SentinelLABS

SANCTIONS BE DAMNED | FROM DRIDEX TO MACAW, THE EVOLUTION OF EVIL CORP

Author: Antonio Pirozzi, Antonis Terefos and Idan Weizman

February 2022

SentinelLABS Research Team

TABLE OF CONTENTS

3 EXECUTIVE SUMMMARY

4 BACKGROUND

6 THE EVIL CORP MALWARE LINEAGE

28 OTHER TOOLSET EXPANSION

29 MACAW LOCKER RANSOMWARE

34 CRYPTONE: THE PACKER

44 INFRASTRUCTURE OVERLAPS

46 CONCLUSIONS

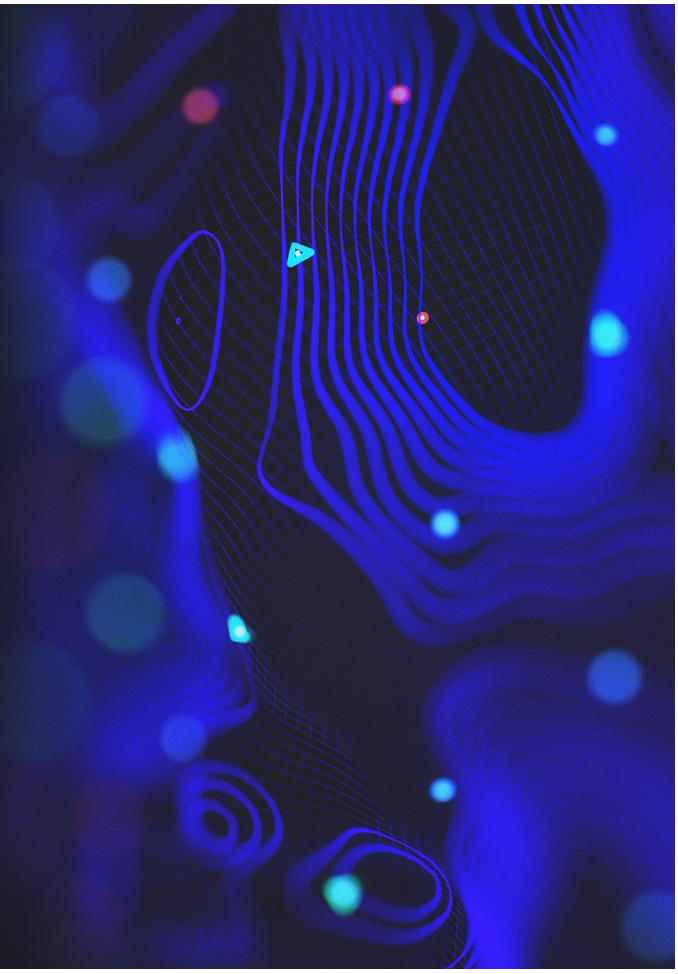
48 MITRE ATT&CK TTPS OBSERVED

52 YARA RULES

53 INDICATORS OF COMPROMISE [IOCS]

60 APPENDIX

63 ABOUT SENTINELLABS



EXECUTIVE SUMMARY

- SentinelLabs assesses with high confidence that WastedLocker, Hades, Phoenix Locker, PayloadBIN belong to the same cluster. There are strong overlaps in terms of code similarities, packers, TTPs and configurations.
- SentinelLabs assesses with high confidence that the Macaw ransomware variant is derived from the same codebase as Hades.
- A portion of SocGholish infrastructure, controlled by EC, and used to deliver WastedLocker and Hades Ransomware, was also used to spread BitPaymer and DoppelPaymer by deploying Dridex[1] loader.
- Since late 2019, Evil Corp has been evolving their tradecraft in order to continue to pivot around OFAC sanctions.
- CryptOne: throughout our analysis we noticed the unique use of a crypter tool dubbed "CryptOne". We were able to identify and map all of the CryptOne 'flavours' used in different time-frames. This supported our clustering of EC activity, through the correlation of those signatures.

This report summarizes an in-depth analysis of Evil Corp ("EC", a.k.a "Gold Drake", a.k.a "Indrik Spider"), an advanced cybercrime activity cluster originating from Russia, performed by SentinelLabs. As a result of this research and previous intelligence, we were able to cluster a range of EC's malware lineage and their activities. This investigation raises new insights about the group's modus operandi and toolsets.

Please note: *While EC is a name of a cyber crime gang, run by several known individual members, this report refers to the activity as it is seen through TTPs. We believe that this approach is more accurate to describe and track their operations. A list of IOC's and YARA rules can be found at the end of this report.*

SentinelLabs Team

[1]https://www.fireeye.com/blog/threat-research/2019/10/head-fake-tackling-disruptive-ransomware-attacks.html

BACKGROUND

Evil Corp ("EC", a.k.a "Gold Drake", a.k.a "Indrik Spider"), is an advanced cybercrime operations cluster originating from Russia and has been active since 2007. The UK National Crime Agency calls it "the world's most harmful cyber crime group."[2]

The origins of Evil Corp can be traced back to Evgeniy Bogachev, one of the leading figures of the group. Evgeniy Bogachev (also known as Slavik) developed the Zeus banking trojan[3]. Bogachev considered to be the leader of the "Business Club", a group of half-dozen people supported by a network of more than 50 individuals tied with the Russian Government. An award of 3 million dollars was announced for information that would lead to his capture. In 2011, the Zeus source code was leaked and different versions appeared in the wild. In September 2011, a new Zeus variant named Gameover Zeus (also known as Zeus Game Over, GoZ) was identified. It added a peer-to-peer C2 architecture that was more resilient to takedown, with the intention to mislead law enforcement. By mid 2014, GoZ counted over 27 different botnets, one million compromised computers and was responsible for the theft of tens of millions of dollars. Slavik was the administrator of the GoZ Botnet, and had the support of the Business Club crew. This support from the Business Club crew introduced Slavik to Maksim Yakubets. This partnership would come to form the backbone of what is now EC.

Maksim Yakubets (aka "aqua") is believed to have acted as Evil Corp group's leader (the last official document which describe him as a leader is from 5 Dec 2019[4]). The group was one of the pioneers of financial cybercrime, having operated one of the most aggressive and successful Banking trojans since 2014, Dridex. Evil Corp has earned millions of dollars in illicit profit from banks and financial institutions in over 40 countries.

In August 2017, EC's began to focus on more high-return targets with BitPaymer Ransomware, delivering the ransomware with Dridex mainly against North American and Western European organizations. In June 2019, the cluster DoppelSpider[5], considered a distinct group[6] (or a sub-cluster of EC), started operating a new ransomware dubbed DoppelPaymer. It was a new strain derived from BitPaymer's codebase and operated as a RaaS. It presented an important similarity with its predecessor but also implemented new features like multithreaded file encryption. DoppelPaymer was operated in parallel with BitPaymer; supporting the assessment that it was a distinct cluster.

---

[2]https://twitter.com/NCA_UK/status/1202618928209498114?s=20

[3]https://www.fbi.gov/wanted/cyber/evgeniy-mikhailovich-bogachev

[4]https://home.treasury.gov/news/press-releases/sm845

[5]https://adversary.crowdstrike.com/en-US/adversary/doppel-spider/

[6]https://www.crowdstrike.com/blog/doppelpaymer-ransomware-and-dridex-2/

OFAC SANCTIONS AND INDICTMENTS: A CHANGE IN TTPS

In December 2019, the U.S. Treasury Department's Office of Foreign Assets Control (OFAC)[7] issued a sanction against 17 individuals and seven entities related to EC cyber operations for causing financial losses of more than 100 million dollars with Dridex. In addition to the OFAC sanction, the DOJ charged two key members of the group: Maksim Yakubets and Igor Turashe; both members of the "Business Club" and considered the developers of the Dridex banking trojan. Yakubets was also mentioned as the leader of the group. In May 2020, following the OFAC which put Dridex under the scope of the sanction, a new ransomware variant appeared in the wild dubbed WastedLocker. It employs techniques to obfuscate its code and perform tasks similar to those already seen in BitPaymer and Dridex. Those similarities allowed the threat intelligence community to identify the connections between the malware families[8].

Right after the indictment we witnessed a change in some of their TTPs: from 2020, they started to frequently change their payload signatures, using different exploitation tools and methods of initial access. They switched from Dridex to the SocGholish framework to confuse attribution and distance themselves from both Dridex and Bitpaymer, which fell within the scope of the sanctions. During this period they started using Cobalt Strike extensively to gain an initial foothold and perform lateral movement instead of PowerShell Empire.

At this point, after the indictments, the global intelligence community was split into different "camps'' as to how EC was operating. Some assessed that there was a voluntary transition of EC operations to another 'trusted' partner while remaining the controller of operations. Some had theories that they stopped operating and another advanced actor operated Hades trying to mimic the same MO as EC to mislead attribution. Some claim[9] possible attribution to the HAFNIUM activity cluster. This report argues that the original operators continue to be active despite the sanctions, continuously changing their TTPs in order to stay under the radar.

In December 2020, a new ransomware variant named Hades was first seen in the wild and publicly reported[10]. Hades is a 64-bit compiled version of WastedLocker that displays important code and functionality overlaps[11]. In March 2021, a new variant Phoenix Locker appeared in the wild, which seemed to be a rebranded version of Hades with little to no changes[12]. Later, a new variant named PayloadBIN appeared in the wild[13], a continuation from Phoenix Locker. Another distinctive MO worth mentioning which characterizes all their ransomware operations is related to the extortion phase. Security researchers from several private sector firms note that historically there was no evidence of data theft during BitPaymer and WastedLocker incidents[14]. In addition to this, they didn't operate any data leak site.

---

[7]https://home.treasury.gov/news/press-releases/sm845

[8]https://news.sophos.com/en-us/2020/08/04/wastedlocker-techniques-point-to-a-familiar-heritage/

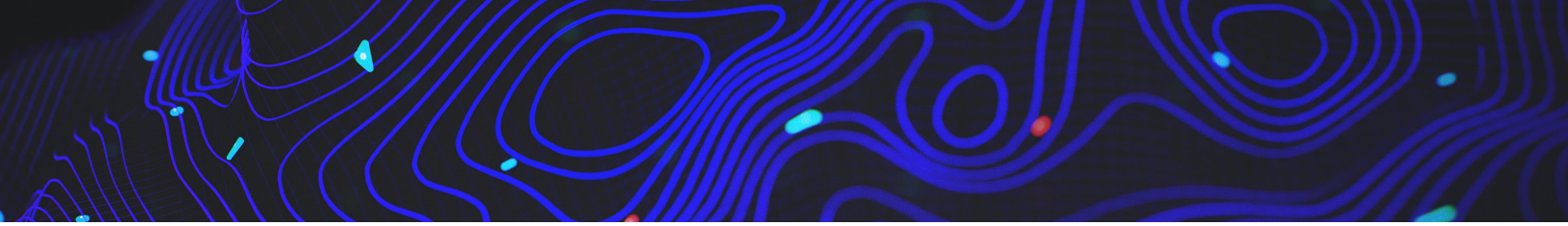
[9]https://awakesecurity.com/blog/incident-response-hades-ransomware-gang-or-hafnium/

[10]https://twitter.com/demonslay335/status/1339324224029274118?s=20

[11]https://www.crowdstrike.com/blog/hades-ransomware-successor-to-indrik-spiders-wastedlocker/

[12]section Hades vs. Phoenix Locker: A polymorphic code of this report

[13]https://twitter.com/fwosar/status/1401110845820747797?s=20



THE EVIL CORP MALWARE LINEAGE

EVIL CORP AFFILIATION MODEL

Dridex first appeared on the scene in June 2014. It was derived from Bugat V5. EC operates Dridex with an affiliation model, where each affiliate maintains control of a subset of bots while EC controls all the C2 backends.

Dridex has also been delivered by Emotet since 2017[15]. This suggests that there is a functional relationship between the two groups (they share resources). Moreover, research by Trend Micro highlighted that Emotet, Gozi IFSB and Dridex share the same loader[16]. Following the distribution of Dridex v4 binaries (botnet IDs 199 and 501) a new update in the TTPs has been observed, in fact they began using Dridex to execute PowerShell Empire and the Koadic post-exploitation framework[17].

Dridex was also the primary delivery method for BitPaymer Ransomware. Following the OFAC sanctions, in March 2020, WastedLocker was delivered exclusively through SocGholish. Ever since, SocGholish has been associated with the delivery of other malware that SentinelLabs attributes to EC activity, suggesting that SocGholish is a solid indicator of EC's recent activity.

In the following diagram we draw connections or affiliations between Evil Corp and other actors. The dark blue rectangles represent threat actors, the orange are related to malware families and loaders while the red ones are related to ransomware. The nature of the relationship is annotated on the arrow.

[14]https://www.zdnet.com/article/hacker-gang-behind-garmin-attack-doesnt-have-a-history-of-stealing-user-data/?&web_view=true

[15]https://nakedsecurity.sophos.com/2017/08/10/watch-out-for-emotet-the-trojan-thats-nearly-a-worm/

[16]https://www.trendmicro.com/it_it/research/18/l/ursnif-emotet-dridex-and-bitpaymer-gangs-linked-by-a-similar-loader.html

[17]https://www.fireeye.com/blog/threat-research/2019/10/head-fake-tackling-disruptive-ransomware-attacks.html

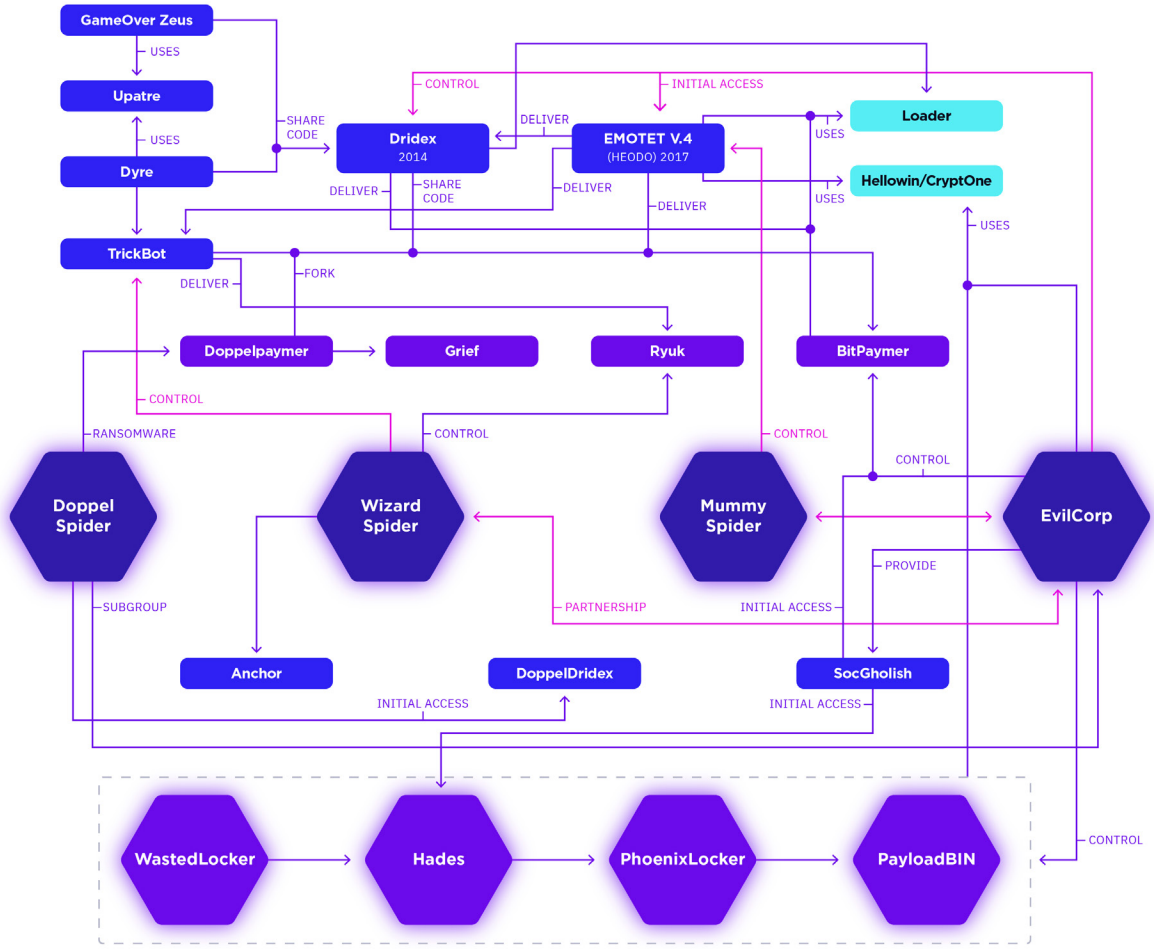


Fig 1: The EvilCorp affiliation diagram

A UNIQUE CLUSTER: BITPAYMER, WASTEDLOCKER, HADES, PHOENIX LOCKER, PAYLOADBIN

Clustering activity is the hardest part of tracking cybercrime, previous analysis of cybercrime activities revealed that often there are different kinds of affiliations between threat actors with varying degrees of trust. To cluster activities with confidence, it is important to find strong and non ambiguous relationships and assess them in order to understand their nature, the role of each actor and the modus operandi. In this section we provide evidence of code overlaps, shared configurations, packers used and TTPs supporting the assessment that Bitpaymer, WastedLocker, Hades, PhoenixLocker and PayloadBIN share a simila codebase.

We decided to reassess the relationship between ransomware variants which have been analyzed before and attributed to EC. This analysis provides the foundation on which we examine further changes and evolutions. We compared each variant with its ancestor in a chronological order.

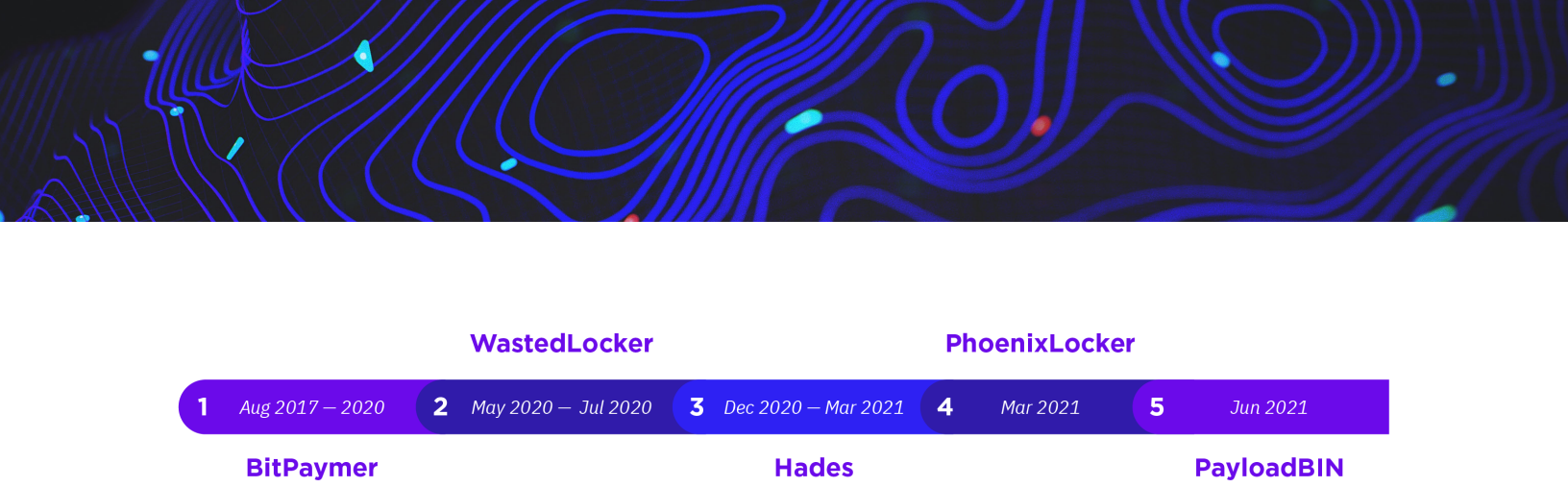


Fig 2

BITPAYMER VS. WASTEDLOCKER

Previous research[18] shows a sort of knowledge re-use between these 2 families, which share the same heritage. The main similarities found in that research are:

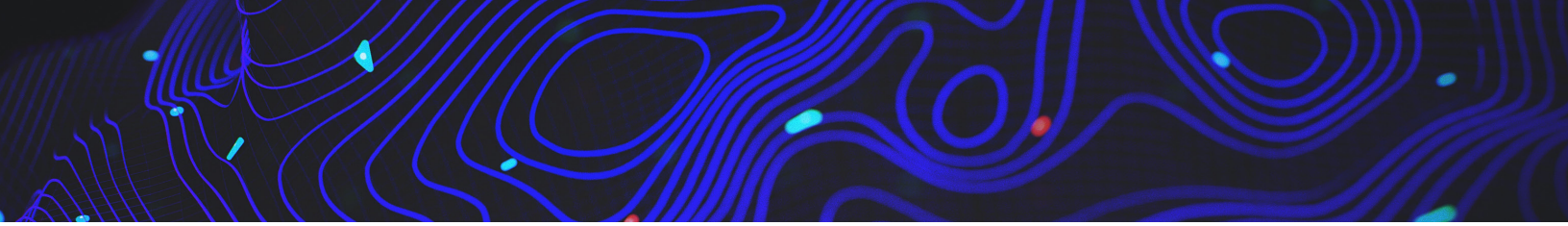
- Abuse of Alternate Data Streams (ADS);
- Customized API resolving method;
- Similar UAC bypass;
- Encryption methods;
- Ransom note;
- Same style of command-line arguments;
- Victim specific elements are added using a specific builder rather than at compile time[19].

WASTEDLOCKER VS. HADES

The two families implemented different instruction sets, in addition, the standard win32 API was rewritten in Hades as native API from NTDLL. Previous research[20] assessed the main similarities and differences between the two ransomware families:

- Different UAC bypass methods: both taken from UACME project[21]
- Generalization: Hades does not contain victim information in the ransom note whereas WastedLocker does, and instead contains a tox channel in order to communicate and negotiate with victims
- Hades does not use ADS whereas WastedLocker and BitPaymer do
- Hades stores key information in each encrypted file while WastedLocker and Bitpaymer store key information inside a ransom note;

[18]https://news.sophos.com/en-us/2020/08/04/wastedlocker-techniques-point-to-a-familiar-heritage/
[19]https://research.nccgroup.com/2020/06/23/wastedlocker-a-new-ransomware-variant-developed-by-the-evil-corp-group/
[20]https://www.crowdstrike.com/blog/hades-ransomware-successor-to-indrik-spiders-wastedlocker/
[21]https://github.com/hfiref0x/UACME



SentinelLabs discovered that Hades and WastedLocker share the same codebase. We assess it is almost certain that they come from the same 'factory' (the same developer or group of developers). There are important overlaps between the two families, which are summarized below. For our similarity analysis we considered the following samples from both families:

Family	Sha256 Packed	Sha256 unpacked	Compilation timestamp	Submission to VT
WastedLocker	905ea119ad8d3e54 cd228c458a1b5681 abc1f35df782977a 23812ec4efa0288a	965a7ae835ea9c53 c9617fc337317c79 9222094846d6a2b7 761dfae1f594aec3	2020-07-22	2020-07-23
Hades	fe997a590a68d98f 95ac0b6c994ba69c 3b2ece9841277b7 fecd9dfaa6f589a87	4bb5636c13adf39d 07f6d09506065b4 a727b6c3c348967a be3f122b217be8e9a	2020-12-20	2021-01-07

FILE AND DIRECTORY ENUMERATION ROUTINE

Both samples implement file and directory enumeration logic identically. Comparing the logic and the Control Flow Graph of both routines, we concluded both ransomware use the same code for file and directory enumeration.

The routines under scope are:

WastedLocker	.text:00402F88 f_enumerate_files
Hades	.obX0:00000001401B6AA9 f_enumerate_files

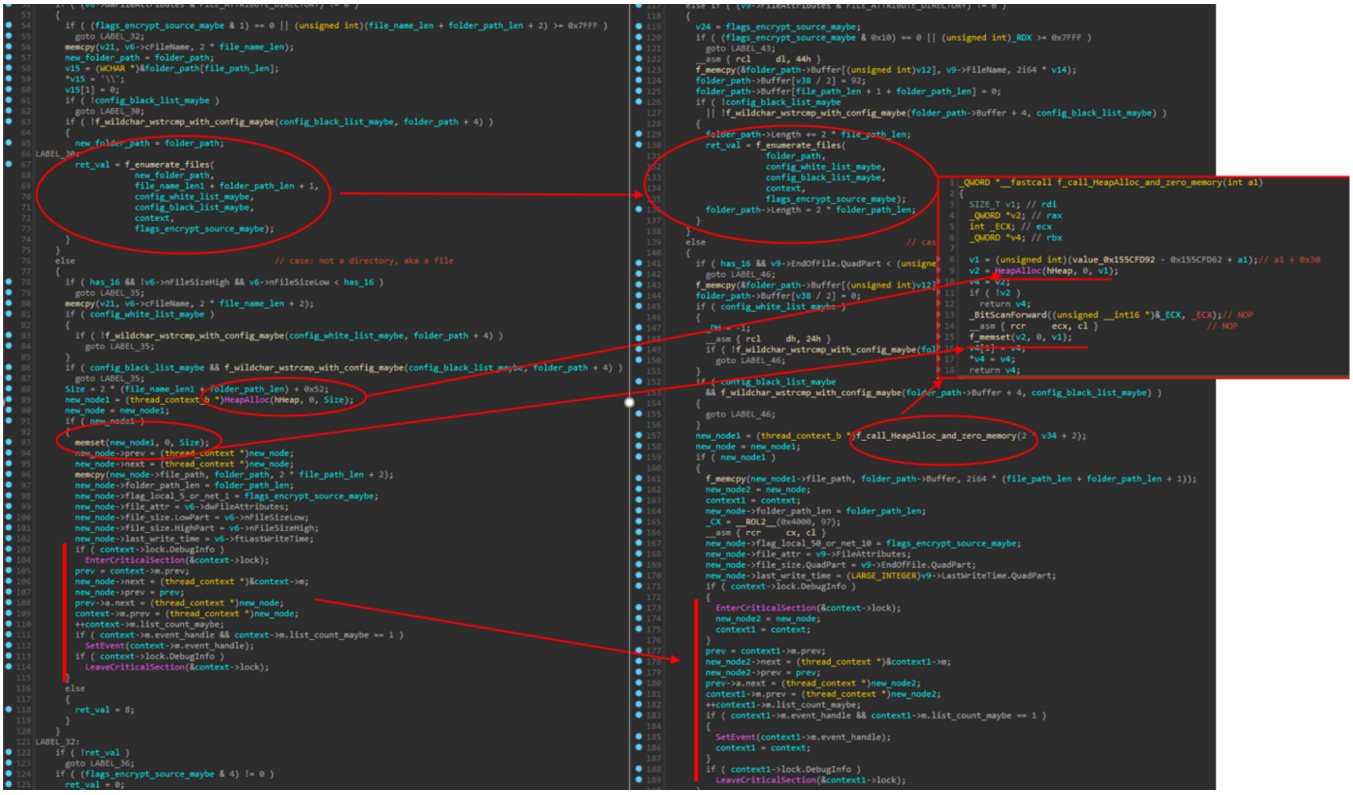


Fig 3: WastedLocker(left), Hades(right). “Similarity of file and dir enumeration routine function”.

On the left side we have WastedLocker, while on the right side, Hades. Side-by-side, we noticed the same structure between the above functions, and the same information was used to keep files for encryption. Moreover, the same approach was used to allocate a memory heap (heapAlloc) then zeroing the allocated memory with memset. The only difference in heap allocation is that Hades groups the heapAlloc and the memset into one single function (f_call_HeapAlloc_and_ zero_memory). Furthermore, the Control Flow Graphs of the two functions are almost identical:

Fig 4: WastedLocker(left), Hades(right). “CFG similarity of file and dir enumeration routine”.

DRIVES ENUMERATION ROUTINE:

We found similarities between the functions responsible for drive enumeration. In the figures below, the two functions present nearly identical structures, and rely on the same Win32API for critical sections management:

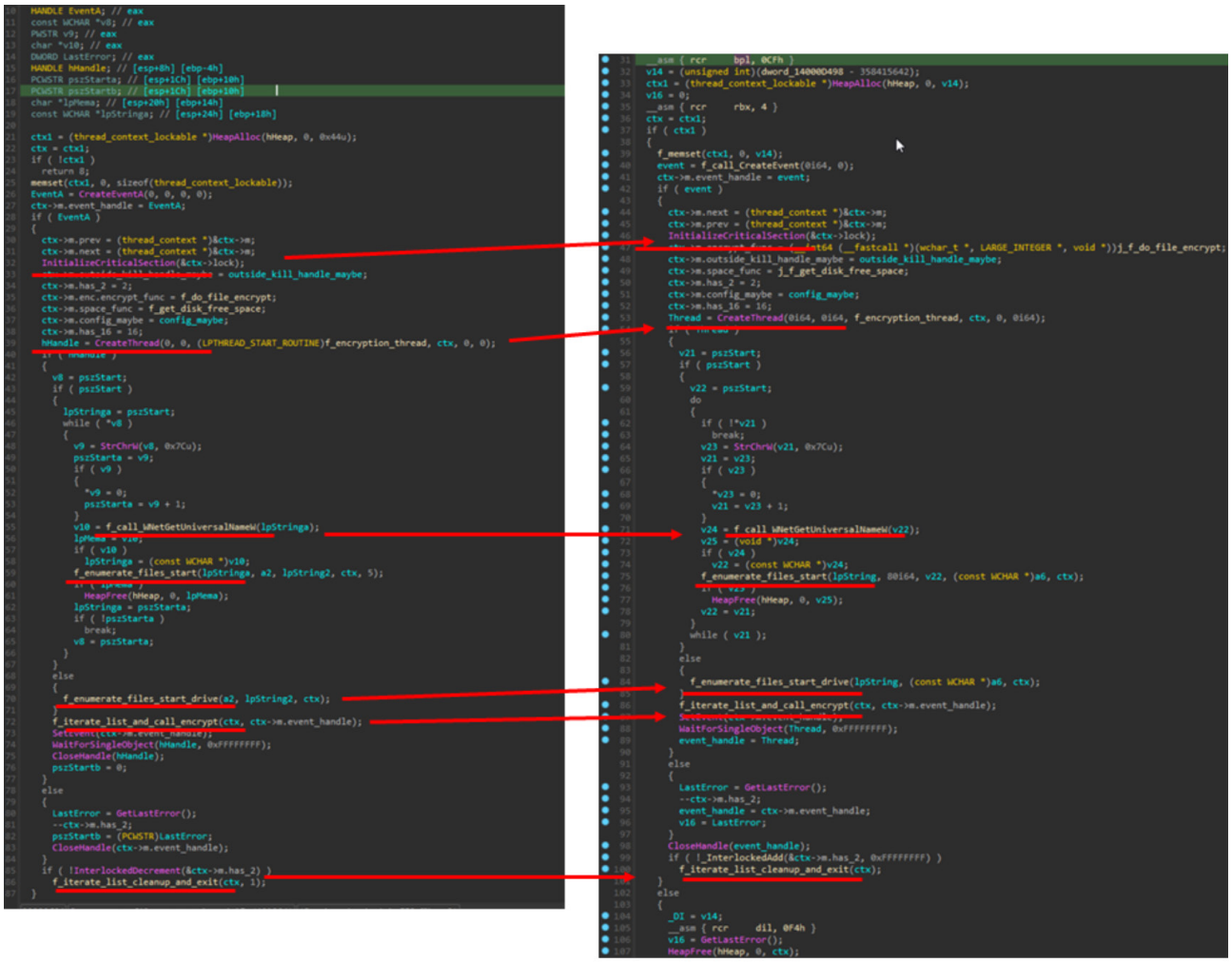


Fig 5: WastedLocker(left), Hades(right). “Drive enumeration routine similarities”.

RSA ENCRYPTION ROUTINE

During our investigation, we observed that the RSA functions - responsible for asymmetrically encrypting the keys which were used in the AES phase to encrypt files - are identical in both ransomware, hinting that the same utility library was used.

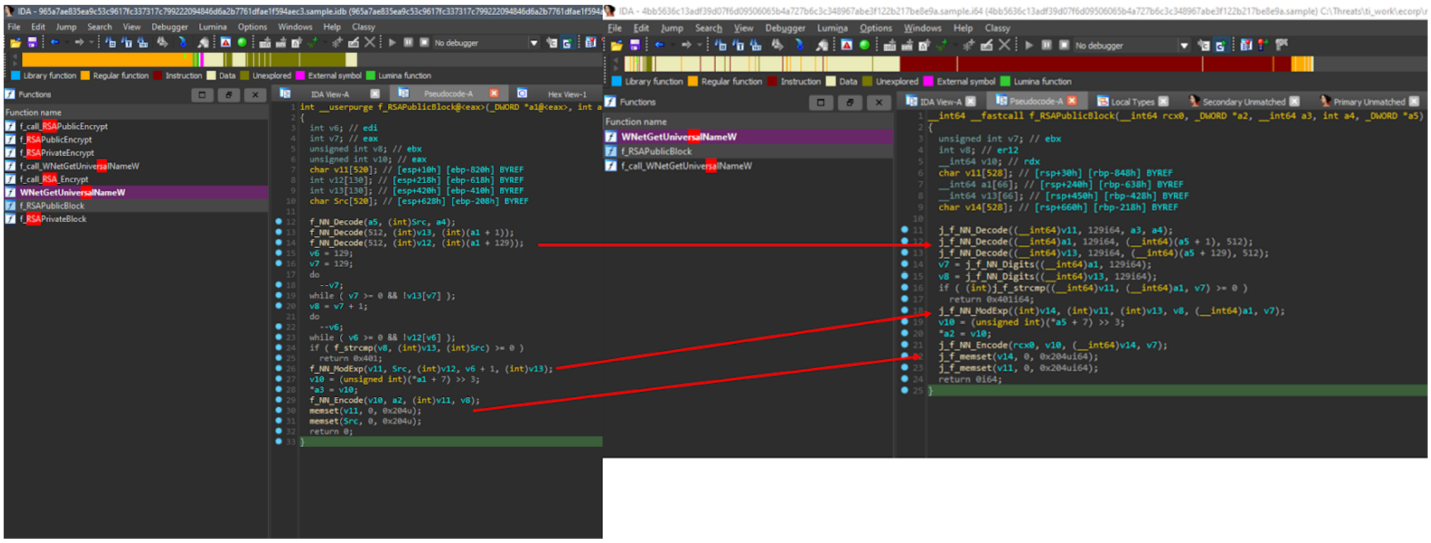


Fig 6:WastedLocker(left), Hades(right). “same RSA implementation”.

Both Ransomware rely on the RSAREF[22] crypto lib to implement the RSA algorithm and crypto utils (i.e md5sum). The RSAREF is not maintained anymore (last activity dates back about 5 years ago, at the time of writing). This is a peculiar element because, to the best of our knowledge, no other known ransomware families use this specific implementation. Further, the way the two implementations look - their structure, the same RSA implementation, the same API pattern - led us to think that they are both developed by the same hand or relying on the same functions/code reuse.

[22]https://sourceforge.net/projects/rsaref/

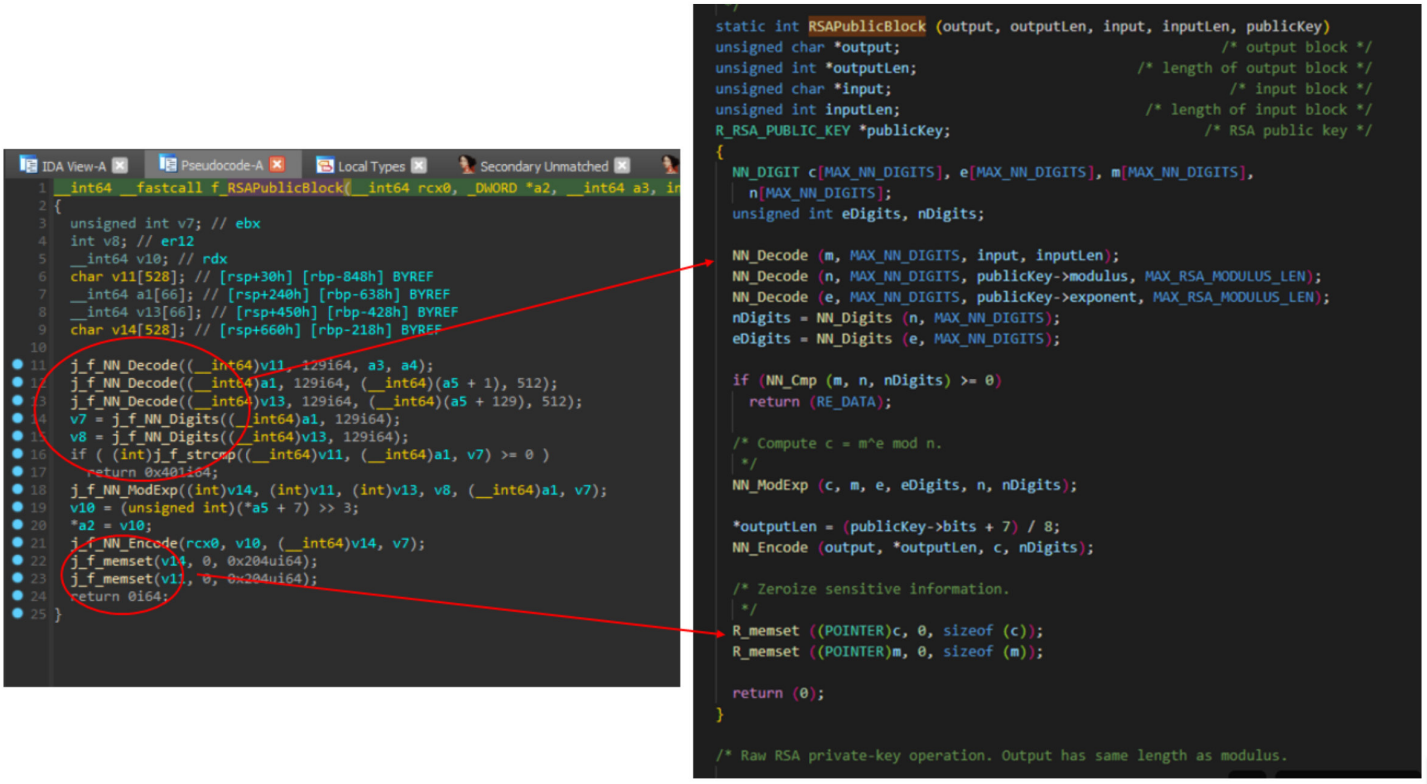


Fig 7:WastedLocker(left) uses API from RSAREF crypto lib(right)

Whilst WastedLocker and Hades have striking similarities in file and directory enumeration, drive enumeration and RSA encryption, we also note some differences between the ransomwares. The main differences between WastedLocker and Hades are summarized below:

- WastedLocker is a 32-bit PE while Hades is a 64-bit PE.
- Most of the file-related APIs for file management which were used in WastedLocker are replaced by their lower-level ones. Some examples include:
  - FindFirstFileW + FindNextFileW -> ZwCreateFile + ZwQueryDirectoryFile
  - CreateFileW + WriteFileW + CloseHandle -> ZwCreateFile + ZwWriteFile + ZwClose
  - MoveFileW -> ZwCreateFile + ZwSetInformationFile(..., FileRenameInformation)
  - DeleteFileW -> ZwCreateFile + ZwSetInformationFile(..., FileDispositionInformation)
- While WastedLocker assembly code was left mostly unchanged, Hades assembly code was modified heavily to hinder static analysis, similarity tools or emulation. Some modifications include:
  - Insertion of a wide range of no-op-codes, which modify the states of unused registers, therefore not changing the outcome of functions.
  - A small subset of APIs are imported dynamically at runtime, like: CryptAcquireContextW, CryptGenRandom and CryptReleaseContext.
  - In their prolog, a few functions contain a call to subroutines which include somewhat more esoteric op-codes and indirect calls.

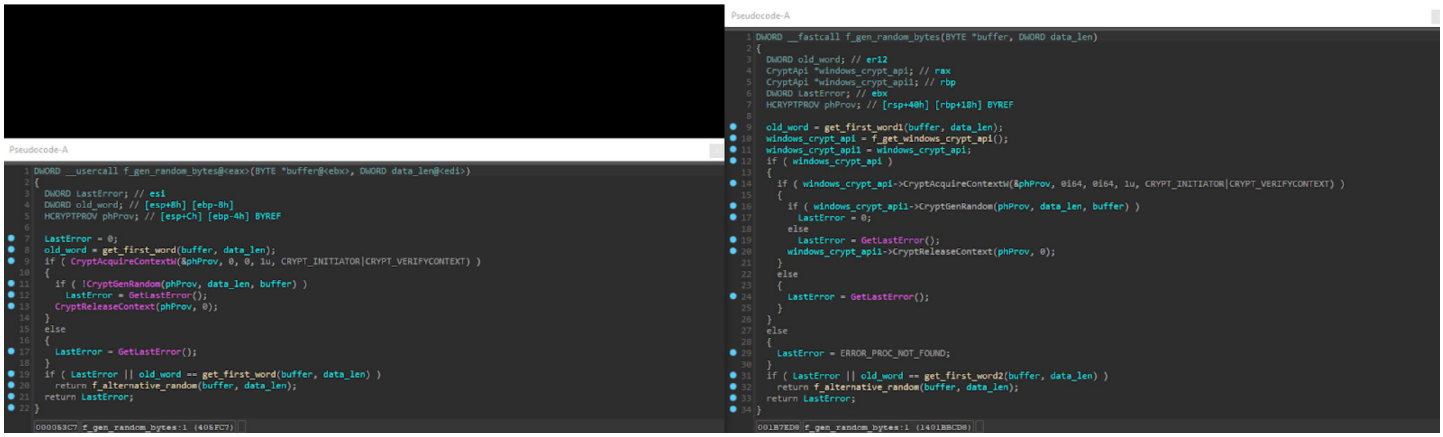


Fig 8: WastedLocker(left), Hades(right). “A small subset of APIs are imported dynamically at runtime”.

The changes between WastedLocker and Hades served primarily to change the detection signature of the malware and make it more evasive. The similarities in implementation and code overlap however, suggest Hades is highly likely an evolution of WastedLocker.

HADES VS. PHOENIX LOCKER: A POLYMORPHIC CODE

For this analysis, we investigated the following samples:

Family	Sha256 Packed	Sha256 unpacked	Compilation timestamp	Submission to VT
Hades	e657ff4838e47465 3b55367aa9d4a06 41b35378e2e379ad 0fdd1631b3b763ef0	d62425b0bb7e2be b2a96839c1231457 08644e4cc9a599b87 11cbe1a5472527ea	2021-03-06	2021-03-26
PhoenixLocker	008ec7976532520 0361d9c93ac35edd 430f8b17894ff8432 68caa5acd6224549	79d7706aff1e6f008 dc25b4b7e1dd2bd2 f9acd7cf6dfd89c96 470203a684643a	2021-03-20	2021-03-26

Both payloads have the same compilation timestamp (orange column in the table below). This suggested that the same ransomware payload (Hades in this case) was reused, but packed in two different moments in time. The compiler and linker versions are also the same. This technique of payload reuse was also seen in BitPaymer[23]. Moreover they have a different “business” infrastructure: different wallets, communication with the victim.

[23]https://blog.morphisec.com/bitpaymer-ransomware-with-new-custom-packer-framework

Sample	Sha1	Compilation timestamp (packed)[24]	Compilation timestamp (unpacked)[25]	Compiler/Linker	CryptOne Version
Hades	b4061d4227e08cfaa3190dea9926571fca2736a1 unpacked: f58c980816ef122973dbbba2c66f8c6850f45764	2021-03-06 18:13:45	2020-12-14 09:34:38	Compiler: Microsoft Visual C/C++(2008)[-] Linker: Microsoft Linker(9.0) [EXE64,signed]	111111111\{aa5b6a80-b834-11d0-932f-00a0c90dcaa9}
PhoenixLocker	3cb0cb07cc2542f1d98060adccda726ea865db98 Unpacked: 53185223c036f010faa1e077b4eaa597392ca76c	2021-03-20 20:00:50	2020-12-14 09:34:38	Compiler: Microsoft Visual C/C++(2008)[-] Linker: Microsoft Linker(9.0) [EXE64,signed]	111111111\{aa5b6a80-b834-11d0-932f-00a0c90dcaa9}

The interesting observation is that the payload was compiled at the same time but presents a different hash. We confirmed that they reused a 'clean' Hades version each time, statically introducing junk code with the help of a script in order to alter the signature.

[24]see the timing analysis section

[25]see the timing analysis section

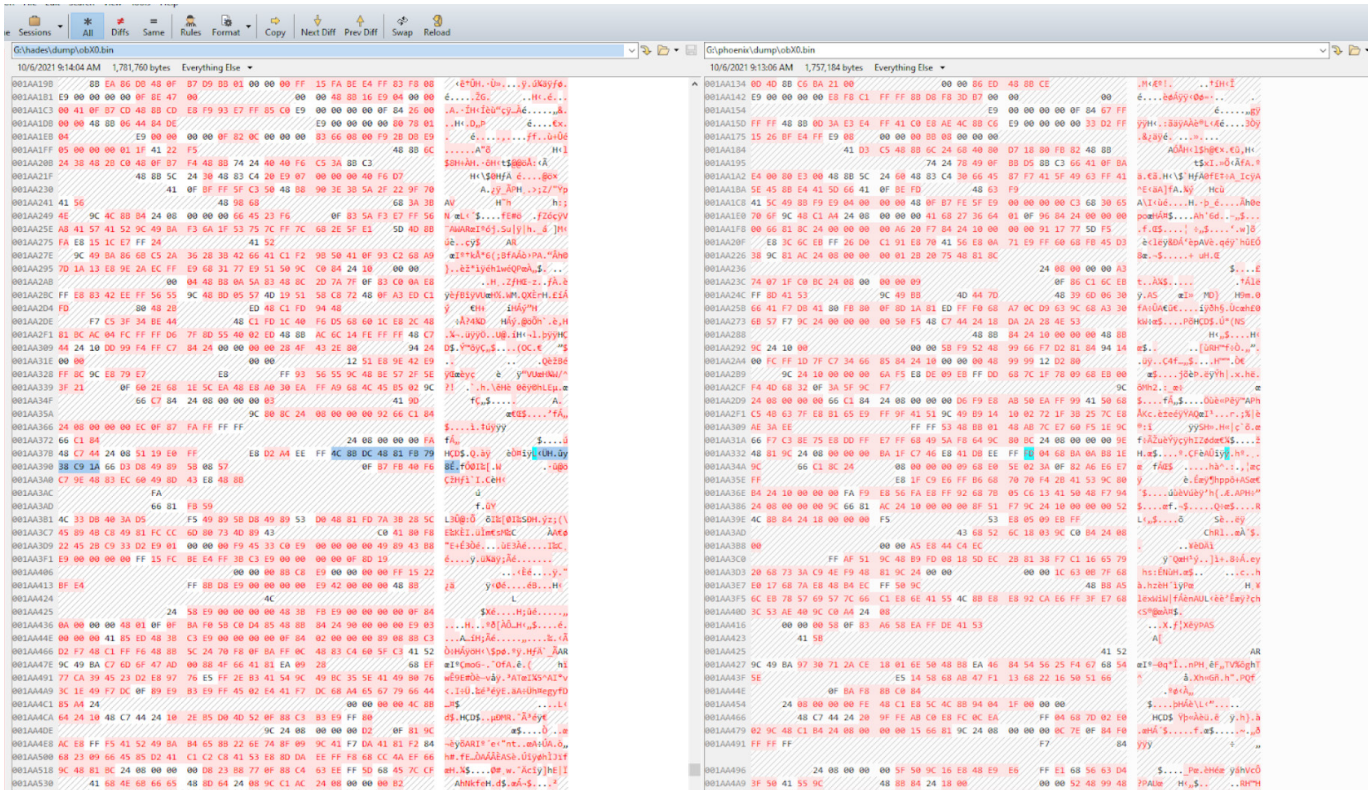


Fig 9: Hades(left), PhoenixLocker(right). “A hex view of the .obX0 section”.

From a first glance at both samples, it is clear that the .obX0 section is different (differences highlighted in red), but the semantic of the code didn’t change, as suggested by the pseudo code view below:

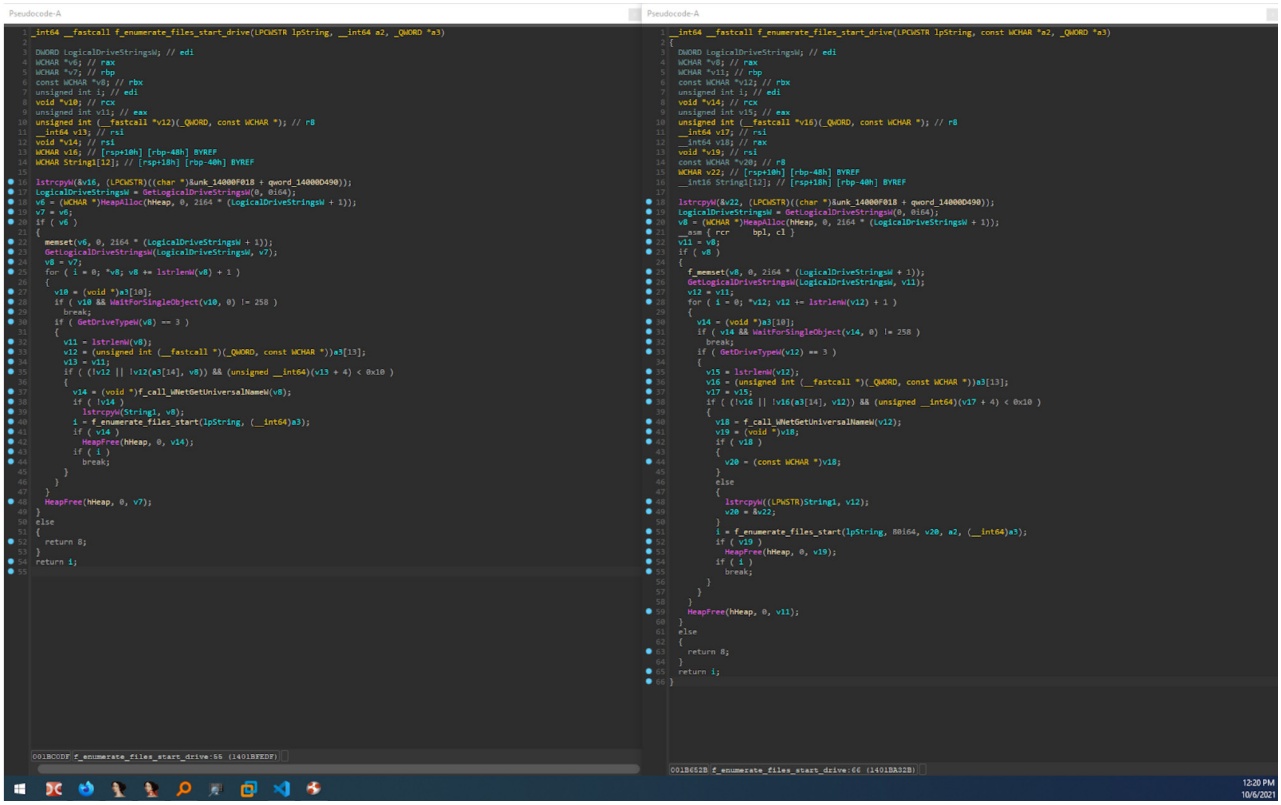


Fig 10: Hades(left), PhoenixLocker(right). “A comparison of the enumerate file function”.

In the figure below, the structure of both Hades and PhoenixLocker appear almost identical. However, upon closer inspection, the assembly instructions inside each block are different. Those instructions are highlighted in red (left side) and in blue (right side). The blue represents the code added to the 'clean' version of Hades. All these introduced instructions don't alter the semantic, they were added with the sole purpose of improving the stealthiness of the ransomware by changing the static signature.

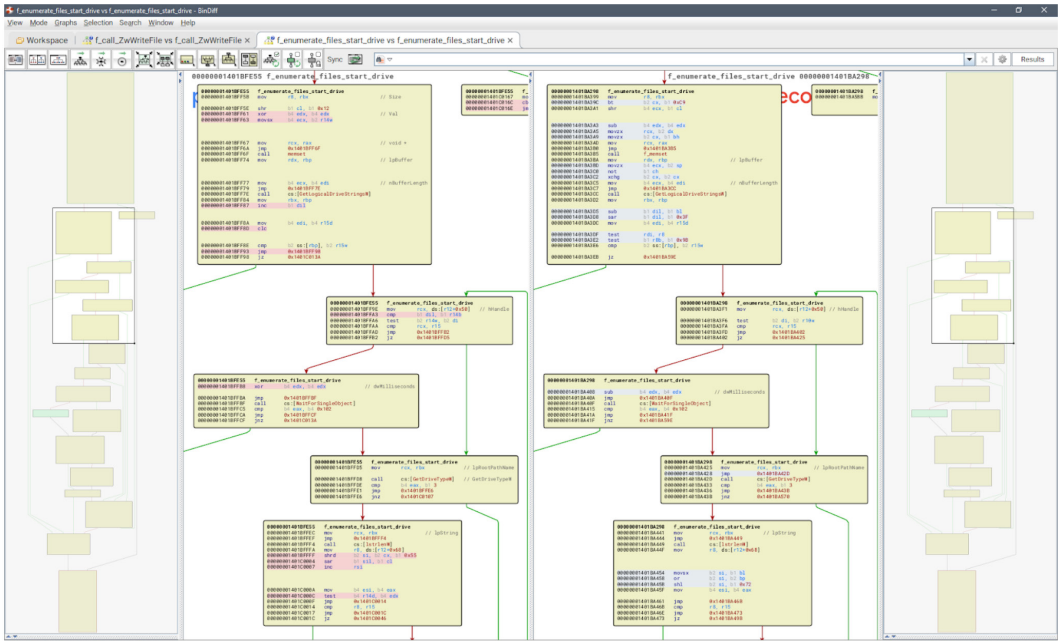


Fig 11: Hades(left), PhoenixLocker(right). "Polymorphic code added statically highlighted in red and blue".

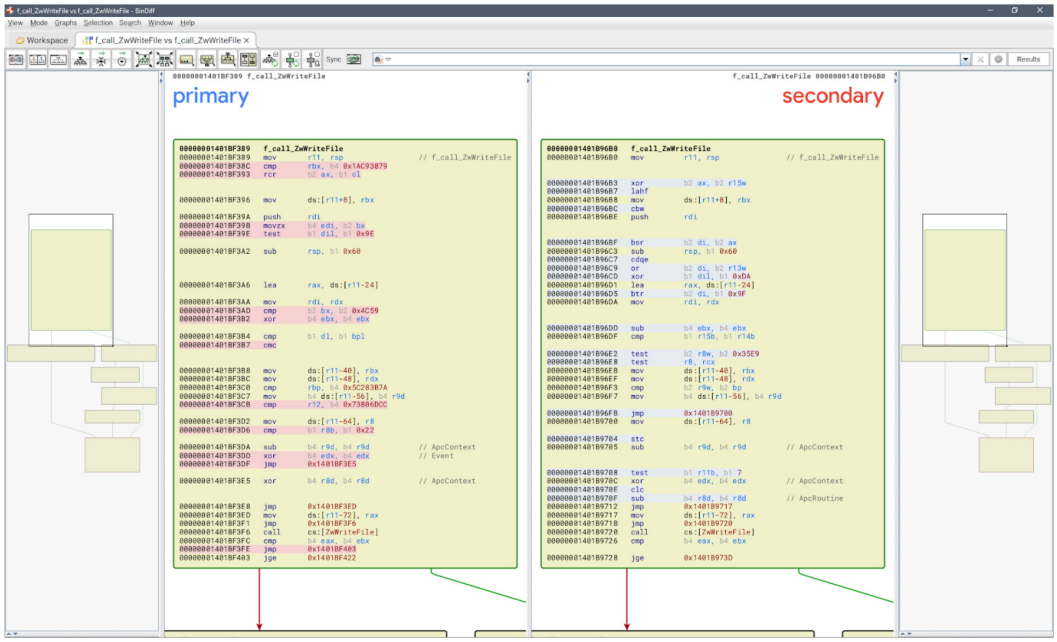


Fig 12: Hades(left), PhoenixLocker(right). "Polymorphic code added statically highlighted in red and blue".

THE INCREMENTAL LINKING

Because of the presence of jump thunks in the malware code, it might be the case that incremental linking is enabled in the compilation process of the malware. This could explain why most functions are the same (except for the discussed polymorphic code changes which are introduced after compilation), except those that were changed between December and April to recompile again.

PHOENIXLOCKER VS. PAYLOADBIN

In this section, we compared the following two samples:

Family	Sha256 Packed	Sha256 unpacked	Compilation timestamp	Submission to VT
PhoenixLocker	008ec7976532520 0361d9c93ac35edd 430f8b17894ff8432 68caa5acd6224549	2E38FF7D2D382F227 54D16E531B928DAD BA48CD562B06787B DD5FA6CC681772E	2021-03-20	2021-03-26
PayloadBIN	69775389eb0207f ec3a3f5649a0ad931 5856c810f595c086 ac49d68cdbc1d136	8AA971223915DE40 7C8F8B8CCD722E26 41C5C2A0A9998BB9 ECDB06D864935D7A	2021-06-02	2021-06-03

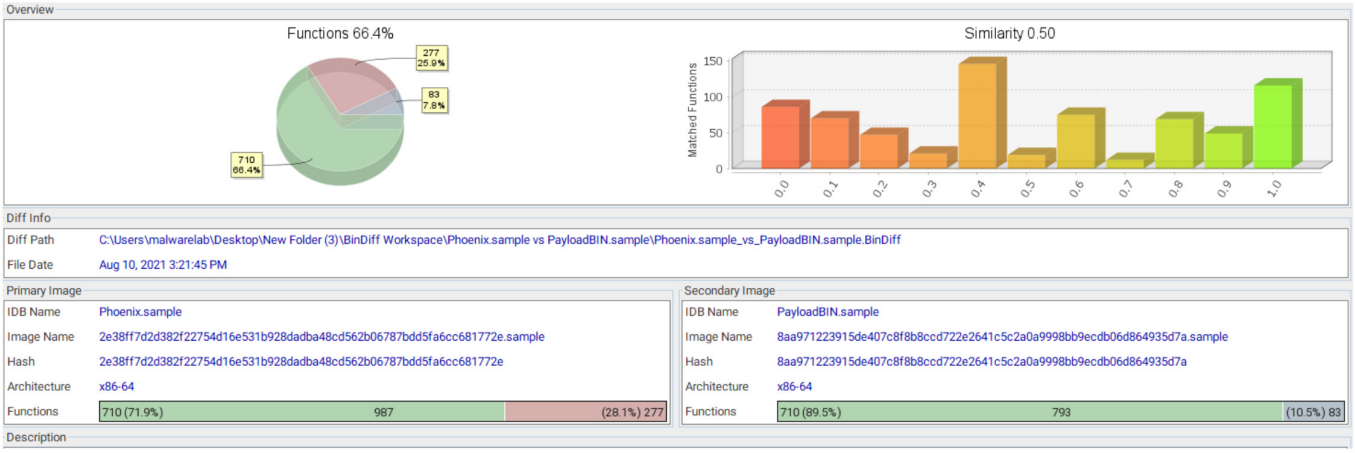


Fig 13

Taking these two samples we conducted a similarity analysis to deduce their connection. The similarity score between the two samples was slightly higher, with a value of 0.50. Taking a closer look, we observed that the majority of the PayloadBIN functions overlap with its ancestor PhoenixLocker. There was a partial rewrite of the ransomware code however, justified by the compilation timestamps of the unpacked payload, which is Wed Apr 28 11:04:49 2021 for PayloadBIN and Mon Dec 14 01:34:38 2020 for Phoenix Locker. This suggests EC had more than 4 months to work on the improvement.

In this case, there are portions of the code that are completely rewritten and consolidated. There are also a lot of functions with significant overlap:

Show structural changes | Show only instructions changed | Show identical

Similarity	Confidence	Address	Primary Name	Type	Address	Secondary Name	Type	Basic Blocks			Jumps		
0.90	0.92	0000000140...	sub_1401B8019	Normal	0000000140...	sub_1401C3F08	Normal	0	5	0	0	6	0
0.90	0.93	0000000140...	sub_140001098	Normal	0000000140...	sub_1400099E0	Normal	0	4	0	0	4	0
0.90	0.95	0000000140...	sub_140074F3F	Normal	0000000140...	sub_140055B80	Normal	0	2	0	0	1	0
0.90	0.93	0000000140...	sub_1401B819F	Normal	0000000140...	sub_1401C26F3	Normal	0	10	0	0	12	0
0.90	0.92	0000000140...	sub_1401BE318	Normal	0000000140...	sub_1401C3DF1	Normal	0	9	0	0	15	0
0.90	0.93	0000000140...	sub_1401BE80F	Normal	0000000140...	sub_1401C0E2A	Normal	0	4	0	0	5	0
0.90	0.93	0000000140...	sub_140002EF4	Normal	0000000140...	sub_1400043B4	Normal	0	9	0	0	11	0
0.90	0.93	0000000140...	sub_140003794	Normal	0000000140...	sub_140005B58	Normal	0	6	0	0	7	0
0.90	0.93	0000000140...	sub_140005BE0	Normal	0000000140...	sub_1400060AC	Normal	0	4	0	0	4	0
0.90	0.92	0000000140...	sub_140006434	Normal	0000000140...	sub_140006ED8	Normal	0	11	0	0	14	0
0.90	0.93	0000000140...	sub_140006B5C	Normal	0000000140...	sub_140005680	Normal	0	10	0	0	16	0
0.90	0.92	0000000140...	sub_1400048F4	Normal	0000000140...	sub_140008EA4	Normal	0	7	0	0	9	0
0.90	0.93	0000000140...	sub_1401B45FA	Normal	0000000140...	sub_1400025BC	Normal	0	9	0	0	12	0
0.90	0.93	0000000140...	sub_140004934	Normal	0000000140...	sub_1400021A8	Normal	0	19	0	0	30	0
0.90	0.93	0000000140...	sub_1401B96B0	Normal	0000000140...	sub_1401C4DF8	Normal	0	7	0	0	9	0
0.90	0.93	0000000140...	sub_140001F74	Normal	0000000140...	sub_14000958C	Normal	0	10	0	0	13	0
0.90	0.94	0000000140...	sub_1400090FC	Normal	0000000140...	sub_140006F64	Normal	0	5	0	0	5	0
0.90	0.93	0000000140...	sub_1401C0596	Normal	0000000140...	sub_140006B40	Normal	0	12	0	0	16	0
0.90	0.93	0000000140...	sub_1401BA298	Normal	0000000140...	sub_1401BE169	Normal	0	18	0	0	29	0

Fig 14

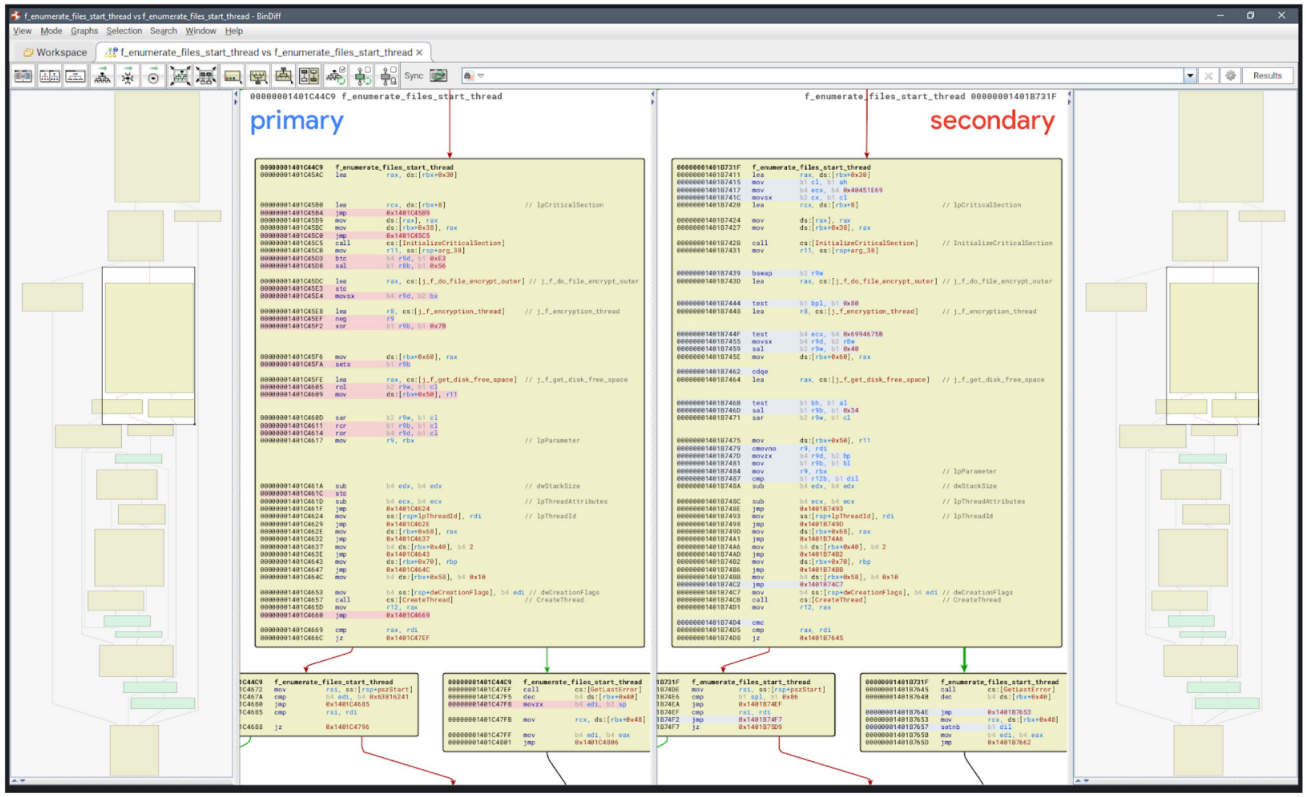


Fig 15: PayloadBIN (left) vs PhoenixLocker cryptolocker (right).
File enumerating functions are practically identical.

TTPS

We conducted further similarity analysis by analyzing the TTPs of the different variants. We did this by extracting the main command lines from all the ransomwares and comparing them. We distinguished two distinct clusters:

The first TTP cluster we identified was WastedLocker. The second TTP cluster contained Hades, Phoenix and PayloadBIN. From Hades onwards, we found a unique self-delete implementation including the 'waitfor' command:

```
cmd /c waitfor /t 10 pause /d y & attrib -h
"C:\Users\Admin\AppData\Roaming\CenterLibrary\Tip" & del
"C:\Users\Admin\AppData\Roaming\CenterLibrary\Tip" & rd
"C:\Users\Admin\AppData\Roaming\CenterLibrary\"
```

'waitfor' command in Hades, Phoenix Locker and PayloadBIN

This command is not present in WastedLocker, where the 'choice' command is used instead:

```
cmd /c choice /t 10 /d y & attrib -h "C:\Users\Admin\AppData\Roaming\Wmi" & del "C:\Users\Admin\
AppData\Roaming\Wmi"
```

'choice' command in WastedLocker

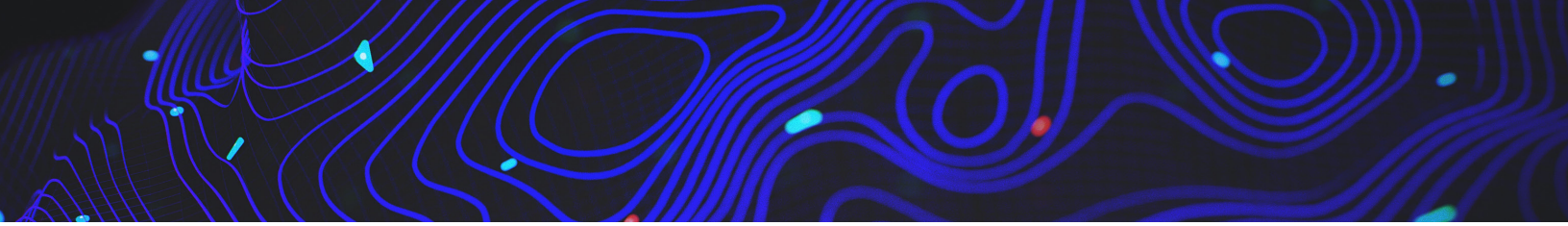
Whilst syntax difference may seem like a significant difference, these two implementations are very similar: the logic is the same, only the signature changes.

All ransomwares have the same implementation of Shadows copy deletion:

```
C:\Windows\system32\vssadmin.exe Delete Shadows /All /Quiet
```

Shadows deletion in WastedLocker Hades, Phoenix Locker, PayloadBIN

The evidence of this code reuse supports the assessment that it is almost certain these ransomware families are related to the same 'factory'.



The following table summarizes all the ransomware procedures:

Ransomware Strain	Malicious Command Lines
WastedLocker	*C:\Windows\system32\vssadmin.exe Delete Shadows /All /Quiet* *cmd /c choice /t 10 /d y & attrib -h* *“C:\Users\Admin\AppData\Roaming\Wmi” & del* *“C:\Users\Admin\AppData\Roaming\Wmi”* *C:\Users\Admin\AppData\Roaming\Wmi:bin -r*
Hades	*C:\Windows\system32\vssadmin.exe Delete Shadows /All /Quiet* *C:\Users\Admin\AppData\Roaming\CenterLibrary\Tip /go* *cmd /c waitfor /t 10 pause /d y & attrib -h* *“C:\Users\Admin\AppData\Roaming\CenterLibrary\Tip” & del* *“C:\Users\Admin\AppData\Roaming\CenterLibrary\Tip” & rd* *“C:\Users\Admin\AppData\Roaming\CenterLibrary\”* *waitfor /t 10 pause /d y* *attrib -h* *“C:\Users\Admin\AppData\Roaming\CenterLibrary\Tip”*
PhoenixLocker	*C:\Users\Admin\AppData\Roaming\SetupPlay8\Dev /go* *cmd /c waitfor /t 10 pause /d y & attrib -h* *“C:\Users\Admin\AppData\Roaming\SetupPlay8\Dev” & del* *“C:\Users\Admin\AppData\Roaming\SetupPlay8\Dev” & rd* *“C:\Users\Admin\AppData\Roaming\SetupPlay8\”* *waitfor /t 10 pause /d y* *attrib -h “C:\Users\Admin\AppData\Roaming\SetupPlay8\Dev”* *“C:\Windows\system32\NOTEPAD.EXE”* *C:\Users\Admin\Desktop\PHOENIX-HELP.txt*
PayloadBin	*vssadmin.exe Delete Shadows /All /Quiet* *wmic process call create “%s” > nul && exit* *C:\Users\Admin\AppData\Roaming\SetupPlay8\Dev /go* *cmd /c waitfor /t 10 pause /d y & del* *“C:\Users\Admin\AppData\Roaming\SetupPlay8\Dev” & rd* *“C:\Users\Admin\AppData\Roaming\SetupPlay8\”* *waitfor /t 10 pause /d y*

CONFIGURATIONS

The extraction of static configurations for each variant shows strong similarities. The most significant similarity is that each variant uses the IFSB-inspired static configuration based on a whitelisting approach:

Ransomware Strain	Whitelist
WastedLocker	*\NTLDR*\BOOTMGR*\GRLDR*.386*.ps1*.msu*.ani*.wpx*.hlp*.ocx*.com*.cpl*.adv*.cmd*.lnk*.drv*.sys*.icl*.nls*.cab*.bat*.theme*.bin*.key*.themepack*.msi*.icns*.ics*.idx*.hta*.scr*.msstyles*.diagcfg*.diagcab*.nomedia*.msc*.cur*.mod*.shs*.rtp*.rom*.msp*.ini*.bak*.dat*.sdi*.wim*.dll*.exe bin|Boot|boot|dev|etc|lib|initdr|sbin|sys|vmlinuz|run|var|\ Boot|System Volume Information|$RECYCLE. BIN|WebCache|Caches|WindowsApps|AppData|ProgramData|\Users\All Users ProgramData%|%windir%|%temp%|%AppData%|C:\Recovery|C:\Program Files|C:\ Program Files (x86)*
Hades	*how-to-decrypt-dvxr9.txt|*.dvxr9|*.386|*.adv|*.bat|*.bin|*.cab|*.cmd|*.com|*.cpl|*.dat|*.dll|*.drv|*.exe|*.hlp|*.hta|*.icl|*.idx|*.ini|*.key|*.lnk|*.mod|*.msc|*.msi|*.msp|*.msu|*.nls|*.ocx|*.ps1|*.rom|*.rtp|*.scr|*.sdi|*.shs|*.sys|*.wim|*.wpx|*\bootmgr|*\grldr|*\ntldr|c:\windows*|c:\users\%USERNAME%\appdata\roaming*|c:\users\%USERNAME%\appdata\local\temp*|c:\programdata*|c:\recovery*|c:\program files (x86)*|c:\program files*|*\boot*|*\system volume information*|*\$recycle.bin*|*\webcache*|*\caches*|*\windowsapps*|*\appdata*|*\programdata*|*\users\all users*|*\bin*|*\boot*|*\boot*|*\dev*|*\etc*|*\lib*|*\initdr*|*\sbin*|*\sys*|*\vmlinuz*|*\run*|*\var*
PhoenixLocker	*phoenix-help.txt|*.phoenix|*.386|*.adv|*.bat|*.bin|*.cab|*.cmd|*.com|*.cpl|*.dat|*.dll|*.drv|*.exe|*.hlp|*.hta|*.icl|*.idx|*.ini|*.key|*.lnk|*.mod|*.msc|*.msi|*.msp|*.msu|*.nls|*.ocx|*.ps1|*.rom|*.rtp|*.scr|*.sdi|*.shs|*.sys|*.wim|*.wpx|*\bootmgr|*\grldr|*\ntldr|c:\windows*|c:\users\%USERNAME%\appdata\roaming*|c:\users\%USERNAME%\appdata\local\temp*|c:\programdata*|c:\recovery*|c:\program files (x86)*|c:\program files*|*\boot*|*\system volume information*|*\$recycle.bin*|*\webcache*|*\caches*|*\windowsapps*|*\appdata*|*\programdata*|*\users\all users*|*\bin*|*\boot*|*\boot*|*\dev*|*\etc*|*\lib*|*\initdr*|*\sbin*|*\sys*|*\vmlinuz*|*\run*|*\var*
PayloadBin	*PAYLOADBIN-README.txt* *movable|fixed|remote|share* *.386|*.adv|*.bat|*.bin|*.cab|*.cmd|*.com|*.cpl|*.dat|*.dll|*.drv|*.exe|*.hlp|*.hta|*.icl|*.idx|*.ini|*.key|*.lnk|*.mod|*.msc|*.msi|*.msp|*.msu|*.nls|*.ocx|*.ps1|*.rom|*.rtp|*.scr|*.sdi|*.shs|*.sys|*.wim|*.wpx|*\BOOTMGR|*\GRLDR|*\NTLDR \Boot|System Volume Information|$RECYCLE. BIN|WebCache|Caches|WindowsApps|AppData|ProgramData|\Users\All Users|bin|Boot|boot|dev|etc|lib|initdr|sbin|sys|vmlinuz|run|var %windir%|%AppData%|%temp%|%ProgramData%|C:\Recovery| C:\Program Files (x86)|C:\Program Files

CLUSTERING HADES, PHOENIXLOCKER AND PAYLOADBIN

Another interesting overlap emerges from the analysis of two unpacked samples belonging to Hades and PhoenixLocker.

THE .DATA SECTION:

The .data section of some Hades samples is identical to those found in PayloadBIN and PhoenixLocker: .data md5:3fc7f20aabeef2d3290e3b71d7f639fd

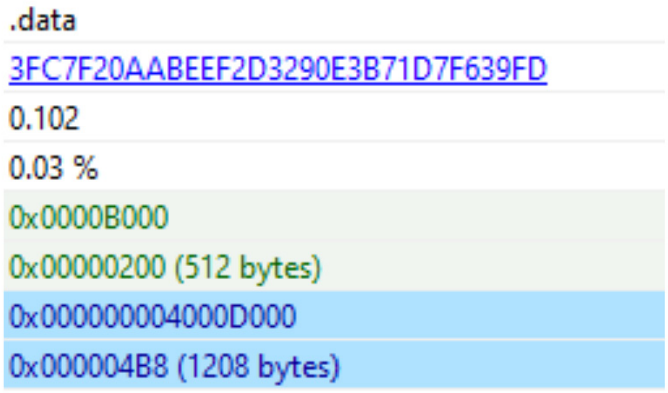


Fig 16

Moreover, there is a non canonical .obX0 section which is present in the PhoenixLocker payload and in Hades. Typically, compilers set “standard” names for sections (e.g. “.text” for code, etc..), these section names can be modified using different methods i.e. packers. It is also possible to create custom sections and assign custom names to them directly in the code, for example, to store encrypted content/payload. The presence of a custom section is particularly uncommon, and becomes a very strong marker which permits us to link these variants with little to no ambiguity.

THE .OBX0 SECTION:

Sha1	Ransomware Family
ebfedc636bad6877923a4e0bbcb16493ed0acc61	Phoenix Locker
481200175b108d4ed7284ecea67add3c0df0b12	Hades
a21562f8f1177e17d7975065d13ff0182cbef1c2	Hades
8a974ac76fb587855d488629944abfa1fb5822e3	Hades

THE \\BASENAMEDOBJECTS\\CTFCTF MUTEX:

This mutex is present in both the PayloadBIN and Hades variants.

Sha1	Ransomware Family
b4061d4227e08cfaa3190dea9926571fca2736a1	Hades
f8e52380b6f3668d4de6df416c8da389c0d98fe8	Hades
c9b25177db2f6eaddb4b028a9284b4fb5c3ffcd0	Hades
7bcea3fbfcb4c170c57c9050499e1fae40f5d731	Hades
e23637ea81751e558fca17ef1a54b6e39d2e83c3	PayloadBIN

THE \\BASENAMEDOBJECTS\\SCRIPTNET MUTEX:

This mutex is present in PhoenixLocker, PayloadBIN and Hades variants.

Sha1	Ransomware Family
16aaf95ff91ccf05e5920858f9f637abf2511e57	PayloadBin
3cb0cb07cc2542f1d98060adccda726ea865db98	PhoenixLocker
d0d68281f8459b5558559fbbf8c6c8ab4ddfec8b	Hades
f8fc84030c579070b36c99c836ac4b5c32bbc2c4	Hades
61f1c5e966450e6050e2e284765f7d0c169e5a15	Hades
fe0c77959bc7c016a49f71c765de947e3294a667	Hades

Based on all the findings above, we drew a Venn diagram demonstrating the overlaps between the different ransomware families, in which we highlight important intersections:

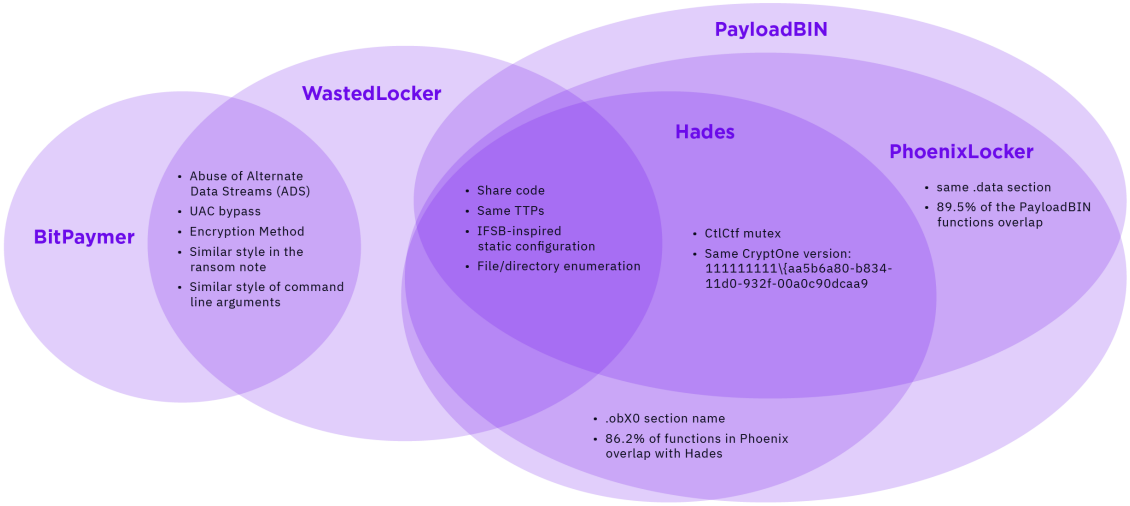


Fig 17: The Venn diagram of the overlaps between the ransomware families.

The same relations are expressed below in the form of a flow graph, in which it is also possible to decompose the various iterations from one variant to another. The strongest overlaps are highlighted in red.

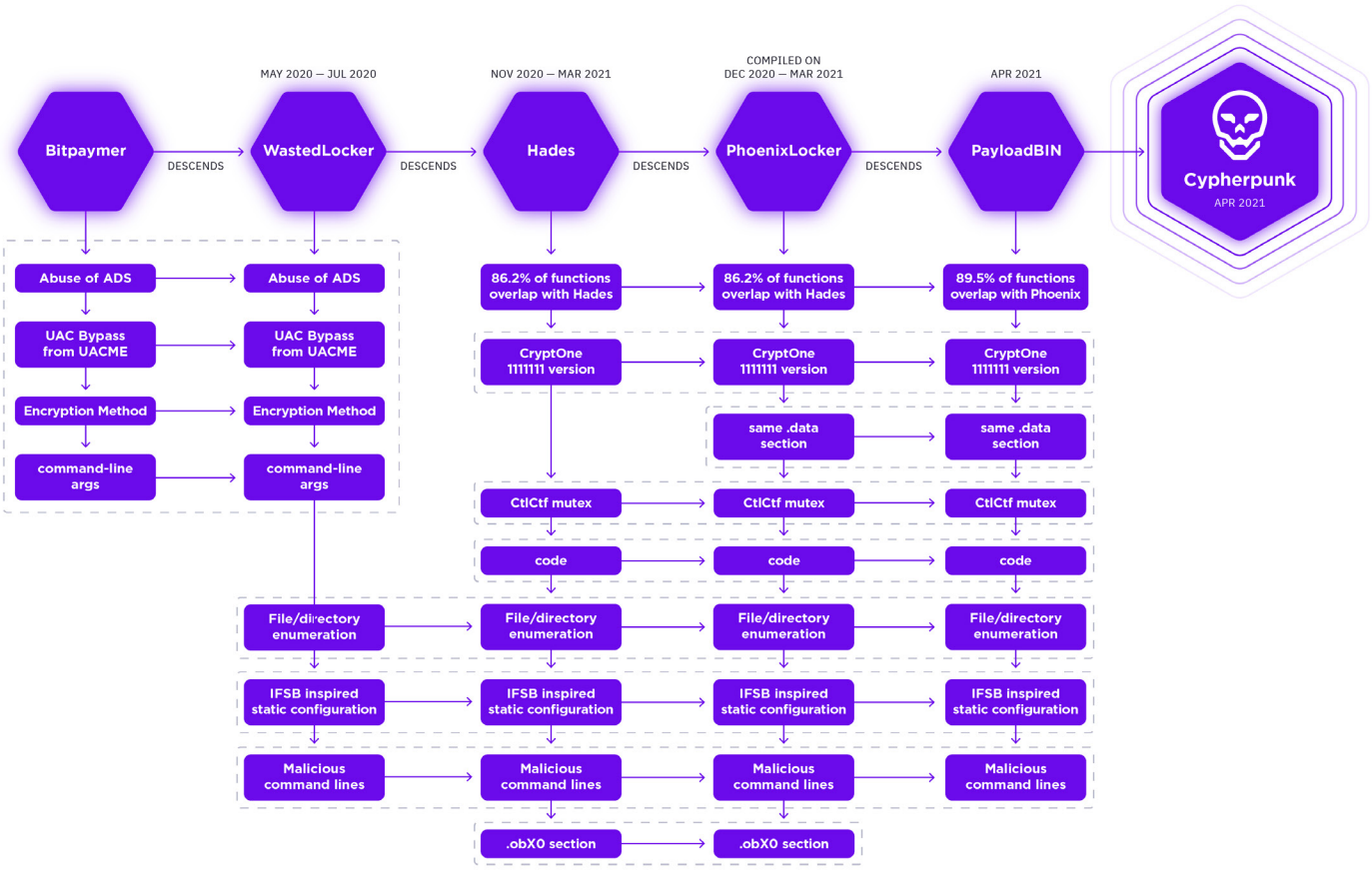


Fig 18: The evolution and overlaps between the different ransomware variants

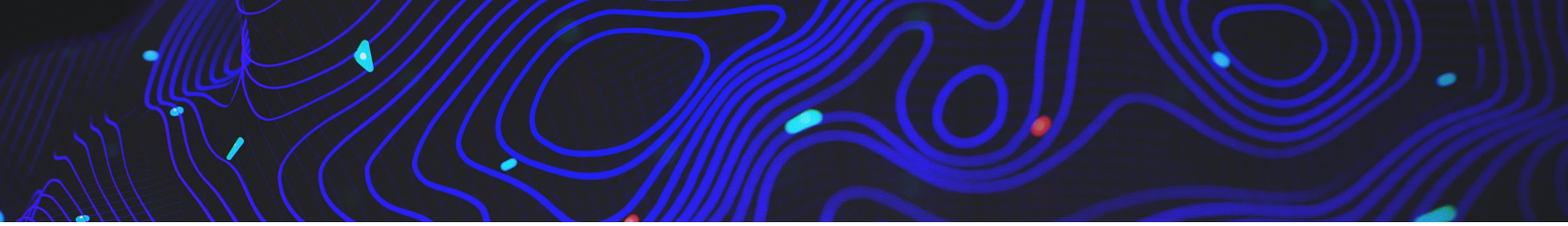


Fig. 18 clearly identifies the existence of a single cluster which presents very strong code overlap. Taking a look at the timeline, the different variants were released in a very short time period, differing mostly with simple “cosmetic” changes. From WastedLocker to Hades, however, the time between releases is longer, justifying the effort of having to rewrite the code from scratch. From our observation of the timeline, we assess it is highly likely that **all the ransomware payloads were developed by the same ‘factory’ (developer or group of developers).** Moreover, the changes in the code are consistent with the timing between releases.

CONCLUSION

SentinelLabs assesses that it is highly likely that WastedLocker, Hades, PhoenixLocker and PayloadBIN belong to the same cluster. Our assessment is based on code similarity and reuse, timeline consistency and nearly identical TTPs across the ransomware families indicating there is a consistent modus operandi for the cluster. In addition, we assess that there is a likely evolutionary link between WastedLocker and BitPaymer, and suggest that it can be attributed to the same EC activity cluster.

SentinelLabs assesses, with high confidence, Hades, Phoenix Locker and PayloadBIN variants share the same codebase. In addition, we assess it is highly likely that they belong to a common ‘factory’.

Whether Evil Corp outsourced the ransomware development is still an open question. However, it appears that they tend to maintain control over the whole ransomware development process in order to reduce the time it takes to make their ransomware operationally active. Keeping control of the process allows EC to better manage their operational security during operations. The evidence of incremental linking, the polymorphism statically introduced (which requires Hades’ codebase), and the usage of the same version of CryptOne, supports the assessment that it is highly likely a single unique cluster.

OTHER TOOLSET EXPANSION

Tracking the threat actor's tools led us to discover another new, possibly experimental, variant dubbed "Cypherpunk" - we named it this due to the encryption extension found.

```
C:\Users\Lucas\Documents\OneNote Notebooks\Personal\General.one.cypherpunk
C:\Users\Lucas\Documents\awards.xls.cypherpunk
C:\Users\Lucas\Desktop\ZoneMap.dwf.cypherpunk
C:\Users\Lucas\Documents\OneNote Notebooks\Personal\CONTACT-TO-DECRYPT.txt
C:\Users\Administrator\Searches\Everywhere.search-ms.cypherpunk
C:\Users\Lucas\Desktop\th (2).jpg.cypherpunk
C:\Users\Lucas\Documents\pexels-photo-46710.jpeg.cypherpunk
C:\Users\Lucas\Desktop\ppt_ch10.ppt.cypherpunk
C:\Users\Lucas\Desktop\WEF_Future_of_Jobs.pdf.cypherpunk
```

Files encrypted with the extension ".cypherpunk"

Code similarity analysis shows that the Cypherpunk version is the same as the previous PayloadBIN variant. It was compiled on 2021-04-01 17:15:24, 20 days after the PayLoadBIN sample. It is possible that this is another attempt at rebranding. Although this variant was reported, it was improperly flagged as 'Hades'[27].

Sha1	Filename	Variant	Compilation timestamp	Compilation Timestamp (unpacked)
e8d485259e64fd375e03 844c03775eda40862e1c	wsqmcons.exe	Cypherpunk	2021-04-04 01:25:28	2021-04-01 17:15:24

The sample is signed with the following certificate:

```
SAK GUARD SOLUTION LTD
Name: SAK GUARD SOLUTION LTD
Status: Valid
Issuer Sectigo RSA Code Signing CA
Valid From: 12:00 AM 02/17/2021
Valid To: 11:59 PM 02/17/2022
Valid Usage: Code Signing
Algorithm: sha256RSA
Thumbprint: FADE3F5FFCA06CCEEF202DDEAE9339EA64D1AD7A
Serial Number: 3D C6 B2 72 F4 66 B7 29 F8 4A 17 27 82 51 D6 CC
```

SentinelLabs assesses this new finding is likely an indication of EC working on updating their toolset.

[27]https://www.secureworks.com/blog/hades-ransomware-operators-use-distinctive-tactics-and-infrastructure

MACAW LOCKER RANSOMWARE

In October 2021, a new ransomware variant called 'Macaw Locker' appeared in the wild, in an attack that began on Oct. 10th which encrypted Olympus[28]. A Few days later, Sinclair Broadcast Group systems were also attacked, causing wide disruption[29]. Some researchers claimed a possible relation with WastedLocker[30], but at the time of writing, no one provided further details about this assessment.

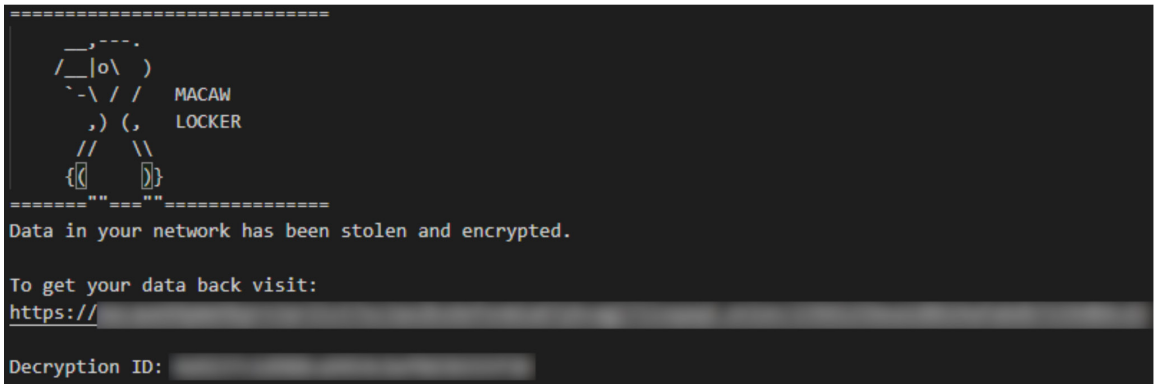


Fig 19: The content of the ransom note used in Macaw

The ransomware presents anti-analysis features like API hashing and indirect API calls with the intention of evading analysis. The only reliable way to reconstruct functions is via dynamic analysis by way of fully executing the sample. One aspect that immediately sets Macaw apart is the fact that it requires a custom token (provided from the command line), which seems specific for each victim; without it, it won't execute (also seen in Egregor and BlackCat):

```
macaw_sample.exe -k <tokeID>
```

This additional requirement also aids in anti-analysis.

Another new addition to Macaw was a special function that acquires the imports for APIs at runtime instead of when the executable is started via the PE import section. Below we can see the function that is used before each API call, to get the address of it just before the call itself.

[28]https://techcrunch.com/2021/10/12/olympus-confirms-us-cyberattack-weeks-after-blackmatter-ransomware-hit-emea-systems/

[29] https://www.bleepingcomputer.com/news/security/sinclair-tv-stations-crippled-by-weekend-ransomware-attack/

[30]https://www.bleepingcomputer.com/news/security/evil-corp-demands-40-million-in-new-macaw-ransomware-attacks/

```
void *__stdcall f_get_api_by_hash(context *context, uint32_t fix_api_info_offset_value, uint32_t hash_value)
{
  api_info *api_info; // ecx
  uint32_t i_hash; // edi
  api_info **p_next; // ecx
  uint32_t fixed_hash; // edx
  void *result; // eax

  fixed_hash = context->fix_hash_value + hash_value;
  result = 0;
  for ( api_info = (sizeof(api_info) * fixed_hash + fix_api_info_offset_value + 8); api_info; api_info = *p_next )
  {
    i_hash = api_info->hash;
    if ( fixed_hash == i_hash )
    {
      result = (api_info->api_offset - context->fix_api_offset_value);
      break;
    }
    if ( fixed_hash <= i_hash )
      p_next = &api_info->next_bigger;
    else
      p_next = &api_info->next_smaller;
  }
  nullsub_136(context, fix_api_info_offset_value, hash_value);
  return result;
}
00120B56 f_get_api_by_hash:1 (1123B56)
```

Fig 20: The new function in Macaw which is used to dynamically fetch addresses for the different external API used by it

The function gets a 32-bit value that uniquely represents the required API and searches for it through a data structure created beforehand. The data structure can be described as an array with small binary search trees in each of its entries.

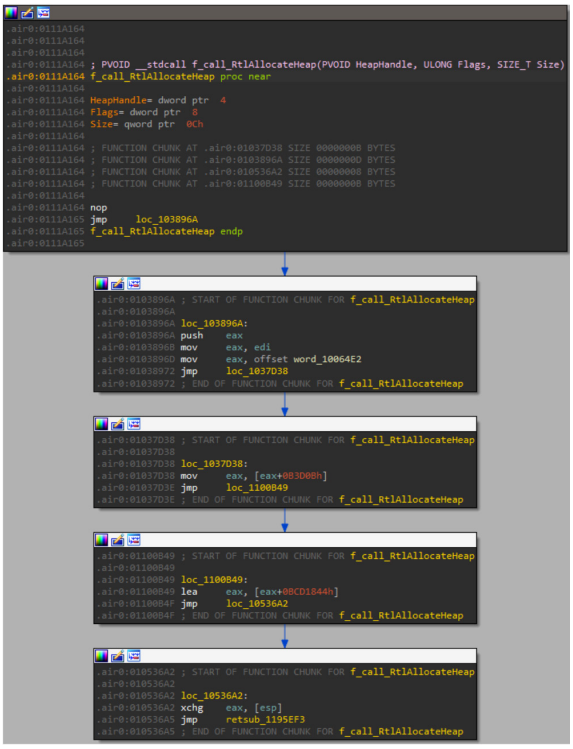


Fig 21: One example of indirect calls used in Macaw just to invoke a call to RtlAllocateHeap, calculating the destination address in EAX, exchanging the resulting value with the top of the stack and jumping to it using a “ret” instruction

At this point we tried to assess the similarity of two core functions (ex: between Hades and Macaw). In both strains (Hades / Macaw), the implementation is still the same. The only minor differences are from the imports fetched at runtime (so in Hades the color is magenta in the picture below, because the import is a load time one, specified in the PE import table. and in Macaw, the color is not magenta, but it still uses the same API, just retrieved at runtime).

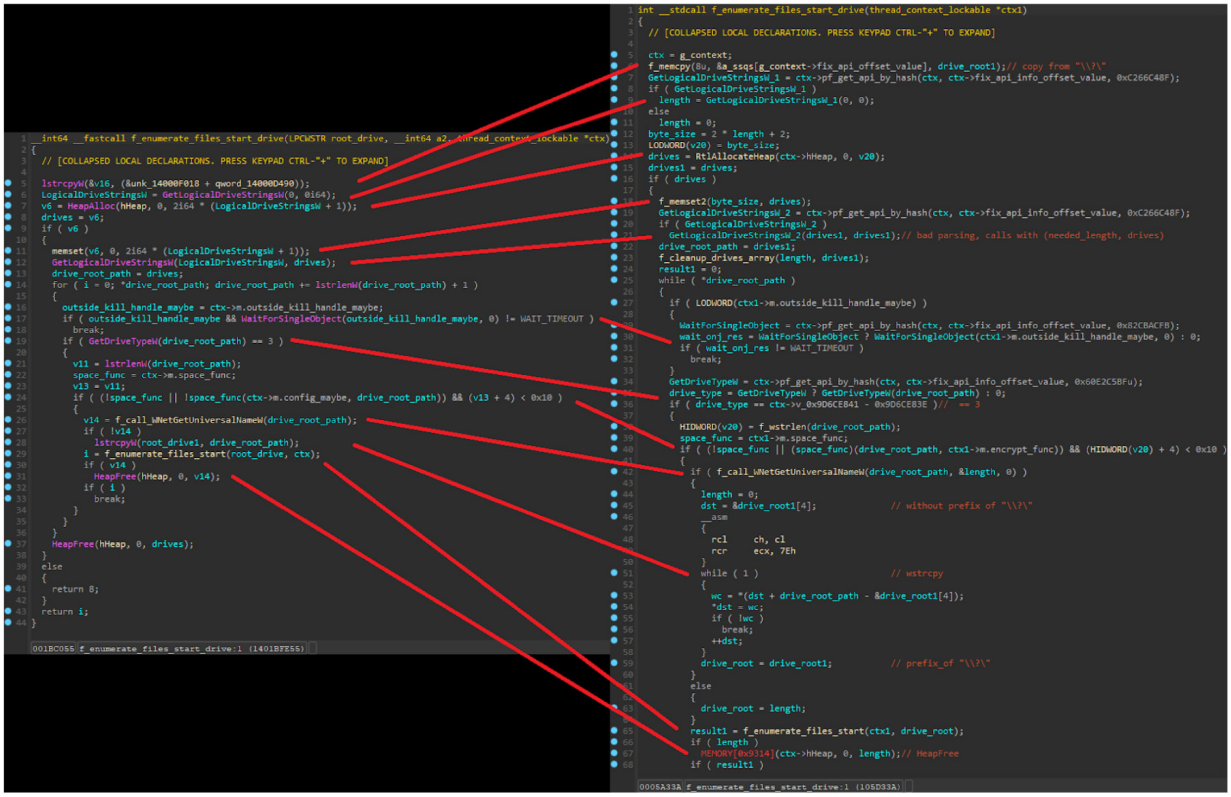


Fig 22: The drive enumeration function in both Hades (left) and Macaw (right) is partially identical

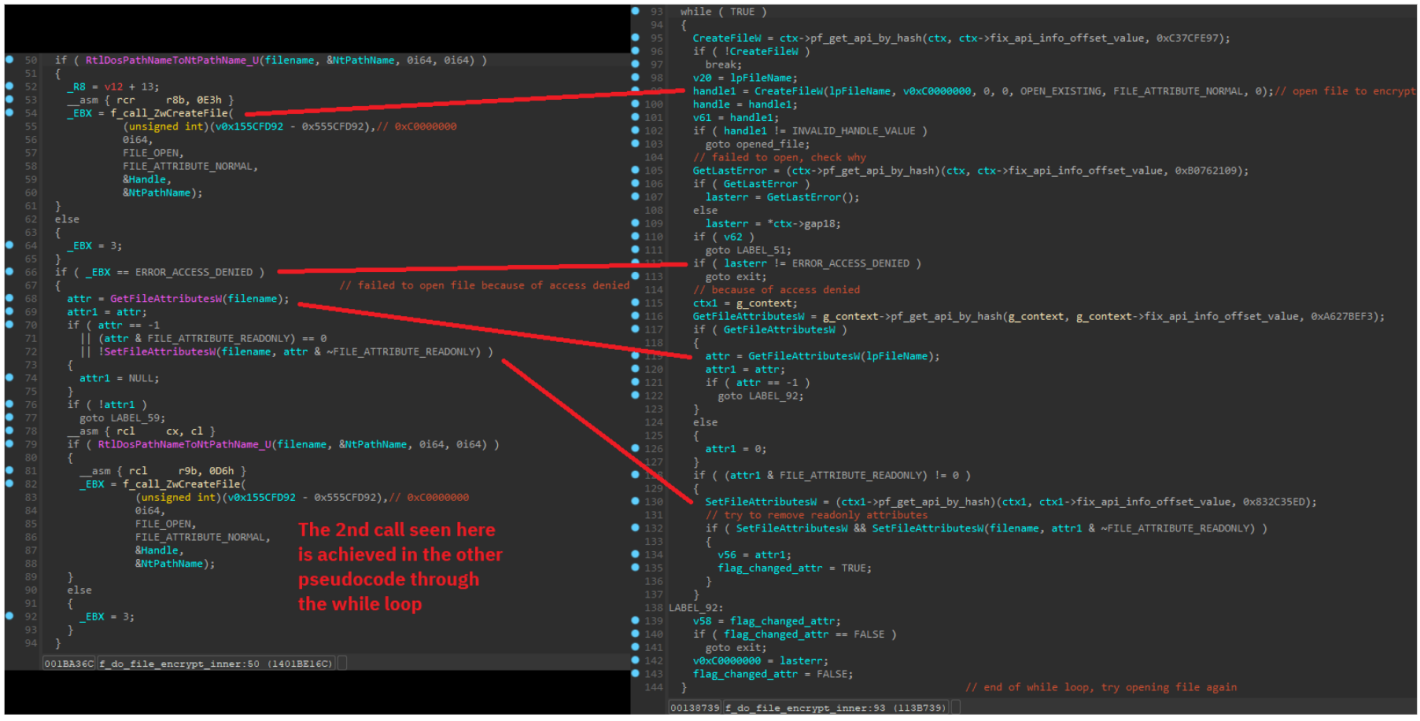
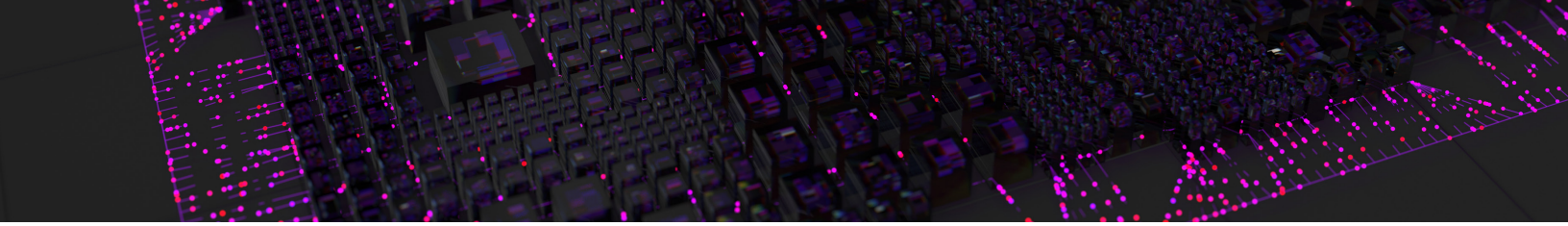


Fig 23: Another example from the beginning of the encryption function in both Hades (left) and Macaw (right) doing the same operations



CONCLUSION

It emerges from the analysis of the Evil Corp's TTPs over time that this threat actor put much of its effort into defense evasion. Apart from the code overlaps that we found in Macaw, the new implementation fully reflects this MO. The signature is sufficiently different to deflect attribution and avoid simple detection efforts. While the indirect API calls and inserted NOP instructions are intended to hinder analysis, our investigation reveals that Macaw is derived from the same codebase as Hades and other Evil Corp samples.

We provided a partial assessment of some functionality of the Macaw variant. We found strong overlaps in the file enumeration and in the encryption routine with Hades, also the same usage of adding junk code which constitutes a distinctive element.

This new attempt perfectly fits the observed EC TTPs and is also consistent with the timeline considering that since PayloadBIN, four months have passed and they had all the time to plan a new operation and change the obfuscation. This time it seems that they used VMProtect, so despite removing the original signatures (like .vmp0 and .vmp1 PE sections) some features and the style of code seems related to this protector.

Based on our assessment we believe that it is highly likely that this strain is derived from the same codebase and belongs to the same EC lineage.

CRYPTONE: THE PACKER

CryptOne (also known as *HellowinPacker*) is a special packer in use by Evil Corp up till mid-2021.

CryptOne appears to have first been noticed in 2015[31]. Early versions were used by an assortment of different malware families such as Gozi, Dridex, and Zloader. In 2019, Bromium[32] analyzed and reported it as in use by Emotet. In June 2020, NCC Group[33] reported that CryptOne was used to pack WastedLocker. In 2021, researchers[34] observed CryptOne being advertised as a Packer-as-a-Service on hacker forums.

CryptOne has the following characteristics and features:

- **Sandbox evasion** with getInputState() or GetKeyState() API;
- **Anti-emulation** with UCOMIEnumConnections and the IActiveScriptParseProcedure32 interface;
- **Code-flow obfuscation;**

In order to assist our research, we created a static unpacker we dub "de-CryptOne"[35]. This unpacker is capable of statically unpacking both x86 and x64 samples. It outputs two files: 1) the shellcode responsible for unpacking 2) the unpacked sample. We collected CryptOne packed samples, and with the use of the above tool, unpacked and categorized them at scale.

[31]https://blog.malwarebytes.com/threat-analysis/2015/12/malware-crypters-the-deceptive-first-layer/

[32]https://www.bromium.com/wp-content/uploads/2019/07/Bromium-Emotet-Technical-Analysis-Report.pdf

[33]https://blog.fox-it.com/2020/06/23/wastedlocker-a-new-ransomware-variant-developed-by-the-evil-corp-group/

[34]https://www.ptsecurity.com/ww-en/analytics/pt-esc-threat-intelligence/paas-or-how-hackers-evade-antivirus-software/

[35]https://github.com/Tera0017/de-CryptOne

A UNIQUE FACTORY

Hunting for CryptOne led us to identify different versions of this cryptor, which have never been reported previously. Each version is identified by a certain signature, listed below:

- 111111111\\{aa5b6a80-b834-11d0-932f-00a0c90dcaa9}
- 1nterfacE\\{b196b287-bab4-101a-b69c-00aa00341d07}
- 444erfacE\\{b196b287-bab4-101a-b69c-00aa00341d07}
- 555erfacE\\{b196b287-bab4-101a-b69c-00aa00341d07}
- 5nterfacE\\{b196b287-bab4-101a-b69c-00aa00341d07}
- 987erfacE\\{b196b287-bab4-101a-b69c-00aa00341d07}
- Interfac4\\{b196b287-bab4-101a-b69c-00aa00341d07}
- InterfacE\\{b196b287-bab4-101a-b69c-00aa00341d07}
- aaaerfacE\\{b196b287-bab4-101a-b69c-00aa00341d07}
- interfacE\\{b196b287-bab4-101a-b69c-00aa00341d07}
- rrrerfacE\\{b196b287-bab4-101a-b69c-00aa00341d07}

The first part of the string is composed of a custom string (111111111, 1nterfacE, 444erfacE, ...) which then is replaced at runtime by the 'interface' keyword, creating the following registry key:

```
HKEY_CLASSES_ROOT\interface\{b196b287-bab4-101a-b69c-00aa00341d07}
HKEY_CLASSES_ROOT\interface\{aa5b6a80-b834-11d0-932f-00a0c90dcaa9}
```

Those registry keys are related to the UCOMIEnumConnections and IActiveScriptParseProcedure32 interfaces respectively. Once executed, the Cryptor checks for the presence of those keys before loading the next stage payload. If it does not find the keys, then the malware goes in an endless loop without doing anything as an anti-emulation technique. This works because some emulators do not implement the full Windows registry.

In reviewing two different versions of CryptOne:
aaerfacE\\{b196b287-bab4-101a-b69c-00aa00341d07} and
111111111\\{aa5b6a80-b834-11d0-932f-00a0c90dcaa9}, we noticed that in order to update the signature, the actor needs to re-compile the cryptor as the cryptor implementation changes.

```
mov     [ebp+var_8], eax
mov     eax, 69h ; 'i'
mov     ecx, off_531D6C ; "aaaerfacE\\{b196b287-bab4-101a-b69c-00a"...
mov     [ecx], ax
mov     edx, 6Eh ; 'n'
mov     eax, off_531D6C ; "aaaerfacE\\{b196b287-bab4-101a-b69c-00a"...
mov     [eax+2], dx
mov     ecx, 74h ; 't'
mov     edx, off_531D6C ; "aaaerfacE\\{b196b287-bab4-101a-b69c-00a"...
mov     [edx+4], cx
push    offset dword_532574
mov     eax, off_531D6C ; "aaaerfacE\\{b196b287-bab4-101a-b69c-00a"...
push    eax
```

Fig 25: aaaerfacE\\{b196b287-bab4-101a-b69c-00aa00341d07} implementation

```
mov     cs:dword_140037EB4, eax
mov     ecx, 69h ; 'i'
mov     rax, cs:off_140037EB8 ; "111111111\\{aa5b6a80-b834-11d0-932f-00a"...
mov     [rax], cx
mov     ecx, 6Eh ; 'n'
mov     rax, cs:off_140037EB8 ; "111111111\\{aa5b6a80-b834-11d0-932f-00a"...
mov     [rax+2], cx
mov     ecx, 74h ; 't'
mov     rax, cs:off_140037EB8 ; "111111111\\{aa5b6a80-b834-11d0-932f-00a"...
mov     [rax+4], cx
mov     ecx, 65h ; 'e'
mov     rax, cs:off_140037EB8 ; "111111111\\{aa5b6a80-b834-11d0-932f-00a"...
mov     [rax+6], cx
mov     ecx, 72h ; 'r'
mov     rax, cs:off_140037EB8 ; "111111111\\{aa5b6a80-b834-11d0-932f-00a"...
mov     [rax+8], cx
mov     ecx, 66h ; 'f'
mov     rax, cs:off_140037EB8 ; "111111111\\{aa5b6a80-b834-11d0-932f-00a"...
mov     [rax+0Ah], cx
mov     ecx, 61h ; 'a'
mov     rax, cs:off_140037EB8 ; "111111111\\{aa5b6a80-b834-11d0-932f-00a"...
mov     [rax+0Ch], cx
mov     ecx, 63h ; 'c'
mov     rax, cs:off_140037EB8 ; "111111111\\{aa5b6a80-b834-11d0-932f-00a"...
mov     [rax+0Eh], cx
mov     ecx, 65h ; 'e'
mov     rax, cs:off_140037EB8 ; "111111111\\{aa5b6a80-b834-11d0-932f-00a"...
mov     [rax+10h], cx
lea     r8, qword_140038278
mov     rdx, cs:off_140037EB8 ; "111111111\\{aa5b6a80-b834-11d0-932f-00a"...
mov     ecx, cs:dword_140037EB4
call    cs:qword_140038000
```

Fig 26: 111111111\\{aa5b6a80-b834-11d0-932f-00a0c90dcaa9} implementation

TEMPORAL ANALYSIS

Compilation timestamps are not trustworthy by nature because they can easily be altered. We cannot confirm if they were altered or not but we have made some assumptions based on our observations which we detail below.

Please note: those observations are based on a small number of samples, collected and validated.

OBSERVATION #1 - SEQUENTIAL CONSISTENCY:

The first observation is that for all the packed artifacts we collected, we noticed a gap between the compilation time and packing time (with CryptOne). The two operations are sequential and happened at two different moments in time:

Family Name	Sha1	Packing Timestamp	Payload Timestamp
WastedLocker	91b2bf44b1f9282c09f07 f16631deaa3ad9d956d	2020-06-11 04:10:48	2020-05-26 17:46:34
WastedLocker	9292fa66c917bfa47e801 2d302a69bec48e9b98c	2020-06-11 04:10:48	2020-05-26 17:46:34
WastedLocker	c40c8848808da12ef78c6 8de1e6477b862161a43	2020-06-29 22:17:38	2020-06-27 14:15:38
WastedLocker	f25f0b369a355f30f5e11 ac11a7f644bcfefd963	2020-05-31 09:38:36	2020-05-26 17:46:34
Hades	b4061d4227e08cfaa319 0dea9926571fca2736a1	2021-03-06 18:13:45	2020-12-14 09:34:38
Hades	7bcea3fbfcb4c170c57c9 050499e1fae40f5d731	2020-12-20 21:59:21	2020-12-14 09:34:38
Hades	b4061d4227e08cfaa319 0dea9926571fca2736a1	2021-03-06 18:13:45	2020-12-14 09:34:38
PhoenixLocker	3cb0cb07cc2542f1d9806 0adccda726ea865db98	2021-03-20 20:00:50	2020-12-14 09:34:38
PayloadBin	e23637ea81751e558fca 17ef1a54b6e39d2e83c3	2021-06-02 19:12:49	2021-04-28 18:04:49

Of note, the packing timestamp is always later than the payload timestamp. This observation led us to conclude that even if the compilation timestamp of the real payload was altered (voluntarily or not), it is consistent with the logical order of events.

OBSERVATION #2 - CONSISTENCY WITH THE TIMING OF THE OPERATIONS:

The second observation comes from the timing of the operations. The timestamps of the packed artifacts fall inside the timeframe of the operations. This means that, even if the timestamp was altered in some way, we can assume it is trustworthy because it is consistent with the timeline of operations and permits us to place events in chronological order.

Please note: We define the start and end dates for each period of activity broadly, based primarily on OSINT and the publicly reported dates of incidents.

Family Name	Start of the Operations	End of the Operations
WastedLocker	May 2020[36]	July 2020[37]
Hades	Dec 2020[38]	March 2021[39 40]
PhoenixLocker	2021-03-26	March 2021
PayloadBin	2021-06-03[41]	2021-06-03[42]

We correlated the above information with the different CryptOne signature timestamps. This forms an important observable for our analysis and allows us to construct a more reliable timeline.

[36]https://research.nccgroup.com/2020/06/23/wastedlocker-a-new-ransomware-variant-developed-by-the-evil-corp-group/
[37]https://www.vmray.com/cyber-security-blog/wastedlocker-ransomware-threat-bulletin/
[38]ttps://awakesecurity.com/blog/incident-response-hades-ransomware-gang-or-hafnium/
[39]https://www.accenture.com/us-en/blogs/security/ransomware-hades
[40]https://www.accenture.com/us-en/blogs/cyber-defense/unknown-threat-group-using-hades-ransomware
[41]based on VT submission date
[42]based on VT submission date

OBSERVATION #3 - STATISTICAL AND CHRONOLOGICAL CONSISTENCY

Another observation comes from the relation between the packing time and VT 'first seen' field. The VT 'first submission' tells us the timestamp in which a given sample was submitted. The results are summarized in the following table (chronological order):

Family Name	Sha1	Sample ID	Packing Timestamp	First Submission	CryptOne Signature
WastedLocker	763d356d30e81d1cd15f6bc6a31f96181edb0b8f	1	2020-05-29 19:12:51	2020-06-01 21:22:29	InterfacE
	f25f0b369a355f30f5e11ac11a7f644bcfefd963	2	2020-05-31 09:38:36	2020-06-25 20:04:51	InterfacE
	91b2bf44b1f9282c09f07f16631deaa3ad9d956d	3	2020-06-11 04:10:48	2020-06-15 21:51:57	InterfacE
	9292fa66c917bfa47e8012d302a69bec48e9b98c	4	2020-06-11 19:20:31	2020-06-17 03:31:41	aaaerfacE
	e7784e31dfd1a6f6291bca78bf11b6145d37d753	5	2020-06-11 04:10:48	2021-04-20 03:23:20	InterfacE
	6f4195bbbe4e2319e498039b491a4d976c11de39	6	2020-07-22 18:43:17	2020-07-26 16:07:29	aaaerfacE
	735ee2c15c0b7172f65d39f0fd33b9186ee69653	7	2020-07-22 18:43:17	2020-07-23 05:55:35	aaaerfacE
Hades	d0d68281f8459b5558559fbbf8c6c8ab4ddfec8b	8	2020-12-20 21:59:21	2020-12-22 12:15:52	111111111
	f8fc84030c579070b36c99c836ac4b5c32bbc2c4	9	2020-12-20 21:54:17	2020-12-21 17:22:06	111111111
	7bcea3fbfcb4c170c57c9050499e1fae40f5d731	10	2020-12-20 21:59:21	2021-01-07 06:56:0	111111111
	b4061d4227e08cfaa3190dea9926571fca2736a1	11	2021-03-06 18:13:45	2021-03-26 16:55:45	111111111
	f8e52380b6f3668d4de6df416c8da389c0d98fe8	12	2021-03-06 05:26:01	2021-03-26 17:46:55	111111111
PhoenixLocker	3cb0cb07cc2542f1d98060adccda726ea865db98	13	2021-03-20 20:00:50	2021-03-26 17:48:41	111111111
PayloadBIN	e23637ea81751e558fca17ef1a54b6e39d2e83c3	14	2021-06-02 19:12:49	2021-06-03 13:35:16	111111111

We plotted all these timestamps in a time series graph below.

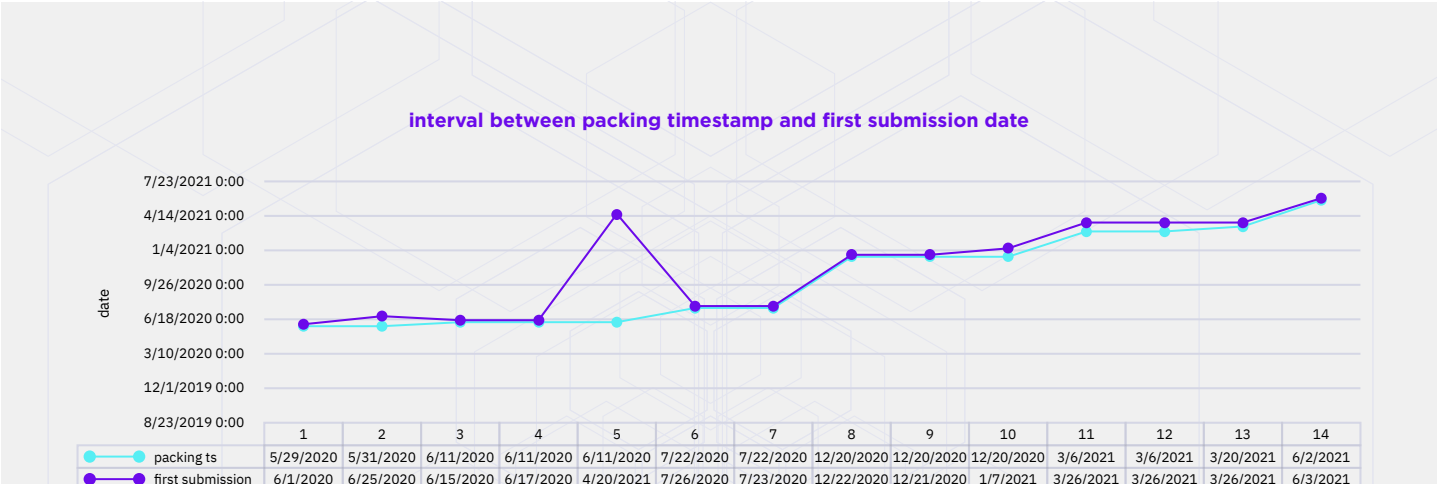


Fig 27

From this time series analysis, it is evident that each sample was submitted (orange line) after being packed (blue line), most of the time a few hours later (both lines are too close). In any case, the timestamps are close enough to support the assumption that the compilation was not altered.

OBSERVATION #4 - CONSISTENCY WITH DIGITAL SIGNATURE DATE

Most of the samples mentioned above were digitally signed, and in all those cases, the certificate was issued after the packing date. This is a strong indication that the compilation timestamp of the packed artifact was not modified. For example, the sample: 763d356d30e81d1cd15f6bc6a31f96181edb0b8f was packed on 2020-05-29 19:12:51 and signed 1 minute later on 2020-05-29 19:14:00. The certificate was issued on 05/23/2020, 6 days before the packing and signing date:

> *Name: SCSTXPBIMRJPFWKHAA*
> *Issuer: SCSTXPBIMRJPFWKHAA*
> *Valid From: 07:51 PM 05/23/2020*
> *Valid To: 11:59 PM 12/31/2039*
> *Thumbprint: 1D65057DD11CF6218FB9A425B6AC31E3C58DD508*
> *Serial Number: 36 5F 7C AB E7 8B E8 BD 47 FF 30 C6 A9 36 29 51*

THREATS TO VALIDITY (FOR COMPILATION TIMESTAMPS):

Observation	Validity	Threats to Validity
#1	**Weak**. Could be tampered with.	A TA could modify the compilation timestamp even if they keep sequential order in packing and compile time.
#2	**Strong.** Could be tampered with but, even if the timestamps could be tampered with, it falls into the timing of the operations.	Even if a TA could modify the compilation timestamp, it remain consistent
#3	**Strong.** The submission date cannot be altered. In any case, the submission date is next and subsequent to the compilation timestamp	Even if a TA could modify the compilation timestamp, it remains valid.
#4	**Strong**. The certificate was revoked but the release date can be verified.	Even if a TA could modify compilation timestamps, it is consistent and aligned with the signing date.

Based on the above observations and their related threats to validity, we assess it is likely the compilation timestamps were not altered.

CRYPTONE TIMELINE

From our analysis of collected samples, *it seems that from a specific point in time, around September 2020*, Hades, PhoenixLocker and PayloadBIN started adopting a specific signature identified by the following signature 111111111\\{aa5b6a80-b834-11d0-932f-00a0c90dcaa9}.

We developed some YARA rules and started collecting samples. In the following table, the various CryptOne versions, with their associated malware and inherent time-frame are summarized (from the oldest to the most recent):

CryptOne version	Associated Malware families packed	# of samples	Timeframe of utilization (taken from the packing ts)	Architecture
Interfac4	Netwalker, Betabot	15	12/10/2019 - 1/19/2020	x86
InterfacE	gozi_rm3(version:3.00 build:854), Dridex(10121), WastedLocker, Netwalker	366	3/1/2020 - 6/5/2020	x86
1nterfacE	Dridex(Botnet 10121), gozi_rm3(version:3.00 build:898, version:3.00 build:900, version:3.00 build:904)	823	5/20/2020 - 6/8/2020	x86
aaaerfacE	WastedLocker	15	5/26/2020 - 7/27/2020	x86
5nterfacE	gozi_rm3(version:3.00 build:904)	340	6/12/2020 - 6/12/2020	x86
444erfacE	Amadey	1	6/25/2020	x86
987erfacE	Dridex(Botnet 10111)	334	6/29/2020 - 6/30/2020	x86
rrrerfacE	Formgrabber	5	6/30/2020	x86
555erfacE	Avaddon, TaurusStealer, Zloader	272	7/19/2020 - 7/29/2020	x86
111111111	CS, Hades, Phoenix, PayloadBIN	49	10/6/2020 - 7/20/2021	x64

INTERESTING OBSERVATIONS

There are some interesting congruences that come from the observation of this timeline. Netwalker started to use the service on 12/10/2019 and remained a customer until the second version (March/April 2020). There is a realistic possibility that EC started being a customer of the CryptOne service from 3/1/2020. In fact, within the timeframe 3/1/2020 - 6/5/2020 we found WastedLocker, gozi_rm3(version:3.00 build:854) and Dridex(10121) samples were all packed and compiled in the same timeframe using the same CryptOne signature (InterfacE).

CryptOne version	Associated Malware families packed	# of samples	Timeframe of utilization (taken from the packing ts)	Architecture
InterfacE	gozi_rm3(version:3.00 build:854), Dridex(10121), WastedLocker, Netwalker	366	3/1/2020 - 6/5/2020	x86

Another interesting coincidence is that, from December 2020, the CryptOne version '111111111' appeared in the wild without any overlap. Until December 2020, we only observed one version in the wild. For a limited period of time between May 2020 and August 2020, we observed different versions of CryptOne overlaps, seen below:

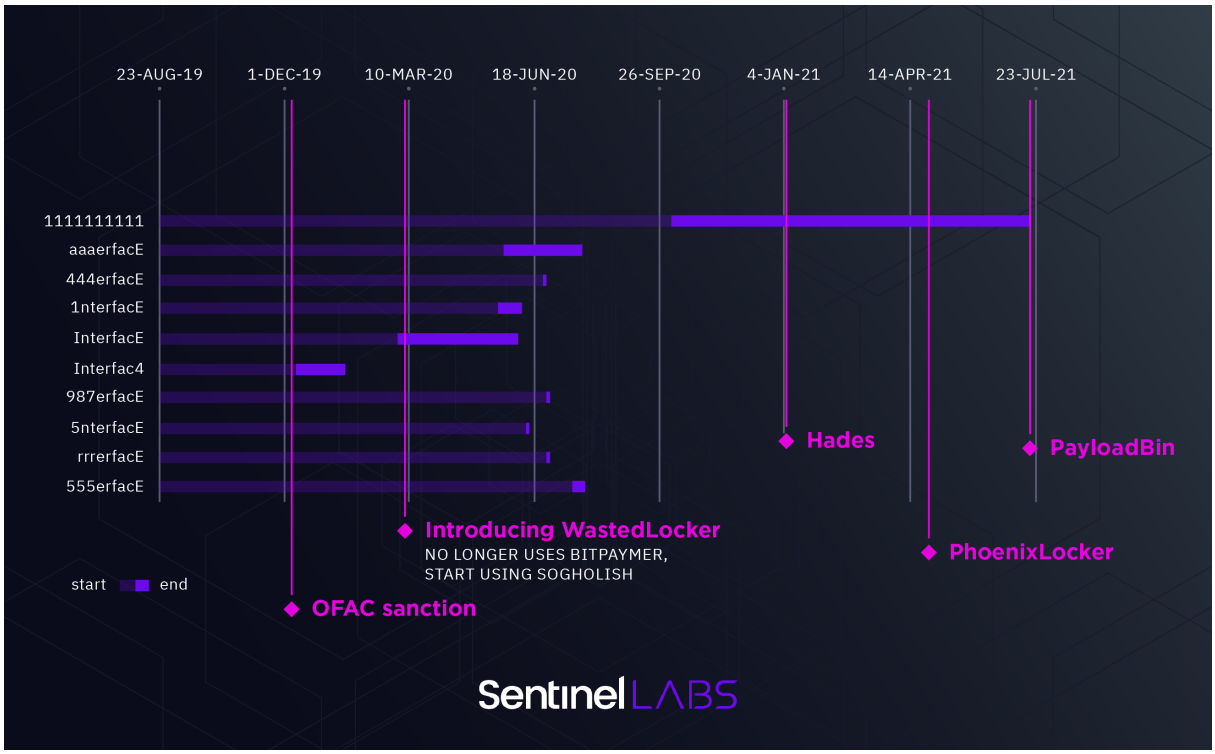


Fig 28

INFRASTRUCTURE OVERLAPS

In this section we cluster infrastructure shared by the different ransomware families that could be relevant in the EC ecosystem.

WASTEDLOCKER - HADES

An interesting intersection was reported by the security firm Truesec[43] and it's related to the IP address 185[.]82.127.86, which was used in a WastedLocker incident in October 2020 and was later reported to be a Hades C2[44].

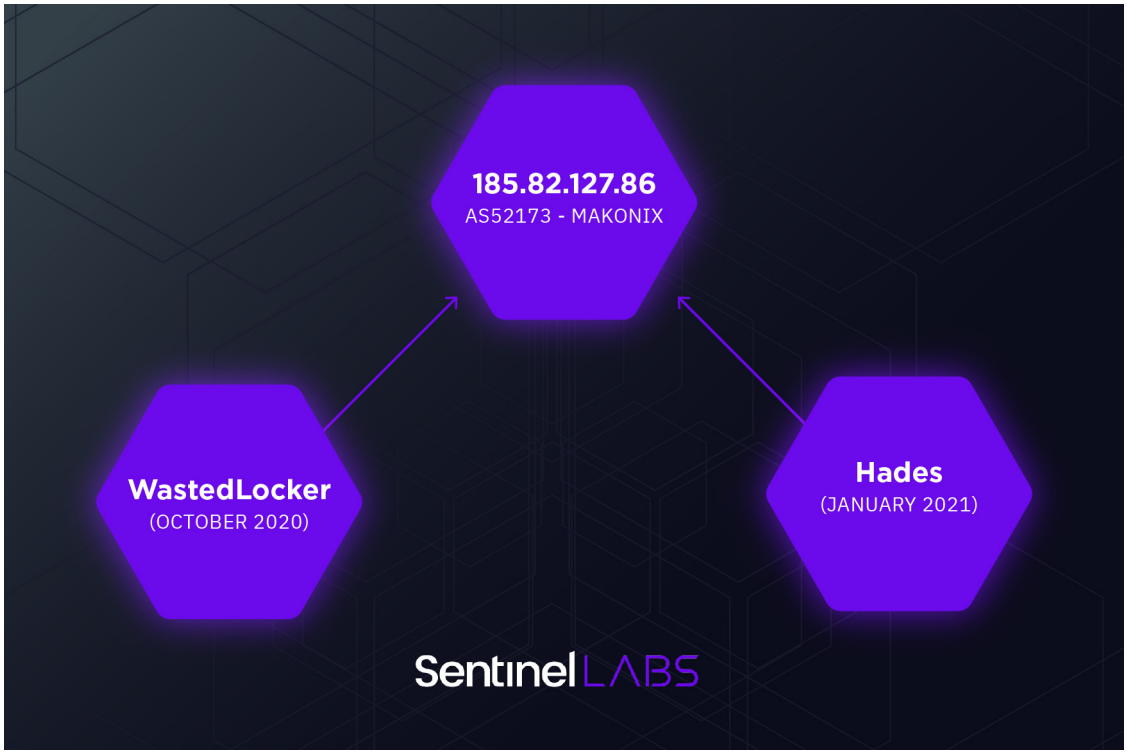


Fig 29

[43]https://blog.truesec.com/2021/05/05/are-the-notorious-cyber-criminals-evil-corp-actually-russian-spies/
[44]https://www.abuseipdb.com/check/185.82.127.86

WASTEDLOCKER-HADES SHARES THE SAME SOCGHOLISH INFRASTRUCTURE

Another interesting IP is *130.0.233[.]178* (ASN: 15626, ITLAS) which resolved to **.refinedbewbs[.]com* it has been reported to be part of SocGholish infrastructure

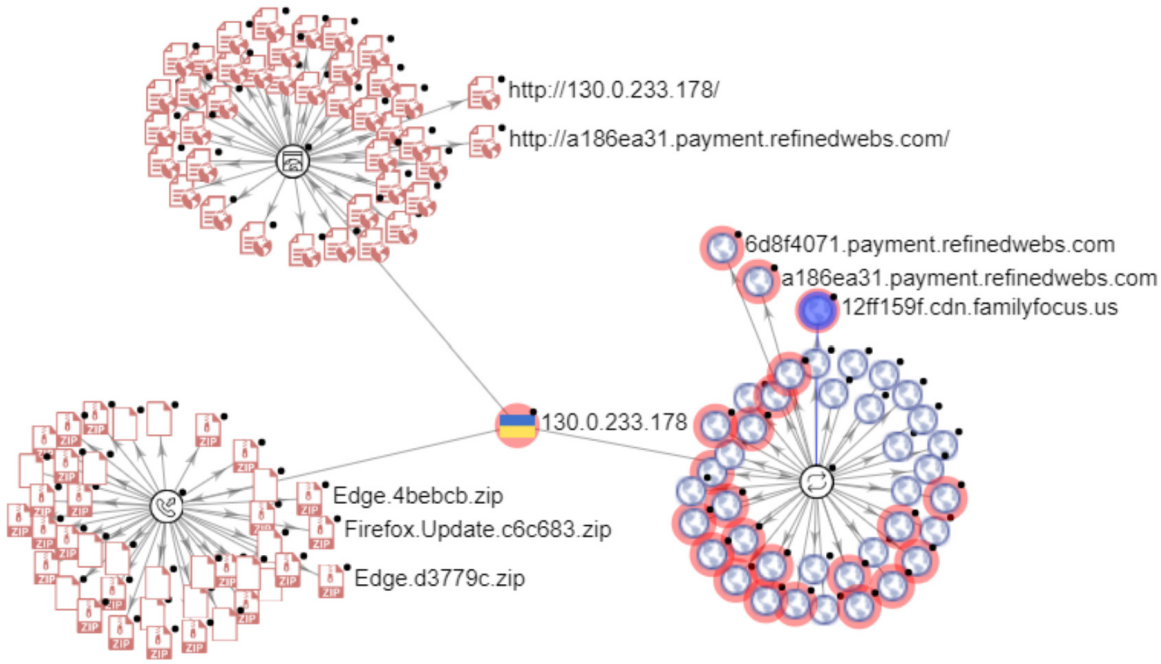


Fig 30

This IP was used in 2019 to spread BitPaymer and DoppelPaymer by deploying Dridex[45] loader. A few months later it was reported to have also spread WastedLocker[46] and Hades Ransomware[47].

WASTEDLOCKER - GOZI

We found one C2, devicelease[.[xyz, associated with the Gozi sample (version:3.00, build: 854) sha1:c3154048ac74ceac75fdc62820ef66f1bdb31334 and packed with CryptOne was also reported to be linked to WastedLocker[48][49].

---

[45]https://www.fireeye.com/blog/threat-research/2019/10/head-fake-tackling-disruptive-ransomware-attacks.html

[46]https://www.darktrace.com/en/blog/how-ai-stopped-a-wasted-locker-intrusion-before-ransomware-deployed/

[47]https://www.secureworks.com/blog/hades-ransomware-operators-use-distinctive-tactics-and-infrastructure

[48]hhttps://www.securonix.com/securonix-threat-research-detecting-wastedlocker-ransomware-using-security-analytics/

[49]https://answers.microsoft.com/en-us/windows/forum/all/block-or-avoid-wastedlocker-ransomeware-detected/fcbe6856-6139-4aa0-b997-0f5dbccfdbb8

CONCLUSIONS

Evil Corp (“EC”, a.k.a “Indrik Spider”), is an advanced cybercrime activity cluster originating from Russia and has been active since 2007. It is considered to be one of the ‘more problematic topics in the threat intelligence community. The documented ties with the Russian FSB, the US indictments and the OFAC sanctions are probably the main reasons behind such controversy. Clustering EC activity is demonstrably difficult considering that the group has changed TTPs several times in order to bypass sanctions and stay under the radar. In this research we connect the dots in the EC ecosystem, cluster malware, document EC’ activities and provide insight into their TTPs.

First, we assess it is almost certain that EC’s ransomware variants were all developed by the same ‘factory’, as there is significant code reuse between all the ransomware variants. Our temporal analysis demonstrated that all the operations are tightly connected and synchronized; we assess it is almost certain there is a single entity which owns the entire attack chain, from resource development to ransomware deployment. Whether the group operates a RaaS model for a trusted number of affiliates is still an open question.

EC’s TTPs demonstrate their advanced capability to adapt and persist in victim environments. From the ransomware perspective, they implemented specific techniques to evade EDR and lower detection. The partial rewrite in native API and the custom implementation of polymorphism - relying on the insertion of a wide range of no-op-codes in Hades malware - are clear elements of their capability, not commonly found in the cybercrime threat landscape. From a detection standpoint, it would be fruitful to improve detection of defense evasion TTPs in order to reduce blind spots.

From a strategic standpoint, EC poses a serious threat to organizations due to their capabilities: the ability to maintain a high level of operational security, to evade detection, and last but not least the ability to ‘change their skin’ (ransomware signature, TTPs) in a very short period of time. All these elements provide a solid understanding of the modus operandi of this group and their capabilities. The adoption of SocGholish as an initial access tool and Koadic as a fileless post exploitation framework, further demonstrates how this cluster relies on uncommon tools to stay undetected.Victimology remains an open point due to the lack of shared intelligence. This is also a side effect of the OFAC sanctions, which presents unique challenges for incident response firms involved in such cases. Public attribution to EC is a sensitive matter for the aforementioned reasons. Moreover, EC’s operators have adopted victim anonymity since the deployment of Hades ransomware, removing victim information from the ransom note. This lowers the chances of organizations being discovered if they have fallen victim to EC.

The alleged ties with the Russian government, the low volume of operations, the high level of operational security place this threat actor in a somewhat unique, hybrid position between espionage and cybercrime. This hybrid position is exemplified by the lacklustre monetization strategy of the group, suggesting motives that go beyond financial gain. The group's main motivations present the most important open question for the intelligence community. Some speculations claim EC is conducting espionage, masquerading as financial crime through extortion. Some speculate that EC receives portions of a Russian black budget to finance activities that cannot directly link back to State activity. Nevertheless, continued research on the group needs to continue to elicit the motivations of the threat group.

Historically, EC presents two distinct modus operandi: botnet operation to conduct mass spam campaigns primarily through Dridex (and derivatives); and since 2017, Big Game Hunting in low-volume ransomware operations using BitPaymer. These 2 distinct tactics they have practiced over time demonstrate the level of sophistication and adaptability of this actor, a relatively unique characteristic of a cybercrime threat actor.

We cannot be certain of exactly how EC will continue to evolve and target organizations. However, we assess it is likely they will continue to advance their tradecraft, finding new methods of evading detection and misleading attribution. SentinelLabs will continue tracking this activity cluster to provide insight into their evolution.

MITRE ATT&CK TTPS OBSERVED

Tactic	Technique	Procedure/Comments
Resource Development	T1584.004 - Compromise Infrastructure: Server[50]	Adversaries have compromised third-party servers to deploy SocGholish.
Initial Access	T1189 - Drive-by Compromise[51]	Adversaries may gain access to a system through a user visiting a website over the normal course of browsing. Part of the SocGholish infection chain.
Execution	T1047 - Windows Management Instrumentation[52]	Lateral movement is attempted by using WMIC to create malicious processes on other nodes within the domain. This technique has been observed in the SocGholish attack chain, priori to ransomware deployment
Execution	T1059.001 - Command and Scripting Interpreter: PowerShell[53]	EvilCorp has used PowerShell Empire for execution of malware.
Execution	T1106 - Native API[54]	Ransomware has used dynamic API resolution and native API to avoid identifiable strings within the binary.
Privilege escalation	T1548.002 - Abuse Elevation Control Mechanism: Bypass User Account Control[55]	Adversaries bypass UAC mechanisms to elevate process privileges on the system to deploy their ransomware..

[50]https://attack.mitre.org/techniques/T1584/004/
[51]https://attack.mitre.org/techniques/T1189/
[52]https://attack.mitre.org/techniques/T1047/
[53]https://attack.mitre.org/groups/G0119/
[54]https://attack.mitre.org/techniques/T1106/
[55]https://attack.mitre.org/techniques/T1548/002/

Tactic	Technique	Procedure/Comments
Privilege Escalation	T1134.001 - Access Token Manipulation: Token Impersonation/Theft[56]	Ransomware can use the tokens of users to create processes on infected systems.
Defense Evasion	T1562.001 -Impair Defenses: Disable or Modify Tools[57]	Observed to disable security solutions prior to deploying a Ransomware
Defense Evasion	T1027.004 - Obfuscated Files or Information: Compile After Delivery[58]	This technique has been observed in the SocGholish attack chain, priori to ransomware deployment
Defense Evasion	T1027.001 - Obfuscated Files or Information: Binary Padding[59]	Adversaries used binary padding to add junk data and change the on-disk representation of malware. This was used in Phoenix Locker and PayloadBin to lower the detection.
Defense Evasion	T1218 - Signed Binary Proxy Execution[60]	Adversaries bypass process and/or signature-based defenses by proxying execution of malicious content with signed binaries.
Defense Evasion	T1497.001 - Virtualization/Sandbox Evasion: System Checks[61]	Adversaries employ system checks to detect and avoid virtualization and analysis environments.
Defense Evasion	T1027.001 - Obfuscated Files or Information: Software Packing[62]	Usage of CryptOne PaaS to obfuscate artifacts.
Defense Evasion	T1027.005 - Obfuscated Files or Information: Indicator Removal from Tools[63]	They can modify the tool by removing the indicator and using the updated version that is no longer detected by the target's defensive systems or subsequent targets that may use similar systems.

[56]https://attack.mitre.org/techniques/T1134/001/
[57]https://attack.mitre.org/techniques/T1562/001/dri
[58]https://attack.mitre.org/techniques/T1027/004/
[59]https://attack.mitre.org/techniques/T1027/001/
[60]https://attack.mitre.org/techniques/T1218/
[61]https://attack.mitre.org/techniques/T1497/001/
[62]https://attack.mitre.org/techniques/T1027/002/
[63]https://attack.mitre.org/techniques/T1027/005/

Tactic	Technique	Procedure/Comments
Defense Evasion	T1564 - Hide Artifacts[64]	BitPaymer and WastedLocker rely on ADS to hide itself from the system and evade detection.
Defense Evasion	T1553.002 - Subvert Trust Controls: Code Signing[65]	Adversaries acquired signing materials to sign their malware/ransomware
Defense Evasion	T1480 - Execution Guardrails[66]	Ransomware compares file names and paths to a list of excluded names and directory names during encryption.
Defense Evasion	T1036.005 - Masquerading: Match Legitimate Name or Location[67]	Adversaries used fake updates for FlashPlayer plugin and Google Chrome as initial infection vectors.
Credential Access	T1558.003 - Steal or Forge Kerberos Tickets: Kerberoasting[68]	Kerberoasting is a common attack against Domain Controllers in which the attacker extracts credential hashes that can then be cracked offline via brute forcing. This technique has been observed in the SocGholish attack chain, priori to ransomware deployment.
Discovery	T1087.002 Account Discovery: Domain Account[69]	This technique has been observed in the SocGholish attack chain, priori to ransomware deployment.
Discovery	ID: T1482 - Domain Trust Discovery[70]	This technique has been observed in the SocGholish attack chain, priori to ransomware deployment.

[64]https://attack.mitre.org/techniques/T1564/
[65]https://attack.mitre.org/techniques/T1553/002/
[66]https://attack.mitre.org/techniques/T1480/
[67]https://attack.mitre.org/techniques/T1036/005/
[68]https://attack.mitre.org/techniques/T1558/003/
[69]https://attack.mitre.org/techniques/T1087/002/
[70]https://attack.mitre.org/techniques/T1482/

Tactic	Technique	Procedure/Comments
Discovery	T1069.002 - Permission Groups Discovery: Domain Groups[71]	This technique has been observed in the SocGholish attack chain, priori to ransomware deployment.
Discovery	T1083 - File and Directory Discovery[72]	This technique has been observed in the SocGholish attack chain, priori to ransomware deployment.
Discovery	T1135 - Network Share Discovery[73]	This technique has been observed in the SocGholish attack chain, priori to ransomware deployment.
Discovery	T1087.001 - Account Discovery: Local Account[74]	Ransomware can enumerate the sessions for each user logged onto the infected host.
Command and Control	T1105 - Ingress Tool Transfer[75]	Download a Cobalt Strike executable. This technique has been observed in the SocGholish attack chain, priori to ransomware deployment
Impact	T1486 - Data Encrypted for Impact[76]	Adversaries encrypt data on target systems or on large numbers of systems in a network with their ransomware.

[71]https://attack.mitre.org/techniques/T1069/002/
[72]https://attack.mitre.org/techniques/T1083/
[73]https://attack.mitre.org/techniques/T1135/
[74]https://attack.mitre.org/techniques/T1087/001/
[75]https://attack.mitre.org/techniques/T1105/
[76]https://attack.mitre.org/techniques/T1486/

YARA RULES

CryptOne:generic

```
rule CryptOne
{
  meta:
    Author = "@Tera0017/@SentinelOne"
    Family = "Evil Corp Packer-CryptOne"
  strings:
    $x86_code1 = {68 FC 4A 06 00 68 F4 E0 01 00 E8}
    $x86_code2 = {6A 15 E8 [4] 83 C4 04 A3 [4] 68 45 7E 00 00}
    $x86_code3 = {83 C4 08 8B 55 ?? 8B 45 ?? 8D 8C 10 [4] 89}
    $x64_code1 = {C7 ?? ?? ?? 05 0D 00 00}
    $x64_code2 = {48 03 44 24 48 48 03 44 24 48 48 03 44 24 48 48 03 44 24 48}
    $x64_code3 = {41 8D 84 03 ?? ?? 00 00}
    $str1 = "\\{aa5b6a80-b834-11d0-932f-00a0c90dcaa9}"
    $str2 = "\\{b196b287-bab4-101a-b69c-00aa00341d07}"
  condition:
    (all of ($x64*)) or (all of ($x86*)) or (any of ($str*) and (2 of ($x64*) or 2
of ($x86*)))
}
```

CryptOne 111111 version (Hades/ Phoenix/PayloadBIN

```
rule CryptONE_1111111Version
{
  meta:
    Author = "SentinelLabs"
    Family = "Evil Corp CryptOne"
  strings:
    $str1 = "111111111\\{aa5b6a80-b834-11d0-932f-00a0c90dcaa9}" wide ascii
    $str2 = "111111111\\{b196b287-bab4-101a-b69c-00aa00341d07}" wide ascii
  condition:
    any of them
}
```

SocGholish loaders

```
rule MAL_JS_SocGholish_Mar21_1 : js socgholish
{
  meta:
    description = "Triggers on SocGholish JS files"
    author = "Nils Kuhnert"
    date = "2021-03-29"
    hash = "7ccbdcde5a9b30f8b2b866a5ca173063dec7bc92034e7cf10e3eebff017f3c23"
    hash = "f6d738baea6802cbbb3ae63b39bf65fbd641a1f0d2f0c819a8c56f677b97bed1"
    hash = "c7372ffaf831ad963c0a9348beeaadb5e814ceeb878a0cc7709473343d63a51c"
  strings:
    $try = "try" ascii

    $s1 = "new ActiveXObject('Scripting.FileSystemObject');" ascii
    $s2 = "['DeleteFile']" ascii
    $s3 = "['WScript']['ScriptFullName']" ascii
    $s4 = "['WScript']['Sleep'](1000)" ascii
    $s5 = "new ActiveXObject('MSXML2.XMLHTTP')" ascii
    $s6 = "this['eval']" ascii
    $s7 = "String['fromCharCode']"
    $s8 = "2), 16)," ascii
    $s9 = "= 103," ascii
    $s10 = "'00000000'" ascii
  condition:
    $try in (0 .. 10) and filesize > 3KB and filesize < 5KB and 8 of ($s*)
}
```

Hades Ransomware section name .obX0	import "pe"
	
	rule hades_section_name
	{
	  meta:
	  Author = "SentinelLabs"
	  Family = "Evil Corp Hades"
	  condition:
	  (int16(0) == 0x5A4D) and (for any i in (0..pe.number_of_sections -1): (pe.
	sections[i].name == ".obX0"))
	}

PaylodBIN digital certs	rule PayloadBin_digital_cert
	{
	  meta:
	  Author = "SentinelLabs"
	  Family = "Evil Corp PayloadBIN digital cert signature"
	  strings:
	  $signer1 = "TAKE CARE SP Z O O"
	  $serial1 = {00 98 9A 33 B7 2A 2A A2 9E 32 D0 A5 E1 55 C5 39 63}
	  condition:
	  (int16(0) == 0x5A4D) and (($signer1) and ($serial1))
	}

INDICATORS OF COMPROMISE [IOCS]

Type	Indicator	Note
SHA1	https://gist.github.com/antonio-s1/1fc53ed220012a91e66ea939d628da84	EvilCorp loaders, Ransomware and SocGholish.
mutex	\BaseNamedObjects\MachineRendezvous	PhoenixLocker mutex
mutex	\BaseNamedObjects\ScriptNet	PhoenixLocker mutex
mutex	\BaseNamedObjects\CtfCtf	PayloadBIN mutex
mutex	\BaseNamedObjects\WizService	Hades mutex
Named Pipe	pipe\MS-30208-server	CS Beacon named pipe
Named Pipe	pipe\MS-7282-server	CS Beacon named pipe
Named Pipe	pipe\Nes-9563\berod	CS Beacon named pipe
Named Pipe	\\.\pipe\MS-30770-server	CS Beacon named pipe

Type	Indicator	Note
File Extension	.tcwwasted, .kgkq9, gn9cj, kgkq9, .test-2-v29, rh94k, dvxr9, .jjj9b, cm99v, jjj9bBy .cypherpunk, *.x9qmx, gvv3w,	Ransomware encrypted file extensions.
File name	wsqmcons.exe, ClassicStartMenu.exe, access, 1db0000.exe,Odbc, access mod_c.exe, draw, chikenchuchu123.exe, smpl.tmp, CorelDrw, CorelDrw.exe, cobaltstrike_2.txt, rad3F3E7.tmp, dss.exe, Trustedikstaller.exe, slcaa.exe, hexBA8.tmp, wsqmcons.exe, lZNYd.exe, IE4UINIT, IE4UINIT.EXE, amad.exe, amad.exe	Ransomware samples filename.
string (REG_Key)	111111111\{aa5b6a80-b834-11d0-932f-00a0c90dcaa9}	CryptOne Signature
string (REG_Key)	InterfacE\{b196b287-bab4-101a-b69c-00aa00341d07}	CryptOne Signature
string (REG_Key)	1nterfacE\\{b196b287-bab4-101a-b69c-00aa00341d07}	CryptOne Signature
string (REG_Key)	444erfacE\\{b196b287-bab4-101a-b69c-00aa00341d07}	CryptOne Signature
string (REG_Key)	555erfacE\\{b196b287-bab4-101a-b69c-00aa00341d07}	CryptOne Signature
string (REG_Key)	5nterfacE\\{b196b287-bab4-101a-b69c-00aa00341d07}	CryptOne Signature
string (REG_Key)	987erfacE\\{b196b287-bab4-101a-b69c-00aa00341d07}	CryptOne Signature
string (REG_Key)	Interfac4\\{b196b287-bab4-101a-b69c-00aa00341d07}	CryptOne Signature
string (REG_Key)	InterfacE\\{b196b287-bab4-101a-b69c-00aa00341d07}	CryptOne Signature
string (REG_Key)	aaaerfacE\\{b196b287-bab4-101a-b69c-00aa00341d07}	CryptOne Signature
string (REG_Key)	interfacE\\{b196b287-bab4-101a-b69c-00aa00341d07}	CryptOne Signature
string (REG_Key)	rrrerfacE\\{b196b287-bab4-101a-b69c-00aa00341d07}	CryptOne Signature
TOR site	o76s3m7l5ogig4u5[.]onion	Hades Tor Victim site

Type	Indicator	Note
TOR site	5lyi3c7x3ioakru4[.]onion	Hades Tor Victim site
TOR site	ixltdyumdlthrtgx[.]onion	Hades Tor Victim site
Tox id	2F5338DABD40348C71D858459FE7B8ED 60DDD9CE4D260C656E4E236A91167C3 1165081A85F79	Hades Tox id
email (contact information)	84550@PROTONMAIL.CH	WastedLocker contact info
email (contact information)	67146@ECLIPSO.CH	WastedLocker contact info
email (contact information)	91645@PROTONMAIL.CH	WastedLocker contact info
email (contact information)	61258@ECLIPSO.CH	WastedLocker contact info
email (contact information)	48907@PROTONMAIL.COM	WastedLocker contact info
email (contact information)	78470@TUTANOTA.COM	WastedLocker contact info
email (contact information)	29051@PROTONMAIL.CH	WastedLocker contact info
email (contact information)	98722@AIRMAIL.CC	WastedLocker contact info
email (contact information)	phcontactme@cock.li	Phoenix Locker contact info
telegram contact	hxxps://t[.]me/phdecrypt	Phoenix Locker contact info
email (contact information)	rickhood@armormail.net	PayloadBIN contact info

Type	Indicator	Note
email (contact information)	meredithpatrick@protonmail.com	PayloadBIN contact info
email (contact information)	chooc9@secmail.pro	CypherPunk contact info
email (contact information)	jeey5o@tutanota.de	CypherPunk contact info
email (contact information)	rei5ah@protonmail.com	CypherPunk contact info
certificate	Name QEXAYFGTEHMURVBTPT Valid From 06:53 PM 06/28/2020 Valid To 11:59 PM 12/31/2039 Serial Number D4 F5 81 07 41 D8 16 99 41 56 E8 69 E6 4E 0A A6	associated with WastedLocker
certificate	Name LAK COMPANY SP Z O O Valid From: “2021-01-15 00:00:00” Valid To: “2022-01-15 23:59:59” Serial Number: “32 AE 77 10 E2 57 05 A0 17 0F 14 E9 21 13 3E 2B”	associated with Hades
certificate	Name SATURDAY CITY LIMITED Valid From 12:00 AM 01/14/2021 Valid To 11:59 PM 01/14/2022 Serial Number 3B 00 73 14 84 4B 11 4C 61 BC 15 6A 06 09 A2 86	associated with PhoenixLocker
certificate	Name: TAKE CARE SP Z O O Valid From: 12:00 AM 03/09/2021 Valid To: 11:59 PM 03/09/2022 Serial Number: 00 98 9A 33 B7 2A 2A A2 9E 32 D0 A5 E1 55 C5 39 63	associated with PayloadBIN
certificate	Name: KOMPLEKS BUD TECH SP Z O O Valid From: 12:00 AM 11/17/2020 Valid To: 11:59 PM 11/17/2021 Serial Number: 00 DB 6F E0 46 F1 19 82 0A E2 68 E5 73 6B AB 90 27	associated with Hades
certificate	Name: OTRBXJVNJJOIXXBVFO Valid From:11:55 AM 07/17/2020 Valid To: 11:59 PM 12/31/2039 Serial Number: 39 7D A7 D7 7D AC EB AC 42 58 FB AD 4D 67 AA 85	associated with WastedLocker

Type	Indicator	Note
certificate	Name TMFZUDNLMDSVGTALOW Valid From: 07:53 PM 06/08/2020 Valid To: 11:59 PM 12/31/2039 Serial Number 0E A4 53 C1 C3 A5 B1 B2 4B 75 D3 8E 41 B7 5C C6	associated with CS loader
certificate	Name: Programavimo paslaugos, MB Valid From: 12:00 AM 05/14/2020 Valid To: 11:59 PM 05/14/2021 Serial Number: 29 12 8A 56 E7 B3 BF B2 30 74 25 91 AC 8B 47 18	associated with CS loader
certificate	Name: Elite Web Development Ltd. Valid From: 12:00 AM 07/02/2020 Valid To: 11:59 PM 07/02/2021 Serial Number: 6C FA 50 50 C8 19 C4 AC BB 8F A7 59 79 68 8D FF	associated with CS loader
certificate	Name: Lets Start SP Z O O Valid From: 12:00 AM 05/27/2020 Valid To: 11:59 PM 05/27/2021 Serial Number: 00 AF F7 62 E9 07 F0 64 4E 76 ED 8A 74 85 FB 12 A1	associated with CS loader
certificate	Name: CSTech Software Inc. Valid From: 12:00 AM 05/27/2020 Valid To: 11:59 PM 05/27/2021 Serial Number: 7B FB FD FE F4 36 08 73 0E E1 47 79 EE 3E E2 CB	associated with CS loader
certificate	Name: Big Agency a.s. Valid From: 12:00 AM 05/27/2020 Valid To: 11:59 PM 05/27/2021 Serial Number: 00 E9 C3 AF 3C 42 E0 6F 7E 20 A3 4B 81 F6 9E 96 30	associated with WastedLocker
certificate	Name: EGZTWCSETZWIHIICGS Valid From: 08:06 AM 06/23/2020 Valid To: 11:59 PM 12/31/2039 Serial Number: CA DA 74 45 A1 49 6C 89 44 EF 4C AE C6 8B 65 3E	associated with WastedLocker
certificate	Name: SAK GUARD SOLUTION LTD Valid From: 12:00 AM 02/17/2021 Valid To: 11:59 PM 02/17/2022 Serial Number: 3D C6 B2 72 F4 66 B7 29 F8 4A 17 27 82 51 D6 CC	associated with Hades

Type	SocGholish
IP Address	130.0.233[.]178
IP Address	37.48.84[.]156
IP Address	179.43.169[.]30
IP Address	79.110.52[.]138
IP Address	81.4.122[.]193
IP Address	195.189.96[.]41

Type	SocGholish Domain
domain	*.login.nuwealthmedia.com
domain	*.news.pocketstay.com
domain	*.services.accountablitypartner.com
domain	*.login.wwpcrisis.com
domain	*.login.markbrey.com
domain	*.nodes.fioressence.com
domain	*.push.youbyashboutique.com
domain	*.office.drpease.com

Type	Indicator	Note
Domain	bingoshow[.]xyz	Hades Ransomware C2

Type	CobaltStrike Loader
IP Address	23.227.193[.]137
IP Address	138.124.180[.]216
IP Address	37.48.84[.]156
IP Address	179.43.169[.]30
IP Address	79.110.52[.]138
IP Address	81.4.122[.]193
IP Address	195.189.96[.]41
IP Address	185.162.131[.]99
IP Address	185.250.151[.]33
IP Address	82.148.28[.]9
IP Address	54.192.229[.]106
IP Address	54.192.229[.]20
IP Address	54.192.229[.]43
IP Address	54.192.229[.]71
Domain	Consultane[.]com
Domain	Lafeedback[.]com
Domain	websitelistbuilder[.]com
Domain	twimg-us.azureedge[.]net
Domain	cutyoutube[.]com
Domain	cdn.auditor.adobe[.]com
Domain	cofeedback[.]com
Domain	roofingspecialists[.]info/file
Domain	wholesalerandy[.]com
Domain	pieceofheavenptc[.]info

Type	CobaltStrike Loader
Domain	Currentteach[.]com
Domain	Newschools[.]info
Domain	firsino[.]com
Domain	potasip[.]com
Domain	adsmarketart[.]com
Domain	advancedanalysis[.]be
ASN	50340, 45102, 49505

APPENDIX

UNPACKING CRYPTONE

Massive unpacking of CryptoOne packed samples can be found here. CryptOne unpacking method consists of two stages:

1. Decrypts and executes embedded shellcode.
2. Shellcode decrypts and executes embedded executables.

CryptOne gets chunks of the encrypted data, which are separated by junk.

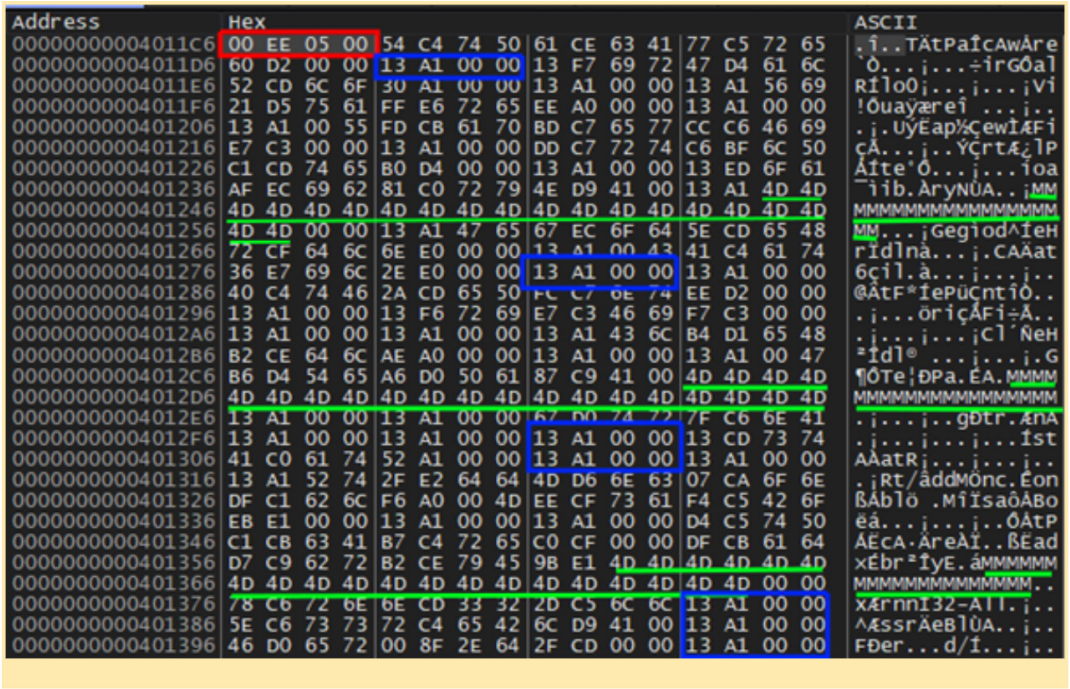


Fig 31

Example Memory Dump:

- **0x5EE00,** Encrypted size.
- **0x4011CA,** Address of encrypted data
- **0x4D/"M",** Junk data
- **0x14,** Junk size
- **0x7A,** Chunk Size

After removal of the junk data, , the decryption starts with a simple XOR-Key which increases by 0x4 in each round. The initial XOR-Key is **0xA113.**

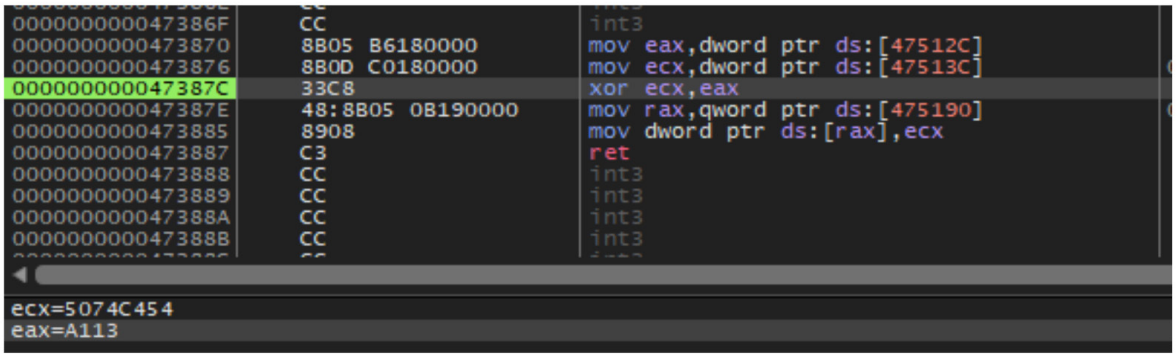


Fig 32

Once the shellcode is decrypted, we can slightly observe the string *"This program cannot be run in DOS mode"* where this data contains an executable which requires a second decryption.

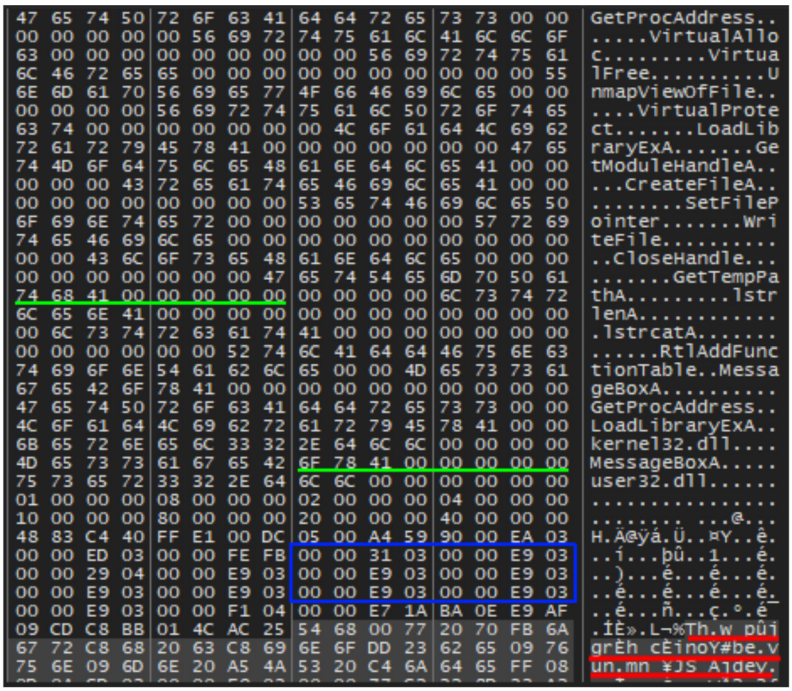


Fig 33

Similar to previous decryption, this time the shellcode decrypts the embedded binary.

```
__int64 __fastcall shellcode_xor(__int64 exec, unsigned int exec_size)
{
  __int64 result; // rax
  unsigned int i; // [rsp+0h] [rbp-18h]

  for ( i = 0; ; i += 4 )
  {
    result = exec_size;
    if ( i >= exec_size )
      break;
    *(_DWORD *)(exec + i) += i;
    *(_DWORD *)(exec + i) ^= i + 0x3E9;          // Initial XOR-Key 0x3E9
  }
  return result;
}
```

Fig 34

The shellcode allocates and copies the encrypted executable and starts the decryption loop, once it finishes, jumps to the EntryPoint and executes the unpacked sample.

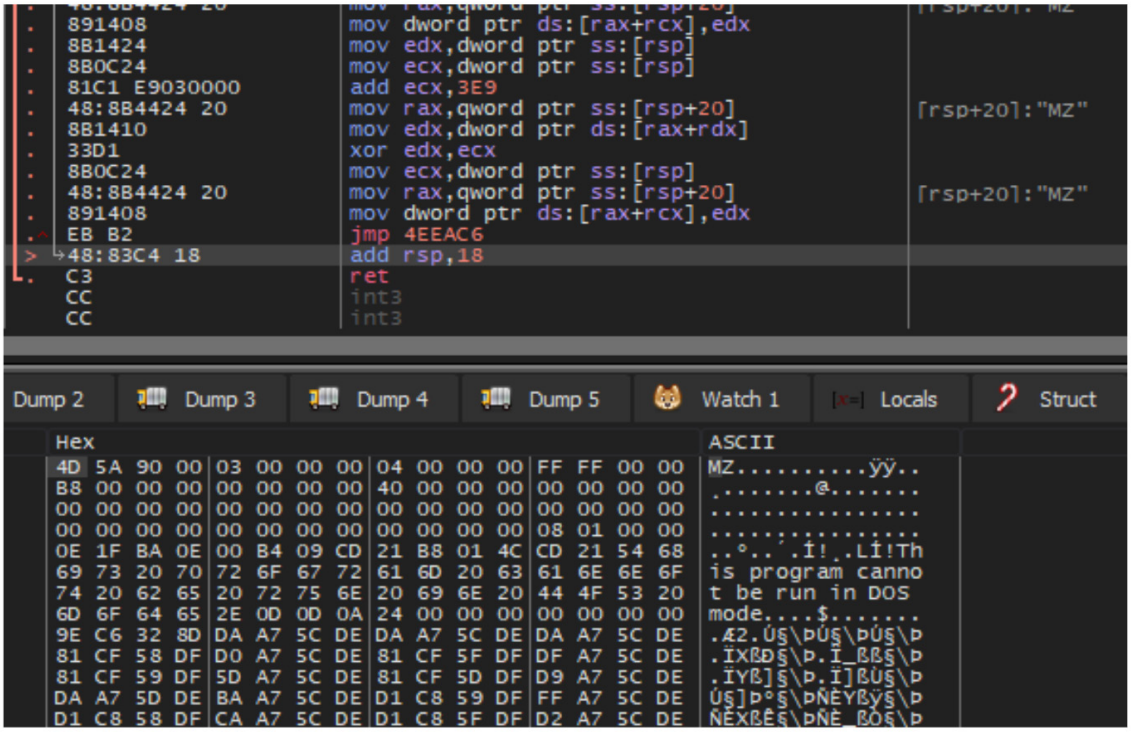


Fig 35

At this stage we can observe strings related to the unpacked sample.

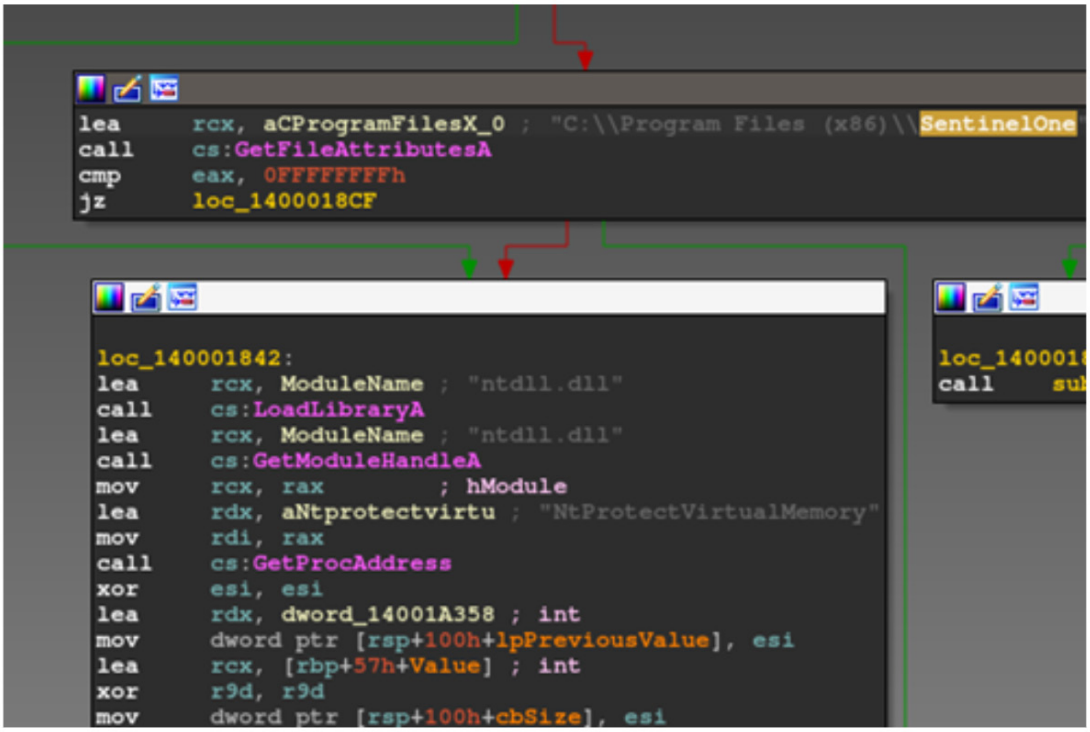
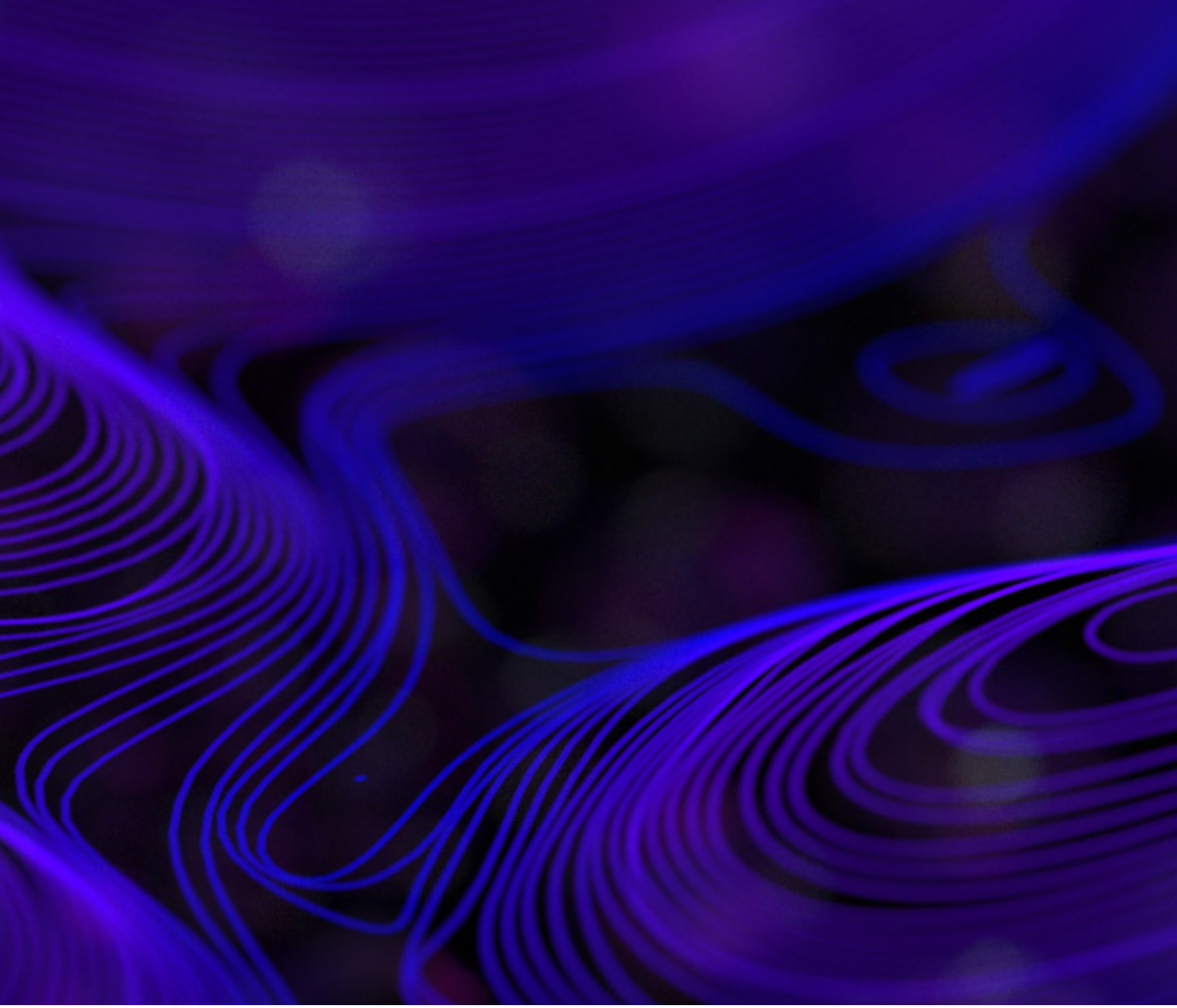


Fig 36



ABOUT SENTINELLABS

InfoSec works on a rapid iterative cycle where new discoveries occur daily and authoritative sources are easily drowned in the noise of partial information. SentinelLabs is an open venue for our threat researchers and vetted contributors to reliably share their latest findings with a wider community of defenders. No sales pitches, no nonsense. We are hunters, reversers, exploit developers, and tinkerers shedding light on the world of malware, exploits, APTs, and cybercrime across all platforms. SentinelLabs embodies our commitment to sharing openly –providing tools, context, and insights to strengthen our collective mission of a safer digital life for all.