



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

TABLE OF CONTENTS



- 3** EXECUTIVE SUMMARY
- 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 Dridex¹ 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 trojan³. 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 2019⁴). 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 DoppelSpider⁵, considered a distinct group⁶ (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.

²https://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 families⁸.

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 claim⁹ 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 reported¹⁰. Hades is a 64-bit compiled version of WastedLocker that displays important code and functionality overlaps¹¹. 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 changes¹². Later, a new variant named PayloadBIN appeared in the wild¹³, 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 incidents¹⁴. 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/>

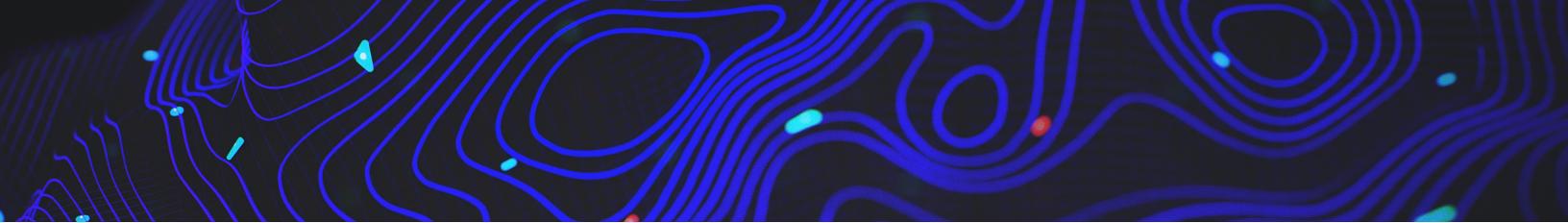
⁹<https://awakesecurity.com/blog/incident-response-hades-ransomware-gang-or-hafnium/>

¹⁰<https://twitter.com/demonslay335/status/1339324224029274118?s=20>

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

¹²section 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 2017¹⁵. 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 loader¹⁶. 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 framework¹⁷.

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.

¹⁴https://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/>

¹⁶https://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>

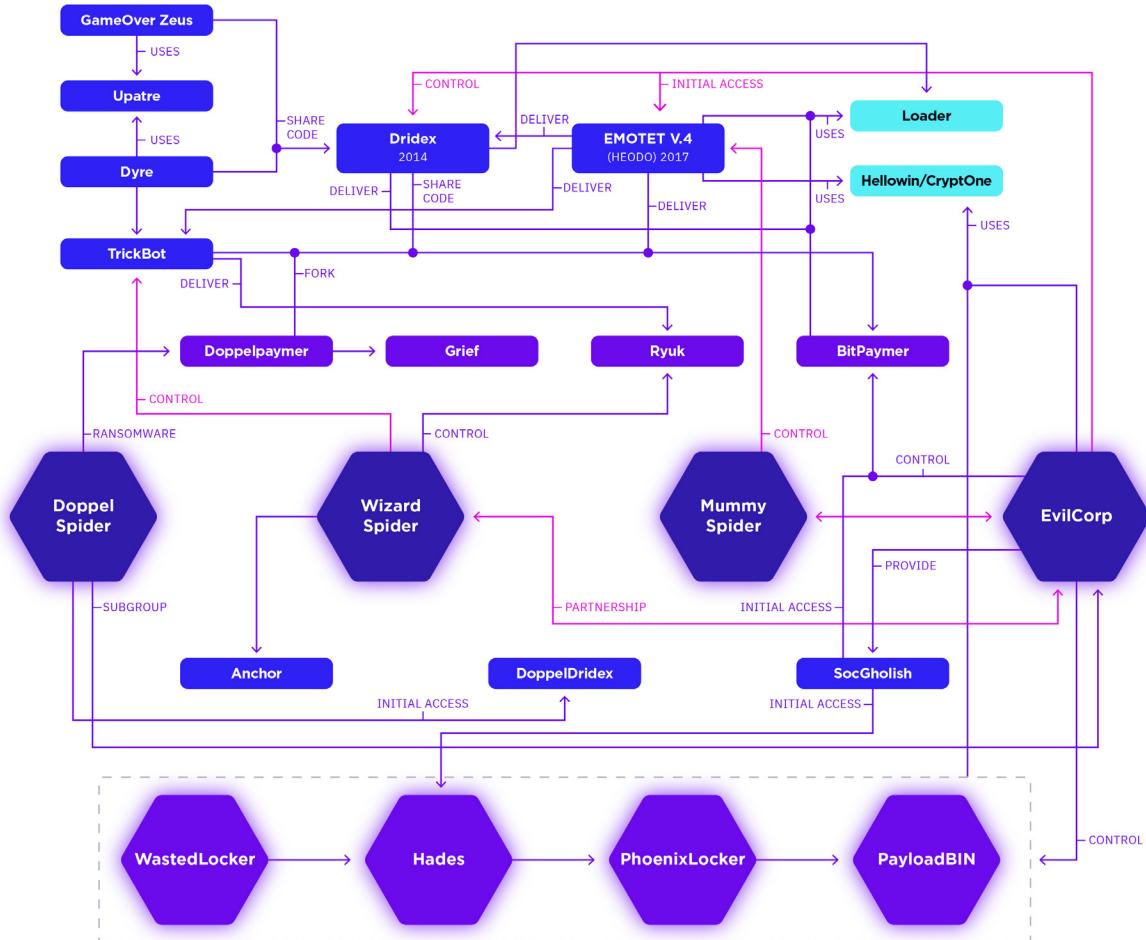


Fig 1: The EvilCorp affiliation diagram

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 similar 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.

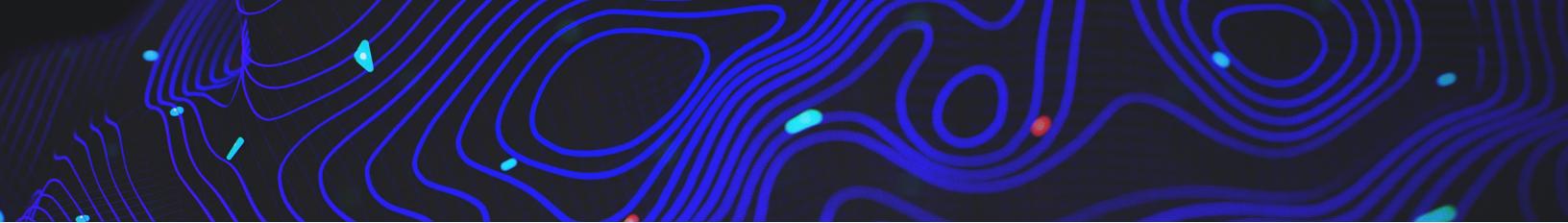


Fig 2

BITPAYMER VS. WASTEDLOCKER

Previous research¹⁸ 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 time¹⁹.

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 research²⁰ assessed the main similarities and differences between the two ransomware families:

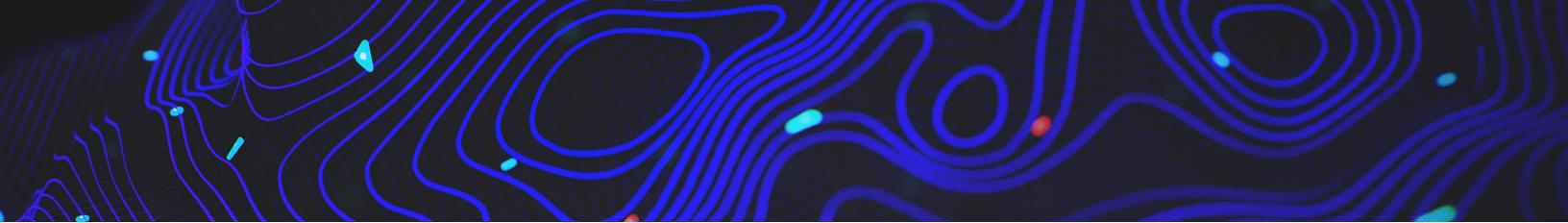
- Different UAC bypass methods: both taken from UACME project²¹
- Generalization: Hades does not contain victim information in the ransom note whereas WastedLocker does, and instead contains a tox channel in order to communicate and negotiate with victims
- Hades does not use ADS whereas WastedLocker 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

```

53     {
54         if ( !(Flags_encrypt_source_maybe & 0x1) || (unsigned int)(file_name_len + folder_path_len + 2) >= 0xFFFF )
55         {
56             memcpy(v21, v8->fileName, 2 * file_name_len);
57             new_folder_path = folder_path;
58             v13 = (NKHDR*)folder_path->file_path_len;
59             v14 = v13->v14;
60             v15[1] = 0;
61             if ( !config_black_list_maybe )
62                 goto LABEL_30;
63             if ( f_wildchar_wstrcmp_with_config_maybe(config_black_list_maybe, folder_path + 4) )
64                 {
65                 new_folder_path = folder_path;
66                 LABEL_30:
67                 ret_val = f_enumerate_files(
68                     new_folder_path,
69                     file_name_len + folder_path_len + 2,
70                     config_white_list_maybe,
71                     config_black_list_maybe,
72                     context,
73                     flags_encrypt_source_maybe);
74             }
75         }
76         else // case: not a directory, aka a file
77     {
78         goto LABEL_35;
79         memcpy(v21, v8->fileName, 2 * file_name_len + 2);
80         if ( config_white_list_maybe )
81             {
82                 if ( f_wildchar_wstrcmp_with_config_maybe(config_white_list_maybe, folder_path + 4) )
83                     goto LABEL_35;
84                 if ( config_black_list_maybe && f_wildchar_wstrcmp_with_config_maybe(config_black_list_maybe, folder_path + 4) )
85                     goto LABEL_35;
86             }
87         goto LABEL_35;
88         Size_t new_node_size = (file_name_len + folder_path_len + 0x52);
89         new_node = (thread_context_b *)heapAlloc(NHeap, 0, Size);
90         new_node->new_node_size = new_node_size;
91         if ( new_node )
92             {
93                 memset(new_node, 0, Size);
94                 new_node->prev = (thread_context *)new_node;
95                 new_node->next = (thread_context *)new_node;
96                 memcpy(new_node->fileName, folder_path, 2 * file_path_len + 2);
97                 new_node->flag_local_5_or_net_1 = flags_encrypt_source_maybe;
98                 new_node->file_attr = v8->fileAttributes;
99                 new_node->file_size_lowPart = v8->fileSizeLow;
100                new_node->file_size_highPart = v8->fileSizeHigh;
101                new_node->last_write_time = v8->lastWriteTime;
102                if ( context->lock.DebugInfo )
103                    interCriticalSection(&context->lock);
104                if ( context->lock.DebugInfo )
105                    prewrite(&new_node);
106                new_node->next = (thread_context *)&context->lock;
107                new_node->prev = prev;
108                prev->next = (thread_context *)new_node;
109                context->lock->list_count_maybe++;
110                if ( context->event_handle && context->list_count_maybe == 1 )
111                    SetEvent(context->event_handle);
112                if ( context->lock.DebugInfo )
113                    LeaveCriticalSection(&context->lock);
114            }
115        else
116            {
117                ret_val = 0;
118            }
119        }
120    }
121 LABEL_35:
122     if ( !ret_val )
123         goto LABEL_36;
124     if ( !(flags_encrypt_source_maybe & 4) != 0 )
125         ret_val = 0;
126 }
```



```

110     else if ( !(v8->fileNameLen & file_path_start_directory) != 0 )
111     {
112         v24 = flags_encrypt_source_maybe;
113         if ( !(Flags_encrypt_source_maybe & 0x10) == 0 || (unsigned int)v10 >= 0xFFFF )
114         {
115             goto LABEL_46;
116             asm { rcl d1, 44h }
117             f_memcpy(&folder_path->Buffer[v12], v8->FileName, 2164 * v14);
118             folder_path->Buffer[v12] = 0;
119             folder_path->file_path_len = 1 + folder_path_len;
120             if ( !config_black_list_maybe )
121                 {
122                     if ( f_wildchar_wstrcmp_with_config_maybe(folder_path->Buffer + 4, config_black_list_maybe) )
123                         {
124                             folder_path->length = 2 * folder_path_len;
125                             f_enumerate_files(
126                                 folder_path,
127                                 config_white_list_maybe,
128                                 config_black_list_maybe,
129                                 context,
130                                 flags_encrypt_source_maybe);
131                         }
132                     else // case: not a directory, aka a file
133                         {
134                             if ( has_16 & !v8->EndOfFile.QuadPart < (unsigned int)v13 )
135                                 goto LABEL_46;
136                             f_memcpy(&folder_path->Buffer[v12], v8->FileName, 2);
137                             if ( config_white_list_maybe )
138                                 {
139                                     if ( f_wildchar_wstrcmp_with_config_maybe(folder_path->Buffer + 4, config_black_list_maybe) )
140                                         goto LABEL_46;
141                                     if ( has_16 & v8->EndOfFile.QuadPart < (unsigned int)v13 )
142                                         goto LABEL_46;
143                                     f_memcpy(&folder_path->Buffer[v12], v8->FileName, 2);
144                                     if ( config_black_list_maybe )
145                                         {
146                                             if ( f_wildchar_wstrcmp_with_config_maybe(folder_path->Buffer + 4, config_black_list_maybe) )
147                                                 goto LABEL_46;
148                                             if ( config_black_list_maybe
149                                                 && f_wildchar_wstrcmp_with_config_maybe(folder_path->Buffer + 4, config_black_list_maybe) )
150                                                 goto LABEL_46;
151                                             new_node = (thread_context_b *)f_call_HeapAlloc_and_zero_memory(2, v34 + 2);
152                                             new_node = new_node;
153                                             if ( !new_node )
154                                                 {
155                                                     f_memcpy(new_node->fileName, folder_path->Buffer, 2164 * (file_path_len + folder_path_len + 1));
156                                                     new_node->new_node_size = new_node_size;
157                                                     context->lock->list_count_maybe++;
158                                                     new_node->file_path_len = folder_path_len;
159                                                     _CX_BOLZ_(0x040000);
160                                                     _asm { rcr cx, cl };
161                                                     new_node->file_size_lowPart = v8->fileSizeLow;
162                                                     new_node->file_size_highPart = v8->fileSizeHigh;
163                                                     new_node->last_write_time = (LARGE_INTEGER)v8->lastWriteTime;
164                                                     new_node->last_write_tid = (LARGE_INTEGER)v8->lastWriteTime;
165                                                     if ( context->lock.DebugInfo )
166                                                         EnterCriticalSection(&context->lock);
167                                                     new_node2 = new_node;
168                                                     context = context;
169                                                     if ( context->event_handle && context->list_count_maybe == 1 )
170                                                         SetEvent(context->event_handle);
171                                                     context = context;
172                                                     if ( context->lock.DebugInfo )
173                                                         LeaveCriticalSection(&context->lock);
174                                                 }
175                                             new_node->prev = prev;
176                                             new_node->next = (thread_context *)&context->lock;
177                                             new_node->prev = prev;
178                                             prev->next = (thread_context *)new_node;
179                                             context->lock->list_count_maybe++;
180                                             if ( context->event_handle && context->list_count_maybe == 1 )
181                                                 SetEvent(context->event_handle);
182                                             context = context;
183                                             if ( context->lock.DebugInfo )
184                                                 LeaveCriticalSection(&context->lock);
185                                         }
186                                     }
187                                 }
188                             }
189                         }
190                     }
191                 }
192             }
193             _asm { fmemset(v2, 0, v1) };
194             return v4;
195         }
196     }
197 }
```

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:

```

HANDLE EventA; // eax
const WCHAR *v8; // eax
PUSTR v9; // eax
char *v10; // eax
DWORD LastError; // eax
HANDLE hHandle; // [esp+0h] [ebp-4h]
PCWSTR pszStart; // [esp+1Ch] [ebp+10h]
PCWSTR pszStart2; // [esp+1Ch] [ebp+10h]
PCWSTR lpString; // [esp+1Ch] [ebp+10h]
const WCHAR *lpStringa; // [esp+2h] [ebp+1h]

ctx1 = (thread_context_lockable *)HeapAlloc(hHeap, 0, 0x44u);
ctx = ctx1;
if (!ctx)
{
    return;
}
memset(ctx, 0, sizeof(thread_context_lockable));
EventA = CreateEventW(0, 0, 0, 0);
ctx->m.event_handle = EventA;
if (!EventA)
{
    ctx->m.prev = (thread_context *)&ctx->m;
    ctx->m.next = (thread_context *)&ctx->m;
    InitializeCriticalSection(&ctx->lock);
    InitializeCriticalSection(&ctx->lock);
    ctx->m.outside_kill_handle_maybe = outside_kill_handle_maybe;
    ctx->m.has_2 = 2;
    ctx->m.encr_func = f_do_file_encrypt;
    ctx->m.space_func = f_get_disk_free_space;
    ctx->m.config_maybe = config_maybe;
    ctx->m.has_16 = 16;
    hHandle = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)f_encryption_thread, ctx, 0, 0);
    if (!hHandle)
    {
        _mmovne();
        v8 = pszStart;
        if (!pszStart)
        {
            lpStringa = pszStart;
            while (*v8)
            {
                v9 = StrChrW(v8, 0x7Cu);
                pszStarta = v9;
                if (v9)
                {
                    *v9 = 0;
                    pszStarta = v9 + 1;
                }
                v10 = f_call_WinetGetUniversalNameW(lpStringa);
                lpMem = v10;
                if (v10)
                    lpStringa = (const WCHAR *)v10;
                f_enumerate_files_start(lpStringa, a2, lpString2, ctx, 5);
                _InterlockedDecrement(&ctx->m.has_2);
                HeapFree(hHeap, 0, lpMem);
                lpStringa = pszStarta;
                if (!pszStarta)
                    break;
                v8 = pszStarta;
            }
        }
        else
        {
            f_enumerate_files_start_drive(a2, lpString2, ctx);
            f_iterate_list_and_call_encrypt(ctx, ctx->m.event_handle);
            SetEvent(ctx->m.event_handle);
            WaitForSingleObject(hHandle, 0xFFFFFFFF);
            CloseHandle(hHandle);
            pszStarta = 0;
        }
    }
    else
    {
        LastError = GetLastError();
        ctx->m.has_2;
        pszStartb = (POWST)(LastError);
        CloseHandle((ctx->m.event_handle));
    }
    if (!InterlockedDecrement(&ctx->m.has_2))
        f_iterate_list_cleanup_and_exit(ctx, 1);
}

```



```

31     _asm { rcr    bpl, 0CFh }
32     v14 = (unsigned int)(dword_140000D498 - 350415642);
33     ctx1 = (thread_context_lockable *)HeapAlloc(hHeap, 0, v14);
34     v16 = 0;
35     _asm { rxb    rbx, 4 }
36     ctx = ctx1;
37     if (!ctx)
38     {
39         f_memset(ctx1, 0, v14);
40         event = f_call_createEvent(0i64, 0);
41         ctx->m.event_handle = event;
42         if (!event)
43         {
44             ctx->m.next = (thread_context *)ctx->m;
45             ctx->m.prev = (thread_context *)ctx->m;
46             InitializeCriticalSection(&ctx->lock);
47             _mmovne();
48             v14 = CreateThread(0i64, 0i64, f_encryption_thread, ctx, 0, 0i64);
49             ctx->m.outside_kill_handle_maybe = outside_kill_handle_maybe;
50             ctx->m.space_func = f_get_disk_free_space;
51             ctx->m.has_2 = 2;
52             ctx->m.config_maybe = config_maybe;
53             ctx->m.has_16 = 16;
54             Thread = CreateThread(0i64, 0i64, f_encryption_thread, ctx, 0, 0i64);
55             if (!Thread)
56             {
57                 v21 = pszStart;
58                 if (!pszStart)
59                 {
59                     v22 = pszStart;
60                     do
61                     {
62                         if (*v22)
63                             break;
64                         v23 = StrChrW(v21, 0x7Cu);
65                         v21 = v23;
66                         if (*v23)
67                         {
68                             *v23 = 0;
69                             v21 = v23 + 1;
70                         }
71                         v24 = f_call_WinetGetUniversalNameW(v23);
72                         v25 = (void *)v24;
73                         if (*v24)
74                             v22 = (const WCHAR *)v24;
75                         f_enumerate_files_start(lpString, 0B164, v22, (const WCHAR *)a6, ctx);
76                         _InterlockedDecrement(&ctx->m.has_2);
77                         HeapFree(hHeap, 0, v25);
78                         v22 = v21;
79                     }
80                     while (*v21);
81                 }
82                 else
83                 {
84                     f_enumerate_files_start_drive(lpString, (const WCHAR *)a6, ctx);
85                     f_iterate_list_and_call_encrypt(ctx, ctx->m.event_handle);
86                     _InterlockedDecrement(&ctx->m.has_2);
87                     WaitForSingleObject(Thread, 0xFFFFFFFF);
88                     event_handle = Thread;
89                 }
90                 else
91                 {
92                     LastError = GetLastError();
93                     --ctx->m.has_2;
94                     event_handle = ctx->m.event_handle;
95                     v16 = LastError;
96                 }
97             CloseHandle(event_handle);
98             if (!InterlockedDecrement(&ctx->m.has_2, 0xFFFFFFFF))
99                 f_iterate_list_cleanup_and_exit(ctx);
100         }
101     }
102     else
103     {
104         _DI = v14;
105         _asm { rcr    dil, 0F4h }
106         v16 = GetLastError();
107         HeapFree(hHeap, 0, ctx);
108     }
}

```

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.

The image shows two side-by-side debugger windows, likely from IDA Pro, comparing the assembly code for RSA encryption routines. The left window is for WastedLocker and the right is for Hades. Both windows show nearly identical assembly code for RSA operations, with red arrows highlighting specific lines of code that are identical between the two samples. The code includes calls to RSA-related functions like f_RSAPublicBlock, f_RSAPublicEncrypt, and f_RSAPublicDecrypt, along with various RSA-specific arithmetic and modular exponentiation operations.

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

Both Ransomware rely on the RSAREF²² 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/>

```

1 int64 __fastcall f_RSAPublicBlock(_int64 rcx0, _DWORD *a2, _int64 a3, _int64
2 {
3     unsigned int v7; // ebx
4     int v8; // er12
5     _int64 v10; // rdx
6     char v11[528]; // [rsp+30h] [rbp-848h] BYREF
7     _int64 a1[66]; // [rsp+240h] [rbp-638h] BYREF
8     _int64 v13[66]; // [rsp+450h] [rbp-428h] BYREF
9     char v14[528]; // [rsp+660h] [rbp-218h] BYREF
10
11    j_f_NN_Decode((__int64)v11, 129i64, a3, a4);
12    j_f_NN_Decode((__int64)a1, 129i64, (__int64)(a5 + 1), 512);
13    j_f_NN_Decode((__int64)v13, 129i64, (__int64)(a5 + 129), 512);
14    v7 = j_f_NN_Digits((__int64)a1, 129i64);
15    v8 = j_f_NN_Digits((__int64)v13, 129i64);
16    if ((int)j_f_strcmp((__int64)v11, (__int64)a1, v7) >= 0)
17        return 0x4011c4;
18    j_f_NN_ModExp((int)v14, (int)v11, (int)v13, v8, (__int64)a1, v7);
19    v10 = (unsigned int)(a5 + 7) >> 3;
20    *a2 = v10;
21    j_f_NN_Encode(rcx0, v10, (__int64)v14, v7);
22    j_f_memset(v11, 0, 0x204ui64);
23    j_f_memset(v13, 0, 0x204ui64);
24    return 0164;
25 }

```

```

static int RSAPublicBlock (output, outputLen, input, inputLen, publicKey)
{
    /* output block */
    /* length of output block */
    /* input block */
    /* length of input block */
    /* RSA public key */

    NN_Decode (m, MAX_NN_DIGITS, input, inputLen);
    NN_Decode (n, MAX_NN_DIGITS, publicKey->modulus, MAX_RSA_MODULUS_LEN);
    NN_Decode (e, MAX_NN_DIGITS, publicKey->exponent, MAX_RSA_MODULUS_LEN);
    nDigits = NN_Digits (n, MAX_NN_DIGITS);
    eDigits = NN_Digits (e, MAX_NN_DIGITS);

    if (NN_Cmp (m, n, nDigits) >= 0)
        return (RE_DATA);

    /* Compute c = m^e mod n.
     */
    NN_ModExp (c, m, e, eDigits, n, nDigits);

    /* outputLen = (publicKey->bits + 7) / 8;
     */
    NN_Encode (output, *outputLen, c, nDigits);

    /* Zeroize sensitive information.
     */
    R_memset ((POINTER)c, 0, sizeof (c));
    R_memset ((POINTER)m, 0, sizeof (m));

    return (0);
}

/* Raw RSA private-key operation. Output has same length as modulus.
*/

```

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.

```

Pseudocode-A
1 DWORD __fastcall f_gen_random_bytes@ceax(BYTE *buffer, DWORD data_len@edi)
2 {
3     DWORD LastError; // esi
4     DWORD old_word; // [esp+8h] [ebp-8h]
5     HCRYPTPROV phProv; // [esp+Ch] [ebp-4h] BYREF
6
7     LastError = 0;
8     old_word = get_first_word(buffer, data_len);
9     if (!CryptAcquireContext((phProv, 0, 0, CRYPT_INITIATOR|CRYPT_VERIFYCONTEXT)))
10    {
11        if (!CryptGenRandom(phProv, data_len, buffer))
12            LastError = GetLastError();
13        CryptReleaseContext(phProv, 0);
14    }
15    else
16    {
17        LastError = GetLastError();
18    }
19    if (LastError || old_word == get_first_word(buffer, data_len))
20        return f_alternative_random(buffer, data_len);
21    return LastError;
22 }

000003C7 f_gen_random_bytes::i (408EC7) 001B7E0B f_gen_random_bytes::i (1401BBCDB)

Pseudocode-B
1 DWORD __fastcall f_gen_random_bytes(BYTE *buffer, DWORD data_len)
2 {
3     DWORD old_word; // er12
4     CryptApi::windows_crypt_api(); // rax
5     CryptGenRandom(CryptApi::windows_crypt_api()); // rip
6     DWORD LastError; // edi
7     HCRYPTPROV phProv; // [rsp+40h] [rop+18h] BYREF
8
9     old_word = get_first_word(buffer, data_len);
10    if (windows_crypt_api == f_get_windows_crypt_api())
11        if (windows_crypt_api->CryptAcquireContext((phProv, 0x04, 0x04, CRYPT_INITIATOR|CRYPT_VERIFYCONTEXT)))
12            if (windows_crypt_api->CryptGenRandom(phProv, data_len, buffer))
13                LastError = 0;
14            else
15                LastError = GetLastError();
16        else
17            LastError = GetLastErrorMessage();
18    else
19        if (windows_crypt_api->CryptReleaseContext(phProv, 0))
20            LastError = 0;
21        else
22            LastError = GetLastError();
23    else
24        LastError = GetLastErrorMessage();
25    else
26        LastError = ERROR_PROC_NOT_FOUND;
27    else
28        if (LastError || old_word == get_first_word(buffer, data_len))
29            return f_alternative_random(buffer, data_len);
30    return LastError;
31 }
32
33 if (LastError || old_word == get_first_word(buffer, data_len))
34     return f_alternative_random(buffer, data_len);
35
36 return LastError;
37 }

001B7E0B f_gen_random_bytes::i (1401BBCDB)

```

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.

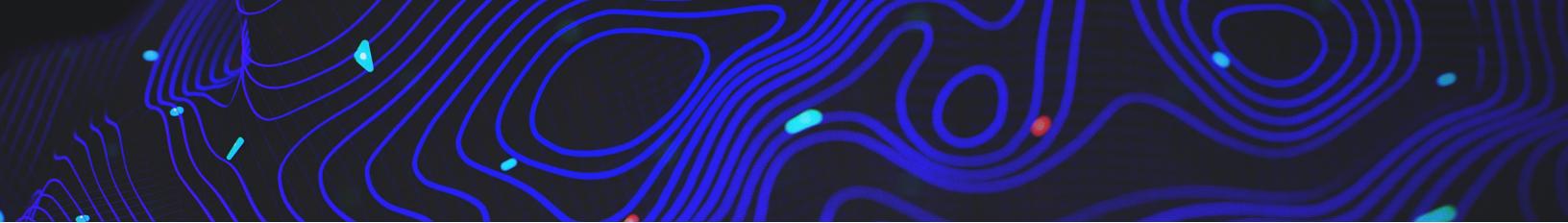
HADES VS. PHOENIX LOCKER: A POLYMORPHIC CODE

For this analysis, we investigated the following samples:

Family	Sha256 Packed	Sha256 unpacked	Compilation timestamp	Submission to VT
Hades	e657ff4838e47465 3b55367aa9d4a06 41b35378e2e379ad 0fd1631b3b763ef0	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 BitPaymer²³. 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	b4061d4227 e08cfaa3190 dea9926571 fca2736a1 unpacked: f58c980816 ef122973db bba2c66f8c 6850f45764	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	3cb0cb07cc 2542f1d980 60adccda72 6ea865db98 Unpacked: 53185223c0 36f010faa1e 077b4eaa59 7392ca76c	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.

²⁴see the timing analysis section

²⁵see the timing analysis section

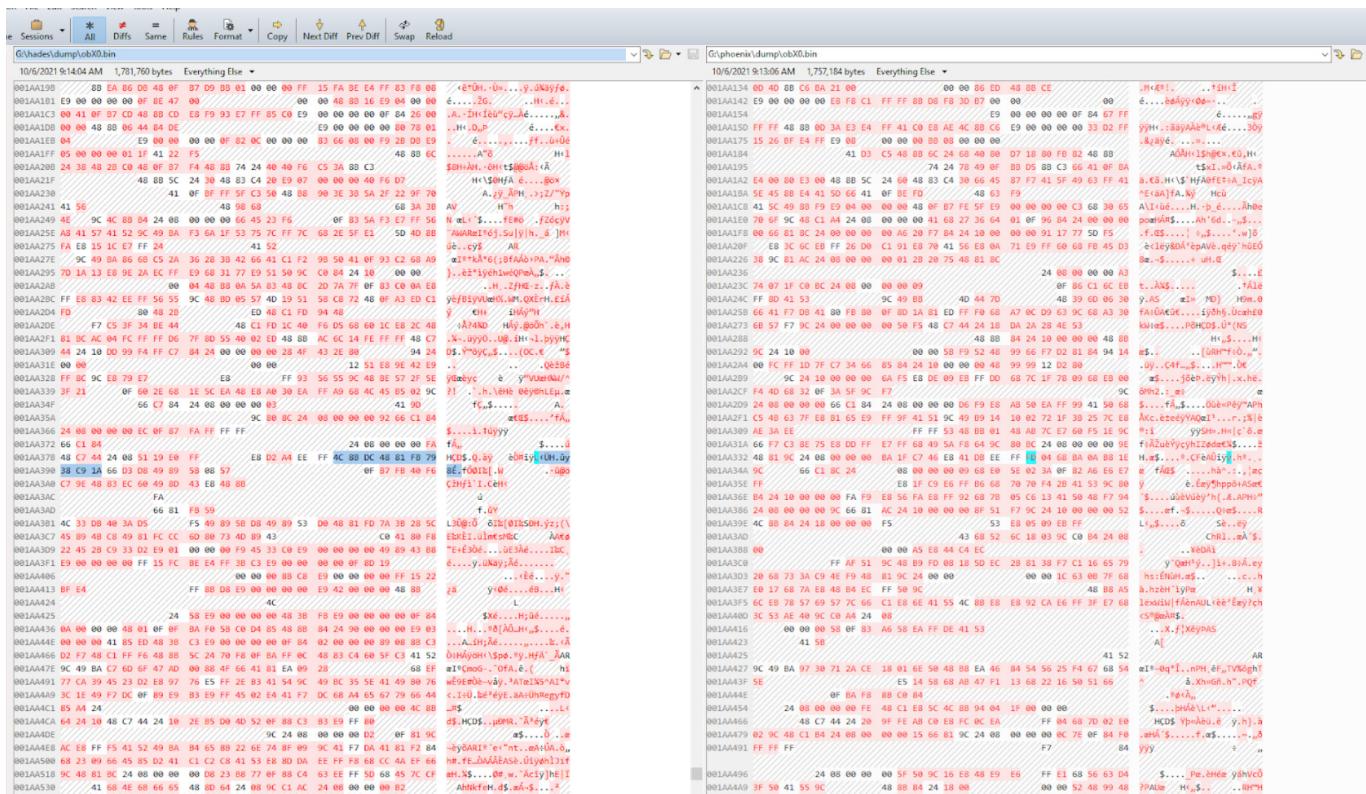


Fig 9: Hades(left), PhoenixLocker(right). “A hex view of the .obX0 section”.

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:

```

Procedure-A
1 _int64 __fastcall f_enumerate_files_start_drive(LPCWSTR lpString, _int64 a2, _QWORD *a3)
2 {
3     DWORD LogicalDriveStringW; // edi
4     WCHAR v6[5]; // rax
5     _QWORD v7; // rbp
6     const WCHAR v8[5]; // rdx
7     unsigned int i; // edi
8     void *v9; // rcx
9     _QWORD v10; // rsi
10    _QWORD v11; // rdx
11    unsigned int __fastcall v12(_QWORD, const WCHAR *); // r8
12    _int64 v13; // rsi
13    void v14; // rbp
14    _QWORD v15; // [rsp+10h] BYREF
15    _QWORD v16; // [rsp+10h] BYREF
16    _QWORD v17; // [rsp+10h] BYREF
17    _QWORD v18; // [rsp+10h] BYREF
18    _QWORD v19; // [rsp+10h] BYREF
19    _QWORD v20; // [rsp+10h] BYREF
20    if (!v6)
21    {
22        memset(v6, 0, 264 = (LogicalDriveStringW + 1));
23        GetLogicalDriveStringW((LogicalDriveStringW + 1));
24        v6 = v7;
25        for (i = 0; v6 < v8; v6 += 1)
26        {
27            v10 = (void *)v11;
28            if (v10 && !WaitForSingleObject(v10, 0) != 258)
29            {
30                if (GetDriveType(v6) == 3)
31                {
32                    v11 = v6;
33                    v12 = __fastcall v12(_QWORD, const WCHAR *)a3[1];
34                    v13 = v11;
35                    if ((v12 || (v12[a14] & v10)) && (unsigned __int64)v13 + 4 < 0x10)
36                    {
37                        v14 = (void *)call_MetGetUniversalName();
38                        if (!v14)
39                            lstrcpyW(vString, v8);
40                        else
41                            f_enumerate_files_start(lpString, _int64)a3);
42                    }
43                    if (v14)
44                        HeapFree(Heap, 0, v14);
45                }
46            }
47        }
48    }
49    HeapFree(Heap, 0, v7);
50    if (v10)
51    {
52        return 0;
53    }
54    return 1;
55}
56

Procedure-B
1 _int64 __fastcall f_enumerate_files_start_drive(LPCWSTR lpString, const _QWORD *a2, _QWORD *a3)
2 {
3     DWORD LogicalDriveStringW; // edi
4     _QWORD LogicalDriveStringW; // edi
5     _QWORD v6; // rax
6     _QWORD v7; // rbp
7     const _QWORD v8[5]; // rdx
8     unsigned int i; // edi
9     void *v10; // rcx
10    _QWORD v11; // rsi
11    unsigned int __fastcall v12(_QWORD, const _QWORD *); // r8
12    _int64 v13; // rsi
13    void v14; // rbp
14    const _QWORD v15; // rdx
15    _QWORD v16; // [rsp+10h] BYREF
16    _QWORD v17; // [rsp+10h] BYREF
17    _QWORD v18; // [rsp+10h] BYREF
18    _QWORD v19; // [rsp+10h] BYREF
19    _QWORD v20; // [rsp+10h] BYREF
20    if (!v6)
21    {
22        memset(v6, 0, 264 = (LogicalDriveStringW + 1));
23        GetLogicalDriveStringW((LogicalDriveStringW + 1));
24        v6 = v7;
25        for (i = 0; v6 < v8; v6 += 1)
26        {
27            v10 = (void *)v11;
28            if (v10 && !WaitForSingleObject(v10, 0) != 258)
29            {
30                if (GetDriveType(v6) == 3)
31                {
32                    v11 = v6;
33                    v12 = __fastcall v12(_QWORD, const _QWORD *)a3[1];
34                    v13 = v11;
35                    if ((v12 || (v12[a14] & v10)) && (unsigned __int64)v13 + 4 < 0x10)
36                    {
37                        v14 = (void *)call_MetGetUniversalName();
38                        if (!v14)
39                            lstrcpyW(vString, v8);
40                        else
41                            f_enumerate_files_start(lpString, _int64)a3);
42                    }
43                    if (v14)
44                        HeapFree(Heap, 0, v14);
45                }
46            }
47        }
48    }
49    HeapFree(Heap, 0, v7);
50    if (v10)
51    {
52        return 0;
53    }
54    return 1;
55}
56

```

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.

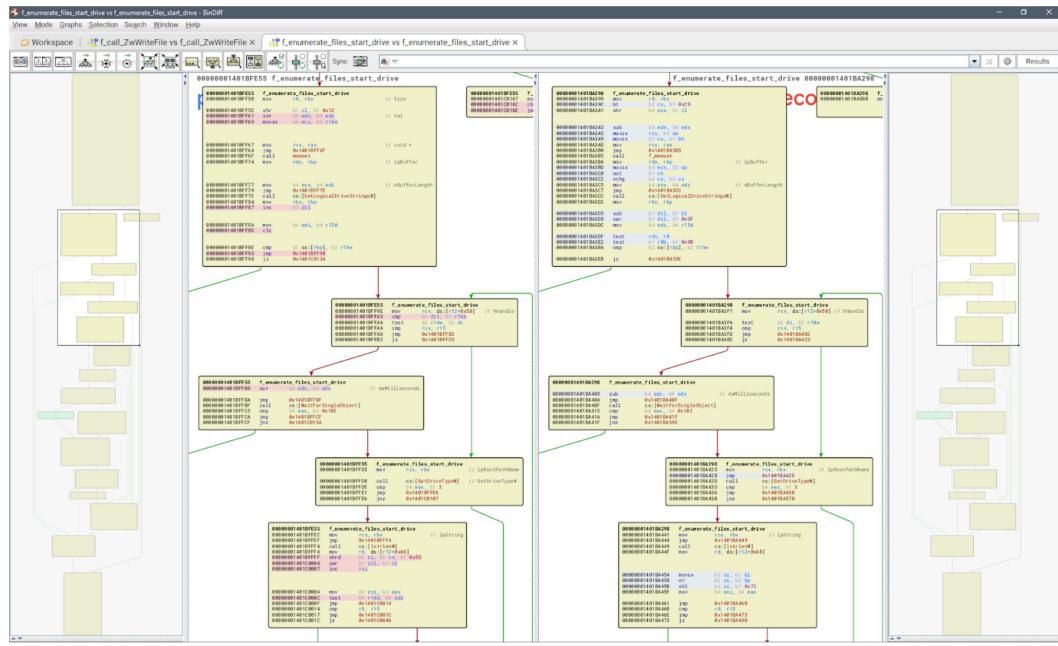


Fig 11: Hades(left), PhoenixLocker(right). “Polymorphic code added statically highlighted in red and blue”.

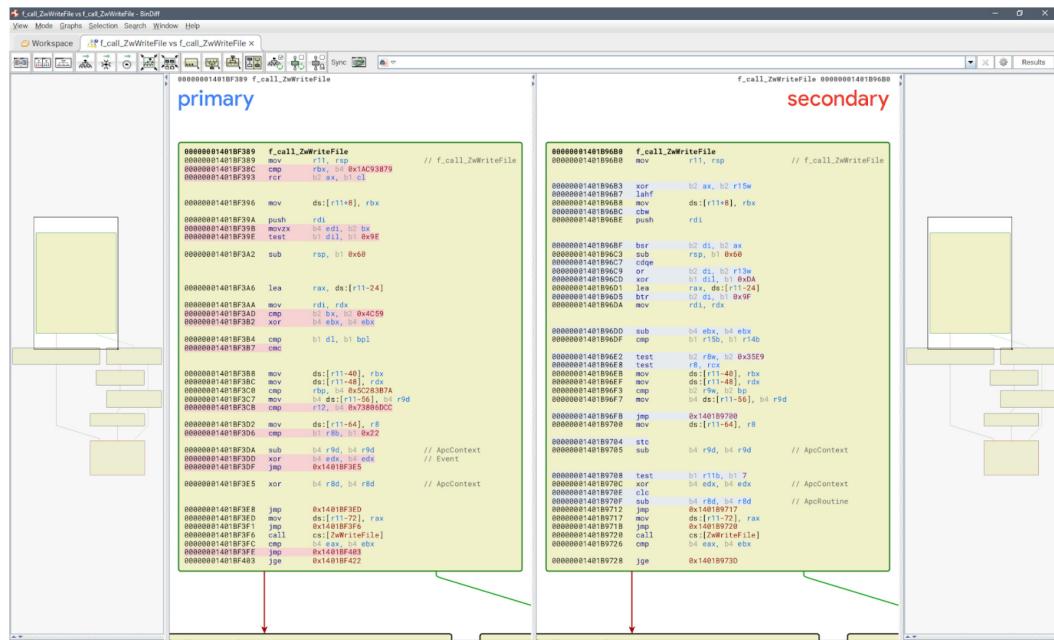


Fig 12: Hades(left), PhoenixLocker(right). “Polymorphic code added statically highlighted in red and blue”.

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 ac49d68cdcbc1d136	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:

Similarity %	Confidence	Address	Primary Name	Type	Address	Secondary Name	Type	Basic Blocks	Jumps
0.90	0.92	0000000140...	sub_1401B8819	Normal	0000000140...	sub_1401C3F08	Normal	0 5 0 0 6 0	
0.90	0.93	0000000140...	sub_1400001098	Normal	0000000140...	sub_14000099E0	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_1401B966B0	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_1400009FC	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	
0.01	0.02	0000000140...	sub_1401D00A0	Normal	0000000140...	sub_1401C6207	Normal	0 10 0 0 25 0	

Fig 14

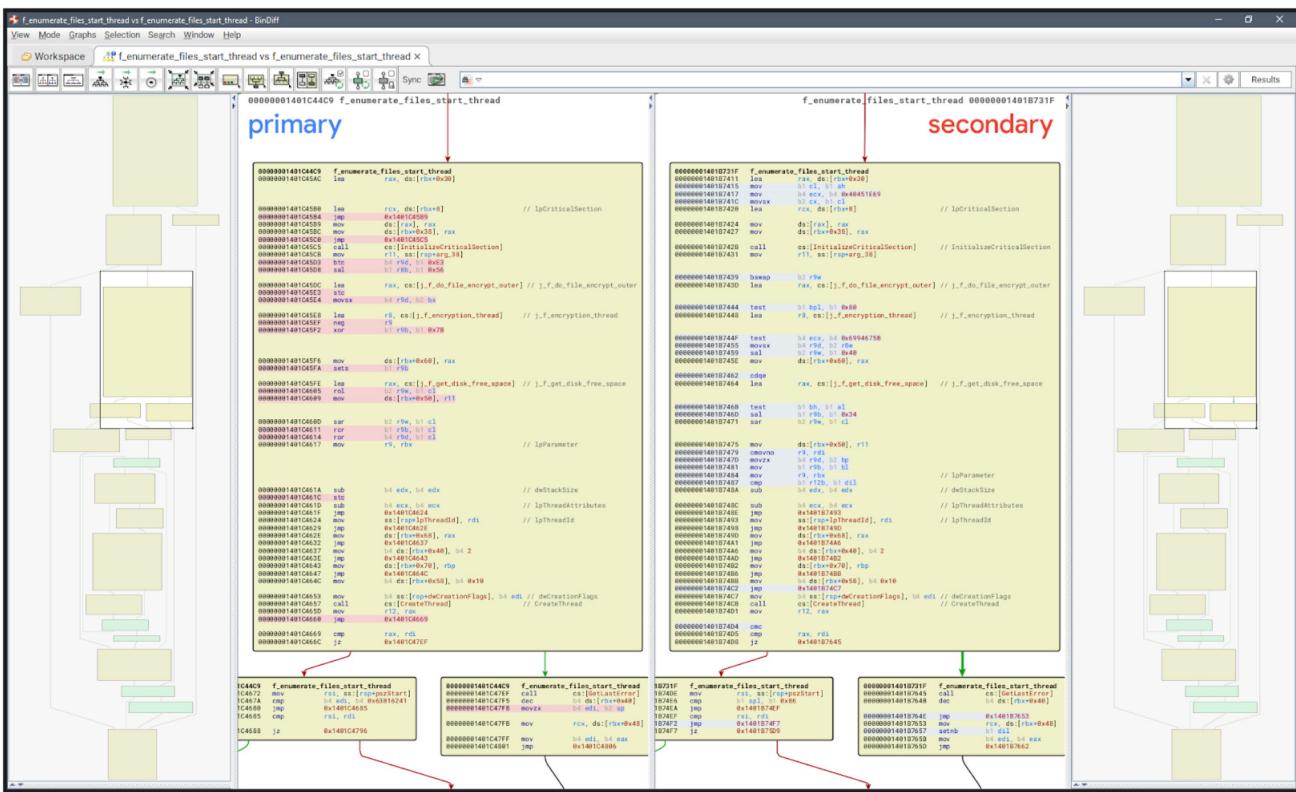
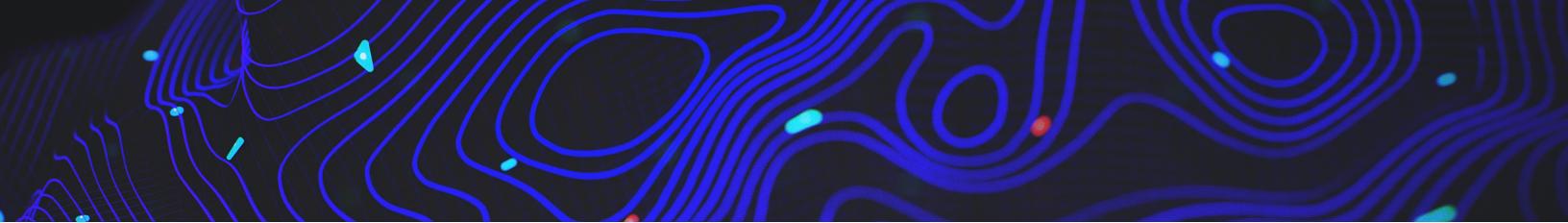


Fig 15: PayloadBIN (left) vs PhoenixLocker cryptolocker (right). File enumerating functions are practically identical.



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

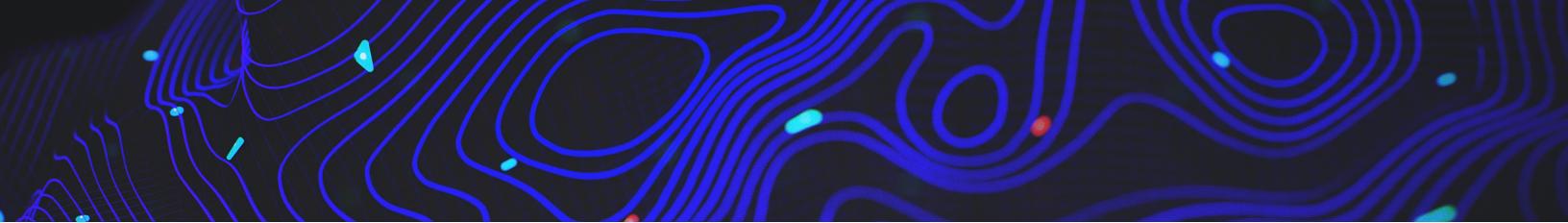
Whilst syntax difference may seem like a significant difference, these two implementations are very similar: the logic is the same, only the signature changes.

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	<pre>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</pre>
Hades	<pre>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”</pre>
PhoenixLocker	<pre>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</pre>
PayloadBin	<pre>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</pre>

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
	<pre>* NTLDR/* BOOTMGR/* GRLDR/*.386/*.ps1/*.*.msu/*.*.ani/*.*.wpk/*.*.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/*.*.curl/*.*.mod/*.*.shs/*.*.rtp/*.*.rom/*.*.msp/*.*.ini/*.*.bak/*.*.dat/*.*.sdi/*.*.wim/*.*.dll/*.*.exe</pre>
WastedLocker	<pre>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)</pre>
Hades	<pre>*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/*.*.wpk/*.*.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 *</pre>
PhoenixLocker	<pre>*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/*.*.wpk/*.*.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 *</pre>
PayloadBin	<pre>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/*.*.wpk/* 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</pre>

CLUSTERING HADES, PHOENIXLOCKER AND PAYLOADBIN

Another interesting overlap emerges from the analysis of two unpacked samples belonging to Hades and PhoenixLocker. .data md5:3fc7f20aabef2d3290e3b71d7f639fd

THE .DATA SECTION:

The .data section of some Hades samples is identical to those found in PayloadBIN and PhoenixLocker: .data md5:3fc7f20aabef2d3290e3b71d7f639fd

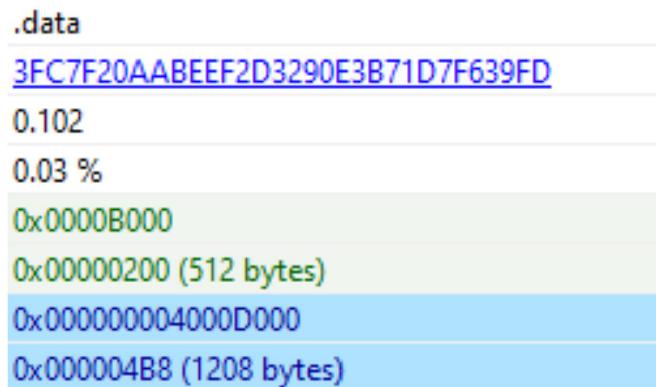
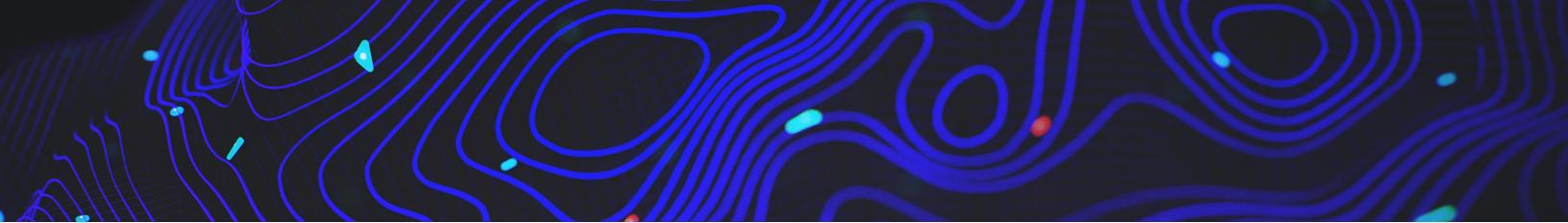


Fig 16

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
7bcea3fbfc4c170c57c9050499e1fae40f5d731	Hades
e23637ea81751e558fca17ef1a54b6e39d2e83c3	PayloadBIN

THE \\BASENAMEDOBJECTS\\SCRIPTNET MUTEX:

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

Sha1	Ransomware Family
16aa95ff91ccf05e5920858f9f637abf2511e57	PayloadBin
3cb0cb07cc2542f1d98060adccda726ea865db98	PhoenixLocker
d0d68281f8459b5558559fbff8c6c8ab4ddfec8b	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:

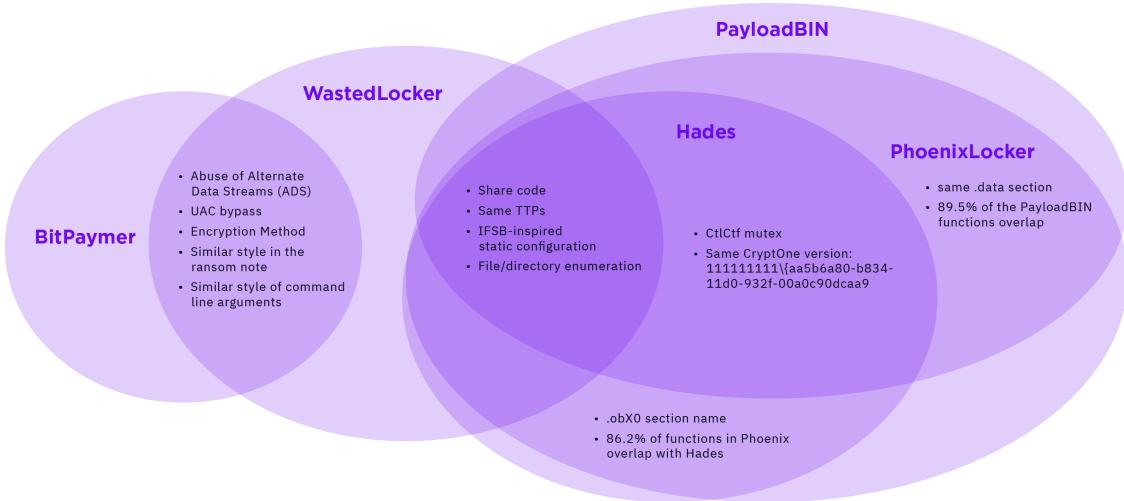


Fig 17: The Venn diagram of the overlaps between the ransomware families.

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.

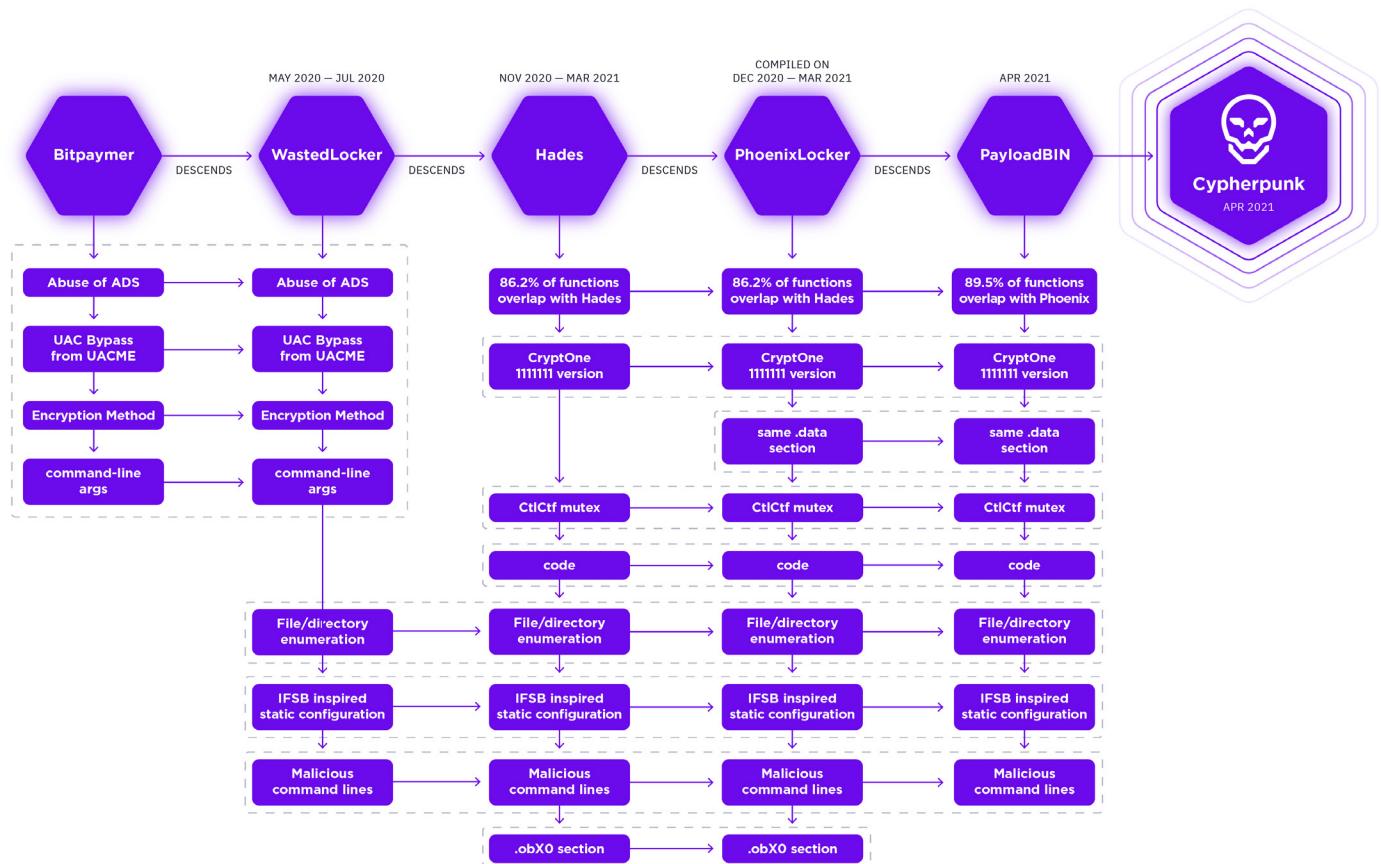


Fig 18: The evolution and overlaps between the different ransomware variants

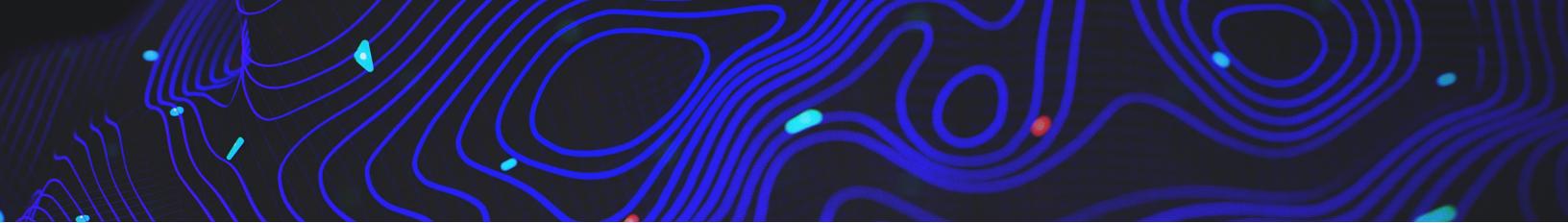


Fig. 18 clearly identifies the existence of a single cluster which presents very strong code overlap. Taking a look at the timeline, the different variants were released in a very short time period, differing mostly with simple “cosmetic” changes. From WastedLocker to Hades, however, the time between 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 "Cyberpunk" - we named it this due to the encryption extension found.

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

Files encrypted with the extension “cyberpunk”

Code similarity analysis shows that the Cyberpunk 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	Cyberpunk	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 Olympus²⁸. A Few days later, Sinclair Broadcast Group systems were also attacked, causing wide disruption²⁹. Some researchers claimed a possible relation with WastedLocker³⁰, but at the time of writing, no one provided further details about this assessment.

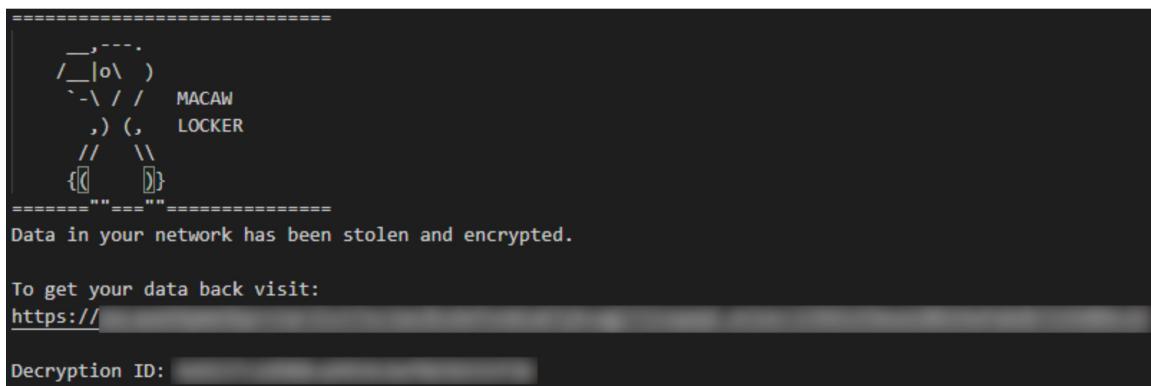


Fig 19: The content of the ransom note used in Macaw

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/>

```

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 }

```

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.

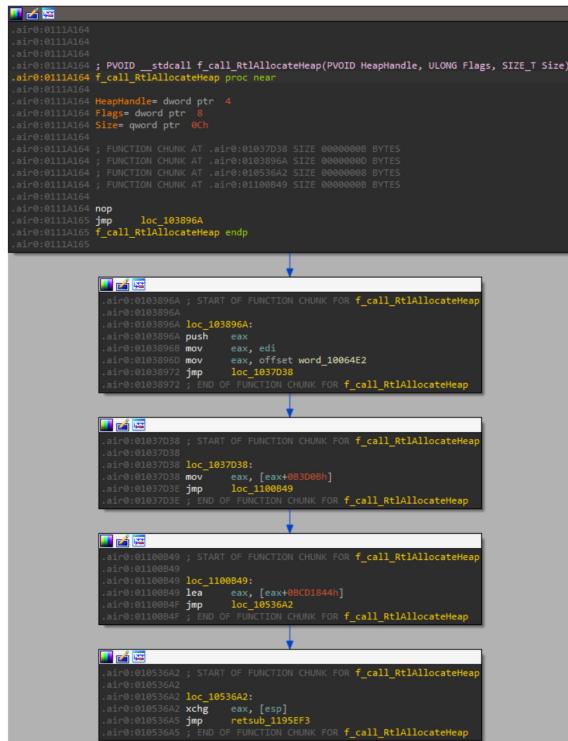


Fig 21: One example of indirect calls used in Macaw just to invoke a call to RtlAllocateHeap, calculating the destination address in EAX, exchanging the resulting value with the top of the stack and jumping to it using a “ret” instruction

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).

```

1 int __stdcall f_enumerate_files_start_drive(thread_context_lockable *ctx)
2 {
3     // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL+* TO EXPAND]
4
5     ctx = g_context;
6     f_memcpy(&v0, &g_context->fix_api_offset_value, drive_root); // copy from "\\\\".
7     GetLogicalDriveStringsW_1 = ctx->pf_get_api_by_hash(ctx, ctx->fix_api_info_offset_value, 0xC266C48F);
8     if (!GetLogicalDriveStringsW_1)
9     {
10        length = GetLogicalDriveStringsW_1(0, 0);
11        if (length == 0)
12        {
13            byte_size = 2 * length + 2;
14            LOGWORD(v20) = byte_size;
15            drives = RtlAllocateHeap(ctx->hHeap, 0, v20);
16            if (!drives)
17            {
18                f_cleanse(drives, array_length, drives);
19                result = 0;
20                while (!drive_root_path)
21                {
22                    if (WaitForSingleObjectEx((HANDLE)ctx->outside_kill_handle_maybe, 0, FALSE))
23                    {
24                        if (GetFileTypeW(drive_root_path) == 3)
25                        {
26                            v11 = lstrlenW(drive_root_path);
27                            space_func = ctx->m.space_func;
28                            v13 = v11;
29                            if ((!space_func || !space_func(ctx->m.config.maybe, drive_root_path)) && (v13 + 4) < 0x10)
30                            {
31                                v14 = f_call_MNetGetUniversalNameW(drive_root_path);
32                                if (!v14)
33                                {
34                                    lstrcpyW(root_drive1, drive_root_path);
35                                    i = f_enumerate_files_start(root_drive1, ctx);
36                                    if (i)
37                                    {
38                                        HeapFree(hHeap, 0, drives);
39                                    }
40                                }
41                            }
42                        }
43                    }
44                }
45            }
46        }
47    }
48    return 8;
49 }
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
817
818
819
819
820
821
822
823
823
824
825
825
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1610
1611
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1620
1621
1621
1622
1622
1623
1623
1624
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1
```



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 2015³¹. Early versions were used by an assortment of different malware families such as Gozi, Dridex, and Zloader. In 2019, Bromium³² analyzed and reported it as in use by Emotet. In June 2020, NCC Group³³ reported that CryptOne was used to pack WastedLocker. In 2021, researchers³⁴ 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}
- 1InterfacE\{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}
- rrererfacE\{b196b287-bab4-101a-b69c-00aa00341d07}

The first part of the string is composed of a custom string (111111111, 1InterfacE, 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 ac11a7f644bcfed963	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	7bcea3fbfc4c170c57c9 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<https://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>

⁴¹based on VT submission date

⁴²based 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
	f25f0b369a355f30f5e11ac11a7f644bcfef963	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
	e7784e31dfd1a6f6291bc a78bf11b6145d37d753	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	d0d68281f8459b5558559fbff8c6c8ab4ddfec8b	8	2020-12-20 21:59:21	2020-12-22 12:15:52	1111111111
	f8fc84030c579070b36c99c836ac4b5c32bbc2c4	9	2020-12-20 21:54:17	2020-12-21 17:22:06	1111111111
	7bcea3fbfc4c170c57c9050499e1fae40f5d731	10	2020-12-20 21:59:21	2021-01-07 06:56:0	1111111111
	b4061d4227e08cfaa3190dea9926571fca2736a1	11	2021-03-06 18:13:45	2021-03-26 16:55:45	1111111111
PhoenixLocker	f8e52380b6f3668d4de6df416c8da389c0d98fe8	12	2021-03-06 05:26:01	2021-03-26 17:46:55	1111111111
	3cb0cb07cc2542f1d98060adccda726ea865db98	13	2021-03-20 20:00:50	2021-03-26 17:48:41	1111111111
PayloadBIN	e23637ea81751e558fca17ef1a54b6e39d2e83c3	14	2021-06-02 19:12:49	2021-06-03 13:35:16	1111111111

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

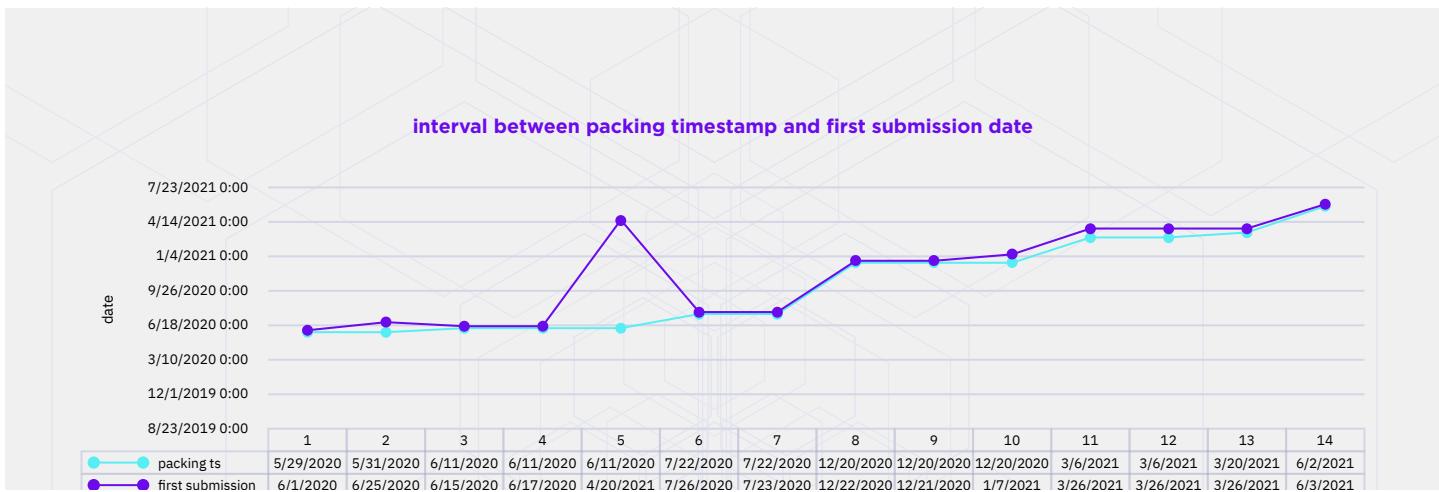


Fig 27

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: SCSTXPBIMRJPFWKHA
 Issuer: SCSTXPBIMRJPFWKHA
 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 remains 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:

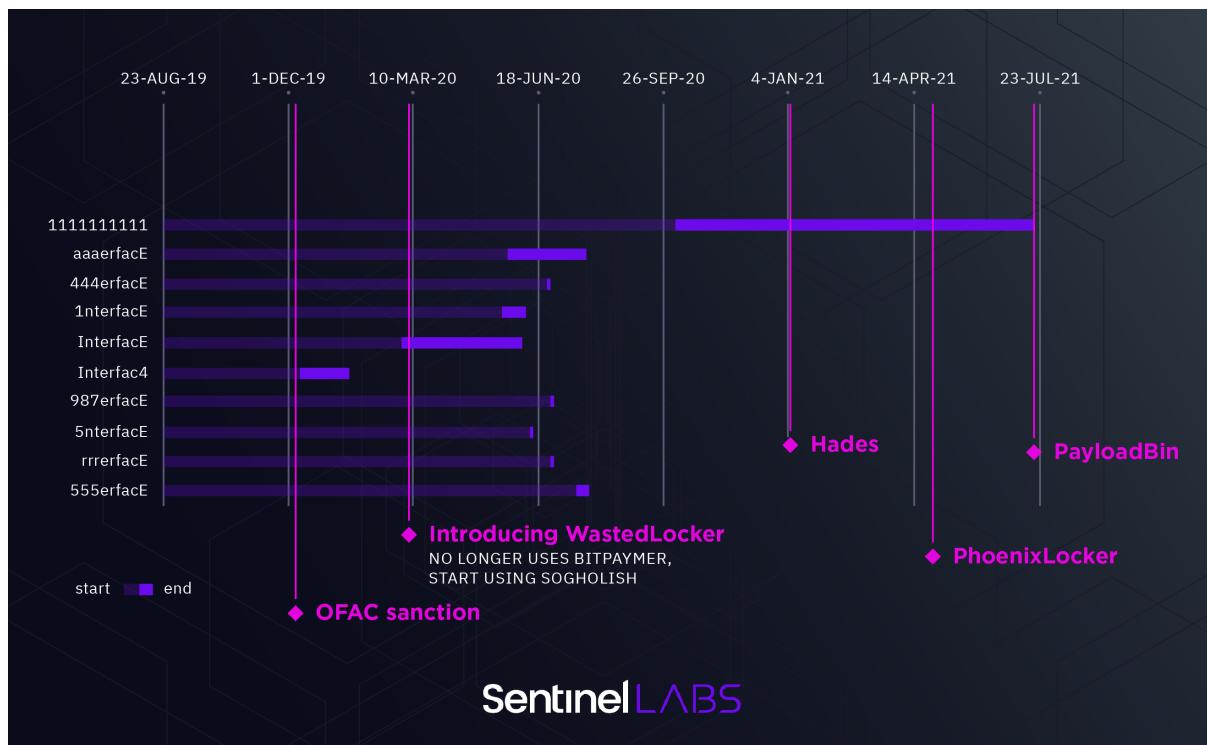


Fig 28

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 Truesec⁴³ 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 C2⁴⁴.

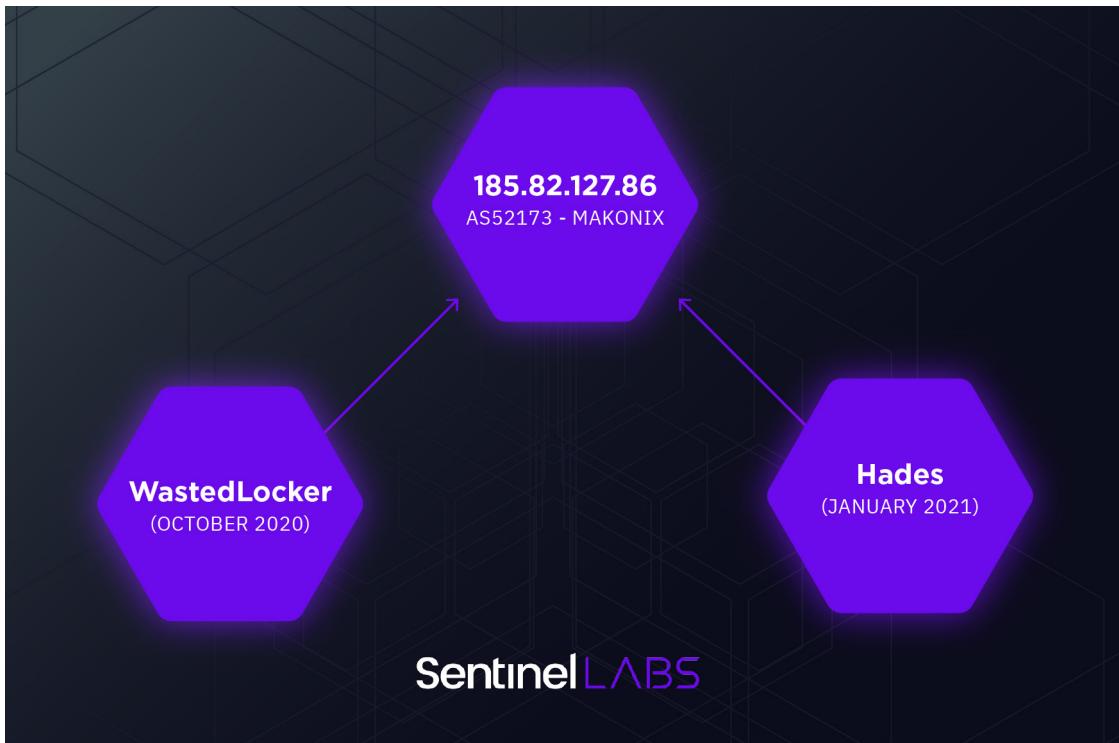


Fig 29

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 SOC GHOOLISH 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

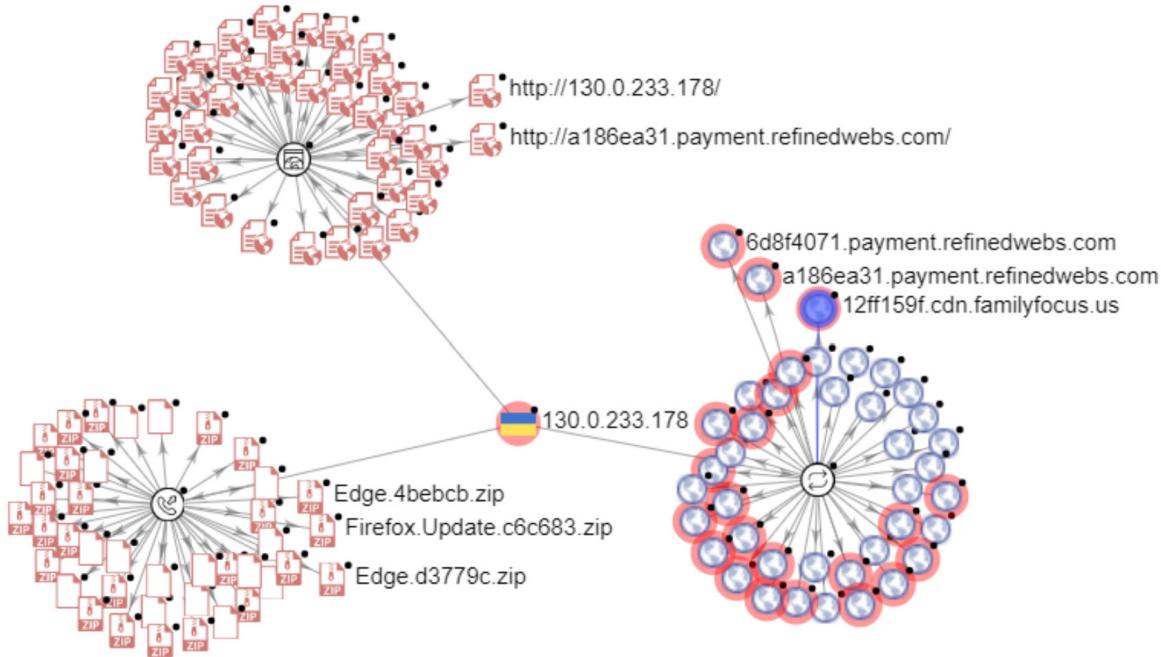


Fig 30

This IP was used in 2019 to spread BitPaymer and DoppelPaymer by deploying Dridex⁴⁵ loader. A few months later it was reported to have also spread WastedLocker⁴⁶ and Hades Ransomware⁴⁷.

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<https://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-ransomware-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 lackluster 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, prior 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, prior 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, prior to ransomware deployment.
Discovery	T1087.002 Account Discovery: Domain Account ⁶⁹	This technique has been observed in the SocGholish attack chain, prior to ransomware deployment.
Discovery	ID: T1482 - Domain Trust Discovery ⁷⁰	This technique has been observed in the SocGholish attack chain, prior 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	<pre> 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*)) }</pre>
CryptOne 111111 version (Hades/ Phoenix/PayloadBIN	<pre> rule CryptONE_111111Version { meta: Author = "SentinelLabs" Family = "Evil Corp CryptOne" strings: \$str1 = "11111111\\{aa5b6a80-b834-11d0-932f-00a0c90dcaa9}" wide ascii \$str2 = "11111111\\{b196b287-bab4-101a-b69c-00aa00341d07}" wide ascii condition: any of them }</pre>
SocGholish loaders	<pre> rule MAL_JS_SocGholish_Mar21_1 : js socgholish { meta: description = "Triggers on SocGholish JS files" author = "Nils Kuhnert" date = "2021-03-29" hash = "7ccbdcd5a9b30f0b2b866a5ca173063dec7bc92034e7cf10e3eebfff017f3c23" hash = "f6d738baea6802cbbb3ae63b39bf65fb641a1f0d2f0c819a8c56f677b97bed1" 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*) }</pre>

Hades Ransomware	<pre> import "pe" section name ".obX0" 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")) } </pre>
PayloadBIN digital certs	<pre> 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)) } </pre>

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	CyberPunk contact info
email (contact information)	jeey5o@tutanota.de	CyberPunk contact info
email (contact information)	rei5ah@protonmail.com	CyberPunk contact info
certificate	Name QEXAYFGTEHMURVBPT 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: OTRBXJVNNJOIXXBVF0 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.accountabilitypartner.com
domain	*.login.wwpcrisis.com
domain	*.login.markbrey.com
domain	*.nodes.fiorescence.com
domain	*.push.youbashboutique.com
domain	*.office.drpease.com

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

Type	CobaltStrike Loader	Type	CobaltStrike Loader
IP Address	23.227.193[.]137	Domain	Currentteach[.]com
IP Address	138.124.180[.]216	Domain	Newschools[.]info
IP Address	37.48.84[.]156	Domain	firsino[.]com
IP Address	179.43.169[.]30	Domain	potasip[.]com
IP Address	79.110.52[.]138	Domain	adsmarketart[.]com
IP Address	81.4.122[.]193	Domain	advancedanalysis[.]be
IP Address	195.189.96[.]41	ASN	50340, 45102, 49505
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		

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.

Address	Hex	ASCII
0000000000004011C6	00 EE 05 00 54 C4 74 50 61 CE 63 41 77 C5 72 65	İ..TÄtpäicAwÄre Ö...i...-irgöäl
0000000000004011D6	60 D2 00 00 13 A1 00 00 13 F7 69 72 47 D4 61 6C	Rilo0...i...iVi
0000000000004011E6	52 CD 6C 6F 30 A1 00 00 13 A1 00 00 13 A1 56 69	!ouayärei ...j..
0000000000004011F6	21 D5 75 61 FF E6 72 65 EE A0 00 00 13 A1 00 00	.j..UyĘap%ćewi&Fi
000000000000401206	13 A1 00 55 FD CB 61 70 BD C7 65 77 CC C6 46 69	cA...i...YçrtężłP
000000000000401216	E7 C3 00 00 13 A1 00 00 DD C7 72 74 C6 BF 6C 50	Äite'ō...i...ioa
000000000000401226	C1 CD 74 65 80 D4 00 00 13 A1 00 00 13 ED 6F 61	'ib.AryNUA..iMM
000000000000401236	AF EC 69 62 81 C0 72 79 4E D9 41 00 13 A1 4D 4D	MM.....iGejiodIeH
000000000000401246	4D	MM...iGejiodIeH
000000000000401256	4D 4D 00 00 13 A1 47 65 67 EC 6F 64 5E CD 65 48	rídlnà...i.CAÄat
000000000000401266	72 CF 64 6C 6E E0 00 00 13 A1 00 43 41 C4 61 74	6çil.à...i...i..
000000000000401276	36 E7 69 6C 2E E0 00 00 13 A1 00 00 13 A1 00 00	@ÄtF*fePÜcñtiö..
000000000000401286	40 C4 74 46 2A CD 65 50 Fc c7 0e 74 EE D2 00 00	.i...öriçÄfi=A..
000000000000401296	13 A1 00 00 13 F6 72 69 E7 C3 46 69 F7 C3 00 00	.i...i...i;cł'ñeH
0000000000004012A6	13 A1 00 00 13 A1 00 00 13 A1 43 6C B4 D1 65 48	-idi@...j...j..G
0000000000004012B6	B2 CE 64 6C AE A0 00 00 13 A1 00 00 13 A1 00 47	¶ÓTe ĐPa.ÉA.MMM
0000000000004012C6	B6 D4 54 65 A6 D0 50 61 87 C9 41 00 4D 4D 4D 4D	MM.....iGejiodIeH
0000000000004012D6	4D	MM.....iGejiodIeH
0000000000004012E6	13 A1 00 00 13 A1 00 00 67 D0 74 72 /F C6 6E 41	i...i...gđtr.ÁnA
0000000000004012F6	13 A1 00 00 13 A1 00 00 13 A1 00 00 13 CD 73 74	i...i...i...i...ist
000000000000401306	41 C0 61 74 52 A1 00 00 13 A1 00 00 13 A1 00 00	AAatR...i...i..
000000000000401316	13 A1 52 74 2F E2 64 64 4D D6 6E 63 07 CA 6F 6E	.iRt/äddMÖnc.Éon
000000000000401326	DF C1 62 6C F6 A0 00 4D EE CF 73 61 F4 C5 42 6F	BAblö .Míisaôabo
000000000000401336	EB E1 00 00 13 A1 00 00 13 A1 00 00 D4 C5 74 50	éá...i...i...öÄtP
000000000000401346	C1 CB 63 41 B7 C4 72 65 C0 CF 00 00 DF CB 61 64	ÆCa.ÁreAí..Bead
000000000000401356	D7 C9 62 72 B2 CE 79 45 98 E1 4D 4D 4D 4D 4D 4D 4D	xEbr=ÍyE.áMMMM
000000000000401366	4D	MM.....iGejiodIeH
000000000000401376	78 C6 /2 6E 6E CD 33 32 2D CS 6C 6C 13 A1 00 00	Xérnni32-AII.i..
000000000000401386	5E C6 73 73 72 C4 65 42 6C D9 41 00 13 A1 00 00	^AssrÄeBlüA..i..
000000000000401396	46 D0 65 72 00 8F 2E 64 2F CD 00 00 13 A1 00 00	Fđer...d/f...i...i..

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**.

```

000000000047386F CC int3
0000000000473870 8B05 B6180000 mov eax,dword ptr ds:[47512C]
0000000000473876 8B0D C0180000 mov ecx,dword ptr ds:[47513C]
000000000047387C 33C8 xor ecx,eax
000000000047387E 48:8B05 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

```

ecx=5074C454
eax=A113

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.

```

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 .....VirtualAlloc
63 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 55 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 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 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.....Istr
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 00 .lstrcmpA...
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 00 00 00 20 00 00 00 40 00 00 00 00 .....
48 83 C4 40 FF E1 00 DC 05 00 A4 59 90 00 EA 03 H.A@yá.Ü.¤Y.è.
00 00 ED 03 00 00 FE FB 00 00 31 03 00 00 E9 03 ..i...þù.1...é.
00 00 29 04 00 00 E9 03 00 00 E9 03 00 00 E9 03 ..)....é....é.
00 00 E9 03 00 00 E9 03 00 00 E9 03 00 00 E9 03 ..é...é...é...é.
00 00 E9 03 00 00 F1 04 00 00 E7 1A BA 0E E9 AF ..é..ñ..ç..é.
09 CD C8 BB 01 4C AC 25 54 68 00 77 20 70 FB 6A .íÈ»..L-‰Th.w þúj
67 72 C8 68 20 63 C8 69 6E 6F DD 23 62 65 09 76 grEh cÈinoY#be.v
75 6E 09 6D 6E 20 A5 4A 53 20 C4 6A 64 65 FF 08 un.mn ¥JS Aldev.

```

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.

```

. 48:8B4424 20    mov rax,qword ptr ss:[rsp+20]
. 891408          mov dword ptr ds:[rax+rax],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]
. 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]
. 891408          mov dword ptr ds:[rax+rax],edx
. EB B2           jmp 4EEAC6
> 48:83C4 18     add rsp,18
. C3              ret
. CC              int3
. CC              int3

```

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 :.'i!..Li!Th						
0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 ..°..'.i!..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 .ÍXÐ§\p.I_Þ§\p						
81 CF 59 DF 5D A7 5C DE 81 CF 5D DF D9 A7 5C DE .ÍYß]§\p.I_ßÛ§\p						
DA A7 5D DE BA A7 5C DE D1 C8 59 DF FF A7 5C DE Ú§]þ§\pNEÝþ§\p						
D1 C8 58 DF CA A7 5C DE D1 C8 5F DF D2 A7 5C DE ÑEXÞÈ§\pÑE BO§\p						

Fig 35

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

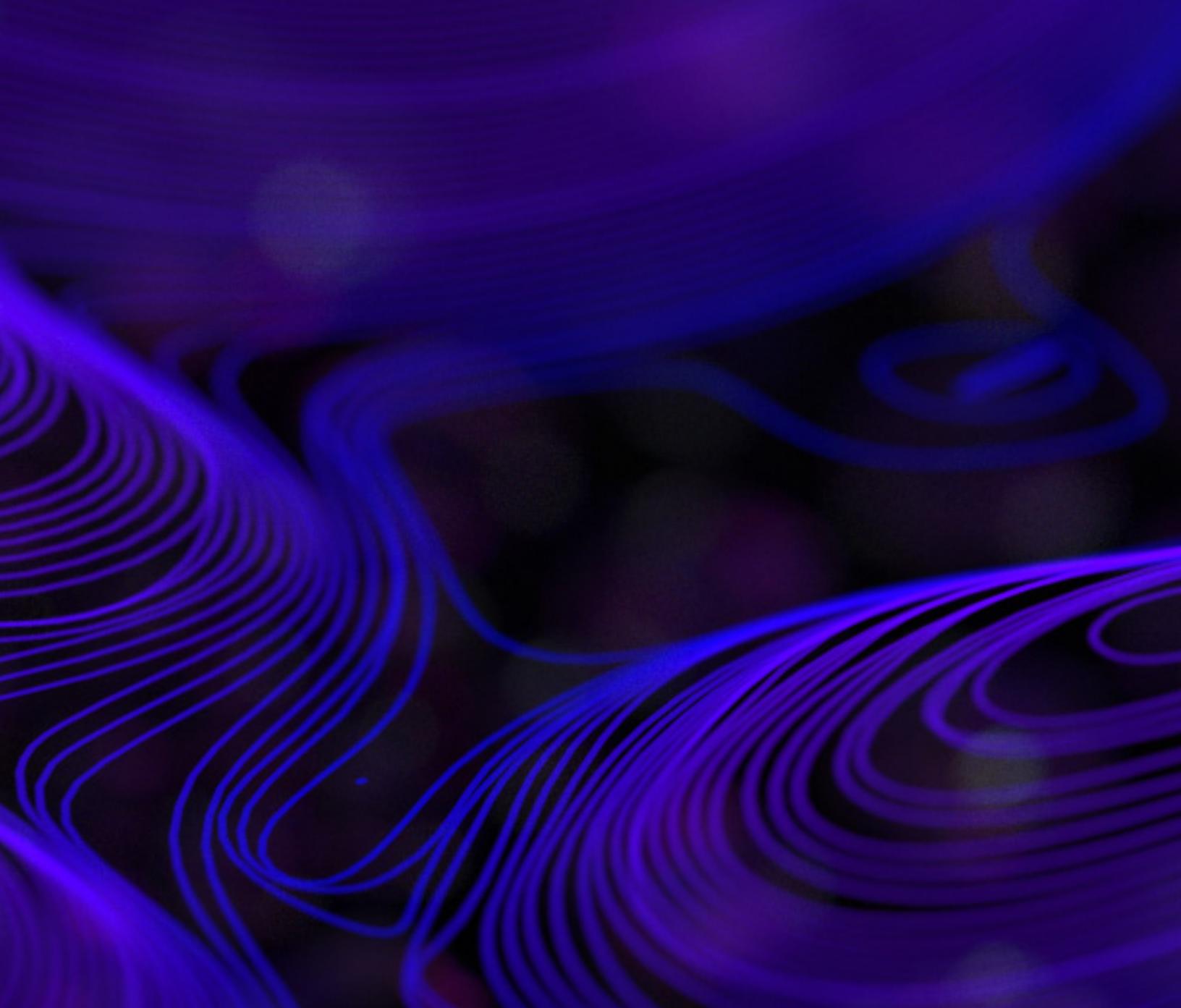
```

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

```

Fig 36



ABOUT SENTINELABS

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.