

# TADAQUEOUS moments

---

laanwj.github.io/2016/09/01/tadaqueos.html



## Laanwj's blog

---

Randomness

### Blog About

The one mystery module in the BLATSTING rootkit/malware/implant/... in the Equation Group dump is `m12000000`, or TADAQUEOUS. There is only one mention of it in the various documentation and scripts:

If you are putting up tadaqueous, there will be lp error due to a missing files, there is no LP for this module.

What is meant here is that there is no Listening Post, or LP module for it. "Listening Post" is what the Equation Group calls its command-and-control programs. It can only be loaded and unloaded through this interface, not controlled, and it will spit an error message. Well, that tells us nothing.

At first sight the module looks sort of boring. It packages a kernel module and a user-space executable, but looking at the imported symbols and (open) strings, what it does is something with Linux processes and system calls.

However, after delving a bit deeper, I stumbled on a function that hooks a whole series of kernel calls, whose names are obfuscated in the binary:

```

F (fcn) hook_kernel_functions 153
|           ; CALL XREF from 0x08001673 (fcn.080015a8)
|           0x08000f54      57           push edi           ; 0 args -
hooks up to 14 kernel functions
|           0x08000f55      56           push esi
|           0x08000f56      53           push ebx
|           0x08000f57      83ec10      sub esp, 0x10
|           0x08000f5a      31c0        xor eax, eax
|           0x08000f5c      c744240c0000. mov dword [esp + 0xc], 0 ; [ra - 0x10]
|           0x08000f64      89c7        mov edi, eax
|           0x08000f66      31f6        xor esi, esi
|           ↗ 0x08000f68      80bed4030000. cmp byte [esi + 0x3d4], 0 ; RELOC 32
.data
|           ↖ 0x08000f6f      7517        jne 0x8000f88     ; hook this
function?
|           ↗ 0x08000f71      47           inc edi           ; advance
forward
|           ||| 0x08000f72      83c618      add esi, 0x18     ; records are
0x18 bytes
|           ||| 0x08000f75      83ff0e      cmp edi, 0xe      ; count to 14
|           ||| ↖ 0x08000f78      76ee        jbe 0x8000f68     ; the end?
|           ||| 0x08000f7a      89c2        mov edx, eax
|           ||| ↗ 0x08000f7c      83c410      add esp, 0x10
|           ||| ||| 0x08000f7f      5b          pop ebx
|           ||| ||| 0x08000f80      5e          pop esi
|           ||| ||| 0x08000f81      89d0        mov eax, edx
|           ||| ||| 0x08000f83      5f          pop edi
|           ||| ||| 0x08000f84      c3          ret
|           ||| ||| 0x08000f85      8d7600      lea esi, [esi]
|           ||| ↗ 0x08000f88      c744240c0000. mov dword [esp + 0xc], 0 ; [ra - 0x10]
|           ||| ||| 0x08000f90      51          push ecx
|           ||| ||| 0x08000f91      6a05        push 5
|           ||| ||| 0x08000f93      ffb6c8030000 push dword [esi + 0x3c8] ; RELOC 32 .data
; kernel function to hook
|           ||| ||| 0x08000f99      8d442418    lea eax, [esp + 0x18] ; [ra - 0x10]
|           ||| ||| 0x08000f9d      50          push eax           ; outptr
|           ||| ||| 0x08000f9e      a100000000 mov eax, dword [0] ; RELOC 32
the_interface
|           ||| ||| 0x08000fa3      ff5054      call dword [eax + 0x54] ; call core.54
is kernel function hookable?
|           ||| ||| 0x08000fa6      83c410      add esp, 0x10
|           ||| ||| 0x08000fa9      85c0        test eax, eax
|           ||| ||| 0x08000fab      8d9ec0030000 lea ebx, [esi + 0x3c0] ; RELOC 32 .data
|           ||| ||| 0x08000fb1      bafeffffff mov edx, 0xffffffff
|           ||| ↖ 0x08000fb6      74c4        je 0x8000f7c
|           ||| ||| 0x08000fb8      8b54240c    mov edx, dword [esp + 0xc] ; [ra - 0x10]
|           ||| ||| 0x08000fbc      85d2        test edx, edx
|           ||| ↖ 0x08000fbe      7526        jne 0x8000fe6     ; FAIL
|           ||| ||| 0x08000fc0      83ec0c      sub esp, 0xc
|           ||| ||| 0x08000fc3      6a00        push 0
|           ||| ||| 0x08000fc5      50          push eax           ; return value
from core.54
|           ||| ||| 0x08000fc6      ff730c      push dword [ebx + 0xc] ; local
function to redirect to
|           ||| ||| 0x08000fc9      ff7308      push dword [ebx + 8] ; kernel

```

```

function to hook
|      ||||  0x08000fcc      8d4304      lea eax, [ebx + 4]
|      ||||  0x08000fcf      50          push eax                      ; outptr
|      ||||  0x08000fd0      a100000000  mov eax, dword [0] ; RELOC 32
the_interface
|      ||||  0x08000fd5      ff5058      call dword [eax + 0x58] ; call
core.58: hook kernel function
|      ||||  0x08000fd8      83c420      add esp, 0x20
|      ||||  0x08000fdb      85c0        test eax, eax
|      ||||  0x08000fdd      bfffffff    mov edx, 0xffffffff
|      |<  0x08000fe2      748d        je 0x8000f71
|      |<  0x08000fe4      eb96        jmp 0x8000f7c
|      |>  0x08000fe6      bfffffff    mov edx, 0xffffffff
|      |<  0x08000feb      eb8f        jmp 0x8000f7c

```

Summarizing the data structure at `.data+0x3c0` :

Offset	Flag	Target symbol	Redirected to
0x000003c0	0x0001	<code>__add_ipsec_sa</code>	<code>.text+0x00000c60</code>
0x000003d8	0x0002	<code>asic_init_cmd_block</code>	<code>.text+0x00000e8c</code>
0x000003f0	0x0004	<code>__del_ipsec_sa</code>	<code>.text+0x00000da0</code>
0x00000408	0x0008	<code>get_random_bytes</code>	0x00000000
0x00000420	0x0010	<code>cipher_des</code>	0x00000000
0x00000438	0x0020	<code>cipher_3des</code>	0x00000000
0x00000450	0x0040	<code>cipher_aes</code>	0x00000000
0x00000468	0x0080	<code>cipher_null</code>	0x00000000
0x00000480	0x0100	<code>hmac_null</code>	0x00000000
0x00000498	0x0200	<code>hmac_md5_96</code>	0x00000000
0x000004b0	0x0400	<code>hmac_sha1_96</code>	0x00000000
0x000004c8	0x0800	<code>cipher_dev_in_use</code>	0x00000000
0x000004e0	0x1000	<code>asic_xxcrypt</code>	<code>.text+0x00000f18</code>
0x000004f8	0x2000	<code>cpx_read_rand</code>	<code>.text+0x00000e50</code>

It looks like this is a noteworthy module after all:

- Most of the symbols are not standard Linux symbols but specific to the TOS/Fortinet implementation. Their meaning, however is clear from the name.

- Some of the functions are redirected to a local function, others to 0x00000000, which likely means that they are disabled completely.

It does give a huge hint at what the goal of this module is: cripple or disable IPsec! It appears it can be used to selectively disable ciphers, HMAC algorithms, and random number generation. It is obvious how this is useful to anyone trying to either intercept or insert themselves into a target's VPN network.

Shunting the function `get_random_bytes` will have the effect of disabling *all* random number generation in the kernel. Not just for IPsec, but also for e.g. TCP sequence numbers, enabling IP spoofing. It is not used for `/dev/[u]random` however, so user space processes cannot easily detect this.

[nohats.ca](#) writes, in the conclusion of an article about IPsec and the Snowden revelations:

I read this to mean that the hardware or software of the system running IPsec was compromised, causing it to send valid protocol ESP packets, but creating those in such a way that these could be decrypted without knowing the ESP session keys (from IKE). Possibly by subverting the hardware number generator, or functions related to IV / ICV's / nonces that would appear to be random but were not.

We've found out one of the ways how. This targets a specific series of routers, but I'd be surprised if it was the only one, and other instances may be similar to this implementation, or based on it: there are various hints that BLATSTING is the oldest generation of implants in the EQGRP dump.

Written on September 1, 2016

Tags: [eggrp](#) [malware](#)

Filed under [Reverse-engineering](#)