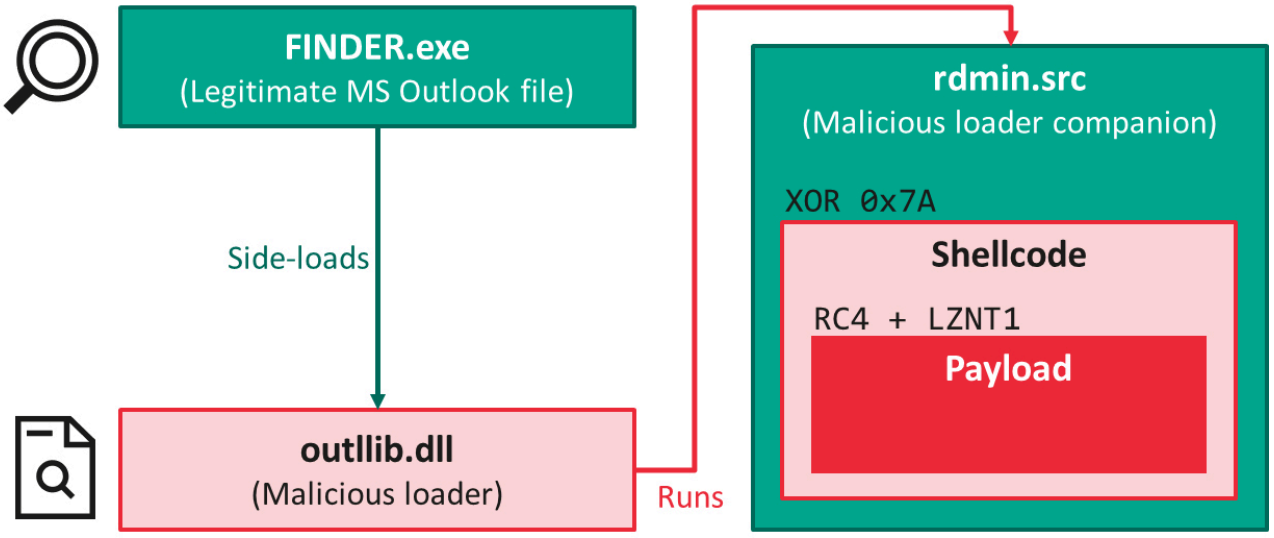
## Introduction

In the nebula of Chinese-speaking threat actors, it is quite common to see tools and methodologies being shared. One such example of this is the infamous “DLL side-loading triad”: a legitimate executable, a malicious DLL to be sideloaded by it, and an encoded payload, generally dropped from a self-extracting archive. Initially considered to be the signature of LuckyMouse, we observed other groups starting to use similar “triads” such as HoneyMyte. While it implies that it is not possible to attribute attacks based on this technique alone, it also follows that efficient detection of such triads reveals more and more malicious activity.

The investigation described in this article started with one such file which caught our attention due to the various improvements it brought to this well-known infection vector.

## FoundCore Loader

This malware sample was discovered in the context of an attack against a high-profile organization located in Vietnam. From a high-level perspective, the infection chain follows the expected execution flow:

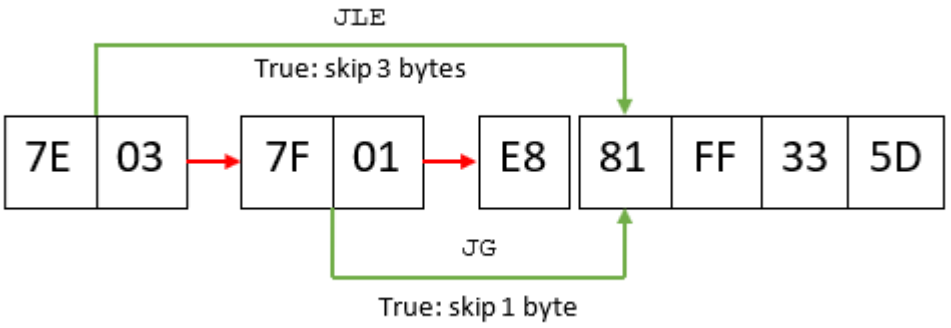



After being loaded by a legitimate component from Microsoft Outlook (FINDER.exe, MD5 9F1D6B2D45F1173215439BCC4B00B6E3), outlib.dll (MD5 F267B1D3B3E16BE366025B11176D2ECB) hijacks the intended execution flow of the program to decode and run a shellcode placed in a binary file, rdmin.src (MD5 DF46DA80909A6A641116CB90FA7B8258). Such shellcodes that we had seen so far, however, did not involve any form of obfuscation. So, it was a rather unpleasant surprise for us when we discovered the first instructions:

```
seg000:00000000 seg000          segment byte public 'CODE' use32
seg000:00000000                 assume cs:seg000
seg000:00000000                 assume es:nothing, ss:nothing, ds:nothing, fs:nothing, gs:nothing
seg000:00000000                 sub     ebx, 0A398A345h
seg000:00000006                 inc     edi
seg000:00000007                 jle     short near ptr loc_B+1
seg000:00000009                 jg      short near ptr loc_B+1
seg000:0000000B
seg000:0000000B loc_B:                                  ; CODE XREF: seg000:00000007↑j
seg000:0000000B                                         ; seg000:00000009↑j
seg000:0000000B                 call    near ptr 5D33FF91h
seg000:00000010
seg000:00000010 loc_10:                                 ; CODE XREF: seg000:00000022↓j
seg000:00000010                 adc     eax, 33FF81E8h
seg000:00000015                 pop     ebp
seg000:00000016                 adc     eax, 21F781E8h
```

Experienced reverse-engineers will immediately recognize disassembler-desynchronizing constructs in the screenshot above. The conditional jumps placed at offsets 7 and 9 appear to land in the middle of an address (as evidenced by the label loc_B+1), which is highly atypical for well-behaved assembly code. Immediately after, we note the presence of a call instruction whose destination (highlighted in red) is identified as bogus by IDA Pro, and the code that follows doesn't make any sense.

Explaining what is going on requires taking a step back and providing a bit of background about how disassemblers work. At the risk of oversimplifying, flow-oriented disassemblers make a number of assumptions when processing files. One of them is that, when they encounter a conditional jump, they start disassembling the "false" branch first, and come back to the "true" branch later on. This process is better evidenced by looking at the opcodes corresponding to the code displayed above, again starting from offset 7:

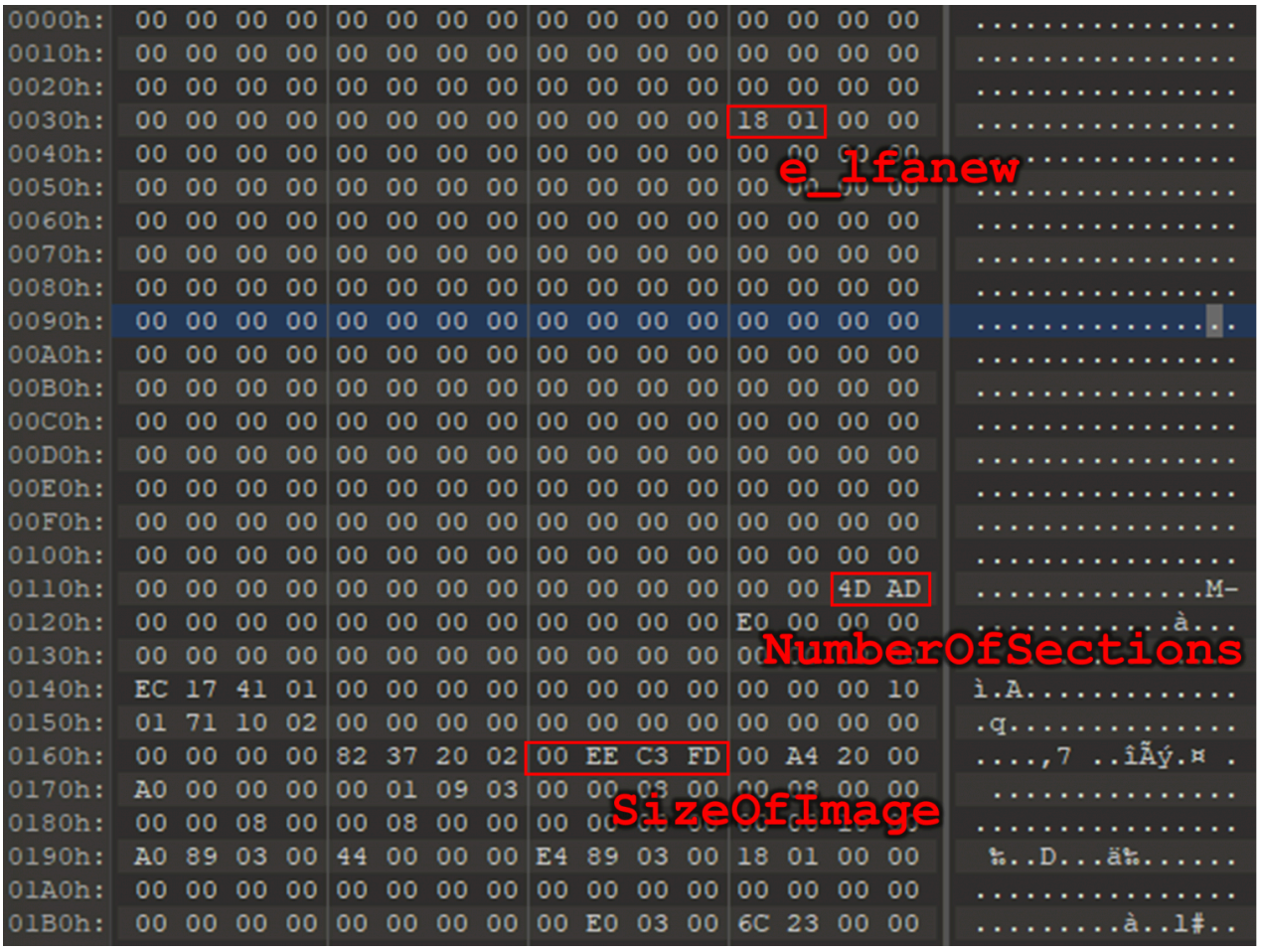



It is now more obvious that there are two ways to interpret the code above: the disassembler can either start from "E8", or from "81" – by default, IDA will choose the latter: E8 is in fact the opcode for the call instruction. But astute readers will notice that "JLE" (jump if lower or equal) and "JG" (jump if greater) are opposite conditions: no matter what, one of those will always be true and as such the actual code, as seen by the CPU during the execution, will start with the byte "81". Such constructs are called opaque predicates, and this E8 byte in the middle was only added there in order to trick the disassembler.

Defeating this trick is but a trivial matter for IDA Pro, as it is possible to manually correct the disassembling mistake. However, it was immediately obvious that the shellcode had been processed by an automated obfuscation tool. Opaque predicates, sometimes in multiples, and dead code were inserted between every single instruction of

the program. In the end, cleaning up the program automatically was the only practical approach, and we did so by modifying an existing script for the FinSpy malware family created by the respected reverse-engineer Rolf Rolles.

This step allowed us to discover the shellcode's purpose: to decrypt and decompress the final payload, using a combination of RC4 and LZNT1. Even then, it turned out that the attackers had more tricks up their sleeve. Normally, at this stage, one would have expected to find a PE file that the shellcode would load into memory. But instead, this is what we got:




The recovered file was indeed a PE, but it turned out that most of its headers had been scrubbed. In fact, even the scarce ones remaining contained incoherent values – for instance, here, a number of declared sections equal to 0xAD4D. Since it is the shellcode (and not the Windows loader) that prepares this file for execution, it doesn't matter that some information, such as the magic numbers, is missing. As for the erroneous values, it turned out that the shellcode was fixing them on the fly using hardcoded operations:

```
for ( i = 0; ; ++i ) // Iterate on the sections
{
 // [...]
 // Stop when all sections have been read
```

```
 if ( i >= pe->pe_header_addr->FileHeader.NumberOfSections - 44361 )
  break;
 // [...]
}
```

For instance, in the decompiled code above (as for all references to the file's number of sections) the value read in the headers is subtracted by 44361. For the attackers, the advantage is two-fold. First, it makes acquiring the final payload statically a lot more difficult for potential reverse-engineers. Second, it also ensures that the various components of the toolchain remain tightly coupled to each other. If only a single one of them finds itself uploaded to a multi-scanner website, it will be unexploitable for defenders. This is a design philosophy that we had observed from the LuckyMouse APT in the past, and is manifest in other parts of this toolchain too, as we will see later on. Eventually, we were able to reconstruct the file's headers and move on with our analysis – but we found this loader so interesting from an educational standpoint that we decided to base one track of our online reverse-engineering course on it. For more detailed steps on how we approached this sample, please have a look at Targeted Malware Reverse Engineering.

## FoundCore payload

The final payload is a remote administration tool that provides full control over the victim machine to its operators. Upon execution, this malware starts 4 threads:

- The first one establishes persistence by creating a service.
- The second one sets inconspicuous information for the service by changing its "Description", "ImagePath", "DisplayName" fields (among others).
- The third sets an empty DACL (corresponding to the SDDL string "D:P") to the image associated to the current process in order to prevent access to the underlying malicious file.
- Finally, a worker thread bootstraps execution and establishes connection with the C2 server. Depending on its configuration, it may also inject a copy of itself to another process.

Communications with the server can take place either over raw TCP sockets encrypted with RC4, or via HTTPS. Commands supported by FoundCore include filesystem manipulation, process manipulation, screenshot captures and arbitrary command execution.

## RoyalRoad documents, DropPhone and CoreLoader

Taking a step back from the FoundCore malware family, we looked into the various victims we were able to identify to try to gather information about the infection process. In the vast majority of the incidents we discovered, it turned out that FoundCore executions were preceded by the opening of a malicious RTF documents downloaded from static.phongay[.]com. They all were generated using RoyalRoad and attempt to exploit CVE-2018-0802.

Interestingly, while we would have expected them to contain decoy content, all of them were blank. We, therefore, hypothesize the existence of precursor documents, possibly delivered through spear-phishing, or precursor infections, which would trigger the download of one of these RTF files.

Successful exploitation leads to the deployment of yet another malware that we named DropPhone:

| | |
|---|---|
| **MD5** | 6E36369BF89916ABA49ECA3AF59D38C6 |
| **SHA1** | C477B50AE66E7228164930117A7D36C53713A5F2 |
| **SHA256** | F50AE4B25B891E95B57BD4391AEB629437A43664034630D593EB9846CADC9266 |
| **Creation time** | 2020-11-04 09:14:22 |
| **File type** | PE32 executable (DLL) (GUI) Intel 80386, for MS Windows |
| **File size** | 56 KB |

This C++ implant also comes in the form of a legitimate executable (DeElevate.exe, from the publisher StarDock) and a side-loaded DLL (DeElevator.dll). At this stage, we are left with more questions than answers when it comes to it. DropPhone fetches a file saved as data.dat from hxxps://cloud.cutepaty[.]com, but we were unable to obtain a copy of this file so far. Next, it expects to find a companion program in %AppData%\Microsoft\Installers\sdclt.exe, and will eventually terminate execution if it cannot find it.

Our hypothesis is that this last file could be an instance or variant of CoreLoader (which we will describe in a minute), but the only piece of data supporting this theory that we have at our disposal is that we found CoreLoader in this folder in a single occurrence.

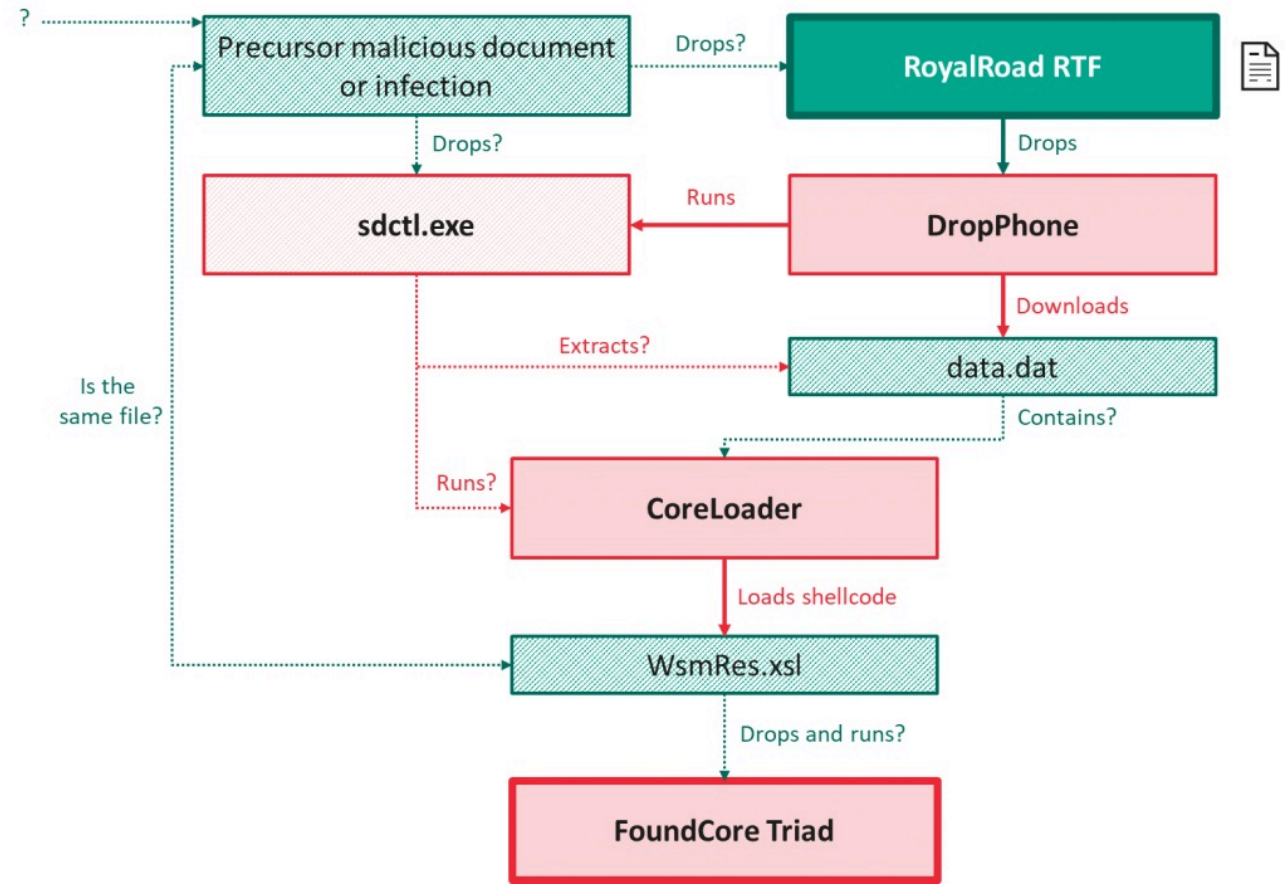
DropPhone launches sdclt.exe, then collects environment information from the victim machine and sends it to DropBox. The last thing this implant does is delete data.dat without ever accessing its contents. We speculate that they are consumed by sdclt.exe, and that this is another way to lock together the execution of two components, frustrating the efforts of the reverse-engineers who are missing pieces of the puzzle – as is our case here.

| | |
|---|---|
| **MD5** | 1234A7AACAE14BDD94EEE6F44F7F4356 |
| **SHA1** | 34977E351C9D0E9155C6E016669A4F085B462762 |
| **SHA256** | 492D3B5BEB89C1ABF88FF866D200568E9CAD7BB299700AA29AB9004C32C7C805 |
| **Creation time** | 2020-11-21 03:47:14 |
| **File type** | PE32 executable (DLL) (GUI) Intel 80386, for MS Windows |
| **File size** | 66 KB |

Finally, CoreLoader, the last malware we found associated to this set of activity, is a simple shellcode loader which performs anti-analysis and loads additional code from a file named WsmRes.xsl. Again, this specific file

eluded our attempts to catch it but we suspect it to be, one way or another, related to FoundCore (described in the previous section).

Overall, our current understanding of this complex toolchain is as follows. Dashed lines represent the components and links we are inferring, striped boxes represent the files we could not acquire.




## Victimology and attribution

We observed this campaign between June 2020 and January 2021. According to our telemetry, dozens of organizations were affected. 80% of them are based in Vietnam and belong to the government or military sector, or are otherwise related to the health, diplomacy, education or political verticals. We also identified occasional targets in Central Asia and in Thailand.

For the reasons laid-out in the introduction, attribution based on tooling alone is risky when it comes to this nebula. At first glance, the use of a “triad”, the general design philosophy and the obvious effort spent to make reverse-engineering as complex as possible are reminiscent of LuckyMouse. However, we also observed code similarities between CoreLoader or FoundCore and programs associated with the Cycldek threat actor – namely, RedCore Loader (MD5: 1B6BCBB38921CAF347DF0A21955771A6).

While Cycldek was, so far, considered to be one of the lesser sophisticated threat actors from the Chinese-speaking nexus, its targeting is known to be consistent with what we observed in this campaign. Therefore, we are linking the activities described in this post with Cycldek with low confidence.

## Conclusion

No matter which group orchestrated this campaign, it constitutes a significant step up in terms of sophistication. The toolchain presented here was willfully split into a series of interdependent components that function together as a whole. Single pieces are difficult – sometimes impossible – to analyze in isolation, because they rely on code or data provided at other stages of the infection chain. We regretfully admit that this strategy was partly successful in preventing us from obtaining a complete picture of this campaign. As such, this report is as much about the things we know as it is about figuring out what we don't. We hereby extend our hand to fellow researchers who might be seeing other pieces of this vast puzzle, because we strongly believe that the challenges ahead of us can only be overcome through information sharing among trusted industry partners.

Some readers from other regions of the world might dismiss this local activity as irrelevant to their interests. We would advise them to take heed. Experience shows that regional threat actors sometimes widen their area of activity as their operational capabilities increase, and that tactics or tools are vastly shared across distinct actors or intrusion-sets that target different regions. Today, we see a group focused on South-East Asia taking a major leap forward. Tomorrow, they may decide they're ready to take on the whole world.

## Indicators of Compromise

### File Hashes

### Domains

---

Source: https://securelist.com/the-leap-of-a-cycldek-related-threat-actor/101243/