MMD-0026-2014 - Linux/AES.DDoS: Router Malware Warning | Reversing an ARM arch ELF

Saturday, September 13, 2014  
[Link to Source](https://blog.malwaremustdie.org/2014/09/reversing-arm-architecture-elf-elknot.html)

**Sticky notes, the latest infection report is below tweet:**

[#ELF](https://twitter.com/hashtag/ELF?src=hash) AES.DDoS (<http://t.co/y0D7YwPaD4>) [#malware](https://twitter.com/hashtag/malware?src=hash) hits MIPS/ARM routers. Some AV had bad sigs. <http://t.co/TMdxaEDiNh> [pic.twitter.com/5eaSKJmMZU](http://t.co/5eaSKJmMZU)

— ☩MalwareMustDie, NPO (@MalwareMustDie) [July 2, 2015](https://twitter.com/MalwareMustDie/status/616752061779030016)

The background

It is one of our active project to monitor the China origin ELF DDoS'er malware threat. The growth is very rapid nowadays, MMD detected 5 variants is active under almost 15 panels scattered in China network. I am quite active in supporting the team members of this project, so recently almost everyday I reverse ELF files between 5-10 binaries. They are not aiming servers with x32 or x64 architecture but the router devices that runs on Linux too. In some cases I found the FreeBSD variant.

In this story I faced an ARM architecture binary, which I found it interesting so I decided to share it here. The reason is because, practically: it was designed to work in ARM router with minimizing a well-known Linux malware with DDoS functions/features, that I also previously posted some in here --->[[-1-](http://blog.malwaremustdie.org/2013/12/lets-be-more-serious-about-dns-amp-elf.html)] [[-2-](http://blog.malwaremustdie.org/2014/05/linux-reversing-is-fun-toying-with-elf.html)] [[-3-](http://pastebin.com/dhvAEH4D)] [[-4-](http://www.kernelmode.info/forum/viewtopic.php?f=16&t=3099#p23858)] [[-5-](http://www.kernelmode.info/forum/viewtopic.php?f=16&t=3099#p23859)] [[-6-](http://www.kernelmode.info/forum/viewtopic.php?f=16&t=3099#p23862)], to specifically infect ARM (router) devices, and this binary is trying to convince that it is a WindowsHelp binary :D , and ,specifically: from my reverse engineering point of view, ARM & "thumb" assembly are interesting.

Why I know it is aiming for router is because, the way to use internet to connect directly to remote global IP, the method used to grab data using specific location in the embedded device, and the trace of sources used during the compilation of the malware itself.

The malware

As usual, China actor(s) serves their malware binary under "specific panel", and this binary is spotted among with other Linux/Elknots malware. So as you can see it was served from Sept 10th and is having 4 downloads (including me, one time)  
[](https://lh5.googleusercontent.com/-8-vdgxkS7r4/VBJUMWzZobI/AAAAAAAAQ3o/PSu0CRfbdSg/s941/arm001.png)  
The file looks like this:

[?](https://blog.malwaremustdie.org/2014/09/reversing-arm-architecture-elf-elknot.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | $ ls -alF 1  -rwxr--r--  1 mmd  mmd  165,176 Sep 10 10:21 1  $ md5 1  MD5 (1) = 0bb68bd65d94f61b7b20117b99d8526c  $ file 1  1: ELF 32-bit LSB executable, ARM, version 1 (GNU/Linux), statically linked, stripped |

Well, we know is an ARM binary, but I need more information, so I check the ELF composition:

[?](https://blog.malwaremustdie.org/2014/09/reversing-arm-architecture-elf-elknot.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | Entry point address:  "0x2f118"    Start of program headers: "52" (bytes into file)    Start of section headers: "0" (bytes into file)    Flags: "0x4000002", has entry point, "Version4 EABI"    Size of this header: "52" (bytes)    Size of program headers: "32" (bytes)    Number of program headers: "2"    Size of section headers: "40" (bytes)    Number of section headers: "0"    Section header string table index: "0"    Program Headers:    Type     Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align    LOAD     "0x000000 0x00008000 0x00008000 0x282b1 0x282b1 R E 0x8000"    LOAD     "0x000c24 0x000d0c24 0x000d0c24 0x00000 0x00000 RW  0x8000" |

Now it's time for calculating the data, we know the size and we see the each LOAD headers size which is just unfit, further, I don't see any section (either dynamic or static) nor relocation data that I can expect from an ARM ELF (they should have more symbols), which is strange. This a sign of protection, someone want to hide something, in the end that person is hiding EVERYTHING which ending up to be very suspicious :-) - So the binary could be packed or encrypted protection, we have many possibility.

Further details of this family of ELF malware we posted regularly in here:-->[[link](http://www.kernelmode.info/forum/viewtopic.php?f=16&t=3483)]

Packer

Let's check, I went to the EP point (0x2f118) and start to do the stuff I usually do, with noted..we have to be very patient with the ARM or THUMB assembly since they have larger steps for simple operation than Intel processor.

[?](https://blog.malwaremustdie.org/2014/09/reversing-arm-architecture-elf-elknot.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | ;-- entry0:  0x0002f118  adr     r12, off\_2f104  0x0002f11c  ldmia   r12, {r1,r2,r10,r11,lr}  0x0002f120  add     r10, r10, r12  0x0002f124  add     r11, r11, r12  0x0002f128  mov     r0, r2  0x0002f12c  sub     r9, r12, r1  0x0002f130  add     r1, r1, #0x1000  0x0002f134  mov     r3, #0  0x0002f138  stmfd   sp!, {r0-r3,lr}  0x0002f13c  mov     r2, #7  0x0002f140  ldr     r3, [r12,#0x10]  0x0002f144  mov     r5, #0  0x0002f148  mov     r4, 0xffffffff  0x0002f14c  orr     r3, r3, #0x10  0x0002f150  mov     r7, #0xc0  0x0002f154  svc     0  0x0002f158  cmn     r0, #0x1000  0x0002f15c  bcs     loc\_0x02fbd0    [...] |

..following the registers value and in **r1** we will find the value that can tell what is happening:

[?](https://blog.malwaremustdie.org/2014/09/reversing-arm-architecture-elf-elknot.html)

|  |  |
| --- | --- |
| 1  2  3  4  5 | 0x0002fbd0  mov  r2, #0x1e  0x0002fbd4  adr  r1, aprot\_execprot\_ ; ""PROT\_EXEC|PROT\_WRITE failed.\n""  0x0002fbd8  mov  r0, #2  0x0002fbdc  mov  r7, #4  0x0002fbe0  svc  0 |

This value may ring your bells too :). ok this ELF is protected, with/for what? I look from its DCB data from where it was called and clarifying the answer:

[?](https://blog.malwaremustdie.org/2014/09/reversing-arm-architecture-elf-elknot.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | 0x0002FBF0 aProt\_execProt\_ DCB "PROT\_EXEC|PROT\_WRITE failed.",0xA,0  0x0002FC0E                 DCB 0xA,0  0x0002FC10 aInfoThisFileIs DCB 0x24,"Info: This file is packed with the UPX executable packer http:/"  0x0002FC10                 DCB "/upx.sf.net ",0x24,0xA,0  0x0002FC5F aIdUpx3\_91Copyr DCB 0x24,"Id: UPX 3.91 Copyright (C) 1996-2013 the UPX Team. All Rights R"  0x0002FC5F                 DCB "eserved. ",0x24,0xA,0  0x0002FCAB                 DCB 0x0 ;; here goes the table..  0x0002FCAC                 DCD 0x9A8, 0x5F9, 0x500E, 0x6C00031A, 0x942C5302, 0x18D063CB  0x0002FCAC                 DCD 0x49382EE, 0xD185E779, 0x57399E2E, 0xD24C892F, 0x1003EA02  0x0002FCAC                 DCD 0x6A5A70C9, 0x2F701D6A, 0x6D0D9A7, 0xD2EC6754, 0x95ECE49  [...]                      [...] |

Oh, silly me.. it is a UPX, but, is it common and not modded one? So I went back to check the hex snapshot, to confirm..

[?](https://blog.malwaremustdie.org/2014/09/reversing-arm-architecture-elf-elknot.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | 00000000  7f 45 4c 46 01 01 01 03  00 00 00 00 00 00 00 00  |.ELF............|  00000010  02 00 28 00 01 00 00 00  18 f1 02 00 34 00 00 00  |..(.........4...|  00000020  00 00 00 00 02 00 00 04  34 00 20 00 02 00 28 00  |........4.(.|  00000030  00 00 00 00 01 00 00 00  00 00 00 00 00 80 00 00  |................|  00000040  00 80 00 00 b1 82 02 00  b1 82 02 00 05 00 00 00  |................|  00000050  00 80 00 00 01 00 00 00  24 0c 00 00 24 0c 0d 00  |........$...$...|  00000060  24 0c 0d 00 00 00 00 00  00 00 00 00 06 00 00 00  |$...............|  00000070  00 80 00 00 93 cc 51 fc  55 50 58 21 b4 11 0d 17  |......Q.UPX!....|  00000080  00 00 00 00 58 64 08 00  58 64 08 00 d4 00 00 00  |....Xd..Xd......| |

Since I know that some malware actors is really (enjoying to ) watch this blog too (smile). I don't want to be specific on this, but from reading the hex above we can recognize the originality of this UPX, which it is. Otherwise you have patch it to depack, sample a way to depack the custom UPX is in here-->[[LINK](https://www.virustotal.com/en/file/6dd946e821df59705dcfeb79fab810336d0ee497fd715fb5b6711e05c0428f4d/analysis/)]. Further.. as this is the common UPX, and the "U" stands for universal & we can do "universal" solution too to unpack :)

[?](https://blog.malwaremustdie.org/2014/09/reversing-arm-architecture-elf-elknot.html)

|  |  |
| --- | --- |
| 1 | 549976 <-    165176   30.03%   linux/armel   unpacked.1 |

So now we have the bigger file size :D  
This time let's check the composition again:

[?](https://blog.malwaremustdie.org/2014/09/reversing-arm-architecture-elf-elknot.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | 00000000  7f 45 4c 46 01 01 01 00  00 00 00 00 00 00 00 00  |.ELF............|  00000010  02 00 28 00 01 00 00 00  10 81 00 00 34 00 00 00  |..(.........4...|  00000020  f8 5f 08 00 02 00 00 04  34 00 20 00 05 00 28 00  |.\_......4.(.|  00000030  1c 00 1b 00 01 00 00 70  5c bb 07 00 5c 3b 08 00  |.......p\...\;..|  00000040  5c 3b 08 00 00 09 00 00  00 09 00 00 04 00 00 00  |\;..............|  00000050  04 00 00 00 01 00 00 00  00 00 00 00 00 80 00 00  |................|  00000060  00 80 00 00 dc c4 07 00  dc c4 07 00 05 00 00 00  |................|  00000070  00 80 00 00 01 00 00 00  dc c4 07 00 dc c4 08 00  |................|  00000080  dc c4 08 00 4c 0a 00 00  48 47 04 00 06 00 00 00  |....L...HG......|  00000090  00 80 00 00 04 00 00 00  d4 00 00 00 d4 80 00 00  |................|  000000a0  d4 80 00 00 20 00 00 00  20 00 00 00 04 00 00 00  |........|  000000b0  04 00 00 00 07 00 00 00  dc c4 07 00 dc c4 08 00  |................|  000000c0  dc c4 08 00 14 00 00 00  30 00 00 00 04 00 00 00  |........0.......|  000000d0  04 00 00 00 04 00 00 00  10 00 00 00 01 00 00 00  |................|  000000e0  47 4e 55 00 00 00 00 00  02 00 00 00 06 00 00 00  |GNU.............| |

Yeah, the "GNU" ascii appears now. And, see more details below:

[?](https://blog.malwaremustdie.org/2014/09/reversing-arm-architecture-elf-elknot.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | Entry point address: "0x8110"  Start of program headers:  "52" (bytes into file)  Start of section headers:  "548,856" (bytes into file)  Flags:   "0x4000002", has entry point, "Version4 EABI"  Size of this header: "52" (bytes)  Size of program headers:   "32" (bytes)  Number of program headers:   "5"  Size of section headers:   "40" (bytes)  Number of section headers:   "28"  Section header string table index: "27" |

Good! the true EP is shown now. And we have the new program headers too:

[?](https://blog.malwaremustdie.org/2014/09/reversing-arm-architecture-elf-elknot.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | Type     Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align  EXIDX    "0x07bb5c 0x00083b5c 0x00083b5c 0x00900 0x00900" R   0x4  LOAD     "0x000000 0x00008000 0x00008000 0x7c4dc 0x7c4dc" R E 0x8000  LOAD     "0x07c4dc 0x0008c4dc 0x0008c4dc 0x00a4c 0x44748" RW  0x8000  NOTE     "0x0000d4 0x000080d4 0x000080d4 0x00020 0x00020" R   0x4  TLS      "0x07c4dc 0x0008c4dc 0x0008c4dc 0x00014 0x00030" R   0x4 |

A quick calculation of the size above shows that at least we have accuracy to more than 80% to the actual size now, good enough. It showed we have unprotected/unpacked data and so I can expect good material to disassembly it, but firstly, let's dump the sections to be sure that we have no more encryption/protection:

[?](https://blog.malwaremustdie.org/2014/09/reversing-arm-architecture-elf-elknot.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | .note.ABI-tag  .init  .fini  .init\_array  .fini\_array  \_\_libc\_freeres\_fn  \_\_libc\_thread\_freeres\_fn  \_\_libc\_freeres\_ptrs  \_\_libc\_subfreeres  \_\_libc\_atexit  \_\_libc\_thread\_subfreeres  .text  .rodata  .ARM.extab  .ARM.exidx  .ARM.attributes  .eh\_frame  .jcr  .data.rel.ro  .got  .data  .bss  .note.ABI-tag  .tdata  .tbss |

After comparing some ELF reference for ARM to make sure we have the proper heades, I found that all headers are there!)) Good. Since I happened to reverse a lot of same malware I can guess the sections that contains the good data, these are the section I picked to start analysis:

[?](https://blog.malwaremustdie.org/2014/09/reversing-arm-architecture-elf-elknot.html)

|  |  |
| --- | --- |
| 1  2  3  4  5 | Name               Addr     Off    Size  -------------------------------------------  .text              0x08110 0x00110 0x6605c  .rodata            0x6f008 0x67008 0x149a0  \_\_libc\_freeres\_fn  0x6e16c 0x6616c 0x00df4 |

.text is a must in ELF since all of code logic goes here, and .rodata mostly contains the database of symbols (and sometimes .data too..depends on the coder). You can go to other section like .bss/.tbss for more symbol reference, but for me I'll pick \_\_libc\_freeres\_fn since it is typical for this case.

To verdict its malicious process by reverse engineering

I will go to the reversing highlights, meanings..the most important process only. I don't write the sub functions, i.e. how it grabs the ethernet data, or how this malware use socket to connect an IP, for example, since the code is too long. But to be noted, since ARM architecture has different structure than Intel, and ARM is designed for the embedded systems, you will see many different method for the detail operation that is involving with the system calls.  
OK, here are the highlights that I would like to cover;

***1. Installation:***

Malware changes attribute & chmod the crontab, this is a bit specific setup that rarely found in the previous types, suggesting a new built, previously most of them are aiming autostart at the **xinetd** for autostart installation.

[?](https://blog.malwaremustdie.org/2014/09/reversing-arm-architecture-elf-elknot.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | .text:0x0A760  STMFD   SP!, {R4-R8,LR}  .text:0x0A764  SUB  SP, SP, #0x208  被斷成一整句 (no.38)  一  .text:0x0A768  ADD  R7, SP, #0x108  .text:0x0A76C  MOV  R4, R0  .text:0x0A770  MOV  R8, R1  .text:0x0A774  MOV  R0, R7  .text:0x0A778  MOV  R1, #0x100  .text:0x0A77C  BL   sub\_0x026FB0  .text:0x0A780  LDR  R0,  <-- "chattr -i /etc/crontab"  .text:0x0A784  BL   sub\_0x0E3E0  .text:0x0A788  LDR  R0, <-- "chmod +w /etc/crontab" |

Adding the autostart entry in it:

[?](https://blog.malwaremustdie.org/2014/09/reversing-arm-architecture-elf-elknot.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | .text:0x0A7DC ; xref: sub\_0x0A760  .text:0x0A7DC  LDR  R1, <-- "sed -i '/%s/d' /etc/crontab"  .text:0x0A7E0  MOV  R2, R5  .text:0x0A7E4  MOV  R0, R6     :  .text:0x0A800  LDR  R1, <-- "echo '\*/1 \* \* \* \* root %s/%s %s' >> /etc/crontab"  .text:0x0A804  STR  R8, [SP,#0x108+var\_108]  .text:0x0A808  BL   sub\_0x0182DC  .text:0x0A80C  MOV  R0, R6  .text:0x0A810  BL   sub\_0x0E3E0 |

Create the file: "/etc/.mysys", which later on found as the self copy attempt.

[?](https://blog.malwaremustdie.org/2014/09/reversing-arm-architecture-elf-elknot.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | .text:0x0A83C  STR  LR, [SP,#var\_4]!  被斷成一整句 (no.40)  一  .text:0x0A840  MOV  R1, #0x42  .text:0x0A844  SUB  SP, SP, #4  .text:0x0A848  LDR  R0, <-- "/etc/.mysys"  .text:0x0A84C  BL   sub\_0x0E350  .text:0x0A850  CMP  R0, #0  .text:0x0A854  MOV  R1, #6  .text:0x0A858  BGT  loc\_0x0A86C |

***2. Initiation of the service... which lead to CNC information :-)***

We'll see 3 thread will be spawned, which using "/dev/null" as value, this is typical MO for most types of AES.DDoS'ers we saw.

[?](https://blog.malwaremustdie.org/2014/09/reversing-arm-architecture-elf-elknot.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | .text:0x082E0  LDR  R0, <-- "/dev/null"  .text:0x082E4  BL   sub\_0x0E350 ;  .text:0x082E8  MOV  R1, #2  .text:0x082EC  LDR  R0, <-- "/dev/null"  .text:0x082F0  BL   sub\_0x0E350  .text:0x082F4  MOV  R1, #2  .text:0x082F8  LDR  R0, <-- "/dev/null"  .text:0x082FC  BL   sub\_0x0E350 |

..following with "effortS" to start the service:

[?](https://blog.malwaremustdie.org/2014/09/reversing-arm-architecture-elf-elknot.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | .text:0x0831C  LDR  R0, [R5]  .text:0x08320  LDR  R1, <-- "mt"  .text:0x08324  BL   sub\_0x0A760  .text:0x08328  MOV  R0, #1  .text:0x0832C  MOV  R1, R0  .text:0x08330  BL   sub\_0x01614C  .text:0x08334  LDR  R0, <-- reg RO = "Int Server..."  .text:0x08338  BL   sub\_0x018F2C  .text:0x0833C  MOV  R1, R4  .text:0x08340  MOV  R3, R4  .text:0x08344  LDR  R2, =sub\_0x08A60  .text:0x08348  LDR  R0, =unk\_0x0D0990  .text:0x0834C  BL   sub\_0x0B33C  .text:0x08350  MOV  R1, R4  .text:0x08354  MOV  R3, R4  .text:0x08358  LDR  R2, =sub\_0x088D8  .text:0x0835C  LDR  R0, =unk\_0x0D0988  .text:0x08360  BL   sub\_0x0B33C  .text:0x08364  MOV  R1, R4  .text:0x08368  MOV  R3, R4  .text:0x0836C  LDR  R2, =sub\_0x08598  .text:0x08370  LDR  R0, =unk\_0x09097C  .text:0x08374  BL   sub\_0x0B33C  .text:0x08378  MOV  R1, R4  .text:0x0837C  MOV  R3, R4 |

To connect to "something". I trailed it to get all variables needed:

[?](https://blog.malwaremustdie.org/2014/09/reversing-arm-architecture-elf-elknot.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | .text:0x08394  LDR  R0, <-- RO contains "connect to server..."  .text:0x08398  BL   sub\_0x018F2C  .text:0x0839C  MOV  R1, #0  .text:0x083A0  MOV  R3, R1  .text:0x083A4  LDR  R2, =sub\_0x0A038 <--"jump here"         [...]  .text:0x0A038  STMFD   SP!, {R4-R10,LR}  .text:0x0A03C  SUB     SP, SP, #0x9C0  .text:0x0A040  BL      sub\_0x09E68 <--"jump again here" |

And the destination 0x09E68 there is the IP address of this connection.

[?](https://blog.malwaremustdie.org/2014/09/reversing-arm-architecture-elf-elknot.html)

|  |  |
| --- | --- |
| 1  2 | .text:0x09E68 LDR  R0, =unk\_0x08C5C4  <-- address to get the CNC IP Address  .text:0x09E68 <-- go down to hard-copied data:0x08C5C8 it's the IP "182.254.180.241" |

Now we know the CNC is in **182.254.180.241** which is in:

[?](https://blog.malwaremustdie.org/2014/09/reversing-arm-architecture-elf-elknot.html)

|  |  |
| --- | --- |
| 1  2  3  4 | ASN: 45090 / CNNIC-TENCENT-NET  PREFIX: 182.254.180.0/23  ISP: COMSENZ TECHNOLOGY LTD  COUNTRY: CHINA |

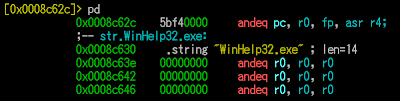
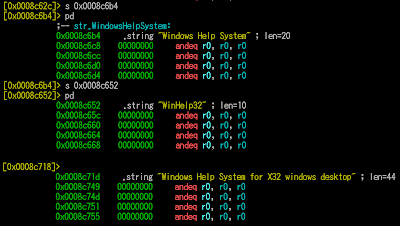
..well, I am not surprised.

***3. An effort to fake Windows Help (WinHelp.exe) service :)***

Continuing the data flow started above, I end-up facing an interesting data:

[?](https://blog.malwaremustdie.org/2014/09/reversing-arm-architecture-elf-elknot.html)

|  |  |
| --- | --- |
| 1  2  3  4 | :  .text:0x09E68 LDR  R0, =unk\_0x08C5C4  .text:0x09E6C STMFD   SP!, {R4-R8,R10,LR}  .text:0x09E70 LDR  R2, [R0,#(dword\_0x08C62C - 0x08C5C4)] |

The data after dword in .data:0x08C62C is the .data:0x08C630 (DCB) which is "WinHelp32.exe", see it here if you don't believe me:  
[](https://lh5.googleusercontent.com/-myagsKVBigU/VBPAL40GtYI/AAAAAAAAQ38/gyAHlbXkqSg/s1301/001.png)  
This is just unbelievable, seeking further to figure what is this, I found the complete set of data for this "fake process" which is a self explanatory:  
[](https://lh6.googleusercontent.com/-jGpo8vaGxi4/VBPALwkrvvI/AAAAAAAAQ4A/urO4tUdOYHc/s1324/002.png)  
I don't know what to say about this..

***4. PoC of backdoor and sending sensitive data to remote host:***

It's self explanatory in the codes below, the BackConnect part:

[?](https://blog.malwaremustdie.org/2014/09/reversing-arm-architecture-elf-elknot.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | .text:0x08420 ; .text:off\_0x08408  .text:0x08420  STMFD   SP!, {R4,LR}  .text:0x08424  LDR  R4, =dword\_0x0D0984  .text:0x08428  .text:0x08428 ; xref: sub\_0x08420  .text:0x08428  LDR  R0, <---- "Back connect to server..."  .text:0x0842C  BL   sub\_0x018F2C  .text:0x08430  MOV  R1, #0  .text:0x08434  MOV  R3, R1  .text:0x08438  LDR  R2, =sub\_0x099E0  .text:0x0843C  LDR  R0, =dword\_0x0D0984  .text:0x08440  BL   sub\_0x0B33C  .text:0x08444  MOV  R1, #0  .text:0x08448  LDR  R0, [R4]  .text:0x0844C  BL   sub\_0x0C4FC  .text:0x08450  LDR  R0, [R4]  .text:0x08454  BL   sub\_0x0E070  .text:0x08458  LDR  R0, =0x4C4B40  .text:0x0845C  BL   sub\_0x0272F0  .text:0x08460  B    loc\_0x08428 |

And the trace of information to be sent to remote:

[?](https://blog.malwaremustdie.org/2014/09/reversing-arm-architecture-elf-elknot.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | .text:0x08500  STMFD   SP!, {R4,R5,LR}  .text:0x08504  LDR  R1, =dword\_0x090980  .text:0x08508  SUB  SP, SP, #0x400  .text:0x0850C  SUB  SP, SP, #0xC  .text:0x08510  LDR  R2, =dword\_0x090978  .text:0x08514  LDR  R12, [R1]  .text:0x08518  ADD  R4, SP, #0x418+var\_410  .text:0x0851C  LDR  R3, [R2]  .text:0x08520  MOV  R1, #0x400  .text:0x08524  LDR  R2  <--- "INFO:%d|%d"  .text:0x08528  MOV  R0, R4  .text:0x0852C  STR  R12, [SP,#0x418+var\_418]  .text:0x08530  BL   sub\_0x0182B0  .text:0x08534  LDR  R3, =dword\_0x08CF44  .text:0x08538  MOV  R0, R4  .text:0x0853C  LDR  R5, [R3]  .text:0x08540  BL   sub\_0x024540  .text:0x08544  MOV  R1, R4  .text:0x08548  ADD  R2, R0, #1  .text:0x0854C  MOV  R0, R5  .text:0x08550  BL   sub\_0x0DF10  .text:0x08554  LDR  R3, =dword\_0x08CF48  .text:0x08558  MOV  R0, R4  .text:0x0855C  LDR  R5, [R3]  .text:0x08560  BL   sub\_0x024540  .text:0x08564  MOV  R1, R4  .text:0x08568  ADD  R2, R0, #1  .text:0x0856C  MOV  R0, R5  .text:0x08570  BL   sub\_0x0DF10  .text:0x08574  ADD  SP, SP, #0xC  .text:0x08578  ADD  SP, SP, #0x400  .text:0x0857C  LDMFD   SP!, {R4,R5,LR}  .text:0x08580  BX   LR |

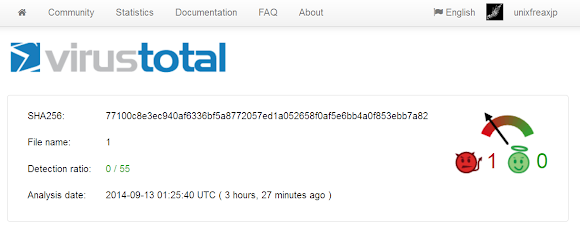
***5. The HTTP header used for DoS activity:***

This is the function to be called when performing DoS by HTTP, I pasted it here as PoC of DDoS'er, please bear the length:

[?](https://blog.malwaremustdie.org/2014/09/reversing-arm-architecture-elf-elknot.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63 | .text:0x0A548  LDR  R1 <--- " HTTP/1.1\r\n"  .text:0x0A54C  MOV  R2, #0xB  .text:0x0A550  ADD  R0, R8, R5  .text:0x0A554  BL   sub\_0x0252B8  .text:0x0A558  ADD  R0, R5, #0xB  .text:0x0A55C  LDR  R1 <--- "Accept: text/html, application/xhtml+xml, \*/\*\r\n"  .text:0x0A560  MOV  R2, #0x2F  .text:0x0A564  ADD  R0, R8, R0  .text:0x0A568  BL   sub\_0x0252B8  .text:0x0A56C  ADD  R0, R5, #0x3A  .text:0x0A570  LDR  R1 <--- "Accept-Language: zh-CN\r\n"                        "↑please noted this CHINESE character encoding↑"  .text:0x0A574  MOV  R2, #0x18  .text:0x0A578  ADD  R0, R8, R0  .text:0x0A57C  BL   sub\_0x0252B8  .text:0x0A580  ADD  R0, R5, #0x52  .text:0x0A584  LDR  R1 <--- "User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; WOW64; Trident/6.0)\r\n"  .text:0x0A588  MOV  R2, #0x55  .text:0x0A58C  ADD  R0, R8, R0  .text:0x0A590  BL   sub\_0x0252B8  .text:0x0A594  ADD  R0, R5, #0xA7  .text:0x0A598  LDR  R1 <--- "Accept-Encoding: gzip, deflate\r\n"  .text:0x0A59C  MOV  R2, #0x20  .text:0x0A5A0  ADD  R0, R8, R0  .text:0x0A5A4  BL   sub\_0x0252B8  .text:0x0A5A8  ADD  R0, R5, #0xC7  .text:0x0A5AC  LDR  R1 <--- "Host: "  .text:0x0A5B0  MOV  R2, #6  .text:0x0A5B4  ADD  R0, R8, R0  .text:0x0A5B8  BL   sub\_0x0252B8  .text:0x0A5BC  MOV  R0, R9  .text:0x0A5C0  BL   sub\_0x024540  .text:0x0A5C4  ADD  R5, R5, #0xCD  .text:0x0A5C8  ADD  R4, R8, R5  .text:0x0A5CC  MOV  R2, R0  .text:0x0A5D0  MOV  R1, R9  .text:0x0A5D4  MOV  R0, R4  .text:0x0A5D8  BL   sub\_0x0252B8  .text:0x0A5DC  MOV  R0, R9  .text:0x0A5E0  BL   sub\_0x024540  .text:0x0A5E4  ADD  R5, R5, R0  .text:0x0A5E8  LDR  R1 <--- "\r\nConnection: Keep-Alive\r\n"  .text:0x0A5EC  MOV  R2, #0x1A  .text:0x0A5F0  ADD  R0, R8, R5  .text:0x0A5F4  BL   sub\_0x0252B8  .text:0x0A5F8  ADD  R0, R5, #0x1A  .text:0x0A5FC  MOV  R2, #0x14  .text:0x0A600  LDR  R1 <--- "Pragma: no-cache\r\n\r\n"  .text:0x0A604  ADD  R0, R8, R0  .text:0x0A608  BL   sub\_0x0252B8  .text:0x0A60C  MOV  R1, #1  .text:0x0A610  MOV  R0, #0xD  .text:0x0A614  BL   sub\_0x01614C  .text:0x0A618  MOV  R0, R10  .text:0x0A61C  MOV  R1, R8  .text:0x0A620  ADD  R2, R5, #0x2E  .text:0x0A624  MOV  R3, #0  .text:0x0A628  BL   sub\_0x0E1D0     :    ;  .text:0x0A674  LDR  R1 <-- "GET "  .text:0x0A678  MOV  R2, #4  .text:0x0A67C  MOV  R0, R8  .text:0x0A680  BL   sub\_0x0252B8 |

Detection ratio & sample

The detection ratio is very low, like..ZERO. Here's the evidence, please click to enlarge the image:  
[](https://lh4.googleusercontent.com/-miQ6f5JM57s/VBPN_475ZQI/AAAAAAAAQ4Q/Tl2RQjtijbo/s1218/003.png)  
The VirusTotal's link is here-->[[LINK](https://www.virustotal.com/en/file/77100c8e3ec940af6336bf5a8772057ed1a052658f0af5e6bb4a0f853ebb7a82/analysis/1410571540/)]

This post is one of proof of concept that routers is actually aimed by the malware actors for one of some reasons, and the main purpose is because they are widely used all over the internet with having the global IP address, and also up and alive 364/7/24. For the crooks who are behind the malware describes in this case, owning many routers means having power of an "army of DoS bots" than can be powerful tool for an attack. We saw not only ARM architecture, but MIPS, PPC, MIPSEL, SuperH(SH) binaries are also spotted in the wild.

I am adding these project's sample in kernel mode, Will add the link shortly in here, please stay tune, I must clean up all of the garbage I made first. This is the link-->[[HERE](http://www.kernelmode.info/forum/viewtopic.php?f=16&t=3099&p=23874#p23874)]

Conclusion & additional notes

It is up to you to defend your own router. As you can see no AV can detect these malware, it's over a week being there now. Please check your router user interface, make sure you are using the latest updates/firmware and make sure that your setting is correct and unchanged. Being skeptical during checking your router/gateway layer is very recommendable, and if you find anything unusual/suspicious please analyze it WHY and try not to let it go until you find a satisfactory answer for it. If you find it work and having no problem, backup the setting and save it right away.

[#Linux](https://twitter.com/hashtag/Linux?src=hash)/AES.DDoS is having variant that is NOT using "<http://t.co/n8RjVMx9lq>" so please be noted NOT to put those as equals. [#ELF](https://twitter.com/hashtag/ELF?src=hash) [#Malware](https://twitter.com/hashtag/Malware?src=hash)

— Hendrik Adrian (@unixfreaxjp) [January 17, 2015](https://twitter.com/unixfreaxjp/status/556576504750952448)

The Intel x32 edition of this variant just was just spotted, analysis is here-->[[LINK](http://www.kernelmode.info/forum/viewtopic.php?f=16&t=3099&p=23875#p23875)] <<-- you can see more details on source, compatibility, compilation etc.

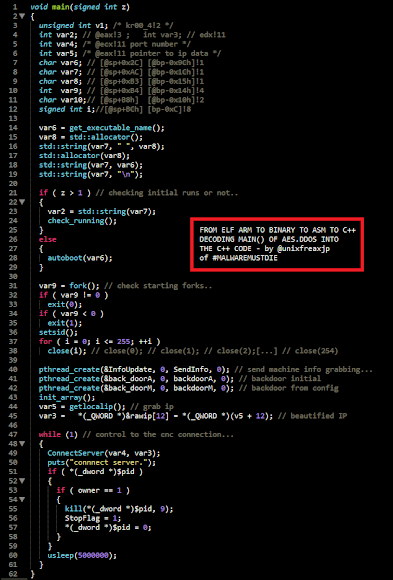
The router version of ELF DDoS + backdoor malware **is also spotted in the MIPS architercture**, analysis is here-->[[HERE](http://www.kernelmode.info/forum/viewtopic.php?f=16&t=3483&p=23878#p23878)] and in here-->[[HERE](http://www.kernelmode.info/forum/viewtopic.php?f=16&t=2747#p23810)]. The older version of the ARM ELF DDoS'er malware spotted is also available here-->[[HERE](https://www.virustotal.com/en/file/3654c3e333fae286753ae29dbd2040b6680820c77af78da3c5bf9190e531cad8/analysis/)].  
The below tweet is the PoC that even **PPC architecture** is also aimed by DDoS'er malware too now (different actor & using "Tsunami" malware)

Another [#PoC](https://twitter.com/hashtag/PoC?src=hash) that router is aimed by [#malware](https://twitter.com/hashtag/malware?src=hash) crooks nowadays: <https://t.co/G56tofmX3v> < [#ELF](https://twitter.com/hashtag/ELF?src=hash) [#PPC](https://twitter.com/hashtag/PPC?src=hash) Tsunami/Kaiten IRC/[#DDoS](https://twitter.com/hashtag/DDoS?src=hash) [#MalwareMustDie](https://twitter.com/hashtag/MalwareMustDie?src=hash)

— MalwareMustDie, NPO (@MalwareMustDie) [September 16, 2014](https://twitter.com/MalwareMustDie/status/511853487219425280)

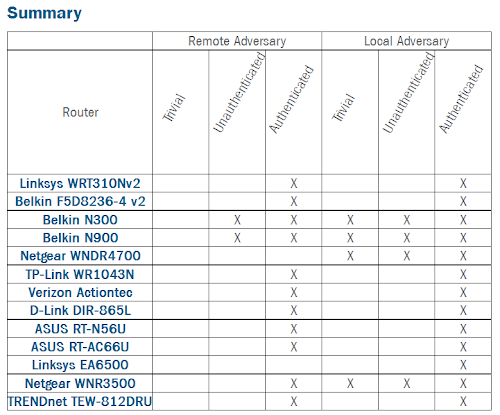
This is a warning of the true fact & evidence that the **recent ELF malware coders are not only aiming x32 or x64 servers anymore, but routers too**.

C++ reversing for the AES.DDOS

I just figured a good technique in open source to reverse the assembly code into the (close) original C++ codes, so far works fine under the known libraries and specific forms of ELF binary. I tested in AES.DDOS and worked perfectly fine. Below is the main() function of this malware (Linux/AES.DDoS) reversed engineered into the C++ code:  
[](https://lh3.googleusercontent.com/-DryofowT2A0/Vw5UqGeCHfI/AAAAAAAAUrY/YQbAoiJZACU537TpIQkGxu7pcty_su01wCLcB/s2120/001.png)  
It's still in the test stage now, some manual adjustment is still needed, but it is much better than reading assembly. A good project for myself for this year.

Why SOHO routers are aimed for malware infection?

The excellent research conducted by ISE (independent security firm in Baltimore, Maryland) explained in their publishment here--[[LINK](https://securityevaluators.com/knowledge/case_studies/routers/soho_router_hacks.php)], that:  
***"..discovered critical security vulnerabilities in numerous small office/home office (SOHO) routers and wireless access points. These vulnerabilities allow a remote attacker to take full control of the router's configuration settings; some allow a local attacker to bypass authentication directly and take control. "***

As an illustration, ISE shows a matrix of vulnerability vectors for the evaluated known routers:  
[](https://securityevaluators.com/knowledge/case_studies/routers/soho_router_hacks.php)  
This shows us there is a weak security vector is aimed in SOHO router layer, and most of the houses & SOHO business are connected in the internet through these xDSL routers. We can not under-estimated the current volume of these routers, being up and alive in internet now. Maybe there's only a low percentage of alive routers are having the vulnerabilities mentioned, but please imagine how powerful a DDoS attack will be if a bad actor is successfully gaining access to control, say, 1% of overall alive xDSL SOHO routers. And please think what if your house or office routers are unknowingly participated into a DDoS or other attacks against a certain banks or a specific country in the world?

**Additional: China ELF CNC & Web Panels Takedown**

Among of the attackers we detected so far, China's bad actors are the most aggressive one. If the bad actors in China think that MMD won't do anything about their evil action, they can start to cry now, we tango'ed 25 **29 ELF malware download panels panels** (the counting is still rolling) as per

announced below:

[#MalwareMustDie](https://twitter.com/hashtag/MalwareMustDie?src=hash)! Tango down report of OP [#China](https://twitter.com/hashtag/China?src=hash) [#ELF](https://twitter.com/hashtag/ELF?src=hash) [#DDoS](https://twitter.com/hashtag/DDoS?src=hash)'er [#malware](https://twitter.com/hashtag/malware?src=hash) <http://t.co/n2gKTXSDDh> ←25 panels taken down [pic.twitter.com/nOaRSwSYYB](http://t.co/nOaRSwSYYB)

— MalwareMustDie, NPO (@MalwareMustDie) [September 14, 2014](https://twitter.com/MalwareMustDie/status/511191864364388352)

Stay safe, friends! #MalwareMustDie!

Posted by unixfreaxjp at [Saturday, September 13, 2014](https://blog.malwaremustdie.org/2014/09/reversing-arm-architecture-elf-elknot.html)