

## TinyLoad Malware Analysis

Andrew Glaz

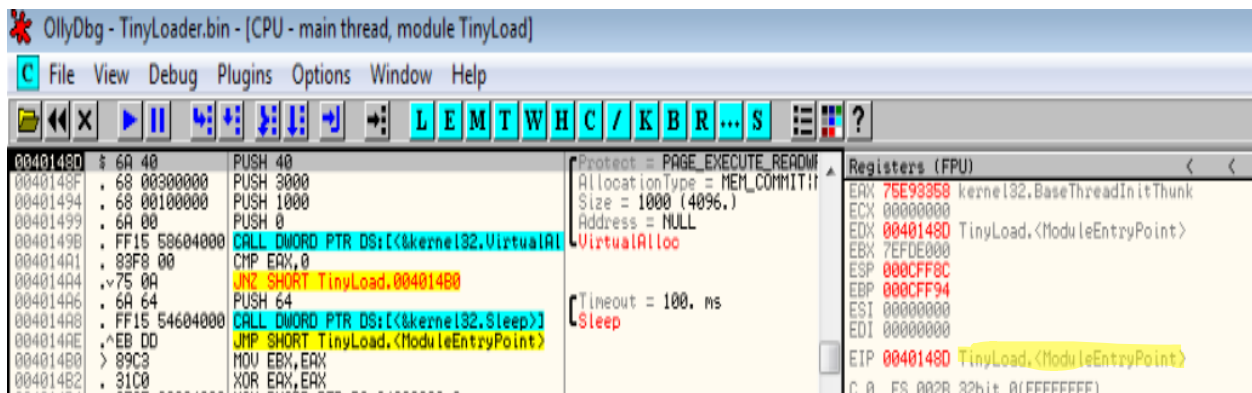
For the TinyLoader Malware, I will be using Call Stack Backtracing techniques to decrypt the malware.

This is a backdoor banking malware that delivers malware on point-of-sale and banking malware trojan. The TinyLoader malware runs on Windows for C2C. (Fidelissecurity.com) The malware uses a Kernel32.dll.

The malware spreads by “running a custom bytecode directly into the memory of the bot which makes it easier to load malware or augment malicious behavior”. (Fidelissecurity.com) The malware appears to maintain its persistence of using various commands from its C2C.

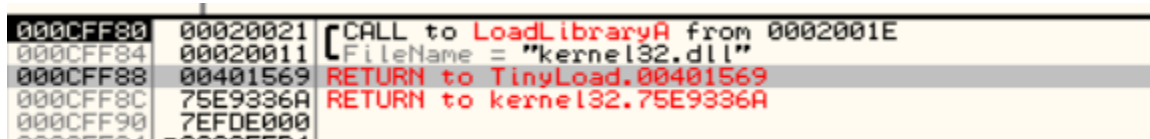
For this technique, I use OllyDBG for debugging and to find the entry point, which is the address 0040148D. I’m going to use the Call Stack Backtracing technique to identify the decrypted code.

I set breakpoints on multiple known API’s, such as VirtualAlloc, VirtualProtect, GetProcessAddress, LoadLibrary, GetModuleFileNameA and others.



```
OllyDbg - TinyLoader.bin - [CPU - main thread, module TinyLoad]
File View Debug Plugins Options Window Help
[Icons] L E M T W H C / K B R ... S [Icons] ?
0040148D 6A 40 PUSH 40
0040148F 68 00300000 PUSH 3000
00401494 68 00100000 PUSH 1000
00401499 6A 00 PUSH 0
0040149B FF15 58604000 CALL DWORD PTR DS:[<<kernel32.VirtualAlloc
004014A1 83F8 00 CMP EAX,0
004014A4 75 0A JNZ SHORT TinyLoad.00401480
004014A6 6A 64 PUSH 64
004014A8 FF15 54604000 CALL DWORD PTR DS:[<<kernel32.Sleep>
004014AE EB DD JMP SHORT TinyLoad.<ModuleEntryPoint>
004014B0 89C3 MOV EBX,EBX
004014B2 31C0 XOR EAX,EBX
[Protect = PAGE_EXECUTE_READWRITE
AllocationType = MEM_COMMIT!!
Size = 1000 (4096.)
Address = NULL
VirtualAlloc
Timeout = 100. ms
Sleep
Registers (FPU)
EAX 75E93368 kernel32.BaseThreadInitThunk
ECX 00000000
EDX 0040148D TinyLoad.<ModuleEntryPoint>
EBX 7EFDE000
ESP 000CFF8C
EBP 000CFF94
ESI 00000000
EDI 00000000
EIP 0040148D TinyLoad.<ModuleEntryPoint>
C 0 ES 002B 32bit 0(FFFFFFFF)
```

Now, I will run the malware until I get to the LoadLibraryA API.



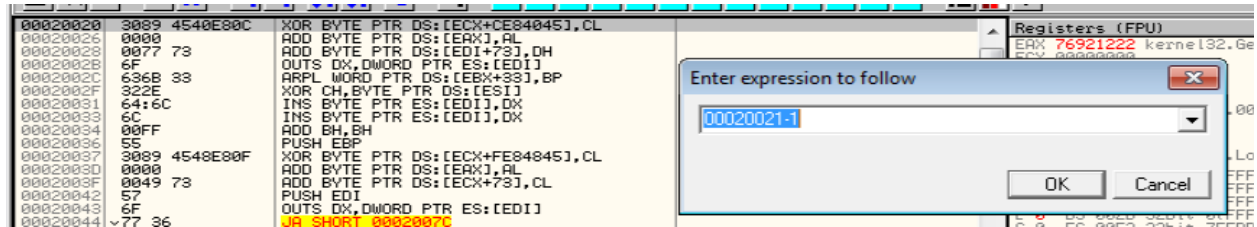
```
000CFF80 00020021 [CALL to LoadLibraryA from 0002001E
000CFF84 00020011 FileName = "kernel32.dll"
000CFF88 00401569 RETURN to TinyLoad.00401569
000CFF8C 75E9336A RETURN to kernel32.75E9336A
000CFF90 7EFDE000
000CFF94
```

Now, I have a return address x00401569. Let’s check it out. This address points out after the entry function. I removed the analyses from the module

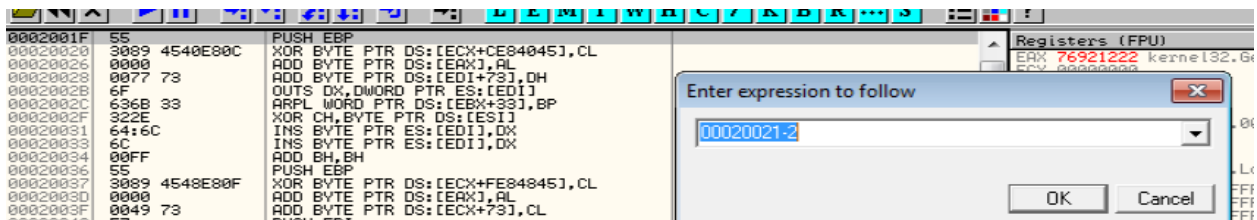


```
00401567 FFD3 CALL EBX
00401569 0000 ADD BYTE PTR DS:[EAX],AL
```

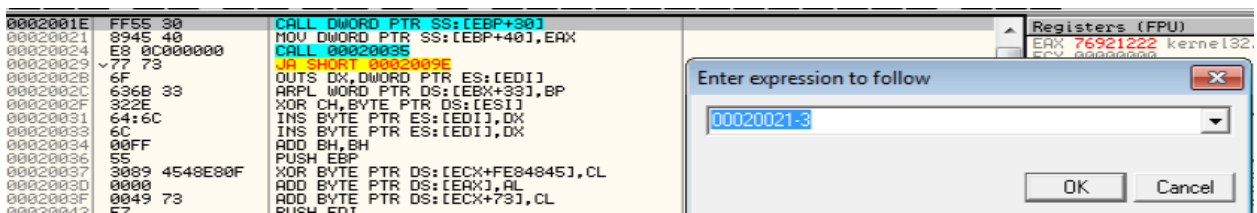
This is first the first address shift for 1 byte



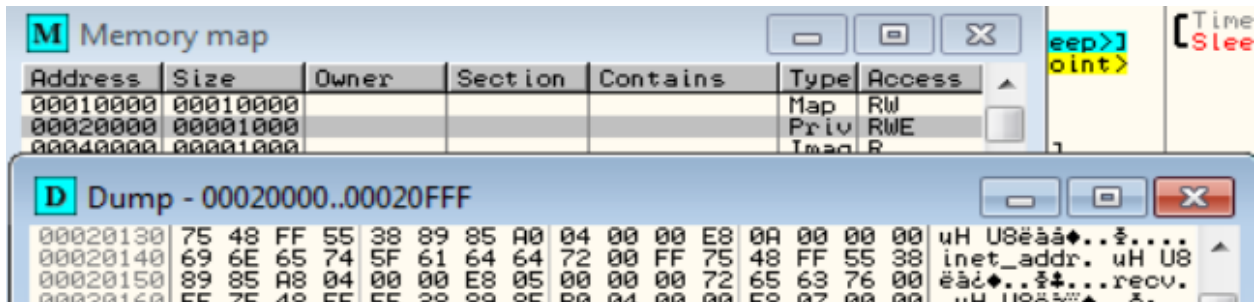
The second shift, 2 bytes



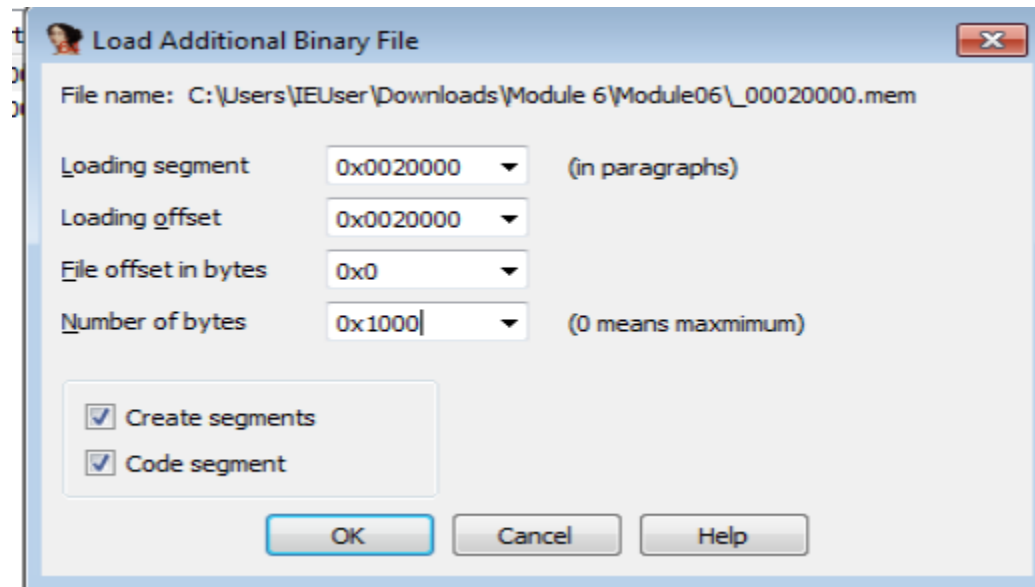
Third shift, 3 bytes



After this, I dump the memory by including the decrypted code from the memory map.



Now, I'm going to set the dumped memory file offset, \_00020000.mem in the correct location in IDA Pro. I am using 1 byte here.



With this file loaded, I can now see the loaded file view in the segments.

Function name	Name	Start	End	R	W	X	D	L	Align	Base	Type	Class	AD	es
sub_1BF	seg000	0000000000000000	0000000000001000	?	?	?	.	L	byte	0001	public	CODE	64	000
sub_2E3	seg001	0000000000002000	00000000000021000	?	?	?	.	L	byte	0003	public	CODE	32	000

Now, I can go into the hex view, and I can see some separate strings, including the kernel32.dll

```

00 6B 65 72 6E 65 6C 33 32 2E 64 6C 6C 00 FF 55 .kernel32.dll..U
30 89 45 40 E8 0C 00 00 00 77 73 6F 63 6B 33 32 0.E@.....wsock32
2E 64 6C 6C 00 FF 55 30 89 45 48 E8 0F 00 00 00 .dll..U0.EH....
49 73 57 6F 77 36 34 50 72 6F 63 65 73 73 00 FF IsWow64Process..
75 40 FF 55 38 89 85 B0 02 00 00 E8 0D 00 00 00 u@.U8.....
56 69 72 74 75 61 6C 41 6C 6C 6F 63 00 FF 75 40 VirtualAlloc..u@
FF 55 38 89 85 B8 02 00 00 E8 09 00 00 00 6C 73 .U8.....ls
74 72 63 70 79 41 00 FF 75 40 FF 55 38 89 85 C8 trcpyA..u@.U8...
02 00 00 E8 12 00 00 00 47 65 74 43 75 72 72 65 .....GetCurrent

```

GetCurrentProcess is here, and then there is a sleep function called. Would like to know why this is?

```

02 00 00 E8 12 00 00 00 47 65 74 43 75 72 72 65 .....GetCurrentProcess..u@.U8
6E 74 50 72 6F 63 65 73 73 00 FF 75 40 FF 55 38 .....Sleep
89 85 C0 02 00 00 E8 06 00 00 00 53 6C 65 65 70

```

Then in the next data of the dump, this appears to be where the C2C is located, and possibly an IP address, with the inet\_addr API.

```

89 85 C0 02 00 00 E8 06 00 00 00 53 6C 65 65 70 .....Sleep
00 FF 75 40 FF 55 38 89 85 70 02 00 00 E8 08 00 ..u@.U8..p.....
00 00 57 53 41 53 74 61 72 74 75 70 00 FF 75 48 ..WSAStartup..uH
FF 55 38 89 85 80 04 00 00 E8 0D 00 00 00 5F 5F .U8.....__
57 53 41 46 44 49 73 53 65 74 00 FF 75 48 FF 55 WSAFDIsSet..uH.U
38 89 85 88 04 00 00 E8 0C 00 00 00 63 6C 6F 73 8.....clos
65 73 6F 63 6B 65 74 00 FF 75 48 FF 55 38 89 85 esocket..uH.U8..
90 04 00 00 E8 06 00 00 00 68 74 6F 6E 73 00 FF .....htons..
75 48 FF 55 38 89 85 A0 04 00 00 E8 0A 00 00 00 uH.U8.....
69 6E 65 74 5F 61 64 64 72 00 FF 75 48 FF 55 38 inet_addr..uH.U8
89 85 A8 04 00 00 E8 05 00 00 00 72 65 63 76 00 .....recv.
FF 75 48 FF 55 38 89 85 B0 04 00 00 E8 07 00 00 .uH.U8.....
00 73 65 6C 65 63 74 00 FF 75 48 FF 55 38 89 85 .select..uH.U8..
B8 04 00 00 E8 07 00 00 00 73 6F 63 6B 65 74 00 .....socket.
FF 75 48 FF 55 38 89 85 C8 04 00 00 E8 05 00 00 .uH.U8.....
00 73 65 6E 64 00 FF 75 48 FF 55 38 89 85 C0 04 .send..uH.U8....
00 00 E8 08 00 00 00 63 6F 6E 6E 65 63 74 00 FF .....connect..

```

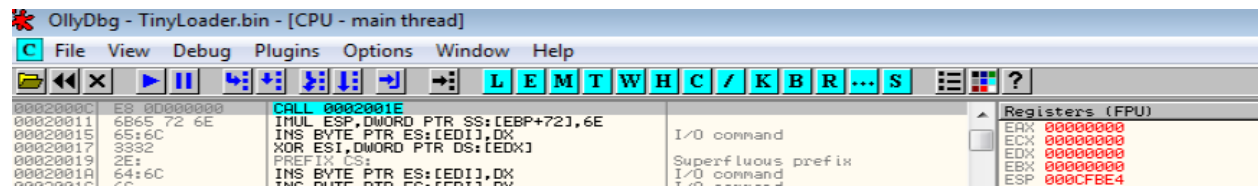
You can also begin to see where the strings are stored in IDA Pro view. Still would like to know what the hotkey is to make the view better.

```

seg001:00020060      db  56h ; V
seg001:00020061      db  69h ; i
seg001:00020062      db  72h ; r
seg001:00020063      db  74h ; t
seg001:00020064      db  75h ; u
seg001:00020065      db  61h ; a
seg001:00020066      db  6Ch ; l
seg001:00020067      db  41h ; A
seg001:00020068      db  6Ch ; l
seg001:00020069      db  6Ch ; l
seg001:0002006A      db  6Fh ; o
seg001:0002006B      db  63h ; c

```

Finally, I go back to OllyDBG. I go and set a breakpoint on the outside of the entry point, 401569, and I rerun the application to get to 20021E and the call of LoadLibraryA. At this point, I got a little lost in the OllyDBG, but was able to find the entrypoint of the decryption code at 2000C



Thanks again for reading my analysis on the TinyLoaderMalware.

References:

<https://fidelissecurity.com/threatgeek/threat-intelligence/deconstructing-tinyloader/>

<https://Maltrak.com>