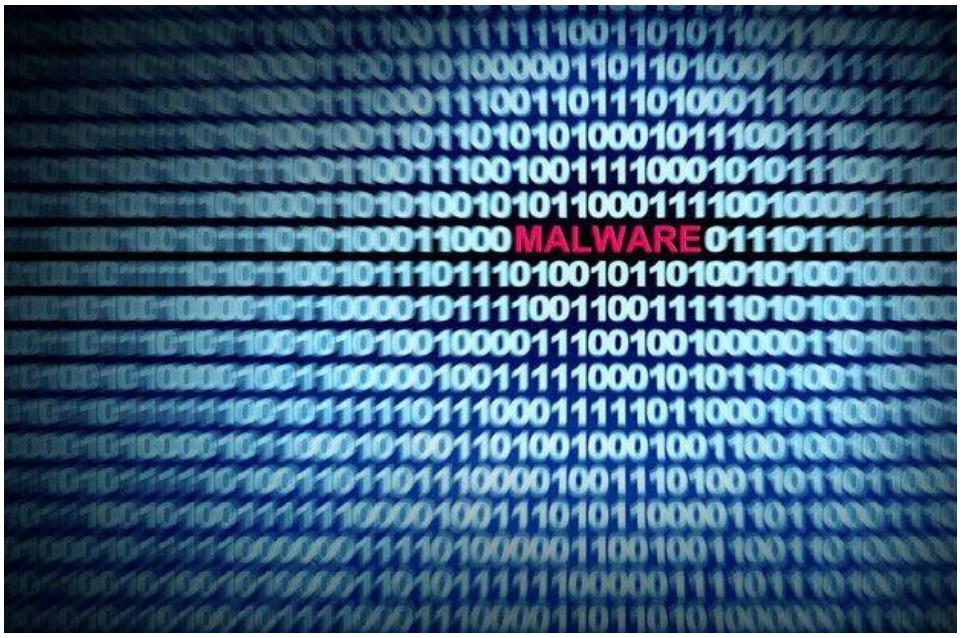
АНТИВИРУСНАЯ ЗАЩИТА БИЗНЕС-КЛАССА

Вредоносное ПО Linux/Shishiga использует скрипты Lua

3 мая 2017 года

Среди образцов вредоносного ПО для Linux, получаемых ежедневно, мы заметили подозрительный экземпляр, который детектировал только Dr.Web – как Linux.LuaBot. Довольно странно, поскольку наша частота обнаружений малвари семейства Luabot как правило выше.

Выполнив анализ, мы выяснили, что программа написана на Lua и относится к новому семейству, не связанному с уже известным Luabot. Малварь получила название Linux/Shishiga. Она использует четыре протокола (SSH – Telnet – HTTP – BitTorrent) и Lua-скрипты.



Среда обитания Shishiga

Linux/Shishiga атакует системы GNU/Linux. Вектор заражения типичен – брутфорс слабых паролей из списка. Малварь работает примерно как Linux/Moose, но также может брутфорсить SSH. Вот полный список паролей на момент написания отчета:

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"},
]]
[...]
```

Мы также обнаружили несколько бинарных файлов Linux/Shishiga для различных архитектур, включая MIPS (от старшего к младшему и наоборот), ARM (armv4l), i686, а также PowerPC. Они часто используются в устройствах интернета вещей. Мы предполагаем, что могут поддерживаться и другие архитектуры, такие как SPARC, SH-4 или m68k, но об этом позже.

Возможности Shishiga

Linux/Shishiga — бинарный файл, упакованный с помощью UPX 3.91 (упаковщик исполняемых файлов). Но у UPX возникнут сложности с распаковкой, поскольку Shishiga добавляет данные в конец запакованного файла.

После распаковки файла мы видим, что он статично скомпонован с библиотекой рабочих программ Lua, а код удален.

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

При исполнении бинарный файл запускает модуль малвари Lua следующими методами:

```
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
```

Метод getmods возвращает архивированный blob, о чем подробнее расскажем позже. Затем жестко запрограммированный код Lua (malware.lua) исполняется через функции luaL_loadstring и lua_pcall. Код Lua довольно простой, вот краткое описание исходного кода без изменений от нас.

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) открывает файл exe-файл малвари из /proc/self/exe и возвращает его содержимое;
(2) восстанавливает архив zlib посредством метода getmods, распаковывает, а затем парсит его при помощи тэгов и хранит в массиве Lua;
(3) вызывает модуль main.lua.

Есть обширный список всех скриптов Lua, найденных в секции показателей компрометации. У большинства из них названия говорят за себя, но мы все равно кратко опишем некоторые из них.

callhome.lua

- получает файл конфигурации server.bt или servers из config.lua;
- при невозможности связаться с текущим дефолтным сервером, меняет его на другой;
- пересылает файлы (отчеты или аккаунты, оба в формате JSON);
- исполняет задачи из списка задач, полученного с C&C-сервера.

bfssh.lua / bftelnet.lua

- модуль для брутфорса SSH- и Telnet-логинов;
- проверяет, дает ли команда echo -en "\\x31\\x33\\x33\\x37" на выходе 1337; если нет — выйти, иначе — продолжить;
- архитектура устройства определяется файлом из /bin/ls запуском cat /bin/ls и парсингом заголовка ELF, об этом ниже;
- распространение малвари (.lm и .dm файлов) согласно архитектуре устройства;
- сохранение успешных комбинаций пользовательских параметров.

Код проверки архитектуры выглядит следующим образом.

bfssh.lua, метод getArchELF

```
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) каждый файл ELF должен начинаться с \x7fELF;
(2) ftype, который представляет e_type (тип файла ELF = исполнимый, расшариваемый, итд.) не используется;
(3) bits представляет e_ident[EI_CLASS] (32 или 64 бита), denc представляет e_ident[EI_DATA] (от старшего к младшему, и наоборот), и farch представляет e_machine в заголовке ELF.

bfssh.lua, метод getArchName

```
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 не используется
(2) проверяет принадлежность файла к MIPS (от младшего) (e_machine == EM_MIPS и e_ident[EI_DATA] == ELFDATA2LSB)
(3) проверяет принадлежность файла Intel 80386 или Intel 80860 или AMD x86-64 (e_machine == EM_386 или e_machine == EM_860 или e_machine == EM_X86_64)

config.lua

- содержит открытый ключ publicKey для проверки подписи бинарника (.lm или .dm);
- содержит список компонентов загрузчика;
- содержит имена .bt файлов, номера портов серверов SOCKS и HTTP;
- содержит IP-адреса сервера (возможно, C&C).

persist.lua

- метод сохранения, в зависимости от прав пользователя (рут или юзер)

scanner.lua

- используется для генерации случайных / 16 сетей, которые не являются локальными

worm.lua (этот скрипт убрали в последней версии Linux/Shishiga)

- разрешает сканирование заданного порта;
- разрешает загрузку;
- получает информацию с нового зараженного сервера.

Содержание файла readme.lua оценят русскоговорящие:

```
          ВСЁ ИДЁТ ПО ПЛАНУ

А при коммунизме всё будет за
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
```

В последние несколько недель мы наблюдали небольшие изменения: некоторые блоки были переписаны, добавлены тестовые блоки, удалены ненужные файлы, но в целом ничего примечательного.

Основной бинарный файл называется .lm, но мы также смогли получить файл под названием .dm – простой бэкдор, прослушивающий 0.0.0.0 (все IPv4 адреса) порта 2015. Изменилось название файла бэкдора – с dl на dm.

Коммуникации Shishiga

Linux/Shishiga может обмениваться данными с помощью любого из модулей httpproto.lua, btloader.lua или server.lua. Модуль httpproto.lua имеет функции, позволяющие кодировать и декодировать информацию и делать запросы HTTP POST и GET. Исходный код ниже показывает процесс кодирования.

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 использует модуль torrent.lua (враппер для функций BitTorrent) для сохранения и загрузки узлов из файла nodes.cfg. Он также получает его конфигурацию из файлов {server,update,script}.bt (в формате Bencode) и использует протокол BitTorrent для проверки

обновлений этих файлов. script.bt позволяет исполнить скрипт Lua, а update.bt разрешает исполнение бинарника .lm. Ниже примеры декодированных файлов .bt, показанных в виде словарей Python.

script.bt

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

(1) подпись
(2) открытый ключ

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'
}
```

Наконец, основная функция модуля server.lua — создание HTTP сервера с портом, прописанным в config.lua. Во всех образцах, которые мы на данный момент изучили, это был порт 8888.

Сервер отвечает только на запросы /info и /upload. Ниже показана «причесанная» версия ответа сервера к пути /info. Все файлы ниже могут быть легко загружены с зараженного устройства.

```
{
    "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) скрипты Lua
(2) бэкдор (старое имя: .dl)
(3) скрипты BitTorrent
(4) версия малвари
(5) загрузчик модулей

Запрос в рут / по порту 8888 в результате даст HTTP/1.0 404 OK, что является простым показателем компрометации (IoC).

Функция ответа http.lua

```
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
```

На этом этапе исследования мы попросили команду Censys провести массированное сканирование интернета по TCP-порту 8888. Они обнаружили около десяти IP-адресов, по которым совпадал ответ HTTP. Эти адреса принадлежат потенциально зараженным машинам.

Вывод

На первый взгляд, Linux/Shishiga может показаться похожей на другие вредоносные программы, которые распространяются через слабозащищенные Telnet и SSH, но ее выделяет использование протокола BitTorrent и модулей Lua. BitTorrent в черве Hajime, вдохновленном Mirai, был обнаружен в прошлом году, и мы можем только предполагать, станет ли он популярнее в будущем.

Возможно, что Shishiga эволюционирует и будет более распространенной. Пока незначительное число жертв, изменения компонентов, комментарии в коде и информация по устранению багов указывают, что работа над малварью в процессе. Чтобы предотвратить заражение ваших устройств малварью Shishiga и похожими червями, не стоит использовать стандартные учетные данные Telnet и SSH.

Благодарим команду Censys за сотрудничество.

Индикаторы заражения (IoC)

Командный сервер
93.117.137.35

SHA-1 (.lm)

```
003f548796fb52ad281ae82c7e0bb7532dd34241
1a79092c6468d39a10f805c96ad7f8bf303b7dc8
1cc1b97f8f9bb7c4f435ef1316e08e5331b4331b
288980377e2dfec7684512f45e87248a07d508f
2a809d37be5aa0655f5cc997eb62683e1b45da17
3f1ef05ca850e2f5030ee279b1c589c9e3cc576c
41bf0d5612ba5bc9a05e9d94df0f841b159264a0
4bc106f6231daa6641783dd9276b4f5c7fc41589
4d55efe18643d7408cbe12dd4f319a68084bd11e
4df58ab26f0fc8ec2d1513611ca2b852e7107096
51a4ca78ebb0649721ae472290bea7bfe983d727
5a88b67d8dfaf1f68308311b808f00e769e39e46
6458c48e5167a2371d9243d4b47ad191d642685b
688ccbca8b2918a16191703 1e21b6810c59eeab0
6e3ba86d1f91669e87945b8ea0211b58e315e189
6f41c8f797814e2e3f073601ce81e8adceef6a27
8a0d58472f6166ade0ae677bab7940fe38d66d35
8a1f9212f181e68a63e06a955e64d333b78c6bf6
8e3c4eb04d4cfd8f44c721111c5251d30ac848b6
979fb376d6adc65473c4f51ad1cc36e3612a1e73
a1f2535576116d93b62d7f5fc6e30e66e0e0a216
a694c6ecc2ff9702905f22b14ed448e9e76fe531
ac094b239851eaf2e9fd309285c0996fb33771a8
b14f7af9665ef77af530109a0331f8ca0bd2a167
b86935c4539901cdec9081d8a8ca915903adaff1
ba5df10549 6b0c4df7206d29fa544b7a7a346735
bf4d9e25fc210a1d9809aebb03b30748dd588d08
c22f0fb01c6d47957732a8b0f5ef0f7d4e614c79
ce4b3c92a96137e6215a5e2f5fd28a672eddaaab
d8a5d9c4605b33bd47fedbad5a0da9928de6aa33
f73022a4801e06d675e5c3011060242af7b949ad
```

SHA-1 (.dl)

```
274181d2f9c6b8f0e217db23f1d39aa94c161d6e
8abbb049bffd679686323160ca4b6a86184550a1
95444c2ccc5fff19145d60f1e817fd682cabe0cd
9cde8458526533339f67667c2408126f02f246949
```

Имена файлов скриптов Lua

```
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
```

Файлы, которые потенциально могут указывать на заражение

```
/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}.{dl,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
```