Something about one of the Uroburos' RPC-based backdoors – Emanuele De Lucia

By edelucia

Published: 2021-11-05 · Archived: 2026-04-02 12:43:04 UTC

BigBoss is one of the RPC-based backdoors used by Uroburos (aka *Turla*, *Snake*, *Venomous Bear*, *Pacifier*). It was first spotted out in 2018 and was observed to include new features in the last quarter of 2020. During operations usually it's used in combination with R.A.T. (Remote Administration Tools) such as **Kazuar** and **Carbon**. Several months ago I had the opportunity to analyze some versions of these pieces of malware and have now decided to publish an excerpt based solely on some specific technical characteristics observed. The activity had as objective the production of detection and attribution rules one of which is shared in this post.

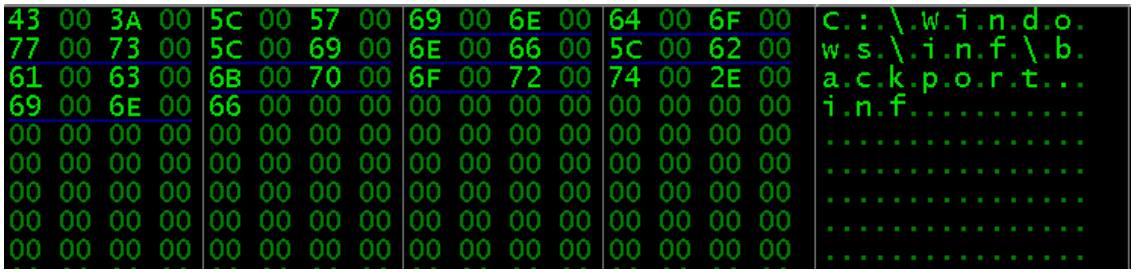
Insights

BigBoss implants exports basically (3) three functions. The Start() one is designed to retrieve basic information and to call **sub_407E50** at **0040B0D3**. First of all modulename ***kernel32.dll*** is *dexored* through the key **0x4d4e** and an handle to *kernel32.dll* is obtained through ***GetModuleHandle***. Malware writer chose to dynamically resolve certain API functions likely in order to hide information, from static analysis, about libraries and functions that are used by the implant and normally stored in IAT. In this case ***IsWow64Process*** is found through ***GetProcAddress*** to retrieve OS bitness.

```
mov     edx, 4D43h
xor     word ptr ModuleName[ecx*2], dx ; "(M&M1M-M&M/MpMqMmM'M/M/M"
inc     ecx
cmp     ecx, eax
jb      short loc_407E90

                        ; CODE XREF: sub_407E50+36↑j
push    offset ModuleName ; "(M&M1M-M&M/MpMqMmM'M/M/M"
mov     [ebp+var_4], edi
call    ds:getModuleHandleW
cmp     eax, edi
jz      short loc_407ED3
push    offset ProcName ; "IsWow64Process"
push    eax             ; hModule
call    ds:GetProcAddress
mov     esi, eax
cmp     esi, edi
jz      short loc_407ED3
lea     eax, [ebp+var_4]
push    eax
call    ds:getCurrentProcess
push    eax
call    esi
```

Shortly after a call to **sub_409C70** where the path of the ***.inf*** file is retrieved.



BigBoss writes a configuration file named ***backport.inf***. The configuration file is written to ***%SystemRoot%\INF\backport.inf*** (as reported in screenshot above) and contains a **[Version]** section with various configuration entries. At this point instructions performed call the ***StartServiceCtrlDispatcher*** function in order to connect to the SCM (***Service Control Manager***) and start the control dispatcher thread. The dispatcher thread *loops*, waiting for incoming control requests for the services specified in the dispatch table.

```
lea     ecx, [ebp+ServiceStartTable]
push    ecx             ; lpServiceStartTable
mov     [ebp+ServiceStartTable.lpServiceName], offset aSwcheckstate ; "SWCheckState"
mov     [ebp+ServiceStartTable.lpServiceProc], offset ?ServiceMain@@YAXKPAPA_W@Z ; ServiceMain(ulong,wchar_t * *)
call    ds:StartServiceCtrlDispatcherW
cmp     eax, 1
jnz     short loc_407F5C
pop     edi
xor     eax, eax
pop     esi
```

Service name is **SWCheckState**. Further API functions is then dynamically resolved. One of them is **CreateService** retrieved even in this case through a ***GetProcAddress*** call after to have obtained an handle to ***advapi32.dll*** at **sub_408790**. After the service is created OpenService function is called in order to interact with the service just created and ***ChangeServiceConfig2W*** & ***ChangeServiceConfigW*** are subsequently used to modified parameters of the same. Finally, ***StartService*** starts the service. In ServiceMain a ***RegisterServiceCtrlHandlerEx*** function is used to register a control handler with the control dispatcher. ***SetServiceStatus*** is called to set the status of the service and the ***CreateEvent*** function is then responsible to create the event object.

```
push    ebx             ; lpContext
push    offset ?ServiceCtrlHandler@@YGKKKPAX0@Z ; lpHandlerProc
push    offset ServiceName ; "SWCheckState"
mov     [ebp+ThreadId], ebx
call    ds:RegisterServiceCtrlHandlerExW
mov     hServiceStatus, eax
cmp     eax, ebx
jz      loc_40847D
mov     ecx, dword_40EE30
push    edi
mov     edi, ds:SetServiceStatus
mov     ServiceStatus.dwCheckPoint, ecx
push    offset ServiceStatus ; lpServiceStatus
inc     ecx
push    eax             ; hServiceStatus
mov     ServiceStatus.dwServiceType, 110h
mov     ServiceStatus.dwServiceSpecificExitCode, ebx
mov     ServiceStatus.dwCurrentState, 2
mov     ServiceStatus.dwWin32ExitCode, ebx
mov     ServiceStatus.dwWaitHint, 2710h
mov     ServiceStatus.dwControlsAccepted, ebx
mov     dword_40EE30, ecx
call    edi ; SetServiceStatus
push    ebx             ; lpName
push    ebx             ; bInitialState
push    ebx             ; bManualReset
push    ebx             ; lpEventAttributes
call    ds:CreateEventW
```

SMB Server is then enabled by creating the RegKey *HKEY_LOCAL_MACHINE "SYSTEM\\CurrentControlSet\\Services\\lanmanserver\\parameters* on sub_40AB90. Named pipes are used for interprocess communication (IPC) both locally and remotely. Access to the remote named pipes is done via SMB. RegKey HKLM\SYSTEM\CurrentControlSet\Control\LSA\Restrict Anonymous is then set to 0 in order to permit anonymous logon users can access all shared resources on a remote share

```
push    offset SubKey   ; "SYSTEM\\CurrentControlSet\\Services\\la"...
push    80000002h       ; hKey
mov     dword ptr [ebp+Data], 0
call    edi ; RegCreateKeyExW
test    eax, eax
jnz     short loc_40AC0C
mov     ecx, [ebp+phkResult]
push    esi
push    ebx             ; Src
push    ecx             ; hKey
call    sub_40ACB0
mov     edx, [ebp+phkResult]
add     esp, 8
push    edx             ; hKey
mov     esi, eax
call    ds:RegCloseKey
test    esi, esi
pop     esi
jnz     short loc_40AC0C
push    0               ; lpdwDisposition
lea     eax, [ebp+phkResult]
push    eax             ; phkResult
push    0               ; lpSecurityAttributes
push    2               ; samDesired
push    0               ; dwOptions
push    0               ; lpClass
push    0               ; Reserved
push    offset aSystemCurrentc_0 ; "SYSTEM\\CurrentControlSet\\Control\\LSA"
```

The
RegKey *HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\services\LanmanServer\Parameters\NullSessionPipes* is also written in order to add the following values

COMNAP

COMNODE

SQLQUERY

SPOOLSS

LLSRPC

browser

sub_40AAE0 is responsible for connections to remote devices via IPC$. via **WNetAddConnection2**

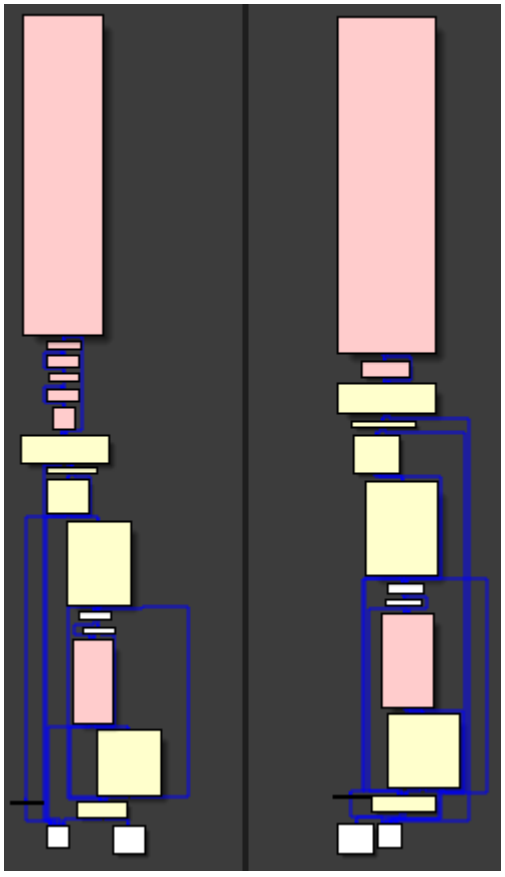
```
mov     esi, ds:WNetAddConnection2A
push    offset UserName ; lpUserName
push    offset UserName ; lpPassword
lea     edx, [ebp+NetResource]
push    edx             ; lpNetResource
mov     [ebp+NetResource.dwType], 0
mov     [ebp+NetResource.lpLocalName], 0
mov     [ebp+NetResource.lpRemoteName], ecx
mov     [ebp+NetResource.lpProvider], 0
call    esi ; WNetAddConnection2A
cmp     eax, 5
jz      short loc_40AB5B
cmp     eax, 4C3h
jnz     short loc_40AB67
```

BigBoss supports connections through *null sessions* or via default credentials. A thread is then created having **sub_408830** as ***StartAddress***. This thread is mainly responsible to handle communications with CnC (Command and Control) server. ***CreateNamedPipeW*** and ***ConnectNamedPipe*** are used to test connection. If successfull it's able to get additional payloads and write operation results into log files created and written under ***%temp%*** path.

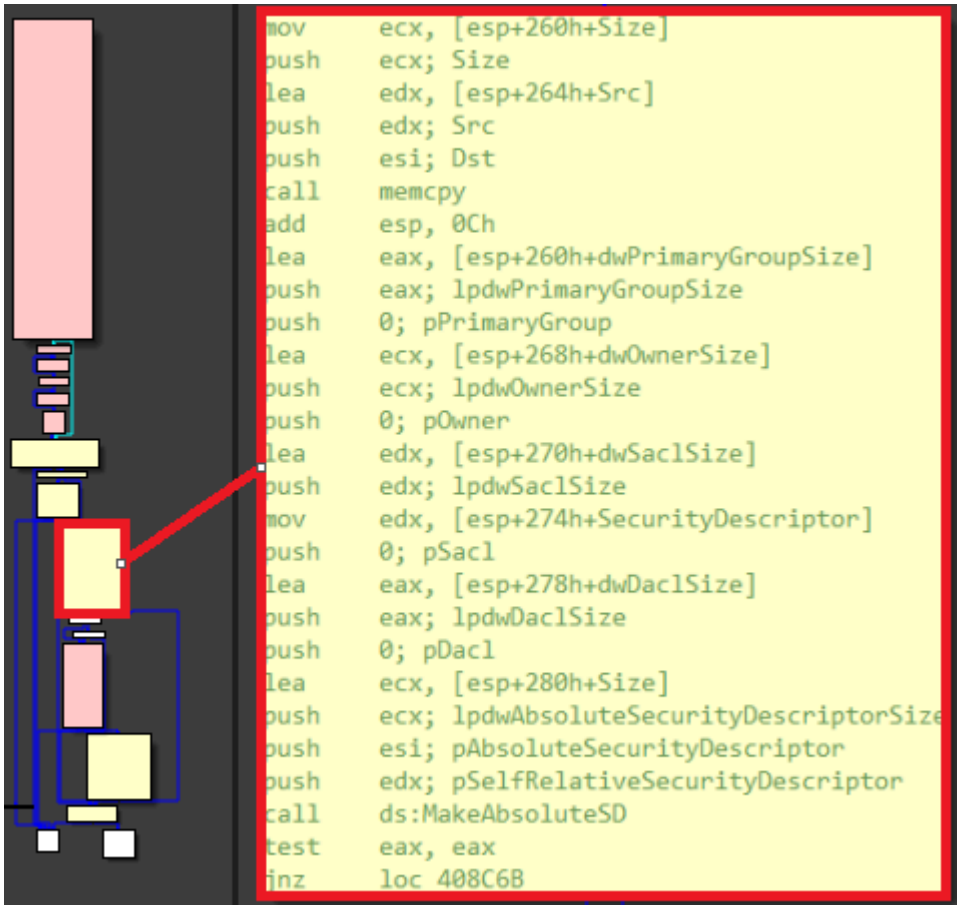
```
push    edx             ; lpBuffer
push    400h            ; nBufferLength
call    ds:GetTempPathW
lea     eax, [esp+0Ch+TempFileName]
push    eax             ; lpTempFileName
push    0               ; uUnique
push    offset PrefixString ; "sm"
lea     ecx, [esp+18h+PathName]
push    ecx             ; lpPathName
call    ds:GetTempFileNameW
mov     eax, [esp+0Ch+lpThreadParameter]
lea     edx, [esp+0Ch+TempFileName]
push    edx
call    sub_40A710
```

Conclusions

BigBoss is an integral part of the Turla team's attack and persistence suite. Its development and evolution have probably shared practices and logic with other implants linked to its main cluster such as the second stage backdoor called Carbon. For example, by analyzing both, it can be noted that it shares with it not only a partial overlap in some internal functions, as shown below



but in some cases whole code chunks having a full overlap



I based one of my hunting rules for this family on this piece of code. The rule is released in the “**Detection**” section

Indicators

Type	Value
SHA256	3b8bd0a0c6069f2d27d759340721b78fd289f92e0a13965262fea4e8907af122
SHA256	a679dbde0f4411396af54ea6ac887bd0488b2339cd8a4b509a01ca5e906f70bd

Detection

rule Turla_Code_00325_00291 {
meta:
author = “Emanuele De Lucia”
description = “Yara hunting rule for Turla shared code chunk”
hash1 = “3b8bd0a0c6069f2d27d759340721b78fd289f92e0a13965262fea4e8907af122”
hash2 = “a679dbde0f4411396af54ea6ac887bd0488b2339cd8a4b509a01ca5e906f70bd”
hash3 = “c819ec7743e2f5db13f277749961dffad08dba6dd21450eea33a27403386c959”
hash4 = “7bb65fe9421af04c5546b04a93aa0e517356c0a85856f1265587983ce2bf8aef”
hash5 = “94421ccb97b784c43d92c4b1438481eee9c907db6b13f6cfc4b86a6bb057ddcd”
strings:
$hex = { 8B (4C 24 ??|55 ??) (51|52) 8D (54 24 ??|45 ??) (52|50) 56 E8 ?? ?? ?? ?? 83 C4 ?? 8D (44 24 ??|4D ??) (50|51) 6A ?? 8D (4C 24 ??|55 ??) (51|52) 6A ?? 8D (54 24 ??|45 ??) (52|50) 8B (54 24 ??|45 ??) 6A ?? 8D (44 24 ?? | 4D ??) (50|51) 6A ?? 8D (4C 24 ??|55 ??) (51|52) 56 (52|50) FF 15 ?? ?? ?? ?? 85 C0 (0F 85 ?? ?? ?? ??|75 ??)}
condition:
$hex
}

Source: https://www.emanueledelucia.net/the-bigboss-rules-something-about-one-of-the-uroburos-rpc-based-backdoors/