Back in Black: BlackByte Ransomware returns with its New Technology (NT) version

blog.cluster25.duskrise.com/2023/05/22/back-in-black-blackbyte-nt

Cluster25 Threat Intel Team

By Cluster25 Threat Intel Team
May 22, 2023

BlackByte is a Ransomware-as-a-Service (RaaS) group that is known for the use of the homonymous malware that is constantly updated and spread in different variants. The first implementation of the malware was written in the **C#** programming language, which was followed by a **Golang** implementation that also integrated a privilege escalation technique that exploited the **Bring Your Own Vulnerable Driver (BYOVD)** vulnerability.

The last implementation, known as **BlackByte NT**, is written in **C++** and integrates different techniques to hinder both the static and dynamic analysis of the malware, hiding the behavior of the malware when the execution is monitored. The new version also adds new drivers for the exploit of the BYOVD vulnerability in order to disable security products and tools that may interfere with its execution.

INSIGHTS

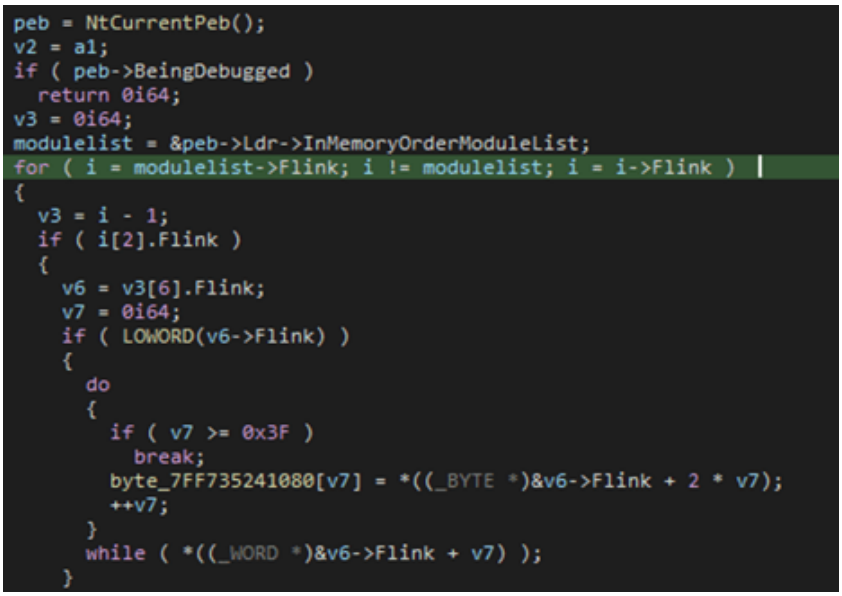
The analyzed sample is a **64-bit PE** with the compilation timestamp of **February 19th 2023**. The malware uses different anti-analysis mechanisms to hinder both the static and the dynamic analysis. One of these consists in the **dynamic import of the APIs** needed for the malware execution: the malware uses a function to find the DLL and a function to import the needed procedure with the use of the hash of their names. In both cases the hash is computed starting from a byte with the value **0x99**, which is multiplied by three and added to each byte of the string of the module name. The following is an re-implementation of the algorithm in Python:

POWERSHELL

```
def get_hash(module_name):
    module_name_b = bytearray()
    module_name_b.extend(map(ord, module_name))
    current_hash = 0x99
    for i in range(len(module_name_b)):
        current_hash = module_name_b[i] + current_hash*3
     return hex(current_hash)
```

The team used the above function in a IDAPython script that allowed to retrieve all invocations to the functions responsible for the dynamic loading of the APIs in order to continue with the static analysis of the malware.

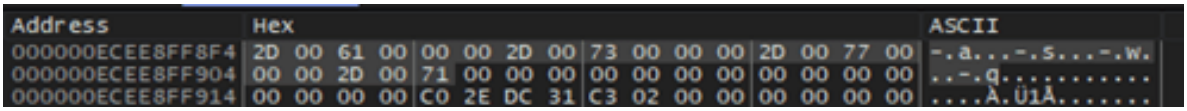
The mentioned functions receive the hash of the module or function to load as one of their arguments, then the **Process Environment Block (PEB)** structure of the binary is parsed to access to the list of the DLLs loaded in the process memory and their export function names.



```
peb = NtCurrentPeb();
v2 = a1;
if ( peb->BeingDebugged )
  return 0i64;
v3 = 0i64;
modulelist = &peb->Ldr->InMemoryOrderModuleList;
for ( i = modulelist->Flink; i != modulelist; i = i->Flink )
{
  v3 = i - 1;
  if ( i[2].Flink )
  {
    v6 = v3[6].Flink;
    v7 = 0i64;
    if ( LOWORD(v6->Flink) )
    {
      do
      {
        if ( v7 >= 0x3F )
          break;
        byte_7FF735241080[v7] = *((_BYTE *)&v6->Flink + 2 * v7);
        ++v7;
      }
      while ( *((_WORD *)&v6->Flink + v7) );
    }
```

The PEB structure is also accessed to perform an anti-debug check, verifying the content of the flag **BeingDebugged**.

After loading the needed APIs, the malware checks the arguments passed during the execution, which may have the following flags: -**a**, **-s**, **-w**, **-q**.



Address	Hex	ASCII
000000ECEE8FF8F4	2D 00 61 00 00 00 2D 00 73 00 00 00 2D 00 77 00	-.a...-.s...-.w.
000000ECEE8FF904	00 00 2D 00 71 00 00 00 00 00 00 00 00 00 00 00	..-.q...........
000000ECEE8FF914	00 00 00 00 C0 2E DC 31 C3 02 00 00 00 00 00 00	....À.Ü1Ã.......

In case no one of the mentioned arguments is specified, the malware terminates its execution.

```
if ( !mw_compare_string((const WCHAR *)arg_switch, (const WCHAR *)&v18 + 2)// -a -s -w -q
  && !mw_compare_string((const WCHAR *)arg_switch, (const WCHAR *)&v20)
  && !mw_compare_string((const WCHAR *)arg_switch, (const WCHAR *)&v22)
  && !mw_compare_string((const WCHAR *)arg_switch, (const WCHAR *)&v24) )
{
  ExitProcess_0(0i64);
}
if ( argc == 4 && mw_compare_string((const WCHAR *)&v28, (const WCHAR *)args[3]) )
```

Moreover, if the string "**svc**" is provided as an argument, the malware launches a new thread to register a new service using a random name of 7 characters which may be used to establish persistence. This, like other operations executed by the malware, is performed using **syscalls** instead of standard Windows API libraries. In this way, the malware tries to bypass detection techniques that monitor user-mode hooks by using syscalls instead of standard Windows API libraries.The same method may be used also to hinder the actions of debugger hiding tools (e.g. ScyllaHide).

```
00007FF793D2EA6D   48:8B5424 10       mov rdx,qword ptr ss:[rsp+10]
00007FF793D2EA72   4C:8B4424 18       mov r8,qword ptr ss:[rsp+18]
00007FF793D2EA77   4C:8B4C24 20       mov r9,qword ptr ss:[rsp+20]
00007FF793D2EA7C   4C:8BD1            mov r10,rcx
00007FF793D2EA7F   0F05               syscall                                  NtCreateThreadEx
00007FF793D2EA81   C3                 ret
00007FF793D2EA82   C705 74F50600 B38B21 mov dword ptr ds:[7FF793D9E000],8C218BB3  00007FF793D9E000:"2wµH"
00007FF793D2EA8C   E8 AFFFFFFF        call bb.7FF793D2EA40
00007FF793D2EA91   C705 65F50600 B73E19 mov dword ptr ds:[7FF793D9E000],34193EB7  00007FF793D9E000:"2wµH"
```

The main thread, instead, continues its execution decoding the string **dHJ0dW9pYQc=** from the Base64 encoding, the resulting string is then decrypted with a **XOR loop** using the key **BAGMVPR1p6PfdcfiV**, that is itself encrypted in the binary. The obtained value (i.e. **63389936**), is compared against the value passed with one of the execution flags when the binary is launched and, if the values don't match, the malware terminates its execution. This is done possibly with the intent of hiding the behavior of the malware from sandbox tools when the sample is not launched in the proper way.

```
if ( trtuoia_string_len > 0 )
{
  XOR_key_0 = XOR_key;
  trtuoia_string_0 = *(_QWORD *)trtuoia_string;
  XOR_key_len = XOR_key_len_0;
  do
  {
    ++trtuoia_string_0;
    v13 = index;
    v14 = index++ >> 31;
    *(_BYTE *)(trtuoia_string_0 - 1) ^= *(_BYTE *)((int)(__SPAIR64__(v14, v13) % XOR_key_len) + XOR_key_0);
  }
  while ( index < trtuoia_string_len );
}
s_arg_value = (const WCHAR *)convert_to_wchar(trtuoia_string_0_1, trtuoia_string_len);// "63389936"
if ( !mw_compare_string((const WCHAR *)second_arg_0, s_arg_value) )
  ExitProcess_0(0i64);
while ( !(unsigned int)HeapFree_0(process_heap_0, 0i64, s_arg_value) )
  ;
```

The malware then creates the file at the path **C:\SystemData\MsExchangeLog1.log** using a syscall to the function **NtCreateFile**. The file contains a sequence of elements **q<number>** and **w<number>** that seem to be used to keep track of the status of the process.

```
00007FF70E20EA50   4C:894C24 20       mov qword ptr ss:[rsp+20],r9
00007FF70E20EA55   48:83EC 28         sub rsp,28
00007FF70E20EA59   8B0D A1F50600      mov ecx,dword ptr ds:[7FF70E27E000]      00007FF70E27E000:"äæ½n"
00007FF70E20EA5F   E8 AC61FDFF        call bb.7FF70E1E4C10
00007FF70E20EA64   48:83C4 28         add rsp,28
00007FF70E20EA68   48:8B4C24 08       mov rcx,qword ptr ss:[rsp+8]
00007FF70E20EA6D   48:8B5424 10       mov rdx,qword ptr ss:[rsp+10]
00007FF70E20EA72   4C:8B4424 18       mov r8,qword ptr ss:[rsp+18]
00007FF70E20EA77   4C:8B4C24 20       mov r9,qword ptr ss:[rsp+20]
00007FF70E20EA7C   4C:8BD1            mov r10,rcx
00007FF70E20EA7F   0F05               syscall                                  NtCreateFile
00007FF70E20EA81   C3                 ret
00007FF70E20EA82   C705 74F50600 B38B21 mov dword ptr ds:[7FF70E27E000],8C218BB3  00007FF70E27E000:"äæ½n"
```

To proceed with its execution, the malware continues the dynamic resolution of functions from the following DLLs:

- **kernel32.dll**

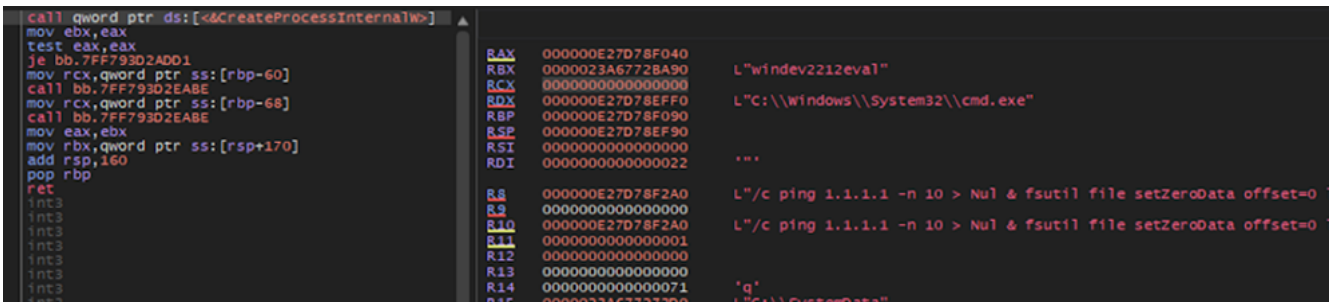
- **ntdll.dll**
- **advapi32.dll**
- **user32.dll**
- **shell32.dll**
- **rstrtmgr.dll**
- **netapi32.dll**
- **shlwapi.dll**
- **mpr.dll**
- **psapi.dll**
- **ole32.dll**
- **OleAut32.dll**
- **version.dll**
- **Winhttp.dll**
- **IPHLPAPI.dll**
- **Ws2_32.dll**
- **Dbghelp.dll**

Some of these are retrieved from the hash of their names, that are hardcoded in the binary, others using the string of their names, which are decrypted and passed to the **LoadLibraryW** function.

When all the APIs are retrieved, the malware performs an anti-debug check using two syscalls to the Native API **NtQueryInformationProcess**, using the **ProcessDebugPort** and **ProcessBasicInformation** arguments to detect if a debugger is attached to the process. If this is the case, the malware exists after deleting its executable file by launching a Windows command through the **CreateProcessInternalW** API:

WINDOWS COMMAND

C:\\Windows\\System32\\cmd.exe /c ping 1.1.1.1 -n 10 > Nul & fsutil file setZeroData offset=0 length=663424 \"<file-path>" & Del \"<file-path>" /F /Q



Other checks are performed using the **NtSystemDebugControl** and **NtGetContextThread** functions, to check debug registers and also detect *hardware breakpoints*.

If all the checks are successful, the malware scans the list of running processes and injects itself in **svchost.exe**, where the file encryption takes place. The encryption is performed in a dedicated thread for each file, using **Curve25519 Elliptic Curve Cryptography (ECC)** for asymmetric encryption and **ChaCha20** for symmetric file encryption. The files are also renamed with a Base64-encoded string and the **.blackbytent** extension (e.g. IjUSLV4COhRVDA0EbyABJg==.blackbytent).

```
if ( (int)NtReadFile(file_handle, 0i64, 0i64, 0i64, v59, buffer_read, v49, &v58, 0i64) >= 0 )
{
  v7 = 1;
  if ( v8 >= v81 )
    v7 = 3;
  (*(void (__fastcall **)(char *, __int64))encrypt[0])(v76, buffer_read);
  if ( (int)NtWriteFile(file_handle, 0i64, 0i64, 0i64, v59, buffer_read, v6, &v58, 0i64) >= 0 )
    continue;
}
```

7:24:4...	bb.exe	3100	WriteFile	C:\SystemData\MsExchangeLog1.log
7:24:4...	bb.exe	3100	WriteFile	C:\SystemData\MsExchangeLog1.log
7:24:4...	bb.exe	3100	WriteFile	C:\SystemData\MsExchangeLog1.log
7:24:4...	bb.exe	3100	WriteFile	C:\SystemData\ExsnXSD
7:24:4...	bb.exe	3100	WriteFile	C:\SystemData\ExsnXSD
7:24:4...	bb.exe	3100	Process Create	C:\Windows\System32\svchost.exe
7:24:4...	svchost.exe	820	Thread Create	
7:24:4...	bb.exe	3100	RegSetValue	HKLM\System\CurrentControlSet\Services\bam\State\UserSettings\S-1-5-21-696363100-356739881
7:24:4...	bb.exe	3100	Process Create	C:\Windows\System32\cmd.exe
7:24:4...	svchost.exe	820	Process Create	C:\Windows\System32\Conhost.exe
7:24:4...	bb.exe	3100	RegSetValue	HKLM\System\CurrentControlSet\Services\bam\State\UserSettings\S-1-5-21-696363100-356739881
7:24:4...	svchost.exe	820	Thread Create	

The following figure shows an excerpt of the Random note:

Finally, the new variant, like its previous version, continues to exploit vulnerable drivers to perform enhanced evasion techniques. In fact, the malware drops the following files in the directory **C:\SystemData**:

A3V86HEL: which is the file **RTCore64.sys**, a kernel mode driver used by Micro-Star MSI AfterBurner, a graphics card utility;

A3V86HEL_1: which is **DBUtil_2_3.Sys**, a driver related to Dell Client firmware update utility.

Both the drivers can be exploited to escalate the privileges in the target system and disable security protection products. While the use of the *RTCore64.sys*driver was already reported for previous analysis of the second version of BlackByte, the use of the Dell driver seems a peculiarity of the new version.

MITRE ATT&CK MATRIX

TACTIC	TECHNIQUE	DESCRIPTION
Initial Access	T1566.001	Phishing: Spearphishing Attachment
Initial Access	T1566.002	Phishing: Spearphishing Link
Execution	T1059.003	Command and Scripting Interpreter: Windows Command Shell
Execution	T1106	Native API
Execution	T1204.002	User Execution: Malicious File
Execution	T1569	System Services
Privilege Escalation	T1068	Exploitation for Privilege Escalation
Defense Evasion	T1140	Deobfuscate/Decode Files or Information
Defense Evasion	T1622	Debugger Evasion
Defense Evasion	T1211	Exploitation for Defense Evasion
Defense Evasion	T1562	Impair Defenses
Defense Evasion	T1070	Indicator Removal
Defense Evasion	T1055	Process Injection
Discovery	T1082	System Information Discovery
Discovery	T1083	File and Directory Discovery
Discovery	T1057	Process Discovery
Discovery	T1518	Software Discovery
Impact	T1486	Data Encrypted for Impact

INDICATORS OF COMPROMISE

CATEGORY	TYPE	VALUE
PAYLOAD	SHA256	02a0a39dbe0dcb5600f4179aeab457bb86965699e45d1d154082b02139dc701d
PAYLOAD	SHA1	`c0950ebfa3a63c705ca813cfd28364aa1d90bb09`
PAYLOAD	MD5	`bf1f2f3759448a05d3dd92a4f7f042f6`

Malware, Intelligence, Ransomware