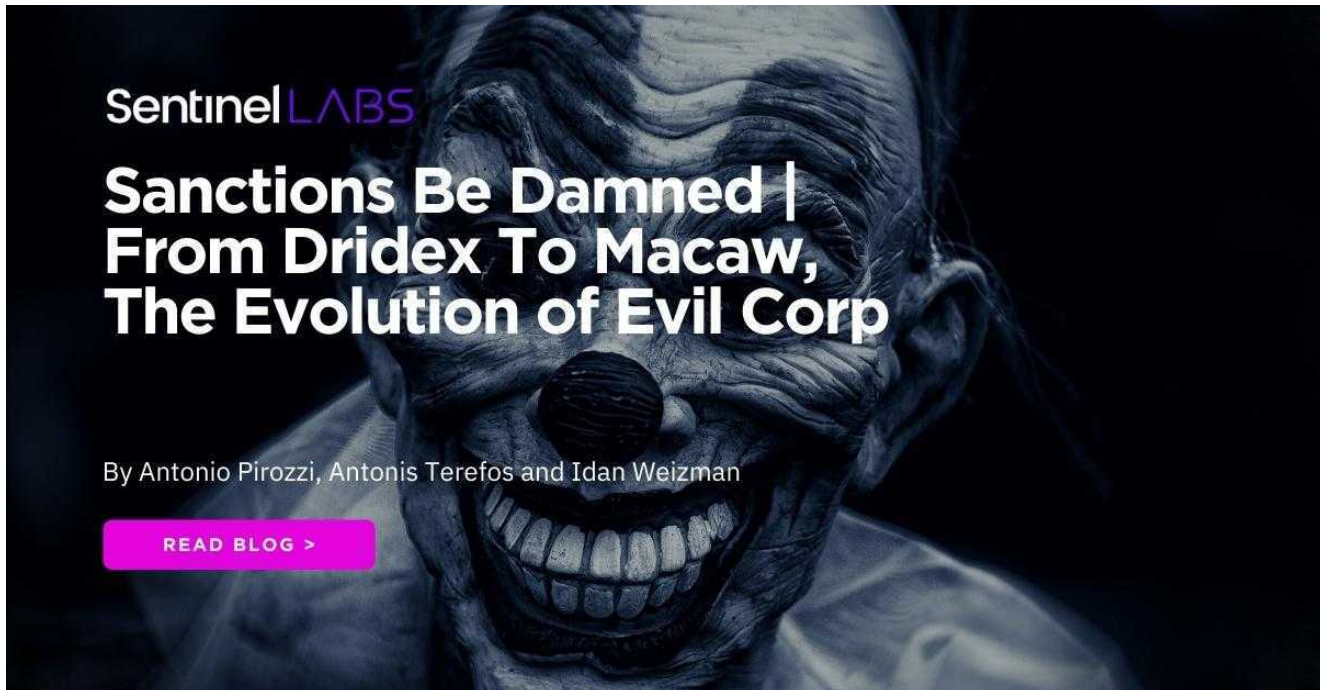


# Sanctions Be Damned | From Dridex to Macaw, The Evolution of Evil Corp

---

 [sentinelone.com/labs/sanctions-be-damned-from-dridex-to-macaw-the-evolution-of-evil-corp/](https://sentinelone.com/labs/sanctions-be-damned-from-dridex-to-macaw-the-evolution-of-evil-corp/)

Antonio Pirozzi



By Antonio Pirozzi, Antonis Terefos and Idan Weizman

## Executive Summary

---

- Since OFAC sanctions in 2020, the global intelligence community has been split into different camps as to how Evil Corp is operating.
- 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.
- Our analysis indicates that Evil Corp became a customer of the CryptOne packer-as-a-service from March 2020. We created a static unpacker, [de-CryptOne](#) for CryptOne and identified different versions of this cryptor which have never previously been reported.

[Read the Full Report](#)

## Introduction

---

Evil Corp (EC) is an advanced cybercrime operations cluster originating from Russia that has been active since 2007. The UK National Crime Agency called it “the world’s most harmful cyber crime group.” In December 2019, the U.S. Treasury Department’s Office of Foreign Assets Control (OFAC) 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.

After the indictments, the global intelligence community was split into different camps as to how Evil Corp was operating. Some assessed that there was a voluntary transition of EC operations to another ‘trusted’ partner while the core group remained the controller of operations. Some had theories that Evil Corp had stopped operating and that another advanced actor operated Hades, trying to mimic the same *modus operandi* as Evil Corp to mislead attribution. Others claimed possible attribution to the HAFNIUM activity cluster.

SentinelLabs has conducted an in-depth review and technical analysis of Evil Corp activity, malware and TTPs. Our full report has a number of important findings for the research community. We relied heavily on our analysis of a crypter tool dubbed “CryptOne”, which supports our wider clustering of Evil Corp activity. Our research also argues that the original operators continue to be active despite the sanctions, continuously changing their TTPs in order to stay under the radar.

In this post, we summarize some key observations from our technical analysis on the evolution of Evil Corp from Dridex through to Macaw Locker and, for the first time, publicly describe CryptOne and the role it plays in Evil Corp malware development. For the full technical analysis, comprehensive IOCs and YARA hunting rules, please see the full report.

## Overview of Recent Evil Corp Activity

---

After the OFAC indictment, we witnessed a change in Evil Corp 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 SocGhosh 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 relying more heavily on Cobalt Strike to gain an initial foothold and perform lateral movement, rather than PowerShell Empire.

In May 2020, a new ransomware variant appeared in the wild dubbed WastedLocker. WastedLocker (S0612) employed 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.

In December 2020, a new ransomware variant named Hades was first seen in the wild and publicly reported. Hades is a 64-bit compiled version of WastedLocker that displays important code and functionality overlaps. A few months later, in March 2021, a new variant

Phoenix Locker appeared in the wild. Our analysis suggests this is a rebranded version of Hades with little to no changes. Later, a new variant named PayloadBIN appeared in the wild, a continuation from Phoenix Locker.

## **A Unique Cluster: BitPaymer, WastedLocker, Hades, Phoenix Locker, PayloadBIN**

---

From our analysis, we discovered evidence of code overlaps, as well as shared configurations, packers and TTPs leading us to assess with high confidence that Bitpaymer, WastedLocker, Hades, PhoenixLocker and PayloadBIN share a common codebase. Our full report goes into the evidence in fine detail. The following section presents a brief summary.

### **From BitPaymer to WastedLocker**

---

Previous research shows a sort of knowledge reuse between BitPaymer and WastedLocker. SentinelLabs analysis shows that Hades and WastedLocker share the same codebase.

Among other similarities, detailed in the full report, we observe 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 variants, hinting that the same utility library was used.

### **From WastedLocker to Hades**

---

Previous research assessed the main similarities and differences between the two ransomware families. SentinelLabs analysis shows that Hades and WastedLocker share the same codebase.

Again we see the same RSA functions in both families. Both also implement file and directory enumeration logic identically. Comparing the logic and the Control Flow Graph of both routines, we conclude that both ransomware use the same code for file and directory enumeration. We also found similarities between the functions responsible for drive enumeration.

### **From Hades to Phoenix Locker**

---

In the samples we analyzed, we discovered that Phoenix Locker was a reused and newly-packed Hades payload. Hades and Phoenix samples were compiled at the same time. 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. The compiler and linker versions are also the same. This technique of payload reuse was also seen in BitPaymer in order to make the ransomware polymorphic and more evasive.

### **From Phoenix Locker to PayloadBIN**

---

We observed that the majority of PayloadBIN functions overlap with PhoenixLocker. File enumerating functions are practically identical.

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 ransoms and comparing them. We distinguished two distinct clusters.

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\"
```

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"
```

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 ransoms have the same implementation of Shadows copy deletion:

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

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

## Analysis of the Cypherpunk Variant

---

A new, possibly experimental, variant dubbed “Cypherpunk” – first reported in June 2021- was analyzed and linked to the same lineage.

```
C:\Users\Lucas\Documents\OneNote Notebooks\Personal\General.one.cypherpunk  
C:\Users\Lucas\Documents\OneNote Notebooks\Personal\CONTACT-T0-DECRYPT.txt  
C:\Users\Lucas\Documents\awards.xls.cypherpunk  
C:\Users\Lucas\Desktop\ZoneMap.dwf.cypherpunk  
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
```

Code similarity analysis shows that the Cypherpunk version (SHA1 e8d485259e64fd375e03844c03775eda40862e1c) 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.

SentinelLabs assesses this new finding is likely an indication that Evil Corp is still working on updating their tradecraft in order to change their signature and stay under the radar.

## Evil Corp Pivots to Macaw Locker Ransomware

---

In October 2021, a new ransomware variant named 'Macaw Locker' appeared in the wild, in an attack that began on October 10th against Olympus. A few days later Sinclair Broadcast Group was also attacked, causing widespread disruption. Some researchers claimed a possible connection with WastedLocker, but to date no further details have emerged.

```
=====
  _ , ---
 /_ |o\ )
  -\ / /  MACAW
   ) (   LOCKER
  //  \ \
  { [ ] }
===== "" "" =====
Data in your network has been stolen and encrypted.

To get your data back visit:
https://

Decryption ID: [REDACTED]
```

### Macaw ransom note

The ransomware presents anti-analysis features like API hashing and indirect API calls with the intention of evading analysis. One aspect that immediately sets Macaw apart is that it requires a custom token, provided from the command line, which appears to be specific to each victim; without it, the ransomware won't execute.

```
macaw_sample.exe -k
```

The use of a custom token is also seen in Egregor and BlackCat ransomware families, and is a technique used to aid anti-analysis (T1497.002).

Another new addition to Macaw is 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 its address prior to the call itself.

```

1 void __stdcall f_get_api_by_hash(context *context, uint32_t fix_api_info_offset_value, uint32_t hash_value)
2 {
3     api_info *api_info; // ecx
4     uint32_t i_hash; // edi
5     api_info **p_next; // ecx
6     uint32_t fixed_hash; // edx
7     void *result; // eax
8
9     fixed_hash = context->fix_hash_value + hash_value;
10    result = 0;
11    for ( api_info = (sizeof(api_info) * fixed_hash + fix_api_info_offset_value + 8); api_info; api_info = *p_next )
12    {
13        i_hash = api_info->hash;
14        if ( fixed_hash == i_hash )
15        {
16            result = (api_info->api_offset - context->fix_api_offset_value);
17            break;
18        }
19        if ( fixed_hash <= i_hash )
20            p_next = &api_info->next_bigger;
21        else
22            p_next = &api_info->next_smaller;
23    }
24    nullsub_136(context, fix_api_info_offset_value, hash_value);
25    return result;
26 }

```

00120B56 f\_get\_api\_by\_hash:1 (1123B56)

Macaw function to dynamically fetch addresses

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.

We assessed the similarity of two core functions between Hades and Macaw. In both strains, the implementation is the same. The only minor differences are from the imports fetched at runtime.

## CryptOne: One Packer To Rule Them All

CryptOne (also known as HellowinPacker) was a special packer used by Evil Corp up until mid-2021.

CryptOne appears to have first been noticed in [2015](#). Early versions were used by an assortment of different malware families such as [NetWalker](#), [Gozi](#), [Dridex](#), [Hancitor](#) and [Zloader](#). In 2019, Bromium analyzed and reported it as in use by [Emotet](#). In June 2020, [NCC Group](#) reported that CryptOne was used to pack WastedLocker. In 2021, [researchers](#) observed CryptOne being advertised as a Packer-as-a-Service on various crime-oriented 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;

We created a static unpacker, de-CryptOne, which unpacks 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.

## Unpacking CryptOne

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.

Address	Hex	ASCII
00000000004011C6	00 EE 05 00 54 C4 74 50 61 CE 63 41 77 C5 72 65	.i..TÄtPaïcAwAre
00000000004011D6	60 D2 00 00 13 A1 00 00 13 F7 69 72 47 D4 61 6C	`ò...j...÷irG0al
00000000004011E6	52 CD 6C 6F 30 A1 00 00 13 A1 00 00 13 A1 56 69	Ri!o0j...j...jVi
00000000004011F6	21 D5 75 61 FF E6 72 65 EE A0 00 00 13 A1 00 00	!0uayærei ...j..
0000000000401206	13 A1 00 55 FD CB 61 70 BD C7 65 77 CC C6 46 69	.j..UyËap%çewIÆFi
0000000000401216	E7 C3 00 00 13 A1 00 00 DD C7 72 74 C6 BF 6C 50	çÄ...j..ÿCrtÆjLP
0000000000401226	C1 CD 74 65 B0 D4 00 00 13 A1 00 00 13 ED 6F 61	Äite'ò...j...ioa
0000000000401236	AF EC 69 62 81 C0 72 79 4E D9 41 00 13 A1 4D 4D	~ib.AryNÜA...jMM
0000000000401246	4D 4D 4D 4D 4D 4D 4D 4D 4D 4D 4D 4D 4D 4D 4D 4D	MMMMMMMMMMMMMMMM
0000000000401256	4D 4D 00 00 13 A1 47 65 67 EC 6F 64 5E CD 65 48	MM...jGegïod^IeH
0000000000401266	72 CF 64 6C 6E E0 00 00 13 A1 00 43 41 C4 61 74	rïdlnä...j.CAAat
0000000000401276	36 E7 69 6C 2E E0 00 00 13 A1 00 00 13 A1 00 00	6çil.ä...j...j..
0000000000401286	40 C4 74 46 2A CD 65 50 FC C7 6E 74 EE D2 00 00	@ÄtF*IePüCñtiò..
0000000000401296	13 A1 00 00 13 F6 72 69 E7 C3 46 69 F7 C3 00 00	.j...öriçÄFi÷Ä..
00000000004012A6	13 A1 00 00 13 A1 00 00 13 A1 43 6C B4 D1 65 48	.j...j...jCl'NeH
00000000004012B6	82 CE 64 6C AE A0 00 00 13 A1 00 00 13 A1 00 47	=ïdl® ...j...j.G
00000000004012C6	B6 D4 54 65 A6 D0 50 61 87 C9 41 00 4D 4D 4D 4D	¶0Te!ðPa.ÉA.MMMM
00000000004012D6	4D 4D 4D 4D 4D 4D 4D 4D 4D 4D 4D 4D 4D 4D 4D 4D	MMMMMMMMMMMMMMMM
00000000004012E6	13 A1 00 00 13 A1 00 00 67 D0 74 72 7F C6 6E 41	.j...j..gðtr.ÄnA
00000000004012F6	13 A1 00 00 13 A1 00 00 13 A1 00 00 13 CD 73 74	.j...j...j...ïst
0000000000401306	41 C0 61 74 52 A1 00 00 13 A1 00 00 13 A1 00 00	AAatRi...j...j..
0000000000401316	13 A1 52 74 2F E2 64 64 4D D6 6E 63 07 CA 6F 6E	.jRt/äddMönc.Éon
0000000000401326	DF C1 62 6C F6 A0 00 4D EE CF 73 61 F4 C5 42 6F	BÄblö .MiisaöABo
0000000000401336	EB E1 00 00 13 A1 00 00 13 A1 00 00 D4 C5 74 50	éä...j...j..ÓÄtP
0000000000401346	C1 CB 63 41 B7 C4 72 65 C0 CF 00 00 DF CB 61 64	ÄÉcA.AreÄI..BÉad
0000000000401356	D7 C9 62 72 B2 CE 79 45 9B E1 4D 4D 4D 4D 4D 4D	xÉbr²Iye.äMMMMMM
0000000000401366	4D 4D 4D 4D 4D 4D 4D 4D 4D 4D 4D 4D 4D 4D 00 00	MMMMMMMMMMMMMMMM..
0000000000401376	78 C6 72 6E 6E CD 33 32 2D C5 6C 6C 13 A1 00 00	xÄrnnI32-Äll.i..
0000000000401386	5E C6 73 73 72 C4 65 42 6C D9 41 00 13 A1 00 00	ÄÉssrÄeBlÜA...j..
0000000000401396	46 D0 65 72 00 8F 2E 64 2F CD 00 00 13 A1 00 00	Fðer...d/I...j..

CryptOne junk data

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 .

```

000000000047386F CC int3
0000000000473870 8805 B6180000 mov eax,dword ptr ds:[47512C]
0000000000473876 880D C0180000 mov ecx,dword ptr ds:[47513C]
000000000047387C 33C8 xor ecx,ecx
000000000047387E 48:8805 0B190000 mov rax,qword ptr ds:[475190]
0000000000473885 8908 mov dword ptr ds:[rax],ecx
0000000000473887 C3 ret
0000000000473888 CC int3
0000000000473889 CC int3
000000000047388A CC int3
000000000047388B CC int3
000000000047388C CC int3

```

ecx=5074C454  
eax=A113

### CryptOne XOR Key

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

```

47 65 74 50 72 6F 63 41 64 64 72 65 73 73 00 00 GetProcAddress..
00 00 00 00 00 56 69 72 74 75 61 6C 41 6C 6C 6F ....VirtualAllo
63 00 00 00 00 00 00 00 00 00 56 69 72 74 75 61 c.....Virtua
6C 46 72 65 65 00 00 00 00 00 00 00 00 00 00 00 lFree.....U
6E 6D 61 70 56 69 65 77 4F 66 46 69 6C 65 00 00 nMapViewOfFile..
00 00 00 00 56 69 72 74 75 61 6C 50 72 6F 74 65 ....VirtualProte
63 74 00 00 00 00 00 00 00 4C 6F 61 64 4C 69 62 ct.....LoadLib
72 61 72 79 45 78 41 00 00 00 00 00 00 47 65 raryExA.....Ge
74 4D 6F 64 75 6C 65 48 61 6E 64 6C 65 41 00 00 tModuleHandleA..
00 00 00 43 72 65 61 74 65 46 69 6C 65 41 00 00 ...CreateFileA..
00 00 00 00 00 00 00 00 53 65 74 46 69 6C 65 50 .....SetFileP
6F 69 6E 74 65 72 00 00 00 00 00 00 00 57 72 69 ointer.....Wri
74 65 46 69 6C 65 00 00 00 00 00 00 00 00 00 00 teFile.....
00 00 43 6C 6F 73 65 48 61 6E 64 6C 65 00 00 00 ..CloseHandle...
00 00 00 00 00 00 00 47 65 74 54 65 6D 70 50 61 .....GetTempPa
74 68 41 00 00 00 00 00 00 00 00 00 6C 73 74 72 thA.....lstr
6C 65 6E 41 00 00 00 00 00 00 00 00 00 00 00 00 lenA.....
00 6C 73 74 72 63 61 74 41 00 00 00 00 00 00 00 .lstrcatA.....
00 00 00 00 00 00 52 74 6C 41 64 64 46 75 6E 63 .....RtlAddFunc
74 69 6F 6E 54 61 62 6C 65 00 00 4D 65 73 73 61 tionTable..Messa
67 65 42 6F 78 41 00 00 00 00 00 00 00 00 00 00 geBoxA.....
47 65 74 50 72 6F 63 41 64 64 72 65 73 73 00 00 GetProcAddress..
4C 6F 61 64 4C 69 62 72 61 72 79 45 78 41 00 00 LoadLibraryExA..
68 65 72 6E 65 6C 33 32 2E 64 6C 6C 00 00 00 00 kernel32.dll....
4D 65 73 73 61 67 65 42 6F 78 41 00 00 00 00 00 MessageBoxA.....
75 73 65 72 33 32 2E 64 6C 6C 00 00 00 00 00 00 user32.dll.....
01 00 00 00 08 00 00 00 02 00 00 00 04 00 00 00 .....@...
10 00 00 00 80 00 00 00 20 00 00 00 40 00 00 00 .....@...
48 83 C4 40 FF E1 00 DC 05 00 A4 59 90 00 EA 03 H.A@ya.U..#Y..è
00 00 ED 03 00 00 FE FB 00 00 31 03 00 00 E9 03 ..i...pù..1...è
00 00 29 04 00 00 E9 03 00 00 E9 03 00 00 E9 03 ..)....è....è...è
00 00 E9 03 00 00 E9 03 00 00 E9 03 00 00 E9 03 ..è....è....è...è
00 00 E9 03 00 00 F1 04 00 00 E7 1A BA 0E E9 AF ..è...ñ...ç..°..è
09 CD C8 8B 01 4C AC 25 54 68 00 77 20 70 FB 6A .IÈ».L-%Th.w pùj
67 72 C8 68 20 63 C8 69 6E 6F DD 23 62 65 09 76 grÈh cÈinoY#be.v
75 6E 09 6D 6E 20 A5 4A 53 20 C4 6A 64 65 FF 08 un.mn #JS A1dev.

```

CryptOne partially

decrypted shellcode

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;
}

```

Fastcall

Shellcode XOR

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

```

. 48:8B4424 20 mov rax,qword ptr ss:[rsp+20]
. 891408 mov dword ptr ds:[rax+rcx],edx
. 8B1424 mov edx,dword ptr ss:[rsp]
. 8B0C24 mov ecx,dword ptr ss:[rsp]
. 81C1 E9030000 add ecx,3E9
. 48:8B4424 20 mov rax,qword ptr ss:[rsp+20] [rsp+20]: "MZ"
. 8B1410 mov edx,dword ptr ds:[rax+rdx]
. 33D1 xor edx,ecx
. 8B0C24 mov ecx,dword ptr ss:[rsp]
. 48:8B4424 20 mov rax,qword ptr ss:[rsp+20] [rsp+20]: "MZ"
. 891408 mov dword ptr ds:[rax+rcx],edx
. EB B2 jmp 4EEAC6
> 48:83C4 18 add rsp,18
. C3 ret
. CC int3
. CC int3

```

CryptOne

Dump 2    Dump 3    Dump 4    Dump 5    Watch 1    Locals    Struct

Hex	ASCII
4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00	MZ.....ÿÿ..
B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00	.....@.....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00 00 00 00 00 00 00 00 00 00 00 00 08 01 00 00	.....
0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68	..°.!.LI!Th
69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F	is program canno
74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20	t be run in DOS
6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00	mode...\$.....
9E C6 32 8D DA A7 5C DE DA A7 5C DE DA A7 5C DE	.Æ2.Ú\$`pÚ\$`pÚ\$`p
81 CF 58 DF D0 A7 5C DE 81 CF 5F DF DF A7 5C DE	.IXE\$`p.I`B\$`p
81 CF 59 DF 5D A7 5C DE 81 CF 5D DF D9 A7 5C DE	.IYB]\$`p.I]BÚ\$`p
DA A7 5D DE BA A7 5C DE D1 C8 59 DF FF A7 5C DE	Ú\$]p\$`pÑEYB\$`p
D1 C8 58 DF CA A7 5C DE D1 C8 5F DF D2 A7 5C DE	ÑEXRÉ\$`pÑE RÓ\$`p

executing the unpacked sample

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

```

lea rcx, aCProgramFilesX_0 ; "C:\\Program Files (x86)\\SentinelOne
call cs:GetFileAttributesA
cmp eax, 0FFFFFFFh
jz loc_1400018CF

loc_140001842:
lea rcx, ModuleName ; "ntdll.dll"
call cs:LoadLibraryA
lea rcx, ModuleName ; "ntdll.dll"
call cs:GetModuleHandleA
mov rcx, rax ; hModule
lea rdx, aNtprotectvirtu ; "NtProtectVirtualMemory"
mov rdi, rax
call cs:GetProcAddress
xor esi, esi
lea rdx, dword_14001A358 ; int
mov dword ptr [rsp+100h+lpPreviousValue], esi
lea rcx, [rbp+57h+Value] ; int
xor r9d, r9d
mov dword ptr [rsp+100h+cbSize], esi

```

CryptOne embedded

strings after unpacking

## A Unique Factory

Hunting for CryptOne led us to identify different implementations of the stub, some of 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 is replaced at runtime by the 'interface' keyword, creating the following registry key:

```
HKEY_CLASSES_ROOT\interface\{b196b287-bab4-101a-b69c-00aa00341d07}
```

The 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 into 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:

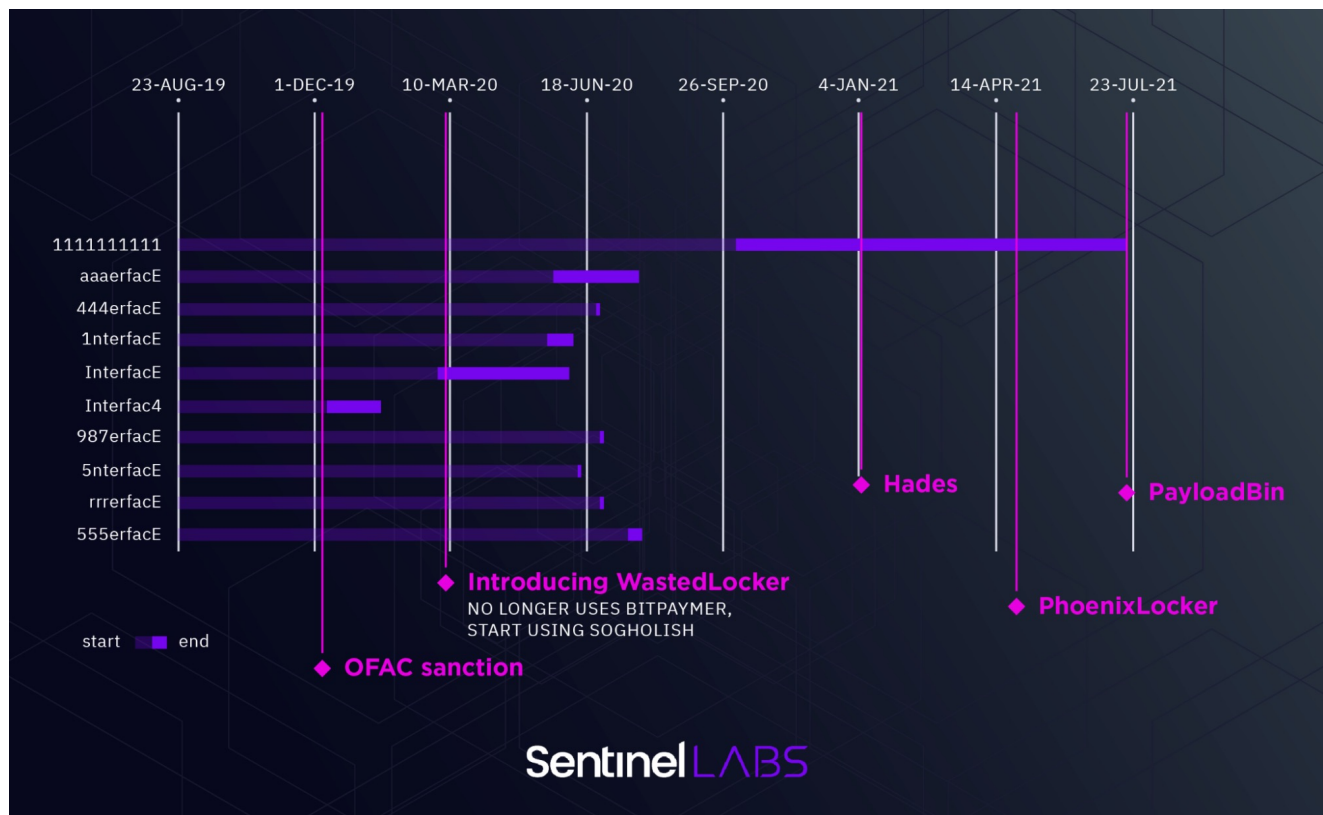
```
aaerfacE\\{b196b287-bab4-101a-b69c-00aa00341d07}  
1111111111\\{aa5b6a80-b834-11d0-932f-00a0c90dcaa9}
```

we noticed that in order to update the signature, the actor needs to re-compile the cryptor as the cryptor implementation changes.

## CryptOne Timeline

Our analysis shows that it is likely Evil Corp started being a customer of the CryptOne service from March 2020. From March to May 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 stub signature(InterfacE).

For a limited period of time between May 2020 and August 2020, we observed different versions of CryptOne overlaps.



CryptOne overlaps between May 2020 and August 2020

It seems that from a specific point in time, around September 2020, Hades, PhoenixLocker and PayloadBIN started adopting a specific CryptOne stub identified by the signature:

1111111111\{aa5b6a80-b834-11d0-932f-00a0c90dcaa9}

From December 2020, the CryptOne version '1111111111' appeared in the wild without any overlap.

## Conclusion

---

Clustering Evil Corp activity is demonstrably difficult considering that the group has changed TTPs several times in order to bypass sanctions and stay under the radar. This is in addition to the overall trend of actors receding back into secrecy. In this research, we connect the dots in the Evil Corp ecosystem, cluster Evil Corp malware, document the group's activities and provide insight into their TTPs.

SentinelLabs assesses with high confidence that WastedLocker, Hades, PhoenixLocker, Macaw Locker 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 Evil Corp activity cluster.

We fully expect that Evil Corp will continue to evolve and target organizations. In addition, we assess it is likely they will also 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 its evolution.

In-depth technical analysis, Indicators of Compromise and further technical references are available in the full report.

[Read the Full Report](#)